ABSTRACT

Title of Dissertation: Reliable Deep Learning: a Robustness Perspective
Sahil Singla
Doctor of Philosophy, 2022

Dissertation Directed by: Professor Soheil Feizi
Department of Computer Science

Deep learning models achieve impressive performance on many benchmark tasks often surpassing human-level performance. However, the performance drops considerably when there is a mismatch between training and test distributions. A key reason for this mismatch can be due to the presence of spurious features in the training set i.e. features that happen to co-occur with the main object but are not essential for predicting the same. A related limitation of these models is their vulnerability to adversarial perturbations: input perturbations imperceptible to a human that can arbitrarily change the prediction of the model. These limitations can potentially create life-threatening situations when these models are deployed in safety-critical applications.

In this dissertation, we develop several algorithms for addressing these challenges.

We first propose a framework for defending against unforeseen adversarial attacks by approximating the set of all imperceptible adversarial examples using deep neural networks. Following this, we study the effect of loss curvature on saliency maps, robust overfitting and provable adversarial robustness. Next, we introduce several building blocks for provably 1-Lipschitz neural networks namely an orthogonal convolution layer, activation functions with orthogonal

jacobian, learnable lipschitz pooling layers and procedures for certifying adversarial robustness. Our newly introduced methods lead to significant improvements in both the clean and certified robust accuracy across several lipschitz networks with varying depth. Finally, we introduce several algorithms for identifying and mitigating failure modes of deep networks. Using the neurons of adversarially robust models as visual attribute detectors, we identify clusters of data with high error rate when certain visual attributes are either absent or present. Our analysis reveals several ImageNet classes that are highly susceptible to spurious features. Finally, to mitigate the discovered failure modes, we introduce a framework for improving model performance on the failure mode distribution by discovering images from the web that are perceptually similar to the images from the failure mode and adding the newly discovered images to the training set.

Reliable Deep Learning: a Robustness Perspective

by

Sahil Singla

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2022

Advisory Committee:
Professor Soheil Feizi, Chair/Advisor
Professor Tom Goldstein
Professor Ming Lin
Professor Rene Vidal
Professor Behtash Babadi
Dr. Eric Horvitz

© Copyright by
Sahil Singla
2022

Preface

This thesis presents the research conducted by Sahil Singla, in collaboration with several other people. Below is the list of the scientific papers by Sahil Singla that were published through the course of his studies as a Doctor of Philosophy (PhD) candidate at the University of Maryland. The chapters discussing the relevant papers in the dissertation are also given below.

Defenses against unforeseen adversarial attacks.

- **Chapter 2:** Cassidy Laidlaw, Sahil Singla, Soheil Feizi. Perceptual Adversarial Robustness: Defense Against Unseen Threat Models. *ICLR 2021*.

Second-order methods for interpretability and robustness.

- **Chapter 3:** Sahil Singla, Eric Wallace, Shi Feng, Soheil Feizi. Understanding Impacts of High-Order Loss Approximations and Group Features in Interpretation. *ICML 2019*.
- **Chapter 4:** Vasu Singla, Sahil Singla, Soheil Feizi, David Jacobs. Low Curvature Activations Reduce Overfitting in Adversarial Training. *ICCV 2021*.
- **Chapter 5:** Sahil Singla, Soheil Feizi. Second-Order Provable Defenses against Adversarial Attacks. *ICML 2020*.

Lipschitz neural networks.

- **Chapter 6:** Sahil Singla, Soheil Feizi. Fantastic Four: Differentiable and Efficient Bounds on Singular Values of Convolution Layers. *ICLR 2021*.

- **Chapter 7:** Sahil Singla, Soheil Feizi. Skew Orthogonal Convolutions. *ICML 2021*.

- **Chapter 8:** Sahil Singla, Surbhi Singla, Soheil Feizi. Improved deterministic l2 robustness on CIFAR-10 and CIFAR-100. *ICLR 2022*.

- **Chapter 9:** Sahil Singla, Soheil Feizi. Improved techniques for deterministic l2 robustness. *NeurIPS 2022*.

Understanding and mitigating model failures.

- **Chapter 10:** Sahil Singla, Besmira Nushi, Shital Shah, Ece Kamar, Eric Horvitz. Understanding Failures of Deep Networks via Robust Feature Extraction. *CVPR 2021*.

- **Chapter 11:** Sahil Singla, Soheil Feizi. Salient Imagenet, How to discover spurious features in deep learning? *ICLR 2022*.

- **Chapter 12:** Mazda Moayeri, Sahil Singla, Soheil Feizi. Hard ImageNet: Segmentations for Objects with Strong Spurious Cues. *NeurIPS 2022*.

- **Chapter 13:** Mazda Moayeri, Wenxiao Wang, Sahil Singla, Soheil Feizi. Spuriosity Rankings: Sorting Data for Spurious Correlation Robustness. Under submission. `https://arxiv.org/abs/2212.02648`.

- **Chapter 14:** Sahil Singla, Atoosa Malemir Chegini, Mazda Moayeri, Soheil Feizi. Data-Centric Debugging: mitigating model failures via targeted data collection. Under submission. `https://arxiv.org/abs/2211.09859`.

Dedication

To my mother.

Acknowledgments

First and foremost, I would like to thank my rockstar advisor: Prof. Soheil Feizi. I believe words cannot do justice at describing what a great privilege it was to be Soheil's student, here is a very modest attempt at that.

When I started my PhD, I would come up with several new ideas from varied topics and Soheil would always be available to discuss, critique and provide his valuable guidance. As a new student, it was very encouraging to see an established researcher like him taking me seriously. It gave me the confidence to conduct independent research. He would always encourage me to think about new angles to tackle a problem and quick experiments to validate a hypothesis.

Soheil's ability to construct a convincing "story" around a paper and sell it to the audience is unmatched. He knows exactly how to write a paper in a way that even the harshest reviewers would find convincing. He spent countless hours giving valuable feedback for my writing and teaching me how to give more engaging presentations. Fun fact: during my undegrad in India (at IIT-Delhi), I received the lowest possible grade in a course where the evaluation was based on a single talk. I was the only person in the class to receive that grade. That is how bad I was at giving presentations before my PhD. Today, I often receive praise from the audience for giving very engaging talks. Most people who have listened to my talks find it hard to believe that my grade could have been so bad in a presentation focused course. That is how much of a difference Soheil has brought in my presentation skills.

Soheil tries his best to ensure that we get introduced to other researchers with related areas of research. In several instances, he provided valuable feedback about the "soft skills" needed to succeed in your career in either industry or academia. Everything that I learned from Soheil is priceless and I am indebted to him forever. I wish someday my research makes a significant contribution in the world and I can make Soheil a proud advisor.

During my second year, I had the privilege to work with Besmira Nushi, Shital Shah, Ece Kamar and Eric Horvitz. I learned about the practical aspects of using AI at a large scale in the industry settings and had great insightful conversations with everyone.

I learned a great deal from various collaborators throughout my PhD: Eric Wallance, Shi Feng, Alex Levine, Vedant Nanda, Samuel Dooley, John Dickerson, Vasu Singla, David Jacobs, Cassidy Laidlaw, Surbhi Singla, Mazda Moayeri, Atoosa Chegini, Wenxiao Wang. I am grateful to everyone for giving me a chance to contribute.

I had several fun discussions with my labmates: Amir Nili, Vedant Nanda, Philip Pope, Alex Levine, Chenghao Deng, Michael Curry, Aviva Prins, Mazda Moayeri, Atoosa Chegini, Saptarashmi Bandyopadhyay. The discussions would range from all sorts of random topics like politics, philosophy, childhood experiences growing up in different countries, technology and what not. If anyone was bored of work, we could start a random conversation, waste some time, recharge and get back to work. I loved swimming visits to the University's outdoor pool with Michael and Philip. Thank you everyone, for making every single day in the lab so beautiful.

I cannot emphasize enough how much support I received from Tom Hurst throughout my graduation. Tom is every graduate student's guardian angel. If I had any questions or concerns, I would reach out to him and the problem would be quickly solved. Often times, I would make mistakes and Tom would be there to correct. Several times, I felt that he was more worried about

my problems than I was. For graduate students, Tom is the HOD, dean and president all combined in one. Thank you Tom for making my stay at UMD a beautiful experience.

Back in 2017 (one year before starting my PhD), I was working as a computer vision engineer in India. I had no plans to move to the US or do a PhD. I was very lucky that I was laid off by my employer because in the scramble to find a job, I met Abhishek Sharma. Abhishek is a graduate from the University of Maryland and was working on his startup in India at that time. At that time, I spent a lot of time working on several deep learning side projects. Abhishek liked my work and felt that I could do good research. He spent countless hours trying to convince me that I should pursue a PhD. At that time, I had zero publications and hardly any research experience. I did not have the confidence that I could publish good papers. Abhishek would give strong convincing reasons why I can be a good researcher. He built that confidence in me. Because of my largely non-existent research record, finding recommendation letters from my undegrad professors would be difficult. Abhishek wrote my recommendation letter. Often times, I felt that he was more worried about my career than I was. For the rest of my life, I will support promising researchers the way Abhishek supported me. I am forever indebted to Abhishek for everything he did for me and I wish that someday my research makes a significant impact in the world and I can give him a reason to be proud of.

I want to thank my family: mother, father, brothers, sister, uncle, aunt and grandmother who provided me with unconditional love and support throughout the graduate school. I would like to thank my late maternal and paternal grandfathers for inculcating the spirit of working hard and becoming self reliant in me.

My mother is the one person who loved me unconditionally, believed in me, encouraged me and made several sacrifices for me. She swallowed her pain and let me move to the US even

though she wanted me to stay in India. None of this would have been possible without you, mom. There are no words to express my gratitude to you.

Finally, I would like to thank my committee members for taking the time to review my thesis, attend my talk and suggesting improvements.

Table of Contents

Preface ii

Dedication iv

Acknowledgements v

Table of Contents ix

List of Tables xviii

List of Figures xx

Chapter 1: Introduction 1
1.1 Motivation . . . 1
1.2 Contributions . . . 3
1.3 Organization . . . 9

I Defenses against unforeseen adversarial attacks 11

Chapter 2: Perceptual Adversarial Robustness: Defense Against Unseen Threat Models 12
2.1 Introduction . . . 12
2.2 Related Work . . . 16
2.3 Neural Perceptual Threat Model (NPTM) . . . 18
2.4 Perceptual Adversarial Attacks . . . 19
2.5 Perceptual Adversarial Training (PAT) . . . 22
2.6 Perceptual Evaluation . . . 23
2.7 Experiments . . . 26
2.8 Conclusion . . . 30
2.A Perceptual Attack Algorithms . . . 32
2.A.1 Perceptual PGD . . . 32
2.A.2 Lagrangian Perceptual Attack (LPA) . . . 37
2.A.3 Fast Lagrangian Perceptual Attack . . . 40
2.A.4 Perceptual Projection . . . 41
2.B Additional Adversarial Examples . . . 45
2.C Additional Experiments . . . 45

2.C.1 Experiment Details . . . 45
2.C.2 Layers for LPIPS Calculation . . . 46
2.C.3 Additional Perceptual Study Results . . . 47
2.C.4 Perceptual Attack Experiments . . . 47
2.C.5 PAT Experiments . . . 48
2.C.6 Common Corruptions Evaluation . . . 53

II Second-order methods for interpretability and robustness 55

Chapter 3: Understanding Impacts of Higher-Order Loss Approximations and Group Features in Deep Learning Interpretation 56
3.1 Introduction . . . 56
3.2 Problem Setup and Notation . . . 59
3.3 Understanding Impact of the Hessian in Deep Learning Interpretation . . . 62
3.3.1 A Closed-form Hessian Formula for Deep ReLU Networks . . . 63
3.3.2 Theoretical results on the Hessian Impact . . . 65
3.3.3 Empirical results on the Hessian impact . . . 66
3.4 Understanding Impact of Group-features . . . 69
3.4.1 Empirical results on the impact of group features . . . 72
3.5 Qualitative Comparision of Deep learning Interpretation Methods . . . 73
3.6 Conclusion and Future Work . . . 75
3.A Proofs . . . 77
3.A.1 Proof of Proposition 1 . . . 77
3.A.2 Proof of Theorem 2 . . . 80
3.A.3 Proof of Theorem 3 . . . 81
3.A.4 Proof of Theorem 4 . . . 81
3.A.5 Proof of Theorem 5 . . . 87
3.B Convergence of Gradient Descent to Solve CASO . . . 89
3.C Additional analysis . . . 90
3.C.1 Efficient Computation of the Hessian Matrix Using the Cholesky decomposition . . . 90
3.C.2 Saliency Visualization Methods . . . 91
3.C.3 Tightness of the $L_0 - L_1$ Relaxation . . . 92
3.C.4 Empirical analysis of the Hessian impact . . . 94
3.C.5 Comparison with existing methods . . . 97

Chapter 4: Low Curvature Activations Reduce Overfitting in Adversarial Training 101
4.1 Introduction . . . 101
4.2 Related Works . . . 105
4.3 Background . . . 107
4.3.1 Adversarial Training . . . 107
4.3.2 Robust Overfitting . . . 108
4.4 Impact of Activation Curvature on Adversarial Training . . . 109
4.4.1 Analyzing the influence of activations on robustness . . . 111

4.4.2 Activation Curvature and Generalization Gap . . . 113
4.4.3 Curvature effects with Parameteric Swish . . . 116
4.5 Does smoothness matter? . . . 118
4.6 Double descent curves . . . 119
4.7 Conclusion . . . 120

Chapter 5: Second-Order Provable Defenses against Adversarial Attacks 122
5.1 Introduction . . . 122
5.2 Related work . . . 125
5.3 Notation . . . 127
5.4 Using duality to solve the attack and certificate problems . . . 128
5.5 Curvature Bounds for deep networks . . . 131
5.5.1 Closed form expression for the Hessian . . . 131
5.5.2 Curvature bounds for Two Layer networks . . . 132
5.5.3 Curvature bounds for Deep networks . . . 134
5.6 Adversarial training with curvature regularization . . . 135
5.7 Experiments . . . 137
5.7.1 Fraction of inputs with tightest robustness certificate . . . 137
5.7.2 Comparison with existing provable defenses . . . 139
5.7.3 Comparison with existing certificates . . . 141
5.7.4 Results using local curvature bounds . . . 142
5.8 Conclusion . . . 143
5.A Implementation Details . . . 144
5.A.1 Computing the derivative of largest singular value . . . 144
5.A.2 Computing certificate using local curvature bounds . . . 144
5.A.3 Computing g, h, h_U and h_L for different activation functions . . . 145
5.B Proofs . . . 149
5.B.1 Proof of Theorem 6 . . . 149
5.B.2 Proof of Theorem 7 . . . 152
5.B.3 Proof of Lemma 3 . . . 154
5.B.4 Proof of Theorem 8 . . . 163
5.B.5 Proof of Theorem 9 . . . 170
5.B.6 Proof of Theorem 10 . . . 177
5.C Additional experiments . . . 179
5.C.1 Computing K_{lb} and K_{ub} . . . 180
5.C.2 Comparison with provable defenses . . . 181
5.C.3 Comparing Randomized Smoothing with CRT . . . 183
5.C.4 Results on CIFAR-10 . . . 184

III Lipschitz neural networks 185

Chapter 6: Fantastic Four: Differentiable Bounds on Singular Values of Convolution Layers 186
6.1 Introduction . . . 186

6.2 Notation . . . 192
6.3 Main Results . . . 193
6.3.1 Tightness analysis . . . 195
6.3.2 Gradient Computation . . . 196
6.4 Experiments . . . 197
6.4.1 Comparison with existing methods . . . 197
6.4.2 Effects on generalization . . . 197
6.4.3 Effects on provable adversarial robustness . . . 199
6.5 Conclusion . . . 201
6.A Notation for supplementary material . . . 202
6.B Review of prior works . . . 204
6.B.1 Sedghi et al's method . . . 204
6.B.2 Second-order provable robustness framework . . . 205
6.C Proofs . . . 208
6.C.1 Proof of Theorem 11 . . . 208
6.C.2 Proof of Lemma 4 . . . 215
6.D Additional analysis . . . 216
6.D.1 Computing the gradient of convolution layer . . . 216
6.D.2 Effect of increasing β on singular values . . . 217
6.D.3 Effect of increasing γ on certified robust accuracy . . . 217

Chapter 7: Skew Orthogonal Convolutions 221
7.1 Introduction . . . 221
7.2 Related work . . . 225
7.3 Notation . . . 226
7.4 Filters with Skew Symmetric Jacobians . . . 227
7.5 Skew Orthogonal Convolution layers . . . 229
7.5.1 Bounding the Approximation Error . . . 231
7.5.2 Complete Set of Skew Orthogonal Convolutions . . . 232
7.5.3 Extensions to 3D and Complex Convolutions . . . 233
7.6 Implementation details of SOC . . . 234
7.6.1 Bounding the norm of Jacobian . . . 234
7.6.2 Different input and output channels . . . 235
7.6.3 Strided convolution . . . 235
7.6.4 Number of terms for the approximation . . . 235
7.6.5 Network architecture . . . 236
7.7 Experiments . . . 237
7.7.1 Provable Defenses against Adversarial Attacks . . . 238
7.7.2 Standard Training . . . 239
7.7.3 Adversarial Training . . . 240
7.8 Discussion and Future work . . . 240
7.A Notation for supplementary material . . . 242
7.B Proofs . . . 243
7.B.1 Proof of Theorem 12 . . . 243
7.B.2 Proof of Theorem 13 . . . 246

7.B.3 Proof of Theorem 14 . . . 251
7.B.4 Proof of Theorem 15 . . . 256
7.B.5 Proof of Theorem 16 . . . 259
7.B.6 Proof of Theorem 17 . . . 261
7.B.7 Proof of Theorem 18 . . . 263
7.C MaxMin Activation function . . . 268

Chapter 8: Novel activations, regularization and certification procedures for lipschitz networks 269
8.1 Introduction . . . 269
8.2 Related work . . . 273
8.3 Problem setup and Notation . . . 274
8.4 Last Layer Normalization . . . 275
8.5 Certificate Regularization . . . 277
8.6 Householder Activation Functions . . . 278
8.7 Experiments . . . 282
8.7.1 Results on CIFAR-100 . . . 284
8.7.2 Results on CIFAR-10 . . . 284
8.8 Conclusion . . . 286
8.A Proofs . . . 288
8.A.1 Proof of Proposition 2 . . . 288
8.A.2 Proof of Theorem 19 . . . 289
8.A.3 Proof of Corollary 2 . . . 292
8.A.4 Proof of Theorem 20 . . . 293
8.B Selection of γ using cross validation . . . 298
8.C Differences between l_1 and l_2 certificate . . . 298
8.D Verification that $\sigma_\theta(z_1, z_2)$ always lies on one side of the hyperplane . . . 300
8.E Higher order Householder activation functions . . . 300
8.F Extension to higher dimensions . . . 304
8.F.1 Additional experiments . . . 306
8.F.2 Additional experiments on CIFAR-10 . . . 306
8.F.3 Additional experiments on CIFAR-100 . . . 311

Chapter 9: Novel pooling layer, faster gradient computation and second-order robustness certification procedures for lipschitz networks 316
9.1 Introduction . . . 316
9.2 Related work . . . 320
9.3 Problem setup and Notation . . . 322
9.4 Faster gradient computation for Skew Orthogonal Convolutions . . . 323
9.5 Curvature-based Robustness Certificate . . . 326
9.6 Projection pooling layers . . . 328
9.7 Experiments . . . 331
9.7.1 Results using Faster gradient computation . . . 334
9.7.2 Results using projection pooling and CRC-Lip . . . 335
9.A Proofs . . . 336

9.A.1 Proof of Theorem 21 . . . 336
9.A.2 Proof of Proposition 3 . . . 340
9.A.3 Proof of Theorem 22 . . . 341
9.A.4 Proof of Corollary 3 . . . 342
9.B Additional analysis . . . 344
9.B.1 Approximating the convolution weight gradient . . . 344
9.B.2 Iterations for $\mathbf{z}$ and $\nabla_{\mathbf{z}}\,\ell$ . . . 348

IV Understanding and mitigating model failures 350

Chapter 10: Understanding Failures of Deep Networks via Robust Feature Extraction 351
10.1 Introduction . . . 351
10.2 Related work . . . 354
10.3 Background and method overview . . . 355
10.4 Feature extraction and visualization . . . 357
10.5 Failure mode generation . . . 359
10.5.1 Automatic evaluation of decision tree . . . 361
10.6 Failure modes discovered by Barlow . . . 363
10.7 Experiments with automated evaluation . . . 365
10.8 Crowd study on feature interpretability . . . 366
10.9 Study with machine learning practitioners . . . 368
10.10 Conclusion and future work . . . 371
10.A Failure mode generation . . . 372
10.B Automatic evaluation of decision tree . . . 373
10.C Formal Definitions of Core and Spurious visual attributes . . . 381
10.D Failure modes discovered by Barlow . . . 383
10.D.1 Failure explanation for a standard model . . . 383
10.D.2 Failure explanation for a robust model . . . 387
10.E User study with ML practitioners . . . 391
10.F Comparison between the interpretations of a robust and non-robust model . . . 391
10.F.1 Class name: water jug . . . 393
10.F.2 Class name: horned viper . . . 394
10.F.3 Class name: tiger cat . . . 395
10.F.4 Class name: tape player . . . 396
10.F.5 Class name: overskirt . . . 397

Chapter 11: Salient ImageNet: How to discover spurious features in deep learning? 398
11.1 Introduction . . . 398
11.2 Related work . . . 403
11.3 A general framework for discovering Spurious Features . . . 405
11.3.1 Notation and Definitions . . . 405
11.3.2 Extracting, visualizing and selecting neural features . . . 406
11.3.3 Mechanical Turk study for discovering spurious and core features . . . 407
11.3.4 Generalization of visual attributes encoded in neural features . . . 409

11.4 The Salient Imagenet dataset . . . 410
11.5 Evaluating deep models using the Salient Imagenet dataset . . . 412
11.5.1 Core and Spurious accuracy . . . 413
11.5.2 Results from evaluating deep models on the Salient Imagenet dataset . . . 414
11.A Visualizing the neural features of a robust model . . . 417
11.A.1 Neural Activation Map (NAM) . . . 417
11.A.2 Heatmap . . . 417
11.A.3 Feature attack . . . 418
11.B Selecting the classes for discovering spurious features . . . 419
11.C Mechanical Turk study for discovering spurious features . . . 419
11.D Quality control . . . 421
11.E Mechanical Turk study for validating heatmaps . . . 421
11.F Discovering spurious visual attributes in other image datasets using our framework 422
11.G Analyzing the reasons provided by Mechanical Turk workers . . . 424
11.H Additional results from evaluating deep models on the Salient Imagenet dataset . 428
11.I Examples of spurious features . . . 430
11.I.1 Background spurious features . . . 430
11.I.2 Foreground spurious features . . . 433
11.J Diagnosing the confusing visual attribute between different classes . . . 436
11.J.1 Feature index: 1541 . . . 436
11.J.2 Feature index: 2025 . . . 438
11.K Comparing between the accuracy drops due to spurious and core neural features . 440
11.K.1 Class name: Ostrich, Train accuracy (Standard Resnet-50): 98.615% . . . 441
11.K.2 Class name: Brambling, Train accuracy (Standard Resnet-50): 95.615% . 443
11.K.3 Class name: Drake, Train accuracy (Standard Resnet-50): 95.462% . . . 445

Chapter 12: Hard ImageNet: Segmentations for Objects with Strong Spurious Cues 450
12.1 Introduction . . . 450
12.2 Related work . . . 452
12.3 Hard ImageNet Dataset . . . 454
12.3.1 Overview . . . 454
12.3.2 Collection Procedure . . . 455
12.4 Benchmarking Spurious Feature Reliance . . . 457
12.4.1 Ablation . . . 458
12.4.2 Relative Foreground Sensitivity . . . 459
12.4.3 Saliency Alignment . . . 461
12.5 Hard ImageNet Objects are Smaller and Less Centered . . . 462
12.6 Ranking Images by Spurious Feature Presence . . . 465
12.7 Improving Models with Hard ImageNet Annotations . . . 467
12.8 Discussion . . . 469
12.A Distinguishing Hard ImageNet from Related Challenge Datasets . . . 471
12.B Evaluation of Additional Pretrained Models . . . 472
12.C Overview of Salient ImageNet . . . 473
12.D Inspecting Prediction Confidence on Ablated Images . . . 476
12.E Datasheet . . . 478

12.E.1 Motivation . . . 478
12.E.2 Composition . . . 479
12.E.3 Collection Process . . . 479
12.E.4 Preprocessing/cleaning/labeling . . . 480
12.E.5 Uses . . . 481
12.E.6 Distribution . . . 481
12.E.7 Maintenance . . . 481
12.F Mechanical Turk Forms . . . 482

Chapter 13: Spuriosity Rankings: Sorting Data for Spurious Correlation Robustness 485
13.1 Introduction . . . 485
13.2 Related Work . . . 489
13.3 Defining and Computing Spuriosity . . . 491
13.4 Assessing Spurious Feature Reliance . . . 493
13.5 Closing Spurious Gaps . . . 496
13.6 Feature Reliability . . . 498
13.7 Beyond ImageNet: Lightweight Feature Discovery in New Domains . . . 502
13.A Impact and Limitations . . . 505
13.B Characterizing Feature Sensitivities . . . 505
13.C More Examples of Spuriosity Rankings . . . 509
13.D Full Pretrained Model Evaluation Results . . . 511
13.E Additional Discovered Features for Waterbirds and Celeb-A . . . 514
13.F Robust Neural Feature Annotation Details . . . 516
13.F.1 Salient ImageNet Framework . . . 516
13.F.2 On Feature Discovery Using Only One Model . . . 517
13.F.3 Our Contributions . . . 519

Chapter 14: Data-Centric Debugging: mitigating model failures via targeted data collection 524
14.1 Introduction . . . 524
14.2 Related work . . . 530
14.3 Notation . . . 531
14.4 Framework for model debugging . . . 531
14.4.1 Visual similarity matching at scale . . . 533
14.4.2 Collecting large pool of images from the web . . . 535
14.4.3 Removing images visually similar to test sets . . . 535
14.4.4 Debug-seed and -heldout sets . . . 536
14.4.5 Debug-train and -validation sets . . . 538
14.5 Experiments . . . 539
14.5.1 Baseline models . . . 540
14.5.2 Table details . . . 541
14.5.3 Discussion . . . 541
14.A Implementation details . . . 544
14.A.1 Selecting 160 classes for model debugging . . . 544
14.A.2 Set of models used in the multiple-model setting . . . 544

14.A.3 Models used for visual similarity matching . . . 545
14.A.4 Removing images "visually similar" to test-sets . . . 545
14.A.5 Algorithm for constructing debug-validation and debug-train sets . . . 547
14.B Additional results . . . 548
14.B.1 Results in the multiple-model setting . . . 548
14.B.2 Table details . . . 549
14.B.3 Discussion . . . 549
14.B.4 Examples of images from the seed-debug set . . . 551
14.B.5 Examples of images from the heldout-debug set . . . 556
14.B.6 Comparing between images discovered using weak labels and no labels . . . 557

V Conclusion **560**

Chapter 15: Conclusions and Future work **561**
15.0.1 Summary . . . 561
15.0.2 Future Directions . . . 562

List of Tables

1.1 ImageNet-100 accuracies against the union of five threat models 3
1.2 Comparison between previous methods and ours for provable robustness 5
1.3 Results for provable adversarial robustness using Lipschitz Networks 6

2.1 ImageNet-100 accuracies against the union of five threat models 15
2.2 Accuracies against various attacks for models trained with adversarial training . . 27
2.3 Comparison of adversarial training and Perceptual Adversarial Training (PAT) . . 28
2.4 Hyperparameters for adversarial training on CIFAR-10 and ImageNet-100 46
2.5 Bounds and results from the perceptual study 47
2.6 Accuracy of a PAT-trained ResNet-50 against perceptual adversarial attacks . . . 48
2.7 Accuracies against various attacks for models in the PAT ablation study 49
2.8 Accuracies with and without projection step for PAT-trained models 50
2.9 Accuracies when trained with self- and AlexNet-bounded PAT 51
2.10 Results training against StAdv and ReColorAdv with double the default bounds . 53
2.11 Robustness against common corruptions (CIFAR-10-C) 54
2.12 Robustness against common corruptions (ImageNet-100-C) 54

4.1 Performance of different activations with ResNet-18 113
4.2 Performance of PSwish with different β values 116
4.3 Performance of the LeakyReLU activation with different slope values 117

5.1 A summary of various primal and dual concepts used in the chapter 124
5.2 Certificate and attack success rates on the MNIST dataset 138
5.3 Comparison with interval-bound propagation based methods (MNIST) 140
5.4 Comparison with interval-bound propagation based methods (Fashion-MNIST) . 140
5.5 Effect of curvature regularization and CRT on certificates 141
5.6 Comparison between using global and local curvature bounds 142
5.7 Comparison with interval-bound propagation methods for MNIST ($\rho = 0.5$) . . . 181
5.8 Comparison with interval-bound propagation for Fashion-MNIST ($\rho = 0.5$) . . . 182
5.9 Comparison with interval-bound propagation methods on MNIST ($\rho = 1.58$) . . 182
5.10 Comparison between CRT and Randomized Smoothing 183
5.11 Results on CIFAR-10 dataset using only curvature regularization 184

6.1 Comparison between the exact spectral norm and our proposed bound 189

6.2 Comparing various methods for computing the norm of convolution layers 191
6.3 Tightness analysis between our proposed bound and exact method 196
6.4 Comparison between our proposed regularizer and singular value clipping 198
6.5 Comparison between robustness certificates using prior works and our method . 201
6.6 Effect of increasing β on the singular value bounds 219
6.7 Effect of curvature regularization on the certified robust accuracy 220

7.1 LipConvnet Architecture 236
7.2 Results for provable robustness against adversarial examples 237
7.3 Results for standard accuracy 239
7.4 Results for empirical robustness against adversarial attacks 239

8.1 Results for provable adversarial robustness on CIFAR-100 283
8.2 Results for provable adversarial robustness on CIFAR-10 285
8.3 Inference times for models on the CIFAR-10 and CIFAR-100 test sets 286
8.4 Selecting γ using cross-validation 298
8.5 Results on CIFAR-10 using shallow networks 307
8.6 Results on CIFAR-10 using deep networks 308
8.7 Results on CIFAR-10 using Order 2 HH activation 309
8.8 Results on CIFAR-10 dataset with and without LLN 310
8.9 Results on CIFAR-100 with LLN and without LLN 312
8.10 Results on CIFAR-100 using shallow networks 313
8.11 Results on CIFAR-100 using deeper networks 314
8.12 Results on CIFAR-100 using Order 2 HH activation 315

9.1 Results on CIFAR-10 using faster gradients, projection pooling, CRC-Lip 317
9.2 LipConvnet and LipBlock architevtures 330
9.3 Results for provable adversarial robustness on CIFAR-10 332
9.4 Results for provable adversarial robustness on CIFAR-100 333
9.5 Reduction in time per epoch using faster gradient computation 334

10.1 Fraction of leaves with an error increase of $> 10\%$ that cover $> 20\%$ of errors . . 374
10.2 Distribution of roles and years of ML experience among participants 391
10.3 Distribution of the class groupings among machine-learning practitioners 391

12.1 Final layer retraining improves faithful learning on Hard ImageNet 468
12.2 Evaluation of several pretrained models without finetuning 472

14.1 Results using DINO ViT-S/8 in our framework 529
14.2 LPIPS vs our method for finding similar images from 1 million images 530
14.3 Sizes of various datasets used in this chapter 537
14.4 Results for single-model seed and heldout sets 539
14.5 Results for multiple-model seed and heldout sets 549

List of Figures

1.1 Illustration of adversarial examples . . . 2
1.2 Illustration of spurious features . . . 2
1.3 Effect of second-order information on explainability and robustness . . . 4
1.4 Feature visualization . . . 7
1.5 Examples of visually similar images discovered using our framework . . . 8

2.1 Relationships between various adversarial threat models . . . 14
2.2 Area-proportional Venn diagram validating our threat model from Figure 2.1 . . . 14
2.3 Adversarial examples using self and externally-bounded LPA . . . 19
2.4 Results of the perceptual study described in Section 2.6 . . . 25
2.5 Distribution of different L_p and LPIPS distances for different adversarial attacks . . . 30
2.6 Adversarial examples using self and externally-bounded PPGD and LPA attacks . . . 45
2.7 Clean and robust accuracy of models trained with PAT, adversarial, standard training 52

3.1 Scatter plot showing the norm of difference between CASO and CAFO . . . 67
3.2 Comparing CASO and CAFO interpretations for log/high confidence . . . 68
3.3 Large λ_1 values lead to high sparsity ratios (η) . . . 71
3.4 A qualitative comparison of existing interpretation methods . . . 73
3.5 Error due to Hessian rank one approximation . . . 95
3.6 Scatter plot showing the Frobenius norm difference between CASO and CAFO . . . 96
3.7 Example 1 . . . 97
3.8 Example 2 . . . 98
3.9 Example 3 . . . 98
3.10 Example 4 . . . 99
3.11 Example 5 . . . 99
3.12 Example 6 . . . 100

4.1 Learning curves for robust training using different activations . . . 102
4.2 Activation functions and their first and second derivatives. . . . 104
4.3 Maximum hessian eigenvalues for different smooth activations . . . 108
4.4 Learning curves for standard and robust training using different activations . . . 110
4.5 Visualization of PSwish with different β values. . . . 116
4.6 Visualization of LeakyReLU with different k values. . . . 117
4.7 Generalization curves comparing double descent between ReLU/SiLU activations 119

5.1 Illustration of lower and upper bounds on the curvature for 2 and 3 layer networks 135
5.2 Effect of γ on lower and upper curvature bounds for 4 layer network 180

6.1 Effect of increasing β on the sum of spectral norm bounds and true spectral norms 218

7.1 Framework for the SOC layer 222
7.2 Illustration of the convolution transpose operation 227
7.3 Illustration of the invertible downsampling operation 234

8.1 Illustration of the Householder activation 279
8.2 Constructing Higher Order Householder activations 302

9.1 Example of 2D projection pooling using a piecewise linear curve 319
9.2 Illustration of the rearrangement operation and 2D projection pooling 327

10.1 Failure modes discovered using our proposed methodology 352
10.2 Barlow error analysis workflow 357
10.3 Illustration of heatmaps 359
10.4 Illustration of feature attack 359
10.5 Illustration of the ALER metric 361
10.6 For images with label purse, error rate increases by +10.94% 364
10.7 For images with prediction Rhodesian Ridgeback, error increases by +10.91% . . 364
10.8 Comparison between grouping strategies using a decision tree of depth 3. 365
10.9 Amazon Mechanical Turk questionnaire. 367
10.10 Cumulative distribution of answers in the crowd study 368
10.11 Answers of practitioners for action items on mitigating failure modes 369
10.12 Agreement scores of participants on the general evaluation of Barlow. 369
10.13 Comparison between grouping strategies using a decision tree of depth 1 375
10.14 Comparison between standard and robust models using a decision tree of depth 1 376
10.15 Comparison between decision trees of different depths for a standard model . . . 377
10.16 Comparison between grouping strategies using a decision tree of depth 3 378
10.17 Comparison between standard and robust models using a decision tree of depth 3 379
10.18 Comparison between decision trees of different depths using a robust model . . . 380
10.19 Simple graphical model relationship between core, spurious and label variables . 381
10.20 Increase in error for images with label purse 384
10.21 Increase in error for images with label monastery 384
10.22 Increase in error for images with prediction bakery 385
10.23 Increase in error for images with prediction polaroid camera 386
10.24 Increase in error for images with label tiger cat 387
10.25 Increase in error for images with label lighter 388
10.26 Increase in error for images with label prediction paper towel 389
10.27 Increase in error for images with prediction seat belt 390
10.28 Feature visualization for class water jug (robust model) 393
10.29 Feature visualization for class water jug 393
10.30 Feature visualization for class horned viper (robust model) 394
10.31 Feature visualization for class horned viper (non-robust model) 394

10.32 Feature visualization for class tiger cat (robust model) . . . 395
10.33 Feature visualization for class tiger cat (non-robust model) . . . 395
10.34 Feature visualization for class tape player (robust model) . . . 396
10.35 Feature visualization for class tape player (non-robust model) . . . 396
10.36 Feature visualization for class overskirt (robust model) . . . 397
10.37 Feature visualization for class overskirt (non-robust model) . . . 397

11.1 Examples of spurious features discovered using our framework . . . 398
11.2 Our framework for constructing the Salient Imagenet dataset . . . 401
11.3 Usefulness of heatmap and feature attack visualization . . . 408
11.4 Number of images per class in the Salient ImageNet dataset . . . 410
11.5 Heatmaps for different neural features for the class drake . . . 414
11.6 Similar accuracy drop irrespective of whether noise in core or spurious regions . . . 415
11.7 Neural activation map generation . . . 417
11.8 Feature attack generation . . . 418
11.9 Mechanical Turk study for discovering spurious features . . . 420
11.10 Mechanical Turk study for validating heatmaps . . . 425
11.11Visualization of feature for class barn spider . . . 426
11.12 Visualization of feature for class ruffed grouse . . . 427
11.13 Comparing between the core accuracy and usual accuracy of different models . . . 428
11.14 Core accuracy for different classes as noise level σ increases . . . 429
11.15 Visualization of feature for the class sorrel . . . 430
11.16 Visualization of feature for the class black swan . . . 431
11.17 Visualization of feature for the class geyser . . . 431
11.18 Visualization of feature for the class oystercatcher . . . 432
11.19 Visualization of feature for the class jacamar . . . 433
11.20 Visualization of feature for the class potter's wheel . . . 434
11.21Visualization of feature for the class desk . . . 434
11.22 Visualization of feature for the class wok . . . 435
11.23 Visualization for the class junco . . . 436
11.24 Visualization for the class titi . . . 437
11.25 Visualization for the class plate . . . 438
11.26 Visualization for the class icecream . . . 439
11.27 Visualization of feature for the class ostrich . . . 441
11.28 Visualization of feature for the class ostrich . . . 442
11.29 Visualization of feature for the class brambling . . . 443
11.30 Visualization of feature for the class brambling . . . 444
11.31 Visualization of feature for the class drake . . . 445
11.32 Visualization of feature for the class drake . . . 446
11.33 Visualization of feature for the class drake . . . 447
11.34 Visualization of feature for the class drake . . . 448
11.35 Visualization of feature for the class drake . . . 449

12.1 Hard ImageNet classes . . . 451
12.2 Agreement of individual segmentations with final averaged segmentation . . . 456

12.3 Smaller accuracy drops when Hard ImageNet objects are ablated 458
12.4 Relative Foreground Sensitivity (RFS) for DeiT and ResNet 460
12.5 GradCAMs of Hard ImageNet instances with poor saliency alignment 461
12.6 Alignment of GradCAM saliencies to object masks 462
12.7 Average object position for Hard ImageNet and RIVAL10 463
12.8 Hard ImageNet objects occupy less space in images than RIVAL10 objects 464
12.9 Hard ImageNet objects are often cropped out by standard augmentations 464
12.10 Examples of high and low spurious feature presence 466
12.11 Visualizations of the five most important features for the class Dog Sled 474
12.12 Visualizations of the five most important features for the class Patio 475
12.13 Confidence of the true class under ablation 477
12.14 Qualification exam. 482
12.15 Consent Form 483
12.16 MTurk task after passing the qualification exam and signing the consent form . . 484

13.1 Spuriosity rankings sort images based on the presence of spurious cues 485
13.2 Feature can be core/spurious depending on the class 488
13.3 Examples of feature visualizations 489
13.4 Model accuracy with spurious cues most/least present 493
13.5 Effect of spurious features depends strongly on the feature and class 494
13.6 Spurious gap varies much more across data classes than models. 495
13.7 Accuracy of linear heads trained on data subsets with different spuriosity ranks . 497
13.8 Spurious gaps and accuracy on validation samples with few spurious cues 499
13.9 Average OOD accuracy vs ImageNet accuracy for classification heads 499
13.10 Kernel density estimates of the core and spurious feature reliability distribution . 500
13.11 Samples with unreliable spurious but reliable core features and vice versa 502
13.12 Our framework can detect spurious features in unseen data 503
13.13 Examples of features that are sensitive to color, texture, shape corruptions 506
13.14 Sensitivities of class feature pairs to different corruptions 507
13.15 Images with highest/lowest spuriosity for classes with lowest spurious gaps . . . 508
13.16 Images with highest/lowest spuriosity for classes with highest spurious gaps . . . 509
13.17 Images with highest/lowest spuriosity for five randomly selected classes 510
13.18 Effective robustness for all models and effective robustness 511
13.19 Correlation of per-class spurious gaps for all model pairs 513
13.20 Core features discovered for the waterbird vs. landbird classification task 515
13.21 Core features discovered for the Celeb-A blond vs. brown hair classification task 516
13.22 Example of a landing page for single class-feature dependency 522
13.23 Screencap of our web-interface for assessing all 5000 class-feature dependencies 523

14.1 Examples of visually similar images discovered using our framework 524
14.2 Our framework for debugging models via targeted image retrieval 525
14.3 Class and context retention for nearest-neigbhor retrieval on the FOCUS dataset . 532
14.4 Example where nearest neighbors using DINO ViT-S/8 retain context better . . . 533
14.5 Selecting images without using weak labels can introduce mislabeled images. . . 536
14.6 Comparison between DCD-DINO and DCD-Random accuracy for 160 classes. . 540

14.7 Comparison between robust and standard models for visual similarity matching . 547
14.8 Difficult seed images on which our trained models make correct predictions . . . 552
14.9 Difficult seed images on which our trained models make correct predictions . . . 553
14.10 Difficult seed images on which our trained models make correct predictions . . . 554
14.11 Difficult seed images on which our trained models make correct predictions . . . 555
14.12 Difficult heldout images on which our trained models make correct predictions . 556
14.13 left: image with label banana, right: similar images 557
14.14 left: image with label candle, right: similar images 557
14.15 left: image with label church, right: similar images 558
14.16 left: image with label church, right: similar images 558
14.17 left: image with label coffee mug, right: similar images 558
14.18 left: image with label frypan, right: similar images 558
14.19 left: image with label ice cream, right: similar images 559
14.20 left: image with label toy poodle, right: similar images 559
14.21 left: image with label wine bottle, right: similar images 559

Chapter 1: Introduction

1.1 Motivation

Deep neural networks achieve impressive performance in several diverse domains such as visual recognition [1, 2, 3], speech processing [4], natural language understanding [5, 6, 7], reinforcement learning [8, 9, 10, 11] and robotics [12, 13, 14], far exceeding the performance of traditional statistical analysis methods. This success has led to their deployment in several sensitive applications such as autonomous driving, medicine, finance, surveillance etc.

The growing use of deep learning systems in such sensitive application areas with human lives at stake raises concerns about their trustworthiness and reliability in the real world. A key limitation of these models is their vulnerability to adversarial perturbations [15]: carefully crafted perturbations that are imperceptible to a human but can easily fool models into giving the incorrect predictions. A related vulnerability is the presence of spurious features [16] i.e. visual features that happen to co-occur with the main object in the training set but are not essential for predicting the same. As a result, a model that correctly predicts malignant/benign tumor for images from one hospital system may catastrophically fail on images from some other hospitals [17, 18] (e.g. malignant/benign markings made by the physicians in one hospital may not be present in the images from other hospitals). Such vulnerabilities are worrisome, reduce human trust and can be fatal in safety-critical domains such as autonomous driving, medical diagnosis.

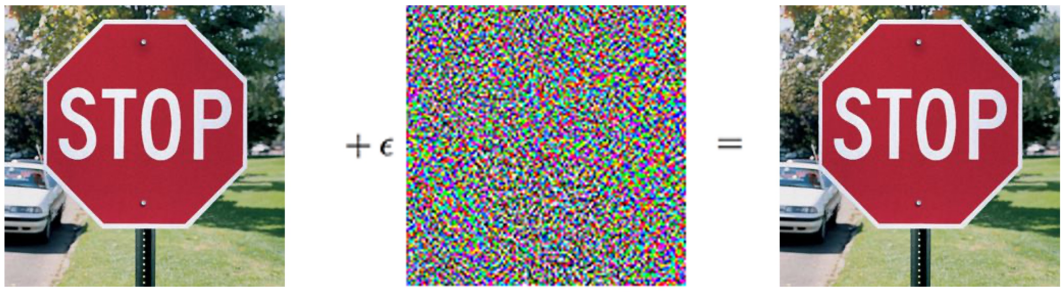
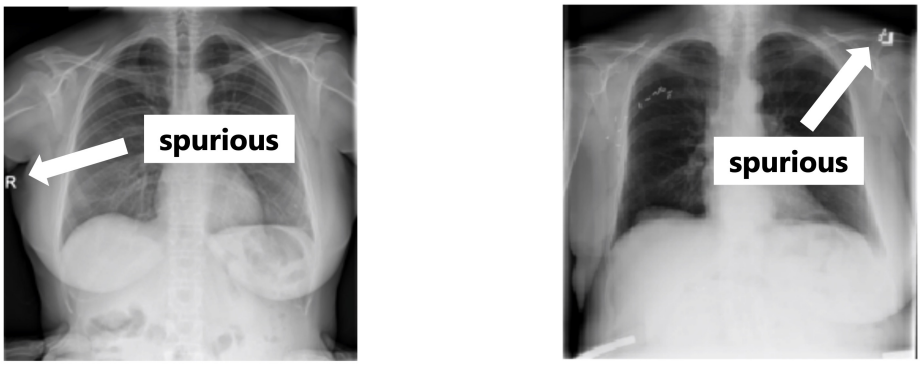


Figure 1.1: Illustration of adversarial examples phenomenon. We add a small perturbation (shown in middle) to the stop sign image (left) such that the new image looks identical to the original to a human but the model predicts it as a speed limit: 30 sign.



COVID-positive COVID-negative

Figure 1.2: Illustration of spurious features. COVID-positive (left) and -negative Chest X-ray (right). A model trained on such images may learn to predict correctly using the shown spurious artifacts but will not generalize to images from novel hospitals. Example from [18].

These phenomenon are further illustrated in Figures 1.1 and 1.2. In Figure 1.1, we illustrate how a malicious actor could make changes to the stop sign so that the self driving car classifies it as a speed limit:30 sign. Several such examples are given in [19]. In Figure 1.2, we show how a model can latch onto the spurious features prevalent in the training set of a specific hospital system. Consequently, the predictions of the model on images from other hospitals would likely be erroneous. In both these examples, these vulnerabilities can produce outputs that can lead to life threatening situations in the real world. Hence, there is a pressing need to design deep

Training	Accuracy
PGD l_∞	0.5%
PGD l_2	12.3%
JPEG	0.1%
StAdv	0.6%
ReColorAdv	0.0%
PAT (ours)	**32.5%**

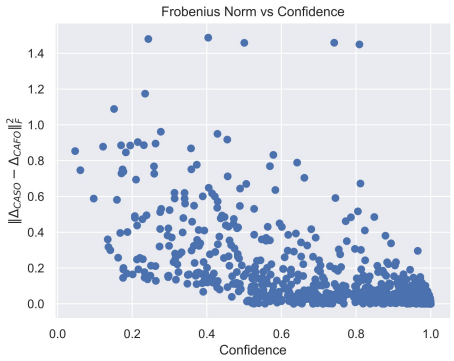
Table 1.1: ImageNet-100 accuracies against the union of five threat models (l_∞, l_2, JPEG, StAdv, ReColorAdv) for different adversarially trained classifiers and model trained using PAT.

learning systems that are robust in such settings. This is the focus of this dissertation.

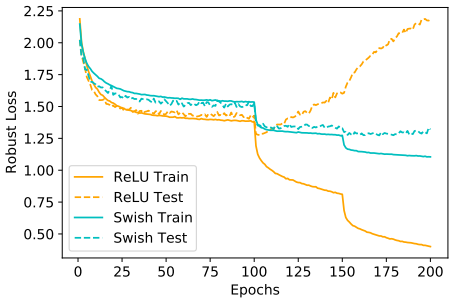
1.2 Contributions

In this dissertation, we develop several techniques for improving adversarial robustness and identifying/mitigating various failure modes of deep networks. Below, we provide a summary of these approaches.

Defenses against unforeseen adversarial attacks: A key challenge in adversarial robustness is the lack of a precise mathematical characterization of human perception. As a result, most current adversarial attacks consider restrictive adversarial threat models such as those bounded by l_2 or l_∞ distance, spatial perturbations, etc. However, models that are robust against any of these restrictive threat models are still fragile against other threat models, i.e. they have poor generalization to unforeseen attacks. To address these limitations, in Part I of the dissertation, we propose a novel adversarial training procedure against the set of all imperceptible adversarial examples. Since this set is intractable to compute without a human in the loop, we approximate it using deep neural networks. The resulting training procedure is called Perceptual Adversarial Training (PAT). Remarkably, using PAT, the trained model achieves high robustness against a



(a) Scatter plot showing the norm of the difference between first- (Δ_{CAFO}) and second-order (Δ_{CASO}) saliency maps. When the classification confidence is low, Δ_{CASO} differs significantly from Δ_{CAFO}.



(b) Learning curves for adversarial training. Using an activation function with low curvature such as SiLU prevents robust overfitting: maintaining low test robust loss compared to the network with ReLU activation function.

Figure 1.3: Left: Difference between saliency maps using second- and first-order methods, Right: Learning curves for adversarial training using activations with different curvatures.

variety of imperceptible perturbations, including l_∞, l_2, spatial, recoloring and JPEG attacks. For example, PAT on ImageNet-100 gives 32.5% accuracy against the union of these five attacks, whereas l_∞ and l_2 adversarial training give 0.5% and 12.3% accuracy, respectively (Results in Table 1.1). Compared to the next best model, PAT achieves more than double the accuracy against this union of five threat models despite not explicitly training against any of them. Thus, PAT generalizes well to unseen threat models.

Second-order methods for interpretability and robustness: Current adversarial training and saliency map methods for neural networks use first-order approximations of the loss function, neglecting higher-order terms such as the loss curvature. In Part II, we study the effect of loss curvature on deep learning interpretability, provable adversarial robustness and overfitting during adversarial training[20]. We find that second order information can lead to saliency maps that are significantly different from the first order maps (used in previous works) when the model confidence in the predicted class is low. This phenomenon is illustrated in Figure 1.3a. This sug-

Network	Training	Accuracy		% of exact certificates
		Clean	Provably Robust	
2-layer	**Ours**	**98.68%**	**69.79%**	**43.85%**
	Previous	88.48%	42.36%	_
3-layer	**Ours**	**97.43%**	**57.78%**	**22.43%**
	Previous	86.58%	42.14%	_
4-layer	**Ours**	**95.60%**	**53.19%**	**19.17%**
	Previous	82.74%	41.34%	_

Table 1.2: Comparison on MNIST for clean and provably robust accuracy (l_2 radius of 1.58), "% of exact certificates" denotes the fraction of points for which we solve the exact certificate.

gests that second-order information can be useful for inputs with low model confidence. Next, we empirically show that using smooth activation functions with low curvature significantly reduces overfitting during adversarial training (illustrated in Figure 1.3b). Finally, we introduce a method to compute the robustness certificates using second-order information. We provide verifiable conditions under which our method is able to compute *exact* certificates. We show that this condition holds for a significant fraction of examples. Our proposed method leads to improved certified accuracy on 2,3 and 4 layer networks compared to prior methods. Results on MNIST dataset are in Table 1.2. We find that (a) our model achieves significantly improved clean and provably robust accuracy compared to prior works (e.g. gain of 27% provably robust accuracy for the 2-layer network) and (b) we are able to solve for the exact certificates for significant fraction of test inputs for shallow networks (e.g. 44% inputs for the 2-layer network).

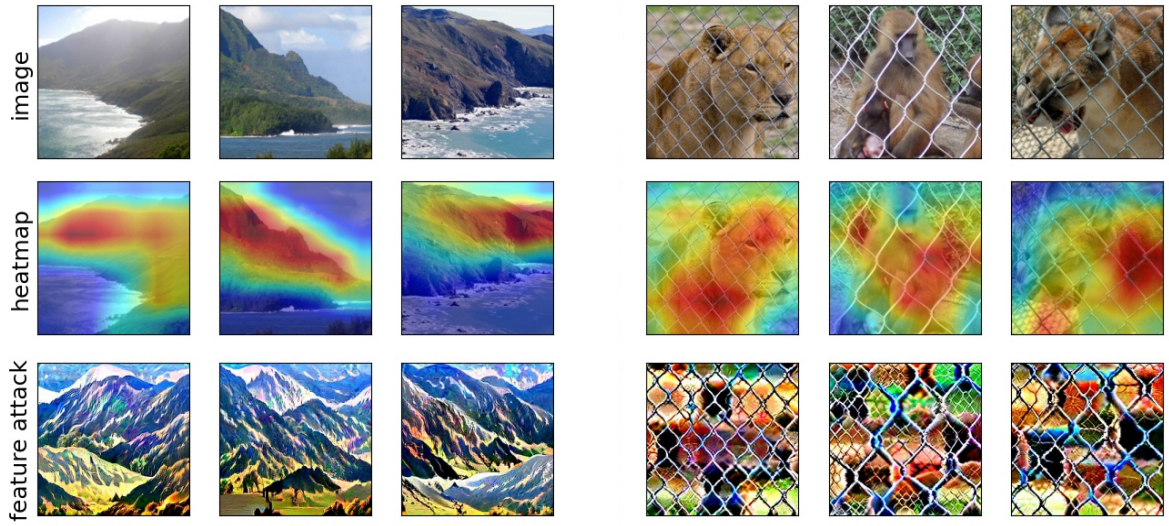
Lipschitz neural networks: In Part III, we introduce several building blocks for constructing provably 1-Lipschitz neural networks. We introduce differentiable and efficient to compute spectral bounds for convolution layers, an orthogonal convolution layer, activation functions with orthogonal jacobian for all inputs and lipschitz pooling layers with arbitrary number of learn-

# conv layers	Methods	CIFAR-10 (Accuracy)			CIFAR-100 (Accuracy)		
		Clean	**36/255**	**108/255**	**Clean**	**36/255**	**108/255**
6	Prev.	74.25%	58.01%	25.21%	46.31%	31.55%	12.52%
	Ours	**79.36%**	**67.13%**	**38.19%**	**49.73%**	**36.05%**	**16.31%**
11	Prev.	74.47%	58.48%	26.16%	45.79%	31.91%	12.78%
	Ours	**79.57%**	**66.75%**	**38.60%**	**50.45%**	**36.60%**	**16.93%**
16	Prev.	73.86%	57.39%	24.85%	45.05%	31.60%	12.93%
	Ours	**79.44%**	**66.99%**	**38.30%**	**50.79%**	**37.50%**	**17.27%**
21	Prev.	69.84%	52.10%	21.09%	39.77%	27.20%	10.49%
	Ours	**79.13%**	**66.45%**	**38.12%**	**51.84%**	**38.54%**	**18.53%**
26	Prev.	68.47%	49.92%	18.62%	34.15%	21.57%	7.97%
	Ours	**79.19%**	**66.28%**	**37.99%**	**51.59%**	**39.27%**	**19.06%**
31	Prev.	64.11%	43.39%	12.15%	29.73%	18.69%	6.00%
	Ours	**78.64%**	**66.05%**	**37.30%**	**50.97%**	**38.77%**	**19.28%**
36	Prev.	63.05%	41.71%	11.36%	27.75%	16.37%	5.40%
	Ours	**78.57%**	**65.94%**	**37.63%**	**51.42%**	**39.01%**	**20.29%**
41	Prev.	60.17%	38.86%	9.08%	25.54%	14.91%	4.40%
	Ours	**78.41%**	**65.51%**	**37.30%**	**50.11%**	**38.69%**	**20.05%**

Table 1.3: Results for provable adversarial robustness using Lipschitz Networks. "Clean" is the Clean Accuracy, "36/255" and "108/255" are the provably robust accuracy using l_2 perturbation radius of 36/255 and 108/255 respectively.

able parameters. Finally, we introduce certification procedures for these networks which unlike prior works does not require restrictive assumptions on the last layer of the model. Using our introduced techniques, we significantly improve upon the clean and provable robust accuracy for several different models. In Table 1.3, we observe CIFAR-10 gains of 5% for "clean" accuracy and 13% "108/255" accuracy. On CIFAR-100, we observe 3% for "clean" accuracy and 4% "108/255" accuracy. We observe much higher gains for deeper networks for both datasets, e.g. 18.24% (CIFAR-10) and 19.45% gap (CIFAR-100) in clean accuracy for 41 conv layers.

Understanding and mitigating model failures: In Part IV, we introduce several algorithms to identify clusters of data on which the model achieves low performance due to the presence/ab-



(a) Heatmap suffices to explain that the focus is on ground

(b) Heatmap covers both cougar and wires. Feature attack clarifies that focus is on wires.

Figure 1.4: Left and Right show visualizations for different neurons. Top row ("image") shows images with highest activation of neuron, "heatmap" shows regions of the image that highly activate the neuron, "feature attack" shows images generated by performing an adversarial attack to increase neuron activations.

sence of some visual attribute(s). Our methodology is based on the observation that for a robust model, neurons in the penultimate layer correspond to human-interpretable visual attributes. We first introduce a method for visualizing the penultimate neurons (example visualizations in Figure 1.4). Using high neuron activation values as the evidence of the presence of visual attributes (similarly, low for the absence) enables us to discover clusters of data where these attributes are either present or absent. Identifying the clusters with high error rates enables us diagnose the *failure modes* i.e. visual attributes whose presence or absence leads to poor model performance. For each ImageNet class, we label the 5 highly predictive penultimate neurons as either "core" (i.e. the visual attribute encoded in the neuron is a part of the label) and "spurious" (not part of the label). This results in the Salient ImageNet dataset. Each instance in this dataset consists of core masks (i.e. image regions covered by the main object) and spurious masks (i.e. regions

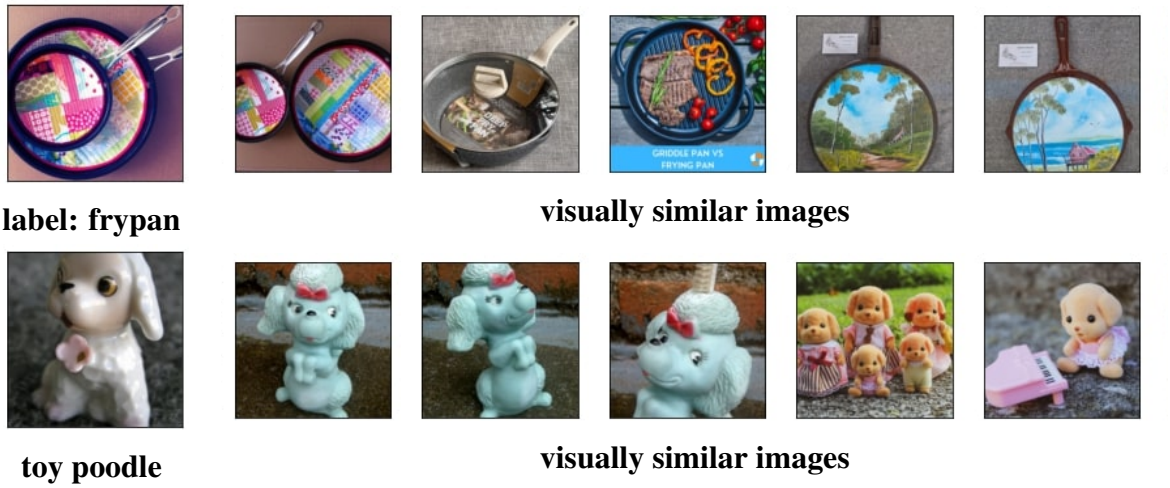


Figure 1.5: We introduce a method for mitigating model failures by discovering "visually similar" images from the web. Left: sample images on which 20 ImageNet trained models with near state-of-the-art accuracy make incorrect predictions. Our proposed debugging methodology can fix such failure modes. Right: "visually similar" images discovered using our method.

covered by spurious attributes). Using Salient ImageNet, we carry out a large scale analysis of several trained models to evaluate their sensitivity to core and spurious features. We find that even the most highly accurate models do not differentiate between the core and spurious features.

Finally, we introduce a new framework for mitigating failure modes of deep models. Our framework relies on the assumption that we have access to a very small set of samples for some specific deployment scenario where the model fails. We first divide the set into two parts (debug-seed and debug-heldout). We artificially expand the size of the debug-seed set by obtaining large number of images from the web that are "visually similar" to the images in this set. We find that adding such newly discovered images significantly improves model performance in the desired deployment setting (measured using debug-heldout set) while maintaining performance on the original test set(s) (illustrated in Figure 1.5).

1.3 Organization

Defenses against unforeseen adversarial attacks (Part I): In Chapter 2, we introduce our framework for training a model against the set of all imperceptible perturbations and how they can be used for defending against unseen adversarial attacks.

Second-order methods for interpretability and robustness (Part II): Chapter 3 presents our interpretability framework using second-order information that can be useful when the model confidence in the predicted class is low. Chapter 4 discusses the relationship between activation function curvature and robust overfitting. Chapter 5 introduces our framework for certifying adversarial robustness using the second-order information.

Lipschitz neural networks (Part III): In Chapter 6, we introduce a differentiable and efficient to compute bound over the singular values of convolution layers. We introduce the orthogonal convolution layer called SOC in Chapter 7. We introduce orthogonal activation functions, certificate regularization and a procedure to certify robustness by relaxing the orthogonality requirements of the last layer in Chapter 8. In Chapter 9, we introduce a procedure for faster weight gradient computation for SOC, a lipschitz pooling layer and a procedure to certify robustness by further relaxing the linearity requirements of the last layer.

Understanding and mitigating model failures (Part IV): In Chapter 10, we introduce our neuron visualization method and procedure for discovering failure modes of trained models. Chapter 11 introduces the Salient ImageNet dataset and analyzes the sensitivity of trained models to spurious features. Chapter 12 introduces the Hard ImageNet dataset which consists of image segmentations for the classes that were found to be highly susceptible to spurious features from our analysis. Chapter 13 introduces our spuriousity ranking framework for ranking data according to

dependencies on specific spurious attributes. Finally, Chapter 14 introduces our framework for mitigating model failures by discovering visually similar images from the web.

Part I

Defenses against unforeseen adversarial attacks

Chapter 2: Perceptual Adversarial Robustness: Defense Against Unseen Threat Models

2.1 Introduction

Many modern machine learning algorithms are susceptible to adversarial examples: carefully crafted inputs designed to fool models into giving incorrect outputs [15, 21, 22, 23]. Much research has focused on increasing classifiers' robustness against adversarial attacks [20, 24, 25]. However, existing adversarial defenses for image classifiers generally consider simple threat models. An adversarial threat model defines a set of perturbations that may be made to an image in order to produce an adversarial example. Common threat models include L_2 and L_∞ threat models, which constrain adversarial examples to be close to the original image in L_2 or L_∞ distances. Some work has proposed additional threat models which allow spatial perturbations [26, 27, 28], recoloring [29, 30, 31], and other modifications [32, 33] of an image.

There are multiple issues with these unrealistically constrained adversarial threat models. First, hardening against one threat model assumes that an adversary will only attempt attacks within that threat model. Although a classifier may be trained to be robust against L_∞ attacks, for instance, an attacker could easily generate a spatial attack to fool the classifier. One possible solution is to train against multiple threat models simultaneously [30, 34, 35, 36]. However, this

generally results in a lower robustness against any one of the threat models when compared to hardening against that threat model alone. Furthermore, not all possible threat models may be known at training time, and adversarial defenses do not usually generalize well to unforeseen threat models [37].

The ideal solution to these drawbacks would be a defense that is robust against a wide, unconstrained threat model. We differentiate between two such threat models. The *unrestricted adversarial threat model* [38] encompasses any adversarial example that is labeled as one class by a classifier but a different class by humans. On the other hand, we define the *perceptual adversarial threat model* as including all perturbations of natural images that are imperceptible to a human. Most existing narrow threat models such as L_2, L_∞, etc. are near subsets of the perceptual threat model (Figure 2.1). Some other threat models, such as adversarial patch attacks [39], may perceptibly alter an image without changing its true class and as such are contained in the unrestricted adversarial threat model. In this work, we focus on the perceptual threat model.

The perceptual threat model can be formalized given the true perceptual distance $d^*(\mathbf{x}_1, \mathbf{x}_2)$ between images $\mathbf{x}_1$ and $\mathbf{x}_2$, defined as how different two images appear to humans. For some threshold ϵ^*, which we call the perceptibility threshold, images $\mathbf{x}$ and $\mathbf{x}'$ are indistinguishable from one another as long as $d^*(\mathbf{x}, \mathbf{x}') \leq \epsilon^*$. Note that in general ϵ^* may depend on the specific input. Then, the perceptual threat model for a natural input $\mathbf{x}$ includes all adversarial examples $\widetilde{\mathbf{x}}$ which cause misclassification but are imperceptibly different from $\mathbf{x}$, i.e. $d^*(\mathbf{x}, \widetilde{\mathbf{x}}) \leq \epsilon^*$.

The true perceptual distance $d^*(\cdot, \cdot)$, however, cannot be easily computed or optimized against. To solve this issue, we propose to use a *neural perceptual distance*, an approximation of the true perceptual distance between images using neural networks. Fortunately, there have been many surrogate perceptual distances proposed in the computer vision literature such as SSIM

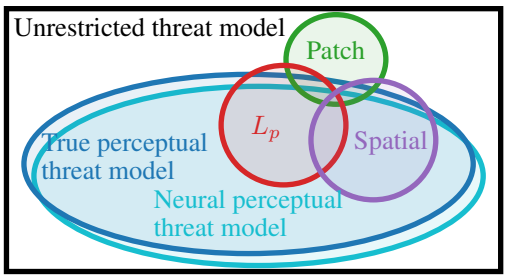


Figure 2.1: Relationships between various adversarial threat models. L_p and spatial adversarial attacks are nearly within the perceptual threat model. In this chapter, we propose a *neural perceptual threat model* (NPTM) that is based on an approximation of the true perceptual distance using neural networks.

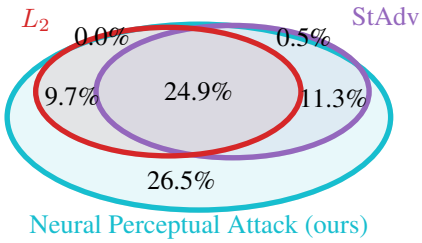


Figure 2.2: Area-proportional Venn diagram validating our threat model from Figure 2.1. Each ellipse indicates a set of vulnerable ImageNet-100 examples to one of three attacks: L_2, StAdv spatial [28], and our neural perceptual attack (LPA, Section 2.4). Percentages indicate the proportion of test examples successfully attacked. Remarkably, NPTM encompasses both types of attacks and includes additional examples not vulnerable to either.

[40]. Recently, [41] discovered that comparing the internal activations of a convolutional neural network when two different images are passed through provides a measure, Learned Perceptual Image Patch Similarity (LPIPS), that correlates well with human perception. We propose to use the LPIPS distance $d(\cdot, \cdot)$ in place of the true perceptual distance $d^*(\cdot, \cdot)$ to formalize the *neural perceptual threat model* (NPTM).

We present adversarial attacks and defenses for the proposed NPTM. Generating adversarial examples bounded by the neural perceptual distance is difficult compared to generating L_p adversarial examples because of the complexity and non-convexness of the constraint. However, we develop two attacks for the NPTM, **P**erceptual **P**rojected **G**radient **D**escent (**PPGD**) and **L**agrangian **P**erceptual **A**ttack (**LPA**) (see Section 2.4 for details). We find that LPA is by far the strongest adversarial attack at a given level of perceptibility (see Figure 2.4), reducing the most robust classifier studied to only 2.4% accuracy on ImageNet-100 (a subset of ImageNet) while remaining imperceptible. LPA also finds adversarial examples outside of any of the other threat models studied (see Figure 2.2). Thus, even if a model is robust to many narrow threat models

Training	Accuracy
PGD L_∞	0.5%
PGD L_2	12.3%
PAT (ours)	**32.5%**

Table 2.1: ImageNet-100 accuracies against the union of five threat models (L_∞, L_2, JPEG, StAdv, and ReColorAdv) for different adversarially trained (AT) classifiers and a single model trained using PAT.

(L_p, spatial, etc.), LPA can still cause serious errors.

In addition to these attacks, which are suitable for evaluating a classifier against NPTM, we also develop Fast-LPA, a more efficient version of LPA that we use in **P**erceptual **A**dversarial **T**raining (**PAT**). Remarkably, using PAT to train a neural network classifier produces a single model with high robustness against a variety of imperceptible perturbations, including L_∞, L_2, spatial, recoloring, and JPEG attacks, on CIFAR-10 and ImageNet-100 (Tables 2.2 and 2.3). For example, PAT on ImageNet-100 gives 32.5% accuracy against the *union* of these five attacks, whereas L_∞ and L_2 adversarial training give 0.5% and 12.3% accuracy, respectively (Table 2.1). PAT achieves more than *double* the accuracy against this union of five threat models despite not explicitly training against any of them. Thus, it generalizes well to unseen threat models.

Does the LPIPS distance accurately reflect human perception when it is used to evaluate adversarial examples? We performed a study on Amazon Mechanical Turk (AMT) to determine how perceptible 7 different types of adversarial perturbations such as L_∞, L_2, spatial, and recoloring attacks are at multiple threat-specific bounds. We find that LPIPS correlates well with human judgements across all the different adversarial perturbation types we examine. This indicates that the NPTM closely matches the true perceptual threat model and reinforces the utility of our perceptual attacks to measure adversarial robustness against an expansive threat model. Furthermore, this study allows calibration of a variety of attack bounds to a

single perceptibility metric. We have released our dataset of adversarial examples along with the annotations made by participants for further study. Code and data can be downloaded at `https://github.com/cassidylaidlaw/perceptual-advex`.

Our contributions may be summarized as follows:

- We define the perceptual adversarial threat model and approximate it using the NPTM with LPIPS distance. Through a perceptual study on Amazon Mechanical Turk (AMT), we verify that this approximation is close.
- We develop two adversarial attacks using the perceptual threat model. The attacks find adversarial examples that are imperceptibly different from natural images, but are not be contained in any previously defined threat models (Figure 2.2).
- We develop a defense using the perceptual threat model, Perceptual Adversarial Training, which gives close to the state-of-the-art robustness against a variety of adversarial perturbation types it was not trained against.

2.2 Related Work

Adversarial robustness: Adversarial robustness has been studied extensively for L_2 or L_∞ threat models [20, 24, 42] and non-L_p threat models such as spatial perturbations [26, 27, 28], recoloring of an image [29, 30, 31], and perturbations in the frequency domain [37]. The most popular known adversarial defense for these threat models is adversarial training [20, 25, 43] where a neural network is trained to minimize the worst-case loss in a region around the input. Recent evaluation methodologies such as Unforeseen Attack Robustness (UAR) [37] and the Unrestricted

Adversarial Examples challenge [38] have raised the problem of finding an adversarial defense which gives good robustness under more general threat models. [44] conduct a perceptual study showing that L_p threat models are a poor approximation of the perceptual threat model. [45] and [46] have developed adversarial attacks that manipulate higher-level, semantic features. [47] train with a manifold regularization term, which gives some robustness to unseen perturbation types. [48] also propose a method which gives robustness against unseen perturbation types, but requires rejecting (abstaining on) some inputs.

Perceptual similarity: Two basic similarity measures for images are the L_2 distance and the Peak Signal-to-Noise Ratio (PSNR). However, these similarity measures disagree with human vision on perturbations such as blurring and spatial transformations, which has motivated other methods including SSIM [40], MS-SSIM [49], CW-SSIM [50], HDR-VDP-2 [51] and LPIPS [41]. MAD competition [52] uses a constrained optimization technique related to our attacks to evaluate perceptual measures.

Perceptual adversarial robustness: Although LPIPS was previously proposed, it has mostly been used for development and evaluation of generative models [53, 54]. [34] first explored quantifying adversarial distortions with LPIPS distance. However, to the best of our knowledge, we are the first to apply a more accurate perceptual distance to the problem of improving adversarial robustness. As we show, adversarial defenses based on L_2 or L_∞ attacks are unable to generalize to a more diverse threat model. Our method, PAT, is the first adversarial training method we know of that can generalize to unforeseen threat models without rejecting inputs.

2.3 Neural Perceptual Threat Model (NPTM)

Since the true perceptual distance between images cannot be efficiently computed, we use approximations of it based on neural networks, i.e. neural perceptual distances. In this chapter, we focus on the LPIPS distance [41] while we note that other neural perceptual distances can also be used in our attacks and defenses. Let $g : \mathcal{X} \to \mathcal{Y}$ be a convolutional image classifier network defined on images $\mathbf{x} \in \mathcal{X}$. Let g have L layers, and let the internal activations (outputs) of the l-th layer of $g(\mathbf{x})$ for an input $\mathbf{x}$ be denoted as $g_l(\mathbf{x})$. [41] have found that normalizing and then comparing the internal activations of convolutional neural networks correlates well with human similarity judgements. Thus, the first step in calculating the LPIPS distance using the network $g(\cdot)$ is to normalize the internal activations across the channel dimension such that the L_2 norm over channels at each pixel is one. Let $\hat{g}_l(\mathbf{x})$ denote these channel-normalized activations at the l-th layer of the network. Next, the activations are normalized again by layer size and flattened into a single vector $\phi(\mathbf{x}) \triangleq \left(\frac{\hat{g}_1(\mathbf{x})}{\sqrt{w_1 h_1}}, \ldots, \frac{\hat{g}_L(\mathbf{x})}{\sqrt{w_L h_L}} \right)$ where w_l and h_l are the width and height of the activations in layer l, respectively. The function $\phi : \mathcal{X} \to \mathcal{A}$ thus maps the inputs $\mathbf{x} \in \mathcal{X}$ of the classifier $g(\cdot)$ to the resulting normalized, flattened internal activations $\phi(\mathbf{x}) \in \mathcal{A}$, where $\mathcal{A} \subseteq \mathbb{R}^m$ refers to the space of all possible resulting activations. The LPIPS distance $d(\mathbf{x}_1, \mathbf{x}_2)$ between images $\mathbf{x}_1$ and $\mathbf{x}_2$ is then defined as:

$$d(\mathbf{x}_1, \mathbf{x}_2) \triangleq \|\phi(\mathbf{x}_1) - \phi(\mathbf{x}_2)\|_2 . \tag{2.1}$$

In the original LPIPS implementation, [41] learn weights to apply to the normalized activations based on a dataset of human perceptual judgements. However, they find that LPIPS is a good

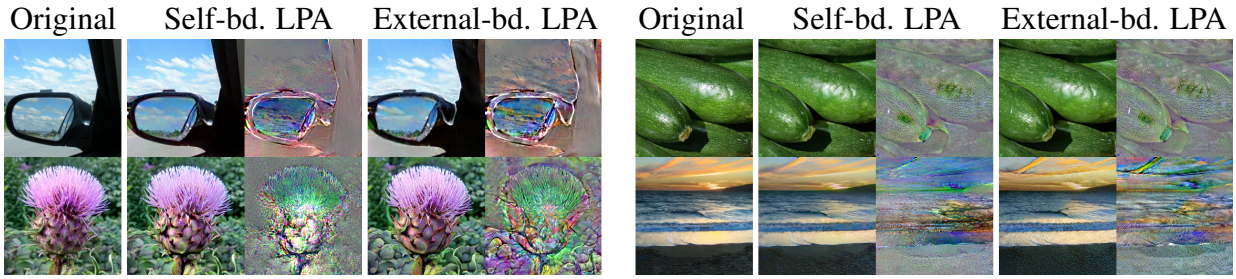


Figure 2.3: Adversarial examples generated using self-bounded and externally-bounded LPA perceptual adversarial attack (Section 2.4) with a large bound. Original images are shown in the left column and magnified differences from the original are shown to the right of the examples. See also Figure 2.6.

surrogate for human vision even without the additional learned weights; this is the version we use since it avoids the need to collect such a dataset.

Now let $f : \mathcal{X} \to \mathcal{Y}$ be a classifier which maps inputs $\mathbf{x} \in \mathcal{X}$ to labels $f(\mathbf{x}) \in \mathcal{Y}$. $f(\cdot)$ could be the same as $g(\cdot)$, or it could be a different network; we experiment with both. For a given natural input $\mathbf{x}$ with the true label y, a neural perceptual adversarial example with a perceptibility bound ϵ is an input $\widetilde{\mathbf{x}} \in \mathcal{X}$ such that $\widetilde{\mathbf{x}}$ must be perceptually similar to $\mathbf{x}$ but cause f to misclassify:

$$f(\widetilde{\mathbf{x}}) \neq y \qquad \text{and} \qquad d(\mathbf{x}, \widetilde{\mathbf{x}}) = \|\phi(\mathbf{x}) - \phi(\widetilde{\mathbf{x}})\|_2 \leq \epsilon. \tag{2.2}$$

2.4 Perceptual Adversarial Attacks

We propose attack methods which attempt to find an adversarial example with small perceptual distortion. Developing adversarial attacks that utilize the proposed neural perceptual threat model is more difficult than that of standard L_p threat models, because the LPIPS distance constraint is more complex than L_p constraints. In general, we find an adversarial example that satisfies (2.2) by maximizing a loss function $\mathcal{L}$ within the LPIPS bound. The loss function we use is similar to

the margin loss from [42], defined as

$$\mathcal{L}(f(\mathbf{x}), y) \triangleq \max_{i \neq y} \big(z_i(\mathbf{x}) - z_y(\mathbf{x})\big),$$

where $z_i(\mathbf{x})$ is the i-th logit output of the classifier $f(\cdot)$. This gives the constrained optimization

$$\max_{\widetilde{\mathbf{x}}} \quad \mathcal{L}(f(\widetilde{\mathbf{x}}), y) \qquad \text{subject to} \quad d(\mathbf{x}, \widetilde{\mathbf{x}}) = \|\phi(\mathbf{x}) - \phi(\widetilde{\mathbf{x}})\|_2 \leq \epsilon. \tag{2.3}$$

Note that in this attack problem, the classifier network $f(\cdot)$ and the LPIPS network $g(\cdot)$ are fixed. These two networks could be identical, in which case the same network that is being attacked is used to calculate the LPIPS distance that bounds the attack; we call this a *self-bounded* attack. If a different network is used to calculate the LPIPS bound, we call it an *externally-bounded* attack.

Based on this formulation, we propose two perceptual attack methods, Perceptual Projected Gradient Descent (PPGD) and Lagrangian Perceptual Attack (LPA). See Figures 2.3 and 2.6 for sample results.

Perceptual Projected Gradient Descent (PPGD): The first of our two attacks is analogous to the PGD [20] attacks used for L_p threat models. In general, these attacks consist of iteratively performing two steps on the current adversarial example candidate: (a) taking a step of a certain size under the given distance that maximizes a first-order approximation of the misclassification loss, and (b) projecting back onto the feasible set of the threat model.

Identifying the ideal first-order step is easy in L_2 and L_∞ threat models; it is the gradient of the loss function and the sign of the gradient, respectively. However, computing this step is not straightforward with the LPIPS distance, because the distance metric itself is defined by a

neural network. Following (2.3), we desire to find a step δ to maximize $\mathcal{L}(f(\mathbf{x}+\delta), y)$ such that $d(\mathbf{x}+\delta, \mathbf{x}) = \|\phi(\mathbf{x}+\delta) - \phi(\mathbf{x})\|_2 \leq \eta$, where η is the step size. Let $\hat{f}(\mathbf{x}) := \mathcal{L}(f(\mathbf{x}), y)$ for an input $\mathbf{x} \in \mathcal{X}$. Let J be the Jacobian of $\phi(\cdot)$ at $\mathbf{x}$ and $\nabla \hat{f}$ be the gradient of $\hat{f}(\cdot)$ at $\mathbf{x}$. Then, we can approximate (2.3) using a first-order Taylor's approximation of ϕ and $\hat{f}$ as follows:

$$\max_{\delta} \quad \hat{f}(\mathbf{x}) + (\nabla \hat{f})^\top \delta \qquad \text{subject to} \quad \|J\delta\|_2 \leq \eta. \tag{2.4}$$

We show that this constrained optimization can be solved in a closed form:

Lemma 1. *Let J^+ denote the pseudoinverse of J. Then the solution to (2.4) is given by*

$$\delta^* = \eta \frac{(J^\top J)^{-1}(\nabla \hat{f})}{\|(J^+)^\top (\nabla \hat{f})\|_2}.$$

See Supp. 2.A.1 for the proof. This solution is still difficult to efficiently compute, since calculating J^+ and inverting $J^\top J$ are computationally expensive. Thus, we approximately solve for δ^* using the conjugate gradient method; see Supp. 2.A.1 for details.

Perceptual PGD consists of repeatedly finding first-order optimal δ^* to add to the current adversarial example $\widetilde{\mathbf{x}}$ for a number of steps. Following each step, if the current adversarial example $\widetilde{\mathbf{x}}$ is outside the LPIPS bound, we project $\widetilde{\mathbf{x}}$ back onto the threat model such that $d(\widetilde{\mathbf{x}}, \mathbf{x}) \leq \epsilon$. The exact projection is again difficult due to the non-convexity of the feasible set. Thus, we solve it approximately with a technique based on Newton's method; see Algorithm 4 in the Supp.

Lagrangian Perceptual Attack (LPA): The second of our two attacks uses a Lagrangian relaxation of the attack problem (2.3) similar to that used by [42] for constructing L_2 and L_∞ adversarial examples. We call this attack the Lagrangian Perceptual Attack (LPA). To derive the

attack, we use the following Lagrangian relaxation of (2.3):

$$\max_{\widetilde{\mathbf{x}}} \qquad \mathcal{L}(f(\widetilde{\mathbf{x}}), y) - \lambda \max\Big(0, \|\phi(\widetilde{\mathbf{x}}) - \phi(\mathbf{x})\|_2 - \epsilon\Big). \tag{2.5}$$

The perceptual constraint cost, multiplied by λ in (2.5), is designed to be 0 as long as the adversarial example is within the allowed perceptual distance; i.e. $d(\widetilde{\mathbf{x}}, \mathbf{x}) \leq \epsilon$; once $d(\widetilde{\mathbf{x}}, \mathbf{x}) > \epsilon$, however, it increases linearly by the LPIPS distance from the original input $\mathbf{x}$. Similar to L_p attacks of [42], we adaptively change λ to find an adversarial example within the allowed perceptual distance; see Supp. 2.A.2 for details.

2.5 Perceptual Adversarial Training (PAT)

The developed perceptual attacks can be used to harden a classifier against a variety of adversarial attacks. The intuition, which we verify in Section 2.7, is that if a model is robust against neural perceptual attacks, it can demonstrate an enhanced robustness against other types of unforeseen adversarial attacks. Inspired by adversarial training used to robustify models against L_p attacks, we propose a method called Perceptual Adversarial Training (PAT).

Suppose we wish to train a classifier $f(\cdot)$ over a distribution of inputs and labels $(\mathbf{x}, y) \sim \mathcal{D}$ such that it is robust to the perceptual threat model with bound ϵ. Let $\mathcal{L}_{\text{ce}}$ denote the cross entropy (negative log likelihood) loss and suppose the classifier $f(\cdot)$ is parameterized by θ_f. Then, PAT consists of optimizing $f(\cdot)$ in a manner analogous to L_p adversarial training [20]:

$$\min_{\theta_f} \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}} \left[\max_{d(\widetilde{\mathbf{x}},\mathbf{x})\leq\epsilon} \mathcal{L}_{\text{ce}}(f(\widetilde{\mathbf{x}}), y) \right]. \tag{2.6}$$

The training formulation attempts to minimize the worst-case loss within a neighborhood of each training point $\mathbf{x}$. In PAT, the neighborhood is bounded by the LPIPS distance. Recall that the LPIPS distance is itself defined based on a particular neural network classifier. We refer to the normalized, flattened activations of the network used to define LPIPS as $\phi(\cdot)$ and θ_ϕ to refer to its parameters. We explore two variants of PAT differentiated by the choice of the network used to define $\phi(\cdot)$. In *externally-bounded* PAT, a separate, pretrained network is used to calculate $\phi(\cdot)$, the LPIPS distance $d(\cdot, \cdot)$. In *self-bounded* PAT, the same network which is being trained for classification is used to calculate the LPIPS distance, i.e. $\theta_\phi \subseteq \theta_f$. Note that in self-bounded PAT the definition of the LPIPS distance changes during the training as the classifier is optimized.

The inner maximization in (2.6) is intractable to compute exactly. However, we can use the perceptual attacks developed in Section 2.4 to approximately solve it. Since the inner maximization must be solved repeatedly during the training process, we use an inexpensive variant of the LPA attack called Fast-LPA. In contrast to LPA, Fast-LPA does not search over values of λ. It also does not include a projection step at the end of the attack, which means it may sometimes produce adversarial examples outside the training bound. While this makes it unusable for evaluation, it is fine for training. Using Fast-LPA, PAT is nearly as fast as adversarial training; see Supp. 2.A.3 and Algorithm 3 for details.

2.6 Perceptual Evaluation

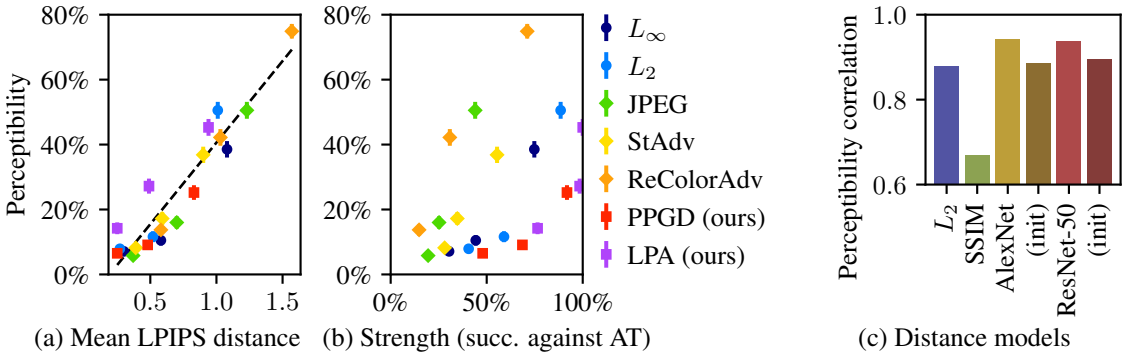
We conduct a thorough perceptual evaluation of our NPTM and attacks to ensure that the resulting adversarial examples are imperceptible. We also compare the perceptibility of perceptual adversarial attacks to five narrow threat models: L_∞ and L_2 attacks, JPEG attacks [37], spa-

tially transformed adversarial examples (StAdv) [28], and functional adversarial attacks (ReColorAdv) [30]. The comparison allows us to determine if the LPIPS distance is a good surrogate for human comparisons of similarity. It also allows us to set bounds across threat models with approximately the same level of perceptibility.

To determine how perceptible a particular threat model is at a particular bound (e.g. L_∞ attacks at $\epsilon = 8/255$), we perform an experiment based on just noticeable differences (JND). We show pairs of images to participants on Amazon Mechanical Turk (AMT), an online crowdsourcing platform. In each pair, one image is a natural image from ImageNet-100 and one image is an adversarial perturbation of the natural image, generated using the particular attack against a classifier hardened to that attack. One of the images, chosen randomly, is shown for one second, followed by a blank screen for 250ms, followed by the second image for one second. Then, participants must choose whether they believe the images are the same or different. This procedure is identical to that used by [41] to originally validate the LPIPS distance. We report the proportion of pairs for which participants report the images are "different" as the *perceptibility* of the attack. In addition to adversarial example pairs, we also include sentinel image pairs which are exactly the same; only 4.1% of these were annotated as "different."

We collect about 1,000 annotations of image pairs for each of 3 bounds for all five threat models, plus our PPGD and LPA attacks (14k annotations total for 2.8k image pairs). The three bounds for each attack are labeled as small, medium, and large; bounds with the same label have similar perceptibility across threat models (see Supp. 2.C.3 Table 2.5). The dataset of image pairs and associated annotations is available for use by the community.

To determine if the LPIPS threat model is a good surrogate for the perceptual threat model, we use various classifiers to calculate the LPIPS distance $d(\cdot, \cdot)$ between the pairs of images



(a) Mean LPIPS distance (b) Strength (succ. against AT) (c) Distance models

Figure 2.4: Results of the perceptual study described in Section 2.6 across five narrow threat models and our two perceptual attacks, each with three bounds. (a) The perceptibility of adversarial examples correlates well with the LPIPS distance (based on AlexNet) from the natural example. (b) The Lagrangian Perceptual Attack (LPA) and Perceptual PGD (PPGD) are strongest at a given perceptibility. Strength is the attack success rate against an adversarially trained classifier. (c) Correlation between the perceptibility of attacks and various distance measures: L_2, SSIM [40], and LPIPS [41] calculated using various architectures, trained and at initialization.

used in the perceptual study. For each classifier, we determine the correlation between the mean LPIPS distance it assigns to image pairs from each attack and the perceptibility of that attack (Figure 2.4c). We find that AlexNet [55], trained normally on ImageNet [56], correlates best with human perception of these adversarial examples ($r = 0.94$); this agrees with [41] who also find that AlexNet-based LPIPS correlates best with human perception (Figure 2.4). A normally trained ResNet-50 correlates similarly, but not quite as well. Because AlexNet is the best proxy for human judgements of perceptual distance, we use it for all externally-bounded evaluation attacks. Note that even with an untrained network at initialization, the LPIPS distance correlates with human perception better than the L_2 distance. This means that even during the first few epochs of self-bounded PAT, the training adversarial examples are perceptually-aligned.

We use the results of the perceptual study to investigate which attacks are strongest at a particular level of perceptibility. We evaluate each attack on a classifier hardened against that attack via adversarial training, and plot the resulting success rate against the proportion of correct

annotations from the perceptual study. Out of the narrow threat models, we find that L_2 attacks are the strongest for their perceptibility. However, our proposed PPGD and LPA attacks reduce a PAT-trained classifier to even lower accuracies (8.2% for PPGD and 0% for LPA), making it the strongest attack studied.

2.7 Experiments

We compare Perceptual Adversarial Training (PAT) to adversarial training against narrow threat models (L_p, spatial, etc.) on CIFAR-10 [57] and ImageNet-100 (the subset of ImageNet [56] containing every tenth class by WordNet ID order). We find that PAT results in classifiers with robustness against a broad range of narrow threat models. We also show that our perceptual attacks, PPGD and LPA, are strong against adversarial training with narrow threat models. We evaluate with externally-bounded PPGD and LPA (Section 2.4), using AlexNet to determine the LPIPS bound because it correlates best with human judgements (Figure 2.4c). For L_2 and L_∞ robustness evaluation we use AutoAttack [58], which combines four strong attacks, including two PGD variants and a black box attack, to give reliable evaluation.

Evaluation metrics: For both datasets, we evaluate classifiers' robustness to a range of threat models using two summary metrics. First, we compute the **union accuracy** against all narrow threat models (L_∞, L_2, StAdv, ReColorAdv, and JPEG for ImageNet-100); this is the proportion of inputs for which a classifier is robust against *all* these attacks. Second, we compute the **unseen mean accuracy**, which is the mean of the accuracies against all the threat models not trained against; this measures how well robustness generalizes to other threat models.

CIFAR-10: We test ResNet-50s trained on the CIFAR-10 dataset with PAT and adversarial

Training	Union	Unseen mean	Narrow threat models					NPTM	
			Clean	L_∞	L_2	StAdv	ReColor	PPGD	LPA
Normal	0.0	0.1	94.8	0.0	0.0	0.0	0.4	0.0	0.0
AT L_∞	1.0	19.6	86.8	49.0	19.2	4.8	54.5	1.6	0.0
TRADES L_∞	4.6	23.3	84.9	52.5	23.3	9.2	60.6	2.0	0.0
AT L_2	4.0	25.3	85.0	39.5	47.8	7.8	53.5	6.3	0.3
AT StAdv	0.0	1.4	86.2	0.1	0.2	53.9	5.1	0.0	0.0
AT ReColorAdv	0.0	3.1	93.4	8.5	3.9	0.0	65.0	0.1	0.0
AT all (random)	0.7	—	85.2	22.0	23.4	1.2	46.9	1.8	0.1
AT all (average)	14.7	—	86.8	39.9	39.6	20.3	64.8	10.6	1.1
AT all (maximum)	21.4	—	84.0	25.7	30.5	40.0	63.8	8.6	1.1
Manifold reg.	21.2	36.2	72.1	36.8	43.4	28.4	63.1	8.7	9.1
PAT-self	21.9	45.6	82.4	30.2	34.9	46.4	71.0	13.1	2.1
PAT-AlexNet	**27.8**	**48.5**	71.6	28.7	33.3	64.5	67.5	**26.6**	**9.8**

Table 2.2: Accuracies against various attacks for models trained with adversarial training and Perceptual Adversarial Training (PAT) variants on CIFAR-10. Attack bounds are $8/255$ for L_∞, 1 for L_2, 0.5 for PPGD/LPA (bounded with AlexNet), and the original bounds for StAdv/ReColorAdv. Manifold regularization is from [47]. See text for explanation of all terms.

training (AT) against six attacks (see Table 2.2): L_∞ and L_2 AutoAttack, StAdv [28], ReColorAdv [30], and PPGD and LPA. This allows us to determine if PAT gives robustness against a range of adversarial attacks. We experiment with using various models to calculate the LPIPS distance during PAT. We try using the same model both for classification and to calculate the LPIPS distance (self-bounded PAT). We also use AlexNet trained on CIFAR-10 prior to PAT (externally-bounded PAT). We find that PAT outperforms L_p adversarial training and TRADES [25], improving the union accuracy from $< 5\%$ to $> 20\%$, and nearly doubling mean accuracy against unseen threat models from 26% to 49%. Surprisingly, we find that PAT even outperforms threat-specific AT against StAdv and ReColorAdv; see Supp. 2.C.5.5 for more details. Ablation studies of PAT are presented in Supp. 2.C.5.1.

ImageNet-100: We compare ResNet-50s trained on the ImageNet-100 dataset with PAT and adversarial training (Table 2.3). Classifiers are tested against seven attacks at the medium bound

Training	Union	Unseen mean	Narrow threat models						NPTM	
			Clean	L_∞	L_2	JPEG	StAdv	ReColor	PPGD	LPA
Normal	0.0	0.1	89.1	0.0	0.0	0.0	0.0	2.4	0.0	0.0
L_∞	0.5	11.3	81.7	55.7	3.7	10.8	4.6	37.5	1.5	0.0
L_2	12.3	31.5	75.3	46.1	41.0	56.6	22.8	31.2	22.0	0.5
JPEG	0.1	7.4	84.8	13.7	1.8	74.8	0.3	21.0	0.5	0.0
StAdv	0.6	2.1	77.1	2.6	1.2	3.7	65.3	2.9	0.6	0.0
ReColorAdv	0.0	0.1	90.1	0.2	0.0	0.1	0.0	69.3	0.0	0.0
All (random)	0.9	—	78.6	38.3	26.4	61.3	1.4	32.5	16.1	0.2
PAT-self	**32.5**	**46.4**	72.6	45.0	37.7	53.0	51.3	45.1	29.2	**2.4**
PAT-AlexNet	25.5	44.7	75.7	46.8	41.0	55.9	39.0	40.8	**31.1**	1.6

Table 2.3: Comparison of adversarial training against narrow threat models and Perceptual Adversarial Training (PAT) on ImageNet-100. Accuracies are shown against seven attacks with the medium bounds from Table 2.5. PAT greatly improves accuracy (33% vs 12%) against the union of the narrow threat models despite not training against any of them. See text for explanation of all terms.

from the perceptual study (see Section 2.6 and Supp. Table 2.5). Self- and externally-bounded PAT give similar results. Both produce more than double the next highest union accuracy and also significantly increase the mean robustness against unseen threat models by around 15%.

Perceptual attacks: On both CIFAR-10 and ImageNet-100, we find that Perceptual PGD (PPGD) and Lagrangian Perceptual Attack (LPA) are the strongest attacks studied. LPA is the strongest, reducing the most robust classifier to 9.8% accuracy on CIFAR-10 and 2.4% accuracy on ImageNet-100. Also, models most robust to LPA in both cases are those that have the best union and unseen mean accuracies. This demonstrates the utility of evaluating against LPA as a proxy for adversarial robustness against a range of threat models. See Supp. 2.C.4 for further attack experiments.

Comparison to other defenses against multiple attacks: Besides the baseline of adversarially training against a single attack, we also compare PAT to adversarially training against multiple attacks [35, 36]. We compare three methods for multiple-attack training: choosing a random

attack at each training iteration, optimizing the average loss across all attacks, and optimizing the maximum loss across all attacks. The latter two methods are very expensive, increasing training time by a factor equal to the number of attacks trained against, so we only evaluate these methods on CIFAR-10. As in [36], we find that the maximum loss strategy leads to the greatest union accuracy among the multiple-attack training methods. However, PAT performs even better on CIFAR-10, *despite training against none of the attacks and taking one fourth of the time to train*. The random strategy, which is the only feasible one on ImageNet-100, performs much worse than PAT. Even the best multiple-attack training strategies still fail to generalize to the unseen neural perceptual attacks, PPGD and LPA, achieving much lower accuracy than PAT.

On CIFAR-10, we also compare PAT to manifold regularization (MR) [47], a non-adversarial training defense. MR gives union accuracy close to PAT-self, but much lower clean accuracy; for PAT-AlexNet, which gives similar clean accuracy to MR, the union accuracy is much higher.

Threat model overlap: In Figure 2.2, we investigate how the sets of images vulnerable to L_2, spatial, and perceptual attacks overlap. Nearly all adversarial examples vulnerable to L_2 or spatial attacks are also vulnerable to LPA. However, there is only partial overlap between the examples vulnerable to L_2 and spatial attacks. This helps explain why PAT results in improved robustness against spatial attacks (and other diverse threat models) compared to L_2 adversarial training.

Why does PAT work better than L_p adversarial training? In Figure 2.5, we further explain why PAT results in improved robustness against diverse threat models. We generate many adversarial examples for the L_∞, L_2, JPEG, StAdv, and ReColorAdv threat models and measure their distance from the corresponding natural inputs using L_p distances and the neural perceptual distance, LPIPS. While L_p distances vary widely, LPIPS gives remarkably comparable distances to

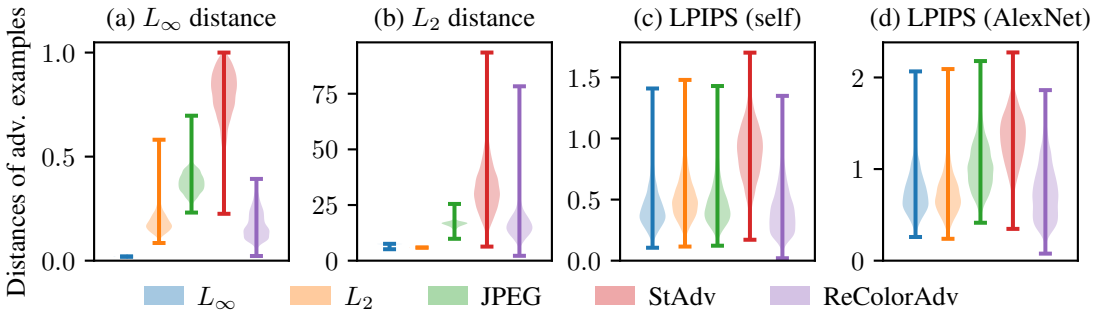


Figure 2.5: We generate samples via different adversarial attacks using narrow threat models in ImageNet-100 and measure their distances from natural inputs using L_p and LPIPS metrics. The distribution of distances for each metric and threat model is shown as a violin plot. (a-b) L_p metrics assign vastly different distances across perturbation types, making it impossible to train against all of them using L_p adversarial training. (c-d) LPIPS assigns similar distances to similarly perceptible attacks, so a single training method, PAT, can give good robustness across different threat models.

different types of adversarial examples. Covering all threat models during L_∞ or L_2 adversarial training would require using a huge training bound, resulting in poor performance. In contrast, PAT can obtain robustness against all the narrow threat models at a reasonable training bound.

Robustness against common corruptions: In addition to evaluating PAT against adversarial examples, we also evaluate its robustness to random perturbations in the CIFAR-10-C and ImageNet/C datasets [59]. We find that PAT gives increased robustness (lower relative mCE) against these corruptions compared to adversarial training; see Supp. 2.C.6 for details.

2.8 Conclusion

We have presented attacks and defenses for the neural perceptual threat model (realized by the LPIPS distance) and shown that it closely approximates the true perceptual threat model, the set of all perturbations to natural inputs which fool a model but are imperceptible to humans. Our work provides a novel method for developing defenses against adversarial attacks that generalize to unforeseen threat models. Our proposed perceptual adversarial attacks and PAT could be

extended to other vision algorithms, or even other domains such as audio and text.

Supplementary material

2.A Perceptual Attack Algorithms

2.A.1 Perceptual PGD

Recall from Section 2.4 that Perceptual PGD (PPGD) consists of repeatedly applying two steps: a first-order step in LPIPS distance to maximize the loss, followed by a projection into the allowed set of inputs. Here, we focus on the first-order step; see Supp. 2.A.4 for how we perform projection onto the LPIPS ball.

We wish to solve the following constrained optimization for the step δ given the step size η and current input $\mathbf{x}$:

$$\max_{\delta} \quad \mathcal{L}(f(\mathbf{x}+\delta), y) \qquad \text{subject to} \quad \|J\delta\|_2 \leq \eta \tag{2.7}$$

Let $\hat{f}(\mathbf{x}) := \mathcal{L}(f(\mathbf{x}), y)$ for an input $\mathbf{x} \in \mathcal{X}$. Let J be the Jacobian of $\phi(\cdot)$ at $\mathbf{x}$ and $\nabla \hat{f}$ be the gradient of $\hat{f}(\cdot)$ at $\mathbf{x}$.

Lemma 1. *The first-order approximation of (2.7) is*

$$\max_{\delta} \quad \hat{f}(\mathbf{x}) + (\nabla \hat{f})^\top \delta \qquad \textit{subject to} \quad \|J\delta\|_2 \leq \eta, \tag{2.8}$$

and can be solved in closed-form by

$$\delta^* = \eta \frac{(J^\top J)^{-1}(\nabla \hat{f})}{\|(J^+)^\top (\nabla \hat{f})\|_2}.$$

where J^+ is the pseudoinverse of J.

Proof. We solve (2.8) using Lagrange multipliers. First, we take the gradient of the objective:

$$\nabla_\delta \left[\hat{f}(\mathbf{x}) + (\nabla \hat{f})^\top \delta \right] = \nabla \hat{f}$$

We can rewrite the constraint by squaring both sides to obtain

$$\delta^\top J^\top J \delta - \epsilon^2 \leq 0$$

Taking the gradient of the constraint gives

$$\nabla_\delta \left[\delta^\top J^\top J \delta - \epsilon^2 \right] = 2 J^\top J \delta$$

Now, we set one gradient as a multiple of the other and solve for δ:

$$J^\top J \delta = \lambda (\nabla \hat{f}) \tag{2.9}$$

$$\delta = \lambda (J^\top J)^{-1} (\nabla \hat{f}) \tag{2.10}$$

Substituting into the constraint from (2.8) gives

$$\begin{aligned}
\|J\delta\|_2 &= \eta \\
\|J\lambda(J^\top J)^{-1}(\nabla \hat{f})\|_2 &= \eta \\
\lambda\|J(J^\top J)^{-1}(\nabla \hat{f})\|_2 &= \eta \\
\lambda\|((J^\top J)^{-1}J^\top)^\top(\nabla \hat{f})\|_2 &= \eta \\
\lambda\|(J^+)^\top(\nabla \hat{f})\|_2 &= \eta \\
\lambda &= \frac{\eta}{\|(J^+)^\top(\nabla \hat{f})\|_2}
\end{aligned}$$

We substitute this value of λ into (2.10) to obtain

$$\delta^* = \eta \frac{(J^\top J)^{-1}(\nabla \hat{f})}{\|(J^+)^\top(\nabla \hat{f})\|_2}. \tag{2.11}$$

□

Solution with conjugate gradient method: Calculating (2.11) directly is computationally intractable for most neural networks, since inverting $J^\top J$ and calculating the pseudoinverse of J are expensive. Instead, we approximate δ^* by using the conjugate gradient method to solve the following linear system, based on (2.9):

$$J^\top J\delta = \nabla \hat{f} \tag{2.12}$$

$\nabla \hat{f}$ is easy to calculate using backpropagation. The conjugate gradient method does not require calculating fully $J^\top J$; instead, it only requires the ability to perform matrix-vector products

$J^\top Jv$ for various vectors v.

We can approximate Jv using finite differences given a small, positive value h:

$$Jv \approx \frac{\phi(\mathbf{x} + hv) - \phi(\mathbf{x})}{h}$$

Then, we can calculate $J^\top Jv$ by introducing an additional variable u and using autograd:

$$\begin{aligned}
\nabla_u \left[(\phi(x+u))^\top Jv \right]_{u=0} &= \left[\left(\frac{d\phi}{du}(x+u) \right)^\top Jv + (\phi(x+u))^\top \frac{d}{du} Jv \right]_{u=0} \\
&= \left[\left(\frac{d\phi}{du}(x+u) \right)^\top Jv + (\phi(x+u))^\top \mathbf{0} \right]_{u=0} \\
&= \left(\frac{d\phi}{du}(x) \right)^\top Jv = J^\top Jv
\end{aligned}$$

This allows us to efficiently approximate the solution of (2.12) to obtain $(J^\top J)^{-1} \nabla \hat{f}$. We use 5 iterations of the conjugate gradient algorithm in practice.

From there, it easy to solve for λ, given that $(J^+)^\top \nabla \hat{f} = J(J^\top J)^{-1} \nabla \hat{f}$. Then, δ^* can be calculated via (2.10). See Algorithm 1 for the full attack.

Computational complexity PPGD's running time scales with the number of steps T and the number of conjugate gradient iterations K. It also depends on whether the attack is self-bounded (the same network is used for classification and the LPIPS distance) or externally-bounded (different networks are used).

For each of the T steps, $\theta(\widetilde{\mathbf{x}})$, $\nabla_{\widetilde{\mathbf{x}}} \mathcal{L}(f(\widetilde{\mathbf{x}}), y)$, and $\phi(\widetilde{\mathbf{x}} + h\delta_k)$ must be calculated once (lines 4 and 15 in Algorithm 1). This takes 2 forward passes and 1 backward pass for the self-bounded

case, and 3 forward passes and 1 backward pass for the externally-bounded case.

In addition, $J^\top Jv$ needs to be calculated (in the MULTIPLYJACOBIAN routine) $K+1$ times. Each calculation of $J^\top Jv$ requires 1 forward and 1 backward pass, assuming $\phi(\widetilde{\mathbf{x}})$ is already calculated.

Finally, the projection step takes $n+1$ forward passes for n iterations of the bisection method (see Section 2.A.4).

In all, the algorithm requires $T(K+n+4)$ forward passes and $T(K+n+3)$ backward passes in the self-bounded case. In the externally-bounded case, it requires $T(K+n+5)$ forward passes and the same number of backward passes.

Algorithm 1 Perceptual PGD (PPGD)

procedure PPGD(classifier $f(\cdot)$, LPIPS network $\phi(\cdot)$, input $\mathbf{x}$, label y, bound ϵ, step η)
  $\widetilde{\mathbf{x}} \leftarrow \mathbf{x} + 0.01 * \mathcal{N}(0, 1)$ ▷ initialize perturbations with random Gaussian noise
  for t in $1, \ldots, T$ **do** ▷ T is the number of steps
    $\nabla \hat{f} \leftarrow \nabla_{\widetilde{\mathbf{x}}} \mathcal{L}(f(\widetilde{\mathbf{x}}), y)$
    $\delta_0 \leftarrow 0$
    $r_0 \leftarrow \nabla \hat{f} - \textsc{MultiplyJacobian}(\phi, \widetilde{\mathbf{x}}, \delta_0)$
    $p_0 \leftarrow r_0$
    for k in $0, \ldots, K-1$ **do** ▷ conjugate gradient algorithm; we use $K = 5$ iterations
      $\alpha_k \leftarrow \frac{r_k^\top r_k}{p_k^\top \textsc{MultiplyJacobian}(\phi, \widetilde{\mathbf{x}}, p_k)}$
      $\delta_{k+1} \leftarrow \delta_k + \alpha_k p_k$
      $r_{k+1} \leftarrow r_k - \alpha_k \textsc{MultiplyJacobian}(\phi, \widetilde{\mathbf{x}}, p_k)$
      $\beta_k \leftarrow \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k}$
      $p_{k+1} \leftarrow r_{k+1} + \beta_k p_k$
    end for
    $m \leftarrow \|\phi(\widetilde{\mathbf{x}} + h\delta_k) - \phi(\widetilde{\mathbf{x}})\|/h$ ▷ $m \approx \|J\delta_k\|$ for small h; we use $h = 10^{-3}$
    $\widetilde{\mathbf{x}} \leftarrow (\eta/m)\delta_k$
    $\widetilde{\mathbf{x}} \leftarrow \textsc{Project}(d, \widetilde{\mathbf{x}}, \mathbf{x}, \epsilon)$
  end for
  return $\widetilde{\mathbf{x}}$
end procedure

procedure $\textsc{MultiplyJacobian}(\phi(\cdot), \widetilde{\mathbf{x}}, \mathbf{v})$ ▷ calculates $J^\top J\mathbf{v}$; J is the Jacobian of ϕ at $\widetilde{\mathbf{x}}$
  $J\mathbf{v} \leftarrow (\phi(\widetilde{\mathbf{x}} + h\mathbf{v}) - \phi(\widetilde{\mathbf{x}}))/h$ ▷ h is a small positive value; we use $h = 10^{-3}$
  $J^\top J\mathbf{v} \leftarrow \nabla_{\mathbf{u}} \left[\phi(\widetilde{\mathbf{x}} + \mathbf{u})^\top J\mathbf{v}\right]_{\mathbf{u}=0}$
  return $J^\top J\mathbf{v}$
end procedure

2.A.2 Lagrangian Perceptual Attack (LPA)

Our second attack, Lagrangian Perceptual Attack (LPA), optimizes a Lagrangian relaxation of the perceptual attack problem (2.3):

$$\max_{\widetilde{\mathbf{x}}} \quad \mathcal{L}(f(\widetilde{\mathbf{x}}), y) - \lambda \max\Big(0, \|\phi(\widetilde{\mathbf{x}}) - \phi(\mathbf{x})\|_2 - \epsilon\Big). \tag{2.13}$$

To optimize (2.13), we use a variation of gradient descent over $\widetilde{\mathbf{x}}$, starting at $\mathbf{x}$ with a small amount of noise added. We perform our modified version of gradient descent for T steps. We use a step size η, which begins at ϵ and decays exponentially to $\epsilon/10$.

At each step, we begin by taking the gradient of (2.13) with respect to $\widetilde{\mathbf{x}}$; let Δ refer to this gradient. Then, we normalize Δ to have L_2 norm 1, i.e. $\hat{\Delta} = \Delta/\|\Delta\|_2$. We wish to take a step in the direction of $\hat{\Delta}$ of size η in LPIPS distance. If we wanted to take a step of size η in L_2 distance, we could just take the step $\eta\hat{\Delta}$. However, taking a step of particular size in LPIPS distance is harder. We assume that the LPIPS distance is approximately linear in the direction $\hat{\Delta}$. We can approximate the directional derivative of the LPIPS distance in the direction $\hat{\Delta}$ using finite differences:

$$\frac{d}{d\alpha} d(\widetilde{\mathbf{x}}, \widetilde{\mathbf{x}} + \alpha\hat{\Delta}) \approx \frac{d(\widetilde{\mathbf{x}}, \widetilde{\mathbf{x}} + h\Delta)}{h} = m.$$

Here, h is a small positive value, and we assign the approximation of the directional derivative to m. Now, we can write the first-order Taylor expansion of the perceptual distance torwards the direction $\hat{\Delta}$ as follows:

$$d(\widetilde{\mathbf{x}}, \widetilde{\mathbf{x}} + \alpha\hat{\Delta}) \approx d(\widetilde{\mathbf{x}}, \widetilde{\mathbf{x}}) + m\alpha = m\alpha.$$

we want to take a step of size η. Plugging in and solving, we obtain

$$\eta = d(\widetilde{\mathbf{x}}, \widetilde{\mathbf{x}} + \alpha\hat{\Delta}) \approx m\alpha$$

$$\eta \approx m\alpha$$

$$\eta/m \approx \alpha.$$

So, the approximate step we should take is $(\eta/m)\hat{\Delta}$. We take this step at each of the T iterations

of our modified gradient descent method.

We begin with $\lambda = 10^{-2}$. After performing gradient descent, if $d(\mathbf{x}, \widetilde{\mathbf{x}}) > \epsilon$ (i.e. the adversarial example is outside the constraint) we increase λ by a factor of 10 and repeat the optimization. We repeat this entire process five times, meaning we search over $\lambda \in \{10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}\}$. Finally, if the resulting adversarial example is still outside the constraint, we project it into the threat model; see Supp. 5.

Computational complexity LPA's running time scales with the number of iterations S used to search for λ as well as the number of gradient descent steps T. $\phi(\mathbf{x})$ may be calculated once during the entire attack, which speeds it up. Then, each step of gradient descent requires 2 forward and 1 backward passes in the self-bounded case, and 3 forward and 2 backward passes in the externally-bounded case.

The projection at the end of the attack requires $n + 1$ forward passes for n iterations of the bisection method (see Section 2.A.4).

In total, the attack requires $2ST + n + 2$ forward passes and $ST + n + 2$ backward passes in the self-bounded case, and $3ST + n + 2$ forward passes and $2ST + n + 2$ backward passes in the externally-bounded case.

Algorithm 2 Lagrangian Perceptual Attack (LPA)

procedure LPA(classifier network $f(\cdot)$, LPIPS distance $d(\cdot, \cdot)$, input $\mathbf{x}$, label y, bound ϵ)
  $\lambda \leftarrow 0.01$
  $\widetilde{\mathbf{x}} \leftarrow \mathbf{x} + 0.01 * \mathcal{N}(0, 1)$ ▷ initialize perturbations with random Gaussian noise
  for i in $1, \ldots, S$ **do** ▷ we use $S = 5$ iterations to search for the best value of λ
    for t in $1, \ldots, T$ **do** ▷ T is the number of steps
      $\Delta \leftarrow \nabla_{\widetilde{\mathbf{x}}} \left[\mathcal{L}(f(\widetilde{\mathbf{x}}), y) - \lambda \max\left(0, d(\widetilde{\mathbf{x}}, \mathbf{x}\right) - \epsilon)\right]$ ▷ take the gradient of (2.5)
      $\hat{\Delta} = \Delta / \|\Delta\|_2$ ▷ normalize the gradient
      $\eta = \epsilon * (0.1)^{t/T}$ ▷ the step size η decays exponentially
      $m \leftarrow d(\widetilde{\mathbf{x}}, \widetilde{\mathbf{x}} + h\hat{\Delta})/h$ ▷ $m \approx$ derivative of $d(\widetilde{\mathbf{x}}, \cdot)$ in the direction of $\hat{\Delta}$; $h = 0.1$
      $\widetilde{\mathbf{x}} \leftarrow \widetilde{\mathbf{x}} + (\eta/m)\hat{\Delta}$ ▷ take a step of size η in LPIPS distance
    end for
    if $d(\widetilde{\mathbf{x}}, \mathbf{x}) > \epsilon$ **then**
      $\lambda \leftarrow 10\lambda$ ▷ increase λ if the attack goes outside the bound
    end if
  end for
  $\widetilde{\mathbf{x}} \leftarrow \textsc{Project}(d, \widetilde{\mathbf{x}}, \mathbf{x}, \epsilon)$
  return $\widetilde{\mathbf{x}}$
end procedure

2.A.3 Fast Lagrangian Perceptual Attack

We use the Fast Lagrangian Perceptual Attack (Fast-LPA) for Perceptual Adversarial Training (PAT, see Section 2.5). Fast-LPA is similar to LPA (Supp. 2.A.2), with two major differences. First, Fast-LPA does not search over λ values; instead, during the T gradient descent steps, λ is increased exponentially from 1 to 10. Second, we remove the projection step at the end of the attack. This means that Fast-LPA may produce adversarial examples outside the threat model. This means that Fast-LPA cannot be used for evaluation, but it is fine for training.

Computational complexity Fast-LPA's running time can be calculated similarly to LPA's (see Section 2.A.2), except that $S = 1$ and there is no projection step. Let T be the number of steps taken during the attack. Then Fast-LPA requires $2T + 1$ forward passes and $T + 1$ backward passes for the self-bounded case, and $3T + 1$ forward passes and $2T + 1$ backward passes for the

Algorithm 3 Fast Lagrangian Perceptual Attack (Fast-LPA)

procedure FASTLPA(classifier network $f(\cdot)$, LPIPS distance $d(\cdot, \cdot)$, input $\mathbf{x}$, label y, bound ϵ)
  $\widetilde{\mathbf{x}} \leftarrow \mathbf{x} + 0.01 * \mathcal{N}(0, 1)$ ▷ initialize perturbations with random Gaussian noise
  for t in $1, \ldots, T$ **do** ▷ T is the number of steps
    $\lambda \leftarrow 10^{t/T}$ ▷ λ increases exponentially
    $\Delta \leftarrow \nabla_{\widetilde{\mathbf{x}}} \left[\mathcal{L}(f(\widetilde{\mathbf{x}}), y) - \lambda \max\left(0, d(\widetilde{\mathbf{x}}, \mathbf{x}\right) - \epsilon)\right]$ ▷ take the gradient of (2.5)
    $\hat{\Delta} = \Delta / \|\Delta\|_2$ ▷ normalize the gradient
    $\eta = \epsilon * (0.1)^{t/T}$ ▷ the step size η decays exponentially
    $m \leftarrow d(\widetilde{\mathbf{x}}, \widetilde{\mathbf{x}} + h\hat{\Delta})/h$ ▷ $m \approx$ derivative of $d(\widetilde{\mathbf{x}}, \cdot)$ in the direction of $\hat{\Delta}$; $h = 0.1$
    $\widetilde{\mathbf{x}} \leftarrow \widetilde{\mathbf{x}} + (\eta/m)\hat{\Delta}$ ▷ take a step of size η in LPIPS distance
  end for
  return $\widetilde{\mathbf{x}}$
end procedure

externally-bounded case.

In comparison, PGD with T iterations requires T forward passes and T backward passes. Thus, Fast-LPA is slightly slower, requiring $T + 1$ more forward passes and no more backward passes.

2.A.4 Perceptual Projection

We explored two methods of projecting adversarial examples into the LPIPS thread model. The method we use throughout the chapter is based on Newton's method and is shown in algorithm 4. However, we also experimented with the bisection root finding method, shown in Algorithm 5 (also see Supp. 2.C.4).

In general, given an adversarial example $\widetilde{\mathbf{x}}$, original input $\mathbf{x}$, and LPIPS bound ϵ, we wish to find a projection $\widetilde{\mathbf{x}}'$ of $\widetilde{\mathbf{x}}$ such that $d(\widetilde{\mathbf{x}}', \mathbf{x}) \leq \epsilon$. Assume for this section that $d(\widetilde{\mathbf{x}}, \mathbf{x}) > \epsilon$, i.e. the current adversarial example $\widetilde{\mathbf{x}}$ is outside the bound. If $d(\widetilde{\mathbf{x}}, \mathbf{x}) \leq \epsilon$, then we can just let $\widetilde{\mathbf{x}}' = \widetilde{\mathbf{x}}$ and be done.

Newton's method The second projection method we explored uses the generalized Newton–Raphson method to attempt to find the closest projection $\widetilde{\mathbf{x}}'$ to the current adversarial example $\widetilde{\mathbf{x}}$ such that the projection is within the threat model, i.e. $d(\widetilde{\mathbf{x}}', \mathbf{x}) \leq \epsilon$. To find such a projection, we again define a function $r(\cdot)$ and look for its roots:

$$r(\widetilde{\mathbf{x}}') = d(\widetilde{\mathbf{x}}', \mathbf{x}) - \epsilon.$$

If we can find a projection $\widetilde{\mathbf{x}}'$ close to $\widetilde{\mathbf{x}}$ such that $r(\widetilde{\mathbf{x}}') \leq 0$, then this projection will be contained within the threat model, since

$$r(\widetilde{\mathbf{x}}') \leq 0 \Rightarrow d(\widetilde{\mathbf{x}}', \mathbf{x}) \leq \epsilon.$$

To find such a root, we use the generalized Newton-Raphson method, an iterative algorithm. Beginning with $\widetilde{\mathbf{x}}'_0 = \widetilde{\mathbf{x}}$, we update $\widetilde{\mathbf{x}}'$ iteratively using the step

$$\widetilde{\mathbf{x}}'_{i+1} = \widetilde{\mathbf{x}}'_i - \Big[\nabla r(\widetilde{\mathbf{x}}'_i)\Big]^+ \Big(r(\widetilde{\mathbf{x}}'_i) + s\Big),$$

where A^+ denotes the pseudoinverse of A, and s is a small positive constant (the "overshoot"), which helps the algorithm converge. We continue this process until $r(\widetilde{\mathbf{x}}'_t) \leq 0$, at which point the projection is complete.

This algorithm usually takes 2-3 steps to converge with $s = 10^{-2}$. Each step requires 1 forward and 1 backward pass to calculate $r(\widetilde{\mathbf{x}}'_t)$ and its gradient. The method also requires 1 forward pass at the beginning to calculate $\phi(\mathbf{x})$.

Algorithm 4 Perceptual Projection (Newton's Method)

procedure PROJECT(LPIPS distance $d(\cdot, \cdot)$, adversarial example $\widetilde{\mathbf{x}}$, original input $\mathbf{x}$, bound ϵ)
  $\widetilde{\mathbf{x}}'_0$
  for i in $0, \ldots$ **do**
    $r(\widetilde{\mathbf{x}}'_i) \leftarrow d(\widetilde{\mathbf{x}}'_i, \mathbf{x}) - \epsilon$
    if $r(\widetilde{\mathbf{x}}'_i) \leq 0$ **then**
      return $\widetilde{\mathbf{x}}'_i$
    end if
    $\widetilde{\mathbf{x}}'_{i+1} = \widetilde{\mathbf{x}}'_i - \Big[\nabla r(\widetilde{\mathbf{x}}'_i)\Big]^+ \Big(r(\widetilde{\mathbf{x}}'_i) + s\Big)$ ▷ s is the "overshoot"; we use $s = 10^{-2}$
  end for
end procedure

Bisection method The first projection method we explored (and the one we use throughout the chapter) attempts to find a projection $\widetilde{\mathbf{x}}'$ along the line connecting the current adversarial example $\widetilde{\mathbf{x}}$ and original input $\mathbf{x}$. Let $\delta = \widetilde{\mathbf{x}} - \mathbf{x}$. Then we can represent our final projection $\widetilde{\mathbf{x}}'$ as a point between $\mathbf{x}$ and $\widetilde{\mathbf{x}}$ as

$$\widetilde{\mathbf{x}}' = \mathbf{x} + \alpha\delta,$$

for some $\alpha \in [0, 1]$. If $\alpha = 0$, $\widetilde{\mathbf{x}}' = \mathbf{x}$; if $\alpha = 1$, $\widetilde{\mathbf{x}}' = \widetilde{\mathbf{x}}$. Now, define a function $r : [0, 1] \to \mathbb{R}$ as

$$r(\delta) = d(\mathbf{x} + \alpha\delta, \mathbf{x}) - \epsilon.$$

This function has the following properties:

1. $r(0) < 0$, since $r(0) = d(\mathbf{x}, \mathbf{x}) - \epsilon = -\epsilon$.
2. $r(1) > 0$, since $r(1) = d(\widetilde{\mathbf{x}}, \mathbf{x}) - \epsilon > 0$ because $d(\widetilde{\mathbf{x}}, \mathbf{x}) > \epsilon$.
3. $r(\alpha) = 0$ iff $d(\widetilde{\mathbf{x}}', \mathbf{x}) = \epsilon$.

We use the bisection root finding method to find a root α^* of $r(\cdot)$ on the interval $[0, 1]$, which exists since $r(\cdot)$ is continuous and because of items 1 and 2 above. By item 3, at this root, the

Algorithm 5 Perceptual Projection (Bisection Method)

procedure PROJECT(LPIPS distance $d(\cdot, \cdot)$, adversarial example $\widetilde{\mathbf{x}}$, original input $\mathbf{x}$, bound ϵ)
  $\alpha_{\text{min}}, \alpha_{\text{max}} \leftarrow 0, 1$
  $\delta \leftarrow \widetilde{\mathbf{x}} - \mathbf{x}$
  for i in $1, \ldots, n$ **do**
    $\alpha \leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}})/2$
    $\widetilde{\mathbf{x}}' \leftarrow \mathbf{x} + \alpha\delta$
    if $d(\mathbf{x}, \widetilde{\mathbf{x}}') > \epsilon$ **then**
      $\alpha_{\text{max}} \leftarrow \alpha$
    else
      $\alpha_{\text{min}} \leftarrow \alpha$
    end if
  end for
  return $\widetilde{\mathbf{x}}'$
end procedure

projected adversarial example is within the threat model:

$$d(\widetilde{\mathbf{x}}', \mathbf{x}) = d(\mathbf{x} + \alpha^*\delta, \mathbf{x}) = r(\alpha^*) + \epsilon = \epsilon$$

We use $n = 10$ iterations of the bisection method to calculate α^*. This requires $n + 1$ forward passes through the LPIPS network, since $\phi(\mathbf{x})$ must be calculated once, and $\phi(\mathbf{x} + \alpha\delta)$ must be calculated n times. See Algorithm 5 for the full projection algorithm.

2.B Additional Adversarial Examples

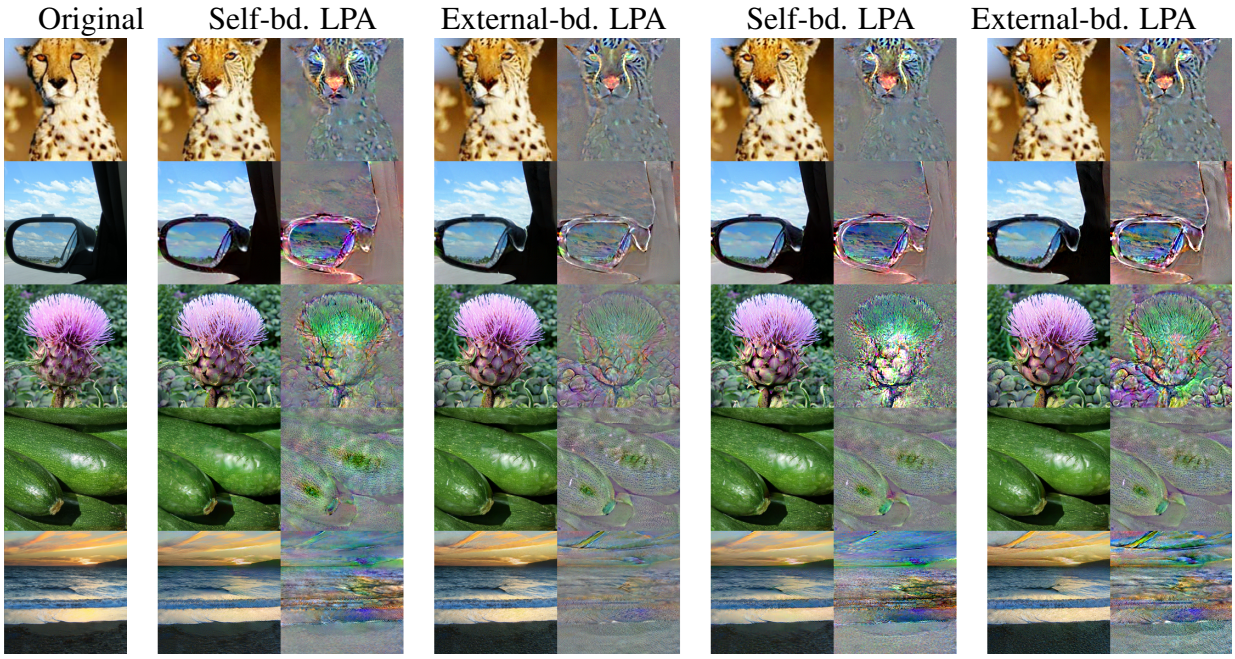


Figure 2.6: Adversarial examples generated using self-bounded and externally-bounded PPGD and LPA perceptual adversarial attacks (Section 2.4) with a large bound. Original images are shown in the left column and magnified differences from the original are shown to the right of the examples.

2.C Additional Experiments

2.C.1 Experiment Details

For all experiments, we train ResNet-50 [1] with SGD for 100 epochs. We use 10 attack iterations for training and 200 for testing, except for PPGD and LPA, where we use 40 for testing since they are more expensive. Self-bounded PAT takes about 12 hours to train for CIFAR-10 on an Nvidia RTX 2080 Ti GPU, and about 5 days to train for ImageNet-100 on 4 GPUs. We implement PPGD, LPA, and PAT using PyTorch [60].

We preprocess images after adversarial perturbation, but before classification, by stan-

dardizing them based on the mean and standard deviation of each channel for all images in the dataset. We use the default data augmentation techniques from the `robustness` library [61]. The CIFAR-10 dataset can be obtained from `https://www.cs.toronto.edu/~kriz/cifar.html`. The ImageNet-100 dataset is a subset of the ImageNet Large Scale Visual Recognition Challenge (2012) [56] including only every tenth class by WordNet ID order. It can be obtained from `http://www.image-net.org/download-images`.

Parameter	**CIFAR-10**	**ImageNet-100**
Architecture	ResNet-50	ResNet-50
Number of parameters	23,520,842	23,712,932
Optimizer	SGD	SGD
Momentum	0.9	0.9
Weight decay	2×10^{-4}	10^{-4}
Batch size	50	128
Training epochs	100	90
Initial learning rate	0.1	0.1
Learning rate drop epochs ($\times 0.1$ drop)	75, 90	30, 60, 80
Attack iterations (train)	10	10
Attack iterations (test)	200	200

Table 2.4: Hyperparameters for the adversarial training experiments on CIFAR-10 and ImageNet-100. For CIFAR-10, hyperparameters are similar to those used by [25]. For ImageNet-100, hyperparameters are similar to those used by [37].

2.C.2 Layers for LPIPS Calculation

Calculating the LPIPS distance using a neural network classifier $g(\cdot)$ requires choosing layers whose normalized, flattened activations $\phi(\cdot)$ should be compared between images. For AlexNet and VGG-16, we use the same layers to calculate LPIPS distance as do [41]. For AlexNet [55], we use the activations after each of the first five ReLU functions. For VGG-16 [62], we use the activations directly before the five max pooling layers. In ResNet-50, we use the outputs of the

conv2_x, conv3_x, conv4_x, and conv5_x layers, as listed in Table 1 of [1].

2.C.3 Additional Perceptual Study Results

Threat model	Small bound			Medium bound			Large bound		
	Bound	Perc.	Str.	Bound	Perc.	Str.	Bound	Perc.	Str.
L_∞	2	7%	30.4%	4	11%	44.3%	8	39%	55.7%
L_2	600	8%	40.6%	1200	12%	59.0%	2400	51%	88.5%
JPEG-L_∞	0.0625	6%	19.5%	0.125	16%	25.2%	0.25	51%	44.0%
StAdv	0.025	8%	28.1%	0.05	17%	34.7%	0.1	37%	55.4%
ReColorAdv	0.03	14%	14.8%	0.06	42%	30.8%	0.12	75%	71.1%
PPGD	0.25	7%	47.9%	0.5	9%	68.6%	1	25%	91.8%
LPA	0.25	14%	76.5%	0.5	27%	98.4%	1	45%	100.0%

Table 2.5: Bounds and results from the perceptual study. Each threat model was evaluated with a small, medium, and large bound. Bounds for L_2, L_∞, and JPEG attacks (first three rows) are given assuming input image is in the range $[0, 255]$. Perceptibility (perc.) is the proportion of natural input-adversarial example pairs annotated as "different" by participants. Strength (str.) is the success rate when attacking a classifier adversarially trained against that threat model (higher is stronger). Perceptual attacks (PPGD and LPA, see Section 2.4) are externally bounded with AlexNet. All experiments on ImageNet-100.

2.C.4 Perceptual Attack Experiments

We experiment with variations of the two validation attacks, PPGD and LPA (Section 2.4). We developed two methods for projecting candidate adversarial examples into the LPIPS ball surrounding a natural input (Supp. Sec 2.A.4). We attack a single model using PPGD and LPA with both projection methods. We also compare self-bounded to externally-bounded attacks. We find that LPA tends to be more powerful than PPGD. Moreover, externally-bounded LPA is extremely powerful, reducing the accuracy of a PAT-trained classifier on ImageNet-100 to just 2.4%.

Besides these experiments, we always use externally-bounded attacks with AlexNet for evaluation. AlexNet correlates with human perception of adversarial examples and provides a

standard measure of LPIPS distance; in contrast, self-bounded attacks by definition have varying bounds across evaluated models.

Attack	LPIPS model	Projection	Accuracy against PAT
PPGD	self	bisection method	43.3
PPGD	self	Newton's method	49.7
PPGD	AlexNet	bisection method	45.2
PPGD	AlexNet	Newton's method	28.7
LPA	self	bisection method	39.6
LPA	self	Newton's method	53.8
LPA	AlexNet	bisection method	4.2
LPA	AlexNet	Newton's method	2.4

Table 2.6: Accuracy of a PAT-trained ResNet-50 on ImageNet-100 against various perceptual adversarial attacks. PPGD and LPA attacks are shown self-bounded and externally-bounded with AlexNet. We also experimented with two different perceptual projection methods (see Supp. 2.A.4). Bounds are $\epsilon = 0.25$ for self-bounded attacks and $\epsilon = 0.5$ for externally-bounded attacks, since the LPIPS distance from AlexNet tends to be about twice as high as that from ResNet-50.

2.C.5 PAT Experiments

2.C.5.1 Ablation Study

We perform an ablation study of Perceptual Adversarial Training (PAT). First, we examine Fast-LPA, the training attack. We attempt training without step size (η) decay and/or without increasing λ during Fast-LPA, and find that PAT performs best with both η decay and λ increase.

Training a classifier with PAT gives robustness against a wide range of adversarial threat models (see Section 2.7). However, it tends to give low accuracy against natural, unperturbed inputs. Thus, we use a technique from [63] to improve natural accuracy in PAT-trained models: at each training step, only inputs which are classified correctly without any perturbation are attacked. In addition to increasing natural accuracy, this also improves the speed of PAT since only some inputs from each batch must be attacked. In this ablation study, we compare attacking

every input with Fast-LPA during training to only attacking the natural inputs which are already classified correctly. We find that the latter method achieves higher natural accuracy at the cost of some robust accuracy.

Ablation	Union	Unseen mean	Narrow threat models					NPTM	
			Clean	L_∞	L_2	StAdv	ReColor	PPGD	LPA
None	21.9	45.6	82.4	30.2	34.9	46.4	71.0	13.1	2.1
No η decay	17.2	48.1	82.7	37.5	43.3	39.7	72.1	10.5	1.1
No λ increase	8.1	42.1	85.1	33.7	35.9	27.8	71.1	11.6	1.0
No η decay or λ increase	19.6	49.2	82.1	36.7	41.9	44.8	73.4	10.9	0.9
Attack all inputs	26.8	46.6	74.5	29.8	33.5	56.6	66.4	24.5	6.5

Table 2.7: Accuracies against various attacks for models in the PAT ablation study. Attack bounds are $8/255$ for L_∞, 1 for L_2, 0.5 for PPGD/LPA, and the original bounds for StAdv/ReColorAdv.

2.C.5.2 Projection During Training

We choose not to add a projection step to the end of Fast-LPA during training because it slows down the attack, requiring many more passes through the network per training step. However, we tested self-bounded PAT with a projection step and found that it increased clean accuracy slightly but decreased robust accuracy significantly. We believe this is because not projecting increases the effective bound on the training attacks, leading to better robustness. To test this, we tried training without projection using a smaller bound ($\epsilon = 0.4$ instead of $\epsilon = 0.5$) and found the results closely matched the results when using projection at the larger bound. That is, PAT with projection at $\epsilon = 0.5$ is similar to PAT without projection at $\epsilon = 0.4$. Results are in Table 2.8.

Projection	Bound (ϵ)	Union	Unseen mean	Narrow threat models					NPTM	
				Clean	L_∞	L_2	StAdv	ReColor	PPGD	LPA
No	0.5	21.9	45.6	82.4	30.2	34.9	46.4	71.0	13.1	2.1
Yes	0.5	5.8	41.4	83.9	35.7	38.5	17.9	73.3	10.7	1.7
No	0.4	4.9	36.3	84.1	31.1	36.3	10.3	67.4	10.9	3.2

Table 2.8: Accuracies against various attacks for PAT-trained models on CIFAR-10, with and without a projection step during training.

2.C.5.3 Self-Bounded vs. AlexNet-Bounded PAT

Performing externally-bounded PAT with AlexNet produces more robust models on CIFAR-10 than self-bounded PAT. This is not the case on ImageNet-100, where self- and AlexNet-bounded PAT perform similarly.

There is a simple explanation for this: the effective training bound on CIFAR-10 is greater for AlexNet-bounded PAT than for self-bounded PAT. To measure this, we generate adversarial examples for all the test threat models on CIFAR-10 (L_2, L_∞, StAdv, and ReColorAdv). We find that the average LPIPS distance for all adversarial examples using AlexNet is 1.13; for a PAT-trained ResNet-50, it is 0.88. Because of this disparity, we use a lower training bound for self-bounded PAT ($\epsilon = 0.5$) than for AlexNet-bounded PAT ($\epsilon = 1$). However, this means that the average test attack has 76% greater LPIPS distance than the training attacks for self-bounded PAT, whereas the average test attack has only 13% greater LPIPS distance for AlexNet-bounded PAT. This explains why AlexNet-bounded PAT gives better robustness; it only has to generalize to slightly larger attacks on average.

We tried performing AlexNet-bounded PAT with a more comparable bound ($\epsilon = 0.7$) to self-bounded PAT. This gives the average test attack about 80% greater LPIPS distance than the training attacks, similar to self-bounded PAT. Table 2.9 shows that the results are more similar

LPIPS model	Bound (ϵ)	Union	Unseen mean	Narrow threat models					NPTM	
				Clean	L_∞	L_2	StAdv	ReColor	PPGD	LPA
Self	0.5	21.9	45.6	82.4	30.2	34.9	46.4	71.0	13.1	2.1
AlexNet	0.7	16.7	45.3	80.0	35.5	41.3	33.2	71.5	17.8	4.9
AlexNet	1.0	27.8	48.5	71.6	28.7	33.3	64.5	67.5	26.6	9.8

Table 2.9: Accuracies against various attacks for classifiers on CIFAR-10 trained with self- and AlexNet-bounded PAT using various bounds.

for self-bounded and AlexNet-bounded PAT with $\epsilon = 0.7$.

2.C.5.4 Accuracy-Robustness Tradeoff

[64] have noted that there is often a tradeoff the between adversarial robustness of a classifier and its accuracy. That is, models which have higher accuracy under adversarial attack may have lower accuracy against clean images. We observe this phenomenon with adversarial training and PAT. Since PAT gives greater robustness against several narrow threat models, models trained with it tend to have lower accuracy on clean images than models trained with narrow adversarial training. In Figure 2.7, we show the robust and clean accuracies of several models trained on CIFAR-10 and ImageNet-100 with PAT and adversarial training. While some PAT models have lower clean accuracy than adversarially trained models, at least one PAT model on each dataset surpasses the Pareto frontier of the accuracy-robustness tradeoff for adversarial training. That is, there are PAT-trained models on both datasets with both higher robust accuracy *and* higher clean accuracy than adversarial training.

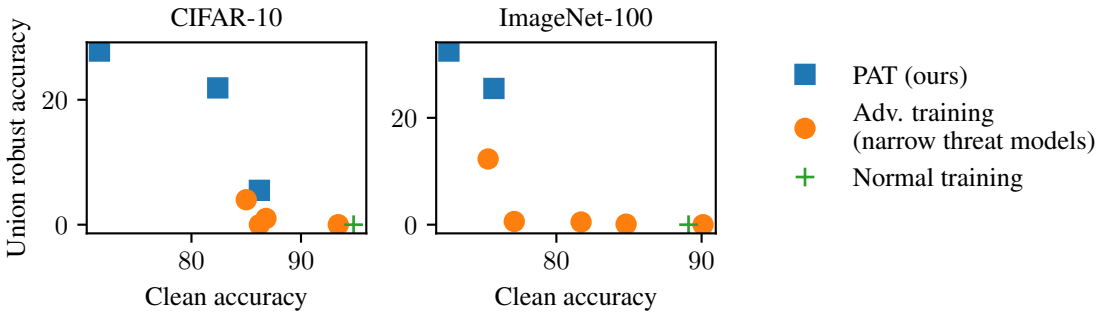


Figure 2.7: Several classifiers trained with PAT, adversarial training, and normal training on CIFAR-10 and ImageNet-100 are plotted with their clean accuracy and accuracy against the union of narrow threat models (see Section 2.7 for robustness evaluation methodology). PAT models on both datasets outperform adversarial trained models in both clean and robust accuracy.

2.C.5.5 Performance Against StAdv and ReColorAdv

It was surprising to find that PAT outperformed threat-specific adversarial training (AT) against the StAdv and ReColorAdv attacks on CIFAR-10 (it does not do so on ImageNet-100). In Table 2.2 (partially reproduced in Table 2.10 below), PAT-AlexNet improves robustness over AT against StAdv from 54% to 65%; PAT-self improves robustness over AT against ReColorAdv from 65% to 71%.

We conjecture that, for these threat models, this is because training against a wider set of perturbations at training time helps generalize robustness to new inputs at test time, even within the same threat model. To test this, we additionally train classifiers using adversarial training against the StAdv and ReColorAdv attacks with *double* the default bound. The results are shown in Table 2.10 below. We find that, because these classifiers are exposed to a wider range of spatial and recoloring perturbations during training, they perform better than PAT against those attacks at test time (76% vs 65% for StAdv and 81% vs 71% for ReColorAdv).

This suggests that PAT not only improves robustness against a wide range of adversarial threat models, it can actually improve robustness over threat-specific adversarial training by

Training	Union	Unseen mean	Narrow threat models					NPTM	
			Clean	L_∞	L_2	StAdv	ReColor	PPGD	LPA
AT StAdv	0.0	1.4	86.2	0.1	0.2	53.9	5.1	0.0	0.0
AT StAdv (double ϵ)	0.2	3.7	83.9	0.2	0.5	76.1	13.9	0.0	0.0
AT ReColorAdv	0.0	3.1	93.4	8.5	3.9	0.0	65.0	0.1	0.0
AT ReColorAdv (double ϵ)	0.0	5.3	92.0	12.5	8.6	0.3	81.2	0.4	0.0
PAT-self	21.9	45.6	82.4	30.2	34.9	46.4	71.0	13.1	2.1
PAT-AlexNet	27.8	48.5	71.6	28.7	33.3	64.5	67.5	26.6	9.8

Table 2.10: Results of our experiments training against StAdv and ReColorAdv on CIFAR-10 with double the default bounds. Columns are identical to Table 2.2.

incorporating a wider range of attacks during training.

2.C.6 Common Corruptions Evaluation

We evaluate the robustness of PAT-trained models to common corruptions in addition to adversarial examples on CIFAR-10 and ImageNet-100. In particular, we test PAT-trained classifiers on CIFAR-10-C and ImageNet-100-C, where ImageNet-100-C is the 100-class subset of ImageNet-C formed by taking every tenth class [59]. These datasets are based on random corruptions of CIFAR-10 and ImageNet, respectively, using 15 perturbation types with 5 levels of severity. The perturbation types are split into four general categories: "noise," "blur," "weather," and "digital."

The metric we use to evaluate PAT against common corruptions is mean relative corruption error (relative mCE). The relative corruption error is defined by [59] for a classifier f and corruption type c as

$$\text{Relative CE}_c^f = \frac{\sum_{s=1}^5 E_{s,c}^f - E_{\text{clean}}^f}{\sum_{s=1}^5 E_{s,c}^{\text{AlexNet}} - E_{\text{clean}}^{\text{AlexNet}}}$$

where $E_{s,c}^f$ is the error of classifier f against corruption type c at severity level s, and E_{clean}^f is the error of classifier f on unperturbed inputs. The relative mCE is defined as the mean relative CE

Training	Perturbation Type				
	Noise	Blur	Weather	Digital	All
Normal	1.72	1.57	1.03	1.87	1.59
L_∞	0.24	0.46	0.96	0.61	0.57
L_2	**0.16**	0.39	1.02	0.61	0.54
PAT-self	0.22	**0.30**	0.90	0.58	0.50
PAT-AlexNet	0.22	0.34	**0.88**	**0.56**	**0.49**

Table 2.11: Robustness of classifiers trained with adversarial training and PAT against common corruptions in the CIFAR-10-C dataset. Results are reported as relative mCE (lower is better). PAT improves robustness against common corruptions overall and for three of the specific perturbation categories.

Training	Perturbation Type				
	Noise	Blur	Weather	Digital	All
Normal	0.85	0.57	**0.56**	0.35	0.55
L_∞	0.54	0.41	0.62	0.25	0.42
L_2	0.41	0.36	0.71	0.27	0.41
PAT-self	**0.39**	0.36	0.60	**0.24**	**0.37**
PAT-AlexNet	0.49	**0.35**	0.61	**0.24**	0.39

Table 2.12: Robustness of classifiers trained with adversarial training and PAT against common corruptions in the ImageNet-100-C dataset. Results are reported as relative mCE (lower is better).

over all perturbation types.

The relative mCE for classifiers trained with normal training, adversarial training, and PAT is shown in Tables 2.11 and 2.12. PAT gives better robustness (lower relative mCE) against common corruptions on both CIFAR-10-C and ImageNet-100-C. The only category of perturbations where L_2 adversarial training outperforms PAT is "noise" on CIFAR-10-C, which makes sense because Gaussian and other types of noise are symmetrically distributed in an L_2 ball. For the other perturbation types and on ImageNet-100-C, PAT outperforms L_2 and L_∞ adversarial training, indicating that robustness against a wider range of worst-case perturbations also gives robustness against a wider range of random perturbations.

Part II

Second-order methods for interpretability and robustness

Chapter 3: Understanding Impacts of Higher-Order Loss Approximations and Group Features in Deep Learning Interpretation

3.1 Introduction

The growing use of deep learning in sensitive applications such as medicine, autonomous driving, and finance raises concerns about human trust in machine learning systems. For trained models, a central question is test-time *interpretability*: how can humans understand the reasoning behind model predictions? A common interpretation approach is to identify the importance of each input feature for a model's prediction. A saliency map can then visualize the important features, e.g., the pixels of an image [65, 66] or words in a sentence [67].

Several approaches have been proposed to tackle this problem. For example, [65] computes the gradient of the class score with respect to the input. [68] computes the average gradient-based importance values generated from several noisy versions of the input. [66] defines a *baseline*, which represents an input absent of information and determines feature importance by accumulating gradient information along the path from the baseline to the original input. [69] builds interpretable neural networks by learning basis concepts that satisfy an interpretability criteria. Although these methods can produce visually pleasing results, they often make weak model approximations [70, 71].

Existing deep learning interpretation methods mainly rely on two key assumptions:

- **The gradient-based loss surrogate assumption:** For computational efficiency, several existing methods (e.g. [65, 66, 68]) assume that the loss function is almost linear at the test point. Thus, they use variations of the input gradient to compute feature importance.
- **The isolated feature importance assumption:** Current methods evaluate the importance of each feature in isolation, assuming all other features are fixed. Features, however, may have complex inter-dependencies that can be learned by the model.

In this chapter, we study the impact of relaxing these assumptions in deep learning interpretation. To relax the first assumption, we use the second-order approximation of the loss function by keeping the Hessian term in its Taylor's expansion. For a deep ReLU network and the cross entropy loss function, we compute the Hessian term in *closed-form*. Using the closed-form formula for the Hessian matrix, we prove the following result:

Theorem 1 (informal version)**.** *If the probability of the predicted class is close to one and the number of classes is large, solutions of the first-order and second-order interpretation methods are sufficiently close to each other.*

We present a formal version of this result in Theorem 5. We validate this result empirically as well. For example, in ImageNet which has more than 1,000 classes, we show that incorporating the Hessian term in deep learning interpretation has small impact for most images. This is consistent with our theory.

The key proof idea of this result follows from the fact that when the number of classes is large and the confidence in the predicted class is high, the Hessian of the loss function is approximately of rank one i.e, the largest eigenvalue squared is significantly larger than the sum

of squared remaining eigenvalues. Moreover, the corresponding eigenvector is approximately parallel to the gradient vector (Theorem 4). This causes the first-order and second-order methods to perform similarly. Note that this result can be extended to some other related problems such as adversarial examples where most common methods are based on the first-order approximation of the loss function [24, 42, 72].

In the second part of the chapter, we relax the isolated feature importance assumption of current approaches. To incorporate feature inter-dependencies in the interpretation method, we define the importance function over *subsets* of features, referred to as *group-features*. We adjust the subset size on a per-example basis using an unsupervised approach, making the interpretation method *context-aware*. Including group-features in the interpretation makes the optimization combinatorial. To circumvent computational issues, we use an $L_0 - L_1$ relaxation as is common in compressed sensing [73, 74], the LASSO regression [75], etc. To efficiently compute a solution for the relaxed optimization, we employ the proximal gradient descent [76]. Our empirical results indicate that incorporating group-features removes noise and makes the interpretation visually coherent with the object of interest. We refer to our interpretation method based on first order (gradient) information as the CAFO (Context-Aware First Order) interpretation. Similarly, the method based on second-order information is called the CASO (Context-Aware Second Order) interpretation. Code is available at https://github.com/singlasahil14/CASO.

In what follows, we explain these results in more details. All proofs have been presented in Supp. Section 3.A.

3.2 Problem Setup and Notation

Consider a prediction problem from input variables (features) $X \in \mathcal{X} \subset \mathbf{R}^d$ to an output variable $Y \in \mathcal{Y}$. For example, in the image classification problem, $\mathcal{X}$ is the space of images and $\mathcal{Y}$ is the set of labels $\{1, ..., c\}$. We observe m samples from these variables, namely $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)\}$. Let $\mathbb{P}_{X,Y}$ be the observed empirical distribution.[1] The empirical risk minimization (ERM) approach computes the optimal predictor $f_\theta : \mathcal{X} \to \mathcal{Y}$ for a loss function $\ell(\cdot, \cdot)$ using the following optimization:

$$\min_{\theta \in \Theta} \ \mathbb{E}_{\mathbb{P}_{X,Y}} \left[\ell \left(f_\theta(\mathbf{x}), y \right) \right]. \tag{3.1}$$

Let $\mathcal{S}$ be a subset of $[d] := \{1, 2, ..., d\}$ with cardinality $|\mathcal{S}|$. For a given sample $(\mathbf{x}, y)$, let $\mathbf{x}(\mathcal{S})$ indicate the features of $\mathbf{x}$ in positions $\mathcal{S}$. We refer to $\mathbf{x}(\mathcal{S})$ as a group-feature of $\mathbf{x}$. The importance of a group-feature $\mathbf{x}(\mathcal{S})$ is proportional to the change in the loss function when $\mathbf{x}(\mathcal{S})$ is perturbed. We select the group-feature with maximum importance and visualize that subset in a saliency map.

Definition 1 (Group-Feature Importance Function)**.** *Let θ^* be the optimizer of the ERM problem (3.1). For a sample $(\mathbf{x}, y)$, we define the group-feature importance function $I^{k,\rho}_{\theta^*}(\mathbf{x}, y)$ as follows:*

$$\begin{aligned} I^{k,\rho}_{\theta^*}(\mathbf{x}, y) := \max_{\tilde{\mathbf{x}}} \ & \ell \left(f_{\theta^*}(\tilde{\mathbf{x}}), y \right) \\ & \|\tilde{\mathbf{x}} - \mathbf{x}\|_0 \leq k, \\ & \|\tilde{\mathbf{x}} - \mathbf{x}\|_2 \leq \rho, \end{aligned} \tag{3.2}$$

[1] Note that for simplicity, we hide the dependency of $\mathbb{P}_{X,Y}$ on m.

where $\|.\|_0$ counts the number of non-zero elements of its argument (known as the L_0 norm). The parameter k characterizes an upper bound on the cardinality of the group-features. The parameter ρ characterizes an upper bound on the L_2 norm of feature perturbations.

If $\tilde{\mathbf{x}}^*$ is the solution of optimization (3.2), then the vector $|\tilde{\mathbf{x}}^* - \mathbf{x}|$ is the feature importance values that are visualized in the saliency map. Note, when $k = 1$ this definition simplifies to current feature importance formulations which consider features in isolation. When $k > 1$, our formulation can capture feature interdependencies. Parameters k and ρ in general depend on the test sample $\mathbf{x}$ (i.e., the size of the group-features are different for each image and model). We introduce an unsupervised metric to determine these parameters in Section 3.4.1, but assume these parameters are given for the time being.

The cardinality constraint $\|\tilde{\mathbf{x}} - \mathbf{x}\|_0 \leq k$ (i.e. the constraint on the group-feature size) leads to a combinatorial optimization problem in general. Such a sparsity constraint has appeared in different problems such as compressed sensing [73, 74] and LASSO regression [75]. Under certain conditions, one can show that without loss of generality, the L_0 norm can be relaxed with the (convex) L_1 norm (Supp. Section 3.C.3).

Our goal is to solve optimization (3.2) which is non-linear and non-concave in $\tilde{\mathbf{x}}$. Current approaches do not consider the cardinality constraint and optimize $\tilde{\mathbf{x}}$ by linearizing the objective function (i.e., using the gradient). To incorporate group features into the current methods, we can add the constraints of optimization (3.2) to the objective function using Lagrange multipliers. This yields the following Context-Aware First-Order (CAFO) interpretation function.

Definition 2 (The CAFO Interpretation)**.** *For a given sample $(\mathbf{x}, y)$, we define the Context-Aware*

First-Order (CAFO) importance function $\tilde{I}_{\theta^*}^{\lambda_1,\lambda_2}(\mathbf{x}, y)$ *as follows:*

$$\begin{aligned}\tilde{I}_{\theta^*}^{\lambda_1,\lambda_2}(\mathbf{x}, y) := \max_{\Delta} \quad & \nabla_{\mathbf{x}} \ell\left(f_{\theta^*}(\mathbf{x}), y\right)^t \Delta \\ & - \lambda_1 \|\Delta\|_1 - \lambda_2 \|\Delta\|_2^2\end{aligned} \tag{3.3}$$

where λ_1 *and* λ_2 *are non-negative regularization parameters. We refer to the objective of this optimization as* $\tilde{\ell}(\Delta)$*, hiding its dependency on* $(\mathbf{x}, y)$ *and* θ^* *to simplify notation.*

Large values of regularization parameters λ_1 and λ_2 in optimization (3.3) correspond to small values of parameters k and ρ in optimization (3.2). Incorporating group-features naturally leads to a sparsity regularizer through the L_1 penalty. Note, this is not a hard constraint which forces a sparse interpretation. Instead, given proper choice of the regularization coefficients, the interpretation will reflect the sparsity used by the underlying model. In Section 3.4.1, we detail our method for setting λ_1 for a given test sample (context-aware) based on the sparsity ratio of CAFO's optimal solution. Moreover, in Supp. Section 3.C.3, we show that under some general conditions, optimization (3.3) can be solved efficiently and its solution matches that of the original optimization (3.2).

To have a better approximation of the loss function, we use the second-order Taylor expansion of the loss function around point $(\mathbf{x}, y)$ as follows:

$$\begin{aligned}\ell\left(f_{\theta^*}(\tilde{\mathbf{x}}), y\right) \approx & \ell\left(f_{\theta^*}(\mathbf{x}), y\right) + \underbrace{\nabla_{\mathbf{x}} \ell\left(f_{\theta^*}(\mathbf{x}), y\right)^t \Delta}_{\text{the first-order term}} \\ & + \underbrace{\frac{1}{2} \Delta^t \mathbf{H}_{\mathbf{x}} \Delta}_{\text{the second-order term}}\end{aligned} \tag{3.4}$$

where $\Delta := \tilde{\mathbf{x}} - \mathbf{x}$ and $\mathbf{H}_{\mathbf{x}}$ is the Hessian of the loss function on the input features $\mathbf{x}$ (note y

is fixed). This second-order expansion of the loss function decreases the interpretation's model approximation error.

We show that by choosing proper values for regularization parameters, the resulting optimization using the second-order surrogate loss is strictly a convex minimization (or equivalently concave maximization) problem, allowing for efficient optimization using gradient descent (Theorem 3). Moreover, even though the Hessian matrix $\mathbf{H}_{\mathbf{x}}$ can be expensive to compute for large neural networks, gradient updates of our method only require the Hessian-vector product (i.e., $\mathbf{H}_{\mathbf{x}}\Delta$) which can be computed efficiently [77]. This yields the following Context-Aware Second-Order (CASO) interpretation function.

Definition 3 (The CASO Interpretation)**.** *For a given sample* $(\mathbf{x}, y)$*, we define the Context-Aware Second-Order (CASO) importance function* $\tilde{I}^{\lambda_1,\lambda_2}_{\theta^*}(\mathbf{x}, y)$ *as follows:*

$$\tilde{I}^{\lambda_1,\lambda_2}_{\theta^*}(\mathbf{x}, y) := \max_{\Delta} \; \nabla_{\mathbf{x}}\ell\left(f_{\theta^*}(\mathbf{x}), y\right)^t \Delta + \frac{1}{2}\Delta^t \mathbf{H}_{\mathbf{x}}\Delta \\ - \lambda_1\|\Delta\|_1 - \lambda_2\|\Delta\|_2^2 \tag{3.5}$$

We refer to the objective of this optimization as $\tilde{\ell}(\Delta)$*.* λ_1 *and* λ_2 *are defined as in* (3.3)*.*

3.3 Understanding Impact of the Hessian in Deep Learning Interpretation

Hessian is by definition useful when the loss function at the test sample has high curvature. However, given the linear nature of popular network architectures with piecewise linear activations (e.g., ReLu [78], Maxout [79]), *do these regions of high curvature even exist?* We answer this question for neural networks with piecewise linear activations by providing an exact calculation of the input Hessian. We use this derivation to understand the impact of including the Hessian

term in interpretation. More specifically, we prove that when the probability of the predicted class is ≈ 1 and the number of classes is large, the second-order interpretation is similar to the first-order one. We verify this theoretical result experimentally over images in the ImageNet 2012 dataset [80]. We also observe that when the confidence in the predicted class is low, the second-order interpretation can be significantly different from the first-order interpretation. Since second-order interpretations take into account the curvature of the model, we conjecture that they are more faithful to the underlying model in these cases.

3.3.1 A Closed-form Hessian Formula for Deep ReLU Networks

We present an abridged version of the exact Hessian calculation here while the details are provided in Supp. Section 3.A.1. Neural network models which use piecewise linear activation functions have class scores (logits) which are linear functions of the input. That is, at a particular input they are just a linear function since they are piecewise linear over the entire domain. [2]. Thus, we can write:

$$f_\theta(\mathbf{x}) = \mathbf{W}^T\mathbf{x} + \mathbf{b},$$

where $\mathbf{x}$ is the input of dimension d, $f_\theta(\mathbf{x})$ are the logits, $\mathbf{W}$ are the weights, and $\mathbf{b}$ are the biases of the linear function. Note that $\mathbf{W}$ combines weights of different layers from the input to the output of the network. Each row $\mathbf{W}_i$ of $\mathbf{W}$ is the gradient of logit $f_\theta(\mathbf{x})_i$ with respect to flattened

[2]Note that we ignore points where the function is non-differentiable as they form a measure zero set.

input $\mathbf{x}$ and can be handled in auto-grad software such as PyTorch [81]. We define:

$$\mathbf{p} = \text{softmax}(f_\theta(\mathbf{x}))$$
$$\ell(f_\theta(\mathbf{x}), y) = -\sum_{i=1}^{\text{c}} y_i \log(\mathbf{p}_i),$$

where c denotes the number of classes, $\mathbf{p}$ denotes the class probabilities, and $\ell(\mathbf{p}, \mathbf{y})$ is the cross-entropy loss function.

In this case, we have the following result:

Proposition 1. $\mathbf{H}_\mathbf{x}$ *is given by:*

$$\mathbf{H}_\mathbf{x} = \nabla^2_\mathbf{x} \ell(\mathbf{p}, \mathbf{y}) = \mathbf{W}(\mathit{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^T)\mathbf{W}^T \tag{3.6}$$

where $\text{diag}(\mathbf{p})$ is a diagonal matrix whose diagonal elements are equal to $\mathbf{p}$.

The first observation from Proposition 1 is as follows:

Theorem 2. $\mathbf{H}_\mathbf{x}$ *is a positive semidefinite matrix.*

These two results allow an extremely efficient computation of the Hessian's eigenvectors and eigenvalues using the Cholesky decomposition of $\text{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^T$. See Supp. Section 3.C.1 for full details. Note the use of decomposition is critical as storing the Hessian requires intractable amounts of memory for high dimensional inputs. The entire calculation of the Hessian's decomposition for ImageNet using a ResNet-50 [1] runs in approximately 4.2 seconds on an NVIDIA GTX 1080 Ti.

To the best of our knowledge, this is the first work which derives the exact Hessian decomposition for piecewise linear networks. [82] also proved the Hessian for piecewise linear

networks is at most rank c but did not derive the exact input Hessian.

One advantage of having a closed-form formula for the Hessian matrix (3.6) is that we can use it to properly set the regularization parameter λ_2 in CASO's formulation. To do this, we rely on the following result:

Theorem 3. *If L is the largest eigenvalue of $\mathbf{H}_\mathbf{x}$, for any value of $\lambda_2 > L/2$, the second-order interpretation objective function (3.5) is strongly concave.*

We use Theorem 3 to set the regularization parameter λ_2 for CASO. We need to set λ_2 to make the optimization convex, but not set λ_2 so large that it overpowers $\mathbf{H}_\mathbf{x}$. In particular, we set $\lambda_2 = L/2 + c_1$, where we choose $c_1 = 10$ for CASO and CAFO. We observe that if c_1 is small, the optimization can become non-convex due to numerical error in the calculation of L. However above a threshold, it does not have a significant impact on the saliency map.

3.3.2 Theoretical results on the Hessian Impact

In this subsection, we leverage the exact Hessian calculation to prove that when the probability of predicted class is ≈ 1 and number of classes is large, the Hessian of a piece-wise linear neural network is approximately of rank one and its eigenvector is approximately parallel to the gradient. Since a constant scaling does not affect the visualization, this causes the two interpretations to be similar to each other.

Theorem 4. *If the probability of the predicted class=1-(c-1)ϵ , where $\epsilon \approx 0$, then as $c \to \infty$ such that $c\epsilon \approx 0$, Hessian is of rank one and its eigenvector is parallel to the gradient.*

Let Δ^*_{CASO} be the optimal solution to the CASO objective 3.5 and Δ^*_{CAFO} be the optimal solution for the CAFO objective 3.3. We assume λ_1=0 for both the objectives.

Theorem 5. *If the probability of the predicted class=1-(c-1)ϵ , where $\epsilon \approx 0$, then as $c \to \infty$ such that $c\epsilon \approx 0$, the CASO solution (3.5) with $\lambda_1 = 0$ is almost parallel to the CAFO solution (3.3) with $\lambda_1 = 0$.*

We emphasize that our theoretical results are valid in the "asymptotic regime". To do an analysis of the approximation in the finite length regime, we run a simulation of the relative error between the true Hessian and the rank-one approximation of the Hessian as the number of classes increases and probability of predicted class tends to 1. Details are given in Supp. Section 3.C.4.1.

3.3.3 Empirical results on the Hessian impact

In this subsection, we present empirical results on the impact of the second-order loss approximation in deep learning interpretation. In our experiments here, to isolate the impact of the Hessian term, we assume $\lambda_1 = 0$ in both CASO and CAFO optimizations.

A consequence of Theorem 3 is that the gradient descent method with Nesterov momentum converges to the global optimizer of the second-order interpretation objective objective with a convergence rate of $\mathcal{O}(1/t^2)$ (Supp. Section 3.B). To optimize Δ, the gradient is given by:

$$\nabla_{\Delta} \tilde{\ell}(\Delta) = \nabla_{\mathbf{x}} \ell\left(f_{\theta^*}(\mathbf{x}), y\right) + \mathbf{H}_{\mathbf{x}} \Delta - 2\lambda_2 \Delta. \tag{3.7}$$

The gradient term $\nabla_{\mathbf{x}} \ell\left(f_{\theta^*}(\mathbf{x}), y\right)$ and the regularization term $-2\lambda_2\Delta$ are straightforward to implement using standard backpropagation.

To compute the Hessian-vector product term $\mathbf{H}_{\mathbf{x}}\Delta$, we rely on the result of Pearlmutter 1994 [77]: a Hessian-vector product can be computed in the same time as the gradient $\nabla_{\mathbf{x}} \ell\left(f_{\theta^*}(\mathbf{x}), y\right)$. This is handled easily in modern auto-grad software. Moreover, for ReLU

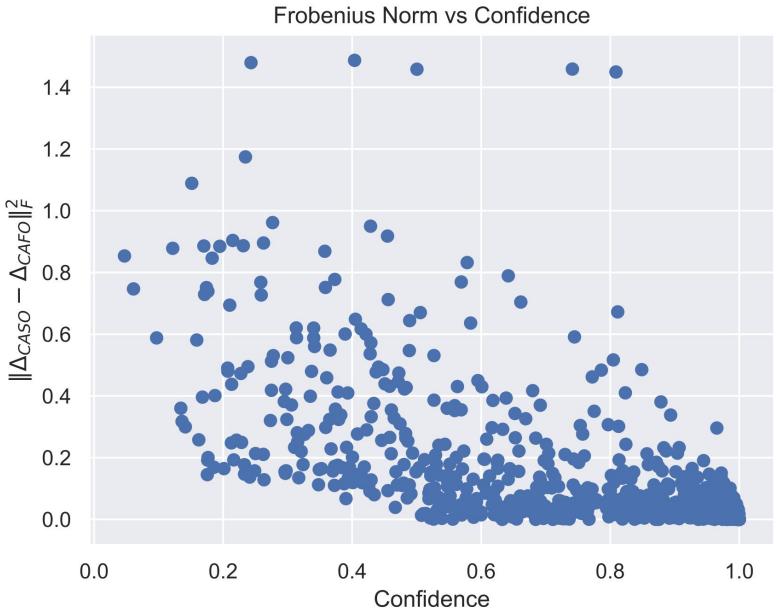


Figure 3.1: Scatter plot showing the Frobenius norm difference between CASO and CAFO (after normalizing both vectors to have the same L_2 norm). Consistent with the result of Theorem 4, when the classification confidence is low, the CASO result differs significantly from CAFO. When confidence is high, CASO and CAFO are approximately the same. To isolate the impact of the Hessian term, we assume $\lambda_1 = 0$ in both CASO and CAFO optimizations.

networks, our closed-form formula for the Hessian term (Theorem 1) can be used in computation of the Hessian-vector product as well. In our experiments, we use the closed-form formula for $\lambda_1 = 0$ and proximal gradient descent for $\lambda_1 > 0$.

We compare second-order interpretations (CASO with $\lambda_1 = 0$) and first-order variant (CAFO with $\lambda_1 = 0$) empirically. Note that when $\lambda_1 = 0$, $\Delta_{CAFO} = \frac{1}{\lambda_2}\mathbf{g_x}$ where $\mathbf{g_x}$ is the gradient and Δ_{CAFO} is the interpretation obtained using the CAFO objective.

We compute second-order and first-order interpretations for 1000 random samples on the ImageNet ILSVRC-2012 [80] validation set using a Resnet-50 [1] model. Our loss function $\ell(\cdot, \cdot)$ is the cross-entropy loss. After calculating Δ for all methods, the values must be normalized for visualization in a saliency map. We apply a normalization technique from existing work which

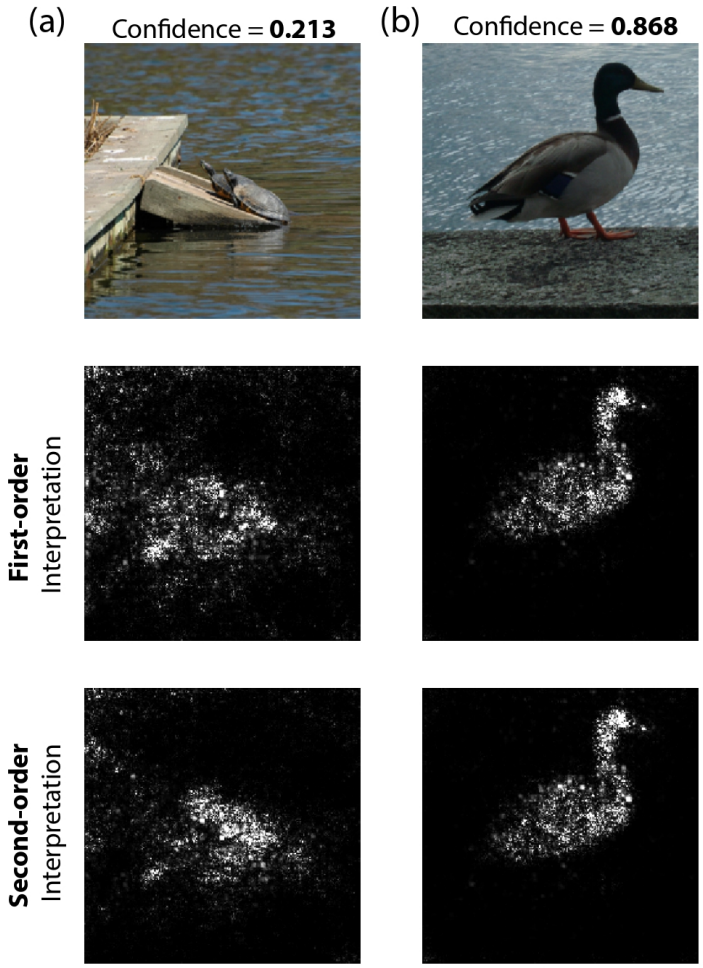


Figure 3.2: Panel (a) shows an example where the classification confidence is low. In this case, the CASO and CAFO interpretations differ significantly. Panel (b) demonstrates an example where the classification confidence is high. In this case, CASO and CAFO lead to similar interpretations as suggested by our theory.

we describe in Supp. Section 3.C.2.

We plot the Frobenius norm of the difference between CASO and CAFO in Figure 3.1. Before taking the difference, we normalize the Δ solutions produced by CASO and CAFO to have the same L_2 norm because a constant scaling of elements of Δ does not change the visualization.

The empirical results are consistent with our theoretical results: the second-order interpretation results are similar to the first-order ones when the classification confidence is high. How-

ever, when the confidence is small, including the Hessian term can be useful in deep learning interpretation.

To observe the difference between CAFO and CASO interpretations qualitatively, we compare them for an image when the confidence is high and for one where it is low in Figure 3.2. When the classification confidence is high, CAFO $\approx$ CASO and when this is low, CASO $\neq$ CAFO. Additional examples have been given in Supp. Section 3.C.4.

We do additional experiments to evaluate the impact of Hessian on a neural network that is not piecewise linear. We use a trained SE-Resnet-50 [83] neural network (since it uses sigmoid non-linearity) with the same random samples. We observe a similar trend as in the case of a relu network suggesting that Theorem 5 is more general and applicable to any neural network. Details are given in the Supp. Section 3.C.4.2.

3.4 Understanding Impact of Group-features

In this section, we study the impact of the group feature in deep learning interpretation. The group feature has been included as the sparsity constraint in optimization (3.2).

To obtain an unconstrained concave optimization for the CASO interpretation, we relaxed the sparsity (cardinality) constraint $\|\Delta\|_0 \leq k$ (often called an L_0 norm constraint) to a convex L_1 norm constraint. Such a $L_0 - L_1$ relaxation is a core component for popular learning methods such as compressed sensing [73, 74] or LASSO regression [75]. Using results from this literature, we show this relaxation is tight under certain conditions on the Hessian matrix $\mathbf{H_x}$ (see Supp. Section 3.C.3). In other words, the optimal Δ of optimization (3.5) is sparse with the proper choice of regularization parameters.

Note that the regularization term $-\lambda_1\|\Delta\|_1$ is a concave function for $\lambda_1 > 0$. Similarly due to Theorem 3, the CASO interpretation objective (3.5) is strongly concave.

One method for optimizing this objective is to apply the gradient descent method used in the second-order interpretation but with the addition of an L_1 regularization penalty. However, we found that this procedure leads to poor convergence properties in practice. This is partially due to the non-smoothness of the L_1 regularization term. To resolve this issue, we instead use the *proximal* gradient descent to compute a solution for CAFO and CASO when $\lambda_1 > 0$. Using the Nesterov momentum method and backtracking with proximal gradient descent gives a convergence rate of $\mathcal{O}(1/t^2)$ where t is the number of gradient updates (Supp. Section 3.B).

Below we explain how we use the proximal gradient descent to optimize our objective. First, we write the objective function as the sum of a smooth and non-smooth function:

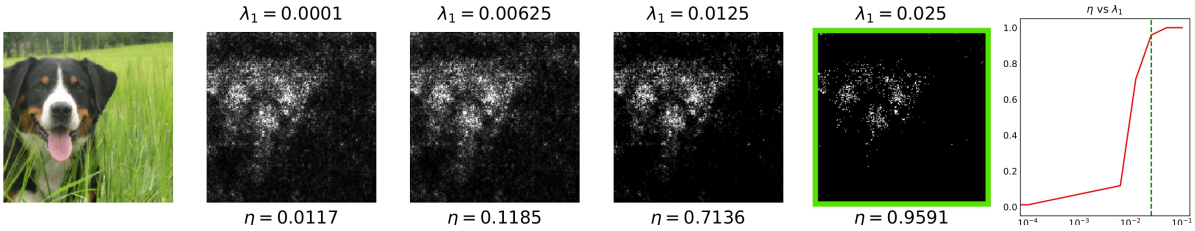
$$\tilde{\ell}(\Delta) = \underbrace{\nabla_{\mathbf{x}}\ell\left(f_{\theta^*}(\mathbf{x}), y\right)^t \Delta + \frac{1}{2}\Delta^t \mathbf{H}_{\mathbf{x}}\Delta - \lambda_2\|\Delta\|_2^2}_{\text{Smooth Part}} \quad - \quad \underbrace{\lambda_1\|\Delta\|_1}_{\text{Non-Smooth Part}}$$

Let $g(\Delta)$ be the smooth, $h(\Delta)$ be the non-smooth part:

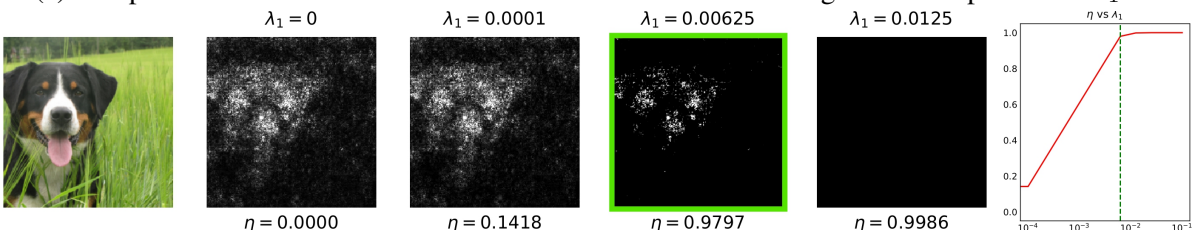
$$\begin{aligned}
\tilde{\ell}(\Delta) &= g(\Delta) + h(\Delta) \\
g(\Delta) &= \nabla_{\mathbf{x}}\ell\left(f_{\theta^*}(\mathbf{x}), y\right)^t \Delta + \frac{1}{2}\Delta^t \mathbf{H}_{\mathbf{x}}\Delta - \lambda_2\|\Delta\|_2^2 \\
h(\Delta) &= -\lambda_1\|\Delta\|_1
\end{aligned}$$

The gradient of the smooth objective is given by:

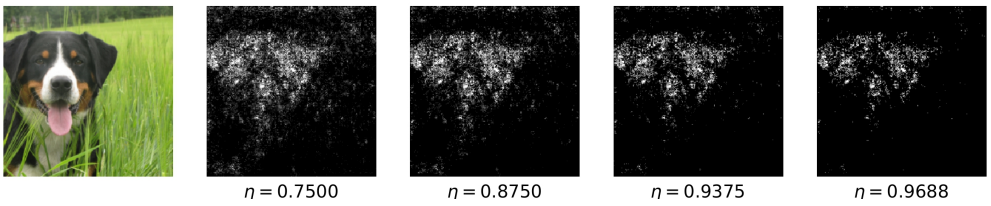
$$\nabla_\Delta g(\Delta) = \nabla_{\mathbf{x}}\ell\left(f_{\theta^*}(\mathbf{x}), y\right) + \mathbf{H}_{\mathbf{x}}\Delta - 2\lambda_2\Delta$$



(a) Interpretation solutions for CAFO with different values of the regularization parameter λ_1



(b) Interpretation solutions for CASO with different values of the regularization parameter λ_1



(c) Interpretation solutions for Vanilla Gradient with different thresholds to induce the given sparsity

Figure 3.3: Large λ_1 values lead to high sparsity ratios (η). Our unsupervised method based on the sparsity ratio selects the interpretations marked with a green box in these examples. Interpretations selected in Panel (a) and (b) are less noisy compared to (c).

The proximal operator is given by:

$$\text{prox}_\alpha(x) = \arg\min_z \frac{1}{\alpha}\|x - z\|_2^2 + \lambda_1\|z\|_1 = \begin{cases} x + \lambda_1\alpha & x \leq -\lambda_1\alpha \\ 0 & -\lambda_1\alpha < x \leq \lambda_1\alpha \\ x - \lambda_1\alpha & \lambda_1\alpha < x \end{cases}$$

This formula can be understood intuitively as follows. If the magnitude of some elements of Δ is below a certain threshold ($\lambda_1\alpha$), proximal mapping sets those values to zero. This leads to values that are exactly zero in the saliency map. This can be viewed as removing noise by a certain thresholding procedure.

To optimize Δ, we use FISTA [84] with backtracking and the Nesterov momentum optimizer with a learning rate of 0.1 for 10 iterations and decay factor of 0.5. We initialize $\Delta = 0$.

FISTA takes a step with learning rate α to reduce the smooth objective loss $g(\Delta)$, then applies a proximal mapping to the resulting Δ. Backtracking reduces the learning rate when the update results in higher loss.

3.4.1 Empirical results on the impact of group features

In this section, our goal is to understand the impact of the group features in deep learning interpretation. In our experiments, we focus on the image classification problem because visual interpretations are intuitive and allow for comparison with prior work. We use a Resnet-50 [1] model on the ImageNet ILSVRC-2012 dataset.

To gain intuition for the effect of λ_1, we show a sweep over values in Figure 3.3. We observe that when λ_1 is too high or too low, the interpretation breaks down as the importance values are constant across the image (all high or all zero). Different approaches to set the regularization parameter λ_1 have been explored in different problems. For example, in LASSO, one common approach is to use Least Angle Regression [85].

We propose an unsupervised method based on the sparsity ratio of the interpretation solution to set a proper value for λ_1. We define η, the sparsity ratio, as the number of zero pixels divided by the total number of pixels. We tune λ_1 in an unsupervised fashion since we do not know the ground truth interpretation. We start with $\lambda_1 = 10^{-5}$ and increase λ_1 by a factor of 10 until Δ reaches all zeros. For interpretations with sparsity in a certain range (e.g. $1 > \eta \geq 0.75$ in our examples), we choose the one with the highest loss. If we do not find any interpretation

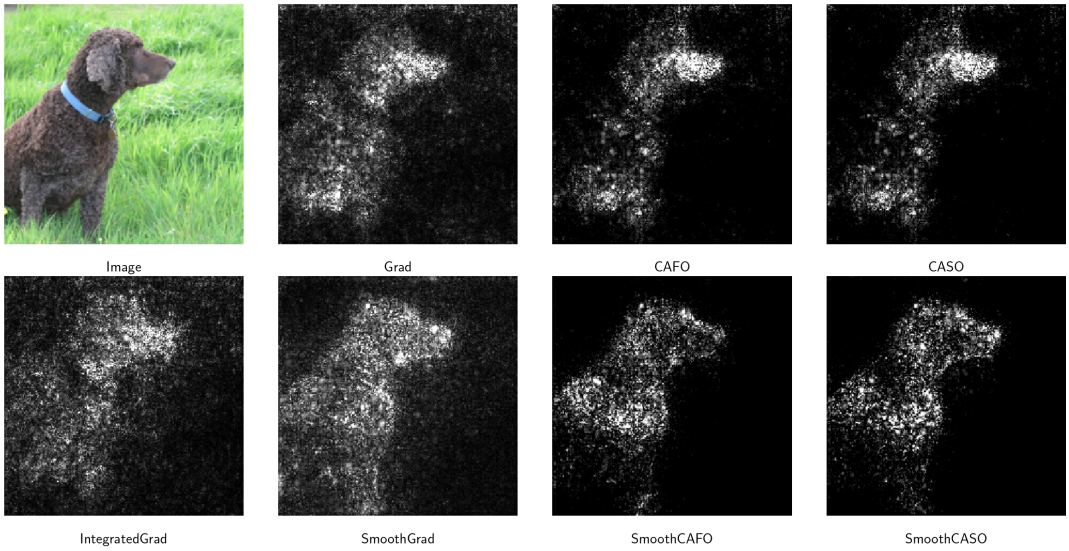


Figure 3.4: A qualitative comparison of existing interpretation methods. More examples are shown in in Supp. Section 3.C.5. Grad stands for Vanilla Gradient and IntegratedGrad stands for Integrated Gradient. For our methods (CAFO, CASO, SmoothCAFO, SmoothCASO) we can see that the saliency map is more visually coherent with the object of interest compared to the existing methods.

that satisfies the sparsity condition, we do more iterations by reducing the first λ_1 that resulted in Δ becoming zero by a factor of 2. In practice, we batch different values of λ_1 to find a reasonable parameter setting efficiently.

This method selects the interpretation marked with a green box in Figures 3.3a and 3.3b. In Figure 3.3c, we show the vanilla gradient interpretation with different values of thresholds that induce the specified sparsity value. We observe that the interpretations obtained using group features (Figures 3.3a and 3.3b) are less noisy compared to Figure 3.3c.

3.5 Qualitative Comparision of Deep learning Interpretation Methods

In this section, we briefly review prior approaches for the deep learning interpretation and compare their performance qualitatively. The proposed Hessian and group feature terms can be po-

tentially included in these approaches as well.

Vanilla Gradient [65] propose to compute the gradient of the class score with respect to the input.

SmoothGrad [68] argue that the input gradient may fluctuate sharply in the region local to the test point. To address this, they average the gradient based importance values generated from many noisy inputs.

Integrated Gradients [66] define a *baseline*, which represents an input absent of information (e.g., a completely zero image). Feature importance is determined by accumulating gradient information along the path from the baseline to the original input: $(\mathbf{x} - \mathbf{x}') \times \int_{\alpha=0}^{1} \nabla_{\mathbf{x}} \ell\left(f_{\theta^*}(\mathbf{x}' + \alpha(\mathbf{x} - \mathbf{x}')), y\right)\, d\alpha$. The integral is approximated by a finite sum.

We use the normalization method from SmoothGrad [68] for visualizing the saliency map. Details of the method are given in Supp. Section 3.C.2.

We can extend behind SmoothGrad to define smooth versions of CASO and CAFO. This yields the following interpretation objective.

Definition 4 (The Smooth CASO Interpretation)**.** *For a given sample* $(\mathbf{x}, y)$*, we define the smooth context-aware second-order (the Smooth CASO) importance function* $\tilde{I}_{\theta^*}^{\lambda_1,\lambda_2}(\mathbf{x}, y)$ *as follows:*

$$\begin{aligned}\tilde{I}_{\theta^*}^{\lambda_1,\lambda_2}(\mathbf{x}, y) := \max_{\Delta}\ \frac{1}{n}\sum_{1}^{n}(\nabla_{\mathbf{z}} \ell\left(f_{\theta^*}(\mathbf{z}), y\right)^t \Delta \\ + \frac{1}{2}\Delta^t \mathbf{H}_{\mathbf{z}} \Delta) - \lambda_1 \|\Delta\|_1 - \lambda_2 \|\Delta\|_2^2\end{aligned} \tag{3.8}$$

where $\mathbf{z} = \mathbf{x} + N(0, \mathbf{m}\sigma^2 I)$ *and* λ_1 *and* λ_2 *are defined similarly as before.*

In the smoothed versions, we average over $n = 50$ samples with $\mathbf{m}\sigma = 0.15$. Smooth CAFO is defined similarly without the Hessian term.

Since quantitative evaluations of saliency maps remains an open problem, we focus on two aspects of qualitative evaluations of different interpretation methods. First, we inspect visual coherence i.e, only the object of interest should be highlighted and not the background. Second, we test for discriminativity i.e, in an image with two objects the predicted object should be highlighted. Figure 3.4 shows a comparison between CAFO, CASO and other existing methods. Including group-features in deep learning interpretation leads to a sparse saliency map, eliminating the spurious noise and creating a visually coherent saliency map. More examples have been presented in Supp. Section 3.C.5

3.6 Conclusion and Future Work

In this chapter, we studied two aspects of the deep learning interpretation problem. First, by characterizing a closed-form formula for the Hessian matrix of a deep ReLU network, we showed that, if the confidence in the predicted class is high and the number of classes is large, first-order and second-order methods produce similar results. In the process, we also proved that the Hessian matrix is of rank one and its eigenvector is parallel to the gradient. These results can be insightful in other related problems such as adversarial examples The extent of the Hessian impact for low confidence predictions and/or the case when the number of classes is small are among interesting directions for the future work. Second, we incorporated high-order feature dependencies in deep learning interpretation using a sparsity regularization term. This extension improves the deep learning interpretation significantly.

Although significant progresses have been made in tackling the deep learning interpretation problem, there remain some open problems as well. For example, since saliency maps are high-

dimensional vectors, they can be sensitive to noise and adversarial perturbations. Moreover, due to the lack of a quantitative evaluation method for the interpretation problem, the evaluation of interpretation methods are often qualitative and can be subjective. Resolving these issues are among interesting directions for the future work.

Supplementary material

3.A Proofs

3.A.1 Proof of Proposition 1

In this section, we derive the closed-form formula for the Hessian of the loss function of a deep ReLU network. Since a ReLU network is piecewise linear, it is locally linear around an input $\mathbf{x}$. Thus the logits can be represented as:

$$f_\theta(\mathbf{x}) = \mathbf{W}^T\mathbf{x} + \mathbf{b},$$

where $\mathbf{x}$ is the input of dimension d, $f_\theta(\mathbf{x})$ are the logits, $\mathbf{W}$ are the weights, and $\mathbf{b}$ are the biases of the linear function. In this proof, we use $\hat{\mathbf{y}}$ to denote the logits, $\mathbf{p}$ to denote the class probabilities, $\mathbf{y}$ to denote the label vector and c to denote the number of classes. Each column $\mathbf{W}_i$ of $\mathbf{W}$ is the gradient of logit $\hat{\mathbf{y}}_i$ with respect to flattened input $\mathbf{x}$ and can be easily handled in auto-grad software such as PyTorch [81].

Thus

$$\frac{\partial \hat{\mathbf{y}}_i}{\partial \mathbf{x}} = \mathbf{W}_i \tag{3.9}$$

$$\mathbf{p} = \text{softmax}(\hat{\mathbf{y}})$$

$$\ell(\mathbf{p}, \mathbf{y}) = -\sum_{i=1}^{\text{c}} \mathbf{y}_i \log(\mathbf{p}_i).$$

$$\nabla_{\hat{\mathbf{y}}} \ell(\mathbf{p}, \mathbf{y}) = \mathbf{p} - \mathbf{y}$$

$$\implies \frac{\partial \ell(\mathbf{p}, \mathbf{y})}{\partial \hat{\mathbf{y}}_i} = \mathbf{p}_i - \mathbf{y}_i \tag{3.10}$$

$$\nabla_{\mathbf{x}} \ell(\mathbf{p}, \mathbf{y}) = \sum_{i=1}^{\text{c}} \frac{\partial \hat{\mathbf{y}}_i}{\partial \mathbf{x}} \times \frac{\partial \ell(\mathbf{p}, \mathbf{y})}{\partial \hat{\mathbf{y}}_i}$$

Using (3.9) and (3.10)

$$\nabla_{\mathbf{x}} \ell(\mathbf{p}, \mathbf{y}) = \sum_{i=1}^{\text{c}} \mathbf{W}_i (\mathbf{p}_i - \mathbf{y}_i)$$

$$\implies \nabla_{\mathbf{x}} \ell(\mathbf{p}, \mathbf{y}) = \mathbf{W}(\mathbf{p} - \mathbf{y})$$

Therefore, we have:

$$\mathbf{H}_{\mathbf{x}} = \nabla_{\mathbf{x}} (\nabla_{\mathbf{x}} \ell(\mathbf{p}, \mathbf{y})) = \nabla_{\mathbf{x}} (\sum_{i=1}^{\text{c}} \mathbf{W}_i (\mathbf{p}_i - \mathbf{y}_i))$$

$$\mathbf{H}_{\mathbf{x}} = \sum_{i=1}^{\text{c}} \mathbf{W}_i (\nabla_{\mathbf{x}} (\mathbf{p}_i - \mathbf{y}_i))^T$$

$$\mathbf{H}_{\mathbf{x}} = \sum_{i=1}^{\text{c}} \mathbf{W}_i (\nabla_{\mathbf{x}} \mathbf{p}_i)^T \tag{3.11}$$

Deriving $\nabla_{\mathbf{x}}\mathbf{p}_i$:

$$\nabla_{\mathbf{x}}\mathbf{p}_i = \sum_{j=1}^{\mathrm{c}} \frac{\partial \hat{\mathbf{y}}_j}{\partial \mathbf{x}} \times \frac{\partial \mathbf{p}_i}{\partial \hat{\mathbf{y}}_j}$$

$$\implies \nabla_{\mathbf{x}}\mathbf{p}_i = \sum_{j=1}^{\mathrm{c}} \left(\mathbf{W}_j \times \frac{\partial \mathbf{p}_i}{\partial \hat{\mathbf{y}}_j} \right) \quad \text{(Using (3.9))} \tag{3.12}$$

$$\frac{\partial \mathbf{p}_i}{\partial \hat{\mathbf{y}}_j} = \begin{cases} \mathbf{p}_i - \mathbf{p}_i^2 & i = j \\ -\mathbf{p}_i \mathbf{p}_j & i \neq j \end{cases}$$

$$\implies \nabla_{\hat{\mathbf{y}}}\mathbf{p} = \mathrm{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^T \tag{3.13}$$

$$\mathbf{H}_{\mathbf{x}} = \sum_{i=1}^{\mathrm{c}} \mathbf{W}_i \left(\sum_{j=1}^{\mathrm{c}} \mathbf{W}_j \times \frac{\partial \mathbf{p}_i}{\partial \hat{\mathbf{y}}_j} \right)^T \quad \text{(Substituting (3.12) in (3.11))}$$

$$\mathbf{H}_{\mathbf{x}} = \sum_{i=1}^{\mathrm{c}} \sum_{j=1}^{\mathrm{c}} \mathbf{W}_i \frac{\partial \mathbf{p}_i}{\partial \hat{\mathbf{y}}_j} \mathbf{W}_j^T$$

$$\implies \mathbf{H}_{\mathbf{x}} = \mathbf{W}(\mathrm{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^T)\mathbf{W}^T \quad \text{(Using (3.13))}$$

Thus we have,

$$\nabla_{\mathbf{x}}\ell(\mathbf{p}, \mathbf{y}) = \mathbf{g}_{\mathbf{x}} = \mathbf{W}(\mathbf{p} - \mathbf{y}) \tag{3.14}$$

$$\mathbf{H}_{\mathbf{x}} = \mathbf{W}\mathbf{A}\mathbf{W}^T \tag{3.15}$$

where

$$\mathbf{A} := \text{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^T. \tag{3.16}$$

This completes the proof.

3.A.2 Proof of Theorem 2

To simplify notation, define $\mathbf{A}$ as in (3.16). For any arbitrary row of the matrix $\mathbf{A}_i$, we have

$$\begin{aligned}
&\sum_{\text{j}\neq\text{i}} |\mathbf{A}_{ij}| = (\sum_{\text{j}\neq\text{i}} | - \mathbf{p}_i\mathbf{p}_j|) \\
\implies &\sum_{\text{j}\neq\text{i}} |\mathbf{A}_{ij}| = \mathbf{p}_i \sum_{\text{j}\neq\text{i}} \mathbf{p}_j \\
\implies &\sum_{\text{j}\neq\text{i}} |\mathbf{A}_{ij}| = \mathbf{p}_i(1 - \mathbf{p}_i) \\
&|\mathbf{A}_{ii}| = \mathbf{p}_i(1 - \mathbf{p}_i)
\end{aligned}$$

Because $|\mathbf{A}_{ii}| >= \sum_{\text{j}\neq\text{i}} |\mathbf{A}_{ij}|$, by the Gershgorin Circle theorem, we have that all eigenvalues of $\mathbf{A}$ are positive and $\mathbf{A}$ is a positive semidefinite matrix. Since $\mathbf{A}$ is psd, we can write $\mathbf{A} = \mathbf{L}\mathbf{L}^T$. Using (3.15):

$$\mathbf{H}_\mathbf{x} = \mathbf{W}\mathbf{A}\mathbf{W}^T = \mathbf{W}\mathbf{L}\mathbf{L}^T\mathbf{W}^T = \mathbf{W}\mathbf{L}(\mathbf{W}\mathbf{L})^T$$

Hence $\mathbf{H}_\mathbf{x}$ is a positive semidefinite matrix as well.

3.A.3 Proof of Theorem 3

The second-order interpretation objective function is given by,

$$\tilde{\ell}(\Delta) = \nabla_{\mathbf{x}} \ell\left(f_{\theta^*}(\mathbf{x}), y\right)^t \Delta + \frac{1}{2}\Delta^t \mathbf{H}_{\mathbf{x}} \Delta - \lambda_2 \|\Delta\|_2^2$$
$$\tilde{\ell}(\Delta) = \nabla_{\mathbf{x}} \ell\left(f_{\theta^*}(\mathbf{x}), y\right)^t \Delta + \frac{1}{2}\Delta^t (\mathbf{H}_{\mathbf{x}} - 2\lambda_2 \mathbf{I})\Delta$$

where $\Delta := \tilde{\mathbf{x}} - \mathbf{x}$ (y is fixed). Therefore if $\lambda_2 > L/2$, $\mathbf{H}_{\mathbf{x}} - 2\lambda_2 \mathbf{I}$ is negative definite and $\tilde{\ell}(\Delta)$ is strongly concave.

3.A.4 Proof of Theorem 4

Let the class probabilities be denoted by $\mathbf{p}$, the number of classes by c and the label vector by $\mathbf{y}$. We again use $\mathbf{g}_{\mathbf{x}}$ and $\mathbf{H}_{\mathbf{x}}$ as defined in (3.14) and (3.15) respectively. Without loss of generality, assume that the first class is the one with maximum probability.

Hence,

$$\mathbf{y} = [1, 0, 0, ..., 0]^T \tag{3.17}$$

We assume all other classes have small probability,

$$\mathbf{p}_i = \epsilon \approx 0 \quad \forall \text{ i} \in [2, \text{c}]$$

Since $\sum_{i=1}^{c} \mathbf{p}_i = 1$,

$$\implies \mathbf{p}_1 = 1 - (c-1)\epsilon,$$

$$\implies \mathbf{p} = [1 - (c-1)\epsilon,\ \epsilon, \ldots,\ \epsilon]^T \tag{3.18}$$

We define:

$$\mathbf{A} = \text{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^T$$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1c} \\ a_{21} & a_{22} & \ldots & a_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ a_{c1} & a_{c2} & \ldots & a_{cc} \end{bmatrix}$$

$$a_{11} = 1 - (c-1)\epsilon - (1 - (c-1)\epsilon)^2$$

$$a_{1i} = a_{i1} = -(1 - (c-1)\epsilon)\epsilon \quad \forall\, i \in [2, c]$$

$$a_{ii} = \epsilon - \epsilon^2 \quad \forall\, i \in [2, c]$$

$$a_{ij} = -\epsilon^2 \quad \forall\, i, j \in [2, c],\ i \neq j$$

Ignoring ϵ^2 terms:

$$a_{11} = (c-1)\epsilon$$

$$a_{1i} = a_{i1} = -\epsilon \quad \forall i \in [2, c]$$

$$a_{ii} = \epsilon \quad \forall i \in [2, c]$$

$$a_{ij} = 0 \quad \forall i, j \in [2, c], i \neq j$$

Let λ be an eigenvalue of $\mathbf{A}$ and $\mathbf{v}$ be an eigenvector of $\mathbf{A}$, then $\mathbf{Av} = \lambda\mathbf{v}$.

Let $v_1, v_2, \ldots, v_n$ be the individual components of the eigenvector. The equation $\mathbf{Av} = \lambda\mathbf{v}$ can be rewritten in terms of its individual components as follows:

$$(c-1)\epsilon v_1 - \epsilon \sum_{i=2}^{c} v_i = \lambda v_1 \tag{3.19}$$

$$-\epsilon v_1 + \epsilon v_i = \lambda v_i \; \forall i \in [2, c]$$

$$\implies v_i = \frac{\epsilon}{\epsilon - \lambda} v_1 \; \forall i \in [2, c], \text{ for } \lambda \neq \epsilon \tag{3.20}$$

$$\implies \text{or } v_1 = 0, \text{ for } \lambda = \epsilon \tag{3.21}$$

We first consider the case $\lambda \neq \epsilon$ (3.20). Substituting v_i in (3.19):

$$\begin{aligned}(c-1)\epsilon v_1 - \epsilon \sum_{i=2}^{c} v_i &= (c-1)\epsilon v_1 - \frac{\epsilon^2}{\epsilon - \lambda} \sum_{i=2}^{c} v_1 \\ &= (c-1)\epsilon v_1 - \frac{\epsilon^2}{\epsilon - \lambda}(c-1)v_1 \\ &= (c-1)\epsilon v_1 - (c-1)\epsilon v_1 \frac{\epsilon}{\epsilon - \lambda} \\ &= \lambda v_1\end{aligned}$$

$$\begin{aligned}&(c-1)\epsilon v_1 \left[1 - \frac{\epsilon}{\epsilon - \lambda}\right] = \lambda v_1 \\ &(c-1)\epsilon v_1 \left[-\frac{\lambda}{\epsilon - \lambda}\right] = \lambda v_1 \\ &(c-1)\epsilon v_1(-\lambda) = \lambda v_1(\epsilon - \lambda) \\ &\implies \lambda v_1(c\epsilon - \lambda) = 0 \\ &\implies \lambda = 0 \text{ or } v_1 = 0 \text{ or } \lambda = c\epsilon \\ &v_1 = 0 \implies v_i = \frac{\epsilon}{\epsilon - \lambda} v_1 = 0 \quad \forall\, i \in [2, c] \\ &\implies \mathbf{v} = 0\end{aligned}$$

Since $\mathbf{v}$ is an eigenvector, it cannot be zero,

$$\implies \lambda = 0 \text{ or } \lambda = c\epsilon.$$

Let $\mathbf{u}_1$ be the corresponding eigenvector for $\lambda = c\epsilon$.

By substituting $\lambda = c\epsilon$ in (3.20):

$$\mathbf{u}_1^T \propto [1-c, 1, ..., 1]$$

Dividing by the normalization constant,

$$\mathbf{u}_1^T = \frac{1}{\sqrt{c(c-1)}}[1-c, 1, ..., 1] \tag{3.22}$$

Now we consider the case $\lambda = \epsilon$ (3.21). Substituting $v_1 = 0,\ \lambda = \epsilon$ in (3.19):

The space of eigenvectors for $\lambda = \epsilon$ is a $c-2$ dimensional subspace with $v_1 = 0,\ \sum_{i=2}^{c} v_i = 0$.

Let $\mathbf{u}_i$ be the eigenvectors with $\lambda = \epsilon \quad \forall\, i \in [2, c-1]$

Let $\mathbf{u}_c$ be the eigenvector with $\lambda = 0$.

Writing $\mathbf{A}$ in terms of its eigenvalues and eigenvectors,

$$\mathbf{A} = c\epsilon \mathbf{u}_1 \mathbf{u}_1^T + \epsilon \sum_{i=2}^{c-1} \mathbf{u}_i \mathbf{u}_i^T$$

Let

$$\mathbf{A}_1 = c\epsilon \mathbf{u}_1 \mathbf{u}_1^T, \quad \mathbf{A}_2 = \epsilon \sum_{i=2}^{c-1} \mathbf{u}_i \mathbf{u}_i^T$$

$$\|\mathbf{A}_1\|_F = c\epsilon, \quad \|\mathbf{A}_2\|_F = \epsilon\sqrt{c-2}$$

Hence as $c \to \infty$,

$$\mathbf{A} = \mathbf{A}_1 + \mathbf{A}_2 \approx \mathbf{A}_1$$

Using (3.15),

$$\mathbf{H_x} = \mathbf{WAW}^T \approx \mathbf{WA}_1\mathbf{W}^T$$

Substituting $\mathbf{A}_1 = c\epsilon\mathbf{u}_1\mathbf{u}_1^T$,

$$\mathbf{H_x} \approx c\epsilon\mathbf{W}\mathbf{u}_1\mathbf{u}_1^T\mathbf{W}^T \tag{3.23}$$

Using (3.14),

$$\mathbf{g_x} = \nabla_{\mathbf{x}}\ell(\mathbf{p}, \mathbf{y}) = \mathbf{W}(\mathbf{p} - \mathbf{y})$$

Let $\mathbf{W}_i$ denote the ith row of $\mathbf{W}$,

Using (3.17) and (3.18),

$$\mathbf{g_x} = \mathbf{W}_1(1-c)\epsilon + \sum_{i=2}^{c}\mathbf{W}_i\epsilon$$

$$\mathbf{g_x} = \epsilon(\mathbf{W}_1(1-c) + \sum_{i=2}^{c}\mathbf{W}_i)$$

Using (3.22),

$$\mathbf{g_x} = \epsilon\sqrt{c(c-1)}\mathbf{W}\mathbf{u}_1$$

$$\implies \mathbf{W}\mathbf{u}_1 = \frac{\mathbf{g_x}}{\epsilon\sqrt{c(c-1)}} \tag{3.24}$$

Using (3.23),

$$\mathbf{H_x} \approx c\epsilon \mathbf{W}\mathbf{u}_1\mathbf{u}_1^T\mathbf{W}^T = c\epsilon \mathbf{W}\mathbf{u}_1(\mathbf{W}\mathbf{u}_1)^T$$

Using (3.24),

$$\mathbf{H_x} \approx c\epsilon \frac{\mathbf{g_x}}{\epsilon\sqrt{c(c-1)}} \frac{\mathbf{g_x}^T}{\epsilon\sqrt{c(c-1)}}$$

$$\mathbf{H_x} \approx c\epsilon \frac{\mathbf{g_x}\mathbf{g_x}^T}{\epsilon^2 c(c-1)} = \frac{\mathbf{g_x}\mathbf{g_x}^T}{\epsilon(c-1)}$$

$$\implies \mathbf{H_x} \approx \frac{\mathbf{g_x}\mathbf{g_x}^T}{\epsilon(c-1)} \tag{3.25}$$

Thus, the Hessian is approximately rank one and the gradient is parallel to the Hessian's only eigenvector.

3.A.5 Proof of Theorem 5

We use $\mathbf{g_x} = \nabla_{\mathbf{x}}\ell(\mathbf{p}, \mathbf{y}) = \mathbf{W}(\mathbf{p} - \mathbf{y})$ (3.14).

When $\lambda_1 = 0$ in the CASO and CAFO objectives, the CASO objective becomes:

$$\max_{\Delta} (\mathbf{g_x}^t\Delta + \frac{1}{2}\Delta^t\mathbf{H_x}\Delta - \lambda_2\|\Delta\|_2^2)$$

Taking the derivative with respect to Δ and solving:

$$\Delta^*_{CASO} = (2\lambda_2\mathbf{I} - \mathbf{H_x})^{-1}\mathbf{g_x}$$

Similarly, for the CAFO objective we get:

$$\Delta^*_{CAFO} = \frac{1}{2\lambda_2}\mathbf{g_x}$$

Using (3.25),

$$\mathbf{H_x} \approx \frac{\mathbf{g_x}\mathbf{g_x}^T}{\epsilon(c-1)} = \frac{\|\mathbf{g_x}\|^2}{\epsilon(c-1)}\frac{\mathbf{g_x}\mathbf{g_x}^T}{\|\mathbf{g_x}\|^2}$$

Define:

$$\mu = \frac{\|\mathbf{g_x}\|^2}{\epsilon(c-1)}$$

Thus μ is the eigenvalue of $\mathbf{H_x}$ for the eigenvector:

$$\frac{\mathbf{g_x}}{\|\mathbf{g_x}\|}$$

Consider the matrix $\mathbf{B} = (2\lambda_2\mathbf{I} - \mathbf{H_x})$:

Let $\mathbf{z}_1, \ldots, \mathbf{z}_d$ be the eigenvectors of $\mathbf{B}$ where:

$$\mathbf{z}_1 = \frac{\mathbf{g_x}}{\|\mathbf{g_x}\|}$$

Eigenvalue for $\mathbf{z}_1 = 2\lambda_2 - \mu$

Eigenvalue for $\mathbf{z}_i = 2\lambda_2 \quad \forall i \in [2, d]$

$$\mathbf{B} = (2\lambda_2 - \mu)\mathbf{z}_1\mathbf{z}_1^T + 2\lambda_2 \sum_{i=2}^{i=d} \mathbf{z}_i\mathbf{z}_i^T$$

$$\mathbf{B}^{-1} = \frac{1}{(2\lambda_2 - \mu)}\mathbf{z}_1\mathbf{z}_1^T + \frac{1}{2\lambda_2}\sum_{i=2}^{i=d}\mathbf{z}_i\mathbf{z}_i^T$$

$$\mathbf{B}^{-1} = \frac{1}{(2\lambda_2 - \mu)}\frac{\mathbf{g_x}\mathbf{g_x}^T}{\|\mathbf{g_x}\|^2} + \frac{1}{2\lambda_2}\sum_{i=2}^{i=d}\mathbf{z}_i\mathbf{z}_i^T$$

$$\Delta^*_{CASO} = \mathbf{B}^{-1}\mathbf{g_x}$$

$$\Delta^*_{CASO} = \left[\frac{1}{(2\lambda_2 - \mu)}\frac{\mathbf{g_x}\mathbf{g_x}^T}{\|\mathbf{g_x}\|^2} + \frac{1}{2\lambda_2}\sum_{i=2}^{i=d}\mathbf{z}_i\mathbf{z}_i^T\right]\mathbf{g_x}$$

Since each $\mathbf{z}_i$ is orthogonal to $\mathbf{g_x}$

$$\implies \Delta^*_{CASO} = \frac{\mathbf{g_x}}{(2\lambda_2 - \mu)} = \frac{2\lambda_2\Delta^*_{CAFO}}{(2\lambda_2 - \mu)}$$

Hence $\Delta^*_{CASO} \parallel \Delta^*_{CAFO}$ and since scaling does not affect the visualization, the two interpretations are equivalent.

3.B Convergence of Gradient Descent to Solve CASO

A consequence of Theorem 3 is that gradient descent converges to the global optimizer of the second-order interpretation objective objective with a convergence rate of $\mathcal{O}(1/t^2)$. More precisely, we have:

Corollary 1. *Let $\tilde{\ell}(\Delta)$ be the objective function of the second-order interpretation objective defined in Section 3.2 (Definition 3). Let $\Delta^{(t)}$ be the value of Δ in the tth step with a learning rate*

$\alpha \leq \lambda_2 - L/2$. *We have*

$$\tilde{\ell}(\Delta^{(t)}) - \tilde{\ell}(\Delta^*) \leq \frac{2\|\Delta^{(0)} - \Delta^*\|_2^2}{\alpha(t+1)^2}.$$

3.C Additional analysis

3.C.1 Efficient Computation of the Hessian Matrix Using the Cholesky decomposition

By Theorem 2, the Cholesky decomposition of $\mathbf{A}$ (defined in (3.16)) exists. Let $\mathbf{L}$ be the Cholesky decomposition of $\mathbf{A}$. Thus, we have

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

$$\mathbf{H_x} = \mathbf{W}\mathbf{L}\mathbf{L}^T\mathbf{W}^T$$

Let $\mathbf{B} := \mathbf{W}\mathbf{L}$. Thus, $\mathbf{H_x}$ can be re-written as $\mathbf{H_x} = \mathbf{B}\mathbf{B}^T$.

Let the SVD of $\mathbf{B}$ be as the following:

$$\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

Thus, we can write:

$$\mathbf{H_x} = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^T$$

Define $\mathbf{C} = \mathbf{B}^T\mathbf{B} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}$. Note that $\mathbf{\Sigma}^2$, the eigenvalues of $\mathbf{C}$ and $\mathbf{H_x}$ are the same. For a dataset such as ImageNet, the input has dimension d = 224×224×3 and c = 1000. Decomposing C (size 1000×1000) into its eigenvalues $\mathbf{\Sigma}$ and eigenvectors $\mathbf{V}$ is computationally efficient. Thus, from $\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, we can compute the eigenvectors $\mathbf{U}$ of $\mathbf{H_x}$.

3.C.2 Saliency Visualization Methods

Normalizing Feature Importance Values After assigning importance values to each input feature, the values must be normalized for visualization in a saliency map. For fair comparison across all methods, we use the non-diverging normalization method for SmoothGrad [68]. This normalization method first takes the absolute value of the importance scores and then sums across the three color channels of the image. Next, the largest importance values are capped to the value of 99th percentile. Finally, the importance values are divided and clipped to enforce the range $[0, 1]$. Code for the method is available.[3]

Domain-Specific Post-Processing Gradient $\odot$ Input [86] multiplies the importance values by the raw feature values. In image tasks where the baseline is zero, Integrated Gradients [66] does the same. This heuristic can visually sharpen the saliency map and has some theoretical justification: it is equivalent to the original Layerwise Relevance Propagation Technique [87] modulo a scaling factor [86, 88]. Additionally, if the model is linear, $y = W\mathbf{x}$, mutliplying the gradient by the input is equivalent to a feature's true contribution to the final class score.

However, multiplying by the input can introduce visual artifacts not present in the importance values [68]. We argue against multiplying by the input: it artificially enhances the visual-

[3] `https://github.com/PAIR-code/saliency/blob/master/saliency/visualization.py`

ization and only yields benefits in the image domain. [89] argue similar and show cases when the input term can dominate the interpretation. Moreover, multiplication by the input removes the input invariance of the interpretation regardless of the invariances of the underlying model [90]. We observed numerous failures in existing interpretation methods when input multiplication is removed.

3.C.3 Tightness of the $L_0 - L_1$ Relaxation

We assume the condition of Theorem 3 holds, thus, the CASO optimization is a concave maximization (equivalently a convex minimization) problem. Note the CASO optimization with the cardinality constraint can be re-written as follows:

$$\min_{\Delta} \quad \|\mathbf{y} - \mathbf{A}\Delta\|^2, \qquad \|\Delta\|_0 \leq k, \tag{3.26}$$

$$\text{where} \quad \mathbf{y} := \left(\lambda_2 \mathbf{I} - \frac{1}{2}\mathbf{H}_{\mathbf{x}}\right)^{1/2}, \quad \mathbf{y} := \frac{1}{2}\mathbf{A}^{-1}\nabla_{\mathbf{x}}\ell\left(f_{\theta^*}(\mathbf{x}), y\right). \tag{3.27}$$

$(.)^{1/2}$ indicates the square root of a positive definite matrix. Equation (3.27) highlights the condition for tuning the parameter λ_2: it needs to be sufficiently large to allow inversion of $\mathbf{A}$ but sufficiently small to not overpower the Hessian term. Note, we are now minimizing Δ for consistency with the compressed sensing literature. To explain the conditions under which the $L_0 - L_1$ relaxation is tight, we define the following notation. For a given subset $S \subset \{1, 2, ..., d\}$

and constant $\alpha \geq 1$, we define the following cone:

$$\mathcal{C}(S;\alpha) := \{\Delta \in \mathbb{R}^d : \|\Delta_{S^c}\|_1 \leq \|\Delta_S\|_1\}, \tag{3.28}$$

where S^c is the complement of S. We say that the matrix $\mathbf{A}$ satisfies the restricted eigenvalue (RE) [91, 92] condition over S with parameters $(\alpha, \gamma) \in [1, \infty) \times (0, \infty)$ if

$$\frac{1}{d}\|\mathbf{A}\Delta\|_2^2 \geq \gamma^2 \|\Delta\|_2^2 \quad \forall \Delta \in \mathcal{C}(S;\alpha). \tag{3.29}$$

If this condition is satisfied for all subsets of S where $|S| = k$, we say that $\mathbf{A}$ satisfies the RE condition of order k with parameters (α, γ). If $\mathbf{A}$ satisfies the RE condition with $\alpha \geq 3$ and $\gamma > 0$, then the $L_0 - L_1$ relaxation of optimization (3.26) is tight [92]. In other words, if Δ^* is the solution of optimization (3.26), it is also the solution of optimization

$$\begin{aligned} \min_{\Delta} \quad & \|\mathbf{y} - \mathbf{A}\Delta\|^2, \\ & \|\Delta\|_1 \leq \|\Delta^*\|_1. \end{aligned} \tag{3.30}$$

The Lagrange relaxation of this optimization leads to the CASO interpretation objective. We note that the RE condition is less severe than other optimality conditions such as the restricted isometry property [93]. Although it is difficult to verify that the RE condition holds for the Hessian matrix of a deep neural network, empirical experiments are consistent with the theoretical results: the resulting Δ of the CASO interpretation objective is sparse for proper choices of the regularization parameters as shown below.

3.C.4 Empirical analysis of the Hessian impact

3.C.4.1 Empirical analysis of the Hessian approximation

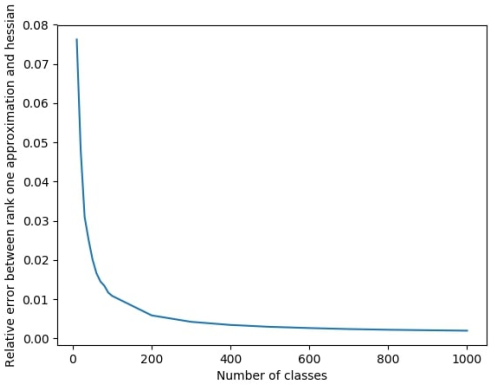
We emphasize that Theorems 4 and 5 are valid only in the asymptotic regime. A similar analysis in the finite regime is more challenging as it requires the use of perturbation analysis of matrix eigenvalues and eigenvectors. However, we can do a simulation to assess the rate of convergence of the Hessian matrix to such a rank one matrix with as the number of classes increases (in Figure 3.5a) and the probability of the predicted class tends to 1 (in Figure 3.5b).

For the simulation, we create a linear model $\hat{\mathbf{y}} = \mathbf{W}^T\mathbf{x} + \mathbf{b}$ where $\mathbf{W}$ and $\mathbf{b}$ are initialized to random values. We emphasize that since Theorem 4 does not assume a trained network, our analysis is valid even with randomly initialized values of $\mathbf{W}$ and $\mathbf{b}$. Let the class probabilities be denoted by $\mathbf{p}$, the number of classes by c and the label vector by $\mathbf{y}$. We again use $\mathbf{g_x}$ and $\mathbf{H_x}$ as defined in (3.14) and (3.15) respectively. Without loss of generality, assume that the first class is the one with maximum probability. Thus we create a probability vector $\mathbf{p} = [1 - (c - 1)\epsilon,\ \epsilon,\ \ldots,\ \epsilon]$ and a prediction vector $\mathbf{y} = [1.,\ 0.,\ \ldots,\ 0.]$. Using Proof 3.A.4, we have that the Hessian of this model can be approximated using:

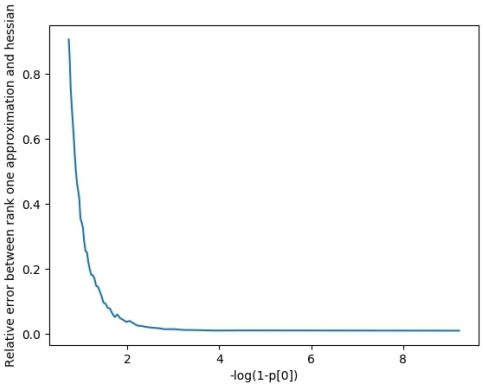
$$\mathbf{H_x} \approx \frac{\mathbf{g_x}\mathbf{g_x}^T}{\epsilon(c-1)}$$

$$\mathbf{g_x} = \mathbf{W}(\mathbf{p} - \mathbf{y})$$

For Figure 3.5a, we fix the probability of predicted class to be 0.9999 (we call it $\mathbf{p}[0]$) and vary c from 10 to 1000. Thus, note that ϵ varies as c varies and is given by $\epsilon = \frac{1-\mathbf{p}[0]}{c-1}$. For figure 3.5b,



(a) Number of classes



(b) p[0], predicted class probability (log scale)

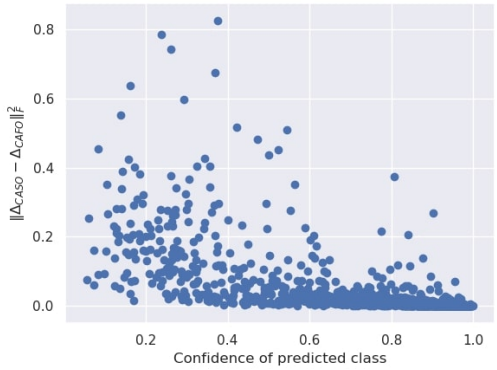
Figure 3.5: Plot of relative error between rank one approximation of the hessian and true hessian (Y-axis) vs (a) Number of classes and (b) log of predicted class probability

we fix number of classes to be 100 and vary ϵ from $5e-3$ to $1e-6$ (an interval length of $1e-6$).

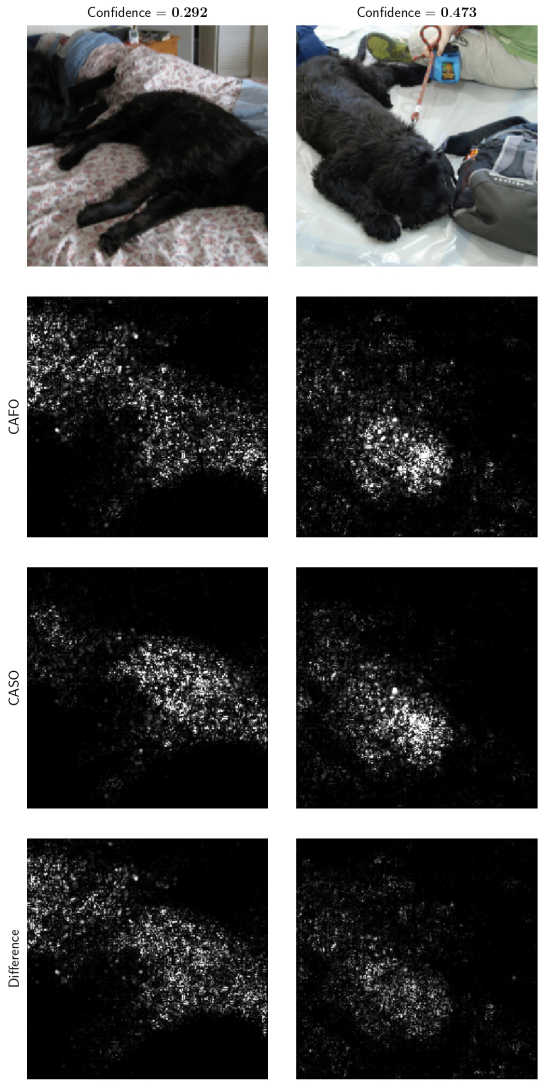
3.C.4.2 Experiments with general non-linearities

We use a trained SE-Resnet-50 [83] neural network (since it uses sigmoid non-linearity) with the same 1000 random samples as in Section 3.3.3.

We plot the Frobenius norm of the difference between CASO and CAFO in Figure 3.6a. We again normalize the solutions produced by CASO and CAFO to have the same L_2 norm before taking the difference.The empirical results are consistent with Theorem 5, even though this neural network contains sigmoid non-linearity while Theorem 5 is valid only for piecewise linear neural networks. To observe the difference between CAFO and CASO interpretations, we compare them for two images with low classification confidence in Figure 3.6b. We can qualitatively see that interpretations produced by CASO and CAFO look different.



(a)



(b)

Figure 3.6: Left: Scatter plot showing the Frobenius norm difference between CASO and CAFO (after normalizing both vectors to have the same L_2 norm), Right: CASO and CAFO interpretations for low confidence examples.

3.C.5 Comparison with existing methods

Figures 3.7–3.12 provide further examples of our interpretation method with existing techniques.

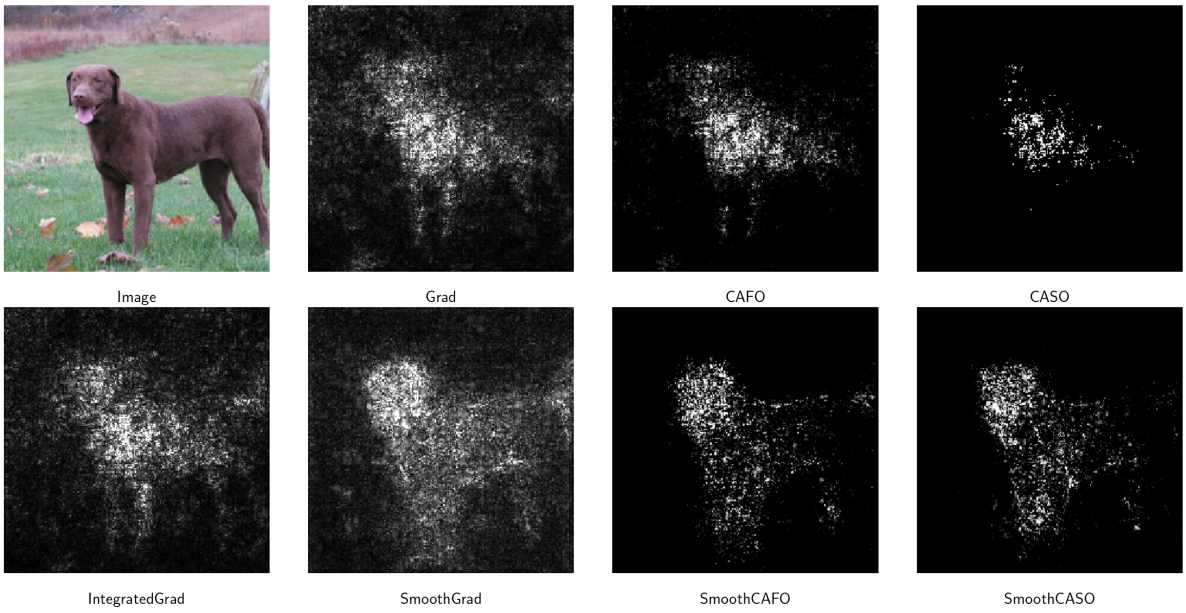


Figure 3.7: Example 1

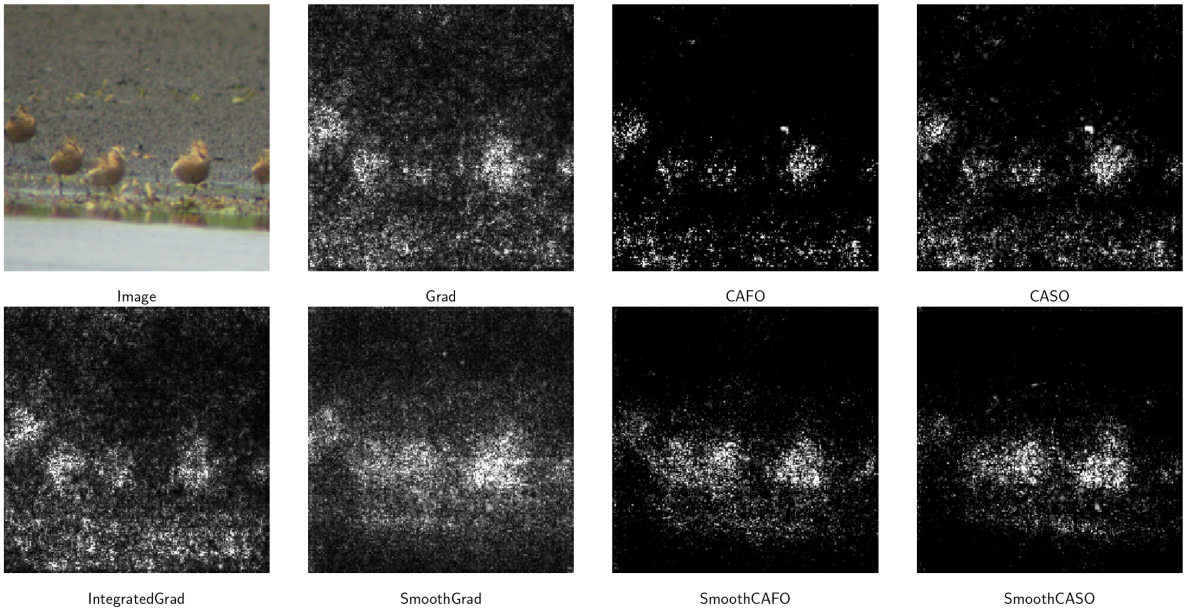


Figure 3.8: Example 2

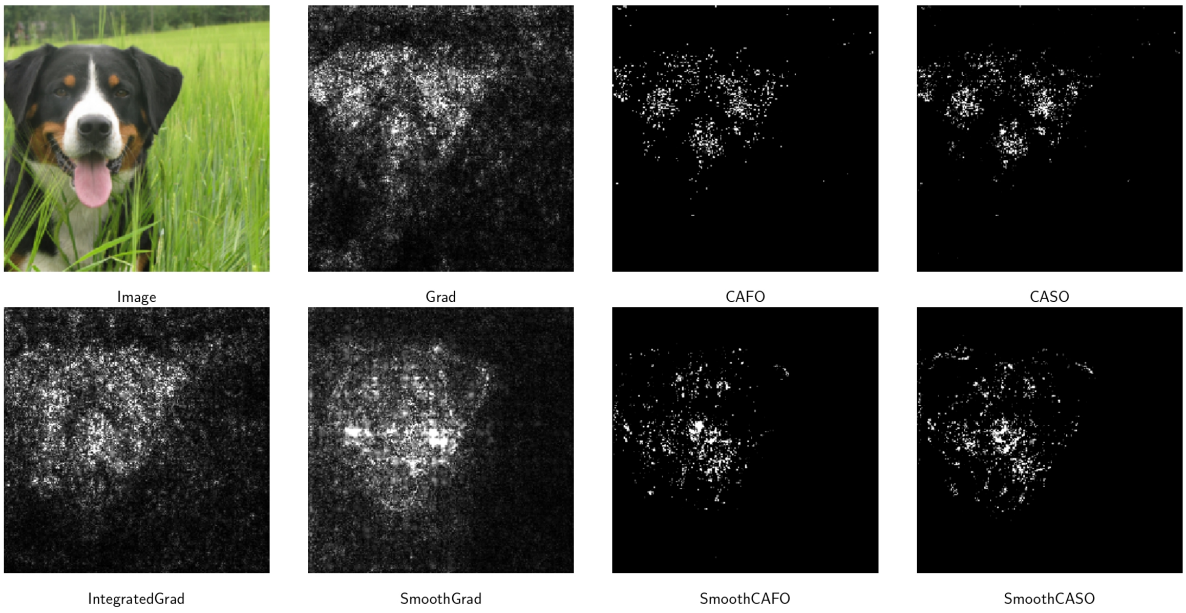


Figure 3.9: Example 3

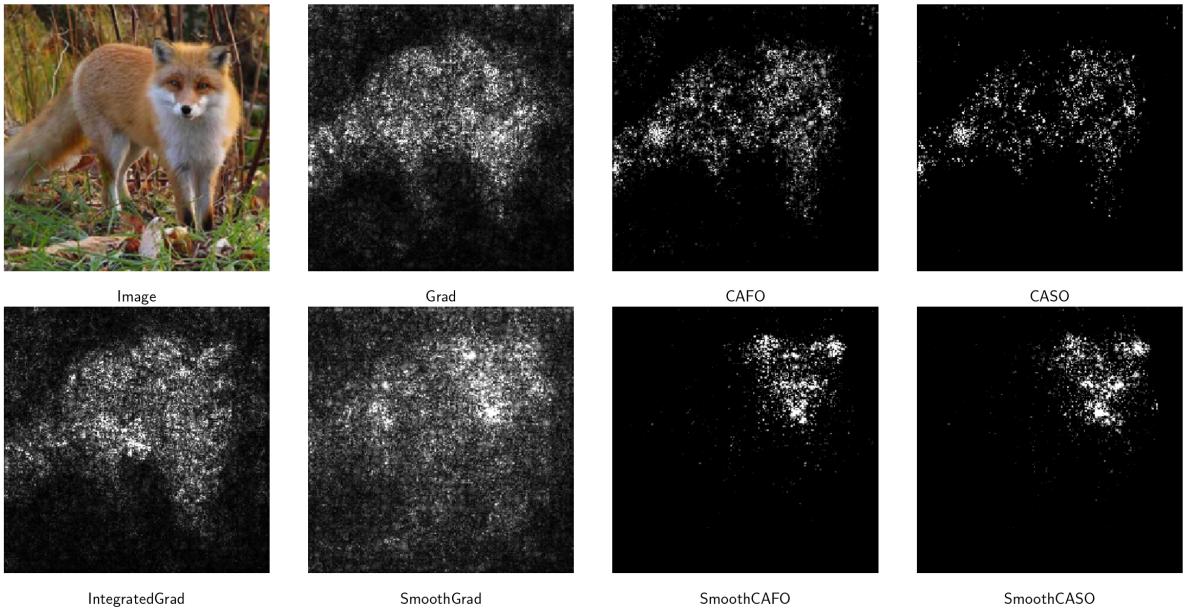


Figure 3.10: Example 4

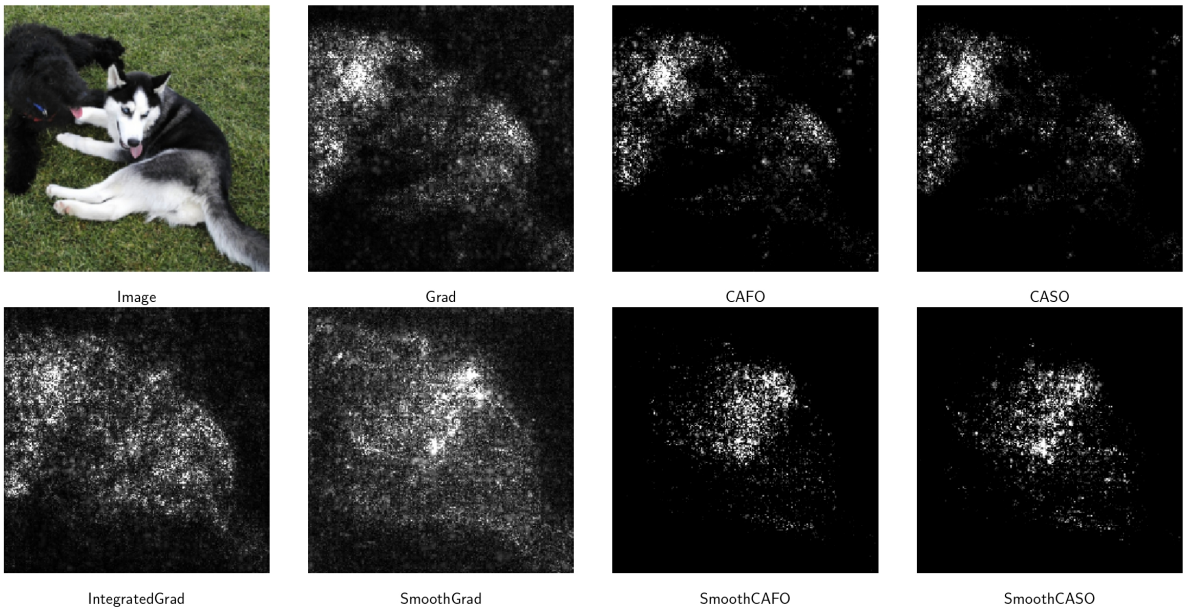


Figure 3.11: Example 5

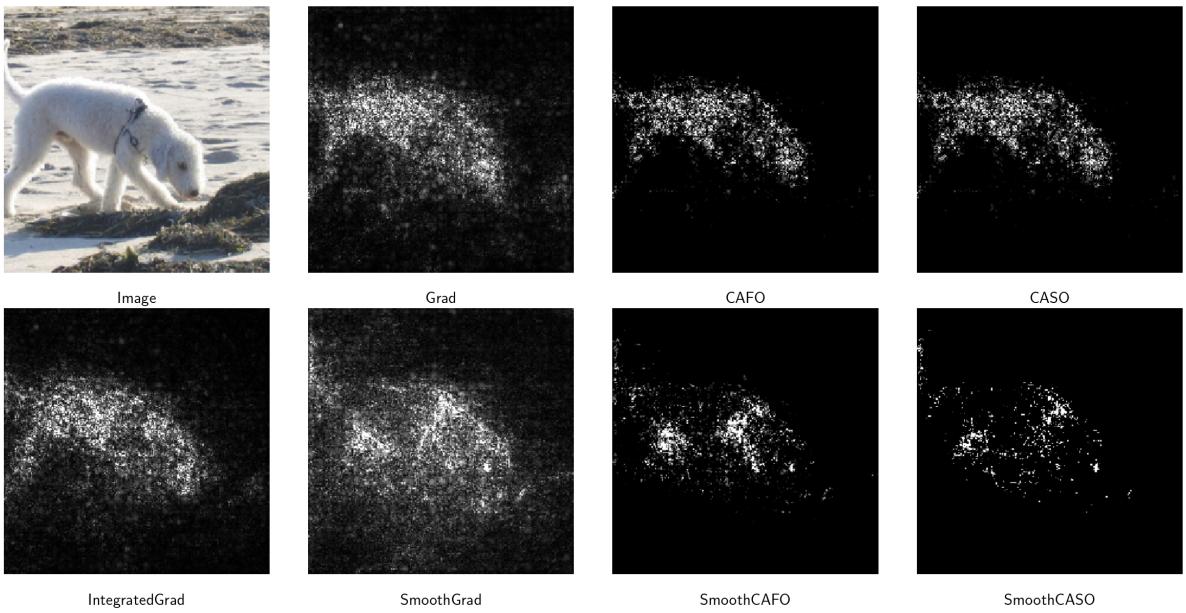


Figure 3.12: Example 6

Chapter 4: Low Curvature Activations Reduce Overfitting in Adversarial Training

4.1 Introduction

Deep Neural Networks can be readily fooled by adversarial examples, which are computed by imposing small perturbations on clean inputs [15]. Adversarial attacks have been well studied in the machine learning community in recent years [20, 24, 30, 42, 94, 95]. There have been several defenses proposed against adversarial attacks in the literature [96, 97, 98]. In our work we focus on adversarial training [20, 24, 43], one of the most effective empirical defenses.

Adversarial training involves training the network on adversarially perturbed data instead of clean data to produce a classifier with better robustness on the test set. However, it has been shown that networks produced through vanilla adversarial training do not robustly generalize well [99, 100, 101]. The gap between robust train and test accuracy for adversarially trained neural networks i.e. the *robust generalization gap* can be far greater than the generalization gap achieved during standard empirical risk minimization. In this chapter, we show that the robust generalization gap is significantly impacted by the curvature of the activation function, and activations with low curvature can act as efficient regularizers for adversarial training, effectively mitigating this phenomenon.

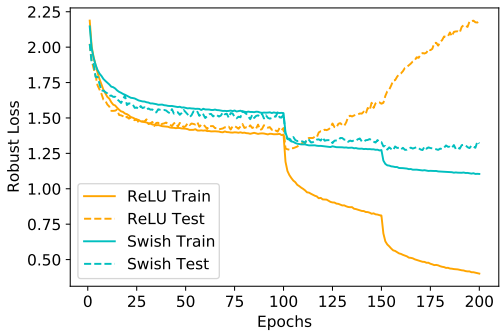


Figure 4.1: Learning curves for a robustly trained ResNet-18 model on CIFAR-10. Using an activation function with low curvature such as SiLU prevents robust overfitting, achieving and maintaining low test robust loss, even compared to the best early-stop checkpoint of a network with ReLU activation function. The learning rate is decreased by a factor of 10 at the 100th and 150th epoch.

Rice et al.[100] showed for adversarially trained ReLU networks, the best robust test accuracy is not achieved by allowing models to train until convergence. Adversarial training has the characteristic that, after a certain point, further training will continue to decrease the robust training loss, while the robust test loss starts increasing. This phenomenon is referred to as *robust overfitting* and ultimately leads to poor robust accuracy on the test set. Rice et al.also showed that while traditional approaches against overfitting such as l_1, l_2 regularization can mitigate robust overfitting, no approach works better than simple early stopping. Since standard accuracy continues to improve even after the network overfits to adversarial examples, early stopping leads to trade-off between selecting a model with high robust accuracy versus a model with high standard accuracy [102].

In this chapter, we systematically study the impact of activation functions on generalization. We first theoretically analyze the relation between maximum curvature of the activation function and adversarial robustness. A key observation of our chapter is that for smooth activation functions the maximum value of the second derivative of the function, i.e. the maximum

curvature has a significant impact on robust generalization. Specifically by using activations with low curvature the robust generalization gap can be reduced, whereas with high curvature the robust generalization gap increases. For instance, in Figure 1 for an adversarially trained CIFAR-10 model, test error on adversarial examples for the ReLU activation function decreases after the first learning rate drop, and keeps increasing afterwards. However, for SiLU [103] a smooth activation function with low curvature, robust test loss keeps decreasing. We also show that the choice of activation has a similar effect on the standard generalization gap. In other words, activations that show a large robust generalization gap also have a large standard generalization gap, and vice versa. Our work therefore provides novel insights to the robust overfitting phenomenon. The main objective of our work is to understand the relation between curvature of the activation function and adversarial training, and highlight findings which can be useful for training adversarially robust models.

Xie et al.[104] showed that replacing ReLU, a widely used activation function, by "smooth" [1] activation functions such as Softplus or SiLU with a weak adversary (single step PGD), improves adversarial robustness on Imagenet [56] for "free". They posit smooth activations allow adversarial training to find harder adversarial examples and compute better gradient updates to weight parameters. Further works have however demonstrated that while smooth activation functions can positively affect clean and robust accuracy, the trend is not as clear as the one observed by Xie et al.. Thus, ReLU networks remains a prominent choice for robust classification [105, 106].

In contrast to Xie et al.[104], we consider a strong adversary for training and show that

[1] We use the same definition of smoothness as Xie et al., that the function is C^1 smooth, that is, that the first derivative is continuous everywhere.

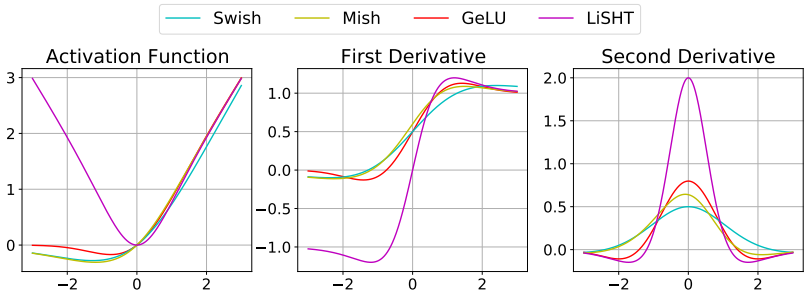


Figure 4.2: Activation functions and their first and second derivatives.

smoothness of activations is not required to obtain a regularization effect on adversarial training. In our experiments, we show that the same regularization can be achieved using non-smooth activations with low "approximate"" curvature. For non-smooth activations however, curvature is not well-defined. We consider LeakyReLU which is a non-smooth activation function and use the difference of activation slopes in positive and negative domains as the approximate maximum curvature of the activation function. Even for such a non-smooth activation function, we observe that if the approximate curvature is low, the robust overfitting phenomenon does not occur. Also in contrast to Xie et al.[104] we empirically show that smooth activations can perform worse than ReLU, if the smooth activation has high curvature.

Finally, we study the phenomenon of double descent generalization curves seen in standard training [107] and robust training [108]. Double descent describes the following phenomenon. Increasing model complexity causes test accuracy to first increase and then decrease. Then upon reaching a critical point known as interpolation threshold, test accuracy starts increasing again. We show that double descent curves reported by [100] for robust overfitting using ReLU do not hold for activation functions with low curvature such as SiLU.

4.2 Related Works

Goodfellow et al.[24] provided one of the first approaches for adversarial training based on generating adversarial examples through the fast sign gradient method (FGSM). Building on this, a stronger adversary known as the basic iterative method [43] was proposed in subsequent work, using multiple smaller steps for generating adversarial examples. Madry et al.[20] extended this adversary with multiple random restarts to train models on adversarial data, referred to as projected gradient descent (PGD) adversarial training. Further works have focused on improving the performance of the adversarial training procedure with methods such as feature denoising [109], hypersphere embedding [110], balancing standard and robust error [25] and using friendly adversarial data [111]. A separate line of works has focused on speeding up adversarial training due to its increased time complexity, by reducing attack iterations and computational complexity for calculating gradients [112, 113, 114]. Another tangent line of work focuses on adversarial training for universal attacks [115, 116].

Besides adversarial training, several other defenses have been proposed such as defensive distillation [96], preprocessing techniques [97, 98, 117] and randomized transformations [118, 119, 120] or detection of adversarial examples [121, 122]. However these methods were later broken by stronger adversaries [123, 124, 125]. These defense methods were shown to rely on obfuscated gradients (gradient masking), which provided a false sense of security. Due to the bitter history of gradient masking as a defense, Xie et al.[104] proposed use of smooth activations with a single step PGD attack, reaching state of the art robust performance on ImageNet [56]. Xie et al.hypothesize that using smooth activations provides networks with better gradient updates and allows adversaries to find harder examples.

Since many defenses proposed in the literature have been broken, another separate line of work has focused on certified defenses, which can guarantee robustness against adversarial attacks. These methods use techniques such as mixed-integer programming methods [126, 127, 128, 129] and satisfiability modulo theories [130, 131, 132]. Some certification methods bound the global Lipschitz constant of the network, which are usually loose for large neural networks with multiple layers [133, 134]. Another line of work has focused on providing loose certificates using other techniques such as randomized smoothing [135, 136, 137, 138], abstract representations [139, 140, 141], interval bound propagation, second-order information [142], [143] and duality and linear programs [144, 145, 146].

Lack of overfitting in overparameterized deep learning models is an intriguing phenomenon for deep learning [147]. These models can be trained to effectively zero training error, without having impact on test time performance. Hence, it is now standard practice in deep learning to train longer and use large overparameterized models, since test accuracy generally improves past an interpolation point also known as double descent generalization [107, 108]. Schmidt et al.[99] however have shown that sample complexity required for adversarially robust generalization is significantly higher than sample complexity for standard generalization. In a recent work, Rice et al.[100] have shown the overfitting phenomenon to be dominant in adversarial training and show that training longer decrease robustness on test data. Rice et al.also show that double descent generalization curves seem to hold with increase in model size but not by training longer. A recent work shows that robust overfitting may be mitigated [148] using a combination of previously proposed techniques such as knowledge-distillation [149] and stochastic weight averaging [150]. Another recent work, proposes the use of adversarial weight perturbations [151] to mitigate robust overfitting, which may also increase the training time. AVMixup [152] also discussed the idea of

robust overfitting and proposed a combination of AVMixup, Label smoothing and Feature Scatter to alleviate robust overfitting on CIFAR-10. In contrast to these works, we discover a novel way to mitigate this phenomenon without using complex regularization techniques that may lead to additional hyper-parameters and increased training time; we only modify the activation function of the network.

4.3 Background

4.3.1 Adversarial Training

To train networks that are robust to adversarial examples, the following robust optimization framework is used:

$$\min_{w} \mathbb{E}_{(x,y)\sim\mathbb{D}} \left[\max_{d(x,\hat{x})\leq\epsilon} l(f_w(\hat{x}), y) \right]$$

$$\mathbb{D}$$

where x is a training sample with ground truth label y sampled from the underlying data distribution $\mathbb{D}$, $l(.,.)$ is the loss function, f_w is the model parameterized by w parameters, $d(.,.)$ is a distance function and ϵ is the maximum distance allowed. Typically, the distance function is chosen to be an l_p-norm ball such as the l_2 and l_∞-norm balls though other non l_p threat models have been considered in [30, 95]. Adversarial training thus consists of two optimization problems, the inner maximization problem to construct adversarial samples, and the outer minimization problem to update weight parameters w. To solve the inner maximization problem, different types of attacks have been used in the literature, such as projected gradient descent (PGD) [20] or fast gradient sign method (FSGM) [114]. For example, an l_∞ PGD adversary starts with a random

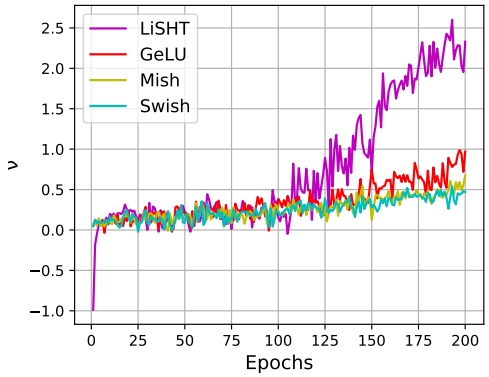


Figure 4.3: Maximum eigenvalues for a batch of test examples for Resnet-18 models with different smooth activations. Eigenvalues are larger for activations with high curvature.

initial perturbation drawn from a uniform distribution $\mathcal{U}$, and iteratively adjusts the perturbation with α step-size towards the l_∞ gradient direction, followed by projection back onto the l_∞ norm ball with maximum radius ϵ:

$$\hat{x}_0 = x + \mathcal{U}(-\epsilon, \epsilon)$$

$$\bar{x}_t = \hat{x}_t + \alpha \cdot \text{sign}\big(\nabla_{\hat{x}_t} l(f(\hat{x}_t), y)\big)$$

$$\hat{x}_{t+1} = \max(\min(\bar{x}_t, x + \epsilon), x - \epsilon)$$

4.3.2 Robust Overfitting

A surprising characteristic of overparametrized models is their good generalization behavior observed in practice. [107, 153]. Although overparameterized models have enough model complexity to memorize the dataset even on random labels [147], they can be trained to zero error on the training set with no detrimental effects on generalization. For the standard (non-adversarial) empirical risk minimization setting, modern convergence curves indicate that while training for

long periods of time, test loss continues to decrease [108]. This has led to the practice of training models for as long as possible to achieve better generalization [154]. However, in adversarial training it was noted that training longer can cause overfitting and result in worse test performance [100]. This phenomenon has been referred to as "robust overfitting" and shown to occur with a variety of datasets, model architectures and different threat models.

Regularizers are standard tools in practice to mitigate the effects of overfitting, especially in the regime when the number of parameters are larger than the number of data points. The standard regularization techniques such as l_1 and l_2 regularization and data augmentation methods such as Cutout [155] and Mixup [156] have been shown to be ineffective against robust overfitting phenomena [100]. Only early-stopping using a validation dataset and semi-supervised learning methods that augment the dataset with unlabelled data have been shown to be effective and reduce the generalization gap for adversarially robust learning. Data augmentation using semi-supervised methods however requires the use of additional data that may not be available. Early stopping leads to selection of an earlier checkpoint and causes a trade off between robust accuracy and standard accuracy, as training longer leads to better standard test accuracy.

4.4 Impact of Activation Curvature on Adversarial Training

In this section we consider the effects of curvature for smooth activation functions on standard and robust generalization gaps. We define curvature for smooth activation functions by the maximum of the second derivative[2] i.e $\max_x f''(x)$. We consider the following smooth activation functions, which are ranked by decreasing curvature as follows (see Figure 4.2 for functions and their first

[2]Note that this definition of curvature is different from the standard definition of curvature used for twice-differentiable functions.

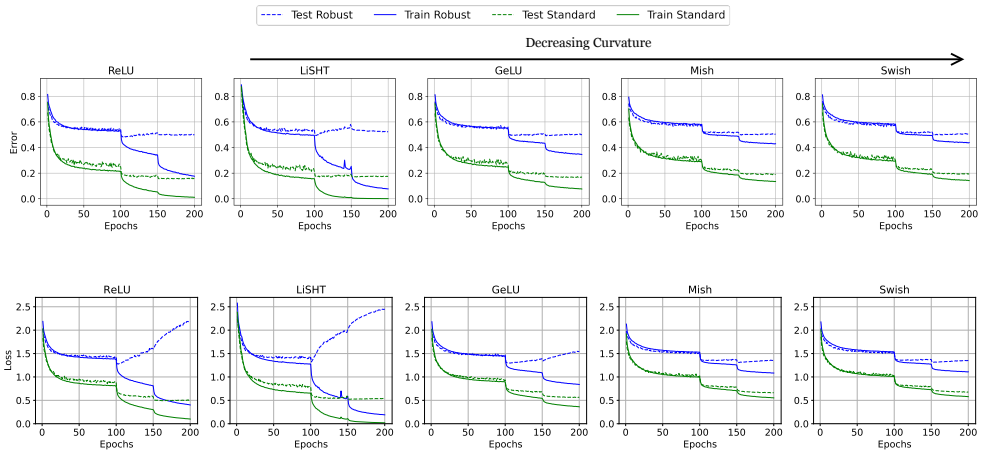


Figure 4.4: Learning curves for CIFAR-10 dataset on Resnet-18 for different activation functions. ReLU activation is non-smooth and included as a baseline, all the other activations are ordered by decreasing curvature from left to right. Top graphs show standard and robust error, and bottom graphs represent loss curves for both train and test data.

and second derivatives):

1. **Linearly Scaled Hyperbolic Tangent** (LiSHT) [157]: $f(x) = x * \tanh(x)$, this function has highest curvature among activations considered.

2. **Gaussian Error Linear Unit** (GeLU) [158]: $f(x) = x * \Phi(x)$, where $\Phi(x)$ is gaussian cummulative distribution function.

3. **Mish** [159]: $f(x) = x * \tanh(ln(1 + \exp(x)))$ is a smooth continuous function similar to SiLU.

4. **SiLU** [103]: $f(x) = x * sigmoid(x)$ is a smooth approximation to ReLU but has a non-monotonic "bump" for $x < 0$.

We also conduct experiments for non-smooth ReLU activation as a baseline. Code for reproducing our experiments can be found at https://github.com/vasusingla/low_curvature_activations.

4.4.1 Analyzing the influence of activations on robustness

In this section, we analyze the theoretical relationship between curvature of the activation function and adversarial robustness. The motivation behind our analysis is to provide an intuition for our observations, we do not rigorously prove a monotonic relationship between robustness and activation curvature. To elucidate this, we first consider the relation between the input Hessian (ie., the second derivatives of the output with respect to the input) and adversarial robustness. We consider a simple binary classifier f, implemented as a two-layer neural network. Let w_1, w_2 be weight matrices for the first and second layers respectively. Let $\sigma(\cdot)$ be a twice differentiable activation function and $\sigma''(\cdot)$ denote the second derivative of the activation function. The two layer neural network can then be represented as $f(x) = w_2^T \sigma(w_1 x)$. Assume the final layer of the network outputs a single logit, which is transformed to probability using a sigmoid function. In other words, the probability of a sample being in class 0 is given as $p(x) = sigmoid(f(x))$. Assuming a sample is classified into class 1 if $p(x) < 0.5$, then a sample x is classified into class 1 iff $f(x) < 0$ and class 0 otherwise. In other words, we use a probability threshold of 0.5, to classify an example into class 1. We assume that the neural network can be locally well approximated using the second order Taylor expansion. We now use the results by [160] about the relation between the input Hessian and robustness. Let x belong to class 1, then for $x + \delta$ to be classified as class 0, the minimal l_2 perturbation that fools the classifier can be written as:

$$\delta^* = \arg\min_{\delta} \|\delta\|$$
$$\text{s.t. } f(x) + \nabla_x f(x)^T \delta + \frac{1}{2} \delta^T \nabla_x^2 f(x) \delta \geq 0$$

It can be shown under these assumptions the magnitude of δ^* can be upper and lower bounded with respect to input curvature. We use the following lemma -

Lemma 2. *[160] Let x be such that $c = -f(x) \geq 0$, and let $g = \nabla_x f(x)$. Assume that $\nu = \lambda_{max}\left(\nabla_x^2 f(x)\right) \geq 0$, denotes the largest eigenvalue and let u be the eigenvector corresponding to ν. Then,*

$$\begin{aligned} \frac{\|g\|}{\nu}\left(\sqrt{1+\frac{2\nu c}{\|g\|^2}}-1\right) &\leq \|\delta^*\| \\ &\leq \frac{\|g^T u\|}{\nu}\left(\sqrt{1+\frac{2\nu c}{(g^T u)^2}}-1\right) \end{aligned} \tag{4.1}$$

This lemma shows that upper and lower bounds on the magnitude of δ^* increase as ν decreases keeping all other factors constant [160]. An increase in $\|\delta^*\|$ therefore increases the minimum l_2 ball required to find an adversarial example for input x, leading to increased robustness. Therefore, a low maximum eigenvalue of the input Hessian leads to higher adversarial robustness.

We now show the relation between activation functions and input curvature. For the considered two layer neural network, the Hessian with respect to the input x is given as:

$$\nabla_x^2 f(x) = w_1^T diag\Big(\sigma''(w_1 x) \odot w_2\Big) w_1 \tag{4.2}$$

where $\odot$ denotes the Hadamard product between two vectors. Equation 4.2 shows that the Hessian of the input directly depends on $\sigma''(.)$, which suggests that an increase in the curvature of the activation function leads to an increase in the norm of the input Hessian. Finally, although we assume our activation to be smooth we expect similar results for non-smooth activations.

We empirically show the relation between ν and activation curvature holds for adversarially

Dataset	Activation	Robust Accuracy				Standard Accuracy			
		Final Train	Final Test	Best Val	Diff.	Final Train	Final Test	Best Val	Diff.
CIFAR-10	LiSHT	92.27	47.21	50.31	**45.06**	99.9	82.53	82.44	**17.37**
	ReLU	82.46	49.25	51.06	**33.21**	98.9	83.73	81.62	**15.17**
	GeLU	65.45	49.31	50.15	**16.14**	92.41	82.81	79.25	**9.6**
	Mish	57	49.18	49.62	**7.82**	86.48	80.05	79.96	**6.43**
	SiLU	56.15	48.91	49.41	**7.24**	85.79	80.55	80.57	**5.24**
CIFAR-100	LiSHT	93.58	18.62	22.48	**74.96**	99.92	49.12	49.13	**50.8**
	ReLU	79.87	18.81	25.91	**61.06**	98.58	51.58	51.05	**47**
	GeLU	57.96	21.56	26.33	**36.4**	89.18	53.67	49.5	**35.51**
	Mish	39.65	24.27	25.88	**15.38**	71.5	53.43	48.37	**18.07**
	SiLU	37.81	24.29	25.82	**13.52**	68.73	52.65	52.18	**16.08**

Table 4.1: Performance of different activations on CIFAR-10 and CIFAR-100 with ResNet-18. We use the best checkpoint based on **best robust accuracy** on the validation set shown in "Best Val"" column. The generalization gap, i.e difference between final train and final test accuracy is shown in "Diff." column. Generalization gap for both standard and robust accuracy increases for activations with high curvature.

trained Resnet-18 models. The learning curves presented in Fig. 4.3 show that for activations with high curvature, the maximum eigenvalue of the input Hessian indeed is larger. This result combined with our previous observation therefore suggests high activation curvature indeed leads to lower robustness.

4.4.2 Activation Curvature and Generalization Gap

In this section we show results for the adversarial training for different smooth activation functions. We hypothesize that for adversarial trained networks, activations with low curvature are more robust and have a small generalization gap.

Experimental Settings - We show our results on the CIFAR-10 and CIFAR-100 dataset [57]. For comparison with best early-stop checkpoint [100], we randomly split the original set into training and validation set with 90% and 10% of the images respectively. We consider the

l_∞ threat model and use PGD-10 step attack with a single restart for training and PGD-20 step attack with 5 restarts for reporting the test accuracy. For the attack hyper-parameters, we use $\epsilon = 8/255$ and $\alpha = 2/255$. We use the ResNet-18 [1] architecture for all our experiments except for experiments with double descent curves where we use Wide ResNet-28 [161]. We use the same training setup as [100] throughout the chapter, an SGD optimizer with momentum of 0.9 and weight decay 5×10^{-4} for 200 epochs with batch size of 128.

We discover that choice of activation function has a large impact on robust overfitting. Figure 4.4 shows our results. First we reproduce the effect of robust overfitting observed by Rice et al.[100] for all the activations. The robust training loss keeps decreasing, however robust test loss rises shortly after the first learning rate drop. For standard training and standard test loss however, both keep decreasing throughout training. Training appears to proceed smoothly at the start, however at the learning rate drop on the 100th and 150th epochs, robust test error decreases briefly and then keeps increasing as training progresses. This phenomenon shows the best performance for robust test accuracy is **not** achieved by training till convergence, unlike standard training. In contrast the best standard accuracy for adversarial training is still reached by training till convergence. We show that for activation functions with lower curvature the robust overfitting phenomenon occurs to a lesser degree. In contrast to Xie et al.[104], we also show that LiSHT a smooth activation function performs worse than the non-smooth ReLU function and shows a larger robust generalization gap as shown in Fig. 4.4. We also note that *for activations that display a large robust generalization gap, the standard generalization gap is also higher*. Finally, the curvature of the activation function has a direct impact on both the robust and standard generalization gaps, as shown in the learning curves. For activations with high curvature such as LiSHT and GeLU the generalization gap is large and for activations with low curvature such as

Mish and SiLU the generalization gap is much lower. Note that although the training loss/error is higher for activations with lower curvature, adversarial training is much more stable and allows training till convergence, achieving better standard accuracy and maintaining similar robust accuracy.

We show the quantitative results in Table 4.1. To show the gap due to robust overfitting (decay in performance from peak robust accuracy) we also show the best robust accuracy found using early stopping with a validation set. We also report the corresponding standard accuracy for the **best robust accuracy checkpoint** (not the best standard accuracy checkpoint). The robust and standard generalization gap decrease for CIFAR-10 and CIFAR-100 as seen in Table 4.1. The effects of robust overfitting, (i.e difference in best and final checkpoint on robust accuracy) also decreases for activations with smaller curvature. For example, the overfitting gap falls from 3.1% for LiSHT to 0.5% for SiLU on CIFAR-10. Standard accuracy however, either remains the same or improves by training longer (compared to the best checkpoint). On CIFAR-100 upon training till convergence, *SiLU simultaneously achieves both robust and standard accuracy higher than ReLU.* These results therefore validate our claim that low curvature activations reduce robust overfitting. Using the best validation checkpoint for CIFAR-100, *SiLU achieves nearly the same robust accuracy and higher standard accuracy than ReLU.* The results therefore show that for adversarial training, curvature of the activation function play an important role in obtaining high robust and standard accuracy.

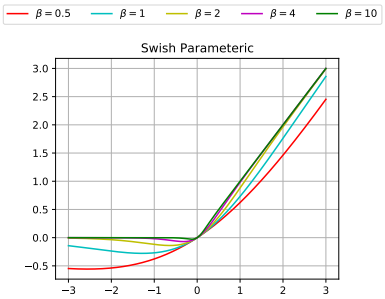


Figure 4.5: Visualization of PSwish with different β values.

β	Robust Accuracy			Standard Accuracy		
	Train	Test	Diff.	Train	Test	Diff.
0.5	47.00	45.24	**1.76**	75.39	73.57	**1.82**
1	56.15	48.91	**7.31**	85.79	80.55	**5.24**
2	69.65	49.6	**20.05**	94.57	83.39	**11.18**
4	83	49.92	**33.08**	98.82	84.48	**14.34**
10	89.2	50.63	**38.57**	99.7	83.57	**16.13**

Table 4.2: Performance of PSwish with different β values, higher β value indicates higher curvature. Results are shown for final checkpoint and show that for activations with high curvature, standard and robust generalization gap increases.

4.4.3 Curvature effects with Parameteric Swish

To further understand the impact of activation curvature on standard and robust generalization gap, we conduct analysis with *Parameteric Swish (PSwish)* [162], defined as follows:

$$f(x) = x \cdot sigmoid(\beta x)$$

The SiLU function defined previously is a special case of PSwish, when $\beta = 1$. PSwish transitions from the identity function for $\beta = 0$, to ReLU for $\beta \to \infty$. The curvature of PSwish increases as β increases. Figure 4.5 shows the PSwish activation function for different values of β.

We show the results with the CIFAR-10 dataset, for final checkpoints for training and

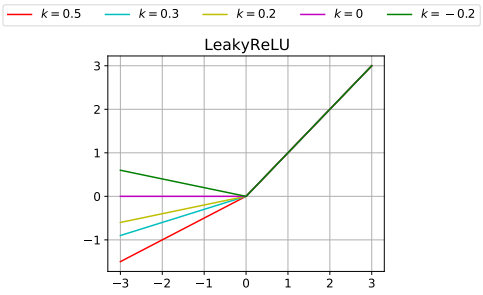


Figure 4.6: Visualization of LeakyReLU with different k values.

k	Robust Accuracy			Standard Accuracy		
	Train	Test	Diff.	Train	Test	Diff.
0.5	52.74	48.71	**4.03**	82.99	79.56	**3.43**
0.3	63.06	49.62	**13.44**	92.00	83.56	**8.44**
0.2	69.64	49.34	**20.3**	95.37	84.21	**11.16**
0	82.46	49.47	**32.99**	98.9	83.73	**15.17**
-0.2	85.89	48.16	**37.73**	99.47	83.01	**16.46**

Table 4.3: Performance of the LeakyReLU activation function with different slope values. The standard and robust generalization gap increases for slopes with larger approximate curvature.

testing set in Table 4.2. Interestingly, we observe that both the standard and robust generalization gap are extremely dependent on the choice of β. The robust generalization gap increases from 1.76 to 38.57 and the standard generalization gap increases from 1.82 to 16.13 for $\beta = 0.5$ and $\beta = 10$ respectively. We also observe that robust test accuracy for the final checkpoint increases from 45.24 to 50.63 for the same β values. For larger values of β i.e $\beta \to \infty$, PSwish behaves like ReLU and standard and robust final test accuracy start decreasing. The results are consistent with our previous experiments and show that the standard and robust generalization gap increases for activations with high curvature. Further using the early stopping checkpoint with the validation set, PSwish with $\beta = 10$ outperforms ReLU baseline by 0.33% on robust accuracy and 1.24% on standard accuracy, highlighting that the choice of activation function can improve standard and robust performance for adversarially trained models.

4.5 Does smoothness matter?

Xie et al.[104] showed that using smooth activations, adversarial training can achieve better standard and robust accuracy on Imagenet [56]. They posit that using smooth activations can improve gradients, which can both strengthen the attacker and provide better gradient updates to weight parameters, thus achieving superior performance.

In contrast, we show that the relation of the generalization gap to activations can be observed for non-smooth activations as well. We use the non-smooth LeakyReLU activation function defined as follows:

$$\text{LeakyReLU}(k, x) = \begin{cases} x & \text{if } x \geq 0 \\ kx & \text{if } x < 0 \end{cases}$$

where k is a hyper-parameter that can be tuned. The first derivative of LeakyReLU is given as:

$$\frac{d}{dx}\text{LeakyReLU}(k, x) = \begin{cases} 1 & \text{if } x \geq 0 \\ k & \text{if } x < 0 \end{cases}$$

For non-smooth activations, curvature of the activation function however is not well defined. Therefore for LeakyReLU, we use the difference of slopes, i.e $|1 - k|$ as the "approximate" curvature of the function. Hence, for $k \leq 1$ the approximate curvature decreases with increasing value of k . We use the same setup as in previous experiments and show the results for final training and test checkpoints in Table 4.3 on CIFAR-10. We observe behavior similar to smooth activations for LeakyReLU. For $k = 0.5$, the approximate curvature is low, and both robust and standard generalization gap, 4.03 and 3.43 respectively is much smaller than for $k = -0.2$, for

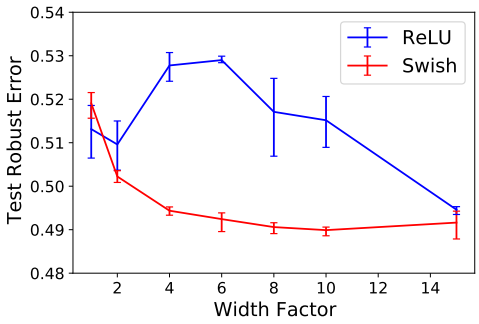


Figure 4.7: The generalization curves show the double descent phenomenon occurs for networks with ReLU activation but does not occur with SiLU activation. We use adversarially trained WideResnet models and the model complexity is controlled by the width of the architecture. Each data point shows the average for last 3 epochs.

which robust and standard generalization gap, 37.73 and 16.46 is large. We therefore hypothesize for non-smooth activations, the "approximate" curvature of the activation function has impact on the generalization gap.

4.6 Double descent curves

The standard bias-variance trade-off from classical machine learning theory fails to explain why deep networks generalize well especially when they have far more parameters than the samples they are trained on [147]. Its now standard practice to use overparameterized models and allow models to train longer [154] since test time performance typically improves for increased model complexity beyond the data interpolation point, a phenomenon known as *double descent* [107]. It was further shown that both training longer and increasing architecture size can be viewed as increase in model complexity and the double descent phenomenon is observed for both settings [108]. The phenomenon of double descent generalization with increase in model width was also briefly noted for l_2 adversarially trained models [108].

Rice et al.[100] show that robust overfitting contradicts the double descent phenomenon

observed with respect to training longer, since training longer harms test time performance. Although, they still observe the double descent phenomenon for ReLU networks with respect to the model size as shown in Fig. 4.7. They therefore posit that, training longer and increasing model size have separate effects on robust generalization.

A recent work [163] suggests that the double descent phenomenon can be mitigated by optimal regularization. We explore whether activations with low curvature can mitigate double descent, by adversarially training Wide Resnets with different width factors. We show results for ReLU and SiLU activation functions in Figure 4.7. Experiments with other activations could not be conducted due to the high expense of training Wide Resnets. We use the SiLU activation function, because it has lowest curvature among all the activations considered. In Figure 4.7, we show the results for ReLU and the SiLU activation function with a PGD-10 adversary. While the double descent phenomenon is observed for ReLU activation, robust test performance continues to decrease for the SiLU activation function. Note that SiLU with *width-factor 4 attains equivalent performance to ReLU with width-factor 15*. None-the-less the final test error achieved by ReLU networks with large width factor is equivalent to the lowest test error achieved by SiLU networks with the same width. This suggest that low curvature activations may not be useful for models with large width. The results also indicate that use of activations with small curvature can act as a regularizer to mitigate the double descent phenomenon.

4.7 Conclusion

In this chapter, we first use both theoretical and empirical approaches to show the impact of curvature of the activation function on robustness. We further show that this property of regu-

larization further extends to non-smooth activations as well. While results from Rice et al.show that classical regularization techniques are unable to prevent robust overfitting, our results show that activation functions with low curvature can largely mitigate that. Since robust overfitting is common in adversarial training, the properties of activation functions that we bring to light in this chapter can be useful for state of the art robust models. Finally our experiments also show that double descent, another phenomenon that has a significant impact on robust generalization, can be mitigated using activations with low curvature.

Chapter 5: Second-Order Provable Defenses against Adversarial Attacks

5.1 Introduction

Modern neural networks achieve high accuracy on tasks such as image classification and speech recognition, but are known to be brittle to small, adversarially chosen perturbations of their inputs [15]. A classifier which correctly classifies an image $\mathbf{x}$, can be fooled by an adversary to misclassify an *adversarial example* $\mathbf{x} + \delta$, such that $\mathbf{x} + \delta$ is indistinguishable from $\mathbf{x}$ to a human. Adversarial examples can also fool systems when they are printed out on a paper and photographed with a smart phone [22]. Even in a black box threat model, where the adversary has no access to the model parameters, attackers could target autonomous vehicles by using stickers or paint to create an adversarial stop sign that the vehicle would interpret as a 'yield' or another sign [164]. This trend is worrisome and suggests that adversarial vulnerabilities need to be appropriately addressed before neural networks can be deployed in security critical applications.

In this chapter, we propose a new approach for developing provable defenses against ℓ_2-bounded adversarial attacks as well as computing robustness certifications of pre-trained deep networks with differentiable activations. In contrast to the existing certificates [165, 166] that use the first-order information (upper and lower bounds on the slope), our approach is based on the second-order information (upper and lower bounds on curvature values i.e. eigenvalues of the Hessian). Our approach is based on two key theoretically-justified steps: First, in Theorems 6

and 7, we show that if the eigenvalues of the Hessian of the network (curvatures of the network) are bounded (globally or locally), we can efficiently compute a robustness certificate and develop a defense method against ℓ_2-bounded adversarial attacks using convex optimization. Second, in Theorem 9, we derive a computationally-efficient differentiable bound on the curvature (eigenvalues of the Hessian) of a deep network. We derive this bound by explicitly characterizing the Hessian of a deep network in Lemma 3.

Although the problem of finding the closest adversarial example to a given point for deep nets leads to a non-convex optimization problem, our proposed Curvature-based Robustness Certificate (CRC), under some verifiable conditions, is able to compute points on the decision boundary that are provably closest to the input. That is, it provides the tightest certificate in those cases. For example, for a 2,3,4 layer networks trained on MNIST, we can find provably closest adversarial points for 44.17%, 22.59%, 19.53% cases, respectively (Table 5.2). To the best of our knowledge, our method is the first approach that can efficiently compute provably closest adversarial examples for a significant fraction of examples in non-trivial neural networks.

We note that un-regularized networks, specially deep ones, can obtain large curvature bounds which can lead to small robustness certificates. However, by using the derived curvature bound as a regularizer during training, we significantly decrease curvature values of the network, with little or no decrease in its performance (Table 5.5, Figure 5.1). Using this technique, our method significantly outperforms interval-bound propagation (IBP) [146, 167] and achieves state of the art certified accuracy (Tables 5.3 and 5.4). In particular, our method achieves certified robust accuracy 69.79%, 57.78% and 53.19% while IBP-based methods achieve 44.96%, 44.74% and 44.66% on 2,3 and 4 layer networks, respectively, on the MNIST-dataset (similar results for Fashion-MNIST).

	Certificate problem $(-) = cert$	**Attack problem** $(-) = attack$
primal problem, $p^*_{(-)}$	$\min_{f(\mathbf{x})=0} 1/2\|\mathbf{x} - \mathbf{x}^{(0)}\|^2$	$\min_{\|\mathbf{x}-\mathbf{x}^{(0)}\|\leq\rho} f(\mathbf{x})$
dual function, $d_{(-)}(\eta)$	$\min_{\mathbf{x}} 1/2\|\mathbf{x} - \mathbf{x}^{(0)}\|^2 + \eta f(\mathbf{x})$	$\min_{\mathbf{x}} f(\mathbf{x}) + \eta/2(\|\mathbf{x} - \mathbf{x}^{(0)}\|^2 - \rho^2)$
When is dual solvable?	$-1/M \leq \eta \leq -1/m$	$-m \leq \eta$
dual problem, $d^*_{(-)}$	$\max_{-1/M\leq\eta\leq-1/m} d_{cert}(\eta)$	$\max_{-m\leq\eta} d_{attack}(\eta)$
When primal = dual?	$f(\mathbf{x}^{(cert)}) = 0$	$\|\mathbf{x}^{(attack)} - \mathbf{x}^{(0)}\| = \rho$

Table 5.1: A summary of various primal and dual concepts used in the chapter. f denotes the function of the decision boundary, i.e. $\mathbf{z}^{(L)}_y - \mathbf{z}^{(L)}_t$ where y is the true label and t is the attack target. m and M are lower and upper bounds on the smallest and largest eigenvalues of the Hessian of f, respectively.

Other recent works (e.g. [160, 168]) empirically show that using an *estimate* of curvature at inputs as a regularizer leads to *empirical* robustness on par with the adversarial training. In this chapter, however, we use a bound on the absolute value of curvature (and not an estimate) as a regularizer and show that it results in high *certified* robustness. Moreover, previous works have tried to certify robustness by bounding the Lipschitz constant of the neural network [15, 133, 169, 170, 171]. Our approach, however, is based on bounding the Lipschitz constant of the gradient which in turn leads to bound on the eigenvalues of the Hessian of deep neural networks.

In summary, we make the following contributions:

- We derive a closed-form expression for the Hessian of a deep network with differentiable activation functions (Lemma 3) and derive bounds on the curvature using this closed-form formula (Theorems 8 and 9).
- We develop computationally efficient methods for both the robustness certification as well as the adversarial attack problems (Theorems 6 and 7).
- We provide verifiable conditions under which our method is able to compute points on the decision boundary that are provably closest to the input. Empirically, we show that this

condition holds for a significant fraction of examples (Table 5.2).

- We show that using our proposed curvature bounds as a regularizer during training leads to improved certified accuracy on 2,3 and 4 layer networks (on the MNIST and Fashion-MNIST datasets) compared to IBP-based adversarial training [145, 167] (Tables 5.3 and 5.4). Our robustness certificate (CRC) outperforms CROWN's certificate [165] significantly when trained with our regularizer (Table 5.5).

To the best of our knowledge, this is the first work that (a) demonstrates the utility of second-order information for provable robustness, (b) derives a framework to find the exact robustness certificates in the l_2 norm and the exact worst case adversarial perturbation in an l_2 ball of given a radius under some conditions, and (c) derives an exact closed form expression for the Hessian and bounds on the curvature values using the same.

5.2 Related work

Many defenses have been proposed to make neural networks robust against adversarial examples. These can be classified into empirical defenses which empirically seem to be robust against known adversarial attacks, and certified defenses, which are provably robust against such attacks.

Empirical defenses The best known empirical defense is adversarial training [20, 25, 43]. In this method, a neural network is trained to minimize the worst-case loss over a region around the input. Although such defenses seem to work on existing attacks, there is no guarantee that a more powerful attack would not break them. In fact, most such defenses proposed in the literature were later broken by stronger attacks [123, 125, 172, 173]. To end this arms race between defenses and attacks, several works have tried to focus on certified defenses with formal robustness guarantees.

Certified defenses A classifier is said to be certifiably robust if one can easily obtain a guarantee that a classifier's prediction remains constant within some region around the input. Such defenses typically rely on certification methods which are either exact or conservative. Exact methods report whether or not there exists a adversarial perturbation inside some l_p norm ball. In contrast, conservative methods either certify that no adversarial perturbation exists or decline to make a certification; they may decline even when no such perturbation exists. Exact methods are usually based on Satisfiability Modulo Theories [130, 132, 174, 175] and Mixed Integer linear programming [129, 176, 177, 178, 179]. Unfortunately, they are computationally inefficient and difficult to scale up to even moderately sized neural networks. In contrast, conservative methods are more scalable and efficient which makes them useful for building certified defenses [140, 143, 145, 146, 165, 166, 180, 181, 182, 183, 184, 185, 186, 187, 188]. However, even these methods have not been shown to scale to practical networks that are large and expressive enough to perform well on ImageNet, for example. To scale to such large networks, randomized smoothing has been proposed as a *probabilistically* certified defense.

Randomized smoothing Randomized smoothing was previously proposed by several works [189, 190] as a empirical defense without any formal guarantees. [136] first proved robustness guarantees for randomized smoothing classifier using inequalities from differential privacy. [191] improved upon the same using tools from information theory. Recently, [135] provided a even tighter robustness guarantee for randomized smoothing. [192] proposed a method of adversarial training for the randomized smoothing classifier giving state of the art results in the l_2 norm.

5.3 Notation

Consider a fully connected neural network with L layers and N_I neurons in the Ith layer ($L \geq 2$ and $I \in [L]$) for a multi-label classification problem with C classes ($N_L = C$). The corresponding function of the neural network is $\mathbf{z}^{(L)} : \mathbf{R}^D \to \mathbf{R}^C$ where D is the dimension of the input. For an input $\mathbf{x}$, we use $\mathbf{z}^{(I)}(\mathbf{x}) \in \mathbf{R}^{N_I}$ and $\mathbf{a}^{(I)}(\mathbf{x}) \in \mathbf{R}^{N_I}$ to denote the input (*before* applying the activation function) and output (*after* applying the activation function) of neurons in the Ith hidden layer of the network, respectively. To simplify notation and when no confusion arises, we make the dependency of $\mathbf{z}^{(I)}$ and $\mathbf{a}^{(I)}$ to $\mathbf{x}$ implicit. We define $\mathbf{a}^{(0)}(\mathbf{x}) = \mathbf{x}$ and $N_0 = D$.

With a fully connected architecture, each $\mathbf{z}^{(I)}$ and $\mathbf{a}^{(I)}$ is computed using a transformation matrix $\mathbf{W}^{(I)} \in R^{N_I \times N_{I-1}}$, the bias vector $\mathbf{b}^{(I)} \in R^{N_I}$ and an activation function $\sigma(.)$ as follows:

$$\mathbf{z}^{(I)} = \mathbf{W}^{(I)}\mathbf{a}^{(I-1)} + \mathbf{b}^{(I)}, \qquad \mathbf{a}^{(I)} = \sigma\left(\mathbf{z}^{(I)}\right)$$

We use $(\mathbf{z}_i^{(L)} - \mathbf{z}_j^{(L)})(\mathbf{x})$ as a shorthand for $\mathbf{z}_i^{(L)}(\mathbf{x}) - \mathbf{z}_j^{(L)}(\mathbf{x})$.

We use $[p]$ to denote the set $\{1, \ldots, p\}$ and $[p, q],\ p \leq q$ to denote the set $\{p, p+1, \ldots, q\}$. We use small letters i, j, k etc to denote the index over a vector or rows of a matrix and capital letters I, J to denote the index over layers of network. The element in the ith position of a vector $\mathbf{v}$ is given by $\mathbf{v}_i$, the vector in the ith row of a matrix $\mathbf{A}$ is $\mathbf{A}_i$ while the element in the ith row and j^{th} column of $\mathbf{A}$ is $\mathbf{A}_{i,j}$. We use $\|\mathbf{v}\|$ and $\|\mathbf{A}\|$ to denote the 2-norm and the operator 2-norm of the vector $\mathbf{v}$ and the matrix $\mathbf{A}$, respectively. We use $|\mathbf{v}|$ and $|\mathbf{A}|$ to denote the vector and matrix constructed by taking the elementwise absolute values. We use $\lambda_{max}(\mathbf{A})$ and $\lambda_{min}(\mathbf{A})$ to denote the largest and smallest eigenvalues of a symmetric matrix $\mathbf{A}$. We use $diag(\mathbf{v})$ to denote

the diagonal matrix constructed by placing each element of $\mathbf{v}$ along the diagonal. We use $\odot$ to denote the Hadamard Product, $\mathbf{I}$ to denote the identity matrix. We use $\preccurlyeq$ and $\succcurlyeq$ to denote Linear Matrix Inequalities (LMIs) such that given two symmetric matrices $\mathbf{A}$ and $\mathbf{B}$ where $\mathbf{A} \succcurlyeq \mathbf{B}$ means $\mathbf{A} - \mathbf{B}$ Positive Semi-Definite (PSD).

5.4 Using duality to solve the attack and certificate problems

Consider an input $\mathbf{x}^{(0)}$ with true label y and attack target t. In the certificate problem, our goal is to find a lower bound of minimum l_2 distance between $\mathbf{x}^{(0)}$ and decision boundary $f(\mathbf{x}) = 0$ where $f(\mathbf{x}) = (\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)})(\mathbf{x})$. The problem for solving the exact distance (*primal*) can be written as:

$$
\begin{aligned}
p^*_{cert} &= \min_{f(\mathbf{x})=0} \left[\frac{1}{2} \left\| \mathbf{x} - \mathbf{x}^{(0)} \right\|^2 \right] \\
p^*_{cert} &= \min_{\mathbf{x}} \max_{\eta} \left[\frac{1}{2} \left\| \mathbf{x} - \mathbf{x}^{(0)} \right\|^2 + \eta f(\mathbf{x}) \right]
\end{aligned} \tag{5.1}
$$

However, solving the above problem can be hard in general. Using the minimax theorem (primal $\geq$ dual), we can write the *dual* of the above problem as follows:

$$
\begin{aligned}
p^*_{cert} &\geq \max_{\eta} d_{cert}(\eta) \\
d_{cert}(\eta) &= \min_{\mathbf{x}} \left[\frac{1}{2} \left\| \mathbf{x} - \mathbf{x}^{(0)} \right\|^2 + \eta f(\mathbf{x}) \right]
\end{aligned} \tag{5.2}
$$

From the theory of duality, we know that $d_{cert}(\eta)$ for each value of η gives a lower bound on the exact certification value (the primal solution) p^*_{cert}. However, since f is non-convex, solving $d_{cert}(\eta)$ for every η can be difficult. In the next section, we will prove that the curvature of the

function f is bounded globally:

$$m\mathbf{I} \preccurlyeq \nabla^2_{\mathbf{x}} f \preccurlyeq M\mathbf{I} \qquad \forall \mathbf{x} \in \mathbb{R}^D \tag{5.3}$$

In this case, we have the following theorem (d^*_{cert} is defined in Table 5.1):

Theorem 6. *$d_{cert}(\eta)$ is a convex optimization problem for $-1/M \leq \eta \leq -1/m$. Moreover, If $\mathbf{x}^{(cert)}$ is the solution to d^*_{cert} such that $f(\mathbf{x}^{(cert)}) = 0$, then $p^*_{cert} = d^*_{cert}$.*

Below, we briefly outline the proof while the full proof is presented in Supp. 5.B.1. The Hessian of the *objective function* of the dual $d_{cert}(\eta)$, i.e the function inside the $\min_{\mathbf{x}}$ is given by:

$$\nabla^2_{\mathbf{x}} \left[\frac{1}{2} \left\| \mathbf{x} - \mathbf{x}^{(0)} \right\|^2 + \eta f(\mathbf{x}) \right] = \mathbf{I} + \eta \nabla^2_{\mathbf{x}} f$$

From equation (5.3), we know that the eigenvalues of $\mathbf{I} + \eta\nabla^2_{\mathbf{x}} f$ are bounded between $(1 + \eta m, 1 + \eta M)$ if $\eta \geq 0$, and in $(1 + \eta M, 1 + \eta m)$ if $\eta \leq 0$. In both cases, we can see that for $-1/M \leq \eta \leq -1/m$, all eigenvalues will be non-negative, making the objective function convex. When $\mathbf{x}^{(cert)}$ satisfies $f(\mathbf{x}^{(cert)}) = 0$, we have $d^*_{cert} = 1/2\|\mathbf{x}^{(cert)} - \mathbf{x}^{(0)}\|^2$. Using the duality theorem we have $d^*_{cert} \leq p^*_{cert}$ and from the definition of p^*_{cert}, we have $p^*_{cert} \leq d^*_{cert}$. Combining the two inequalities, we get $p^*_{cert} = d^*_{cert}$.

Next, we consider the attack problem. The goal here is to find an adversarial example inside an l_2 ball of radius ρ such that $f(\mathbf{x})$ is minimized. Using similar arguments, we can get the following theorem for the attack problem (p^*_{attack}, d^*_{attack} and d_{attack} are defined in Table 5.1):

Theorem 7. *$d_{attack}(\eta)$ is a convex optimization problem for $-m \leq \eta$. Moreover, if $\mathbf{x}^{(attack)}$ is the solution to d^*_{attack} such that $\left\|\mathbf{x}^{(attack)} - \mathbf{x}^{(0)}\right\| = \rho$, $p^*_{attack} = d^*_{attack}$.*

The proof is presented in Supp. 5.B.2. We note that both Theorems 6 and 7 hold for any non-convex function with continuous gradients. Thus they can also be of interest in problems such as optimization of neural nets. Using Theorems 6 and 7, we have the following definitions for certification and attack optimizations:

Definition 5. *(**Curvature-based Certificate Optimization**) Given an input $\mathbf{x}^{(0)}$ with true label y, false target t, we define $(\eta^{(cert)}, \mathbf{x}^{(cert)})$ as the solution of the following max-min optimization:*

$$\max_{-1/M \leq \eta \leq -1/m} \min_{\mathbf{x}} \left[\frac{1}{2} \left\| \mathbf{x} - \mathbf{x}^{(0)} \right\|^2 + \eta f(\mathbf{x}) \right]$$

*We refer to $\left\| \mathbf{x}^{(cert)} - \mathbf{x}^{(0)} \right\|$ as the **C**urvature-based **R**obustness **C**ertificate (CRC).*

Definition 6. *(**Curvature-based Attack Optimization**) Given input $\mathbf{x}^{(0)}$ with label y, false target t, and the l_2 ball radius ρ, we define $(\eta^{(attack)}, \mathbf{x}^{(attack)})$ as the solution of the following optimization:*

$$\max_{\eta \geq -m} \min_{\mathbf{x}} \left[\frac{\eta}{2} \left(\left\| \mathbf{x} - \mathbf{x}^{(0)} \right\|^2 - \rho^2 \right) + f(\mathbf{x}) \right]$$

*When $\mathbf{x}^{(attack)}$ is used for training in an adversarial training framework, we call the method the **C**urvature-based **R**obust **T**raining (CRT).*

A direct implication of Theorems 6 and 7 is that the tightness of our robustness certificate crucially depends on the tightness of our curvature bounds, m and M. If m and M are very large compared to the true eigenvalue bounds of the Hessian of the network, the resulting robustness certificate will be vacuous. In Table 5.5 (and Figure 5.1), we show that by adding the derived bound as a regularization term during the training, we can significantly decrease curvature bounds of the network, with little or no decrease in its performance. This leads to high

robustness certifications against adversarial attacks.

5.5 Curvature Bounds for deep networks

In this section, we provide a computationally efficient approach to compute the curvature bounds for neural networks with differentiable activation functions. To the best of our knowledge, there is no prior work on finding provable bounds on the curvature values of deep neural networks.

5.5.1 Closed form expression for the Hessian

Using the chain rule of second derivatives, we can derive $\nabla^2_{\mathbf{x}} \mathbf{z}_i^{(L)}$ as a sum of matrix products:

Lemma 3. *Given an L layer neural network, the Hessian of the ith hidden unit with respect to the input $\mathbf{x}$, i.e $\nabla^2_{\mathbf{x}} \mathbf{z}_i^{(L)}$ is given by the following formula:*

$$\nabla^2_{\mathbf{x}} \mathbf{z}_i^{(L)} = \sum_{I=1}^{L-1} \left(\mathbf{B}^{(I)}\right)^T diag\left(\mathbf{F}_i^{(L,I)} \odot \sigma''\left(\mathbf{z}^{(I)}\right)\right) \mathbf{B}^{(I)}$$

where $\mathbf{B}^{(I)}$ is the Jacobian of $\mathbf{z}^{(I)}$ with respect to $\mathbf{x}$ (dimensions $N_I \times D$), and $\mathbf{F}^{(L,I)}$ is the Jacobian of $\mathbf{z}^{(L)}$ with respect to $\mathbf{a}^{(I)}$ (dimensions $N_L \times N_I$).

The proof is presented in Supp. 5.B.3. Using the chain rule, we can compute $\mathbf{B}^{(I)}$, $\mathbf{F}^{(L,I)}$ matrices in Lemma 3 recursively as follows:

$$\mathbf{B}^{(I)} = \begin{cases} \mathbf{W}^{(1)}, & I = 1 \\ \mathbf{W}^{(I)} diag\left(\sigma'\left(\mathbf{z}^{(I-1)}\right)\right)\mathbf{B}^{(I-1)}, & I \geq 2 \end{cases}$$

$$\mathbf{F}^{(L,I)} = \begin{cases} \mathbf{W}^{(L)}, & I = L-1 \\ \mathbf{W}^{(L)} diag\left(\sigma'\left(\mathbf{z}^{(L-1)}\right)\right)\mathbf{F}^{(L-1,I)}, & I \leq L-2 \end{cases}$$

This leads to a fast back-propagation like method that can be used to compute the Hessian. Note that Lemma 3 only assumes a matrix multiplication operation from $\mathbf{a}^{(I-1)}$ to $\mathbf{z}^{(I)}$. Since a convolution operation can also be expressed as a matrix multiplication, we can directly extend this lemma to deep convolutional networks. Furthermore, Lemma 3 can also be of independent interest in other related problems such as second-order interpretation methods for deep learning (e.g. [193]).

5.5.2 Curvature bounds for Two Layer networks

For a two-layer network and using Lemma 3, $\nabla^2_{\mathbf{x}}\left(\mathbf{z}^{(2)}_y - \mathbf{z}^{(2)}_t\right)$ is given by:

$$\left(\mathbf{W}^{(1)}\right)^T diag\left(\left(\mathbf{W}^{(2)}_y - \mathbf{W}^{(2)}_t\right) \odot \sigma''\left(\mathbf{z}^{(1)}\right)\right)\mathbf{W}^{(1)}$$

In the above equation, note that only the term $\sigma''(\mathbf{z}^{(1)})$ depends on $\mathbf{x}$. We can maximize and minimize each element in the diag term, $(\mathbf{W}^{(2)}_{y,i} - \mathbf{W}^{(2)}_{t,i})\sigma''(\mathbf{z}^{(1)}_i)$ independently subject to the constraint that $\sigma''(.)$ is bounded. Using this procedure, we construct matrices $\mathbf{P}$ and $\mathbf{N}$ that satisfy properties given in the following theorem:

Theorem 8. *Given a two layer network whose activation function has bounded second derivative:*

$$h_L \leq \sigma''(x) \leq h_U \quad \forall x \in \mathbb{R}$$

(a) We have the following linear matrix inequalities (LMIs):

$$\mathbf{N} \preccurlyeq \nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)} \right) \preccurlyeq \mathbf{P} \qquad \forall \mathbf{x} \in \mathbb{R}^D$$

(b) If $h_U \geq 0$ and $h_L \leq 0$, $\mathbf{P}$ is PSD, $\mathbf{N}$ is a NSD matrix.

(c) This gives the following global bounds on the eigenvalues of the Hessian:

$$m\mathbf{I} \preccurlyeq \nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)} \right) \preccurlyeq M\mathbf{I} \tag{5.4}$$

where $M = \lambda_{max}(\mathbf{P}),\ m = \lambda_{min}(\mathbf{N})$

$\mathbf{P}$ and $\mathbf{N}$ are independent of $\mathbf{x}$ and defined in equations (5.45) and (5.46) in Supp. 5.B.4.

The proof is presented in Supp. 5.B.4. Because power iteration finds the eigenvalue with largest magnitude, we can use it to find m and M only when $\mathbf{P}$ is PSD and $\mathbf{N}$ is NSD. We solve for h_U and h_L for sigmoid, tanh, softplus activation functions in Supp. 5.A.3 and show that this is in fact the case for them.

5.5.3 Curvature bounds for Deep networks

Using Lemma 3, we know that $\nabla^2_{\mathbf{x}}\mathbf{z}_i^{(L)}$ is a sum product of matrices $\mathbf{B}^{(I)}$ and $\mathbf{F}_i^{(L,I)}$. Thus, if we can find upper bounds for $\|\mathbf{B}^{(I)}\|$ and $\|\mathbf{F}_i^{(L,I)}\|_\infty$, we can get upper bounds for $\|\nabla^2_{\mathbf{x}}\mathbf{z}_i^{(L)}\|$. Using this intuition (proof is presented in Supp. 5.B.5), we have the following result:

Theorem 9. *Given an L layer neural network whose activation function satifies:*

$$|\sigma^{'}(x)| \leq g,\ |\sigma^{''}(x)| \leq h \qquad \forall x \in \mathbb{R},$$

the absolute value of eigenvalues of $\nabla^2_{\mathbf{x}}\mathbf{z}_i^{(L)}$ is globally bounded by the following quantity:

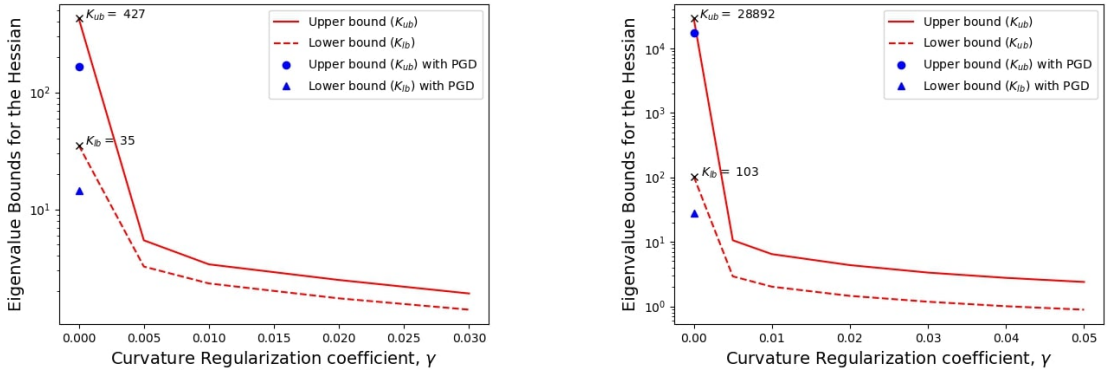
$$\left\|\nabla^2_{\mathbf{x}}\mathbf{z}_i^{(L)}\right\| \leq h\sum_{I=1}^{L-1}\left(r^{(I)}\right)^2 \max_j\left(\mathbf{S}_{i,j}^{(L,I)}\right), \quad \forall \mathbf{x} \in \mathbb{R}^D$$

where $r^{(I)}$ and $\mathbf{S}^{(L,I)}$ are independent of $\mathbf{x}$ and defined recursively as:

$$r^{(I)} = \begin{cases} \left\|\mathbf{W}^{(1)}\right\|, & I = 1 \\ g\left\|\mathbf{W}^{(I)}\right\| r^{(I-1)}, & I \geq 2 \end{cases} \tag{5.5}$$

$$\mathbf{S}^{(L,I)} = \begin{cases} \left|\mathbf{W}^{(L)}\right|, & I = L-1 \\ g\left|\mathbf{W}^{(L)}\right| \mathbf{S}^{(L-1,I)}, & I \leq L-2 \end{cases} \tag{5.6}$$

The above expressions allows for an extremely efficient computation of the curvature bounds for deep networks. We consider simplification of this result for sigmoid, tanh, softplus activations in Supp. 5.A.3. The curvature bounds for $\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)}$ can be computed by replacing



(a) 2 layer network

(b) 3 layer network

Figure 5.1: Illustration of lower (K_{lb}) and upper (K_{ub}) bounds on the curvature of 2 and 3 layer networks with sigmoid activations trained on MNIST. Without any curvature regularization ($\gamma = 0$), curvature bounds increase significantly for deeper networks. Similarly with $\gamma = 0$, networks adversarially trained with PGD have high curvature as well (note the log scale of the y-axis). However, using our curvature bound as a regularizer, the bound becomes tight and CRC gives high certificate values (Table 5.5).

$\mathbf{W}_i^{(L)}$ with $\mathbf{W}_y^{(L)} - \mathbf{W}_t^{(L)}$ in Theorem 9. The resulting bound is independent of $\mathbf{x}$, and only depends on network weights $\mathbf{W}^{(I)}$, the true label y, and the target t. We denote it with $K(\mathbf{W}, y, t)$. To simplify notation, when no confusion arises we denote it with K. In our experiments, for two layer networks, we use M, m from Theorem 8 since it provides tighter curvature bounds. For deeper networks ($L \geq 3$), we use $M = K,\ m = -K$.

5.6 Adversarial training with curvature regularization

Since the term $\mathbf{B}^{(I)}$ in Lemma 3 is the Jacobian of $\mathbf{z}^{(I)}$ with respect to $\mathbf{x}$, $\|\mathbf{B}^{(I)}\|$, it is equal to the lipschitz constant of the neural network constructed from the first I layers of the original network. Finding tight bounds on the lipschitz constant is an active area of research [166, 194, 195] and the product of the operator norm of weight matrices is known to be a loose bound on the lipschitz constant for deep networks. Since we use the same product to compute the bound for $\|\mathbf{B}^{(I)}\|$ in

Theorem 9, the resulting curvature bound is likely to be loose for very deep networks.

In Figure 5.1, we observe the same trend: as the depth of the network increases, the upper bound K_{ub} computed using Theorem 9 becomes significantly larger than the lower bound K_{lb} (computed by taking the maximum of the largest eigenvalue of the Hessian across all test images with label y and the second largest logit t, then averaging across different (y, t)). However, by regularizing the network to have small curvature during training, the bound becomes significantly tighter. Interestingly, using curvature regularization, even with this loose curvature bound for deep nets, we achieve significantly higher robust accuracy than the current state of the art methods while enjoying significantly higher standard accuracy as well (see Tables 5.3 and 5.4).

To regularize the network to have small curvature values, we penalize the curvature bound K during training. To compute the gradient of K with respect to the network weights, note that using Theorem 9, we can compute K using absolute value, matrix multiplications, and operator norm. Since the gradient of operator norm does not exist in standard libraries, we created a new layer where the gradient of $\|\mathbf{W}^{(I)}\|$ i.e $\nabla_{\mathbf{W}^{(I)}}\|\mathbf{W}^{(I)}\|$ is given by:

$$\nabla_{\mathbf{W}^{(I)}}\|\mathbf{W}^{(I)}\| = \mathbf{u}^{(I)}\left(\mathbf{v}^{(I)}\right)^T$$
$$\text{where} \quad \mathbf{W}^{(I)}\mathbf{v}^{(I)} = \|\mathbf{W}^{(I)}\|\mathbf{u}^{(I)}$$

Note that $\|\mathbf{W}^{(I)}\|$, $\mathbf{u}^{(I)}$ and $\mathbf{v}^{(I)}$ can be computed using power iteration. Since the network weights do not change significantly during a single training step, we can use the singular vectors $\mathbf{u}^{(I)}$ and $\mathbf{v}^{(I)}$ computed in the previous training step to update $\mathbf{W}^{(I)}$ using one iteration of power method. This approach to compute the gradient of the largest singular value of a matrix has also been used in previous published work [196]. Thus, the per-sample loss for training with curvature

regularization is given by:

$$\ell\left(\mathbf{z}^{(L)}(\mathbf{x}^{(0)}),\ y\right) + \gamma K(\mathbf{W},\ y,\ t) \tag{5.7}$$

where ℓ denotes the cross entropy loss, y is the true label of the input $\mathbf{x}^{(0)}$, t is the attack target and γ is the regularization coefficient for penalizing large curvature values. Similar to the adversarial training, in CRT, we use $\mathbf{x}^{(attack)}$ instead of $\mathbf{x}^{(0)}$ in equation (5.7).

5.7 Experiments

The *certified robust accuracy* means the fraction of correctly classified test samples whose robustness certificates (computed using CRC) are greater than a pre-specified radius ρ. Unless otherwise specified, we use the class with the second largest logit as the attack target (i.e. the class t). The notation ($L \times [1024]$, activation) denotes a neural network with L layers with the specified activation, ($\gamma = c$) denotes standard training with γ set to c, while (CRT, c) denotes CRT training with $\gamma = c$. Certificates are computed over 150 randomly chosen correctly classified images. We use a single NVIDIA GeForce RTX 2080 Ti GPU.

5.7.1 Fraction of inputs with tightest robustness certificate

Using the verifiable condition of Theorems 6 and 7, our approach is able to (1) find points that are provably the worst case adversarial perturbations (in the l_2 norm) in the attack problem and (2) find points on the decision boundary that are provably closest to the input in the l_2 norm in the certification problem. In particular, in Table 5.2, we observe that for curvature regularized networks, our approach finds provably worst-case adversarial perturbations for *all* of the inputs

with a small drop in the accuracy. Moreover, for 2,3,and 4 layer networks, our method finds provably closest adversarial examples for 44.17%, 22.59% and 19.53% of inputs in the MNIST test set, respectively.

Network	γ	Accuracy	Attack success rate	Certificate success rate
$2 \times [1024]$, sigmoid	0.	98.77%	5.05%	2.24%
	0.01	98.57%	100%	15.68%
	0.02	98.59%	100%	31.56%
	0.03	98.30%	100%	44.17%
$3 \times [1024]$, sigmoid	0.	98.52%	0.%	0.12%
	0.01	98.23%	44.86%	3.34%
	0.03	97.86%	100%	11.51%
	0.05	97.60%	100%	22.59%
$4 \times [1024]$, sigmoid	0.	98.22%	0.%	0.01%
	0.01	97.24%	24.42%	2.68%
	0.03	96.27%	44.42%	6.45%
	0.05	95.77%	99.97%	12.40%
	0.06	95.52%	100%	15.87%
	0.07	95.24%	100%	19.53%

Table 5.2: Certificate success rate denotes the fraction of points satisfying $\mathbf{z}_y - \mathbf{z}_t = 0$, Attack success rate denotes the fraction of points ($\mathbf{x}^{(0)}$) satisfying $\|\mathbf{x}^{(attack)} - \mathbf{x}^{(0)}\|_2 = \rho = 0.5$ implying *primal=dual* in Theorems 6 and 7 respectively. We use the MNIST dataset.

Note that the technique presented in this chapter is not applicable to ReLU networks due to the absence of curvature information. However, since verifying the robustness property in an l_2 ball around the input is known to be an NP-complete problem for ReLU networks [130], it is computationally infeasible to even verify that a given adversarial perturbation is the worst case perturbation in polynomial time unless P=NP. We however show that it is possible to find (and not just verify) the exact worst case perturbation (and robustness certificate) for neural networks with smooth activation functions. We believe that these theoretical and empirical results provide evidence that bounding the curvature of the network and using smooth activation functions can

be critical to achieve high robustness guarantees.

5.7.2 Comparison with existing provable defenses

We compare against certified defense techniques proposed in [146] and [167] in Table 5.3 for the MNIST dataset [197] and Table 5.4 for the Fashion-MNIST dataset [198] with l_2 radius $\rho = 1.58$. Even though our proposed CRT requires fully differentiable activation functions such as softplus, sigmoid, tanh etc, we include comparison with ReLU networks because the methods proposed in [146, 167] use ReLU. Since CROWN-IBP can be trained using the softplus activation function, we include it in our comparison. Similar comparison with l_2 radius $\rho = 0.5$ is given in Supp. Table 5.7 (MNIST dataset) and Table 5.8 (Fashion-MNIST dataset). We observe that CRT (certified with CRC) gives significantly higher certified robust accuracy as well as standard accuracy compared to either of the methods on both MNIST and Fashion-MNIST datasets for both different values of ρ. Since shallow fully connected networks are known to perform poorly on the CIFAR-10 dataset, we do not include those results in our comparison.

Network	Training	Standard Accuracy	Certified Robust Accuracy
$2 \times [1024]$, softplus	**CRT, 0.01**	**98.68%**	**69.79%**
	CROWN-IBP	88.48%	42.36%
$2 \times [1024]$, relu	COAP	89.33%	44.29%
	CROWN-IBP	89.49%	44.96%
$3 \times [1024]$, softplus	**CRT, 0.05**	**97.43%**	**57.78%**
	CROWN-IBP	86.58%	42.14%
$3 \times [1024]$, relu	COAP	89.12%	44.21%
	CROWN-IBP	87.77%	44.74%
$4 \times [1024]$, softplus	**CRT, 0.07**	**95.60%**	**53.19%**
	CROWN-IBP	82.74%	41.34%
$4 \times [1024]$, relu	COAP	90.17%	44.66%
	CROWN-IBP	84.4%	43.83%

Table 5.3: Comparison with interval-bound propagation based adversarial training methods: COAP i.e Convex Outer Adversarial Polytope [146], CROWN-IBP [167]) and Curvature-based Robust Training (Ours) with attack radius $\rho = 1.58$ on MNIST. For CROWN-IBP, we vary the final_beta hyperparameter between 0.8 and 3, and use the model with best certified accuracy.

Network	Training	Standard Accuracy	Certified Robust Accuracy
$2 \times [1024]$, softplus	**CRT, 0.01**	**80.31%**	**54.39%**
	CROWN-IBP	69.23%	47.19%
$2 \times [1024]$, relu	COAP	74.1%	46.3%
	CROWN-IBP	70.73%	48.61%
$3 \times [1024]$, softplus	**CRT, 0.05**	**78.39%**	**53.4%**
	CROWN-IBP	68.72%	46.52%
$3 \times [1024]$, relu	COAP	73.9%	46.3%
	CROWN-IBP	70.79%	48.69%
$4 \times [1024]$, softplus	**CRT, 0.07**	**75.61%**	**49.6%**
	CROWN-IBP	68.31%	46.21%
$4 \times [1024]$, relu	COAP	73.6%	45.1%
	CROWN-IBP	70.21%	48.08%

Table 5.4: Comparison between COAP [146], CROWN-IBP [167]) and Curvature-based Robust Training (Ours) with attack radius $\rho = 1.58$ on Fashion-MNIST.

In Supp. Table 5.10, we compare CRT with Randomized Smoothing [135]. For 2 & 3 layer networks, we achieve higher robust accuracy. However, we note that since our certificate is

deterministic while the smoothing-based certificate is probabilistic (although with high probability), the results are not directly comparable. As a separate result, we also prove that randomized smoothing bounds the curvature of the network (Theorem 10 in Supp. 5.B.6).

Network	**Training**	**Standard Accuracy**	**Certified Robust Accuracy**	**Certificate (mean)**	
				CROWN	**CRC (Ours)**
$2 \times [1024]$, sigmoid	standard	98.37%	54.17%	0.28395	**0.48500**
	$\gamma = 0.01$	98.08%	83.53%	0.32548	**0.84719**
	CRT, 0.01	98.57%	**95.59%**	0.43061	**1.54673**
$3 \times [1024]$, sigmoid	standard	98.37%	0.00%	**0.24644**	0.06874
	$\gamma = 0.01$	97.71%	88.33%	0.39799	**1.07842**
	CRT, 0.01	97.23%	**94.99%**	0.39603	**1.24100**
$4 \times [1024]$, sigmoid	standard	98.39%	0.00%	**0.19501**	0.00454
	$\gamma = 0.01$	97.41%	89.61%	0.40620	**1.05323**
	CRT, 0.01	97.83%	**93.41%**	0.40327	**1.06208**

Table 5.5: Effect of curvature regularization and CRT on certified robust accuracy and certificates

5.7.3 Comparison with existing certificates

In Table 5.5, we compare CRC with CROWN-general [199]. For 2-layer networks, CRC outperforms CROWN significantly. For deeper networks, CRC works better than CROWN when the network is trained with curvature regularization. However, with small $\gamma = 0.01$, we see a significant increase in CRC but a very small drop in the test accuracy (without any adversarial training). We can see that with $\gamma = 0.01$, non-trivial certified accuracies of $83.53\%,\ 88.33\%,\ 89.61\%$ can be achieved on $2, 3, 4$ layer sigmoid networks, respectively, without any adversarial training. Adversarial training using CRT further boosts certified accuracy to $95.59\%,\ 94.99\%$ and 93.41%, respectively. We show some results on CIFAR-10 dataset in Supp. Table 5.11. We again observe improvements in the robustness certificate and certified robust accuracy using CRC and CRT.

5.7.4 Results using local curvature bounds

From Theorems 6 and 7, we can observe that if the curvature is *locally* bounded within a convex region around the input (we call it the "safe" region), then the corresponding dual problems (d^*_{cert}, d^*_{attack}) are again convex optimization problems provided the optimization trajectory does not escape the safe region.

Theorem 8 can be directly extended to compute the local curvature bound using bounds on the second derivatives, i.e. $\sigma^{''}(\mathbf{z}^{(1)})$ in the local region. In Table 5.6, we show significant improvements for the CRC certificate for two-layer sigmoid networks on the MNIST dataset for $\gamma = 0$. However, with the curvature regularization, the difference is insignificant. We also observe that the certified accuracy for (CRT, 0.0) improves from 95.04% to 95.31% and for standard improves from 54.17% to 58.06%. The certified accuracy remains the same for other cases. Implementation details are in the Supp. Section 5.A.2.

Computing local curvature bounds for deeper networks, however, is more challenging due to the presence of terms involving multiplication of first and second derivatives. A straightforward extension of Theorem 9 9, wherein we compute the upper bound on $\sigma^{'}$ and σ'' in a local region around the input across all neurons in all layers does not yield significant improvements over the global method, therefore we do not include those results in our comparison.

Network	Training	CRC (Global)	CRC (Local)
$2 \times [1024]$, sigmoid	standard	0.5013	**0.5847**
	CRT, 0.0	1.0011	**1.1741**
	CRT, 0.01	1.5705	**1.6047**
	CRT, 0.02	1.6720	**1.6831**

Table 5.6: Comparison between Certified Robust accuracy and CRC for 2 layer sigmoid and tanh networks using global and local curvature bounds on MNIST dataset with $\rho = 0.5$

5.8 Conclusion

In this chapter, we develop computationally-efficient convex relaxations for robustness certification and adversarial attack problems given the classifier has a bounded curvature. We also show that this convex relaxation is tight under some general verifiable conditions. To be able to use proposed certification and attack convex optimizations, we derive global curvature bounds for deep networks with differentiable activation functions. This result is a consequence of a closed-form expression that we derive for the Hessian of a deep network. Adversarial training using our attack method coupled with curvature regularization results in a significantly higher certified robust accuracy than the existing provable defense methods. Our proposed curvature-based robustness certificate significantly outperforms the CROWN certificate when trained with our regularizer. Scaling up our proposed curvature-based robustness certification and training methods as well as further tightening the derived curvature bounds are among interesting directions for the future work.

Supplementary material

5.A Implementation Details

5.A.1 Computing the derivative of largest singular value

Our objective is to compute derivative of the largest singular value, i.e $\|\mathbf{W}^{(I)}\|$ with respect to $\mathbf{W}^{(I)}$. Let $\mathbf{u}^{(I)}, \mathbf{v}^{(I)}$ be the singular vectors such that $\mathbf{W}^{(I)}\mathbf{v}^{(I)} = \|\mathbf{W}^{(I)}\|\mathbf{u}^{(I)}$. Then the derivative is given by:

$$\nabla_{\mathbf{W}^{(I)}}\|\mathbf{W}^{(I)}\| = \mathbf{u}^{(I)}\left(\mathbf{v}^{(I)}\right)^T$$

$\mathbf{v}^{(I)}$, $\|\mathbf{W}^{(I)}\|^2$ can be computed by running power iteration on $\left(\mathbf{W}^{(I)}\right)^T\mathbf{W}^{(I)}$. $\mathbf{u}^{(I)}$ can be computed using the identity:

$$\mathbf{u}^{(I)} = \frac{\mathbf{W}^{(I)}\mathbf{v}^{(I)}}{\gamma^{(I)}}$$

We use 25 iterations of the power method to compute the above quantities.

5.A.2 Computing certificate using local curvature bounds

To compute the robustness certificate in a local region around the input, we first compute the certificate using the global bounds on the curvature. Using the same certificate as the initial l_2 radius of the safe region, we can refine our certificate. Due to the reduction in curvature, this will surely increase the value of the certificate. We then use the new robustness certificate as the new l_2 radius of the safe region and repeat. We iterate over this process 5 times to compute the local version of our robustness certificate.

To ensure that the optimization trajectory does not escape the safe region, whenever the gradient descent step lies outside the "safe" region, we reduce the step size by a factor of two until it lies inside the region.

5.A.3 Computing g, h, h_U and h_L for different activation functions

5.A.3.1 Softplus activation

For softplus activation, we have the following. We use $S(x)$ to denote sigmoid:

$$\sigma(x) = \log(1 + \exp(x))$$

$$\sigma^{'}(x) = S(x)$$

$$\sigma^{''}(x) = S(x)(1 - S(x))$$

To bound $S(x)(1 - S(x))$, let α denote $S(x)$. We know that $0 \leq \alpha \leq 1$:

$$\alpha(1 - \alpha) = \frac{1}{4} - \left(\frac{1}{2} - \alpha\right)^2$$

Thus, $S(x)(1 - S(x))$ is maximum at $S(x) = 1/2$ and minimum at $S(x) = 0$ and $S(x) = 1$. The maximum value is 0.25 and minimum value is 0.

$$0 \leq S(x)(1 - S(x)) \leq 0.25 \implies 0 \leq \sigma^{''}(x) \leq 0.25$$

Thus, $h_U = 0.25,\ h_L = 0$ (for use in Theorem 8) and $g = 1,\ h = 0.25$ (for use in Theorem 9).

5.A.3.2 Sigmoid activation

For sigmoid activation, we have the following. We use $S(x)$ to denote sigmoid:

$$\sigma(x) = S(x) = \frac{1}{1 + \exp(-x)}$$

$$\sigma^{'}(x) = S(x)(1 - S(x))$$

$$\sigma^{''}(x) = S(x)(1 - S(x))(1 - 2S(x))$$

The second derivative of sigmoid ($\sigma^{''}(x)$) can be bounded using standard differentiation. Let α denote $S(x)$. We know that $0 \leq \alpha \leq 1$:

$$h_L \leq \sigma^{''}(x) \leq h_U$$

$$h_L = \min_{0 \leq \alpha \leq 1} \alpha(1 - \alpha)(1 - 2\alpha)$$

$$h_U = \max_{0 \leq \alpha \leq 1} \alpha(1 - \alpha)(1 - 2\alpha)$$

To solve for both h_L and h_U, we first differentiate $\alpha(1 - \alpha)(1 - 2\alpha)$ with respect to α:

$$\nabla_\alpha \left(\alpha(1 - \alpha)(1 - 2\alpha)\right) = \nabla_\alpha \left(2\alpha^3 - 3\alpha^2 + \alpha\right)$$

$$= \left(6\alpha^2 - 6\alpha + 1\right)$$

Solving for $6\alpha^2 - 6\alpha + 1 = 0$, we get the solutions:

$$\alpha = \left(\frac{3 + \sqrt{3}}{6}\right), \left(\frac{3 - \sqrt{3}}{6}\right)$$

Since both $(3+\sqrt{3}/6), (3-\sqrt{3}/6)$ lie between 0 and 1, we check for the second derivatives:

$$\begin{aligned}\nabla_\alpha^2\left(\alpha(1-\alpha)(1-2\alpha)\right) &= \nabla_\alpha\left(6\alpha^2-6\alpha+1\right)\\ &= 12\alpha-6 = 6(2\alpha-1)\end{aligned}$$

At $\alpha = (3+\sqrt{3})/6$, $\nabla_\alpha^2 = 6(2\alpha-1) = 2\sqrt{3} > 0$.

At $\alpha = (3-\sqrt{3})/6$, $\nabla_\alpha^2 = 6(2\alpha-1) = -2\sqrt{3} < 0$.

Thus $\alpha = (3+\sqrt{3})/6$ is a local minima, $\alpha = (3-\sqrt{3})/6$ is a local maxima.

Substituting the two critical points into $\alpha(1-\alpha)(1-2\alpha)$, we get $h_U = 9.623 \times 10^{-2}$, $h_L = -9.623 \times 10^{-2}$.

Thus, $h_U = 9.623 \times 10^{-2}$, $h_L = -9.623 \times 10^{-2}$ (for use in Theorem 8) and $g = 0.25$, $h = 0.09623$ (for use in Theorem 9).

5.A.3.3 Tanh activation

For tanh activation, we have the following:

$$\sigma(x) = \tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$

$$\sigma^{'}(x) = (1-\tanh(x))\,(1+\tanh(x))$$

$$\sigma^{''}(x) = -2\tanh(x)\,(1-\tanh(x))\,(1+\tanh(x))$$

The second derivative of tanh , i.e $(\sigma^{''}(x))$ can be bounded using standard differentiation. Let α denote $\tanh(x)$. We know that $-1 \leq \alpha \leq 1$:

$$h_L \leq \sigma^{''}(x) \leq h_U$$

$$h_L = \min_{0 \leq \alpha \leq 1} -2\alpha(1-\alpha)(1+\alpha)$$

$$h_U = \max_{0 \leq \alpha \leq 1} -2\alpha(1-\alpha)(1+\alpha)$$

To solve for both h_L and h_U, we first differentiate $-2\alpha(1-\alpha)(1+\alpha)$ with respect to α:

$$\nabla_\alpha \left(-2\alpha(1-\alpha)(1+\alpha)\right) = \nabla_\alpha \left(2\alpha^3 - 2\alpha\right) = \left(6\alpha^2 - 2\right)$$

Solving for $6\alpha^2 - 2 = 0$, we get the solutions:

$$\alpha = -\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}$$

Since both $-1/\sqrt{3}, 1/\sqrt{3}$ lie between -1 and 1, we check for the second derivatives:

$$\nabla^2_\alpha \left(-2\alpha(1-\alpha)(1+\alpha)\right) = \nabla_\alpha \left(6\alpha^2 - 2\right) = 12\alpha$$

At $\alpha = -1/\sqrt{3}$, $\nabla^2_\alpha = 12\alpha = -4\sqrt{3} < 0$.

At $\alpha = 1/\sqrt{3}$, $\nabla^2_\alpha = 12\alpha = 4\sqrt{3} > 0$.

Thus $\alpha = 1/\sqrt{3}$ is a local minima, $\alpha = -1/\sqrt{3}$ is a local maxima.

Substituting the two critical points into $-2\alpha(1-\alpha)(1+\alpha)$, we get $h_U = 0.76981$, $h_L =$

-0.76981.

Thus, $h_U = 0.76981,\ h_L = -0.76981$ (for use in Theorem 8) and $g = 1,\ h = 0.76981$ (for use in Theorem 9).

5.B Proofs

5.B.1 Proof of Theorem 6

(a)

$$d_{cert}(\eta) = \min_{\mathbf{x}} \left[\frac{1}{2} \left\| \mathbf{x} - \mathbf{x}^{(0)} \right\|^2 + \eta \left(\mathbf{z}_y^{(L)}(\mathbf{x}) - \mathbf{z}_t^{(L)}(\mathbf{x}) \right) \right]$$

$$\nabla_{\mathbf{x}}^2 \left[\frac{1}{2} \left\| \mathbf{x} - \mathbf{x}^{(0)} \right\|^2 + \eta \left(\mathbf{z}_y^{(L)}(\mathbf{x}) - \mathbf{z}_t^{(L)}(\mathbf{x}) \right) \right]$$
$$= \mathbf{I} + \eta \nabla_{\mathbf{x}}^2 \left(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)} \right)$$

We are given that the Hessian $\nabla_{\mathbf{x}}^2 (\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)})$ satisfies the following LMIs:

$$m\mathbf{I} \preccurlyeq \nabla_{\mathbf{x}}^2 \left(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)} \right) \preccurlyeq M\mathbf{I} \qquad \forall \mathbf{x} \in \mathbb{R}^n$$

The eigenvalues of $\mathbf{I} + \eta \nabla_{\mathbf{x}}^2 (\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)})$ are bounded between:

$$(1 + \eta M,\ 1 + \eta m), \text{ if } \eta < 0$$

$$(1 + \eta m,\ 1 + \eta M),\ \text{if } \eta > 0$$

We are given that η satisfies the following inequalities where $m < 0, M > 0$ since $(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)})$ is neither convex, nor concave as a function of $\mathbf{x}$:

$$\frac{-1}{M} \leq \eta \leq \frac{-1}{m}, \qquad m < 0, M > 0$$

We have the following inequalities:

$$1 + \eta M \geq 0,\ 1 + \eta m \geq 0$$

Thus, $\mathbf{I} + \eta \nabla_{\mathbf{x}}^2 (\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)})$ is a PSD matrix for all $\mathbf{x} \in \mathbb{R}^D$ when $-1/M \leq \eta \leq -1/m$. Thus $1/2\|\mathbf{x} - \mathbf{x}^{(0)}\|^2 + \eta(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)})(\mathbf{x})$ is a convex function in $\mathbf{x}$ and $d_{cert}(\eta)$ is a convex optimization problem.

(b) For every value of η, $d_{cert}(\eta)$ is a lower bound for p^*_{cert}. Thus $d^*_{cert} = \max_{-1/M \leq\ \eta\ \leq -1/m} d_{cert}(\eta)$ is a lower bound for p^*_{cert}, i.e:

$$d^*_{cert} \leq p^*_{cert} \tag{5.8}$$

Let $\eta^{(cert)}, \mathbf{x}^{(cert)}$ be the solution of the above dual optimization (d^*_{cert}) such that

$$\mathbf{z}_y^{(L)}(\mathbf{x}^{(cert)}) = \mathbf{z}_t^{(L)}(\mathbf{x}^{(cert)}) \tag{5.9}$$

d^*_{cert} is given by the following:

$$d^*_{cert} = \Bigg[\frac{1}{2}\left\|\mathbf{x}^{(cert)} - \mathbf{x}^{(0)}\right\|^2 + \eta^{(cert)} \underbrace{\left(\mathbf{z}_y^{(L)}(\mathbf{x}^{(cert)}) - \mathbf{z}_t^{(L)}(\mathbf{x}^{(cert)})\right)}_{=0}\Bigg]$$

Since we are given that $\mathbf{z}_y^{(L)}(\mathbf{x}^{(cert)}) = \mathbf{z}_t^{(L)}(\mathbf{x}^{(cert)})$, we get the following equation for d^*_{cert}:

$$d^*_{cert} = \frac{1}{2}\left\|\mathbf{x}^{(cert)} - \mathbf{x}^{(0)}\right\|^2 \tag{5.10}$$

Since p^*_{cert} is given by the following equation:

$$p^*_{cert} = \min_{\mathbf{z}_y^{(L)}(\mathbf{x}) = \mathbf{z}_t^{(L)}(\mathbf{x})} \left[\frac{1}{2}\left\|\mathbf{x} - \mathbf{x}^{(0)}\right\|^2\right] \tag{5.11}$$

Using equations (5.9) and (5.11), p^*_{cert} is the minimum value of $1/2\|\mathbf{x} - \mathbf{x}^{(0)}\|^2 \quad \forall \mathbf{x} : \mathbf{z}_y^{(L)}(\mathbf{x}) = \mathbf{z}_t^{(L)}(\mathbf{x})$:

$$p^*_{cert} \leq \frac{1}{2}\left\|\mathbf{x}^{(cert)} - \mathbf{x}^{(0)}\right\|^2 \tag{5.12}$$

From equation (5.10), we know that $d^*_{cert} = 1/2\|\mathbf{x}^{(cert)} - \mathbf{x}^{(0)}\|^2$. Thus, we get:

$$p^*_{cert} \leq d^*_{cert} \tag{5.13}$$

Using equation (5.8) we have $d^*_{cert} \leq p^*_{cert}$ and using (5.13), $p^*_{cert} \leq d^*_{cert}$

$$p^*_{cert} = d^*_{cert}$$

5.B.2 Proof of Theorem 7

(a)

$$d_{attack}(\eta) = \min_{\mathbf{x}} \Bigg[\left(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)}\right)(\mathbf{x}) + \frac{\eta}{2}\left(\left\|\mathbf{x} - \mathbf{x}^{(0)}\right\|^2 - \rho^2\right) \Bigg]$$

$$\nabla^2_{\mathbf{x}} \Bigg[\left(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)}\right)(\mathbf{x}) + \frac{\eta}{2}\left\|\mathbf{x} - \mathbf{x}^{(0)}\right\|^2 \Bigg]$$
$$= \nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)}\right) + \eta \mathbf{I}$$

Since the Hessian $\nabla^2_{\mathbf{x}}(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)})$ is bounded below:

$$m\mathbf{I} \preccurlyeq \nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)}\right) \qquad \forall \mathbf{x} \in \mathbb{R}^n$$

The eigenvalues of $\nabla^2_{\mathbf{x}}(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)}) + \eta\mathbf{I}$ are bounded below:

$$(m + \eta)\mathbf{I} \preccurlyeq \nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)}\right) + \eta\mathbf{I}$$

Since $\eta \geq -m$.

$$\eta + m \geq 0$$

Thus $\nabla^2_{\mathbf{x}}(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)}) + \eta\mathbf{I}$ is a PSD matrix for all $\mathbf{x} \in \mathbb{R}^D$ when $\eta \geq -m$.

Thus $(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)})(\mathbf{x}) + \eta/2(\|\mathbf{x} - \mathbf{x}^{(0)}\|^2 - \rho^2)$ is a convex function in $\mathbf{x}$ and $d_{attack}(\eta)$ is a convex optimization problem.

(b) For every value of η, $d_{attack}(\eta)$ is a lower bound for p^*_{attack}. Thus $d^*_{attack} = \max_{-m \leq \eta} d_{attack}(\eta)$ is a lower bound for p^*_{attack}:

$$d^*_{attack} \leq p^*_{attack} \tag{5.14}$$

Let $\eta^{(attack)}, \mathbf{x}^{(attack)}$ be the solution of the above dual optimization (d^*_{attack}) such that

$$\left\|\mathbf{x}^{(attack)} - \mathbf{x}^{(0)}\right\| = \rho \tag{5.15}$$

d^*_{attack} is given by the following:

$$\begin{aligned} d^*_{attack} = \Bigg[& \left(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)}\right)(\mathbf{x}^{(attack)}) \\ & + \frac{\eta^{(attack)}}{2} \underbrace{\left(\left\|\mathbf{x}^{(attack)} - \mathbf{x}^{(0)}\right\|^2 - \rho^2\right)}_{=0} \Bigg] \end{aligned} \tag{5.16}$$

Since we are given that $\|\mathbf{x}^{(attack)} - \mathbf{x}^{(0)}\| = \rho$, we get the following equation for d^*_{attack}:

$$d^*_{attack} = \left(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)}\right)(\mathbf{x}^{(attack)}) \tag{5.17}$$

Since p^*_{attack} is given by the following equation:

$$p^*_{attack} = \min_{\left\|\mathbf{x}-\mathbf{x}^{(0)}\right\| \leq \rho} \left[\left(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)} \right) (\mathbf{x}) \right] \tag{5.18}$$

Using equations (5.15) and (5.18), p^*_{attack} is the minimum value of $(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)})(\mathbf{x}) \quad \forall \left\|\mathbf{x} - \mathbf{x}^{(0)}\right\| \leq \rho$:

$$p^*_{attack} \leq \left(\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)} \right) (\mathbf{x}^{(attack)}) \tag{5.19}$$

From equation (5.17), we know that $d^*_{attack} = (\mathbf{z}_y^{(L)} - \mathbf{z}_t^{(L)})(\mathbf{x}^{(attack)})$. Thus, we get:

$$p^*_{attack} \leq d^*_{attack} \tag{5.20}$$

Using equation (5.14) we have $d^*_{attack} \leq p^*_{attack}$ and using (5.20), $p^*_{attack} \leq d^*_{attack}$

$$p^*_{attack} = d^*_{attack}$$

5.B.3 Proof of Lemma 3

We have to prove that for an L layer neural network, the hessian of the ith hidden unit in the Lth layer with respect to the input $\mathbf{x}$, i.e $\nabla^2_{\mathbf{x}} \mathbf{z}_i^{(L)}$ is given by the following formula:

$$\nabla^2_{\mathbf{x}} \mathbf{z}_i^{(L)} = \sum_{I=1}^{L-1} \left(\mathbf{B}^{(I)} \right)^T diag\left(\mathbf{F}_i^{(L,I)} \odot \sigma'' \left(\mathbf{z}^{(I)} \right) \right) \mathbf{B}^{(I)} \tag{5.21}$$

where $\mathbf{B}^{(I)}$, $I \in [L]$ is a matrix of size $N_I \times D$ defined as follows:

$$\mathbf{B}^{(I)} = \left[\nabla_{\mathbf{x}}\mathbf{z}_1^{(I)}, \nabla_{\mathbf{x}}\mathbf{z}_2^{(I)}, \ldots, \nabla_{\mathbf{x}}\mathbf{z}_{N_I}^{(I)}\right]^T \tag{5.22}$$

and $\mathbf{F}^{(L,I)},\ I \in [L-1]$ is a matrix of size $N_L \times N_I$ defined as follows:

$$\mathbf{F}^{(L,I)} = \left[\nabla_{\mathbf{a}^{(I)}}\mathbf{z}_1^{(L)}, \nabla_{\mathbf{a}^{(I)}}\mathbf{z}_2^{(L)}, \ldots, \nabla_{\mathbf{a}^{(I)}}\mathbf{z}_{N_L}^{(L)}\right]^T \tag{5.23}$$

$\nabla_{\mathbf{x}}^2\mathbf{z}_i^{(L)}$ can be written in terms of the activations of the previous layer using the following formula:

$$\nabla_{\mathbf{x}}^2\mathbf{z}_i^{(L)} = \sum_{j=1}^{N_{I-1}} \mathbf{W}_{i,j}^{(L)} \left(\nabla_{\mathbf{x}}^2\mathbf{a}_j^{(L-1)}\right) \tag{5.24}$$

Using the chain rule of the Hessian and $\mathbf{a}^{(I)} = \sigma(\mathbf{z}^{(I)})$, we can write $\nabla_{\mathbf{x}}^2\mathbf{a}_j^{(L-1)}$ in terms of $\nabla_{\mathbf{x}}\mathbf{z}_j^{(L-1)}$ and $\nabla_{\mathbf{x}}^2\mathbf{z}_j^{(L-1)}$ as the following:

$$\begin{aligned}\nabla_{\mathbf{x}}^2\mathbf{a}_j^{(L-1)} = {} & \sigma''\left(\mathbf{z}_j^{(L-1)}\right)\left(\nabla_{\mathbf{x}}\mathbf{z}_j^{(L-1)}\right)\left(\nabla_{\mathbf{x}}\mathbf{z}_j^{(L-1)}\right)^T \\ & + \sigma'\left(\mathbf{z}_j^{(L-1)}\right)\left(\nabla_{\mathbf{x}}^2\mathbf{z}_j^{(L-1)}\right)\end{aligned} \tag{5.25}$$

Replacing $\nabla^2_{\mathbf{x}} \mathbf{a}_j^{(L-1)}$ using equation (5.25) into equation (5.24), we get:

$$\nabla^2_{\mathbf{x}} \left(\mathbf{z}_i^{(L)} \right) =$$
$$\sum_{j=1}^{N_{L-1}} \mathbf{W}_{i,j}^{(L)} \Big[\sigma'' \left(\mathbf{z}_j^{(L-1)} \right) \left(\nabla_{\mathbf{x}} \mathbf{z}_j^{(L-1)} \right) \left(\nabla_{\mathbf{x}} \mathbf{z}_j^{(L-1)} \right)^T$$
$$+ \sigma' \left(\mathbf{z}_j^{(L-1)} \right) \left(\nabla^2_{\mathbf{x}} \mathbf{z}_j^{(L-1)} \right) \Big]$$

$$\nabla^2_{\mathbf{x}} \left(\mathbf{z}_i^{(L)} \right) = \tag{5.26}$$
$$\sum_{j=1}^{N_{L-1}} \mathbf{W}_{i,j}^{(L)} \sigma'' \left(\mathbf{z}_j^{(L-1)} \right) \left(\nabla_{\mathbf{x}} \mathbf{z}_j^{(L-1)} \right) \left(\nabla_{\mathbf{x}} \mathbf{z}_j^{(L-1)} \right)^T$$
$$+ \sum_{j=1}^{N_{L-1}} \mathbf{W}_{i,j}^{(L)} \sigma' \left(\mathbf{z}_j^{(L-1)} \right) \left(\nabla^2_{\mathbf{x}} \mathbf{z}_j^{(L-1)} \right) \tag{5.27}$$

For each $I \in [2, L],\ i \in N_I$, we define the matrix $\mathbf{A}_i^{(I)}$ as the following:

$$\nabla^2_{\mathbf{x}} \left(\mathbf{z}_i^{(I)} \right)$$
$$= \underbrace{\sum_{j=1}^{N_{I-1}} \mathbf{W}_{i,j}^{(I)} \sigma'' \left(\mathbf{z}_j^{(I-1)} \right) \left(\nabla_{\mathbf{x}} \mathbf{z}_j^{(I-1)} \right) \left(\nabla_{\mathbf{x}} \mathbf{z}_j^{(I-1)} \right)^T}_{\mathbf{A}_i^{(I)}}$$
$$+ \sum_{j=1}^{N_{I-1}} \mathbf{W}_{i,j}^{(I)} \sigma' \left(\mathbf{z}_j^{(I-1)} \right) \left(\nabla^2_{\mathbf{x}} \mathbf{z}_j^{(I-1)} \right) \tag{5.28}$$
$$\mathbf{A}_i^{(I)} = \sum_{j=1}^{N_{I-1}} \mathbf{W}_{i,j}^{(I)} \sigma'' \left(\mathbf{z}_j^{(I-1)} \right) \left(\nabla_{\mathbf{x}} \mathbf{z}_j^{(I-1)} \right) \left(\nabla_{\mathbf{x}} \mathbf{z}_j^{(I-1)} \right)^T \tag{5.29}$$

Substituting $\mathbf{A}_i^{(L)}$ using equation (5.29) into equation (5.27), we get:

$$\nabla_{\mathbf{x}}^2 \left(\mathbf{z}_i^{(L)}\right) = \mathbf{A}_i^{(L)} + \sum_{j=1}^{N_{I-1}} \mathbf{W}_{i,j}^{(I)} \sigma' \left(\mathbf{z}_j^{(I-1)}\right) \left(\nabla_{\mathbf{x}}^2 \mathbf{z}_j^{(I-1)}\right) \tag{5.30}$$

We first simplify the expression for $\mathbf{A}_i^{(L)}$. Note that $\mathbf{A}_i^{(L)}$ is a sum of symmetric rank one matrices $\left(\nabla_{\mathbf{x}} \mathbf{z}_j^{(L-1)}\right) \left(\nabla_{\mathbf{x}} \mathbf{z}_j^{(L-1)}\right)^T$ with the coefficient $\mathbf{W}_{i,j}^{(L)} \sigma'' \left(\mathbf{z}_j^{(L-1)}\right)$ for each j. We create a diagonal matrix for the coefficients and another matrix $\mathbf{B}^{(L-1)}$ such that each jth row of $\mathbf{B}^{(L-1)}$ is the vector $\nabla_{\mathbf{x}} \mathbf{z}_j^{(L-1)}$. This leads to the following equation:

$$\begin{aligned} \mathbf{A}_i^{(L)} &= \sum_{j=1}^{N_{L-1}} \mathbf{W}_{i,j}^{(L)} \sigma'' \left(\mathbf{z}_j^{(L-1)}\right) \left(\nabla_{\mathbf{x}} \mathbf{z}_j^{(L-1)}\right) \left(\nabla_{\mathbf{x}} \mathbf{z}_j^{(L-1)}\right)^T \\ &= \left(\mathbf{B}^{(L-1)}\right)^T diag \left(\mathbf{W}_i^{(L)} \odot \sigma'' \left(\mathbf{z}^{(L-1)}\right)\right) \mathbf{B}^{(L-1)} \end{aligned} \tag{5.31}$$

$\mathbf{B}^{(I)}$ where $I \in [L]$ is a matrix of size $N_I \times D$ defined as follows:

$$\mathbf{B}^{(I)} = \left[\nabla_{\mathbf{x}} \mathbf{z}_1^{(I)}, \nabla_{\mathbf{x}} \mathbf{z}_2^{(I)}, \ldots, \nabla_{\mathbf{x}} \mathbf{z}_{N_I}^{(I)}\right]^T, \qquad I \in [L]$$

Thus $\mathbf{B}^{(I)}$ is the jacobian of $\mathbf{z}^{(I)}$ with respect to the input $\mathbf{x}$.

Using the chain rule of the gradient, we have the following properties of $\mathbf{B}^{(I)}$:

$$\mathbf{B}^{(1)} = \mathbf{W}^{(1)} \tag{5.32}$$

$$\mathbf{B}^{(I)} = \mathbf{W}^{(I)} diag \left(\sigma' \left(\mathbf{z}^{(I-1)}\right)\right) \mathbf{B}^{(I-1)} \tag{5.33}$$

Similarly, $\mathbf{F}^{(I,J)}$ where $I \in [L],\ J \in [I-1]$ is a matrix of size $N_I \times N_J$ defined as follows:

$$\mathbf{F}^{(I,J)} = \left[\nabla_{\mathbf{a}^{(J)}}\mathbf{z}_1^{(I)}, \nabla_{\mathbf{a}^{(J)}}\mathbf{z}_2^{(I)}, \ldots, \nabla_{\mathbf{a}^{(J)}}\mathbf{z}_{N_I}^{(I)}\right]^T$$

Thus $\mathbf{F}^{(I,J)}$ is the jacobian of $\mathbf{z}^{(I)}$ with respect to the activations $\mathbf{a}^{(J)}$.

Using the chain rule of the gradient, we have the following properties for $\mathbf{F}^{(L,I)}$:

$$\mathbf{F}^{(L,L-1)} = \mathbf{W}^{(L)} \tag{5.34}$$

$$\mathbf{F}^{(L,I)} = \mathbf{W}^{(L)} diag\left(\sigma'\left(\mathbf{z}^{(L-1)}\right)\right)\mathbf{F}^{(L-1,I)} \tag{5.35}$$

Recall that in our notation: For a matrix $\mathbf{E}$, $\mathbf{E}_i$ denotes the column vector constructed by taking the transpose of the ith row of the matrix $\mathbf{E}$. Thus i^{th} row of $\mathbf{W}^{(L)}$ is $\left(\mathbf{W}_i^{(L)}\right)^T$ and $\mathbf{F}^{(L,I)}$ is $\left(\mathbf{F}_i^{(L,I)}\right)^T$. Equating the ith rows in equation (5.35), we get:

$$\left(\mathbf{F}_i^{(L,I)}\right)^T = \left(\mathbf{W}_i^{(L)}\right)^T diag\left(\sigma'\left(\mathbf{z}^{(L-1)}\right)\right)\mathbf{F}^{(L-1,I)}$$

Taking the transpose of both the sides and expressing the RHS as a summation, we get:

$$\mathbf{F}_i^{(L,I)} = \left(\left(\mathbf{W}_i^{(L)}\right)^T diag\left(\sigma'\left(\mathbf{z}^{(L-1)}\right)\right)\mathbf{F}^{(L-1,I)}\right)^T$$

$$\mathbf{F}_i^{(L,I)} = \sum_{j=1}^{N_{L-1}} \mathbf{W}_{i,j}^{(L)}\sigma'\left(\mathbf{z}_j^{(L-1)}\right)\mathbf{F}_j^{(L-1,I)} \tag{5.36}$$

Substituting $\mathbf{W}^{(L)}$ using equation (5.34) into equation (5.31), we get:

$$\mathbf{A}_i^{(L)} = \left(\mathbf{B}^{(L-1)}\right)^T diag\left(\mathbf{F}_i^{(L,L-1)} \odot \sigma''\left(\mathbf{z}^{(L-1)}\right)\right)\mathbf{B}^{(L-1)} \tag{5.37}$$

Substituting $\mathbf{A}_i^{(L)}$ using equation (5.37) into (5.30), we get:

$$\begin{aligned}
&\nabla_{\mathbf{x}}^2 \mathbf{z}_i^{(L)} = \\
&\Bigg[\left(\mathbf{B}^{(L-1)}\right)^T diag\left(\mathbf{F}_i^{(L,L-1)} \odot \sigma''\left(\mathbf{z}^{(L-1)}\right)\right)\mathbf{B}^{(L-1)} \\
&+ \sum_{j=1}^{N_{L-1}} \mathbf{W}_{i,j}^{(L)} \sigma'\left(\mathbf{z}_j^{(L-1)}\right)\left(\nabla_{\mathbf{x}}^2 \mathbf{z}_j^{(L-1)}\right)\Bigg]
\end{aligned} \tag{5.38}$$

Thus, equation (5.38) allows us to write the hessian of ith unit at layer L, i.e $\left(\nabla_{\mathbf{x}}^2 \mathbf{z}_i^{(L)}\right)$ in terms of the hessian of jth unit at layer $L-1$, i.e $\left(\nabla_{\mathbf{x}}^2 \mathbf{z}_j^{(L-1)}\right)$.

We will prove the following using induction:

$$\nabla_{\mathbf{x}}^2 \mathbf{z}_i^{(L)} = \sum_{I=1}^{L-1} \left(\mathbf{B}^{(I)}\right)^T diag\left(\mathbf{F}_i^{(L,I)} \odot \sigma''\left(\mathbf{z}^{(I)}\right)\right)\mathbf{B}^{(I)} \tag{5.39}$$

Note that for $L = 2, \nabla_{\mathbf{x}}^2 \mathbf{z}_j^{(L-1)} = 0,\ \forall j \in N_1$. Thus using (5.38) we have:

$$\nabla_{\mathbf{x}}^2 \mathbf{z}_i^{(2)} = \left(\mathbf{B}^{(1)}\right)^T diag\left(\mathbf{F}_i^{(2,1)} \odot \sigma''\left(\mathbf{z}^{(1)}\right)\right)\mathbf{B}^{(1)}$$

Hence the induction hypothesis (5.39) is true for $L = 2$.

Now we will assume (5.39) is true for $L-1$. Thus we have:

$$\begin{aligned}
&\nabla^2_{\mathbf{x}} \mathbf{z}_j^{(L-1)} \\
&= \sum_{I=1}^{L-2} \left(\mathbf{B}^{(I)}\right)^T diag\left(\mathbf{F}_j^{(L-1,I)} \odot \sigma''\left(\mathbf{z}^{(I)}\right)\right) \mathbf{B}^{(I)} \\
&\forall j \in N_{L-1}
\end{aligned} \tag{5.40}$$

We will prove the same for L.

Using equation (5.38), we have:

$$\begin{aligned}
&\nabla^2_{\mathbf{x}} \mathbf{z}_i^{(L)} \\
&= \left(\mathbf{B}^{(L-1)}\right)^T diag\left(\mathbf{F}_i^{(L,L-1)} \odot \sigma''\left(\mathbf{z}^{(L-1)}\right)\right) \mathbf{B}^{(L-1)} \\
&+ \sum_{j=1}^{N_{L-1}} \mathbf{W}_{i,j}^{(L)} \sigma'\left(\mathbf{z}_j^{(L-1)}\right)\left(\nabla^2_{\mathbf{x}} \mathbf{z}_j^{(L-1)}\right)
\end{aligned}$$

In the next set of steps, we will be working with the second term of the above equation, i.e: $\sum_{j=1}^{N_{L-1}} \mathbf{W}_{i,j}^{(L)} \sigma'(\mathbf{z}_j^{(L-1)})(\nabla^2_{\mathbf{x}} \mathbf{z}_j^{(L-1)})$

Substituting $\nabla^2_{\mathbf{x}} \mathbf{z}_j^{(L-1)}$ using equation (5.40) we get:

$$\begin{aligned}
&\nabla^2_{\mathbf{x}} \mathbf{z}_i^{(L)} \\
&= \left(\mathbf{B}^{(L-1)}\right)^T diag\left(\mathbf{F}_i^{(L,L-1)} \odot \sigma''\left(\mathbf{z}^{(L-1)}\right)\right) \mathbf{B}^{(L-1)} \\
&+ \sum_{j=1}^{N_{L-1}} \mathbf{W}_{i,j}^{(L)} \sigma'\left(\mathbf{z}_j^{(L-1)}\right) \Bigg[\\
&\sum_{I=1}^{L-2} \left(\mathbf{B}^{(I)}\right) diag\left(\mathbf{F}_j^{(L-1,I)} \odot \sigma''\left(\mathbf{z}^{(I)}\right)\right) \left(\mathbf{B}^{(I)}\right)^T \Bigg]
\end{aligned} \tag{5.41}$$

Combining the two summations in the second term, we get:

$$
\begin{aligned}
&\nabla_{\mathbf{x}}^2 \mathbf{z}_i^{(L)} \\
&= \left(\mathbf{B}^{(L-1)}\right)^T diag\left(\mathbf{F}_i^{(L,L-1)} \odot \sigma^{''}\left(\mathbf{z}^{(L-1)}\right)\right)\mathbf{B}^{(L-1)} \\
&+ \sum_{j=1}^{N_{L-1}} \sum_{I=1}^{L-2} \Bigg[\mathbf{W}_{i,j}^{(L)} \sigma^{'}\left(\mathbf{z}_j^{(L-1)}\right) \\
&\left(\mathbf{B}^{(I)}\right)^T diag\left(\mathbf{F}_j^{(L-1,I)} \odot \sigma^{''}\left(\mathbf{z}^{(I)}\right)\right)\mathbf{B}^{(I)} \Bigg]
\end{aligned}
$$

Exchanging the summation over I and summation over j:

$$
\begin{aligned}
&\nabla_{\mathbf{x}}^2 \mathbf{z}_i^{(L)} \\
&= \left(\mathbf{B}^{(L-1)}\right)^T diag\left(\mathbf{F}_i^{(L,L-1)} \odot \sigma^{''}\left(\mathbf{z}^{(L-1)}\right)\right)\mathbf{B}^{(L-1)} \\
&+ \sum_{I=1}^{L-2} \sum_{j=1}^{N_{L-1}} \mathbf{W}_{i,j}^{(L)} \sigma^{'}\left(\mathbf{z}_j^{(L-1)}\right) \Bigg[\\
&\left(\mathbf{B}^{(I)}\right)^T diag\left(\mathbf{F}_j^{(L-1,I)} \odot \sigma^{''}\left(\mathbf{z}^{(I)}\right)\right)\mathbf{B}^{(I)} \Bigg]
\end{aligned}
$$

Since $\mathbf{B}^{(I)}$ is independent of j, we take it out of the summation over j:

$$
\begin{aligned}
&\nabla_{\mathbf{x}}^2 \mathbf{z}_i^{(L)} \\
&= \left(\mathbf{B}^{(L-1)}\right)^T diag\left(\mathbf{F}_i^{(L,L-1)} \odot \sigma^{''}\left(\mathbf{z}^{(L-1)}\right)\right)\mathbf{B}^{(L-1)} \\
&+ \sum_{I=1}^{L-2} \left(\mathbf{B}^{(I)}\right)^T \Bigg[\\
&\sum_{j=1}^{N_{L-1}} \mathbf{W}_{i,j}^{(L)} \sigma^{'}\left(\mathbf{z}_j^{(L-1)}\right) diag\left(\mathbf{F}_j^{(L-1,I)} \odot \sigma^{''}\left(\mathbf{z}^{(I)}\right)\right) \Bigg] \mathbf{B}^{(I)}
\end{aligned}
$$

Using the property, $\alpha\left(diag(\mathbf{u})\right)+\beta\left(diag(\mathbf{v})\right)=diag\left(\alpha\mathbf{u}+\beta\mathbf{v}\right)\ \forall\alpha,\beta\in\mathbb{R},\mathbf{u},\mathbf{v}\in\mathbb{R}^n$; we can move the summation inside the diagonal:

$$\begin{aligned}\nabla^2_{\mathbf{x}}\mathbf{z}_i^{(L)} &= \left(\mathbf{B}^{(L-1)}\right)^T diag\left(\mathbf{F}_i^{(L,L-1)}\odot\sigma''\left(\mathbf{z}^{(L-1)}\right)\right)\mathbf{B}^{(L-1)}\\ &+\sum_{I=1}^{L-2}\left(\mathbf{B}^{(I)}\right)^T diag\Bigg[\\ &\sum_{j=1}^{N_{L-1}}\mathbf{W}_{i,j}^{(L)}\sigma'\left(\mathbf{z}_j^{(L-1)}\right)\left(\mathbf{F}_j^{(L-1,I)}\odot\sigma''\left(\mathbf{z}^{(I)}\right)\right)\Bigg]\mathbf{B}^{(I)}\end{aligned}$$

Since $\sigma''\left(\mathbf{z}^{(I)}\right)$ is independent of j, we can take it out of the summation over j:

$$\begin{aligned}\nabla^2_{\mathbf{x}}\mathbf{z}_i^{(L)} &= \left(\mathbf{B}^{(L-1)}\right)^T diag\left(\mathbf{F}_i^{(L,L-1)}\odot\sigma''\left(\mathbf{z}^{(L-1)}\right)\right)\mathbf{B}^{(L-1)}\\ &+\sum_{I=1}^{L-2}\left(\mathbf{B}^{(I)}\right)^T diag\Bigg[\\ &\left(\sum_{j=1}^{N_{L-1}}\mathbf{W}_{i,j}^{(L)}\sigma'\left(\mathbf{z}_j^{(L-1)}\right)\mathbf{F}_j^{(L-1,I)}\right)\odot\sigma''\left(\mathbf{z}^{(I)}\right)\Bigg]\mathbf{B}^{(I)}\end{aligned}$$

Using equation (5.36), we can replace $\sum_{j=1}^{N_{L-1}}\mathbf{W}_{i,j}^{(L)}\sigma'\left(\mathbf{z}_j^{(L-1)}\right)\mathbf{F}_j^{(L-1,I)}$ with $\mathbf{F}_i^{(L,I)}$:

$$\begin{aligned}&\nabla^2_{\mathbf{x}}\mathbf{z}_i^{(L)}\\ &=\left(\mathbf{B}^{(L-1)}\right)^T diag\left(\mathbf{F}_i^{(L,L-1)}\odot\sigma''\left(\mathbf{z}^{(L-1)}\right)\right)\mathbf{B}^{(L-1)}\\ &+\sum_{I=1}^{L-2}\left(\mathbf{B}^{(I)}\right)^T diag\left(\mathbf{F}_i^{(L,I)}\odot\sigma''\left(\mathbf{z}^{(I)}\right)\right)\mathbf{B}^{(I)}\\ &\nabla^2_{\mathbf{x}}\mathbf{z}_i^{(L)}=\sum_{I=1}^{L-1}\left(\mathbf{B}^{(I)}\right)^T diag\left(\mathbf{F}_i^{(L,I)}\odot\sigma''\left(\mathbf{z}^{(I)}\right)\right)\mathbf{B}^{(I)}\end{aligned}$$

5.B.4 Proof of Theorem 8

Using Lemma 3, we have the following formula for $\nabla^2_{\mathbf{x}}\left(\mathbf{z}^{(2)}_y - \mathbf{z}^{(2)}_t\right)$:

$$\begin{aligned}
&\nabla^2_{\mathbf{x}}\left(\mathbf{z}^{(2)}_y - \mathbf{z}^{(2)}_t\right) \\
&= \left(\mathbf{W}^{(1)}\right)^T diag\Bigg(\left(\mathbf{W}^{(2)}_y - \mathbf{W}^{(2)}_t\right) \odot \sigma^{''}\left(\mathbf{z}^{(1)}\right)\Bigg)\mathbf{W}^{(1)} \\
&= \sum_{i=1}^{N_1}\left(\mathbf{W}^{(2)}_{y,i} - \mathbf{W}^{(2)}_{t,i}\right)\sigma^{''}\left(\mathbf{z}^{(1)}_i\right)\mathbf{W}^{(1)}_i\left(\mathbf{W}^{(1)}_i\right)^T
\end{aligned} \tag{5.42}$$

We are also given that the activation function σ satisfies the following property:

$$h_L \leq \sigma^{''}(x) \leq h_U \quad \forall x \in \mathbb{R} \tag{5.43}$$

(a) We have to prove the following linear matrix inequalities (LMIs):

$$\mathbf{N} \preccurlyeq \nabla^2_{\mathbf{x}}\left(\mathbf{z}^{(2)}_y - \mathbf{z}^{(2)}_t\right) \preccurlyeq \mathbf{P} \qquad \forall \mathbf{x} \in \mathbb{R}^D \tag{5.44}$$

where $\mathbf{P}$ and $\mathbf{N}$ are given as following:

$$\mathbf{P} = \sum_{i=1}^{N_1} p_i \left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right) \mathbf{W}_i^{(1)} \left(\mathbf{W}_i^{(1)}\right)^T \tag{5.45}$$

$$\mathbf{N} = \sum_{i=1}^{N_1} n_i \left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right) \mathbf{W}_i^{(1)} \left(\mathbf{W}_i^{(1)}\right)^T \tag{5.46}$$

$$p_i = \begin{Bmatrix} h_U, & \mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)} \geq 0 \\ h_L, & \mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)} \leq 0 \end{Bmatrix},$$

$$n_i = \begin{Bmatrix} h_L, & \mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)} \geq 0 \\ h_U, & \mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)} \leq 0 \end{Bmatrix} \tag{5.47}$$

We first prove: $\mathbf{N} \preccurlyeq \nabla_{\mathbf{x}}^2 \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right) \quad \forall \mathbf{x} \in \mathbb{R}^D$:

We substitute $\nabla_{\mathbf{x}}^2 \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right)$ and $\mathbf{N}$ from equations (5.42) and (5.46) respectively in $\nabla_{\mathbf{x}}^2 \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right) - \mathbf{N}$:

$$\begin{aligned} &\nabla_{\mathbf{x}}^2 \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right) - \mathbf{N} \\ &= \sum_{i=1}^{N_1} \left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right) \left(\sigma'' \left(\mathbf{z}_i^{(1)}\right) - n_i\right) \mathbf{W}_i^{(1)} \left(\mathbf{W}_i^{(1)}\right)^T \end{aligned}$$

Thus $\nabla_{\mathbf{x}}^2 \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right) - \mathbf{N}$ is a weighted sum of symmetric rank one matrices i.e, $\mathbf{W}_i^{(1)} \left(\mathbf{W}_i^{(1)}\right)^T$ and it is PSD if and only if coefficient of each rank one matrix i.e, $\left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right) \left(\sigma'' \left(\mathbf{z}_i^{(1)}\right) - n_i\right)$ is positive. Using equations (5.43) and (5.47), we have

the following:

$$\left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right) \geq 0 \implies n_i = h_L$$
$$\implies \left(\sigma^{''}\left(\mathbf{z}_i^{(1)}\right) - n_i\right) \geq 0 \qquad \forall i \in [N_1],\ \forall \mathbf{x} \in \mathbb{R}^D$$
$$\left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right) \leq 0 \implies n_i = h_U$$
$$\implies \left(\sigma^{''}\left(\mathbf{z}_i^{(1)}\right) - n_i\right) \leq 0 \qquad \forall i \in [N_1],\ \forall \mathbf{x} \in \mathbb{R}^D$$

Putting the above results together we have:

$$\left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right)\left(\sigma^{''}\left(\mathbf{z}_i^{(1)}\right) - n_i\right) \geq 0$$
$$\forall i \in [N_1],\ \forall \mathbf{x} \in \mathbb{R}^D \tag{5.48}$$

Thus $\nabla_{\mathbf{x}}^2\left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right) - \mathbf{N}$ is a PSD matrix i.e:

$$\nabla_{\mathbf{x}}^2\left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right) - \mathbf{N}$$
$$= \sum_{i=1}^{N_1} \underbrace{\left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right)\left(\sigma^{''}\left(\mathbf{z}_i^{(1)}\right) - n_i\right)}_{\text{always positive using eq. (5.48)}} \mathbf{W}_i^{(1)}\left(\mathbf{W}_i^{(1)}\right)^T$$
$$\implies \mathbf{N} \preccurlyeq \nabla_{\mathbf{x}}^2\left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right) \qquad \forall \mathbf{x} \in \mathbb{R}^D \tag{5.49}$$

Now we prove that $\nabla_{\mathbf{x}}^2\left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right) \preccurlyeq \mathbf{P} \quad \forall \mathbf{x} \in \mathbb{R}^D$:

We substitute $\nabla_{\mathbf{x}}^2\left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right)$ and $\mathbf{P}$ from equations (5.42) and (5.46) respectively in

$\mathbf{P} - \nabla^2_{\mathbf{x}} \left(\mathbf{z}^{(2)}_y - \mathbf{z}^{(2)}_t \right)$:

$$\begin{aligned}
&\mathbf{P} - \nabla^2_{\mathbf{x}} \left(\mathbf{z}^{(2)}_y - \mathbf{z}^{(2)}_t \right) \\
&= \sum_{i=1}^{N_1} \left(\mathbf{W}^{(2)}_{y,i} - \mathbf{W}^{(2)}_{t,i} \right) \left(p_i - \sigma^{''} \left(\mathbf{z}^{(1)}_i \right) \right) \mathbf{W}^{(1)}_i (\mathbf{W}^{(1)}_i)^T
\end{aligned}$$

Thus $\mathbf{P} - \nabla^2_{\mathbf{x}} \left(\mathbf{z}^{(2)}_y - \mathbf{z}^{(2)}_t \right)$ is a weighted sum of symmetric rank one matrices i.e, $\mathbf{W}^{(1)}_i \left(\mathbf{W}^{(1)}_i \right)^T$ and it is PSD if and only if coefficient of each rank one matrix i.e, $\left(\mathbf{W}^{(2)}_{y,i} - \mathbf{W}^{(2)}_{t,i} \right) \left(p_i - \sigma^{''} \left(\mathbf{z}^{(1)}_i \right) \right)$ is positive. Using equations (5.43) and (5.47), we have the following:

$$\begin{aligned}
&\left(\mathbf{W}^{(2)}_{y,i} - \mathbf{W}^{(2)}_{t,i} \right) \geq 0 \implies p_i = h_U \\
&\implies \left(p_i - \sigma^{''} \left(\mathbf{z}^{(1)}_i \right) \right) \geq 0 \qquad \forall i \in N_1,\ \mathbf{x} \in \mathbb{R}^D \\
&\left(\mathbf{W}^{(2)}_{y,i} - \mathbf{W}^{(2)}_{t,i} \right) \leq 0 \implies p_i = h_L \\
&\implies \left(p_i - \sigma^{''} \left(\mathbf{z}^{(1)}_i \right) \right) \leq 0 \qquad \forall i \in N_1,\ \mathbf{x} \in \mathbb{R}^D
\end{aligned}$$

Putting the above results together we have:

$$\begin{aligned}
&\implies \left(\mathbf{W}^{(2)}_{y,i} - \mathbf{W}^{(2)}_{t,i} \right) \left(p_i - \sigma^{''} \left(\mathbf{z}^{(1)}_i \right) \right) \geq 0 \\
&\forall i \in [N_1],\ \mathbf{x} \in \mathbb{R}^D
\end{aligned} \tag{5.50}$$

Thus $\mathbf{P} - \nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)} \right)$ is PSD matrix i.e:

$$\begin{aligned}
&\mathbf{P} - \nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)} \right) \\
&= \sum_{i=1}^{N_1} \underbrace{\left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)} \right) \left(p_i - \sigma'' \left(\mathbf{z}_i^{(1)} \right) \right)}_{\text{always positive using eq. (5.50)}} \mathbf{W}_i^{(1)} \left(\mathbf{W}_i^{(1)} \right)^T \\
&\implies \mathbf{P} \succcurlyeq \nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)} \right) \qquad \forall \mathbf{x} \in \mathbb{R}^D
\end{aligned} \tag{5.51}$$

Thus by proving the LMIs (5.49) and (5.51), we prove (5.44).

(b) We have to prove that if $h_U \geq 0$ and $h_L \leq 0$, $\mathbf{P}$ is a PSD matrix, $\mathbf{N}$ is a NSD matrix.

We are given $h_U \geq 0$, $h_L \leq 0$. Using equation (5.47), we have the following:

$$\begin{aligned}
&\left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)} \right) \geq 0 \implies p_i = h_U \geq 0 \\
&\implies p_i \left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)} \right) \geq 0 \\
&\left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)} \right) \leq 0 \implies p_i = h_L \leq 0 \\
&\implies p_i \left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)} \right) \geq 0
\end{aligned}$$

Putting these results together we have:

$$\implies p_i \left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)} \right) \geq 0 \qquad \forall i \in [N_1] \tag{5.52}$$

Thus $\mathbf{P}$ is a weighted sum of symmetric rank one matrices i.e, $\mathbf{W}_i^{(1)} \left(\mathbf{W}_i^{(1)} \right)^T$ and each

coefficient $p_i \left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right)$ is positive.

$$\mathbf{P} = \sum_{i=1}^{N_1} \underbrace{p_i \left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right)}_{\text{always positive using eq. (5.52)}} \mathbf{W}_i^{(1)} \left(\mathbf{W}_i^{(1)}\right)^T \succcurlyeq 0$$

Using equation (5.47), we have the following:

$$\begin{aligned}
&\left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right) \geq 0 \implies n_i = h_L \leq 0 \\
&\implies n_i \left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right) \leq 0 \\
&\left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right) \leq 0 \implies n_i = h_U \geq 0 \\
&\implies n_i \left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right) \leq 0
\end{aligned}$$

Putting these results together we have:

$$\implies n_i \left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right) \geq 0 \qquad \forall i \in [N_1] \tag{5.53}$$

$$\mathbf{N} = \sum_{i=1}^{N_1} \underbrace{n_i \left(\mathbf{W}_{y,i}^{(2)} - \mathbf{W}_{t,i}^{(2)}\right)}_{\text{always positive using eq. (5.53)}} \mathbf{W}_i^{(1)} \left(\mathbf{W}_i^{(1)}\right)^T \preccurlyeq 0$$

Thus $\mathbf{P}$ is a PSD and $\mathbf{N}$ is a NSD matrix if $h_U \geq 0$ and $h_L \leq 0$.

(c) We have to prove the following global bounds on the eigenvalues of $\nabla^2_{\mathbf{x}}(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)})$:

$$m\mathbf{I} \preccurlyeq \nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right) \preccurlyeq M\mathbf{I},$$

$$\text{where } M = \max_{\|\mathbf{v}\|=1} \mathbf{v}^T\mathbf{P}\mathbf{v},\ m = \min_{\|\mathbf{v}\|=1} \mathbf{v}^T\mathbf{N}\mathbf{v}$$

Since $\nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right) \preccurlyeq \mathbf{P} \quad \forall \mathbf{x} \in \mathbb{R}^D$:

$$\mathbf{v}^T \left[\nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right)\right] \mathbf{v} \leq \mathbf{v}^T\mathbf{P}\mathbf{v}$$

$$\forall \mathbf{v} \in \mathbb{R}^D,\ \forall \mathbf{x} \in \mathbb{R}^D \tag{5.54}$$

Let $\mathbf{v}^*$, $\mathbf{x}^*$ be vectors such that:

$$(\mathbf{v}^*)^T \left[\nabla^2_{\mathbf{x}^*} \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right)\right] \mathbf{v}^*$$

$$= \max_{\mathbf{x}} \max_{\|\mathbf{v}\|=1} \mathbf{v}^T \left[\nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right)\right] \mathbf{v}$$

Thus using inequality (5.54):

$$(\mathbf{v}^*)^T \left[\nabla^2_{\mathbf{x}^*} \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right)\right] \mathbf{v}^* \leq \max_{\|\mathbf{v}\|=1} \mathbf{v}^T\mathbf{P}\mathbf{v} \tag{5.55}$$

Since $\mathbf{N} \preccurlyeq \nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right) \quad \forall \mathbf{x} \in \mathbb{R}^D$:

$$\mathbf{v}^T\mathbf{N}\mathbf{v} \leq \mathbf{v}^T \left[\nabla^2_{\mathbf{x}} \left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right)\right] \mathbf{v}$$

$$\forall \mathbf{v} \in \mathbb{R}^D,\ \forall \mathbf{x} \in \mathbb{R}^D \tag{5.56}$$

Let $\mathbf{v}^*$, $\mathbf{x}^*$ be vectors such that:

$$
\begin{aligned}
&(\mathbf{v}^*)^T \left[\nabla^2_{\mathbf{x}^*}\left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right)\right]\mathbf{v}^* \\
&= \min_{\mathbf{x}} \min_{\|\mathbf{v}\|=1} \mathbf{v}^T \left[\nabla^2_{\mathbf{x}}\left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right)\right]\mathbf{v}
\end{aligned}
$$

Thus using inequality (5.56):

$$
(\mathbf{v}^*)^T \left[\nabla^2_{\mathbf{x}^*}\left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right)\right]\mathbf{v}^* \geq \min_{\|\mathbf{v}\|=1} \mathbf{v}^T\mathbf{N}\mathbf{v} \tag{5.57}
$$

Using the inequalities (5.55) and (5.57), we get:

$$
m\mathbf{I} \preccurlyeq \nabla^2_{\mathbf{x}}\left(\mathbf{z}_y^{(2)} - \mathbf{z}_t^{(2)}\right) \preccurlyeq M\mathbf{I}
$$

where $M = \max_{\|\mathbf{v}\|=1} \mathbf{v}^T\mathbf{P}\mathbf{v},\ m = \min_{\|\mathbf{v}\|=1} \mathbf{v}^T\mathbf{N}\mathbf{v}$

5.B.5 Proof of Theorem 9

We are given that the activation function σ is such that $\sigma^{'},\ \sigma^{''}$ are bounded, i.e:

$$
|\sigma^{'}(x)| \leq g,\ |\sigma^{''}(x)| \leq h \qquad \forall x \in \mathbb{R} \tag{5.58}
$$

We have to prove the following:

$$
\left\|\nabla^2_{\mathbf{x}}\mathbf{z}_i^{(L)}\right\| \leq h\sum_{I=1}^{L-1}\left(r^{(I)}\right)^2 \max_j\left(\mathbf{S}_{i,j}^{(I)}\right) \quad \forall \mathbf{x} \in \mathbb{R}^D
$$

where $\mathbf{S}^{(L,I)}$ is a matrix of size $N_L \times N_I$ defined as follows:

$$\mathbf{S}^{(L,I)} = \begin{cases} \left|\mathbf{W}^{(L)}\right| & I = L-1 \\ g\left|\mathbf{W}^{(L)}\right| \mathbf{S}^{(L-1,I)} & I \in [L-2] \end{cases} \tag{5.59}$$

and $r^{(I)}$ is a scalar defined as follows:

$$r^{(I)} = \begin{cases} \left\|\mathbf{W}^{(1)}\right\| & I = 1 \\ g\left\|\mathbf{W}^{(I)}\right\| r^{(I-1)} & I \in [2, L-1] \end{cases} \tag{5.60}$$

We will prove the same in 3 steps.

In step (a), we will prove:

$$\left|\mathbf{F}_{i,j}^{(L,I)}\right| \leq \mathbf{S}_{i,j}^{(L,I)} \qquad \forall \mathbf{x} \in \mathbb{R}^D \tag{5.61}$$

In step (b), we will prove:

$$\left\|\mathbf{B}^{(I)}\right\| \leq r^{(I)}, \qquad \forall \mathbf{x} \in \mathbb{R}^D \tag{5.62}$$

In step (c), we will use (a) and (b) to prove:

$$\left\|\nabla_{\mathbf{x}}^2 \mathbf{z}_i^{(L)}\right\| \leq h \sum_{I=1}^{L-1} \left(r^{(I)}\right)^2 \max_j \left(\mathbf{S}_{i,j}^{(L,I)}\right) \tag{5.63}$$

Note that $\mathbf{B}^{(I)}$ and $\mathbf{F}^{(L,I)}$ are defined using (5.22) and (5.23) respectively.

(a) We have to prove that for $L \geq 2,\ I \in [L-1],\ i \in N_L,\ j \in N_I$:

$$\left|\mathbf{F}_{i,j}^{(L,I)}\right| \leq \mathbf{S}_{i,j}^{(L,I)} \qquad \forall \mathbf{x} \in \mathbb{R}^D$$

where $\mathbf{S}^{(L,I)}$ is a matrix of size $N_I \times N_J$ defined as follows:

$$\mathbf{S}^{(L,I)} = \begin{cases} \left|\mathbf{W}^{(L)}\right| & I = L-1 \\ g\left|\mathbf{W}^{(L)}\right| \mathbf{S}^{(L-1,J)} & I \in [L-2] \end{cases}$$

We first prove the case when $I = L-1$.

Using equation (5.34), $\mathbf{F}_{i,j}^{(L,L-1)} = \mathbf{W}_{i,j}^{(L)}$.

Since $\mathbf{S}_{i,j}^{(L,L-1)} = \left|\mathbf{W}_{i,j}^{(L)}\right|$:

$$\left|\mathbf{F}_{i,j}^{(L,L-1)}\right| = \mathbf{S}_{i,j}^{(L,L-1)}$$

Hence for $L \geq 2,\ I = L-1$, we have equality in (5.61). Hence proved.

Now, we will use proof by induction.

To prove the base case $L = 2$, note that $I = L - 1 = 1$ is the only possible value for I. Thus, using the result for $I = L-1$, the theorem holds for $L = 2$. This proves the base case.

Now we assume the induction hypothesis is true for depth $= L-1,\ I \in [L-2]$. and prove for depth $= L,\ I \in [L-1]$. Since for $I = L-1$, we have proven already, we prove for $I \leq L-2$.

Using equation (5.36), we have the following formula for $\mathbf{F}_i^{(L,I)}$:

$$\mathbf{F}_i^{(L,I)} = \sum_{k=1}^{N_{L-1}} \mathbf{W}_{i,k}^{(L)} \sigma^{'} \left(\mathbf{z}_k^{(L-1)} \right) \mathbf{F}_k^{(L-1,I)}$$

Taking the jth element of the vectors on both sides:

$$\mathbf{F}_{i,j}^{(L,I)} = \sum_{k=1}^{N_{L-1}} \mathbf{W}_{i,k}^{(L)} \sigma^{'} \left(\mathbf{z}_k^{(L-1)} \right) \mathbf{F}_{k,j}^{(L-1,I)} \tag{5.64}$$

By induction hypothesis, we know that:

$$\left| \mathbf{F}_{k,j}^{(L-1,I)} \right| \leq \mathbf{S}_{k,j}^{(L-1,I)} \tag{5.65}$$

Using the absolute value properties for equation (5.64), we have:

$$\left| \mathbf{F}_{i,j}^{(L,I)} \right| = \left| \sum_{k=1}^{N_{L-1}} \mathbf{W}_{i,k}^{(L)} \sigma^{'} \left(\mathbf{z}_k^{(L-1)} \right) \mathbf{F}_{k,j}^{(L-1,I)} \right|$$

$$\left| \mathbf{F}_{i,j}^{(L,I)} \right| \leq \sum_{k=1}^{N_{L-1}} \left| \mathbf{W}_{i,k}^{(L)} \sigma^{'} \left(\mathbf{z}_k^{(L-1)} \right) \mathbf{F}_{k,j}^{(L-1,I)} \right|$$

$$\left| \mathbf{F}_{i,j}^{(L,I)} \right| \leq \sum_{k=1}^{N_{L-1}} \left| \mathbf{W}_{i,k}^{(L)} \right| \left| \sigma^{'} \left(\mathbf{z}_k^{(L-1)} \right) \right| \left| \mathbf{F}_{k,j}^{(L-1,I)} \right|$$

Using $|\sigma^{'}(x)| \leq g \quad \forall x \in \mathbb{R}$ (inequality (5.58)) :

$$\left| \mathbf{F}_{i,j}^{(L,I)} \right| \leq g \sum_{k=1}^{N_{L-1}} \left| \mathbf{W}_{i,k}^{(L)} \right| \left| \mathbf{F}_{k,j}^{(L-1,I)} \right|$$

Using the induction hypothesis (inequality (5.65)):

$$\left|\mathbf{F}_{i,j}^{(L,I)}\right| \leq g \sum_{k=1}^{N_{L-1}} \left|\mathbf{W}_{i,k}^{(L)}\right| \left|\mathbf{S}_{k,j}^{(L-1,I)}\right|$$

Using equation (5.59) for definition of $\mathbf{S}_{i,j}^{(L,I)}$:

$$\left|\mathbf{F}_{i,j}^{(L,I)}\right| \leq \mathbf{S}_{i,j}^{(L,I)}$$

Hence we prove (5.61) for all $L \geq 2$ and $I \leq L-1$ using induction.

(b) We have to prove that for $1 \leq I \leq M-1$:

$$\left\|\mathbf{B}^{(I)}\right\| \leq r^{(I)}, \qquad \forall \mathbf{x} \in \mathbb{R}^D$$

where $r^{(I)}$ is a scalar given as follows:

$$r^{(I)} = \begin{cases} \left\|\mathbf{W}^{(1)}\right\| & I = 1 \\ g \left\|\mathbf{W}^{(I)}\right\| r^{(I-1)} & I \in [2, L-1] \end{cases}$$

Using equation (5.32), for $I = 1$ we have:

$$\left\|\mathbf{B}^{(1)}\right\| = \left\|\mathbf{W}^{(1)}\right\| = r^{(1)} \tag{5.66}$$

Using equation (5.33), for $I > 1$, we have:

$$\left\|\mathbf{B}^{(I)}\right\| = \left\|\mathbf{W}^{(I)} diag\left(\sigma'\left(\mathbf{z}^{(I-1)}\right)\right)\mathbf{B}^{(I-1)}\right\|$$

$$\left\|\mathbf{B}^{(I)}\right\| \leq \left\|\mathbf{W}^{(I)}\right\| \left\|diag\left(\sigma'\left(\mathbf{z}^{(I-1)}\right)\right)\right\| \left\|\mathbf{B}^{(I-1)}\right\|$$

Since $\left\|diag\left(\sigma'\left(\mathbf{z}^{(I-1)}\right)\right)\right\| = \max_j \left|\sigma'\left(\mathbf{z}_j^{(I-1)}\right)\right|$, using equation (5.58):

$$\left\|\mathbf{B}^{(I)}\right\| \leq g\left\|\mathbf{W}^{(I)}\right\| \left\|\mathbf{B}^{(I-1)}\right\| \leq g\left\|\mathbf{W}^{(I)}\right\| r^{(I-1)} \tag{5.67}$$

Using inequalities (5.66) and (5.67), the proof follows using induction.

(c) We have to prove that:

$$\left\|\nabla^2_{\mathbf{x}} \mathbf{z}_i^{(L)}\right\| \leq h \sum_{I=1}^{L-1} \left(r^{(I)}\right)^2 \max_j \left(\mathbf{S}_{i,j}^{(I)}\right)$$

Using Lemma 3, we have the following equation for $\nabla^2_{\mathbf{x}} \mathbf{z}_i^{(L)}$:

$$\nabla^2_{\mathbf{x}} \mathbf{z}_i^{(L)} = \sum_{I=1}^{L-1} \left(\mathbf{B}^{(I)}\right)^T diag\left(\mathbf{F}_i^{(L,I)} \odot \sigma''\left(\mathbf{z}^{(I)}\right)\right)\mathbf{B}^{(I)}$$

Using the properties of norm we have:

$$\begin{aligned}
&\left\|\nabla_{\mathbf{x}}^2 \mathbf{z}_i^{(L)}\right\| \\
&= \left\|\sum_{I=1}^{L-1} \left(\mathbf{B}^{(I)}\right)^T diag\left(\mathbf{F}_i^{(L,I)} \odot \sigma^{''}\left(\mathbf{z}^{(I)}\right)\right) \mathbf{B}^{(I)}\right\| \\
&\leq \sum_{I=1}^{L-1} \left\|diag\left(\mathbf{F}_i^{(L,I)} \odot \sigma^{''}\left(\mathbf{z}^{(I)}\right)\right)\right\| \left\|\mathbf{B}^{(I)}\right\|^2 \\
&\leq \sum_{I=1}^{L-1} \max_j \left(\left|\mathbf{F}_{i,j}^{(L,I)} \sigma^{''}\left(\mathbf{z}_j^{(I)}\right)\right| \right) \left\|\mathbf{B}^{(I)}\right\|^2
\end{aligned}$$

In the last inequality, we use the property that norm of a diagonal matrix is the maximum absolute value of the diagonal element. Using the product property of absolute value, we get:

$$\left\|\nabla_{\mathbf{x}}^2 \mathbf{z}_i^{(L)}\right\| \leq \sum_{I=1}^{L-1} \max_j \left(\left|\mathbf{F}_{i,j}^{(L,I)}\right| \left|\sigma^{''}\left(\mathbf{z}_j^{(I)}\right)\right| \right) \left\|\mathbf{B}^{(I)}\right\|^2$$

Since $\left|\mathbf{F}_{i,j}^{(L,I)}\right|$ and $\left|\sigma^{''}\left(\mathbf{z}_j^{(I)}\right)\right|$ are positive terms:

$$\begin{aligned}
&\left\|\nabla_{\mathbf{x}}^2 \mathbf{z}_i^{(L)}\right\| \\
&\leq \sum_{I=1}^{L-1} \max_j \left(\left|\mathbf{F}_{i,j}^{(L,I)}\right| \right) \max_j \left(\left|\sigma^{''}\left(\mathbf{z}_j^{(I)}\right)\right| \right) \left\|\mathbf{B}^{(I)}\right\|^2
\end{aligned}$$

Since $\left\|\sigma^{''}\right\|$ is bounded by h:

$$\left\|\nabla_{\mathbf{x}}^2 \mathbf{z}_i^{(L)}\right\| \leq h \sum_{I=1}^{L-1} \max_j \left(\left|\mathbf{F}_{i,j}^{(L,I)}\right| \right) \left\|\mathbf{B}^{(I)}\right\|^2$$

Using inequality (5.61):

$$\left\|\nabla^2_{\mathbf{x}}\mathbf{z}_i^{(L)}\right\| \leq h\sum_{I=1}^{L-1}\max_j\left(\mathbf{S}_{i,j}^{(I)}\right)\left\|\mathbf{B}^{(I)}\right\|^2$$

Using inequality (5.62):

$$\left\|\nabla^2_{\mathbf{x}}\mathbf{z}_i^{(L)}\right\| \leq h\sum_{I=1}^{L-1}\left(r^{(I)}\right)^2\max_j\left(\mathbf{S}_{i,j}^{(I)}\right) \quad \forall\mathbf{x}\in\mathbb{R}^D$$

5.B.6 Proof of Theorem 10

Theorem 10. *For a binary classifier f, let g denote the indicator function such that $g(\mathbf{x}) = 1 \iff f(\mathbf{x}) > 0,\ g(\mathbf{x}) = 0$ otherwise. Let $\hat{g}$ be the function constructed by applying randomized smoothing on g such that:*

$$\hat{g}\left(\mathbf{u}\right) = \frac{1}{(2\pi s^2)^{n/2}}\int_{\mathbb{R}^D} g(\mathbf{v})\exp\left(-\frac{\|\mathbf{v}-\mathbf{u}\|^2}{2s^2}\right)d\mathbf{v}$$

then the curvature of the resulting function $\hat{g}$ is bounded i.e:

$$-\frac{\mathbf{I}}{s^2} \preccurlyeq \nabla^2_{\mathbf{u}}\,\hat{g} \preccurlyeq \frac{\mathbf{I}}{s^2}$$

Proof.

$$\nabla_{\mathbf{u}} \hat{g}(\mathbf{u})$$
$$= \frac{1}{(2\pi s^2)^{n/2}} \int_{\mathbb{R}^D} g(\mathbf{v}) \frac{(\mathbf{v}-\mathbf{u})}{s^2} \exp\left(-\frac{\|\mathbf{v}-\mathbf{u}\|^2}{2s^2}\right) d\mathbf{v}$$

$$\nabla^2_{\mathbf{u}} \hat{g}(\mathbf{u})$$
$$= \frac{1}{(2\pi s^2)^{n/2}} \int_{\mathbb{R}^D} g(\mathbf{v}) \frac{-\mathbf{I}}{s^2} \exp\left(-\frac{\|\mathbf{v}-\mathbf{u}\|^2}{2s^2}\right) d\mathbf{v}$$
$$+ \frac{1}{(2\pi s^2)^{n/2}} \int_{\mathbb{R}^D} g(\mathbf{v}) \frac{(\mathbf{v}-\mathbf{u})(\mathbf{v}-\mathbf{u})^T}{s^4} \Bigg[$$
$$\exp\left(-\frac{\|\mathbf{v}-\mathbf{u}\|^2}{2s^2}\right) \Bigg] d\mathbf{v}$$

Since $0 \leq g(\mathbf{v}) \leq 1$, $-\mathbf{I}/s^2 \preccurlyeq 0$, $(\mathbf{v}-\mathbf{u})(\mathbf{v}-\mathbf{u})^T \succcurlyeq 0$ and $\exp(x) \geq 0 \; \forall x$:

$$\nabla^2_{\mathbf{u}} \hat{g}(\mathbf{u}) = \frac{1}{(2\pi s^2)^{n/2}} \int_{\mathbb{R}^D} \underbrace{g(\mathbf{v}) \frac{-\mathbf{I}}{s^2} \exp\left(-\frac{\|\mathbf{v}-\mathbf{u}\|^2}{2s^2}\right) d\mathbf{v}}_{\text{Negative Semi-Definite}}$$
$$+ \frac{1}{(2\pi s^2)^{n/2}} \int_{\mathbb{R}^D} \underbrace{g(\mathbf{v}) \frac{(\mathbf{v}-\mathbf{u})(\mathbf{v}-\mathbf{u})^T}{s^4}}_{\text{Positive Semi-Definite}} \Bigg[$$
$$\exp\left(-\frac{\|\mathbf{v}-\mathbf{u}\|^2}{2s^2}\right) \Bigg] d\mathbf{v}$$

$$\nabla^2_{\mathbf{u}}\,\hat{g}\,(\mathbf{u}) \preccurlyeq \frac{1}{(2\pi s^2)^{n/2}} \int_{\mathbb{R}^D} \frac{(\mathbf{v}-\mathbf{u})(\mathbf{v}-\mathbf{u})^T}{s^4} \Bigg[\exp\left(-\frac{\|\mathbf{v}-\mathbf{u}\|^2}{2s^2}\right) \Bigg] d\mathbf{v}$$

$$\nabla^2_{\mathbf{u}}\,\hat{g}\,(\mathbf{u}) \preccurlyeq \frac{1}{(2\pi s^2)^{n/2}} \int_{\mathbb{R}^D} \frac{\mathbf{q}\mathbf{q}^T}{s^4} \exp\left(-\frac{\|\mathbf{q}\|^2}{2s^2}\right) d\mathbf{q}$$

$$\nabla^2_{\mathbf{u}}\,\hat{g}\,(\mathbf{u}) \preccurlyeq \frac{\mathbf{I}}{s^2}$$

$$\nabla^2_{\mathbf{u}}\,\hat{g}\,(\mathbf{u}) \succcurlyeq \frac{1}{(2\pi s^2)^{n/2}} \int_{\mathbb{R}^D} \frac{-\mathbf{I}}{s^2} \exp\left(-\frac{\|\mathbf{v}-\mathbf{u}\|^2}{2s^2}\right) d\mathbf{v}$$

$$\nabla^2_{\mathbf{u}}\,\hat{g}\,(\mathbf{u}) \succcurlyeq -\frac{\mathbf{I}}{s^2}$$

□

5.C Additional experiments

Empirical accuracy means the fraction of test samples that were correctly classified after running a PGD attack [20] with an l_2 bound on the adversarial perturbations. Certified accuracy means the fraction of test samples that were classified correctly initially and had the robustness certificate greater than a pre-specified attack radius ρ. Unless otherwise specified, for both empirical and certified accuracy, we use $\rho = 0.5$. Unless otherwise specified, we use the class with the second largest logit as the attack target for the given input (i.e. the class t). Unless specified, the experiments were run on the MNIST dataset while noting that our results are scalable for more complex datasets. The notation ($L \times [1024]$, activation) denotes a neural network with L layers with the specified activation function, ($\gamma = c$) denotes standard training with γ set to c, (CRT, c) denotes CRT training with $\gamma = c$. Certificates CROWN and CRC are computed over 150 correctly classified images.

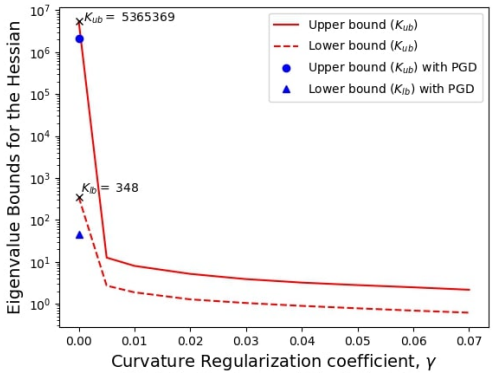


Figure 5.2: Effect of γ on K_{ub} and K_{lb} for a 4 layer network. We observe a similar trend as in 2 and 3 layer networks (Figure 5.1). At $\gamma = 0$, we observe $K_{ub} \approx 15418 \times K_{lb}$.

5.C.1 Computing K_{lb} and K_{ub}

First, note that K does not depend on the input, but on network weights $\mathbf{W}^{(I)}$, label y and target t. Different images may still have different K because label y and target t may be different.

To compute K_{lb} in the table, first for each pair y and t, we find the largest eigenvalue of the Hessian of all test images that have label y and second largest logit of class t. Then we take the max of the largest eigenvalue across all test images. This gives a rough estimate of the largest curvature in the vicinity of test images with label y and target t. We can directly take the mean across all such pairs to compute K_{lb}. However, we find that some pairs y and t were infrequent (with barely 1,2 test images in them). Thus, for all such pairs we cannot get a good estimate of the largest curvature in vicinity. We select all pairs y and t that have at least 100 images in them and compute K_{lb} by taking the mean across all such pairs.

To compute K_{ub} in the table, we compute K for all pairs y and t that have at least 100 images, i.e at least 100 images should have label y and target t. And then we compute the mean across all K that satisfy this condition. This was done to do a fair comparison with K_{lb}. Figure

Network	Training	Standard Accuracy	Certified Accuracy
$2 \times [1024]$, softplus	**CRT, 0.01**	**98.69%**	**95.5%**
	CROWN-IBP	98.72%	89.31%
$2 \times [1024]$, relu	CROWN-IBP	98.69%	91.38%
	COAP	98.8%	90.2%
$3 \times [1024]$, softplus	**CRT, 0.01**	**98.56%**	**94.44%**
	CROWN-IBP	98.55%	88.67%
$3 \times [1024]$, relu	CROWN-IBP	98.9%	90.67%
	COAP	98.9%	89.0%
$4 \times [1024]$, softplus	**CRT, 0.01**	**98.43%**	**93.35%**
	CROWN-IBP	98.34%	87.41%
$4 \times [1024]$, relu	CROWN-IBP	98.78%	90.45%
	COAP	98.9%	89.0%

Table 5.7: Comparison with interval-bound propagation based adversarial training methods with attack radius $\rho = 0.5$ on MNIST dataset. Note that the certified accuracy of softplus network with CROWN-IBP is significantly less than that of a similar ReLU network.

5.2 shows a plot of the K_{ub} and K_{lb} with increasing γ for a sigmoid network (with 4 layers).

5.C.2 Comparison with provable defenses

In this section, we compare Curvature-based Robust Training (Ours) against state-of-the-art interval-bound propagation based adversarial training methods: COAP i.e Convex Outer Adversarial Polytope [145] and CROWN-IBP [167] with different attack radius on MNIST and Fashion-MNIST datasets. For CROWN-IBP, we vary the final_beta parameter between 0.5 to 3 (using an interval of 0.1) and choose the model with best certified accuracy.

Network	Training	Standard Accuracy	Certified Robust Accuracy
$2 \times [1024]$, softplus	**CRT, 0.01**	88.45%	**78.45%**
$2 \times [1024]$, relu	COAP	86.0%	74.0%
	CROWN-IBP	85.89%	74.62%
$3 \times [1024]$, softplus	**CRT, 0.01**	86.21%	**76.94%**
$3 \times [1024]$, relu	COAP	85.9%	74.3%
	CROWN-IBP	86.27%	74.56%
$4 \times [1024]$, softplus	**CRT, 0.01**	86.37%	**75.02%**
$4 \times [1024]$, relu	COAP	85.9%	74.2%
	CROWN-IBP	86.03%	74.38%

Table 5.8: Comparison with interval-bound propagation based adversarial training methods with attack radius $\rho = 0.5$ on Fashion-MNIST dataset.

Network	Training	Standard Accuracy	Certified Robust Accuracy
$2 \times [1024]$, softplus	**CRT, 0.01**	**98.68%**	**69.79%**
	CROWN-IBP	88.48%	42.36%
$2 \times [1024]$, relu	COAP	89.33%	44.29%
	CROWN-IBP	89.49%	44.96%
$3 \times [1024]$, softplus	CRT, 0.01	98.26%	14.21%
	CRT, 0.03	97.82%	50.72%
	CRT, 0.05	**97.43%**	**57.78%**
	CROWN-IBP	86.58%	42.14%
$3 \times [1024]$, relu	COAP	89.12%	44.21%
	CROWN-IBP	87.77%	44.74%
$4 \times [1024]$, softplus	CRT, 0.01	97.80%	6.25%
	CRT, 0.03	97.09%	29.64%
	CRT, 0.05	96.33%	44.44%
	CRT, 0.07	**95.60%**	**53.19%**
	CROWN-IBP	82.74%	41.34%
$4 \times [1024]$, relu	COAP	90.17%	44.66%
	CROWN-IBP	84.4%	43.83%

Table 5.9: Comparison with interval-bound propagation based adversarial training methods with attack radius $\rho = 1.58$ on MNIST dataset. We again observe that the certified accuracy of softplus network with CROWN-IBP is significantly less than that of a similar ReLU network.

5.C.3 Comparing Randomized Smoothing with CRT

Network	Randomized Smoothing			CRT
	$s = 0.25$	$s = 0.50$	$s = 1.0$	
$2 \times [1024]$, sigmoid	93.75%	93.09%	88.91%	95.61%
$2 \times [1024]$, tanh	94.61%	93.08%	82.26%	**95.00%**
$3 \times [1024]$, sigmoid	94.00%	93.03%	86.58%	**94.99%**
$3 \times [1024]$, tanh	93.69%	91.68%	80.55%	**94.16%**
$4 \times [1024]$, sigmoid	**93.68%**	92.45%	84.99%	93.41%
$4 \times [1024]$, tanh	**93.57%**	92.19%	83.90%	91.37%

Table 5.10: Comparison between CRT and Randomized Smoothing[135]. s denotes the standard deviation for smoothing. We use $\rho = 0.5$. For CRT, we use $\gamma = 0.01$

Since, randomized smoothing is designed to work in untargeted attack settings while CRT is for targeted attacks, we make the following changes in randomized smoothing. First, we use $n_0 = 100$ initial samples to select the label class (l) and false target class (t). The samples for estimation were $n = 100,000$ and failure probability was $\alpha = 0.001$. Then we use the binary version of randomized smoothing for estimation, i.e classify between y and t. To find the adversarial example for adversarial training, we use the cross entropy loss for 2 classes (y and t).

5.C.4 Results on CIFAR-10

Network	Training	Standard Accuracy	Empirical Robust Accuracy	Certified Robust Accuracy	Certificate (mean)	
					CROWN	CRC
$2 \times [1024]$, sigmoid	standard	46.23%	37.82%	14.10%	0.37219	**0.38173**
	$\gamma = 0.01$	45.42%	38.17%	26.50%	0.40540	**0.55010**
$3 \times [1024]$, sigmoid	standard	48.57%	34.80%	0.00%	0.19127	0.01404
	$\gamma = 0.01$	50.31%	39.87%	18.28%	0.24778	**0.37895**
$4 \times [1024]$, sigmoid	standard	46.04%	34.38%	0.00%	0.19340	0.00191
	$\gamma = 0.01$	48.28%	40.10%	21.07%	0.29654	**0.40005**

Table 5.11: Results for CIFAR-10 dataset (only curvature regularization, no CRT training)

Part III

Lipschitz neural networks

Chapter 6: Fantastic Four: Differentiable Bounds on Singular Values of Convolution Layers

6.1 Introduction

Bounding singular values of different layers of a neural network is a way to control the complexity of the model and has been used in different problems including robustness, generalization, optimization, generative modeling, etc. In particular, the spectral norm (the maximum singular value) of a layer bounds the factor by which the norm of the signal increases or decreases during both forward and backward propagation within that layer. If all singular values are all close to one, then the gradients neither explode nor vanish [200, 201, 202, 203]. Spectral norm regularizations/bounds have been used in improving the generalization [204, 205], in training deep generative models [196, 206, 207, 208, 209] and in robustifying models against adversarial attacks [15, 133, 142, 165, 169, 171, 210]. These applications have resulted in multiple works to regularize neural networks by penalizing the spectral norm of the network layers [142, 196, 211, 212, 213, 214].

For a fully connected layer with weights $\mathbf{W}$ and bias $\mathbf{b}$, the lipschitz constant is given by the spectral norm of the weight matrix i.e, $\|\mathbf{W}\|_2$, which can be computed efficiently using the power iteration method [215]. In particular, if the matrix $\mathbf{W}$ is of size $p \times q$, the computational

complexity of power iteration (assuming convergence in constant number of steps) is $\mathcal{O}(pq)$.

Convolution layers [216] are one of the key components of modern neural networks, particularly in computer vision. Consider a convolution filter $\mathbf{L}$ of size $c_{out} \times c_{in} \times h \times w$ where c_{out}, c_{in}, h and w denote the number of output channels, input channels, height and width of the filter respectively; and a square input sample of size $c_{in} \times n \times n$ where n is its height and width. A naive representation of the Jacobian of this layer will result in a matrix of size $n^2 c_{out} \times n^2 c_{in}$. For a typical convolution layer with the filter size $64 \times 3 \times 7 \times 7$ and an ImageNet sized input $3 \times 224 \times 224$ [56], the corresponding jacobian matrix has a very large size: 802816×150528. This makes an explicit computation of the jacobian infeasible. [217] provides a way to compute the spectral norm of convolution layers using convolution and transposed convolution operations in power iteration, thereby avoiding this explicit computation. This leads to an improved running time especially when the number of input/output channels is small (Table 6.1).

However, in addition to the running time, there is an additional difficulty in the approach proposed in [217] (and other existing approaches described later) regarding the computation of the spectral norm gradient (often used as a regularization during the training). The gradient of the largest singular value with respect to the jacobian can be naively computed by taking the outer product of corresponding singular vectors. However, due to the special structure of the convolution operation, the jacobian will be a sparse matrix with repeated elements (see Supp. Section 6.D.1). The naive computation of the gradient will result in non-zero gradient values at elements that should be zero throughout training and also will assign different gradient values at elements that should always be identical. These issues make the gradient computation of the spectral norm with respect to the convolution filter weights using the technique of [217] difficult.

Recently, [214, 218] provided a principled approach for exactly computing the singular

values of convolution layers. They construct n^2 matrices each of size $c_{out} \times c_{in}$ by taking the Fourier transform of the convolution filter (details in Supp. Section 6.B.1). The set of singular values of the Jacobian equals the union of singular values of these n^2 matrices. However, this method can have high computational complexity since it requires SVD of n^2 matrices. Although this method can be adapted to compute the spectral norm of n^2 matrices using power iteration (in parallel with a GPU implementation) instead of full SVD, the intrinsic computational complexity (discussed in Table 6.2) can make it difficult to use this approach for very deep networks and large input sizes especially when computational resources are limited. Moreover, computing the gradient of the spectral norm using this method is not straightforward since each of these n^2 matrices contain complex numbers. Thus, [214] suggests to clip the singular values if they are above a certain threshold to bound the spectral norm of the layer. In order to reduce the training overhead, they clip the singular values only after every 100 iterations. The resulting method reduces the training overhead but is still costly for large input sizes and very deep networks. We report the running time of this method in Table 6.1 and its training time for one epoch (using 1 GPU implementation) in Table 6.4c.

Because of the aforementioned issues, efficient methods to control the spectral norm of convolution layers have resorted to heuristics [134, 196, 212]. Typically, these methods reshape the convolution filter of dimensions $c_{out} \times c_{in} \times h \times w$ to construct a matrix of dimensions $c_{out} \times hwc_{in}$, and use the spectral norm of this matrix as an estimate of the spectral norm of the convolution layer. To regularize during training, they use the outer product of the corresponding singular vectors as the gradient of the largest singular value with respect to the reshaped matrix. Since the weights do not change significantly during each training step, they use only one iteration of power method during each step to update the singular values and vectors (using the singular

Filter shape	Spectral norm bounds						Running time (secs)		
	$\sqrt{hw} \times$				**Bound (Ours)**	Exact [214, 217]	**Ours**	Sedghi[214]	Ryu[217]
	$\|\mathbf{R}\|_2$	$\|\mathbf{S}\|_2$	$\|\mathbf{T}\|_2$	$\|\mathbf{U}\|_2$					
$64 \times 3 \times 7 \times 7$	47.29	46.37	**28.89**	77.31	**28.89**	15.92	0.004	0.94	0.03
$64 \times 64 \times 3 \times 3$	9.54	9.61	**9.33**	10.51	**9.33**	6.01	0.033	8.46	0.04
$64 \times 64 \times 3 \times 3$	6.57	**6.30**	6.49	7.86	**6.30**	5.34	0.033	9.19	0.04
$64 \times 64 \times 3 \times 3$	8.92	**8.71**	8.80	10.97	**8.71**	7.00	0.033	9.16	0.04
$64 \times 64 \times 3 \times 3$	**5.40**	5.52	5.81	6.30	**5.40**	3.82	0.033	9.25	0.03
$128 \times 64 \times 3 \times 3$	6.99	6.92	**6.01**	9.12	**6.01**	4.71	0.033	3.08	0.05
$128 \times 128 \times 3 \times 3$	7.45	7.38	**7.21**	8.51	**7.21**	5.72	0.033	8.98	0.08
$128 \times 128 \times 3 \times 3$	**6.78**	7.18	6.85	8.78	**6.78**	4.41	0.032	9.02	0.08
$128 \times 128 \times 3 \times 3$	**7.57**	7.72	8.89	7.87	**7.57**	4.88	0.033	9.00	0.08
$256 \times 128 \times 3 \times 3$	8.58	**8.45**	8.67	8.77	**8.45**	7.39	0.032	3.92	0.14
$256 \times 256 \times 3 \times 3$	8.19	8.07	**8.04**	8.99	**8.04**	6.58	0.033	13.1	0.26
$256 \times 256 \times 3 \times 3$	8.06	8.13	**7.57**	9.26	**7.57**	6.36	0.034	11.2	0.26
$256 \times 256 \times 3 \times 3$	9.91	9.76	10.97	**9.17**	**9.17**	7.68	0.033	11.2	0.26
$512 \times 256 \times 3 \times 3$	11.24	**11.00**	11.50	11.07	**11.00**	9.99	0.034	5.26	0.51
$512 \times 512 \times 3 \times 3$	10.98	10.65	11.91	**10.45**	**10.45**	9.09	0.033	15.7	1.04
$512 \times 512 \times 3 \times 3$	20.22	19.90	21.91	**18.37**	**18.37**	17.60	0.033	15.7	1.04
$512 \times 512 \times 3 \times 3$	7.83	7.99	8.26	**7.60**	**7.60**	7.48	0.034	15.8	1.04

Table 6.1: Comparison between the exact spectral norm of the jacobian of convolution layers (computed using [214, 217]) and our proposed bound for the a Resnet-18 network pre-trained on ImageNet. Our bound is within 1.5 times the exact spectral norm (except the first layer) while being significantly faster to compute compared to the exact methods. All running times were computed on GPU using tensorflow. For [214]'s method, to have a fair comparison, we *only* compute the largest singular value (and not all singular values) for all individual matrices in parallel using power iteration (on 1 GPU) and take the maximum.

vectors computed in the previous step). These methods result in negligible overhead during the training. However, due to lack of theoretical justifications (which we resolve in this chapter), they are not guaranteed to work for all different shapes and weights of the convolution filter. Previous studies have observed under estimation of the spectral norm using these heuristics [219].

On one hand, there are computationally efficient but heuristic ways of computing and bounding the spectral norm of convolutional layers [196, 213]. On the other hand, the exact computation of the spectral norm of convolutional layers proposed by [214, 217] can be expensive for

commonly used architectures especially with large inputs such as ImageNet samples. Moreover, the difficulty in computing the gradient of the spectral norm with respect to the jacobian under these methods make their use as regularization during the training process challenging.

In this chapter, we resolve these issues by deriving a differentiable and efficient *upper bound* on the spectral norm of convolutional layers. Our bound is provable and not based on heuristics. Our computational complexity is similar to that of heuristics [196, 213] allowing our bound to be used as a regularizer for efficiently training deep convolutional networks. In this way, our proposed approach combines the speed of the heuristics and the theoretical rigor of [214]. Table 6.2 summarizes the differences between previous works and our approach. In Table 6.1, we empirically observe that our bound can be computed in a time significantly faster than [214, 217], while providing a guaranteed upper bound on the spectral norm. Moreover, we empirically observe that our upper bound and the exact value are close to each other (Section 6.3.1).

Below, we briefly explain our main result. Consider a convolution filter $\mathbf{L}$ of dimensions $c_{out} \times c_{in} \times h \times w$ and input of size $c_{in} \times n \times n$. The corresponding jacobian matrix $\mathbf{J}$ is of size $n^2 c_{out} \times n^2 c_{in}$. We show that the largest singular value of the jacobian (i.e. $\|\mathbf{J}\|_2$) is bounded as:

$$\|\mathbf{J}\|_2 \leq \sqrt{hw} \min\left(\|\mathbf{R}\|_2, \|\mathbf{S}\|_2, \|\mathbf{T}\|_2, \|\mathbf{U}\|_2\right),$$

where $\mathbf{R}, \mathbf{S}, \mathbf{T}$ and $\mathbf{U}$ are matrices of sizes $hc_{out} \times wc_{in}$, $wc_{out} \times hc_{in}$, $c_{out} \times hwc_{in}$ and $hwc_{out} \times c_{in}$ respectively, and can be computed by appropriately reshaping the filter $\mathbf{L}$ (details in Section 6.3). This upper bound is independent of the input width and height (n). Formal results are stated in Theorem 11 and proved in the Supp. 6.C.1 . Remarkably, $\|\mathbf{T}\|_2$ is the heuristic suggested by

	Exact (Sedghi et al.[214])	Exact (Ryu et al.[217])	**Upper Bound (Ours)**
Computation	Norm of n^2 matrices, each of size: $c_{out}c_{in}$	Norm of one matrix of size: $n^4c_{out}c_{in}$	Norm of four matrices, each of size: $c_{out}c_{in}hw$
Time complexity ($\mathcal{O}$)	$n^2\ c_{out}\ c_{in}$	$n^2\ h\ w\ c_{out}\ c_{in}$	$h\ w\ c_{out}\ c_{in}$
Guaranteed bound	✓	✓	✓
Easy gradient computation	✗	✗	✓

Table 6.2: Comparison of various methods used for computing the norm of convolution layers. n is the height and width for a square input, c_{in} is the number of input channels, c_{out} is the number of output channels, h and w are the height and width of the convolution filter. For [214], we only compute the largest singular value using power iteration (i.e not all singular values).

[196]. To the best of our knowledge, this is the first work that derives a provable bound on the spectral norm of convolution filter as a constant factor (dependant on filter sizes, but not filter weights) times the heuristic of [196]. In Tables 6.1 and 6.3, we show that the other 3 bounds (using $\|\mathbf{R}\|_2, \|\mathbf{S}\|_2, \|\mathbf{U}\|_2$) can be significantly smaller than $\sqrt{hw}\|\mathbf{T}\|_2$ for different convolution filters. Thus, we take the minimum of these 4 quantities to bound the spectral norm of the filter.

In Section 6.4, we show that our bound can be used to improve the generalization and robustness properties of neural networks. Specifically, we show that using our bound as a regularizer during training, we can achieve improvement in accuracy on par with exact method [214] while being significantly faster to train (Table 6.4). We also achieve significantly higher robustness certificates against adversarial attacks than CNN-Cert [220] on a single layer CNN (Table 6.5). These results demonstrate potentials for practical uses of our results. Code is available at the github repository: https://github.com/singlasahil14/fantastic-four.

6.2 Notation

For a vector $\mathbf{v}$, we use $\mathbf{v}_j$ to denote the element in the jth position of the vector. We use $\mathbf{A}_{j,:}$ and $\mathbf{A}_{:,k}$ to denote the jth row and k^{th} column of the matrix $\mathbf{A}$ respectively. We assume both $\mathbf{A}_{j,:}$, $\mathbf{A}_{:,k}$ to be column vectors (thus $\mathbf{A}_{j,:}$ is the transpose of jth row of $\mathbf{A}$). $\mathbf{A}_{j,k}$ denotes the element in j^{th} row and k^{th} column of $\mathbf{A}$. The same rules can be directly extended to higher order tensors. For a matrix $\mathbf{A} \in \mathbb{R}^{q \times r}$ and a tensor $\mathbf{B} \in \mathbb{R}^{p \times q \times r}$, $vec(\mathbf{A})$ denotes the vector constructed by stacking the rows of $\mathbf{A}$ and $vec(\mathbf{B})$ denotes the vector constructed by stacking the vectors $vec(\mathbf{B}_{j,:,:}),\ j \in [p-1]$:

$$vec(\mathbf{A})^T = \begin{bmatrix} \mathbf{A}_{0,:}^T\ ,\ \mathbf{A}_{1,:}^T\ ,\ \ldots\ ,\ \mathbf{A}_{q-1,:}^T \end{bmatrix} \qquad vec(\mathbf{B})^T = \begin{bmatrix} \mathbf{B}_{0,:,:}^T\ ,\ \mathbf{B}_{1,:,:}^T\ ,\ \ldots\ ,\ \mathbf{B}_{p-1,:,:}^T \end{bmatrix}$$

We use the following notation for a convolutional neural network. L denotes the number of layers and ϕ is the activation function. For an input $\mathbf{x}$, we use $\mathbf{z}^{(I)}(\mathbf{x}) \in \mathbf{R}^{N_I}$ and $\mathbf{a}^{(I)}(\mathbf{x}) \in \mathbf{R}^{N_I}$ to denote the raw (*before* applying ϕ) and activated (*after* applying ϕ) neurons in the Ith hidden layer respectively. $\mathbf{a}^{(0)}$ denotes the input image $\mathbf{x}$. To simplify notation and when no confusion arises, we make the dependency of $\mathbf{z}^{(I)}$ and $\mathbf{a}^{(I)}$ to $\mathbf{x}$ implicit. $\phi'(\mathbf{z}^{(I)})$ and $\phi''(\mathbf{z}^{(I)})$ denotes the elementwise first and second derivatives of ϕ at $\mathbf{z}^{(1)}$. $\mathbf{W}^{(I)}$ denotes the weights for the Ith layer i.e $\mathbf{W}^{(I)}$ will be a tensor for a convolution layer and a matrix for a fully connected layer. $\mathbf{J}^{(I)}$ denotes the jacobian matrix of $vec(\mathbf{z}^{(1)})$ with respect to the input $\mathbf{x}$. θ denotes the neural network parameters. $f_\theta(\mathbf{x})$ denotes the softmax probabilities output by the network for an input $\mathbf{x}$. For an input $\mathbf{x}$ and label y, the cross entropy loss is denoted by $\ell\left(f_\theta(\mathbf{x}), y\right)$.

6.3 Main Results

Consider a convolution filter $\mathbf{L}$ of size $c_{out} \times c_{in} \times h \times w$ applied to an input $\mathbf{X}$ of size $c_{in} \times n \times n$. The filter $\mathbf{L}$ takes an input patch of size $c_{in} \times h \times w$ from the input and outputs a vector of size c_{out} for every such patch. The same operation is applied across all such patches in the image. To apply convolution at the edges of the image, modern convolution layers either not compute the outputs thus reducing the size of the generated feature map, or pad the input with zeros to preserve its size. When we pad the image with zeroes, the corresponding jacobian becomes a *toeplitz matrix*. Another version of convolution treats the input as if it were a torus; when the convolution operation calls for a pixel off the right end of the image, the layer "wraps around" to take it from the left edge, and similarly for the other edges. For this version of convolution, the jacobian is a *circulant matrix*. The quality of approximations between toeplitz and circulant colvolutions has been analyzed in the case of 1D [221] and 2D signals [222]. For the 2D case (similar to the 1D case), $O(1/p)$ bound is obtained on the error, where $p \times p$ is the size of both (topelitz and circulant) matrices. Consequently, theoretical analysis of convolutions that wrap around the input (i.e using circulant matrices) has been become standard. This is the case that we analyze in this work. Furthermore, we assume that the stride size to be 1 in both horizontal and vertical directions.

The output $\mathbf{Y}$ produced from applying the filter $\mathbf{L}$ to an input $\mathbf{X}$ is of size $c_{out} \times n \times n$. The corresponding jacobian ($\mathbf{J}$) will be a matrix of size $n^2 c_{out} \times n^2 c_{in}$ satisfying:

$$vec(\mathbf{Y}) = \mathbf{J} vec(\mathbf{X}).$$

Our goal is to bound the norm of jacobian of the convolution operation i.e, $\|\mathbf{J}\|_2$. [214] also derive an expression for the exact singular values of the jacobian of convolution layers. However, their method requires the computation of the spectral norm of n^2 matrices (each matrix of size $c_{out} \times c_{in}$) for every convolution layer. We extend their result to derive a differentiable and easy-to-compute upper bound on singular values stated in the following theorem:

Theorem 11. *Consider a convolution filter* $\mathbf{L}$ *of size* $c_{out} \times c_{in} \times h \times w$ *that applied to input* $\mathbf{X}$ *of size* $c_{in} \times n \times n$ *gives output* $\mathbf{Y}$ *of size* $c_{out} \times n \times n$. *The jacobian of* $\mathbf{Y}$ *with respect to* $\mathbf{X}$ *(i.e.* $\mathbf{J}$*) will be a matrix of size* $n^2 c_{out} \times n^2 c_{in}$. *The spectral norm of* $\mathbf{J}$ *is bounded by:*

$$\|\mathbf{J}\|_2 \leq \sqrt{hw}\ \min\left(\|\mathbf{R}\|_2, \|\mathbf{S}\|_2, \|\mathbf{T}\|_2, \|\mathbf{U}\|_2\right),$$

where the matrices $\mathbf{R}, \mathbf{S}, \mathbf{T}$ *and* $\mathbf{U}$ *are defined as follows:*

$$\mathbf{R} = \begin{bmatrix} \mathbf{L}_{:,:,0,0} & \mathbf{L}_{:,:,0,1} & \cdots & \mathbf{L}_{:,:,0,w-1} \\ \mathbf{L}_{:,:,1,0} & \mathbf{L}_{:,:,1,1} & \cdots & \mathbf{L}_{:,:,1,w-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{L}_{:,:,h-1,0} & \mathbf{L}_{:,:,h-1,1} & \cdots & \mathbf{L}_{:,:,h-1,w-1} \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} \mathbf{L}_{:,:,0,0} & \mathbf{L}_{:,:,1,0} & \cdots & \mathbf{L}_{:,:,h-1,0} \\ \mathbf{L}_{:,:,0,1} & \mathbf{L}_{:,:,1,1} & \cdots & \mathbf{L}_{:,:,h-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{L}_{:,:,0,w-1} & \mathbf{L}_{:,:,1,w-1} & \cdots & \mathbf{L}_{:,:,h-1,w-1} \end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix} \mathbf{A}_0 & \mathbf{A}_1 & \cdots & \mathbf{A}_{h-1} \end{bmatrix}, \quad \textit{where} \quad \mathbf{A}_l = \begin{bmatrix} \mathbf{L}_{:,:,l,0} & \mathbf{L}_{:,:,l,1} & \cdots & \mathbf{L}_{:,:,l,w-1} \end{bmatrix}$$

$$\mathbf{U}^T = \begin{bmatrix} \mathbf{B}_0^T & \mathbf{B}_1^T & \cdots & \mathbf{B}_{h-1}^T \end{bmatrix}, \quad \textit{where} \quad \mathbf{B}_l^T = \begin{bmatrix} \mathbf{L}_{:,:,l,0}^T & \mathbf{L}_{:,:,l,1}^T & \cdots & \mathbf{L}_{:,:,l,w-1}^T \end{bmatrix}$$

Proof of Theorem 11 is in Supp. 6.C.1. The matrices $\mathbf{R}, \mathbf{S}, \mathbf{T}$ and $\mathbf{U}$ are of dimensions $c_{out}h \times c_{in}w, c_{out}w \times c_{in}h, c_{out} \times c_{in}hw$ and $c_{out}hw \times c_{in}$ respectively. In the current literature [196], the heuristic used for estimating the spectral norm involves combining the dimensions of

sizes h, w, c_{in} in the filter $\mathbf{L}$ to create the matrix $\mathbf{T}$ of dimensions $c_{out} \times hwc_{in}$. The norm of resulting matrix is used as a heuristic estimate of the spectral norm of the jacobian of convolution operator. However, in Theorem 11 we show that the norm of this matrix multiplied with a factor of $\sqrt{hw}$ gives a provable upper bound on the singular values of the jacobian. In Tables 6.1 and 6.3, we show that for different convolution filters, there can be significant differences between the four bounds and any of these four bounds can be the minimum.

6.3.1 Tightness analysis

In Supp. 6.C.2, we show that the bound is exact for a convolution filter with $h = w = 1$:

Lemma 4. *For $h = 1, w = 1$, the bounds in Theorem 11 are exact i.e:*

$$\|\mathbf{J}\|_2 = \|\mathbf{R}\|_2 = \|\mathbf{S}\|_2 = \|\mathbf{T}\|_2 = \|\mathbf{U}\|_2$$

In Table 6.3, we analyze the tightness between our bound and the exact largest singular value computed by [214] for different filter shapes. Each convolution filter was constructed by sampling from a standard normal distribution $\mathcal{N}(0, 1)$. We observe that the bound is tight for small filter sizes but the ratio (Bound/Exact) becomes large for large h and w. We also observe that the values computed by the four bounds can be significantly different and that we can get a significantly improved bound by taking the minimum of the four quantities. In Supp. Section 6.D.2, Figure 6.1, we empirically observe that the gap between our upper bound and the exact value can become very small by adding our bound as a regularizer during training.

Filter shape	$\sqrt{hw} \times$				**Bound (Ours)**	Exact (Sedghi)	Bound/ Exact
	$\|\mathbf{R}\|_2$	$\|\mathbf{S}\|_2$	$\|\mathbf{T}\|_2$	$\|\mathbf{U}\|_2$			
$64 \times 64 \times 2 \times 2$	**44.19**	46.11	46.76	47.88	**44.19**	31.96	1.38
$64 \times 64 \times 3 \times 3$	84.03	**82.71**	95.27	97.36	**82.71**	48.77	1.70
$64 \times 64 \times 5 \times 5$	179.09	**177.31**	237.22	238.30	**177.31**	82.39	2.15
$64 \times 64 \times 7 \times 7$	297.02	**296.18**	444.69	451.52	**296.18**	116.05	2.55
$64 \times 16 \times 2 \times 2$	32.27	32.45	**31.18**	38.20	**31.18**	23.25	1.34
$64 \times 16 \times 3 \times 3$	60.74	62.42	**59.67**	82.17	**59.67**	37.09	1.61
$64 \times 16 \times 5 \times 5$	**130.07**	132.62	136.16	216.22	**130.07**	61.01	2.13
$64 \times 16 \times 7 \times 7$	220.25	**220.20**	248.50	415.14	**220.20**	86.54	2.54

Table 6.3: Tightness analysis between our proposed bound and exact method by [214].

6.3.2 Gradient Computation

Since the matrix $\mathbf{R}$ can be directly computed by reshaping the filter weights $\mathbf{L}$ ((6.8)), we can compute the derivative of our bound $\sqrt{hw}\|\mathbf{R}\|_2$ (or $\|\mathbf{S}\|_2, \|\mathbf{T}\|_2, \|\mathbf{U}\|_2$) with respect to filter weights $\mathbf{L}$ by first computing the derivative of $\sqrt{hw}\|\mathbf{R}\|_2$ with respect to $\mathbf{R}$ and then appropriately reshaping the obtained derivative.

Let $\mathbf{u}$ and $\mathbf{v}$ be the singular vectors corresponding to the largest singular value, i.e. $\|\mathbf{R}\|_2$. Then, the derivative of our upper bound $\sqrt{hw}\|\mathbf{R}\|_2$ with respect to $\mathbf{R}$ can be computed as follows:

$$\nabla_{\mathbf{R}} \sqrt{hw}\|\mathbf{R}\|_2 = \sqrt{hw}\ \mathbf{u}\mathbf{v}^T \quad \text{where } \|\mathbf{R}\|_2 = \mathbf{u}^T\mathbf{R}\mathbf{v}$$

Moreover, since the weights do not change significantly during the training, we can use one iteration of power method to update $\mathbf{u}, \mathbf{v}$ and $\|\mathbf{R}\|_2$ during each training step (similar to [196, 213]). This allows us to use our bound efficiently as a regularizer during the training process.

6.4 Experiments

All experiments were conducted using a single NVIDIA GeForce RTX 2080 Ti GPU.

6.4.1 Comparison with existing methods

In Table 6.1, we show a comparison between the exact spectral norms (computed using [214], [217]) and our upper bound, i.e. $\sqrt{hw}\min(\|\mathbf{R}\|_2, \|\mathbf{S}\|_2, \|\mathbf{T}\|_2, \|\mathbf{U}\|_2)$ on a pre-trained Resnet-18 network [1]. Except for the first layer, we observe that our bound is within 1.5 times the value of the exact spectral norm while being significantly faster to compute. Similar results can be observed in Table 6.3 for a standard gaussian filter. Thus, by taking the minimum of the four bounds, we can get a significant gain.

6.4.2 Effects on generalization

In Table 6.4a, we study the effect of using our proposed bound as a training regularizer on the generalization error. We use a Resnet-32 neural network architecture and the CIFAR-10 dataset [57] for training. For regularization, we use the sum of spectral norms of all layers of the network during training. Thus, our regularized objective function is given as follows[1]:

$$\min_{\theta} \quad \mathbb{E}_{(\mathbf{x},y)}\left[\ell\left(f_\theta(\mathbf{x}), y\right)\right] + \beta \sum_{I} u^{(I)} \tag{6.1}$$

where β is the regularization coefficient, $(\mathbf{x}, y)$'s are the input-label pairs in the training data, $u^{(I)}$ denotes the bound for the Ith convolution or fully connected layer. For the convolution layers,

[1]We do not use the sum of log of spectral norm values since that would make the filter-size dependant factor of $\sqrt{hw}$ irrelevant for gradient computation.

β	Test Accuracy	
	No weight decay	weight decay = 10^{-4}
0	91.26%	92.53%
0.0008	91.62%	92.37%
0.0009	91.74%	92.95%
0.0010	91.79%	93.13%
0.0011	91.87%	92.75%
0.0012	91.81%	92.53%
0.0013	92.17%	92.73%
0.0014	**92.23%**	92.61%
0.0015	91.87%	93.15%
0.0016	91.92%	**93.26%**
0.0017	91.70%	92.92%
0.0018	91.52%	91.89%
0.0019	92.00%	92.47%
0.0020	91.82%	92.56%

(a) Our proposed regularizer

Clipping threshold	Test Accuracy	
	No weight decay	weight decay = 10^{-4}
None	91.26%	92.53%
0.1	90.67%	91.87%
0.5	**92.31%**	**93.30%**
1.0	91.92%	92.86%

(b) Singular value clipping [214]

Method	Running time (s)	Increase (%)
Standard	49.58	-
Ours	54.39	9.7%
Clipping [214]	156.46	215.6%

(c) Running time for 1 epoch on a dataset with 10,000 images of size 224x224x3

Table 6.4: Comparison between our proposed regularizer (a) and singular value clipping by [214] (b) on test accuracy. Results are on CIFAR-10 dataset using a Resnet-32 network.

$u^{(I)}$ is computed as $\sqrt{hw}\min(\|\mathbf{R}\|_2, \|\mathbf{S}\|_2, \|\mathbf{T}\|_2, \|\mathbf{U}\|_2)$ using Theorem 11. For fully connected layers, we can compute $u^{(I)}$ using power iteration [196].

Since weight decay [223] indirectly minimizes the Frobenius norm squared which is equal to the sum of squares of singular values, it implicitly forces the largest singular values for each layer (i.e the spectral norm) to be small. Therefore, to measure the effect of our regularizer on the test set accuracy, we compare the effect of adding our regularizer both with and without weight decay. The weight decay coefficient was selected using grid search using 20 values between $[0, 2 \times 10^{-3}]$ using a held-out validation set of 5000 samples.

Our experimental results are reported in Table 6.4a. For the case of no weight decay, we observe an improvement of 0.97% over the case when $\beta = 0$. When we include a weight decay of

10^{-4} in training, there is an improvement of 0.73% over the baseline. Using the method of [214] (i.e. clipping the singular values above 0.5 and 1) results in gain of 0.77% (with weight decay) and 1.05% (without weight decay) which is similar to the results mentioned in their work. In addition to obtaining on par performance gains with the exact method of [214], a key advantage of our approach is its very efficient running time allowing its use for very large input sizes and deep networks. We report the training time of these methods in Table 6.4(c) for a dataset with large input sizes. The increase in the training time using our method compared to the standard training is just 9.7% while that for [214] is around 215.6%.

6.4.3 Effects on provable adversarial robustness

In this part, we show the usefulness of our spectral bound in enhancing provable robustness of convolutional classifiers against adversarial attacks [15]. A robustness certificate is a lower bound on the minimum distance of a given input to the decision boundary of the classifier. For *any* perturbation of the input with a magnitude smaller than the robustness certificate value, the classification output will provably remain the same. However, computing exact robustness certificates requires solving a non-convex optimization, making the computation difficult for deep classifiers. In the last couple of years, several certifiable defenses against adversarial attacks have been proposed (e.g. [135, 138, 138, 140, 142, 165, 166, 220, 224, 225].) In particular, to show the usefulness of our spectral bound in this application, we examine the method proposed in [142] that uses a bound on the lipschitz constant of network gradients (i.e. the curvature values of the network). Their robustness certification method in fact depends on the spectral norm of the Jacobian of different network layers; making a perfect case study for the use of our spectral bound

(details in Supp. Section 6.B.2).

Due to the difficulty of computing $\|\mathbf{J}^{(I)}\|_2$ when the Ith layer is a convolution, [142] restrict their analysis to fully connected networks where $\mathbf{J}^{(I)}$ simply equals the Ith layer weight matrix. However, using our results in Theorem 11, we can further bound $\mathbf{J}^{(I)}$ and run similar experiments for the convolution layers. Thus, for a 2 layer convolution network, our regularized objective function is given as follows:

$$\min_{\theta} \quad \mathbb{E}_{(\mathbf{x},y)}\left[\ell\left(f_\theta(\mathbf{x}), y\right)\right] + \gamma b \max_{j}\left(|\mathbf{W}^{(2)}_{y,j} - \mathbf{W}^{(2)}_{t,j}|\right)\left(u^{(1)}\right)^2$$

where γ is the regularization coefficient, b is a bound on the second-derivative of the activation function ($|\sigma^{''}(.)| \leq b$), $(\mathbf{x}, y)$'s are the input-label pairs in the training data, and $u^{(I)}$ denotes the bound on the spectral norm for the Ith linear (convolution/fully connected) layer. For the convolutional layer, $u^{(1)}$ is again computed as $\sqrt{hw}\min(\|\mathbf{R}\|_2, \|\mathbf{S}\|_2, \|\mathbf{T}\|_2, \|\mathbf{U}\|_2)$ using Theorem 11.

In Table 6.5, we study the effect of the regularization coefficient γ on provable robustness when the network is trained with curvature regularization [142]. We use a 2 layer convolutional neural network with the tanh [226] activation function and 5 filters in the convolution layer. We observe that the method in [142] coupled with our bound gives significantly higher robustness certificate than CNN-Cert, the previous state of the art [220].

In Supp. Table 6.7, we study the effect of the adversarial training method in [142] coupled with our bound on certified robust accuracy. We achieve certified accuracy of 91.25% on MNIST dataset and 29.26% on CIFAR-10 dataset. These results further highlight the usefulness of our spectral bound on convolution layers in improving the provable robustness of the networks against

γ	MNIST				
	Standard Accuracy	Certified Robust Accuracy	CNN-Cert	CRT + Our bound	Certificate Improvement (Percentage %)
0	98.35%	0.0%	0.1503	**0.1770**	17.76%
0.01	94.85%	75.26%	0.2135	**0.8427**	294.70%
0.02	93.18%	74.42%	0.2378	**0.9048**	280.49%
0.03	91.97%	72.89%	0.2547	**0.9162**	259.71%

Table 6.5: Comparison between robustness certificates computed using CNN-Cert [220] and the method proposed in [142] coupled with our spectral bound, for different values of γ for a single hidden layer convolutional neural network with tanh activation function. Certified Robust Accuracy is computed as the fraction of correctly classified samples with robustness-certificate (computed using [142]) greater than 0.5.

adversarial attacks.

6.5 Conclusion

In this chapter, we derive four efficient and differentiable bounds on the spectral norm of convolution layer and take their minimum as our single tight spectral bound. This bound significantly improves over the popular heuristic method of [196, 213], for which we provide the first provable guarantee. Compared to the exact methods of [214] and [217], our bound is significantly more efficient to compute, making it amendable to be used in large-scale problems. Over various filter sizes, we empirically observe that the gap between our bound and the true spectral norm is small. Using experiments on MNIST and CIFAR-10, we demonstrate the usefulness of our spectral bound in enhancing generalization as well as provable adversarial robustness of convolutional classifiers.

Supplementary material

6.A Notation for supplementary material

For a vector $\mathbf{v}$, we use $\mathbf{v}_j$ to denote the element in the jth position of the vector. We use $\mathbf{A}_{j,:}$ to denote the jth row of the matrix $\mathbf{A}$, $\mathbf{A}_{:,k}$ to denote the kth column of the matrix $\mathbf{A}$. We assume both $\mathbf{A}_{j,:}$ and $\mathbf{A}_{:,k}$ to be column vectors (thus $\mathbf{A}_{j,:}$ is constructed by taking the transpose of jth row of $\mathbf{A}$). $\mathbf{A}_{j,k}$ denotes the element in j^{th} row and k^{th} column of $\mathbf{A}$. $\mathbf{A}_{j,:k}$ and $\mathbf{A}_{:j,k}$ denote the vectors containing the first k elements of the jth row and first j elements of kth column, respectively. $\mathbf{A}_{:j,:k}$ denotes the matrix containing the first j rows and k columns of $\mathbf{A}$:

$$\mathbf{A}_{j,:k} = \begin{bmatrix} \mathbf{A}_{j,0} \\ \mathbf{A}_{j,1} \\ \vdots \\ \mathbf{A}_{j,k-1} \end{bmatrix}, \quad \mathbf{A}_{:j,k} = \begin{bmatrix} \mathbf{A}_{0,k} \\ \mathbf{A}_{1,k} \\ \vdots \\ \mathbf{A}_{j-1,k} \end{bmatrix}, \quad \mathbf{A}_{:j,:k} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,k-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,k-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{j-1,0} & \mathbf{A}_{j-1,1} & \cdots & \mathbf{A}_{j-1,k-1} \end{bmatrix}$$

The same rules can be directly extended to higher order tensors.

For $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{0, \ldots, n\}$ and $[p, q]$ $(p < q)$ to denote the set $\{p, p + 1 \ldots, q\}$. We will index the rows and columns of matrices using elements of $[n]$, i.e. numbering from 0. Addition of row and column indices will be done mod n unless otherwise indicated. For a matrix $\mathbf{A} \in \mathbb{R}^{q \times r}$ and a tensor $\mathbf{B} \in \mathbb{R}^{p \times q \times r}$, $vec(\mathbf{A})$ denotes the vector constructed by stacking the rows of $\mathbf{A}$ and $vec(\mathbf{B})$ denotes the vector constructed by stacking the vectors $vec(\mathbf{B}_{j,:,:})$, $j \in$

$[p-1]$:

$$vec(\mathbf{A}) = \begin{bmatrix} \mathbf{A}_{0,:} \\ \mathbf{A}_{1,:} \\ \vdots \\ \mathbf{A}_{q-1,:} \end{bmatrix}, \qquad vec(\mathbf{B}) = \begin{bmatrix} vec(\mathbf{B}_{0,:,:}) \\ vec(\mathbf{B}_{1,:,:}) \\ \vdots \\ vec(\mathbf{B}_{p-1,:,:}) \end{bmatrix}$$

For a given vector $\mathbf{v} \in \mathbb{R}^n$, $circ(\mathbf{v})$ denotes the $n \times n$ circulant matrix constructed from $\mathbf{v}$ i.e rows of $circ(\mathbf{v})$ are circular shifts of $\mathbf{v}$. For a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, $circ(\mathbf{A})$ denotes the $n^2 \times n^2$ doubly block circulant matrix constructed from $\mathbf{A}$, i.e each $n \times n$ block of $circ(\mathbf{A})$ is a circulant matrix constructed from the rows $\mathbf{A}_{j,:},\ j \in [n-1]$:

$$circ(\mathbf{v}) = \begin{bmatrix} \mathbf{v}_0 & \mathbf{v}_1 & \cdots & \mathbf{v}_{n-1} \\ \mathbf{v}_{n-1} & \mathbf{v}_0 & \cdots & \mathbf{v}_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_0 \end{bmatrix}, \qquad circ(\mathbf{A}) = \begin{bmatrix} circ(\mathbf{A}_{0,:}) & \cdots & circ(\mathbf{A}_{n-1,:}) \\ circ(\mathbf{A}_{n-1,:}) & \cdots & circ(\mathbf{A}_{n-2,:}) \\ \vdots & \ddots & \vdots \\ circ(\mathbf{A}_{1,:}) & \cdots & circ(\mathbf{A}_{0,:}) \end{bmatrix}$$

We use $\mathbf{F}$ to denote the Fourier matrix of dimensions $n \times n$, i.e $\mathbf{F}_{j,k} = \omega^{jk}$, $\omega = e^{-2\pi i/n}$, $i^2 = -1$. For a matrix $\mathbf{A}$, $\sigma(\mathbf{A})$ denotes the set of singular values of $\mathbf{A}$. $\sigma_{max}(\mathbf{A})$ and $\sigma_{min}(\mathbf{A})$ denotes the largest and smallest singular values of $\mathbf{A}$ respectively. $\mathbf{A} \otimes \mathbf{B}$ denotes the kronecker product of $\mathbf{A}$ and $\mathbf{B}$. We use $\mathbf{A} \odot \mathbf{B}$ to denote the hadamard product between two matrices (or vectors) of the same size. We use $\mathbf{I}_n$ to denote the identity matrix of dimensions $n \times n$.

We use the following notation for a convolutional neural network. L denotes the number of layers and ϕ denotes the activation ϕ. For an input $\mathbf{x}$, we use $\mathbf{z}^{(I)}(\mathbf{x}) \in \mathbf{R}^{N_I}$ and $\mathbf{a}^{(I)}(\mathbf{x}) \in \mathbf{R}^{N_I}$ to denote the raw (*before* applying the activation ϕ) and activated (*after* applying the activation

function) neurons in the Ith hidden layer of the network, respectively. Thus $\mathbf{a}^{(0)}$ denotes the input image $\mathbf{x}$. To simplify notation and when no confusion arises, we make the dependency of $\mathbf{z}^{(I)}$ and $\mathbf{a}^{(I)}$ to $\mathbf{x}$ implicit. $\phi^{'}(\mathbf{z}^{(I)})$ and $\phi^{''}(\mathbf{z}^{(I)})$ denotes the elementwise derivative and double derivative of ϕ at $\mathbf{z}^{(1)}$. $\mathbf{W}^{(I)}$ denotes the weights for the Ith layer i.e $\mathbf{W}^{(I)}$ will be a tensor for a convolution layer and a matrix for a fully connected layer. $\mathbf{J}^{(I)}$ denotes the jacobian matrix of $vec(\mathbf{z}^{(1)})$ with respect to the input $\mathbf{x}$. θ denotes the neural network parameters. $f_\theta(\mathbf{x})$ denotes the softmax probabilities output by the network for an input $\mathbf{x}$. For an input $\mathbf{x}$ and label y, the cross entropy loss is denoted by $\ell\left(f_\theta(\mathbf{x}), y\right)$.

6.B Review of prior works

6.B.1 Sedghi et al's method

Consider an input $\mathbf{X} \in \mathbb{R}^{c_{in} \times n \times n}$ and a convolution filter $\mathbf{L} \in \mathbb{R}^{c_{out} \times c_{in} \times h \times w}$ to $\mathbf{X}$ such that $n > \max(h, w)$. Using $\mathbf{L}$, we construct a new filter $\mathbf{K} \in \mathbb{R}^{c_{out} \times c_{in} \times n \times n}$ by padding with zeroes:

$$\mathbf{K}_{c,d,k,l} = \left\{ \begin{array}{ll} \mathbf{L}_{c,d,k,l}, & k \in [h-1],\ l \in [w-1] \\ 0, & \text{otherwise} \end{array} \right\}$$

The Jacobian matrix $\mathbf{J}$ of dimensions $c_{out}n^2 \times c_{in}n^2$ is given as follows:

$$\mathbf{J} = \begin{bmatrix} \mathbf{B}^{(0,0)} & \mathbf{B}^{(0,1)} & \dots & \mathbf{B}^{(0,c_{in}-1)} \\ \mathbf{B}^{(1,0)} & \mathbf{B}^{(1,1)} & \dots & \mathbf{B}^{(1,c_{in}-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{B}^{(c_{out}-1,0)} & \mathbf{B}^{(c_{out}-1,1)} & \dots & \mathbf{B}^{(c_{out}-1,c_{in}-1)} \end{bmatrix}$$

where $\mathbf{B}^{(c,d)}$ is given as follows:

$$\mathbf{B}^{(c,d)} = circ(\mathbf{K}_{c,d,:,:})$$

By inspection we can see that:

$$vec(\mathbf{Y}) = \mathbf{J}vec(\mathbf{X})$$

From [214], we know that singular values of $\mathbf{J}$ are given by:

$$\sigma(\mathbf{J}) = \bigcup_{j\in[n-1],k\in[n-1]} \sigma(\mathbf{G}^{(j,k)})$$

where each $\mathbf{G}^{(j,k)}$ is a matrix of dimensions $c_{out} \times c_{in}$ and each element of $\mathbf{G}^{(j,k)}$ is given by:

$$\mathbf{G}^{(j,k)}_{c,d} = (\mathbf{F}^T\mathbf{K}_{c,d,:,:}\mathbf{F})_{j,k}, \qquad c \in [c_{out}-1],\ d \in [c_{in}-1]$$

The largest singular value of $\mathbf{J}$ i.e $\|\mathbf{J}\|_2$ can be computed as follows:

$$\|\mathbf{J}\|_2 = \sigma_{\max}(\mathbf{J}) = \max_{j\in[n-1],k\in[n-1]} \sigma_{\max}(\mathbf{G}^{(j,k)})$$

6.B.2 Second-order provable robustness framework

6.B.2.1 Robustness certification

Given an input $\mathbf{x}^{(0)} \in \mathbb{R}^D$ and a binary classifier f (here $f : \mathbb{R}^D \to \mathbb{R}$ is differentiable everywhere), our goal is to find a lower bound to the decision boundary $f(\mathbf{x}) = 0$. The $primal$ is given

as follows:

$$p^*_{cert} = \min_{\mathbf{x}} \max_{\eta} \left[\frac{1}{2} \|\mathbf{x} - \mathbf{x}^{(0)}\|_2^2 + \eta f(\mathbf{x}) \right]$$

The $dual$ of the above optimization is given as follows:

$$d^*_{cert} = \max_{\eta} \min_{\mathbf{x}} \left[\frac{1}{2} \|\mathbf{x} - \mathbf{x}^{(0)}\|_2^2 + \eta f(\mathbf{x}) \right]$$

The inner minimization can be solved exactly when the function $1/2\|\mathbf{x}-\mathbf{x}^{(0)}\|_2^2+\eta f(\mathbf{x})$ is strictly convex, i.e has a positive-definite hessian. The same condition can be written as:

$$\nabla^2_{\mathbf{x}} \left[\frac{1}{2} \|\mathbf{x} - \mathbf{x}^{(0)}\|_2^2 + \eta f(\mathbf{x}) \right] = \mathbf{I} + \eta \nabla^2_{\mathbf{x}} f \succcurlyeq 0$$

It is easy to verify that if the eigenvalues of the hessian of f i.e $\nabla^2_{\mathbf{x}} f$ are bounded between m and M for all $\mathbf{x} \in \mathbb{R}^D$, i.e:

$$m\mathbf{I} \preccurlyeq \nabla^2_{\mathbf{x}} f \preccurlyeq M\mathbf{I}, \qquad \forall\, \mathbf{x} \in \mathbb{R}^D$$

The hessian $\mathbf{I} + \eta \nabla^2_{\mathbf{x}} f \succcurlyeq 0$ (i.e is positive definite) for $-1/M < \eta < -1/m$. We refer an interested reader to [142] for more details and a formal proof of the above theorem.

Thus the inner minimization can be solved exactly for these set of values resulting in the following optimization:

$$q^*_{cert} = \max_{-1/M \leq \eta \leq -1/m} \min_{\mathbf{x}} \left[\frac{1}{2} \|\mathbf{x} - \mathbf{x}^{(0)}\|_2^2 + \eta f(\mathbf{x}) \right]$$

Note that q^*_{cert} can be solved exactly using convex optimization. Thus, we get the following chain of inequalities:

$$q^*_{cert} \leq d^*_{cert} \leq p^*_{cert}$$

Thus solving q^*_{cert} gives a robustness certificate. The resulting certificate is called *Curvature-based Robustness Certificate*.

6.B.2.2 Adversarial training with curvature regularization

Let $K(\theta, y, t)$ denote the upper bound on the magnitude of curvature values for label y and target t (computed using [142]).

The objective function for training with curvature regularization is given by:

$$\min_{\theta} \quad \mathbb{E}_{(\mathbf{x},y)}\left[\ell\left(f_\theta(\mathbf{x}), y\right)\right] + \gamma K(\theta, y, t)$$

The objective function for adversarial training with curvature regularization is given by:

$$\min_{\theta} \quad \mathbb{E}_{(\mathbf{x},y)}\left[\ell\left(f_\theta(\mathbf{x}^{(attack)}), y\right)\right] + \gamma K(\theta, y, t)$$

$\mathbf{x}^{(attack)}$ is computed by doing an adversarial attack on the input $\mathbf{x}$ either using l_2 bounded PGD or *Curvature-based Attack Optimization* defined in [142]. When *Curvature-based Attack Optimization* is used for adversarial attack, the resulting attack is called *Curvature-based Robust Training* (CRT).

6.C Proofs

6.C.1 Proof of Theorem 11

Proof. Consider an input $\mathbf{X} \in \mathbb{R}^{c_{in} \times n \times n}$ and a convolution filter $\mathbf{L} \in \mathbb{R}^{c_{out} \times c_{in} \times h \times w}$ to $\mathbf{X}$ such that $n > \max(h, w)$. Using $\mathbf{L}$, we construct a new filter $\mathbf{K} \in \mathbb{R}^{c_{out} \times c_{in} \times n \times n}$ by padding with zeroes:

$$\mathbf{K}_{c,d,k,l} = \left\{ \begin{array}{ll} \mathbf{L}_{c,d,k,l}, & k \in [h-1],\ l \in [w-1] \\ 0, & \text{otherwise} \end{array} \right\} \tag{6.2}$$

The output $\mathbf{Y} \in \mathbb{R}^{c_{out} \times n \times n}$ of the convolution operation is given by:

$$\mathbf{Y}_{c,r,s} = \sum_{d=0}^{c_{in}-1} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} \mathbf{X}_{d,r+k,s+l} \mathbf{K}_{c,d,k,l}$$

We construct a matrix $\mathbf{J}$ of dimensions $c_{out}n^2 \times c_{in}n^2$ as follows:

$$\mathbf{J} = \begin{bmatrix} \mathbf{B}^{(0,0)} & \mathbf{B}^{(0,1)} & \dots & \mathbf{B}^{(0,c_{in}-1)} \\ \mathbf{B}^{(1,0)} & \mathbf{B}^{(1,1)} & \dots & \mathbf{B}^{(1,c_{in}-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{B}^{(c_{out}-1,0)} & \mathbf{B}^{(c_{out}-1,1)} & \dots & \mathbf{B}^{(c_{out}-1,c_{in}-1)} \end{bmatrix}$$

where $\mathbf{B}^{(c,d)}$ is given as follows:

$$\mathbf{B}^{(c,d)} = circ(\mathbf{K}_{c,d,:,:})$$

By inspection we can see that:

$$vec(\mathbf{Y}) = \mathbf{J}vec(\mathbf{X})$$

This directly implies that $\mathbf{J}$ is the jacobian of $vec(\mathbf{Y})$ with respect to $vec(\mathbf{X})$ and our goal is to find a differentiable upper bound on the largest singular value of $\mathbf{J}$.

From [214], we know that singular values of $\mathbf{J}$ are given by:

$$\sigma(\mathbf{J}) = \bigcup_{j\in[n-1],k\in[n-1]} \sigma(\mathbf{G}^{(j,k)}) \tag{6.3}$$

where each $\mathbf{G}^{(j,k)}$ is a matrix of dimensions $c_{out} \times c_{in}$ and each element of $\mathbf{G}^{(j,k)}$ is given by:

$$\mathbf{G}^{(j,k)}_{c,d} = (\mathbf{F}^T\mathbf{K}_{c,d,:,:}\mathbf{F})_{j,k}, \qquad c \in [c_{out} - 1],\ d \in [c_{in} - 1] \tag{6.4}$$

Using (6.3), we can directly observe that a differentiable upper bound over the largest singular value of $\mathbf{G}^{(j,k)}$ that is independent of j and k will give an desired upper bound. We will next derive the same.

Using (6.4), we can rewrite $\mathbf{G}^{(j,k)}_{c,d}$ as:

$$\mathbf{G}^{(j,k)}_{c,d} = (\mathbf{F}^T\mathbf{K}_{c,d,:,:}\mathbf{F})_{j,k} = (\mathbf{F}_{:,j})^T\mathbf{K}_{c,d,:,:}\mathbf{F}_{:,k} \tag{6.5}$$

Using (6.2), we know that $\mathbf{K}_{c,d,:,:}$ is a sparse matrix of size $n \times n$ with only the top-left block of size $h \times w$ that is non-zero. We take the first h rows and w columns of $\mathbf{K}_{c,d,:,:}$ i.e $\mathbf{K}_{c,d,:h,:w}$, first h elements of $\mathbf{F}_{:,j}$ i.e $\mathbf{F}_{:h,j}$ and first w elements of $\mathbf{F}_{:,k}$ i.e $\mathbf{F}_{:w,k}$. We have the following set of

equalities:

$$\begin{aligned}\mathbf{G}_{c,d}^{(j,k)} &= (\mathbf{F}_{:,j})^T \mathbf{K}_{c,d,:,:} \mathbf{F}_{:,k} \\ &= (\mathbf{F}_{:h,j})^T \mathbf{K}_{c,d,:h,:w} \mathbf{F}_{:w,k} \\ &= (\mathbf{F}_{:h,j})^T \mathbf{L}_{c,d,:,:} \mathbf{F}_{:w,k} \end{aligned} \tag{6.6}$$

Thus, $\mathbf{G}^{(j,k)}$ can be written as follows:

$$\begin{aligned}\mathbf{G}_{c,d}^{(j,k)} &= \sum_{l=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{F}_{l,j} \mathbf{L}_{c,d,l,m} \mathbf{F}_{m,k} \\ \mathbf{G}^{(j,k)} &= \sum_{l=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{F}_{l,j} \mathbf{L}_{:,:,l,m} \mathbf{F}_{m,k} \end{aligned} \tag{6.7}$$

Now consider a block matrix $\mathbf{R}$ of dimensions $c_{out}h \times c_{in}w$ given by:

$$\mathbf{R} = \begin{bmatrix} \mathbf{L}_{:,:,0,0} & \mathbf{L}_{:,:,0,1} & \cdots & \mathbf{L}_{:,:,0,w-1} \\ \mathbf{L}_{:,:,1,0} & \mathbf{L}_{:,:,1,1} & \cdots & \mathbf{L}_{:,:,1,w-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{L}_{:,:,h-1,0} & \mathbf{L}_{:,:,h-1,1} & \cdots & \mathbf{L}_{:,:,h-1,w-1} \end{bmatrix} \tag{6.8}$$

Thus the block in l^{th} row and m^{th} column of $\mathbf{R}$ is the $c_{out} \times c_{in}$ matrix $\mathbf{L}_{:,:,l,m}$.

Consider two matrices: $(\mathbf{F}_{:h,j})^T \otimes \mathbf{I}_{c_{out}}$ (of dimensions $c_{out} \times c_{out}h$) and $\mathbf{F}_{:w,k} \otimes \mathbf{I}_{c_{in}}$ (of dimensions $c_{in}w \times c_{in}$).

Using (6.7) and (6.8), we can see that:

$$\mathbf{G}^{(j,k)} = \left((\mathbf{F}_{:h,j})^T \otimes \mathbf{I}_{c_{out}}\right) \mathbf{R} \left(\mathbf{F}_{:w,k} \otimes \mathbf{I}_{c_{in}}\right) \tag{6.9}$$

Taking the spectral norm of both sides:

$$\left\|\mathbf{G}^{(j,k)}\right\|_2 = \left\|\left((\mathbf{F}_{:h,j})^T \otimes \mathbf{I}_{c_{out}}\right) \mathbf{R} \left(\mathbf{F}_{:w,k} \otimes \mathbf{I}_{c_{in}}\right)\right\|_2$$

$$\left\|\mathbf{G}^{(j,k)}\right\|_2 \leq \left\|\left((\mathbf{F}_{:h,j})^T \otimes \mathbf{I}_{c_{out}}\right)\right\|_2 \|\mathbf{R}\|_2 \left\|(\mathbf{F}_{:w,k} \otimes \mathbf{I}_{c_{in}})\right\|_2$$

Using $\|\mathbf{A} \otimes \mathbf{B}\|_2 = \|\mathbf{A}\|_2 \|\mathbf{B}\|_2$ and since both $\|\mathbf{I}_{c_{out}}\|_2$ and $\|\mathbf{I}_{c_{in}}\|_2$ are 1:

$$\left\|\mathbf{G}^{(j,k)}\right\|_2 \leq \|\mathbf{F}_{:h,j}\|_2 \|\mathbf{R}\|_2 \|\mathbf{F}_{:w,k}\|_2$$

Further note that since $\mathbf{F}_{j,k} = \omega^{jk}$, we have $\|\mathbf{F}_{:h,j}\|_2 = \sqrt{h}$ and $\|\mathbf{F}_{:w,k}\|_2 = \sqrt{w}$.

$$\left\|\mathbf{G}^{(j,k)}\right\|_2 \leq \sqrt{hw} \|\mathbf{R}\|_2$$

Alternative inequality (1): Note that (6.6) can also be written by taking the transpose of the scalar $(\mathbf{F}_{:h,j})^T \mathbf{L}_{:,:,c,d} \mathbf{F}_{:w,k}$:

$$\mathbf{G}^{(j,k)}_{c,d} = (\mathbf{F}_{:h,j})^T \mathbf{L}_{c,d,:,:} \mathbf{F}_{:w,k} = (\mathbf{F}_{:w,k})^T \left(\mathbf{L}_{c,d,:,:}\right)^T \mathbf{F}_{:h,j}$$

Thus, $\mathbf{G}^{(j,k)}$ can alternatively be written as follows:

$$\mathbf{G}_{c,d}^{(j,k)} = \sum_{l=0}^{w-1} \sum_{m=0}^{h-1} \mathbf{F}_{l,k} \mathbf{L}_{c,d,m,l} \mathbf{F}_{m,j}$$
$$\mathbf{G}^{(j,k)} = \sum_{l=0}^{w-1} \sum_{m=0}^{h-1} \mathbf{F}_{l,k} \mathbf{L}_{:,:,m,l} \mathbf{F}_{m,j} \tag{6.10}$$

Now consider a block matrix $\mathbf{S}$ of dimensions $c_{out}w \times c_{in}h$ given by:

$$\mathbf{S} = \begin{bmatrix} \mathbf{L}_{:,:,0,0} & \mathbf{L}_{:,:,1,0} & \cdots & \mathbf{L}_{:,:,h-1,0} \\ \mathbf{L}_{:,:,0,1} & \mathbf{L}_{:,:,1,1} & \cdots & \mathbf{L}_{:,:,h-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{L}_{:,:,0,w-1} & \mathbf{L}_{:,:,1,w-1} & \cdots & \mathbf{L}_{:,:,h-1,w-1} \end{bmatrix} \tag{6.11}$$

For $\mathbf{S}$ the block in l^{th} row and m^{th} column of $\mathbf{R}$ is the $c_{out} \times c_{in}$ matrix $\mathbf{L}_{:,:,m,l}$.

Consider two matrices: $(\mathbf{F}_{:w,k})^T \otimes \mathbf{I}_{c_{out}}$ (of dimensions $c_{out} \times c_{out}w$) and $\mathbf{F}_{:h,j} \otimes \mathbf{I}_{c_{in}}$ (of dimensions $c_{in}h \times c_{in}$).

Using (6.10) and (6.11), we again have:

$$\mathbf{G}^{(j,k)} = \left((\mathbf{F}_{:w,k})^T \otimes \mathbf{I}_{c_{out}} \right) \mathbf{S} \left(\mathbf{F}_{:h,j} \otimes \mathbf{I}_{c_{in}} \right) \tag{6.12}$$

Taking the spectral norm of both sides and using the same procedure that we used for $\mathbf{R}$, we get the following inequality:

$$\left\| \mathbf{G}^{(j,k)} \right\|_2 \leq \sqrt{hw} \left\| \mathbf{S} \right\|_2$$

Alternative inequality (2): Using (6.7), $\mathbf{G}^{(j,k)}$ can alternatively be written as follows:

$$\mathbf{G}_{c,d}^{(j,k)} = \sum_{l=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{L}_{c,d,l,m} \mathbf{F}_{l,j} \mathbf{F}_{m,k}$$
$$\mathbf{G}^{(j,k)} = \sum_{l=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{L}_{:,:,l,m} \mathbf{F}_{l,j} \mathbf{F}_{m,k} \tag{6.13}$$

Now consider a block matrix $\mathbf{T}$ of dimensions $c_{out} \times c_{in}hw$ given by:

$$\mathbf{T} = \begin{bmatrix} \mathbf{A}_0 & \mathbf{A}_1 & \cdots & \mathbf{A}_{h-1} \end{bmatrix} \tag{6.14}$$

where each block A_l is a matrix of dimensions $c_{out} \times c_{in}w$ given as follows:

$$\mathbf{A}_l = \begin{bmatrix} \mathbf{L}_{:,:,l,0} & \mathbf{L}_{:,:,l,1} & \cdots & \mathbf{L}_{:,:,l,w-1} \end{bmatrix} \tag{6.15}$$

For $\mathbf{T}$, the block in l^{th} column is the $c_{out} \times c_{in}w$ matrix $\mathbf{A}_l$. For $\mathbf{A}_l$, the block in the mth column is the $c_{out} \times c_{in}$ matrix $\mathbf{L}_{:,:,l,m}$.

Consider the matrix: $\mathbf{F}_{:w,k} \otimes \mathbf{F}_{:h,j} \otimes \mathbf{I}_{c_{in}}$ (of dimensions $c_{in}hw \times c_{in}$).

Using (6.13), (6.14) and (6.15), we again have:

$$\mathbf{G}^{(j,k)} = \mathbf{T} \left(\mathbf{F}_{:h,j} \otimes \mathbf{F}_{:w,k} \otimes \mathbf{I}_{c_{in}} \right) \tag{6.16}$$

Taking the spectral norm of both sides and using the same procedure that we used for $\mathbf{R}$ and $\mathbf{S}$,

we get the following inequality:

$$\left\|\mathbf{G}^{(j,k)}\right\|_2 \leq \sqrt{hw}\,\|\mathbf{T}\|_2$$

Alternative inequality (3): Using (6.7), $\mathbf{G}^{(j,k)}$ can alternatively be written as follows:

$$\begin{aligned}\mathbf{G}^{(j,k)}_{c,d} &= \sum_{l=0}^{h-1}\sum_{m=0}^{w-1}\mathbf{F}_{l,j}\mathbf{F}_{m,k}\mathbf{L}_{c,d,l,m}\\ \mathbf{G}^{(j,k)} &= \sum_{l=0}^{h-1}\sum_{m=0}^{w-1}\mathbf{F}_{l,j}\mathbf{F}_{m,k}\mathbf{L}_{:,:,l,m}\end{aligned} \tag{6.17}$$

Now consider a block matrix $\mathbf{U}$ of dimensions $c_{out}hw \times c_{in}$ given by:

$$\mathbf{U} = \begin{bmatrix}\mathbf{B}_0\\ \mathbf{B}_1\\ \vdots\\ \mathbf{B}_{h-1}\end{bmatrix} \tag{6.18}$$

where each block $\mathbf{B}_l$ is a matrix of dimensions $c_{out}w \times c_{in}$ given as follows:

$$\mathbf{B}_l = \begin{bmatrix}\mathbf{L}_{:,:,l,0}\\ \mathbf{L}_{:,:,1,1}\\ \vdots\\ \mathbf{L}_{:,:,l,w-1}\end{bmatrix} \tag{6.19}$$

For $\mathbf{U}$, the block in l^{th} row is the $c_{out}w \times c_{in}$ matrix $\mathbf{B}_l$. For $\mathbf{B}_l$, the block in the mth row is the $c_{out} \times c_{in}$ matrix $\mathbf{L}_{:,:,l,m}$.

Consider the matrix: $\left((\mathbf{F}_{:h,j})^T \otimes (\mathbf{F}_{:w,k})^T \otimes \mathbf{I}_{c_{out}}\right)$ (of dimensions $c_{out} \times c_{out}hw$).

Using (6.17), (6.18) and (6.19), we again have:

$$\mathbf{G}^{(j,k)} = \left((\mathbf{F}_{:h,j})^T \otimes (\mathbf{F}_{:w,k})^T \otimes \mathbf{I}_{c_{out}}\right) \mathbf{U} \tag{6.20}$$

Taking the spectral norm of both sides and using the same procedure that we used for $\mathbf{R}, \mathbf{S}$ and $\mathbf{T}$, we get the following inequality:

$$\left\|\mathbf{G}^{(j,k)}\right\|_2 \leq \sqrt{hw}\, \|\mathbf{U}\|_2$$

Taking the minimum of the four bounds i.e $\sqrt{hw}\min(\|\mathbf{R}\|_2, \|\mathbf{S}\|_2, \|\mathbf{T}\|_2, \|\mathbf{U}\|_2)$, we have the stated result. □

6.C.2 Proof of Lemma 4

Proof. When $h = 1$ and $w = 1$, note that in equations 6.9, 6.12, 6.16 and 6.20, we have:

$$\mathbf{F}_{:h,j} = 1, \qquad \mathbf{F}_{:w,k} = 1$$

This directly implies: $\|\mathbf{J}\|_2 = \|\mathbf{R}\|_2 = \|\mathbf{S}\|_2 = \|\mathbf{T}\|_2 = \|\mathbf{U}\|_2$ □

6.D Additional analysis

6.D.1 Computing the gradient of convolution layer

Consider a 1D convolution filter of size 3 given by $([1, 2, -1])$ applied to an array of size 5. The corresponding Jacobian matrix ($\mathbf{J}$) is given by:

$$\mathbf{J} = \begin{bmatrix} 1 & 2 & -1 & 0 & 0 \\ 0 & 1 & 2 & -1 & 0 \\ 0 & 0 & 1 & 2 & -1 \\ -1 & 0 & 0 & 1 & 2 \\ 2 & -1 & 0 & 0 & 1 \end{bmatrix}$$

The largest singular value $(\|\mathbf{J}\|_2)$, the left and right singular vectors ($\mathbf{u},\ \mathbf{v}$) are given by:

$$\|\mathbf{J}\|_2 = 2.76008, \qquad \mathbf{u} = \begin{bmatrix} -0.55614 \\ 0.11457 \\ 0.62695 \\ 0.27290 \\ -0.45829 \end{bmatrix}, \qquad \mathbf{v} = \begin{bmatrix} -0.63245 \\ -0.19544 \\ 0.51167 \\ 0.51167 \\ -0.19544 \end{bmatrix}$$

The gradient $(\nabla_{\mathbf{J}} \|\mathbf{J}\|_2)$ is given as follows:

$$\nabla_{\mathbf{J}} \|\mathbf{J}\|_2 = \begin{bmatrix} 0.35174 & 0.10870 & -0.28456 & -0.28456 & 0.10870 \\ -0.07246 & 0.02239 & 0.05862 & 0.05862 & -0.02239 \\ -0.39652 & -0.12253 & 0.32079 & 0.32079 & -0.12253 \\ -0.17260 & -0.05333 & 0.13964 & 0.13964 & -0.05334 \\ 0.28984 & 0.08957 & -0.23449 & -0.23449 & 0.08957 \end{bmatrix}$$

Clearly the gradient contains non-zero elements in elements of $\mathbf{J}$ that are always zero and unequal gradient values at elements of $\mathbf{J}$ that are always equal.

6.D.2 Effect of increasing β on singular values

In Figure 6.1, we plot the effect of increasing β on the sum of true singular values of the network and sum of our upper bound. We observe that the gap between the two decreases as we increase β. In Table 6.6, we show the effect of increasing β on the bound on the largest singular value of each layer.

6.D.3 Effect of increasing γ on certified robust accuracy

In Table 6.7, we show the effect of increasing γ on certified robust accuracy.

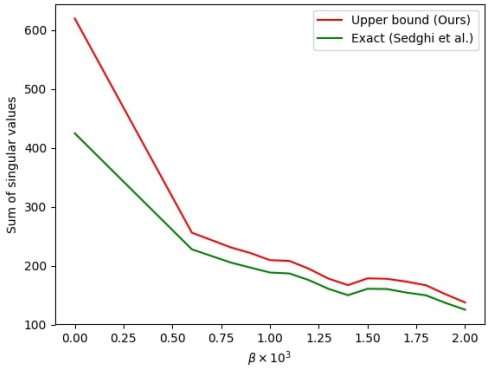


Figure 6.1: We plot the effect of increasing β on the sum of the bounds (computed using our method i.e $\sqrt{hw}\|\mathbf{R}\|_2$) and sum of true spectral norms ($\|\mathbf{J}^{(I)}\|_2$ computed using [214]) for a Resnet-32 neural network trained on CIFAR-10 dataset (without any weight decay). We observe that the gap between the two decreases as we increase β. In Table 6.6, we show the effect of increasing β on the bounds of individual layers of the same network.)

Filter shape	β values							
	0	0.0006	0.0008	0.001	0.0012	0.0014	0.0016	0.0018
$16 \times 3 \times 3 \times 3$	37.96	11.05	9.13	8.2	7.28	8.66	7.06	5.57
$16 \times 16 \times 3 \times 3$	22.23	5.89	5.08	5.26	5.19	6.17	3.41	3.27
$16 \times 16 \times 3 \times 3$	25.33	5.69	4.87	4.9	4.75	5.80	3.26	3.27
$16 \times 16 \times 3 \times 3$	25.29	5.28	4.74	4.08	2.89	6.60	3.72	2.32
$16 \times 16 \times 3 \times 3$	21.66	4.99	4.36	3.64	2.64	5.81	3.61	2.25
$16 \times 16 \times 3 \times 3$	20.15	4.92	3.89	4.61	2.53	3.13	0.07	2.4
$16 \times 16 \times 3 \times 3$	16.36	4.37	3.52	4.08	2.34	2.60	0.07	2.14
$16 \times 16 \times 3 \times 3$	15.39	5.39	5.65	3.15	4.1	4.72	2.31	1.69
$16 \times 16 \times 3 \times 3$	14.67	4.38	4.61	2.32	3.52	3.66	2.37	1.57
$16 \times 16 \times 3 \times 3$	19.26	7.41	6.28	4.96	3.57	0.13	5.38	3.19
$16 \times 16 \times 3 \times 3$	14.60	5.77	4.84	4.01	2.68	0.09	4.31	2.64
$32 \times 16 \times 3 \times 3$	23.24	8.53	8.09	7.06	6.24	6.30	6.4	6.61
$32 \times 32 \times 3 \times 3$	22.19	9.65	9.57	7.99	7.27	7.12	7.72	7.82
$32 \times 32 \times 3 \times 3$	17.97	7.61	7.52	6.32	4.72	5.25	5.06	2.57
$32 \times 32 \times 3 \times 3$	16.76	6.83	6.29	5.38	4.12	4.48	4.48	2.22
$32 \times 32 \times 3 \times 3$	17.62	8.31	6.2	6.73	4.06	5.23	5.65	3.21
$32 \times 32 \times 3 \times 3$	15.7	7.24	5.23	5.5	3.42	4.45	4.73	2.8
$32 \times 32 \times 3 \times 3$	15.82	8.34	5.42	5.29	4.86	5.41	5.03	4.11
$32 \times 32 \times 3 \times 3$	15.83	6.99	4.48	4.47	4.02	4.62	4.24	3.56
$32 \times 32 \times 3 \times 3$	18.46	8.1	6.03	5.31	5.91	5.64	4.83	6.24
$32 \times 32 \times 3 \times 3$	17.75	7.08	5.17	4.53	4.98	4.61	4.15	5.31
$64 \times 32 \times 3 \times 3$	23.57	12.21	10.97	10.41	10.84	10.15	9.7	10.31
$64 \times 64 \times 3 \times 3$	22.97	13.2	12.23	11.72	12.55	11.36	11.14	12.39
$64 \times 64 \times 3 \times 3$	22.35	12.18	11.71	11.13	10.53	10.57	10.2	10.43
$64 \times 64 \times 3 \times 3$	22.02	11.28	10.95	10.43	9.94	9.94	9.52	9.55
$64 \times 64 \times 3 \times 3$	20.91	12.09	11.7	11.07	10.47	10.85	9.87	9.81
$64 \times 64 \times 3 \times 3$	21.77	10.85	10.51	9.81	9.6	9.62	8.56	8.43
$64 \times 64 \times 3 \times 3$	19.63	11.97	11.56	10.28	10.98	10.02	9.32	8.69
$64 \times 64 \times 3 \times 3$	17.61	10	10.03	8.81	9.63	8.26	7.57	7.35
$64 \times 64 \times 3 \times 3$	16.67	10.34	11.61	10.26	10.93	8.64	8.05	8.56
$64 \times 64 \times 3 \times 3$	17.81	8.16	8.73	7.68	8.18	6.61	6.07	6.49
10×64	12.43	10.79	10.55	10.53	10.11	10.14	9.64	9.39

Table 6.6: Effect of increasing β on the bounds ($\sqrt{hw}\|\mathbf{R}\|_2$) of each layer

γ	MNIST			CIFAR-10		
	Standard Accuracy	Empirical Robust Accuracy	Certified Robust Accuracy	Standard Accuracy	Empirical Robust Accuracy	Certified Robust Accuracy
0	98.68%	87.81%	0.00%	56.22%	14.88%	0.00%
0.01	97.08%	92.92%	**91.25%**	53.52%	31.82%	17.39%
0.02	96.36%	90.98%	89.58%	49.55%	31.80%	25.93%
0.03	95.54%	89.99%	88.75%	46.56%	31.98%	**29.26%**

Table 6.7: Comparison between certified robust accuracy for different values of the regularization parameter γ for a single hidden layer convolutional neural network with softplus activation function. Certified Robust Accuracy is computed as the fraction of correctly classified samples with CRC (Curvature-based Robustness Certificate as defined in [142]) greater than 0.5. Empirical robust accuracy is computed by running l_2 bounded PGD (200 steps, step size 0.01). We use convolution layer with 64 filters and stride of size 2.

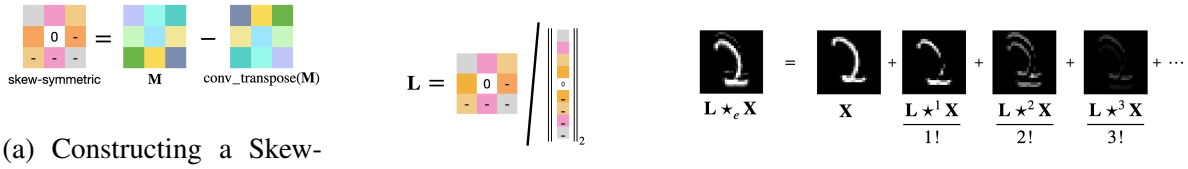
Chapter 7: Skew Orthogonal Convolutions

7.1 Introduction

The Lipschitz constant[2] of a neural network puts an upper bound on how much the output is allowed to change in proportion to a change in input. Previous work has shown that a small Lipschitz constant leads to improved generalization bounds [205, 227], adversarial robustness [15, 228] and interpretable gradients [229]. The Lipschitz constant also upper bounds the change in gradient norm during backpropagation and can thus prevent gradient explosion during training, allowing us to train very deep networks [203]. Moreover, the Wasserstein distance between two probability distributions can be expressed as a maximization over 1-Lipschitz functions [230, 231], and has been used for training Wasserstein GANs [206, 207] and Wasserstein VAEs [208].

Using the Lipschitz composition property $\big($i.e. $Lip(f \circ g) \leq Lip(f)Lip(g)\big)$, a Lipschitz constant of a neural network can be bounded by the product of the Lipschitz constant of all layers. 1-Lipschitz neural networks can thus be designed by imposing a 1-Lipschitz constraint on each layer. However, [133] identified a key difficulty with this approach: because a layer with a Lipschitz bound of 1 can only reduce the norm of the gradient during backpropagation, each step of backprop gradually attenuates the gradient norm, resulting in a much smaller gradient for the layers closer to the input, thereby making training slow and difficult. To address this problem,

[2]Unless specified, we use Lipschitz constant under the l_2 norm.



(a) Constructing a Skew-Symmetric conv. filter (b) Spectral normalization (c) Convolution exponential $(\mathbf{L} \star_e \mathbf{X})$

Figure 7.1: Each color denotes a scalar, the minus sign $(-)$ on top of some color denotes the negative of the scalar with that color. Given any convolution filter $\mathbf{M}$, we can construct a Skew-Symmetric filter (Figure 7.1a). Next, we apply spectral normalization to bound the norm of the Jacobian (Figure 7.1b). On input $\mathbf{X}$, applying convolution exponential $(\mathbf{L} \star_e \mathbf{X})$ results in an Orthogonal convolution (Figure 7.1c).

they introduced Gradient Norm Preserving (GNP) architectures where each layer preserves the gradient norm by ensuring that the Jacobian of each layer is an *Orthogonal* matrix (for all inputs). For convolutional layers, this involves constraining the Jacobian of each convolution layer to be an Orthogonal matrix [203, 232] and using a GNP activation function called GroupSort [133].

[232] introduced an Orthogonal convolution layer called **B**lock **C**onvolutional **O**rthogonal **P**arametrization (BCOP). BCOP uses a clever application of 1D Orthogonal convolution filters of sizes 2×1 and 1×2 to construct a 2D Orthogonal convolution filter. It overcomes common issues of Lipschitz-constrained networks such as gradient norm attenuation and loose lipschitz bounds and enables training of large, provably 1-Lipschitz Convolutional Neural Networks (CNNs) achieving results competitive with existing methods for provable adversarial robustness. However, BCOP suffers from slow training, significant reduction in accuracy and provides no guarantees on its approximation of an Orthogonal Jacobian matrix (details in Section 7.2).

To address these shortcomings, we introduce an Orthogonal convolution layer called **S**kew **O**rthogonal **C**onvolution (SOC). For provably Lipschitz CNNs, SOC results in significantly improved standard and certified robust accuracies compared to BCOP while requiring significantly less training time (Table 7.2). We also derive provable guarantees on our approximation of the

Orthogonal Jacobian matrix.

Our work is based on the following key mathematical property: If $\mathbf{A}$ is a Skew-Symmetric matrix (i.e. $\mathbf{A} = -\mathbf{A}^T$), $\exp(\mathbf{A})$ is an Orthogonal matrix (i.e. $\exp(\mathbf{A})^T \exp(\mathbf{A}) = \exp(\mathbf{A}) \exp(\mathbf{A})^T = \mathbf{I}$) where

$$\exp(\mathbf{A}) = \mathbf{I} + \frac{\mathbf{A}}{1!} + \frac{\mathbf{A}^2}{2!} + \frac{\mathbf{A}^3}{3!} \cdots = \sum_{i=0}^{\infty} \frac{\mathbf{A}^i}{i!}. \tag{7.1}$$

To design an Orthogonal convolution layer using this property, we need to: (a) construct *Skew-Symmetric filters*, i.e. filters whose Jacobian is Skew-Symmetric; and (b) efficiently approximate $\exp(\mathbf{J})$ with a guaranteed small error where $\mathbf{J}$ is the Jacobian of a Skew-Symmetric filter.

To construct Skew-Symmetric convolution filters, we prove (in Theorem 13) that every Skew-Symmetric filter $\mathbf{L}$ can be written as $\mathbf{L} = \mathbf{M} - \text{conv_transpose}(\mathbf{M})$ for some filter $\mathbf{M}$ where conv_transpose represents the *convolution transpose operator* defined in equation (7.3) (note that this is different from the matrix transpose). This result is analogous to the property that every real Skew-Symmetric matrix $\mathbf{A}$ can be written as $\mathbf{A} = \mathbf{B} - \mathbf{B}^T$ for some real matrix $\mathbf{B}$.

We can efficiently approximate $\exp(\mathbf{J})$ using a finite number of terms in equation (7.1) and the convolution exponential [209]. But it is unclear whether the series can be approximated with high precision and how many terms need to be computed to achieve the desired approximation error. To resolve these issues, we derive a bound on the l_2 norm of the difference between $\exp(\mathbf{J})$ and its approximation using the first k terms in equation (7.1), called $\mathbf{S}_k(\mathbf{J})$ when $\mathbf{J}$ is Skew-Symmetric (Theorem 14):

$$\| \exp(\mathbf{J}) - \mathbf{S}_k(\mathbf{J}) \|_2 \leq \frac{\|\mathbf{J}\|_2^k}{k!}. \tag{7.2}$$

This guarantee suggests that when $\|\mathbf{J}\|_2$ is small, $\exp(\mathbf{J})$ can be approximated with high precision using a small number of terms. Also, the factorial term in denominator causes the error to decay very fast as k increases. In our experiments, we observe that using $k = 12,\ \|\mathbf{J}\|_2 \leq 1.8$ leads to an error bound of 2.415×10^{-6}. We can use spectral normalization [196] to ensure $\|\mathbf{J}\|_2$ is provably bounded using the theoretical result of [233]. The design of SOC is summarized in Figure 7.1. Code is available at `https://github.com/singlasahil14/SOC`.

In summary, we make the following contributions:

- We introduce an Orthogonal convolution layer (called **S**kew **O**rthogonal **C**onvolution or SOC) by designing a Skew-Symmetric convolution filter (Theorem 13) and computing the exponential function of its Jacobian using a finite number of terms in its Taylor series.
- For a Skew-Symmetric filter with Jacobian $\mathbf{J}$, we derive a bound on the approximation error between $\exp(\mathbf{J})$ and its k-term approximation (Theorem 14).
- SOC achieves significantly higher standard and provable robust accuracy on 1-Lipschitz convolutional neural networks than BCOP while requiring less training time (Table 7.2.) For example, SOC achieves 2.82% higher standard and 3.91% higher provable robust accuracy with 54.6% less training time on CIFAR-10 using the $\mathrm{LipConvnet}-20$ architecture (details in Section 7.6.5). For deeper networks (≥ 30 layers), SOC outperforms BCOP with an improvement of $\geq 10\%$ on both standard and robust accuracy again achieving $\geq 50\%$ reduction in the training time.
- In Theorem 15, we prove that for every Skew-Symmetric filter with Jacobian $\mathbf{J}$, there exists Skew-Symmetric $\mathbf{B}$ satisfying: $\exp(\mathbf{B}) = \exp(\mathbf{J}),\ \|\mathbf{B}\|_2 \leq \pi$. Since $\|\mathbf{J}\|_2$ can be large, this allows us to reduce approximation error without sacrificing the expressive power.

7.2 Related work

Provably lipschitz convolutional neural networks: [133] proposed a class of fully connected neural networks (FCNs) which are Gradient Norm Preserving (GNP) and provably 1-Lipschitz using the GroupSort activation and Orthogonal weight matrices. Since then, there have been numerous attempts to tightly enforce 1-Lipschitz constraints on convolutional neural networks (CNNs) [134, 214, 225, 228, 234]. However, these approaches either enforce loose lipschitz bounds or are computationally intractable for large networks. [232] introduced an Orthogonal convolution layer called **B**lock **C**onvolutional **O**rthogonal **P**arametrization (BCOP) that avoids the aforementioned issues and allows the training of large, provably 1-Lipschitz CNNs while achieving provable robust accuracy comparable with the existing methods. However, it suffers from some issues: (a) it can only represent a subset of all Orthogonal convolutions, (b) it requires a BCOP convolution filter with $2n$ channels to represent all the connected components of a BCOP convolution filter with n channels thus requiring 4 times more parameters, (c) to construct a convolution filter with size $k \times k$ and n input/output channels, it requires $2k - 1$ matrices of size $2n \times 2n$ that must remain Orthogonal throughout training; resulting in well known difficulties of optimization over the Stiefel manifold [235], (d) it constructs convolution filters from symmetric projectors and error in these projectors can lead to an error in the final convolution filter whereas BCOP does not provide guarantees on the error.

Provable defenses against adversarial examples: A classifier is said to be provably robust if one can guarantee that a classifier's prediction remains constant within some region around the input. Most of the existing methods for provable robustness either bound the Lipschitz constant of the neural network or the individual layers [142, 145, 146, 166, 182, 187, 236, 237, 238].

However, these methods do not scale to large and practical networks on ImageNet. To scale to such large networks, randomized smoothing [135, 136, 137, 190, 191, 192, 239, 240] has been proposed as a *probabilistically certified defense*. In contrast, the defense we propose in this chapter is deterministic and hence not directly comparable to randomized smoothing.

7.3 Notation

For a vector $\mathbf{v}$, $\mathbf{v}_j$ denotes its jth element. For a matrix $\mathbf{A}$, $\mathbf{A}_{j,:}$ and $\mathbf{A}_{:,k}$ denote the jth row and k^{th} column respectively. Both $\mathbf{A}_{j,:}$ and $\mathbf{A}_{:,k}$ are assumed to be column vectors (thus $\mathbf{A}_{j,:}$ is the transpose of jth row of $\mathbf{A}$). $\mathbf{A}_{j,k}$ denotes the element in j^{th} row and k^{th} column of $\mathbf{A}$. $\mathbf{A}_{:j,:k}$ denotes the matrix containing the first j rows and k columns of $\mathbf{A}$. The same rules are directly extended to higher order tensors. Bold zero (i.e. $\mathbf{0}$) denotes the matrix (or tensor) consisting of zero at all elements and $\mathbf{I}$ denotes the identity matrix. $\otimes$ denotes the kronecker product. We use $\mathbb{C}$ to denote the field of complex numbers and $\mathbb{R}$ for real numbers. For a scalar $a \in \mathbb{C}$, $\overline{a}$ denotes its complex conjugate. For a matrix (or tensor) $\mathbf{A}$, $\overline{\mathbf{A}}$ denotes the element-wise complex conjugate. For $\mathbf{A} \in \mathbb{C}^{m \times n}$, $\mathbf{A}^H$ denotes the Hermitian transpose (i.e. $\mathbf{A}^H = \overline{\mathbf{A}^T}$). For $a \in \mathbb{C}$, $\mathrm{Re}(a)$, $\mathrm{Im}(a)$ and $|a|$ denote the real part, imaginary part and modulus of a, respectively. We use ι to denote the imaginary part *iota* (i.e. $\iota^2 = -1$).

For a matrix $\mathbf{A} \in \mathbb{C}^{q \times r}$ and a tensor $\mathbf{B} \in \mathbb{C}^{p \times q \times r}$, $\overrightarrow{\mathbf{A}}$ denotes the vector constructed by stacking the rows of $\mathbf{A}$ and $\overrightarrow{\mathbf{B}}$ by stacking the vectors $\overrightarrow{\mathbf{B}_{j,:,:}}$, $j \in [p-1]$ so that:

$$\left(\overrightarrow{\mathbf{A}}\right)^T = \left[\mathbf{A}_{0,:}^T\ ,\ \mathbf{A}_{1,:}^T\ ,\ \ldots\ ,\ \mathbf{A}_{q-1,:}^T\right]$$
$$\left(\overrightarrow{\mathbf{B}}\right)^T = \left[\left(\overrightarrow{\mathbf{B}_{0,:,:}}\right)^T\ ,\ \left(\overrightarrow{\mathbf{B}_{1,:,:}}\right)^T\ ,\ \ldots\ ,\ \left(\overrightarrow{\mathbf{B}_{p-1,:,:}}\right)^T\right]$$

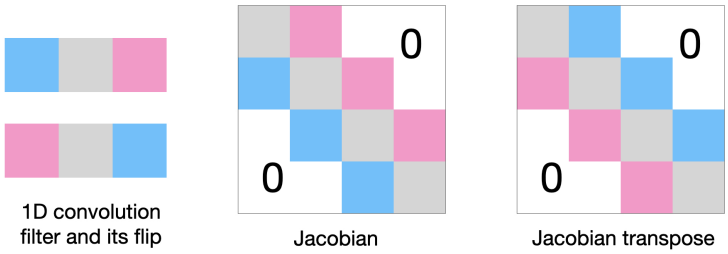


Figure 7.2: Each color denotes a scalar. Flipping a conv. filter (of odd size) transposes its Jacobian. Thus, any odd-sized filter that equals the negative of its flip leads to a Skew-Symmetric Jacobian.

For a 2D convolution filter, $\mathbf{L} \in \mathbb{C}^{p \times q \times r \times s}$, we define the tensor $\text{conv_transpose}(\mathbf{L}) \in \mathbb{C}^{q \times p \times r \times s}$ as follows:

$$[\text{conv_transpose}(\mathbf{L})]_{i,j,k,l} = \overline{[\mathbf{L}]}_{j,i,r-1-k,s-1-l} \tag{7.3}$$

Note that this is very different from the usual matrix transpose. See an example in Section 7.4. Given an input $\mathbf{X} \in \mathbb{C}^{q \times n \times n}$, we use $\mathbf{L} \star \mathbf{X} \in \mathbb{C}^{p \times n \times n}$ to denote the convolution of filter $\mathbf{L}$ with $\mathbf{X}$. We use the the notation $\mathbf{L} \star^i \mathbf{X} \triangleq \mathbf{L} \star^{i-1} (\mathbf{L} \star \mathbf{X})$. Unless specified, we assume zero padding and stride 1 in each direction.

7.4 Filters with Skew Symmetric Jacobians

We know that for any matrix $\mathbf{A}$ that is Skew-Symmetric ($\mathbf{A} = -\mathbf{A}^T$), $\exp(\mathbf{A})$ is Orthogonal:

$$\exp(\mathbf{A}) \left(\exp(\mathbf{A})\right)^T = \left(\exp(\mathbf{A})\right)^T \exp(\mathbf{A}) = \mathbf{I}$$

This suggests that if we can parametrize the complete set of convolution filters with Skew-Symmetric Jacobians, we can use the convolution exponential [209] to approximate an Orthogo-

nal matrix. To construct this set, we first prove that, if convolution using filter $\mathbf{L} \in \mathbb{R}^{m\times m\times p\times q}$ (p and q are odd) has Jacobian $\mathbf{J}$, the convolution using $\mathrm{conv_transpose}(\mathbf{L})$ results in Jacobian $\mathbf{J}^T$. To motivate our proof, consider filter $\mathbf{L} \in \mathbb{R}^{1\times 1\times 3\times 3}$. Applying $\mathrm{conv_transpose}$ (eqn. (7.3)):

$$\mathbf{L} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, \quad \mathrm{conv_transpose}\left(\mathbf{L}\right) = \begin{bmatrix} i & h & g \\ f & e & d \\ c & b & a \end{bmatrix}$$

That is, for a $2\mathrm{D}$ convolution filter with 1 channel, $\mathrm{conv_transpose}$ flips it along the horizontal and vertical directions. To understand why this flipping transposes the Jacobian, we provide another example for a $1\mathrm{D}$ convolution filter in Figure 7.2. Our proof uses the following expression for the Jacobian of convolution using a filter $\mathbf{L} \in \mathbb{R}^{1\times 1\times (2p+1)\times (2q+1)}$ and input $\mathbf{X} \in \mathbb{R}^{1\times n\times n}$:

$$\mathbf{J} = \sum_{i=-p}^{p} \sum_{j=-q}^{q} \mathbf{L}_{0,0,p+i,q+j} \left(\mathbf{P}^{(i)} \otimes \mathbf{P}^{(j)}\right)$$

where $\mathbf{P}^{(k)} \in \mathbb{R}^{n\times n}$, $\mathbf{P}^{(k)}_{i,j} = 1$ if $i - j = k$ and 0 otherwise. The above equation leads to the following theorem:

Theorem 12. *Consider a* $2\mathrm{D}$ *convolution filter* $\mathbf{L} \in \mathbb{R}^{m\times m\times (2p+1)\times (2q+1)}$ *and input* $\mathbf{X} \in \mathbb{R}^{m\times n\times n}$. *Let* $\mathbf{J} = \nabla_{\overrightarrow{\mathbf{X}}} \overrightarrow{(\mathbf{L} \star \mathbf{X})}$, *then* $\mathbf{J}^T = \nabla_{\overrightarrow{\mathbf{X}}} \overrightarrow{(\mathrm{conv_transpose}(\mathbf{L}) \star \mathbf{X})}$.

Next, we prove that any $2\mathrm{D}$ convolution filter $\mathbf{L}$ whose Jacobian is Skew-Symmetric can be expressed as: $\mathbf{L} = \mathbf{M} - \mathrm{conv_transpose}(\mathbf{M})$ where $\mathbf{M}$ has the same dimensions as $\mathbf{L}$. This allows us to parametrize the set of all convolution filters with Skew-Symmetric Jacobian matrices.

Theorem 13. *Consider a* $2\mathrm{D}$ *convolution filter* $\mathbf{L} \in \mathbb{R}^{m\times m\times (2p+1)\times (2q+1)}$ *and input* $\mathbf{X} \in \mathbb{R}^{m\times n\times n}$.

The Jacobian $\nabla_{\overrightarrow{\mathbf{X}}} \overrightarrow{(\mathbf{L} \star \mathbf{X})}$ *is Skew-Symmetric if and only if:*

$$\mathbf{L} = \mathbf{M} - \text{conv_transpose}(\mathbf{M})$$

for some filter $\mathbf{M} \in \mathbb{R}^{m \times m \times (2p+1) \times (2q+1)}$.

We prove Theorems 12 and 13 for the more general case of complex convolution filters ($\mathbf{L}_{i,j,k,l} \in \mathbb{C}$) in Supp. Sections 7.B.1 and 7.B.2. Theorem 13 allow us to convert any arbitrary convolution filter into a filter with a Skew-Symmetric Jacobian. This leads to the following definition:

Definition 7. ***(Skew-Symmetric Convolution Filter)*** *A convolution filter* $\mathbf{L} \in \mathbb{R}^{m \times m \times (2p+1) \times (2q+1)}$ *is said to be Skew-Symmetric if given an input* $\mathbf{X} \in \mathbb{R}^{m \times n \times n}$*, the Jacobian matrix* $\nabla_{\overrightarrow{\mathbf{X}}} \overrightarrow{(\mathbf{L} \star \mathbf{X})}$ *is Skew-Symmetric.*

We note that although Theorem 13 requires the height and width of $\mathbf{M}$ to be odd integers, we can also construct a Skew-Symmetric filter when $\mathbf{M}$ has even height/width by zero padding $\mathbf{M}$ to make the desired dimensions odd.

7.5 Skew Orthogonal Convolution layers

In this section, we derive a method to approximate the exponential of the Jacobian of a Skew-Symmetric convolution filter (i.e. $\exp(\mathbf{J})$). We also derive a bound on the approximation error. Given an input $\mathbf{X} \in \mathbb{R}^{m \times n \times n}$ and a Skew-Symmetric convolution filter $\mathbf{L} \in \mathbb{R}^{m \times m \times k \times k}$ (k is

odd), let $\mathbf{J}$ be the Jacobian of convolution filter $\mathbf{L}$ so that:

$$\mathbf{J}\overrightarrow{\mathbf{X}} = \overrightarrow{(\mathbf{L} \star \mathbf{X})} \tag{7.4}$$

By construction, we know that $\mathbf{J}$ is a Skew-Symmetric matrix, thus $\exp(\mathbf{J})$ is an Orthogonal matrix. We are interested in computing $\exp(\mathbf{J})\,\overrightarrow{\mathbf{X}}$ efficiently where:

$$\exp(\mathbf{J})\,\overrightarrow{\mathbf{X}} = \overrightarrow{\mathbf{X}} + \frac{\mathbf{J}\overrightarrow{\mathbf{X}}}{1!} + \frac{\mathbf{J}^2\overrightarrow{\mathbf{X}}}{2!} + \frac{\mathbf{J}^3\overrightarrow{\mathbf{X}}}{3!} + \cdots$$

Using equation (7.4), the above expression can be written as:

$$\exp(\mathbf{J})\,\overrightarrow{\mathbf{X}} = \overrightarrow{\mathbf{X}} + \frac{\overrightarrow{\mathbf{L} \star \mathbf{X}}}{1!} + \frac{\overrightarrow{\mathbf{L} \star^2 \mathbf{X}}}{2!} + \frac{\overrightarrow{\mathbf{L} \star^3 \mathbf{X}}}{3!} + \cdots$$

where the notation $\mathbf{L} \star^i \mathbf{X} \triangleq \mathbf{L} \star^{i-1} (\mathbf{L} \star \mathbf{X})$. Using the above equation, we define $\mathbf{L} \star_e \mathbf{X}$:

$$\mathbf{L} \star_e \mathbf{X} = \mathbf{X} + \frac{\mathbf{L} \star \mathbf{X}}{1!} + \frac{\mathbf{L} \star^2 \mathbf{X}}{2!} + \frac{\mathbf{L} \star^3 \mathbf{X}}{3!} + \cdots \tag{7.5}$$

The above operation is called *convolution exponential*, and was introduced by [209]. By construction, $\mathbf{L} \star_e \mathbf{X}$ satisfies: $\exp(\mathbf{J})\,\overrightarrow{\mathbf{X}} = \overrightarrow{\mathbf{L} \star_e \mathbf{X}}$. Thus, the Jacobian of $\overrightarrow{\mathbf{L} \star_e \mathbf{X}}$ with respect to $\overrightarrow{\mathbf{X}}$ is equal to $\exp(\mathbf{J})$ which is Orthogonal (since $\mathbf{J}$ is Skew-Symmetric). However, $\mathbf{L} \star_e \mathbf{X}$ can only be approximated using a finite number of terms in the series given in equation (7.5). Thus, we need to bound the error of such an approximation.

7.5.1 Bounding the Approximation Error

To bound the approximation error using a finite number of terms, first note that since the Jacobian matrix $\mathbf{J}$ is Skew-Symmetric, all the eigenvalues are purely imaginary. For a purely imaginary scalar $\lambda \in \mathbb{C}$ $\big($i.e. $\mathrm{Re}(\lambda) = 0\big)$, we first bound the error between $\exp(\lambda)$ and approximation $p_k(\lambda)$ computed using k terms of the exponential series as follows:

$$\big|\exp(\lambda) - p_k(\lambda)\big| \leq \frac{|\lambda|^k}{k!}, \qquad \forall\, \lambda : \mathrm{Re}(\lambda) = 0 \tag{7.6}$$

The above result then allows us to prove the following result for a Skew-Symmetric matrix in a straightforward manner:

Theorem 14. *For Skew-Symmetric* $\mathbf{J}$*, we have the inequality:*

$$\|\exp(\mathbf{J}) - \mathbf{S}_k(\mathbf{J})\|_2 \leq \frac{\|\mathbf{J}\|_2^k}{k!} \quad \textit{where } \mathbf{S}_k(\mathbf{J}) = \sum_{i=0}^{k-1} \frac{\mathbf{J}^i}{i!}$$

A more general proof of Theorem 14 (for $\mathbf{J} \in \mathbb{C}^{n\times n}$ and skew-Hermitian i.e. $\mathbf{J} = -\mathbf{J}^H$) is given in Appendix Section 7.B.3. The above theorem allows us to bound the approximation error between the true matrix exponential (which is Orthogonal) and its k term approximation as a function of the number of terms (k) and the Jacobian norm $\|\mathbf{J}\|_2$. The factorial term in the denominator causes the error to decay very fast as the number of terms increases. We call the resulting algorithm **S**kew **O**rthogonal **C**onvolution (SOC).

We emphasize that the above theorem is valid only for Skew-Symmetric matrices and hence not directly applicable for the more general convolution exponential [209].

7.5.2 Complete Set of Skew Orthogonal Convolutions

Observe that for $\mathrm{Re}(\lambda) = 0$, (i.e. $\lambda = \iota\theta,\ \theta \in \mathbb{R}$), we have:

$$\exp(\lambda) = \exp(\lambda + 2\iota\pi k) = \cos(\theta) + \iota \sin(\theta), \quad k \in \mathbb{Z}$$

This suggests that we can shift λ by integer multiples of $2\pi\iota$ without changing $\exp(\lambda)$ while reducing the approximation error (using Theorem 14). For example, $\exp(\iota\pi/3)$ requires fewer terms to achieve the desired approximation (using equation (7.6)) than say $\exp(\iota(\pi/3 + 2\pi))$ because the latter has higher norm (i.e. $2\pi + \pi/3 = 7\pi/3$) than the former (i.e. $\pi/3$). This insight leads to the following theorem:

Theorem 15. *Given a real Skew-Symmetric matrix* $\mathbf{A}$*, we can construct another real Skew-Symmetric matrix* $\mathbf{B}$ *such that* $\mathbf{B}$ *satisfies: (i)* $\exp(\mathbf{A}) = \exp(\mathbf{B})$ *and (ii)* $\|\mathbf{B}\|_2 \leq \pi$.

A proof is given in Appendix Section 7.B.4. This proves that every real Skew-Symmetric Jacobian matrix $\mathbf{J}$ (associated with some Skew-Symmetric convolution filter $\mathbf{L}$) can be replaced with a Skew-Symmetric Jacobian $\mathbf{B}$ such that $\exp(\mathbf{B}) = \exp(\mathbf{J})$ and $\|\mathbf{B}\|_2 \leq \pi$ $\big($note that $\|\mathbf{J}\|_2$ can be arbitrarily large$\big)$. This strictly reduces the approximation error (Theorem 14) without sacrificing the expressive power.

We make the following observations about Theorem 15: (a) If $\mathbf{J}$ is equal to the Jacobian of some Skew-Symmetric convolution filter, $\mathbf{B}$ may not satisfy this property, i.e. it may not exhibit the block doubly toeplitz structure of the Jacobian of a 2D convolution filter [214] and thus may not equal the jacobian of some Skew-Symmetric convolution filter; (b) even if $\mathbf{B}$ satisfies this property, the filter size of the Skew-Symmetric filter whose Jacobian equals $\mathbf{B}$ can be very

different from that of the filter with Jacobian $\mathbf{J}$.

In this sense, Theorem 15 cannot directly be used to parametrize the complete set of SOC because it is not clear how to efficiently parametrize the set of all matrices $\mathbf{B}$ that satisfy (a) $\|\mathbf{B}\|_2 \leq \pi$ and (b) $\exp(\mathbf{B}) = \exp(\mathbf{J})$ where $\mathbf{J}$ is the Jacobian of some Skew-Symmetric convolution filter. We leave this question of efficient parametrization of Skew Orthogonal Convolution layers open for future research.

7.5.3 Extensions to 3D and Complex Convolutions

When the matrix $\mathbf{A} \in \mathbb{C}^{n \times n}$ is skew-Hermitian ($\mathbf{A} = -\mathbf{A}^H$), then $\exp(\mathbf{A})$ is a unitary matrix:

$$\exp(\mathbf{A}) \left(\exp(\mathbf{A})\right)^H = \left(\exp(\mathbf{A})\right)^H \exp(\mathbf{A}) = \mathbf{I}$$

To use this property to construct a unitary convolution layer with complex weights, we first define:

Definition 8. ***(Skew-Hermitian Convolution Filter)*** *A convolution filter $\mathbf{L} \in \mathbb{C}^{m \times m \times (2p+1) \times (2q+1)}$ is said to be Skew-Hermitian if given an input $\mathbf{X}$, the Jacobian $\nabla_{\vec{\mathbf{X}}} \overrightarrow{(\mathbf{L} \star \mathbf{X})}$ is Skew-Hermitian.*

Using the extensions of Theorems 12 and 13 for complex convolution filters (proofs in Appendix Sections 7.B.1 and 7.B.2), we can construct a 2D Skew-Hermitian convolution filter. Next, using an extension of Theorem 14 for complex Skew-Hermitian matrices (proof in Appendix Section 7.B.3), we can get exactly the same bound on the approximation error. The resulting algorithm is called **S**kew **U**nitary **C**onvolution (SUC). We also prove an extension of Theorem 15 for complex Skew-Hermitian matrices in Appendix Section 7.B.5. We discuss the construction of 3D Skew-Hermitian convolution filters in Appendix Sections 7.B.6 and 7.B.7.

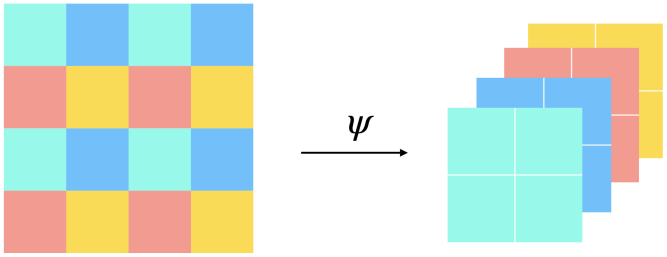


Figure 7.3: Invertible downsampling operation ψ

7.6 Implementation details of SOC

In this section, we explain the key implementation details of SOC .

7.6.1 Bounding the norm of Jacobian

In order to apply the approximation guarantee in Theorem 14, we want to bound the norm of the Jacobian of Skew-Symmetric filter. To this end, we use the following result:

Theorem 7.6.1. *[233] Consider a convolution filter* $\mathbf{L} \in \mathbb{R}^{c_o \times c_i \times h \times w}$ *applied to input* $\mathbf{X}$*. Let* $\mathbf{J}$ *be the Jacobian of* $\mathbf{L} \star \mathbf{X}$ *w.r.t* $\mathbf{X}$*, we have the following inequality:*

$$\|\mathbf{J}\|_2 \leq \sqrt{hw}\ \min\left(\|\mathbf{R}\|_2, \|\mathbf{S}\|_2, \|\mathbf{T}\|_2, \|\mathbf{U}\|_2\right),$$

where $\mathbf{R} \in \mathbb{R}^{c_o h \times c_i w}, \mathbf{S} \in \mathbb{R}^{c_o w \times c_i h}, \mathbf{T} \in \mathbb{R}^{c_o \times c_i hw}, \mathbf{U} \in \mathbb{R}^{c_o hw \times c_i}$ *are obtained by reshaping* $\mathbf{L}$*.*

Using the above theorem, we divide the Skew-Symmetric convolution filter by $\min\left(\|\mathbf{R}\|_2, \|\mathbf{S}\|_2, \|\mathbf{T}\|_2, \|\mathbf{U}\|_2\right)$ so that the spectral norm of the resulting filter is bounded by $\sqrt{hw}$. We next multiply the normalized filter with the hyperparameter, 0.7 as we find that it allows faster convergence with no loss in performance. Unless specified, we use $h = w = 3$ in all of our experiments resulting in the norm bound of 2.1. Note that while the above theorem

also allows us to bound the Lipschitz constant of a convolution layer, for deep networks (say 40 layers), the Lipschitz bound (assuming a 1-Lipschitz activation function) would increase to $2.1^{40} = 7.74 \times 10^{12}$. Thus, this alone is unlikely to enforce a tight global Lipschitz constraint.

7.6.2 Different input and output channels

In general, we may want to construct an orthogonal convolution that maps from c_i input channels to c_o output channels where $c_i \neq c_o$. Consider the two cases:

Case 1 ($c_o < c_i$): We construct a Skew-Symmetric convolution filter with c_i channels. After applying the exponential, we select the first c_o output channels from the output layer.

Case 2 ($c_o > c_i$): We use a Skew-Symmetric convolution filter with c_o channels. We zero pad the input with $c_o - c_i$ channels and then compute the convolution exponential.

7.6.3 Strided convolution

Given an input $\mathbf{X} \in \mathbb{R}^{c_i \times n \times n}$ (n is even), we may want to construct an orthogonal convolution with output $\mathbf{Y} \in \mathbb{R}^{c_o \times (n/2) \times (n/2)}$ (i.e. an orthogonal convolution with stride 2). To perform a strided convolution, we first apply *invertible downsampling* ψ as shown in Figure 7.3 [241] to construct $\mathbf{X}' \in \mathbb{R}^{4c_i \times (n/2) \times (n/2)}$. Next, we apply convolution exponential to $\mathbf{X}'$ using a Skew-Symmetric convolution filter with $4c_i$ input and c_o output channels.

7.6.4 Number of terms for the approximation

During training, we use 6 terms to approximate the exponential function for speed. During evaluation, we use 12 terms to ensure that the exponential of the Jacobian is sufficiently close to being

Output Size	Convolution layer	Repeats
16×16	conv $[3 \times 3, 32, 1]$ conv $[3 \times 3, 64, 2]$	$(n/5) - 1$ 1
8×8	conv $[3 \times 3, 64, 1]$ conv $[3 \times 3, 128, 2]$	$(n/5) - 1$ 1
4×4	conv $[3 \times 3, 128, 1]$ conv $[3 \times 3, 256, 2]$	$(n/5) - 1$ 1
2×2	conv $[3 \times 3, 256, 1]$ conv $[3 \times 3, 512, 2]$	$(n/5) - 1$ 1
1×1	conv $[3 \times 3, 512, 1]$ conv $[1 \times 1, 1024, 2]$	$(n/5) - 1$ 1

Table 7.1: $\mathrm{LipConvnet}$-n Architecture. Each conv. layer is followed by MaxMin activation.

an orthogonal matrix.

7.6.5 Network architecture

We design a provably 1-Lipschitz architecture called $\mathrm{LipConvnet} - n$ (n is the number of convolution layers and a multiple of 5 in our experiments). It consists of $(n/5) - 1$ Orthogonal convolutions of stride 1 (followed by the MaxMin activation function), followed by Orthogonal convolution of stride 2 (again followed by the MaxMin). It is summarized in Table 7.1. conv $[k \times k, m, s]$ denotes convolution layer with filter of size $k \times k$, out channels m and stride s. It is followed by a fully connected layer to output the class logits. The MaxMin activation function [133] is described in Appendix Section 7.C.

Model (LipConvnet-)	Conv. Layer	CIFAR-10			CIFAR-100		
		Clean Acc.	Robust Acc.	Time per epoch (s)	Clean Acc.	Robust Acc.	Time per epoch (s)
5	BCOP	74.35%	58.01%	96.153	42.61%	**28.67%**	94.463
	SOC	**75.78%**	**59.16%**	**31.096**	**42.73%**	27.82%	**30.844**
10	BCOP	74.47%	58.48%	122.115	42.08%	27.75%	119.038
	SOC	**76.48%**	**60.82%**	**48.242**	**43.71%**	**29.39%**	**48.363**
15	BCOP	73.86%	57.39%	145.944	39.98%	26.17%	144.173
	SOC	**76.68%**	**61.30%**	**63.742**	**42.93%**	**28.79%**	**63.540**
20	BCOP	69.84%	52.10%	170.009	36.13%	22.50%	172.266
	SOC	**76.43%**	**61.92%**	**77.226**	**43.07%**	**29.18%**	**76.460**
25	BCOP	68.26%	49.92%	207.359	28.41%	16.34%	205.313
	SOC	**75.19%**	**60.18%**	**98.534**	**43.31%**	**28.59%**	**95.950**
30	BCOP	64.11%	43.39%	227.916	26.87%	14.03%	229.840
	SOC	**74.47%**	**59.04%**	**110.531**	**42.90%**	**28.74%**	**107.163**
35	BCOP	63.05%	41.72%	267.272	21.71%	10.33%	274.256
	SOC	**73.70%**	**58.44%**	**130.671**	**42.44%**	**28.31%**	**126.368**
40	BCOP	60.17%	38.87%	295.350	19.97%	8.66%	289.369
	SOC	**71.63%**	**54.36%**	**144.556**	**41.83%**	**27.98%**	**140.458**

Table 7.2: Results for provable robustness against adversarial examples. Clean Acc. is the Clean Accuracy, Robust Acc. is the provably robust accuracy (l_2 perturbation radius of 36/255).

7.7 Experiments

Our goal is to evaluate the expressiveness of our method (SOC) compared to BCOP for constructing Orthogonal convolutional layers. To study this, we perform experiments in three settings: (a) provably robust image classification, (b) standard training and (c) adversarial training.

All experiments were performed using 1 NVIDIA GeForce RTX 2080 Ti GPU. All networks were trained for 200 epochs with an initial learning rate 0.1, dropped by a factor of 0.1 after 50 and 150 epochs. We use no weight decay for training with BCOP convolution as it significantly reduces its performance. For training with standard convolution and SOC, we use a

weight decay of 10^{-4}.

To evaluate the approximation error for SOC at convergence (using Theorem 14), we compute the norm of the Jacobian of the Skew-Symmetric convolution filter using real normalization [217]. We observe that the maximum norm (across different experiments and layers of the network) is below 1.8 (i.e. slightly below the theoretical upper bound of 2.1 discussed in Section 7.6.1) resulting in a maximum error of $1.8^{12}/12! = 2.415 \times 10^{-6}$.

7.7.1 Provable Defenses against Adversarial Attacks

To certify provable robustness of 1-Lipschitz network f for some input $\mathbf{x}$, we first define the margin of prediction: $\mathcal{M}_f(\mathbf{x}) = \max\left(0, y_t - \max_{i \neq t} y_i\right)$ where $\mathbf{y} = [y_1, y_{2,\ldots}]$ is the predicted logits from f on $\mathbf{x}$ and y_t is the correct logit. Using Theorem 7 in [232], we can derive the robustness certificate as $\mathcal{M}_f(\mathbf{x})/\sqrt{2}$. The provable robust accuracy, evaluated using an l_2 perturbation radius of $36/255$ (same as in [232]) equals the fraction of data points ($\mathbf{x}$) in the test dataset satisfying $\mathcal{M}_f(\mathbf{x})/\sqrt{2} \geq 36/255$.

In Table 7.2, we show the results of our experiments using different $\mathrm{LipConvnet}$ architectures with varying number of layers on CIFAR-10 and CIFAR-100 datasets. We make the following observations: (a) SOC achieves significantly higher standard and provable robust accuracy than BCOP for different architectures and datasets, (b) SOC requires significantly less training time per epoch than BCOP and (c) as the number of layers increases, the performance of BCOP degrades rapidly but that of SOC remains largely consistent. For example, on a $\mathrm{LipConvnet} - 40$ architecture, SOC achieves 11.46% higher standard accuracy; 15.49% higher provable robust accuracy on the CIFAR-10 dataset and 21.86% higher standard accuracy; 19.32% higher provable

Model	Conv. Layer	CIFAR-10		CIFAR-100	
		Clean Accuracy	**Time per epoch (s)**	**Clean Accuracy**	**Time per epoch (s)**
Resnet-18	Standard	95.10%	13.289	77.60%	13.440
	BCOP	92.38%	128.383	71.16%	128.146
	SOC	**94.24%**	**110.750**	**74.55%**	**103.633**
Resnet-34	Standard	95.54%	22.348	78.60%	22.806
	BCOP	93.79%	237.068	73.38%	235.367
	SOC	**94.44%**	**170.864**	**75.52%**	**164.178**
Resnet-50	Standard	95.47%	38.834	78.11%	37.454
	SOC	**94.68%**	**584.762**	**77.95%**	**597.297**

Table 7.3: Results for standard accuracy. For Resnet-50, we observe OOM (Out Of Memory) error when using BCOP.

Conv. Layer	CIFAR-10			CIFAR-100		
	Clean Accuracy	**Robust Accuracy**	**Time per epoch (s)**	**Clean Accuracy**	**Robust Accuracy**	**Time per epoch (s)**
Standard	83.05%	44.39%	28.139	59.87%	22.78%	28.147
BCOP	79.26%	34.85%	264.694	54.80%	16.00%	252.868
SOC	**82.24%**	**43.73%**	**203.860**	**58.95%**	**22.65%**	**199.188**

Table 7.4: Results for empirical robustness against adversarial examples (l_∞ perturbation radius of 8/255). Time (s) is the Time per epoch in seconds. Model is Resnet-18.

robust accuracy on the CIFAR-100 dataset. We further emphasize that none of the other well known deterministic provable defenses (discussed in Section 7.2) are scalable to large networks as the ones in Table 7.2. BCOP, while scalable, achieves significantly lower standard and provable robust accuracies for deep networks than SOC.

7.7.2 Standard Training

For standard training, we perform experiments using Resnet-18, Resnet-34 and Resnet-50 architectures on CIFAR-10 and CIFAR-100 datasets. Results are presented in Table 7.3. We again

observe that SOC achieves higher standard accuracy than BCOP on different architectures and datasets while requiring significantly less time to train. For Resnet-50, the performance of SOC almost matches that of standard convolution layers while BCOP results in an Out Of Memory (OOM) error. However, for Resnet-18 and Resnet-34, the difference is not as significant as the one observed for $\mathrm{LipConvnet}$ architectures in Table 7.2. We conjecture that this is because the residual connections allows the gradient to flow relatively freely compared to being restricted to flow through the convolution layers in $\mathrm{LipConvnet}$ architectures.

7.7.3 Adversarial Training

For adversarial training, we use a threat model with an l_∞ attack radius of $8/255$. Note that we use the l_∞ threat model (instead of l_2) because it is known to be a stronger adversarial threat model for evaluating empirical robustness [20]. For training, we use the FGSM variant by [114]. For evaluation, we use 50 iterations of PGD with step size of $2/255$ and 10 random restarts. Results are presented in Table 7.4. We observe that for Resnet-18 architecture and on both CIFAR-10 and CIFAR-100 datasets, SOC results in significantly improved standard and empirical robust accuracy compared to BCOP while requiring significantly less time to train. The performance of SOC comes close to the performance of a standard convolution layer with the difference being less than 1% for both standard and robust accuracy on both the datasets.

7.8 Discussion and Future work

In this chapter, we design a new orthogonal convolution layer by first constructing a Skew-Symmetric convolution filter and then applying the convolution exponential [209] to the filter.

We also derive provable guarantees on the approximation of the exponential using a finite number of terms. Our method achieves significantly higher accuracy than BCOP for various network architectures and datasets under standard, adversarial and provably robust training setups while requiring less training time per epoch. We suggest the following directions for future research:

Reducing the evaluation time: While SOC requires less time to train than BCOP, it requires more time for evaluation because the convolution filter needs to be applied multiple times to approximate the orthogonal matrix with the desired error. In contrast, BCOP constructs an orthogonal convolution filter that needs to be applied only once during evaluation. From Theorem 14, we know that we can reduce the number of terms required to achieve the desired approximation error by reducing the Jacobian norm $\|\mathbf{J}\|_2$. Training approaches such as spectral norm regularization [233] and singular value clipping [214] can be useful to further lower $\|\mathbf{J}\|_2$ and thus reduce the evaluation time.

Complete Set of SOC convolutions: While Theorem 15 suggests that the complete set of SOC convolutions can be constructed from a subset of Skew-Symmetric matrices $\mathbf{B}$ that satisfy (a) $\|\mathbf{B}\|_2 \leq \pi$ and (b) $\exp(\mathbf{B}) = \exp(\mathbf{A})$ where $\mathbf{A}$ is the Jacobian of some Skew-Symmetric convolution filter, it is an open question how to efficiently parametrize this subset for training Lipschitz convolutional neural networks. This remains an interesting problem for future research.

Supplementary material

7.A Notation for supplementary material

For $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{0, \ldots, n\}$. For a vector $\mathbf{v}$, we use $\mathbf{v}_j$ to denote the element in the jth position of the vector. We use $\mathbf{A}_{j,:}$ and $\mathbf{A}_{:,k}$ to denote the jth row and k^{th} column of the matrix $\mathbf{A}$ respectively. We assume both $\mathbf{A}_{j,:}$, $\mathbf{A}_{:,k}$ to be column vectors (thus $\mathbf{A}_{j,:}$ is the transpose of jth row of $\mathbf{A}$). $\mathbf{A}_{j,k}$ denotes the element in j^{th} row and k^{th} column of $\mathbf{A}$. $\mathbf{A}_{j,:k}$ and $\mathbf{A}_{:j,k}$ denote the vectors containing the first k elements of the jth row and first j elements of kth column, respectively. $\mathbf{A}_{:j,:k}$ denotes the matrix containing the first j rows and k columns of $\mathbf{A}$. The same rules can be directly extended to higher order tensors. We use bold zero i.e $\mathbf{0}$ to denote the matrix (or tensor) consisting of zero at all elements, $\mathbf{I}_n$ to denote the identity matrix of size $n \times n$. We use $\mathbb{C}$ to denote the field of complex numbers and $\mathbb{R}$ for real numbers. For a scalar $a \in \mathbb{C}$, $\overline{a}$ denotes its complex conjugate. For a vector $\mathbf{v}$ or matrix (or tensor) $\mathbf{A}$, $\overline{\mathbf{v}}$ or $\overline{\mathbf{A}}$ denotes the element-wise complex conjugate. For $\mathbf{A} \in \mathbb{C}^{m \times n}$, $\mathbf{A}^H$ denotes the hermitian transpose i.e $\mathbf{A}^H = \overline{\mathbf{A}^T}$. For a scalar $a \in \mathbb{C}$, $\mathrm{Re}(a)$, $\mathrm{Im}(a)$ and $|a|$ denote the real part, imaginary part and modulus of a respectively. We use $[a, b)$ where $a, b \in \mathbb{C}$ to denote the set consisting of complex scalars on the line connecting a and b (including a, but excluding b). $\mathbf{A} \otimes \mathbf{B}$ denotes the kronecker product between matrices $\mathbf{A}$ and $\mathbf{B}$. We use ι to denote *iota* (i.e $\iota^2 = -1$).

For a matrix $\mathbf{A} \in \mathbb{C}^{q \times r}$ and a tensor $\mathbf{B} \in \mathbb{C}^{p \times q \times r}$, $\overrightarrow{\mathbf{A}}$ denotes the vector constructed by

stacking the rows of $\mathbf{A}$ and $\overrightarrow{\mathbf{B}}$ by stacking the vectors $\overrightarrow{\mathbf{B}_{j,:,:}},\ j \in [p-1]$ so that:

$$\left(\overrightarrow{\mathbf{A}}\right)^T = \left[\mathbf{A}_{0,:}^T\ ,\ \mathbf{A}_{1,:}^T\ ,\ \ldots\ ,\ \mathbf{A}_{q-1,:}^T\right]$$
$$\left(\overrightarrow{\mathbf{B}}\right)^T = \left[\left(\overrightarrow{\mathbf{B}_{0,:,:}}\right)^T\ ,\ \left(\overrightarrow{\mathbf{B}_{1,:,:}}\right)^T\ ,\ \ldots\ ,\ \left(\overrightarrow{\mathbf{B}_{p-1,:,:}}\right)^T\right]$$

For a 2D convolution filter, $\mathbf{L} \in \mathbb{C}^{p\times q\times r\times s}$, we define the tensor $\text{conv_transpose}(\mathbf{L}) \in \mathbb{C}^{q\times p\times r\times s}$ as follows:

$$[\text{conv_transpose}(\mathbf{L})]_{i,j,k,l} = \overline{[\mathbf{L}]}_{j,i,r-1-k,s-1-l} \tag{7.7}$$

Note that this is very different from the usual matrix transpose. Given an input $\mathbf{X} \in \mathbb{C}^{q\times n\times n}$, we use $\mathbf{L} \star \mathbf{X} \in \mathbb{C}^{p\times n\times n}$ to denote the convolution of filter $\mathbf{L}$ with $\mathbf{X}$. The notation $\mathbf{L} \star^i \mathbf{X} \triangleq \mathbf{L}\star^{i-1}(\mathbf{L} \star \mathbf{X})$. Unless specified otherwise, we assume zero padding and stride 1 in each direction.

7.B Proofs

7.B.1 Proof of Theorem 12

Theorem 7.B.1. *Consider a convolution filter $\mathbf{L} \in \mathbb{C}^{m\times m\times(2p+1)\times(2q+1)}$ applied to an input $\mathbf{X} \in \mathbb{C}^{m\times n\times n}$ that results in output $\mathbf{Y} = \mathbf{L} \star \mathbf{X} \in \mathbb{C}^{m\times n\times n}$. Let $\mathbf{J}$ be the jacobian of $\overrightarrow{\mathbf{Y}}$ with respect to $\overrightarrow{\mathbf{X}}$, then the jacobian for convolution with the filter $\text{conv_transpose}(\mathbf{L})$ is equal to $\mathbf{J}^H$.*

Proof. We first prove the above result assuming $m = 1$.

Assuming $\mathbf{m = 1}$:

We know that $\mathbf{J}$ is a doubly toeplitz matrix of size $n^2 \times n^2$:

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}^{(0)} & \mathbf{J}^{(-1)} & \dots & \mathbf{J}^{(-p)} & 0 \\ \mathbf{J}^{(1)} & \mathbf{J}^{(0)} & \mathbf{J}^{(-1)} & \ddots & \ddots \\ \vdots & \mathbf{J}^{(1)} & \mathbf{J}^{(0)} & \ddots & \ddots \\ \mathbf{J}^{(p)} & \ddots & \ddots & \ddots & \mathbf{J}^{(-1)} \\ 0 & \ddots & \ddots & \mathbf{J}^{(1)} & \mathbf{J}^{(0)} \end{bmatrix}$$

In the above equation, each $\mathbf{J}^{(i)}$ is a toeplitz matrix of size $n \times n$. Define $\mathbf{P}^{(k)}$ as a $n \times n$ matrix with $\mathbf{P}^{(k)}_{i,j} = 1$ if $i - j = k$ and 0 otherwise. Thus $\mathbf{J}$ can be written as:

$$\mathbf{J} = \sum_{i=-p}^{p} \mathbf{P}^{(i)} \otimes \mathbf{J}^{(i)}$$

Since each matrix $\mathbf{J}^{(i)}$ is a toeplitz matrix, it can be written as follows. Because the first two dimensions of filter $\mathbf{L}$ are of size 1, we index $\mathbf{L}$ using only the last two indices:

$$\mathbf{J}^{(i)} = \sum_{j=-q}^{q} \mathbf{L}_{p+i,q+j} \mathbf{P}^{(j)}$$

Thus, $\mathbf{J}$ can be written as:

$$\mathbf{J} = \sum_{i=-p}^{p} \sum_{j=-q}^{q} \mathbf{L}_{p+i,q+j} \left(\mathbf{P}^{(i)} \otimes \mathbf{P}^{(j)} \right)$$

Thus, $\mathbf{J}^H$ can be written as:

$$\mathbf{J}^H = \sum_{i=-p}^{p} \sum_{j=-q}^{q} \overline{\mathbf{L}_{p+i,q+j}} \left(\mathbf{P}^{(i)} \otimes \mathbf{P}^{(j)}\right)^T$$
$$\mathbf{J}^H = \sum_{i=-p}^{p} \sum_{j=-q}^{q} \overline{\mathbf{L}_{p+i,q+j}} \left(\mathbf{P}^{(i)}\right)^T \otimes \left(\mathbf{P}^{(j)}\right)^T$$
$$\mathbf{J}^H = \sum_{i=-p}^{p} \sum_{j=-q}^{q} \overline{\mathbf{L}_{p+i,q+j}} \left(\mathbf{P}^{(-i)} \otimes \mathbf{P}^{(-j)}\right)$$
$$\mathbf{J}^H = \sum_{i=-p}^{p} \sum_{j=-q}^{q} \overline{\mathbf{L}_{p-i,q-j}} \left(\mathbf{P}^{(i)} \otimes \mathbf{P}^{(j)}\right)$$

Thus $\mathbf{J}^H$ corresponds to the jacobian of the convolution filter flipped along the third, fourth axis and each individual element conjugated.

Next, we prove the result when $m > 1$.

Assuming $\mathbf{m > 1}$:

We know that $\mathbf{J}$ is a matrix of size $mn^2 \times mn^2$. Let $\mathbf{J}^{(i,j)}$ denote the block of size $n^2 \times n^2$ as follows:

$$\mathbf{J}^{(i,j)} = \mathbf{J}_{in^2:(i+1)n^2, jn^2:(j+1)n^2}$$

Note that $\mathbf{J}^{(i,j)}$ is the jacobian of convolution with 1×1 filter $\mathbf{L}_{i:i+1,j:j+1,:,:}$. Now consider the $(i,j)^{th}$ block of $\mathbf{J}^H$. Using definition of conjugate transpose (i.e H operator):

$$\left(\mathbf{J}^H\right)^{(i,j)} = \left(\mathbf{J}^{(j,i)}\right)^H \tag{7.8}$$

Consider the 1×1 filter at the $(i,j)^{th}$ index in $\mathrm{conv_transpose}(\mathbf{L})$. By the definition of

$\mathrm{conv_transpose}$ operator, we have:

$$\begin{aligned}&[\mathrm{conv_transpose}(\mathbf{L})]_{i:i+1,j:j+1,:,:}\\&= \mathrm{conv_transpose}(\mathbf{L}_{j:j+1,i:i+1,:,:})\end{aligned} \tag{7.9}$$

Using equations (7.8) and (7.9) and the proof for the case $\mathbf{m} = 1$, we have the desired proof.

□

7.B.2 Proof of Theorem 13

Theorem 7.B.2. *Consider a convolution filter* $\mathbf{L} \in \mathbb{C}^{m\times m\times(2p+1)\times(2q+1)}$. *Given an input* $\mathbf{X} \in \mathbb{C}^{m\times n\times n}$, *output* $\mathbf{Y} = \mathbf{L} \star \mathbf{X} \in \mathbb{C}^{m\times n\times n}$. *The jacobian of* $\overrightarrow{\mathbf{Y}}$ *with respect to* $\overrightarrow{\mathbf{X}}$ *(call it* $\mathbf{J}$*) will be a matrix of size* $n^2m \times n^2m$. $\mathbf{J}$ *is a skew hermitian matrix if and only if:*

$$\mathbf{L} = \mathbf{M} - \mathrm{conv_transpose}(\mathbf{M})$$

for some filter $\mathbf{M} \in \mathbb{C}^{m\times m\times(2p+1)\times(2q+1)}$*:*

Proof. We first prove that if $\mathbf{J}$ is a skew-hermitian matrix, then:

$$\mathbf{L} = \mathbf{M} - \mathrm{conv_transpose}(\mathbf{M})$$

Let $\mathbf{J}^{(i,j)}$ denote the block of size $n^2 \times n^2$ as follows:

$$\mathbf{J}^{(i,j)} = \mathbf{J}_{in^2:(i+1)n^2,jn^2:(j+1)n^2}$$

so that $\mathbf{J}$ can be written in terms of the blocks $\mathbf{J}^{(i,j)}$:

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}^{(0,0)} & \mathbf{J}^{(0,1)} & \dots & \mathbf{J}^{(0,m-1)} \\ \mathbf{J}^{(1,0)} & \mathbf{J}^{(1,1)} & \dots & \mathbf{J}^{(1,m-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{J}^{(m-1,0)} & \mathbf{J}^{(m-1,1)} & \dots & \mathbf{J}^{(m-1,m-1)} \end{bmatrix}$$

Since $\mathbf{J}$ is skew-hermitian, we have:

$$\mathbf{J}^{(i,j)} = -\left(\mathbf{J}^{(j,i)}\right)^H, \quad \forall\, i,\, j \in [m-1]$$

It is readily observed that $\mathbf{J}^{(i,j)}$ corresponds to the jacobian of convolution with 1×1 filter $\mathbf{L}_{i:i+1,j:j+1,:,:}$. For some given filter $\mathbf{A}$, we use $\mathbf{A}^{(i,j)}$ to denote the 1×1 filter $\mathbf{A}_{i:i+1,j:j+1,:,:}$ for simplicity. Thus, the above equation can be rewritten as:

$$\mathbf{L}^{(i,j)} = -\text{conv_transpose}\left(\mathbf{L}^{(j,i)}\right), \quad \forall\, i,\, j \in [m-1] \tag{7.10}$$

Now construct a filter $\mathbf{M}$ such that for $i \neq j$:

$$\mathbf{M}^{(i,j)} = \begin{cases} \mathbf{L}^{(i,j)}, & i < j \\ \mathbf{0}, & i > j \end{cases} \tag{7.11}$$

For $i = j$, $\mathbf{M}$ is given as follows:

$$\mathbf{M}_{r,s}^{(i,i)} = \begin{cases} \mathbf{L}_{r,s}^{(i,i)}, & r \leq p-1 \\ \mathbf{L}_{r,s}^{(i,i)}, & r = p,\ s \leq q-1 \\ 0.5 \times \mathbf{L}_{r,s}^{(i,i)}, & r = p,\ s = q \\ 0, & otherwise \end{cases} \tag{7.12}$$

Next, our goal is to show that:

$$\mathbf{L} = \mathbf{M} - \text{conv_transpose}(\mathbf{M})$$

Now by the definition of conv_transpose, we have:

$$\begin{aligned} &[\mathbf{M} - \text{conv_transpose}(\mathbf{M})]^{(i,j)} \\ &= \mathbf{M}^{(i,j)} - [\text{conv_transpose}(\mathbf{M})]^{(i,j)} \\ &= \mathbf{M}^{(i,j)} - \text{conv_transpose}\left(\mathbf{M}^{(j,i)}\right) \end{aligned} \tag{7.13}$$

Case 1: For $i < j$, using equations (7.10) and (7.11):

$$\mathbf{M}^{(i,j)} - \text{conv_transpose}\left(\mathbf{M}^{(j,i)}\right) = \mathbf{M}^{(i,j)} = \mathbf{L}^{(i,j)}$$

Case 2: For $i > j$, using equations (7.10) and (7.11):

$$\begin{aligned}
&\mathbf{M}^{(i,j)} - \text{conv_transpose}\left(\mathbf{M}^{(j,i)}\right) \\
&= -\text{conv_transpose}\left(\mathbf{M}^{(j,i)}\right) \\
&= -\text{conv_transpose}\left(\mathbf{L}^{(j,i)}\right) = \mathbf{L}^{(i,j)}
\end{aligned}$$

Case 3: For $i = j$, we further simplify equation (7.13):

$$\begin{aligned}
&\mathbf{M}^{(i,i)}_{r,s} - \left[\text{conv_transpose}\left(\mathbf{M}^{(i,i)}\right)\right]_{r,s} \\
&= \mathbf{M}^{(i,i)}_{r,s} - \overline{\mathbf{M}^{(i,i)}_{2p-r,2q-s}} \qquad (7.14)
\end{aligned}$$

Subcase 3(a): For $(r \leq p-1)$ or $(r = p,\ s \leq q-1)$, we have:

$$\mathbf{M}^{(i,i)}_{2p-r,2q-s} = 0$$

Thus for $(r \leq p-1)$ or $(r = p,\ s \leq q-1)$: equation (7.14) simplifies to $\mathbf{M}^{(i,i)}_{r,s}$. The result follows trivially from the very definition of $\mathbf{M}^{(i,i)}_{r,s}$, i.e equation (7.12).

Subcase 3(b): For $(r \geq p+1)$ or $(r = p,\ s \geq q+1)$, we have:

$$\mathbf{M}^{(i,i)}_{r,s} = 0$$

Thus, equation (7.14) simplifies to:

$$\mathbf{M}_{r,s}^{(i,i)} - \overline{\mathbf{M}_{2p-r,2q-s}^{(i,i)}} = -\overline{\mathbf{M}_{2p-r,2q-s}^{(i,i)}}$$

Since $(r \geq p+1)$ or $(r = p,\ s \geq q+1)$, we have: $(2p-r \leq p-1)$ or $(2p-r = p,\ 2q-s \leq q-1)$ respectively. Thus using equation (7.12), we have:

$$-\overline{\mathbf{M}_{2p-r,2q-s}^{(i,i)}} = -\overline{\mathbf{L}_{2p-r,2q-s}^{(i,i)}}$$

Since $\mathbf{L}^{(i,i)}$ is a skew-hermitian filter, we have from Theorem 12:

$$\mathbf{L}_{r,s}^{(i,i)} = -\overline{\mathbf{L}_{2p-r,2q-s}^{(i,i)}}$$

Thus in this subcase, equation (7.14) simplifies to $\mathbf{L}_{r,s}^{(i,i)}$ again.

Subcase 3(c): For $r = p,\ s = q$, since $\mathbf{L}^{(i,i)}$ is a skew-hermitian filter, we have:

$$\mathbf{L}_{p,q}^{(i,i)} = -\overline{\mathbf{L}_{p,q}^{(i,i)}}$$

$$\mathbf{L}_{p,q}^{(i,i)} + \overline{\mathbf{L}_{p,q}^{(i,i)}} = 0$$

Thus, $\mathbf{L}_{p,q}^{(i,i)}$ is a purely imaginary number. In this subcase

$$\mathbf{M}_{r,s}^{(i,i)} - \overline{\mathbf{M}_{2p-r,2q-s}^{(i,i)}}$$

$$= \mathbf{M}_{p,q}^{(i,i)} - \overline{\mathbf{M}_{p,q}^{(i,i)}} = 2\mathbf{M}_{p,q}^{(i,i)}$$

Using equation (7.12) , we have:

$$2\mathbf{M}_{p,q}^{(i,i)} = \mathbf{L}_{p,q}^{(i,i)}$$

Thus, we get:

$$\mathbf{M}_{r,s}^{(i,i)} - \left[\text{conv_transpose}\left(\mathbf{M}^{(i,i)}\right)\right]_{r,s} = \mathbf{L}_{r,s}^{(i,i)}$$

Thus we have established: $\mathbf{L} = \mathbf{M} - \text{conv_transpose}(\mathbf{M})$. Note that the opposite direction of the if and only if statement follows trivially from the above proof. □

7.B.3 Proof of Theorem 14

Theorem 7.B.3. *(a) For a scalar $\lambda \in \mathbb{C}$ with $\text{Re}(\lambda) = 0$, the error between $\exp(\lambda)$ and approximation $p_k(\lambda)$ given below can be bounded as follows:*

$$\exp(\lambda) = \sum_{i=0}^{\infty} \frac{\lambda^k}{i!}, \qquad p_k(\lambda) = \sum_{i=0}^{k-1} \frac{\lambda^i}{i!} \tag{7.15}$$
$$\left|\exp(\lambda) - p_k(\lambda)\right| \leq \frac{|\lambda|^k}{k!}, \qquad \forall\, \lambda : \text{Re}(\lambda) = 0$$

(b) For a skew-hermitian matrix $\mathbf{A}$, the error between $\exp(\mathbf{A})$ and the series approximation $\mathbf{S}_k(\mathbf{A})$ can be bounded as follows:

$$\exp(\mathbf{A}) = \sum_{i=0}^{\infty} \frac{\mathbf{A}^i}{i!}, \qquad \mathbf{S}_k(\mathbf{A}) = \sum_{i=0}^{k-1} \frac{\mathbf{A}^i}{i!}$$
$$\|\exp(\mathbf{A}) - \mathbf{S}_k(\mathbf{A})\|_2 \leq \frac{\|\mathbf{A}\|_2^k}{k!}$$

Proof. Since $\mathbf{A}$ is skew-hermitian, it is a normal matrix and eigenvectors for distinct eigenvalues must be orthogonal. Let the eigenvalue decomposition of $\mathbf{A}$ be given as follows:

$$\mathbf{A} = \mathbf{U}\Lambda\mathbf{U}^H$$

Note that Λ is a diagonal matrix, and each element along the diagonal is purely imaginary (since $\mathbf{A}$ is skew-hermitian). Exponentiating both sides, we get:

$$\exp(\mathbf{A}) = \mathbf{U}\exp(\Lambda)\mathbf{U}^H$$

Thus the error $\mathbf{E}_k(\mathbf{A})$ is given by:

$$\begin{aligned}
\mathbf{E}_k(\mathbf{A}) &= \exp(\mathbf{A}) - \mathbf{S}_k(\mathbf{A}) \qquad (7.16)\\
\mathbf{E}_k(\mathbf{A}) &= \mathbf{U}\left(\exp(\Lambda) - \mathbf{S}_k(\Lambda)\right)\mathbf{U}^H\\
\|\mathbf{E}_k(\mathbf{A})\|_2 &= \|\mathbf{U}\left(\exp(\Lambda) - \mathbf{S}_k(\Lambda)\right)\mathbf{U}^H\|_2\\
\|\mathbf{E}_k(\mathbf{A})\|_2 &= \|\left(\exp(\Lambda) - \mathbf{S}_k(\Lambda)\right)\|_2
\end{aligned}$$

Since $\left(\exp(\Lambda) - \mathbf{S}_k(\Lambda)\right)$ is a diagonal matrix, we have:

$$\|\mathbf{E}_k(\mathbf{A})\|_2 = \max_i \left|\left(\exp(\Lambda_{i,i}) - p_k(\Lambda_{i,i})\right)\right| \qquad (7.17)$$

Let λ be an arbitrary element along the diagonal of Λ i.e $\lambda = \Lambda_{i,i}$ for some i. First note that:

$$\begin{aligned}
&\lambda \int_0^1 \{\exp(t\lambda) - p_k(t\lambda)\} dt \\
&= \int_0^1 \{\exp(t\lambda) - p_k(t\lambda)\} \lambda dt
\end{aligned}$$

Substituting $u = \lambda t$, we have:

$$\begin{aligned}
&= \int_0^\lambda \{\exp(u) - p_k(u)\} du \\
&= \int_0^\lambda \exp(u) du - \int_0^\lambda p_k(u) du \\
&= \exp(\lambda) - 1 - \int_0^\lambda p_k(u) du
\end{aligned}$$

Substituting $p_k(u)$ using equation (7.15):

$$
\begin{aligned}
&= \exp(\lambda) - 1 - \int_0^{\lambda} \sum_{i=0}^{k-1} \frac{u^i}{i!} du \\
&= \exp(\lambda) - 1 - \sum_{i=0}^{k-1} \int_0^{\lambda} \frac{u^i}{i!} du \\
&= \exp(\lambda) - 1 - \sum_{i=0}^{k-1} \frac{u^{i+1}}{(i+1)!} \Big|_0^{\lambda} \\
&= \exp(\lambda) - 1 - \sum_{i=0}^{k-1} \frac{\lambda^{i+1}}{(i+1)!} \\
&= \exp(\lambda) - 1 - \sum_{i=1}^{k} \frac{\lambda^i}{i!} \\
&= \exp(\lambda) - \sum_{i=0}^{k+1} \frac{\lambda^i}{i!} \\
&= \exp(\lambda) - p_{k+1}(\lambda)
\end{aligned}
$$

This gives the following result:

$$\exp(\lambda) - p_{k+1}(\lambda) = \lambda \int_0^1 \Big(\exp(t\lambda) - p_k(t\lambda) \Big) dt \tag{7.18}$$

We shall now prove the main result using induction and equation (7.18):

Base case:

Use $k = 0$ and the convention that $p_0(\lambda) = 0$. We know that $p_1(\lambda) = 1$.

$$
\begin{aligned}
\exp(\lambda) - p_1(\lambda) &= \lambda \int_0^1 \Big(\exp(t\lambda) - p_0(t\lambda) \Big) dt \\
\big| \exp(\lambda) - 1 \big| &= \big| \lambda \int_0^1 \exp(t\lambda) dt \big|
\end{aligned}
$$

Since λ is purely imaginary and t is purely real, we have $\big|\exp(t\lambda)\big| = 1$:

$$\big|\exp(\lambda) - 1\big| \leq \big|\lambda\big| \int_0^1 \big|\exp(t\lambda)\big| dt = \big|\lambda\big| \int_0^1 1 dt$$

$$\big|\exp(\lambda) - 1\big| \leq \big|\lambda\big|$$

Induction step:

Assuming this holds for all k i.e:

$$\big|\exp(\lambda) - p_k(\lambda)\big| \leq \frac{|\lambda|^k}{k!} \tag{7.19}$$

Now let us consider $\big|\exp(\lambda) - p_{k+1}(\lambda)\big|$:

$$\big|\exp(\lambda) - p_{k+1}(\lambda)\big| \leq \left|\lambda \int_0^1 \Big(\exp(t\lambda) - p_k(t\lambda)\Big) dt\right|$$

$$\big|\exp(\lambda) - p_{k+1}(\lambda)\big| \leq \big|\lambda\big| \int_0^1 \left|\Big(\exp(t\lambda) - p_k(t\lambda)\Big)\right| dt$$

Using equation (7.19), we have:

$$\big|\exp(\lambda) - p_{k+1}(\lambda)\big| \leq \big|\lambda\big| \int_0^1 \frac{|t\lambda|^k}{k!} dt$$

$$\big|\exp(\lambda) - p_{k+1}(\lambda)\big| \leq \big|\lambda\big|^{k+1} \int_0^1 \frac{|t|^k}{k!} dt$$

$$\big|\exp(\lambda) - p_{k+1}(\lambda)\big| \leq \frac{\big|\lambda\big|^{k+1}}{(k+1)!}$$

This proves (a).

Since λ is an arbitrary element along the diagonal of eigenvalue matrix Λ, using equations (7.16)

and (7.17) we have:

$$\begin{aligned}
\|\exp(\mathbf{A}) - \mathbf{S}_k(\mathbf{A})\|_2 &= \max_i \left|\exp(\Lambda_{i,i}) - p_k(\Lambda_{i,i})\right| \\
\|\exp(\mathbf{A}) - \mathbf{S}_k(\mathbf{A})\|_2 &\leq \max_i \frac{\left|\Lambda_{i,i}\right|^k}{k!} \\
\|\exp(\mathbf{A}) - \mathbf{S}_k(\mathbf{A})\|_2 &\leq \frac{1}{k!} \max_i \left|\Lambda_{i,i}\right|^k
\end{aligned} \tag{7.20}$$

Since $\mathbf{A}$ is skew-hermitian, it is a normal matrix and singular values are equal to the magnitude of eigenvalues. Thus we have from equation (7.20):

$$\begin{aligned}
\max_i \left|\Lambda_{i,i}\right| &= \|\Lambda\|_2 = \|\mathbf{A}\|_2 \\
\|\exp(\mathbf{A}) - \mathbf{S}_k(\mathbf{A})\|_2 &\leq \frac{\|\mathbf{A}\|_2^k}{k!}
\end{aligned}$$

This proves (b). □

7.B.4 Proof of Theorem 15

Theorem 7.B.4. *Given a real skew-symmetric matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, we can construct a real skew-symmetric matrix $\mathbf{B} \in \mathbb{R}^{n\times n}$ such that $\mathbf{B}$ satisfies: (a) $\exp(\mathbf{A}) = \exp(\mathbf{B})$ and (b) $\|\mathbf{B}\|_2 \leq \pi$.*

Proof. We know that for eigenvalues of real symmetric matrices are purely imaginary and come in pairs: $\lambda_1\iota,\ -\lambda_1\iota, \lambda_2\iota, -\lambda_2\iota$ where each λ_i is real. When n is an odd integer, 0 is an eigenvalue. Additionally, we know that a real skew symmetric matrix can be expressed in a block diagonal form as follows:

$$\mathbf{A} = \mathbf{Q}\Sigma\mathbf{Q}^T \tag{7.21}$$

Here $\mathbf{Q}$ is a real orthogonal matrix and Σ is a block diagonal matrix defined as follows:

$$\Sigma_{2i:2i+2,2i:2i+2} = \begin{bmatrix} 0 & \lambda_i \\ -\lambda_i & 0 \end{bmatrix}, \qquad 0 \leq i < \left\lfloor \frac{n}{2} \right\rfloor \tag{7.22}$$

In the above equation, $\lambda_i \in \mathbb{R}$ and $\pm\lambda_i \iota$ are the eigenvalues of $\mathbf{A}$. When n is odd, we additionally have:

$$\Sigma_{n-1,n-1} = 0$$

Taking the exponential of both sides of equation (7.21):

$$\exp(\mathbf{A}) = \mathbf{Q} \exp(\Sigma) \mathbf{Q}^T \tag{7.23}$$

We can compute $\exp(\Sigma)$ by computing the exponential of each 2×2 block defined in equation (7.22):

$$\begin{aligned} &\exp\left(\begin{bmatrix} 0 & \lambda_i \\ -\lambda_i & 0 \end{bmatrix} \right) \\ &= \exp\left(\lambda_i \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \right) = \begin{bmatrix} \cos(\lambda_i) & -\sin(\lambda_i) \\ \sin(\lambda_i) & \cos(\lambda_i) \end{bmatrix} \end{aligned} \tag{7.24}$$

From equation (7.24), we observe each λ_i can be shifted by integer multiples of $2\pi\iota$ without changing the exponential.

For each $\lambda_i, i \in [\lfloor n/2 \rfloor - 1]$, we define a scalar μ_i:

$$\mu_i = \lambda_i + 2\pi k_i \iota, \qquad k_i \in \mathbb{Z} \tag{7.25}$$

$$\mu_i \in [-\pi\iota, \pi\iota) \tag{7.26}$$

Construct a new matrix $\mathbf{B}$ defined as follows:

$$\mathbf{B} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T \tag{7.27}$$

The matrix $\mathbf{D}$ in equation (7.27) is defined as follows:

$$\mathbf{D}_{2i:2i+2,2i:2i+2} = \begin{bmatrix} 0 & \mu_i \\ -\mu_i & 0 \end{bmatrix}, \qquad 0 \leq i < \left\lfloor \frac{n}{2} \right\rfloor \tag{7.28}$$

Let us verify that $\mathbf{B}$ satisfies the following properties.

Using equations (7.24), (7.25) and (7.28), we know that:

$$\exp(\mathbf{D}) = \exp(\Lambda)$$

This results in the following set of equations:

$$\exp(\mathbf{B}) = \mathbf{Q}\exp(\mathbf{D})\mathbf{Q}^T$$

$$\exp(\mathbf{B}) = \mathbf{Q}\exp(\Lambda)\mathbf{Q}^T = \exp(\mathbf{A})$$

Using equations (7.26) and (7.28), we have:

$$\|\mathbf{B}\|_2 = \|\mathbf{Q}\mathbf{D}\mathbf{Q}^T\|_2 = \|\mathbf{D}\|_2$$

$$\|\mathbf{D}\|_2 \leq \pi$$

Note that $\mathbf{B}$ is a product of 3 real matrices $\mathbf{Q}$, $\mathbf{D}$ and $\mathbf{Q}^T$ and hence $\mathbf{B}$ is real. Moreover, since $\mathbf{D}$ is skew symmetric, $\mathbf{B}$ is skew symmetric. □

7.B.5 Proof of Theorem 16

Theorem 16. *Given a skew-hermitian matrix* $\mathbf{A}$*, we can construct a skew-hermitian matrix* $\mathbf{B}$ *by adding integer multiples of* $2\pi\iota$ *to eigenvalues of* $\mathbf{A}$ *such that* $\mathbf{B}$ *satisfies: (a)* $\exp(\mathbf{A}) = \exp(\mathbf{B})$ *and (b)* $\|\mathbf{B}\|_2 \leq \pi$.

Proof. Let the eigenvalue decomposition of $\mathbf{A}$ be given:

$$\mathbf{A} = \mathbf{U}\Lambda\mathbf{U}^H$$

Let λ_j be some eigenvalue of $\mathbf{A}$ such that:

$$\lambda_j = \Lambda_{j,j}$$

Construct a new diagonal matrix $\mathbf{D}$ of eigenvalues such that:

$$\mathbf{D}_{j,j} = \lambda_j + 2\pi k_j \iota, \qquad k_j \in \mathbb{Z} \tag{7.29}$$

$$\mathbf{D}_{j,j} \in [-\pi\iota, \pi\iota) \tag{7.30}$$

Construct a new matrix $\mathbf{B}$ defined as follows:

$$\mathbf{B} = \mathbf{U}\mathbf{D}\mathbf{U}^H$$

Let us verify that $\mathbf{B}$ satisfies the following properties.

Using equation (7.29), we have:

$$\exp(\mathbf{B}) = \mathbf{U}\exp(\mathbf{D})\mathbf{U}^H$$

$$\exp(\mathbf{B}) = \mathbf{U}\exp(\Lambda)\mathbf{U}^H = \exp(\mathbf{A})$$

Using equation (7.30), we have:

$$\|\mathbf{B}\|_2 = \|\mathbf{U}\mathbf{D}\mathbf{U}^H\|_2 = \|\mathbf{D}\|_2$$

$$\|\mathbf{D}\|_2 = \max_j |D_{j,j}| \leq \pi$$

□

7.B.6 Proof of Theorem 17

Theorem 17. *Consider a convolution filter* $\mathbf{L} \in \mathbb{C}^{m \times m \times (2p+1) \times (2q+1) \times (2r+1)}$ *applied to an input* $\mathbf{X} \in \mathbb{C}^{m \times n \times n \times n}$ *that results in output* $\mathbf{Y} = \mathbf{L} \star \mathbf{X} \in \mathbb{C}^{m \times n \times n \times n}$. *Let* $\mathbf{J}$ *be the jacobian of* $\overrightarrow{\mathbf{Y}}$ *with respect to* $\overrightarrow{\mathbf{X}}$ *, then the jacobian for convolution with the filter* $\text{conv3d_transpose}(\mathbf{L})$ *is equal to* $\mathbf{J}^H$.

Proof. We first prove the above result assuming $m = 1$.

Assuming $\mathbf{m = 1}$:

Because the first two dimensions of filter $\mathbf{L}$ are of size 1, we index $\mathbf{L}$ using only the last two indices. Define $\mathbf{P}^{(k)}$ as a $n \times n$ matrix with $\mathbf{P}^{(k)}_{i,j} = 1$ if $i - j = k$ and 0 otherwise. We know that $\mathbf{J}$ is a triply toeplitz matrix of size $n^3 \times n^3$ given as follows:

$$\mathbf{J} = \sum_{i=-p}^{p} \sum_{j=-q}^{q} \sum_{k=-r}^{r} \mathbf{L}_{p+i,q+j,r+k} \left(\mathbf{P}^{(i)} \otimes \mathbf{P}^{(j)} \otimes \mathbf{P}^{(k)} \right)$$

Thus, $\mathbf{J}^H$ can be written as:

$$\begin{aligned}
&\mathbf{J}^H \\
&= \sum_{i=-p}^{p} \sum_{j=-q}^{q} \sum_{k=-r}^{r} \overline{\mathbf{L}_{p+i,q+j,r+k}} \left(\mathbf{P}^{(i)} \otimes \mathbf{P}^{(j)} \otimes \mathbf{P}^{(k)} \right)^T \\
&= \sum_{i=-p}^{p} \sum_{j=-q}^{q} \sum_{k=-r}^{r} \overline{\mathbf{L}_{p+i,q+j,r+k}} \mathbf{P}^{(-i)} \otimes \mathbf{P}^{(-j)} \otimes \mathbf{P}^{(-k)} \\
&= \sum_{i=-p}^{p} \sum_{j=-q}^{q} \sum_{k=-r}^{r} \overline{\mathbf{L}_{p-i,q-j,r-k}} \mathbf{P}^{(i)} \otimes \mathbf{P}^{(j)} \otimes \mathbf{P}^{(k)}
\end{aligned}$$

Thus $\mathbf{J}^H$ corresponds to the jacobian of the convolution filter flipped along the third, fourth, fifth

axis and each individual element conjugated.

Next, we prove the above result when $m > 1$.

Assuming $\mathbf{m > 1}$:

We know that $\mathbf{J}$ is a matrix of size $mn^3 \times mn^3$. Let $\mathbf{J}^{(i,j)}$ denote the block of size $n^3 \times n^3$ as follows:

$$\mathbf{J}^{(i,j)} = \mathbf{J}_{in^3:(i+1)n^3, jn^3:(j+1)n^3}$$

Note that $\mathbf{J}^{(i,j)}$ is the jacobian of convolution with 1×1 filter $\mathbf{L}_{i:i+1,j:j+1,:,:}$. Now consider the $(i,j)^{th}$ block of $\mathbf{J}^H$. Using definition of conjugate transpose (i.e H operator):

$$\left(\mathbf{J}^H\right)^{(i,j)} = \left(\mathbf{J}^{(j,i)}\right)^H \tag{7.31}$$

Consider the 1×1 filter at the $(i,j)^{th}$ index in $\mathrm{conv3d_transpose}(\mathbf{L})$. By the definition of $\mathrm{conv3d_transpose}$ operator, we have:

$$\begin{aligned} &[\mathrm{conv3d_transpose}(\mathbf{L})]_{i:i+1,j:j+1,:,:} \\ &= \mathrm{conv3d_transpose}(\mathbf{L}_{j:j+1,i:i+1,:,:}) \end{aligned} \tag{7.32}$$

Using equations (7.31) and (7.32) and the proof for the case $\mathbf{m} = 1$, we have the desired proof.

□

7.B.7 Proof of Theorem 18

Theorem 18. *Consider a convolution filter* $\mathbf{L} \in \mathbb{C}^{m\times m\times(2p+1)\times(2q+1)\times(2r+1)}$. *Given an input* $\mathbf{X} \in \mathbb{C}^{m\times n\times n\times n}$, *output* $\mathbf{Y} = \mathbf{L} \star \mathbf{X} \in \mathbb{C}^{m\times n\times n\times n}$. *The jacobian of* $\overrightarrow{\mathbf{Y}}$ *with respect to* $\overrightarrow{\mathbf{X}}$ *(call it* $\mathbf{J}$*) will be a matrix of size* $n^3m \times n^3m$. $\mathbf{J}$ *is a skew hermitian matrix if and only if:*

$$\mathbf{L} = \mathbf{M} - \text{conv3d_transpose}(\mathbf{M})$$

for some filter $\mathbf{M} \in \mathbb{C}^{m\times m\times(2p+1)\times(2q+1)\times(2r+1)}$*:*

Proof. We first prove that if $\mathbf{J}$ is a skew-hermitian matrix, then:

$$\mathbf{L} = \mathbf{M} - \text{conv3d_transpose}(\mathbf{M})$$

Let $\mathbf{J}^{(i,j)}$ denote the block of size $n^3 \times n^3$ as follows:

$$\mathbf{J}^{(i,j)} = \mathbf{J}_{in^3:(i+1)n^3,jn^3:(j+1)n^3}$$

Since $\mathbf{J}$ is skew-hermitian, we have:

$$\mathbf{J}^{(i,j)} = -\left(\mathbf{J}^{(j,i)}\right)^H, \quad \forall\, i,\ j \in [m-1]$$

It is readily observed that $\mathbf{J}^{(i,j)}$ corresponds to the jacobian of convolution with 1×1 filter $\mathbf{L}_{i:i+1,j:j+1,:,:,:}$. For some given filter $\mathbf{A}$, we use $\mathbf{A}^{(i,j)}$ to denote the 1×1 filter $\mathbf{A}_{i:i+1,j:j+1,:,:,:}$ for

simplicity. Thus, the above equation can be rewritten as:

$$\mathbf{L}^{(i,j)} = -\text{conv3d_transpose}\left(\mathbf{L}^{(j,i)}\right), \quad \forall\, i,\ j \in [m-1] \tag{7.33}$$

Now construct a filter $\mathbf{M}$ such that for $i \neq j$:

$$\mathbf{M}^{(i,j)} = \begin{cases} \mathbf{L}^{(i,j)}, & i < j \\ \mathbf{0}, & i > j \end{cases} \tag{7.34}$$

For $i = j$, $\mathbf{M}$ is given as follows:

$$\mathbf{M}^{(i,i)}_{s,t,u} = \begin{cases} \mathbf{L}^{(i,i)}_{s,t,u}, & s \leq p-1 \\ \mathbf{L}^{(i,i)}_{s,t,u}, & s = p,\ t \leq q-1 \\ \mathbf{L}^{(i,i)}_{s,t,u}, & s = p,\ t = q,\ u \leq r-1 \\ 0.5 \times \mathbf{L}^{(i,i)}_{s,t,u}, & s = p,\ t = q,\ u = r \\ 0, & otherwise \end{cases} \tag{7.35}$$

Next, our goal is to show that:

$$\mathbf{L} = \mathbf{M} - \text{conv3d_transpose}(\mathbf{M})$$

Now by the definition of $\mathrm{conv3d_transpose}$, we have:

$$\begin{aligned}
&[\mathbf{M} - \mathrm{conv3d_transpose}(\mathbf{M})]^{(i,j)} \\
&= \mathbf{M}^{(i,j)} - [\mathrm{conv3d_transpose}(\mathbf{M})]^{(i,j)} \\
&= \mathbf{M}^{(i,j)} - \mathrm{conv3d_transpose}\left(\mathbf{M}^{(j,i)}\right)
\end{aligned} \tag{7.36}$$

Case 1: For $i < j$, using equations (7.33) and (7.34):

$$\mathbf{M}^{(i,j)} - \mathrm{conv3d_transpose}\left(\mathbf{M}^{(j,i)}\right) = \mathbf{M}^{(i,j)} = \mathbf{L}^{(i,j)}$$

Case 2: For $i > j$, using equations (7.33) and (7.34):

$$\begin{aligned}
&\mathbf{M}^{(i,j)} - \mathrm{conv3d_transpose}\left(\mathbf{M}^{(j,i)}\right) \\
&= -\mathrm{conv3d_transpose}\left(\mathbf{M}^{(j,i)}\right) \\
&= -\mathrm{conv3d_transpose}\left(\mathbf{L}^{(j,i)}\right) = \mathbf{L}^{(i,j)}
\end{aligned}$$

Case 3: For $i = j$, we further simplify equation (7.36):

$$\begin{aligned}
&\mathbf{M}^{(i,i)}_{s,t,u} - \left[\mathrm{conv3d_transpose}\left(\mathbf{M}^{(i,i)}\right)\right]_{s,t,u} \\
&= \mathbf{M}^{(i,i)}_{s,t,u} - \overline{\mathbf{M}^{(i,i)}_{2p-s,2q-t,2r-u}}
\end{aligned} \tag{7.37}$$

Subcase 3(a): For $(s \leq p-1)$ or $(s = p,\ t \leq q-1)$ or $(s = p,\ t = q,\ u \leq r-1)$, we have:

$$\mathbf{M}^{(i,i)}_{2p-s,2q-t,2r-u} = 0$$

Thus for $(s \leq p-1)$ or $(s = p,\ t \leq q-1)$ or $(s = p,\ t = q,\ u \leq r-1)$: equation (7.37) simplifies to $\mathbf{M}^{(i,i)}_{s,t,u}$. The result follows trivially from the very definition of $\mathbf{L}^{(i,i)}_{s,t,u}$, i.e equation (7.35).

Subcase 3(b): For $(s \geq p+1)$ or $(s = p,\ t \geq q+1)$ or $(s = p,\ t = q,\ u \geq r+1)$, we have:

$$\mathbf{M}^{(i,i)}_{s,t,u} = 0$$

Thus, equation (7.14) simplifies to:

$$\mathbf{M}^{(i,i)}_{s,t,u} - \overline{\mathbf{M}^{(i,i)}_{2p-s,2q-t,2r-u}} = -\overline{\mathbf{M}^{(i,i)}_{2p-s,2q-t,2r-u}}$$

Since $(s \geq p+1)$ or $(s = p,\ t \geq q+1)$ or $(s = p,\ t = q,\ u \geq r+1)$, we have: $(2p-s \leq p-1)$ or $(2p-s = p,\ 2q-t \leq q-1)$ or $(2p-s = p,\ 2q-t = q,\ 2u-r \leq r-1)$ respectively. Thus using equation (7.35), we have:

$$-\overline{\mathbf{M}^{(i,i)}_{2p-s,2q-t,2r-u}} = -\overline{\mathbf{L}^{(i,i)}_{2p-s,2q-t,2r-u}}$$

Since $\mathbf{L}^{(i,i)}$ is a skew-hermitian filter, we have from Theorem 17:

$$\mathbf{L}^{(i,i)}_{s,t,u} = -\overline{\mathbf{L}^{(i,i)}_{2p-s,2q-t,2r-u}}$$

Thus in this subcase, equation (7.37) simplifies to $\mathbf{L}^{(i,i)}_{s,t,u}$ again.

Subcase 3(c): For $s = p,\ t = q,\ u = r$, since $\mathbf{L}^{(i,i)}$ is a skew-hermitian filter, we have:

$$\mathbf{L}^{(i,i)}_{p,q,r} = -\overline{\mathbf{L}^{(i,i)}_{p,q,r}}$$

$$\mathbf{L}^{(i,i)}_{p,q,r} + \overline{\mathbf{L}^{(i,i)}_{p,q,r}} = 0$$

Thus, $\mathbf{L}^{(i,i)}_{p,q,r}$ is a purely imaginary number. In this subcase

$$\mathbf{M}^{(i,i)}_{s,t,u} - \overline{\mathbf{M}^{(i,i)}_{2p-s,2q-t,2r-u}}$$

$$= \mathbf{M}^{(i,i)}_{p,q,r} - \overline{\mathbf{M}^{(i,i)}_{p,q,r}} = 2\mathbf{M}^{(i,i)}_{p,q,r}$$

Using equation (7.35) , we have:

$$2\mathbf{M}^{(i,i)}_{p,q,r} = \mathbf{L}^{(i,i)}_{p,q,r}$$

Thus, we get:

$$\mathbf{M}^{(i,i)}_{p,q,r} - \left[\text{conv3d_transpose}\left(\mathbf{M}^{(i,i)}\right)\right]_{p,q,r} = \mathbf{L}^{(i,i)}_{p,q,r}$$

Thus we have established: $\mathbf{L} = \mathbf{M} - \text{conv3d_transpose}(\mathbf{M})$. Note that the opposite direc-

tion of the if and only if statement follows trivially from the above proof. □

7.C MaxMin Activation function

Given a feature map $\mathbf{X} \in \mathbb{R}^{2m \times n \times n}$ (we assume the number of channels in $\mathbf{X}$ is a multiple of 2), to apply the MaxMin activation function, we first divide the input into two chunks of equal size: $\mathbf{A}$ and $\mathbf{B}$ such that:

$$\mathbf{A} = \mathbf{X}_{:m,:,:}$$

$$\mathbf{B} = \mathbf{X}_{m:,:,:}$$

Then the MaxMin activation function is given as follows:

$$\mathrm{MaxMin}(\mathbf{X})_{:m,:,:} = \max(\mathbf{A}, \mathbf{B})$$

$$\mathrm{MaxMin}(\mathbf{X})_{m:,:,:} = \min(\mathbf{A}, \mathbf{B})$$

Chapter 8: Novel activations, regularization and certification procedures for lipschitz networks

8.1 Introduction

Given a neural network $f : \mathbb{R}^d \to \mathbb{R}^k$, the Lipschitz constant[1] $\mathrm{Lip}(f)$ enforces an upper bound on how much the output is allowed to change in proportion to a change in the input. Previous work has demonstrated that a small Lipschitz constant is useful for improved adversarial robustness [15, 228], generalization bounds [205, 227], interpretable gradients [229] and Wasserstein distance estimation [230]. $\mathrm{Lip}(f)$ also upper bounds the increase in the norm of gradient during backpropagation and can thus prevent gradient explosion during training, enabling us to train very deep networks [203]. While heuristic methods to enforce Lipschitz constraints [196, 207] have achieved much practical success, they do not provably enforce a bound on $\mathrm{Lip}(f)$ globally and it remains challenging to achieve similar results when $\mathrm{Lip}(f)$ is provably bounded.

Using the property: $\mathrm{Lip}(g \circ h) \leq \mathrm{Lip}(g)\ \mathrm{Lip}(h)$, the Lipschitz constant of the neural network can be bounded by the product of the Lipschitz constant of all layers. While this allows us to construct 1-Lipschitz neural networks by constraining each layer to be 1-Lipschitz, [133] identified a key difficulty with this approach. Because a 1-Lipschitz layer can only reduce the norm of gradient during backpropagation, backprop through each layer reduces the gradient norm, re-

[1] Unless specified, we assume the Lipschitz constant under the l_2 norm in this chapter.

sulting in small gradient values for layers closer to the input, making training slow and difficult. To address this problem, they introduce Gradient Norm Preserving (GNP) architectures where each layer preserves the gradient norm during backpropagation. This involves constraining the Jacobian of each linear layer to be an orthogonal matrix and using a GNP activation function called $\mathrm{GroupSort}$. $\mathrm{GroupSort}$ activation function [133] first separates the vector of preactivations $\mathbf{z} \in \mathbb{R}^m$ into groups of pre-specified sizes, sorts each group in the descending order and then concatenates these sorted groups. When the group size is 2, the resulting activation function is called MaxMin.

For 1-Lipschitz CNNs, the robustness certificate for a sample $\mathbf{x}$ from class l is computed as $\mathcal{M}_f(\mathbf{x})/\sqrt{2}$ where $\mathcal{M}_f(\mathbf{x}) = f_l(\mathbf{x}) - \max_{i \neq l} f_i(\mathbf{x})$. Naturally, larger values of $f_l(\mathbf{x})$ and smaller values of $\max_{j \neq l} f_j(\mathbf{x})$ will lead to larger certificates. This requires the last weight matrix in the network, denoted by $\mathbf{W} \in \mathbb{R}^{k \times m}$ (k is the number of classes, m is the dimension of the penultimate layer, $m > k$), to enforce the following constraints throughout training:

$$\forall j, \;\; \|\mathbf{W}_{j,:}\|_2 = 1, \qquad\qquad i \neq j, \;\; \mathbf{W}_{j,:} \perp \mathbf{W}_{i,:}$$

where $\mathbf{W}_{i,:}$ denotes the ith row of $\mathbf{W}$. Now suppose that for some input image with label l, we want to update $\mathbf{W}$ to increase the logit for the lth class. Since $\|\mathbf{W}_{l,:}\|_2$ is constrained to be 1, the logit can only be increased by changing the direction of the vector $\mathbf{W}_{l,:}$. Because the other rows $\{\mathbf{W}_{i,:},\ i \neq l\}$ are constrained to be orthogonal to $\mathbf{W}_{l,:}$, this further requires an update for all the rows of $\mathbf{W}$. Thus during training, any update made to learn some class must necessitate the forgetting of information relevant for the other classes. This can be particularly problematic when the number of classes k and thus the number of orthogonality constraints per row (i.e.,

$k-1$) is large (such as in CIFAR-100).

To address this limitation, we propose to keep the last weight layer of the network *unchanged*. But then the resulting function is no longer 1-Lipschitz and the certificate $\mathcal{M}_f(\mathbf{x})/\sqrt{2}$ is not valid. Thus, we introduce a new certification procedure that does not require the last weight layer of the network $\mathbf{W}$ to be orthogonal. Our certificate is then given by the following equation:

$$\min_{i \neq l} \frac{f_l(\mathbf{x}) - f_i(\mathbf{x})}{\|\mathbf{W}_{l,:} - \mathbf{W}_{i,:}\|_2}$$

However, a limitation of using the above certificate is that because the weight layers are completely unconstrained, larger norms of rows (i.e., $\|\mathbf{W}_{i,:}\|$) can result in larger values of $\|\mathbf{W}_{l,:} - \mathbf{W}_{i,:}\|_2$ and thus smaller certificate values. To address this limitation, we normalize all rows to be of unit norm before computing the logits. While this still requires all the rows to be of unit norm, their directions are now allowed to change freely thus preventing the need to update other rows and forgetting of learned information. We show that this provides significant improvements when the number of classes is large. We call this procedure *Last Layer Normalization* (abbreviated as *LLN*). On CIFAR-100, this significantly improves both the standard ($> 3\%$) and provable robust accuracy ($> 4\%$ at $\rho = 36/255$) across multiple 1-Lipschitz CNN architectures (Table 8.1). Here, ρ is the l_2 attack radius.

Another limitation of existing 1-Lipschitz CNNs [232, 242, 243] is that their robustness guarantees do not scale properly with the l_2 radius ρ. For example, the provable robust accuracy of [243] drops $\sim 30\%$ at $\rho = 108/255$ compared to $36/255$ on CIFAR-10 (Table 8.2). To address this limitation, we introduce a certificate regularization denoted by CR (Section 8.5) that when used along with the Householder activation results in significantly improved provable

robust accuracy at larger radius values with minimal loss in standard accuracy. On the CIFAR-10 dataset, we achieve significant improvements in the provable robust accuracy for large $\rho = 108/255$ (min gain of $+4.96\%$) across different architectures with minimal loss in the standard accuracy (max drop of -0.56%). Results are in Table 8.2.

Additionally, we characterize the MaxMin activation function as a special case of the more general *Householder* (HH) activations. Recall that given $\mathbf{z} \in \mathbb{R}^m$, the HH transformation is a linear function reflecting $\mathbf{z}$ about the hyperplane $\mathbf{v}^T\mathbf{x} = 0$ ($\|\mathbf{v}\|_2 = 1$), given by $(\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)\mathbf{z}$ where $\mathbf{I} - 2\mathbf{v}\mathbf{v}^T$ is orthogonal because $\|\mathbf{v}\|_2 = 1$. The *Householder* activation function $\sigma_{\mathbf{v}}$ is defined below:

$$\sigma_{\mathbf{v}}(\mathbf{z}) = \begin{cases} \mathbf{z}, & \mathbf{v}^T\mathbf{z} > 0, \\ (\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)\mathbf{z}, & \mathbf{v}^T\mathbf{z} \leq 0. \end{cases} \tag{8.1}$$

First, note that since $\mathbf{z} = (\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)\mathbf{z}$ along $\mathbf{v}^T\mathbf{z} = 0$, $\sigma_{\mathbf{v}}$ is continuous. Moreover, the Jacobian $\nabla_{\mathbf{z}}\, \sigma_{\mathbf{v}}$ is either $\mathbf{I}$ or $\mathbf{I} - 2\mathbf{v}\mathbf{v}^T$ (both orthogonal) implying $\sigma_{\mathbf{v}}$ is GNP. Since these properties hold $\forall\ \mathbf{v} : \|\mathbf{v}\|_2 = 1$, $\mathbf{v}$ can be learned during the training. In fact, we prove that any GNP piecewise linear function that changes from $\mathbf{Q}_1\mathbf{z}$ to $\mathbf{Q}_2\mathbf{z}$ where $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are square orthogonal matrices along $\mathbf{v}^T\mathbf{z} = 0$ must satisfy $\mathbf{Q}_2 = \mathbf{Q}_1\left(\mathbf{I} - 2\mathbf{v}\mathbf{v}^T\right)$ to be continuous (Theorem 19). Thus, this characterization proves that every $\mathrm{GroupSort}$ activation is a special case of the more general Householder activation function (example in Figure 8.1, discussion in Section 8.6).

In summary, we make the following contributions:

- We introduce a certification procedure *without* orthogonalizing the last linear layer called *Last Layer Normalization*. This procedure significantly enhances the standard and provable

robust accuracy when the number of classes is large. Using the LipConvnet-15 network on CIFAR-100, our modification achieves a gain of $+4.71\%$ in provable robust accuracy (at $\rho = 36/255$) with a gain of $+4.80\%$ in standard accuracy (Table 8.1).

- We introduce a *Certificate Regularizer* that significantly advances the provable robust accuracy with a small reduction in standard accuracy. Using LipConvnet-15 network on CIFAR-10, we achieve $+5.81\%$ improvement in provable robust accuracy (at $\rho = 108/255$) with only a -0.29% drop in standard accuracy over the existing methods (Table 8.2).

- We introduce a class of piecewise linear GNP activation functions called *Householder* or HH activations. We show that the MaxMin activation is a special case of the HH activation for certain settings. We prove that Householder transformations are *necessary* for any GNP piecewise linear function to be continuous (Theorem 19).

8.2 Related work

Provably Lipschitz convolutional neural networks: The class of fully connected neural networks (FCNs) which are Gradient Norm Preserving (GNP) and 1-Lipschitz were first introduced by [133]. They orthogonalize weight matrices and use $\mathrm{GroupSort}$ as the activation function to design each layer to be GNP. While there have been numerous works on enforcing Lipschitz constraints on convolution layers [134, 214, 225, 228, 234], they either enforce loose Lipschitz bounds or are not scalable to large networks. To ensure that the Lipschitz constraint on convolutional layers is tight, multiple recent works try to construct convolution layers with an orthogonal Jacobian matrix [232, 242, 243]. These approaches avoid the aforementioned issues and allow the training of large, provably 1-Lipschitz CNNs while achieving impressive results.

Provable defenses against adversarial examples: A provably robust classifier is one for which we can guarantee that the classifier's prediction remains constant within some region around the input. Most of the existing methods for provable robustness either bound the Lipschitz constant of the neural network or the individual layers [142, 145, 146, 166, 182, 187, 236, 237, 238, 244, 245, 246, 247]. However, these methods do not scale to large and practical networks on the ImageNet dataset [56]. To scale to such large networks, randomized smoothing [135, 136, 137, 190, 191, 192, 239, 240, 248] has been proposed as a *probabilistically certified defense*. However, certifying robustness with high probability requires generating a large number of noisy samples leading to high inference-time computation. In contrast, the defense we propose is deterministic and hence not comparable to randomized smoothing. While [249] provide deterministic robustness certificates using smoothing, their certificates are in the l_1 norm and not directly applicable for the l_2 threat model studied in this chapter. We discuss the differences between l_1 and l_2 certificates in Supp. Section 8.C.

8.3 Problem setup and Notation

For a vector $\mathbf{v}$, $\mathbf{v}_j$ denotes its jth element. For a matrix $\mathbf{A}$, $\mathbf{A}_{j,:}$ and $\mathbf{A}_{:,k}$ denote the jth row and k^{th} column respectively. Both $\mathbf{A}_{j,:}$ and $\mathbf{A}_{:,k}$ are assumed to be column vectors (thus $\mathbf{A}_{j,:}$ is the transpose of jth row of $\mathbf{A}$). $\mathbf{A}_{j,k}$ denotes the element in j^{th} row and k^{th} column of $\mathbf{A}$. $\mathbf{A}_{:j,:k}$ denotes the matrix containing the first j rows and k columns of $\mathbf{A}$. The same rules are directly extended to higher order tensors. $\mathbf{I}$ denotes the identity matrix, $\mathbb{R}$ to denote the field of real numbers. For $\theta \in \mathbb{R}$, $\mathbf{J}^+(\theta)$ and $\mathbf{J}^-(\theta)$ denote the orthogonal matrices with determinants $+1$

and -1 defined as follows:

$$\mathbf{J}^{+}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \qquad \mathbf{J}^{-}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{bmatrix} \tag{8.2}$$

We construct a 1-Lipschitz neural network, $f : \mathbb{R}^d \to \mathbb{R}^k$ (d is the input dimension, k is the number of classes) by composing 1-Lipschitz convolution layers and GNP activation functions. To certify robustness for some input $\mathbf{x}$ with prediction l, we first define the margin of prediction: $\mathcal{M}_f(\mathbf{x}) = \max\left(0, f_l(\mathbf{x}) - \max_{i \neq l} f_i(\mathbf{x})\right)$ where $f_i(\mathbf{x})$ is the logit for class i and l is the correct label. Using Theorem 7 in [232], we can derive the robustness certificate (in the l_2 norm) as $\mathcal{M}_f(\mathbf{x})/\sqrt{2}$. Thus, the l_2 distance of $\mathbf{x}$ to the decision boundary is lower bounded by $\mathcal{M}_f(\mathbf{x})/\sqrt{2}$:

$$\min_{i \neq l} \min_{f_i(\mathbf{x}^*) = f_l(\mathbf{x}^*)} \|\mathbf{x}^* - \mathbf{x}\|_2 \geq \frac{\mathcal{M}_f(\mathbf{x})}{\sqrt{2}} \tag{8.3}$$

We often use the abbreviation $f_i - f_j : \mathbb{R}^D \to \mathbb{R}$ to denote the function so that:

$$\left(f_i - f_j\right)(\mathbf{x}) = f_i(\mathbf{x}) - f_j(\mathbf{x}), \qquad \forall\, \mathbf{x} \in \mathbb{R}^D$$

Our goal is to train the neural network f to achieve the maximum possible provably robust accuracy while also simultaneously improving (or maintaining) standard accuracy.

8.4 Last Layer Normalization

To ensure that the network is 1-Lipschitz so that the certificate in equation (8.3) is valid, existing 1-Lipschitz neural networks require the weight matrices of all the linear layers of the network to

be orthogonal. For the weight matrix in the last layer of the network (that maps the penultimate layer neurons to the logits), $\mathbf{W} \in \mathbb{R}^{k \times m}$ (k is the number of classes, m is the dimension of the penultimate layer, $m > k$), this enforces the following constraints on each row $\mathbf{W}_{i,:} \in \mathbb{R}^m$:

$$\forall j, \;\; \|\mathbf{W}_{j,:}\|_2 = 1, \qquad \forall i \neq j, \;\; \mathbf{W}_{j,:} \perp \mathbf{W}_{i,:} \tag{8.4}$$

Now suppose that for some input $\mathbf{x}$ with label l, we want to update $\mathbf{W}$ to increase the logit for the lth class. Since $\|\mathbf{W}_{l,:}\|_2$ is constrained to be 1, the gradient update can only change the direction of the vector $\mathbf{W}_{l,:}$. But now, because the other rows $\{\mathbf{W}_{i,:}, \; i \neq l\}$ are constrained to be orthogonal to $\mathbf{W}_{l,:}$, this further requires an update for all the rows of $\mathbf{W}$. This has the negative effect that during training, any update made to learn some class must necessitate the forgetting of information relevant for the other classes. This can be particularly problematic when the number of classes k is large (such as in CIFAR-100) and thus the number of orthogonality constraints per row i.e. $k - 1$ is large.

To address this limitation, observe that the neural network from the input layer to the penultimate layer (i.e excluding the last linear layer) is 1-Lipschitz. Let $g : \mathbb{R}^d \to \mathbb{R}^m$ be this function so that $f(\mathbf{x}) = \mathbf{W}g(\mathbf{x}) + \mathbf{b}$. This equation suggests that even if $\mathbf{W}$ is not orthogonal, the Lipschitz constant of the function $f_l - f_i$, can be computed by multiplying the Lipschitz constant of g (which is 1) and that of $(\mathbf{W}_{l,:} - \mathbf{W}_{i,:})$ (which is $\|\mathbf{W}_{l,:} - \mathbf{W}_{i,:}\|_2$). The robustness certificate can then be computed as $\mathcal{M}_f(\mathbf{x})/\|\mathbf{W}_{l,:} - \mathbf{W}_{i,:}\|_2$. This procedure leads to the following proposition:

Proposition 2. *Given* 1*-Lipschitz continuous function* $g : \mathbb{R}^d \to \mathbb{R}^m$ *and* $\mathbf{W} \in \mathbb{R}^{k \times m}$, $\mathbf{b} \in \mathbb{R}^k$ *(*k *is the number of classes), construct a new function* $f : \mathbb{R}^d \to \mathbb{R}^k$ *defined as:* $f(\mathbf{x}) = \mathbf{W}g(\mathbf{x}) + \mathbf{b}$.

Let $f_l(\mathbf{x}) > \max_{i \neq l} f_i(\mathbf{x})$. The robustness certificate (under the l_2 norm) is given by:

$$\min_{i \neq l} \min_{f_l(\mathbf{x}^*) = f_i(\mathbf{x}^*)} \|\mathbf{x}^* - \mathbf{x}\|_2 \geq \min_{i \neq l} \frac{f_l(\mathbf{x}) - f_i(\mathbf{x})}{\|\mathbf{W}_{l,:} - \mathbf{W}_{i,:}\|_2}$$

Proof is in Supp. Section 8.A.1. However, in our experiments, we found that using this procedure directly i.e without any constraint on the weight matrix $\mathbf{W}$ often results in large norms of row vectors $\|\mathbf{W}_{i,:}\|_2$, and thus large values of $\|\mathbf{W}_{l,:} - \mathbf{W}_{i,:}\|_2$ and smaller certificates (Theorem 2). To address this problem, we normalize all rows of the matrix to be of unit norm before computing the logits so that for the input $\mathbf{x}$, the logit $g_i(\mathbf{x})$ can be computed as follows:

$$g_i(\mathbf{x}) = \frac{(\mathbf{W}_{i,:})^T f(\mathbf{x})}{\|\mathbf{W}_{i,:}\|_2} + \mathbf{b}_i$$

The robustness certificate can then be computed as follows:

$$\min_{i \neq l} \frac{g_l(\mathbf{x}) - g_i(\mathbf{x})}{\|\mathbf{W}^{(n)}_{l,:} - \mathbf{W}^{(n)}_{i,:}\|_2}, \qquad \text{where } \forall j, \;\; \mathbf{W}^{(n)}_{j,:} = \frac{\mathbf{W}_{j,:}}{\|\mathbf{W}_{j,:}\|_2}, \; g_j(\mathbf{x}) = \left(\mathbf{W}^{(n)}_{j,:}\right)^T f(\mathbf{x}) + \mathbf{b}_j$$

While each row $\mathbf{W}_{i,:}$ is still constrained to be of unit norm, unlike with orthogonality constraints (eq. (8.4)), their directions are allowed to change freely. This provides significant improvements when the number of classes is large. We call this procedure *Last Layer Normalization* (*LLN*).

8.5 Certificate Regularization

A limitation of using cross entropy loss for training 1-Lipschitz CNNs is that it is not explicitly designed to maximize the margin $\mathcal{M}_f(\mathbf{x})$ and thus, the robustness certificate. That is, once the

cross entropy loss becomes small, the gradients will no longer try to further increase the margin $\mathcal{M}_f(\mathbf{x})$ even though the network may have the capacity to learn bigger margins.

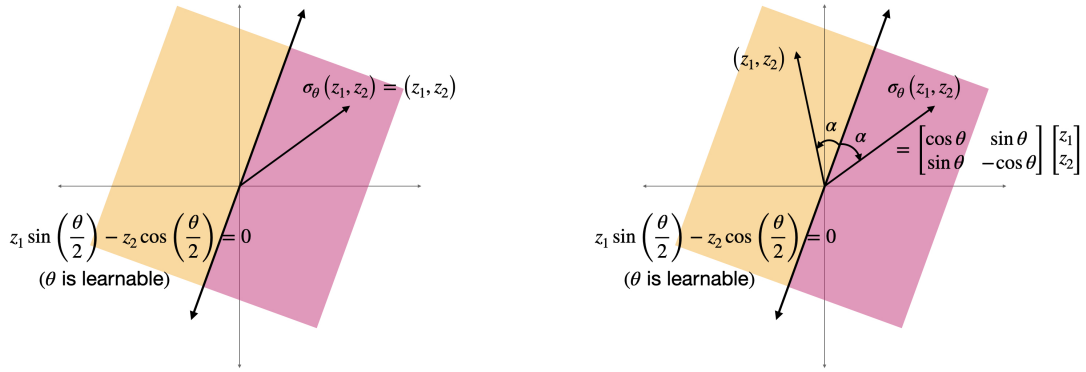
To address this limitation, we can simply *subtract* the certificate i.e $-\mathcal{M}_f(\mathbf{x})/\sqrt{2}$ from the usual cross entropy loss function during training. Observe that we subtract the certificate because we want to *maximize* the certificate values while *minimizing* the cross entropy loss. However, in our experiments we found that this regularization term excessively penalizes the network for the misclassified examples and as a result, the certificate values for the correctly classified inputs are not large. Thus, we propose to use the following regularized loss function during training:

$$\min_{\Omega} \quad \mathbb{E}_{(\mathbf{x},l)\sim D}\left[\ell\left(f_\Omega(\mathbf{x}), l\right) - \gamma \text{ relu}\left(\frac{\mathcal{M}_f(\mathbf{x})}{\sqrt{2}}\right)\right] \tag{8.5}$$

In the above equation, f_Ω denotes the 1-Lipschitz neural network parametrized by Ω, $f_\Omega(\mathbf{x})$ denotes the logits for the input $\mathbf{x}$, $\ell\left(f_\Omega(\mathbf{x}), l\right)$ is the cross entropy loss for input $\mathbf{x}$ with label l and $\gamma > 0$ is the regularization coefficient for maximizing the certificate. We have the minus sign in front of the regularization term γ relu$(\mathcal{M}_f(\mathbf{x})/\sqrt{2})$ because we want to *maximize* the certificate while *minimizing* the cross entropy loss. For wrongly classified inputs, $\mathcal{M}_f(\mathbf{x})/\sqrt{2} < 0 \implies$ relu$(\mathcal{M}_f(\mathbf{x})/\sqrt{2}) = 0$. This ensures that the optimization tries to increase the certificates *only* for the correctly classified inputs. We call the above procedure *Certificate Regularization* (*CR*).

8.6 Householder Activation Functions

Recall that given $\mathbf{z} \in \mathbb{R}^m$, the Householder (HH) transformation reflects $\mathbf{z}$ about the hyperplane $\mathbf{v}^T\mathbf{x} = 0$ where $\|\mathbf{v}\|_2 = 1$. The linear transformation is given by the equation $(\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)\mathbf{z}$ where $\mathbf{I} - 2\mathbf{v}\mathbf{v}^T$ is orthogonal because $\|\mathbf{v}\|_2 = 1$. Now, consider the nonlinear function $\sigma_\mathbf{v}$ below:



(a) Case 1: $z_1 \sin(\theta/2) - z_2 \cos(\theta/2) > 0$ (b) Case 2: $z_1 \sin(\theta/2) - z_2 \cos(\theta/2) \leq 0$

Figure 8.1: Illustration of the Householder activation, σ_θ. In each colored region, σ_θ is linear. The Jacobian is $\mathbf{I}$ when (z_1, z_2) lies in the pink region (Case 1) and $\mathbf{I} - 2\mathbf{v}\mathbf{v}^T$ in the other region (Case 2) where $\mathbf{v} = \begin{bmatrix} \sin(\theta/2) & -\cos(\theta/2) \end{bmatrix}^T$. Both of these matrices are orthogonal implying σ_θ is GNP.

Definition 9. ***(Householder Activation of Order*** 1*)*** *The activation function* $\sigma_{\mathbf{v}} : \mathbb{R}^m \to \mathbb{R}^m$*, applied on* $\mathbf{z} \in \mathbb{R}^m$*, is called the* m*-dimensional Householder Activation of Order* 1*:*

$$\sigma_{\mathbf{v}}(\mathbf{z}) = \begin{cases} \mathbf{z}, & \mathbf{v}^T\mathbf{z} > 0, \\ (\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)\mathbf{z}, & \mathbf{v}^T\mathbf{z} \leq 0. \end{cases} \tag{8.6}$$

Since $\sigma_{\mathbf{v}}$ is linear when $\mathbf{v}^T\mathbf{z} > 0$ or $\mathbf{v}^T\mathbf{z} < 0$, it is also continuous in both cases. At the hyperplane separating the two cases (i.e., $\mathbf{v}^T\mathbf{z} = 0$) we have: $(\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)\mathbf{z} = \mathbf{z} - 2(\mathbf{v}^T\mathbf{z})\mathbf{v} = \mathbf{z}$ (both linear functions are equal). Thus, $\sigma_{\mathbf{v}}$ is continuous $\forall\ \mathbf{z} \in \mathbb{R}^m$. Moreover, the Jacobian is either $\mathbf{I}$ or $\mathbf{I} - 2\mathbf{v}\mathbf{v}^T$ which are both square orthogonal matrices. Thus, $\sigma_{\mathbf{v}}$ is also GNP and 1-Lipschitz. Since these properties hold for all $\mathbf{v}$ satisfying $\|\mathbf{v}\|_2 = 1$, $\mathbf{v}$ can be learnable.

While the above arguments suggest that HH transformations are *sufficient* to ensure such functions are continuous, we also prove that they are *necessary*. That is, we prove that if a GNP piecewise linear function $g : \mathbb{R}^m \to \mathbb{R}^m$ transitions between different linear functions $\mathbf{Q}_1\mathbf{z}$ and

$\mathbf{Q}_2\mathbf{z}$ (in an open set $S \subset \mathbb{R}^m$) along a hyperplane $\mathbf{v}^T\mathbf{z} = 0$ (where $\|\mathbf{v}\|_2 = 1$), then g is continuous in S *if and only if* $\mathbf{Q}_2 = \mathbf{Q}_1(\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)$. This theoretical result provides a general principle for designing piecewise linear GNP activations. The formal result is stated in the following Theorem:

Theorem 19. *Given an open set $S \subset \mathbb{R}^m$, orthogonal square matrices $\mathbf{Q}_1 \neq \mathbf{Q}_2$, and vector $\mathbf{v} \in \mathbb{R}^m$ ($\|\mathbf{v}\|_2 = 1$) such that $S \cap \{\mathbf{z} : \mathbf{v}^T\mathbf{z} = 0\} \neq \emptyset$, the function g defined as follows:*

$$g(\mathbf{z}) = \begin{cases} \mathbf{Q}_1\mathbf{z}, & \mathbf{z} \in S, \mathbf{v}^T\mathbf{z} > 0, \\ \mathbf{Q}_2\mathbf{z}, & \mathbf{z} \in S, \mathbf{v}^T\mathbf{z} \leq 0 \end{cases} \tag{8.7}$$

is continuous in S if and only if $\mathbf{Q}_2 = \mathbf{Q}_1(\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)$.

Proof of Theorem 19 is in Supp. 8.A.2. Note that since the matrix $\mathbf{I} - 2\mathbf{v}\mathbf{v}^T$ has determinant -1, the above theorem necessitates that $\det(\mathbf{Q}_1) = -\det(\mathbf{Q}_2)$ i.e the determinant of the Jacobian must change sign whenever the Jacobian of a piecewise linear GNP activation changes.

Recall that for the MaxMin activation, $\mathrm{MaxMin}(z_1, z_2) = (z_1, z_2)$ if $z_1 > z_2$ and (z_2, z_1) otherwise. Thus, the Jacobian of MaxMin for $z_1 > z_2$ is $\mathbf{I} = \mathbf{J}^+(0)$ while for $z_1 \leq z_2$ is $\mathbf{J}^-(\pi/2)$. Using Theorem 19, we can easily prove that MaxMin is a special case of the more general Householder activations where the Jacobian $\mathbf{J}^-(\pi/2)$ is replaced with $\mathbf{J}^-(\theta)$ and the conditions $z_1 > z_2$ are replaced with $z_1 \sin(\theta/2) > z_2 \cos(\theta/2)$ (similarly for $\leq$). The construction of Householder activations in 2 dimensions, denoted by σ_θ, is given in the corollary below:

Corollary 2. *The function* $\sigma_\theta : \mathbb{R}^2 \to \mathbb{R}^2$ *defined as*

$$\sigma_\theta\left(z_1, z_2\right) = \begin{cases} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} & \text{if } \; z_1 \sin\left(\theta/2\right) - z_2 \cos\left(\theta/2\right) > 0 \\ \begin{bmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} & \text{if } \; z_1 \sin\left(\theta/2\right) - z_2 \cos\left(\theta/2\right) \leq 0 \end{cases} \tag{8.8}$$

is continuous and is called 2D *Householder Activation of Order* 1.

The two cases are demonstrated in Figure 8.1a and Figure 8.1b, respectively. Since σ_θ is continuous, GNP and 1-Lipschitz $\forall\ \theta \in \mathbb{R},\ \theta$ is a learnable parameter. For $\theta = \pi/2$ in equation (8.8), σ_θ is equivalent to MaxMin. Thus, σ_θ is at least as expressive as MaxMin.

A major limitation of using $\sigma_{\mathbf{v}}$ (equation (8.6)) directly is that it has only 2 linear regions and is thus limited in its expressive power. In contrast, MaxMin first divides the preactivation $\mathbf{z} \in \mathbb{R}^m$ (assuming m is divisible by 2) into $m/2$ groups of size 2 each. Since each group has 2 linear regions, we get $2^{m/2}$ linear regions from the $m/2$ groups. Thus, to increase the expressive power, we similarly divide $\mathbf{z}$ into $m/2$ groups of size 2 each and apply the 2-dimensional Householder activation of Order 1 (σ_θ) to each group resulting in $2^{m/2}$ linear regions (same as MaxMin). To apply σ_θ to the output of a convolution layer $\mathbf{z} \in \mathbb{R}^{m \times n \times n}$ (m is the number of channels, $n \times n$ is the spatial size), we first split $\mathbf{z}$ into 2 tensors along the channel dimension giving the tensors: $\mathbf{z}_{:m/2,:,:}$ and $\mathbf{z}_{m/2:,:,:}$. Each of these tensors is of size $m/2 \times n \times n$ giving $n^2 m/2$ groups. We use the same θ for each pair of channels (irrespective of spatial location) resulting in $m/2$ learnable parameters. We initialize $\theta = \pi/2$ so that σ_θ is equivalent to MaxMin at initialization.

8.7 Experiments

Our goal is to evaluate the effectiveness of the three changes proposed in this chapter: (a) Last Layer Normalization, (b) Certificate regularization and (c) Householder activations. We perform experiments under the setting of provably robust image classification on CIFAR-10 and CIFAR-100 datasets using the same 1-Lipschitz CNN architectures used by [243] (LipConvnet-5, 10, 15, . . . , 40) due to their superior performance over the prior works. We compare with the three orthogonal convolution layers in the literature: SOC [243], BCOP [232] and Cayley [242] using MaxMin as the activation function.

We use SOC with MaxMin as the primary baseline for comparison in the maintext due to their superior performance over the prior works (BCOP, Cayley). Results using BCOP and Cayley convolutions are given in Supp. Sections 8.F.2 and 8.F.3 for completeness. We use the same implementations for these convolution layers as given in their respective github repositories. We compare the provable robust accuracy using 3 different values of the l_2 perturbation radius $\rho = 36/255,\ 72/255,\ 108/255$. In both Tables 8.1 and 8.2, for all networks, we use SOC as the convolution layer. Using SOC, we achieve the same bound on the approximation error of an orthogonal Jacobian as achieved in [243] i.e. 2.42×10^{-6} across all convolution layers in all networks. Thus, even for the network with 40 layers, this results in maximum Lipschitz constant of $(1+2.42\times10^{-6})^{40} = 1.000097 \leq 1.0001$. Thus, the Lipschitz constant across all our networks is bounded by 1.0001. The symbol HH (in Tables 8.1, 8.2) is for the $2\mathrm{D}$ Householder Activation of order 1 or σ_θ (defined in equation (8.8)).

All experiments were performed using 1 NVIDIA GeForce RTX 2080 Ti GPU. All networks were trained for 200 epochs with initial learning rate of 0.1, dropped by a factor of 0.1

Lip Conv net-	Methods	Clean Acc.	Provable Robust Acc. ($\rho =$) 36/255	72/255	108/255	Increase (Clean)
5	Baseline	42.71%	27.86%	17.45%	9.99%	_
	+ LLN	45.86%	31.93%	21.17%	13.21%	+3.15%
	+ HH	**46.36%**	32.64%	21.19%	13.12%	**+3.65%**
	+ CR	45.82%	**32.99%**	**22.48%**	**14.79%**	+3.11%
10	Baseline	43.72%	29.39%	18.56%	11.16%	_
	+ LLN	46.88%	33.32%	22.08%	13.87%	+3.16%
	+ HH	**47.96%**	34.30%	22.35%	14.48%	**+4.24%**
	+ CR	47.07%	**34.53%**	**23.50%**	**15.66%**	+3.35%
15	Baseline	42.92%	28.81%	17.93%	10.73%	_
	+ LLN	47.72%	33.52%	21.89%	13.76%	+4.80%
	+ HH	**47.72%**	33.97%	22.45%	13.81%	**+4.80%**
	+ CR	47.61%	**34.54%**	**23.16%**	**15.09%**	+4.69%
20	Baseline	43.06%	29.34%	18.66%	11.20%	_
	+ LLN	46.86%	33.48%	22.14%	14.10%	+3.80%
	+ HH	47.71%	34.22%	22.93%	14.57%	+4.65%
	+ CR	**47.84%**	**34.77%**	**23.70%**	**15.84%**	**+4.78%**
25	Baseline	43.37%	28.59%	18.18%	10.85%	_
	+ LLN	46.32%	32.87%	21.53%	13.86%	+2.95%
	+ HH	**47.70%**	34.00%	22.67%	14.57%	**+4.33%**
	+ CR	46.87%	**34.09%**	**23.41%**	**15.61%**	+3.50%
30	Baseline	42.87%	28.74%	18.47%	11.21%	_
	+ LLN	46.18%	32.82%	21.52%	13.52%	+3.31%
	+ HH	46.80%	33.72%	22.70%	14.31%	+3.93%
	+ CR	**46.92%**	**34.17%**	**23.21%**	**15.84%**	**+4.05%**
35	Baseline	42.42%	28.34%	18.10%	10.96%	_
	+ LLN	45.22%	32.10%	21.28%	13.25%	+2.80%
	+ HH	46.21%	32.80%	21.55%	14.13%	+3.79%
	+ CR	**46.88%**	**33.64%**	**23.34%**	**15.73%**	**+4.46%**
40	Baseline	41.84%	28.00%	17.40%	10.28%	_
	+ LLN	42.94%	29.71%	19.30%	11.99%	+1.10%
	+ HH	**45.84%**	**32.79%**	21.98%	14.07%	**+4.00%**
	+ CR	45.03%	32.57%	**22.37%**	**14.76%**	+3.19%

Table 8.1: Results for provable robustness against adversarial examples on the CIFAR-100 dataset. Results with BCOP and Cayley convolutions are in Supp. Tables 8.10 and 8.11.

after 100 and 150 epochs. For Certificate Regularization (or CR), we set the parameter $\gamma = 0.5$.

We observe negligible inference time overhead due to any of our changes (Table 8.3).

8.7.1 Results on CIFAR-100

In Table 8.1, for each architecture, the row "Baseline" uses the MaxMin activation, "+ LLN" adds Last Layer Normalization (uses MaxMin), "+ HH" replaces MaxMin with σ_θ (also uses LLN) , "+ CR" also adds Certificate Regularization with $\gamma = 0.1$ (uses both σ_θ and LLN). The column, "Increase (Clean)" denotes the increase in standard accuracy relative to "Baseline".

By adding LLN (the row "+ LLN"), we observe gains in standard (min gain of 1.10%) and provable robust accuracy (min gain of 1.71% at $\rho = 36/255$) across all the LipConvnet architectures (gains relative to "Baseline"). These gains are smallest for the LipConvnet-40 network with the maximum depth. However, replacing MaxMin with the σ_θ activation further improves the standard (min gain of 3.65%) and provable robust accuracy (min gain of 4.46% at $\rho = 36/255$) across all networks (again relative to "Baseline"). We observe that replacing MaxMin with σ_θ ("+ HH") significantly improves the performance of the deeper LipConvnet-$35, 40$ networks.

Adding CR further improves the provable robust accuracy while only slightly reducing the standard accuracy. Because LLN significantly improves the standard accuracy, we compare the standard accuracy numbers between rows "+ CR" and "+ LLN" to evaluate the drop due to CR. We observe a small drop in standard accuracy ($-0.04\%, -0.11\%$) only for LipConvnet-5 and LipConvnet-15 networks. For the other networks, the standard accuracy actually *increases*.

8.7.2 Results on CIFAR-10

In Table 8.2, for each architecture, the row "Baseline" uses MaxMin, the row "+ HH" uses σ_θ activation (replacing MaxMin) and the row "+ CR" also adds Certificate Regularization with $\gamma = 0.1$ (again using σ_θ activation). Due to the small number of classes in CIFAR-10, we do not

Lip Conv net-	Methods	Clean Acc.	Provable Robust Acc. ($\rho =$)			Increase
			36/255	72/255	108/255	(108/255)
5	Baseline	75.78%	59.18%	42.01%	27.09%	_
	+ HH	**76.30%**	60.12%	43.20%	27.39%	+0.30%
	+ CR	75.31%	**60.37%**	**45.62%**	**32.38%**	**+5.29%**
10	Baseline	76.45%	60.86%	44.15%	29.15%	_
	+ HH	**76.86%**	61.52%	44.91%	29.90%	+0.75%
	+ CR	76.23%	**62.57%**	**47.70%**	**34.15%**	**+5.00%**
15	Baseline	76.68%	61.36%	44.28%	29.66%	_
	+ HH	**77.41%**	61.92%	45.60%	31.10%	+1.44%
	+ CR	76.39%	**62.96%**	**48.47%**	**35.47%**	**+5.81%**
20	Baseline	76.90%	61.87%	45.79%	31.08%	_
	+ HH	**76.99%**	61.76%	45.59%	30.99%	-0.09%
	+ CR	76.34%	**62.63%**	**48.69%**	**36.04%**	**+4.96%**
25	Baseline	75.24%	60.17%	43.55%	28.60%	_
	+ HH	**76.37%**	61.50%	44.72%	29.83%	+1.23%
	+ CR	75.21%	**61.98%**	**47.93%**	**34.92%**	**+6.32%**
30	Baseline	74.51%	59.06%	42.46%	28.05%	_
	+ HH	**75.25%**	59.90%	43.85%	29.35%	+1.30%
	+ CR	74.23%	**60.64%**	**46.51%**	**34.08%**	**+6.03%**
35	Baseline	73.73%	58.50%	41.75%	27.20%	_
	+ HH	**75.44%**	61.05%	45.38%	30.85%	+3.65%
	+ CR	74.25%	**61.30%**	**47.60%**	**35.21%**	**+8.01%**
40	Baseline	71.63%	54.39%	37.92%	24.13%	_
	+ HH	**73.24%**	58.12%	42.24%	28.48%	+4.35%
	+ CR	72.59%	**59.04%**	**44.92%**	**32.87%**	**+8.74%**

Table 8.2: Results for provable robustness against adversarial examples on the CIFAR-10 dataset. Results with BCOP and Cayley convolutions are in Supp. Tables 8.5 and 8.6.

observe significant gains using Last Layer Normalization or LLN (Supp. Table 8.8). Thus, we do not include LLN for any of the results in Table 8.2. The column, "Increase $(108/255)$" denotes the increase in provable robust accuracy with $\rho = 108/255$ relative to "Baseline".

For LipConvnet-$25, 30, 35, 40$ architectures, we observe gains in both the standard and provable robust accuracy by replacing MaxMin with the HH activation (i.e σ_θ). Similar to what we observe for CIFAR-100 in Table 8.1, the gains in provable robust accuracy ($\rho = 108/255$) are significantly higher for deeper networks: LipConvnet-35 (3.65%) and LipConvnet-40 (4.35%)

Lip Conv net-	Running times (seconds)				
	CIFAR-10 results		CIFAR-100 results		
	Baseline	**HH**	**Baseline**	**LLN**	**LLN + HH**
5	3.7864	3.86	3.7568	3.5002	3.6673
10	5.3677	5.6014	5.3714	5.5482	5.5533
15	7.234	7.3503	7.3092	7.2595	7.3127
20	9.536	9.3753	9.005	9.2043	9.308
25	11.0512	11.2692	10.9321	10.7868	11.726
30	12.5135	13.6866	12.3198	13.2168	13.6275
35	14.5539	15.0921	14.427	14.575	15.7069
40	17.1907	17.1928	16.0911	16.2535	17.1015

Table 8.3: Inference times for various networks on the complete test sets of CIFAR-10 and CIFAR-100 with 10000 samples.

with decent gains in standard accuracy (1.71 and 1.61% respectively).

Again similar to CIFAR-100, adding CR further boosts the provable robust accuracy while slightly reducing the standard accuracy. Comparing "+ CR" with "Baseline", we observe small drops in standard accuracy for LipConvnet-$5, 10, \ldots, 30$ networks (max. drop of -0.56%), and gains for LipConvnet-35 ($+0.52\%$) and LipConvnet-40 ($+0.96\%$). For provable robust accuracy ($\rho = 108/255$), we observe very significant gains of $> 4.96\%$ for all networks and $> 8\%$ for the deeper LipConvnet-35, 40 networks.

8.8 Conclusion

In this chapter, we introduce a procedure to certify robustness of 1-Lipschitz CNNs without orthogonalizing of the last linear layer of the network. Additionally, we introduce a certificate regularization that significantly improves the provable robust accuracy for these models at higher l_2 radii. Finally, we introduce a class of GNP activation functions called Householder (or HH) activations and prove that the Jacobian of any Gradient Norm Preserving (GNP) piecewise linear

function is only allowed to change via Householder transformations for the function to be continuous which provides a general principle for designing piecewise linear GNP functions. These ideas lead to improved deterministic ℓ_2 robustness certificates on CIFAR-10 and CIFAR-100.

Supplementary material

8.A Proofs

8.A.1 Proof of Proposition 2

Proof. We proceed by computing the Lipschitz constant of the function $f_l - f_i$.

The gradient of the function: $f_l - f_i$ at $\mathbf{x}$ can be computed using the chain rule:

$$\nabla_{\mathbf{x}} \left(f_l - f_i \right) = \left(\mathbf{W}_{l,:} - \mathbf{W}_{i,:} \right)^T \nabla_{\mathbf{x}} g$$

Since g is given to be 1-Lipschitz, the Lipschitz constant of $f_l - f_i$ can be computed using the above equation as follows:

$$\| \nabla_{\mathbf{x}} \left(f_l - f_i \right) \|_2 \leq \| \left(\mathbf{W}_{l,:} - \mathbf{W}_{i,:} \right)^T \left(\nabla_{\mathbf{x}} g \right) \|_2$$

$$\| \nabla_{\mathbf{x}} \left(f_l - f_i \right) \|_2 \leq \| \mathbf{W}_{l,:} - \mathbf{W}_{i,:} \|_2 \| \nabla_{\mathbf{x}} g \|_2 \leq \| \mathbf{W}_{l,:} - \mathbf{W}_{i,:} \|_2$$

Thus, the distance of $\mathbf{x}$ to the decision boundary $f_l - f_i = 0$, is lower bounded by:

$$\min_{f_l(\mathbf{x}^*) = f_i(\mathbf{x}^*)} \| \mathbf{x}^* - \mathbf{x} \|_2 \ \geq \frac{f_l(\mathbf{x}) - f_i(\mathbf{x})}{\| \mathbf{W}_{l,:} - \mathbf{W}_{i,:} \|_2}$$

Thus, the distance to decision boundary across all classes $i \neq l$ is lower bounded by:

$$\min_{i \neq l} \ \min_{f_l(\mathbf{x}^*) = f_i(\mathbf{x}^*)} \| \mathbf{x}^* - \mathbf{x} \|_2 \ \geq \min_{i \neq l} \frac{f_l(\mathbf{x}) - f_i(\mathbf{x})}{\| \mathbf{W}_{l,:} - \mathbf{W}_{i,:} \|_2}$$

□

8.A.2 Proof of Theorem 19

Proof. We first prove that if $\mathbf{Q}_2 = (\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)\mathbf{Q}_1$, then the function g is continuous.

First, observe that for $\mathbf{v}^T\mathbf{z} > 0$, $g(\mathbf{z}) = \mathbf{Q}_1\mathbf{z}$ which is continuous.

Similarly, for $\mathbf{v}^T\mathbf{z} < 0$, $g(\mathbf{z}) = \mathbf{Q}_2\mathbf{z}$ which is again continuous.

This proves that the function g is continuous when $\mathbf{v}^T\mathbf{z} > 0$ or $\mathbf{v}^T\mathbf{z} < 0$.

Thus, to prove continuity $\forall\ \mathbf{z} \in S$, we must prove that:

$$\mathbf{Q}_1\mathbf{z} = \mathbf{Q}_2\mathbf{z} \qquad \forall\ \mathbf{z} : \mathbf{v}^T\mathbf{z} = 0 \tag{8.9}$$

Since $\mathbf{Q}_2 = \mathbf{Q}_1(\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)$, we have:

$$\mathbf{Q}_2 - \mathbf{Q}_1 = -2\mathbf{Q}_1\mathbf{v}\mathbf{v}^T$$

$$(\mathbf{Q}_2 - \mathbf{Q}_1)\,\mathbf{z} = -2\left(\mathbf{Q}_1\mathbf{v}\mathbf{v}^T\right)\mathbf{z} = -2\mathbf{Q}_1\mathbf{v}\left(\mathbf{v}^T\mathbf{z}\right)$$

The above equation directly proves (8.9).

Now, we prove the other direction i.e if g is continuous in S then, $\mathbf{Q}_2 = \mathbf{Q}_1(\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)$.

Since g is continuous for all $\mathbf{z} : \mathbf{v}^T\mathbf{z} = 0$, we have:

$$\mathbf{Q}_2\mathbf{z} = \mathbf{Q}_1\mathbf{z} \qquad \forall\ \mathbf{z} : \mathbf{v}^T\mathbf{z} = 0$$

$$(\mathbf{Q}_2 - \mathbf{Q}_1)\,\mathbf{z} = \mathbf{0} \qquad \forall\ \mathbf{z} : \mathbf{v}^T\mathbf{z} = 0$$

Since $\mathbf{z} \in \mathbb{R}^m$, we know that the set of vectors: $\{\mathbf{z} : \mathbf{v}^T\mathbf{z} = 0\}$ spans a $m-1$ dimensional subspace.

Thus, the null space of $\mathbf{Q}_2 - \mathbf{Q}_1$ is of size $m-1$.

This in turn implies that $\mathbf{Q}_2 - \mathbf{Q}_1$ is a rank one matrix given by the following equation:

$$\mathbf{Q}_2 - \mathbf{Q}_1 = \mathbf{u}\mathbf{v}^T \tag{8.10}$$

where the vector $\mathbf{u}$ remains to be determined.

Since $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are orthogonal matrices, we have the following set of equations:

$$\mathbf{Q}_2^T\mathbf{Q}_2 = \left(\mathbf{Q}_1 + \mathbf{u}\mathbf{v}^T\right)^T\left(\mathbf{Q}_1 + \mathbf{u}\mathbf{v}^T\right) \tag{8.11}$$

$$\mathbf{Q}_2\mathbf{Q}_2^T = \left(\mathbf{Q}_1 + \mathbf{u}\mathbf{v}^T\right)\left(\mathbf{Q}_1 + \mathbf{u}\mathbf{v}^T\right)^T \tag{8.12}$$

We first simplify equation (8.11):

$$\begin{aligned}
\mathbf{Q}_2^T\mathbf{Q}_2 &= \left(\mathbf{Q}_1^T + \mathbf{v}\mathbf{u}^T\right)\left(\mathbf{Q}_1 + \mathbf{u}\mathbf{v}^T\right) \\
\mathbf{I} &= \mathbf{I} + \mathbf{v}\left(\mathbf{Q}_1^T\mathbf{u}\right)^T + \left(\mathbf{Q}_1^T\mathbf{u}\right)\mathbf{v}^T + \left(\mathbf{u}^T\mathbf{u}\right)\mathbf{v}\mathbf{v}^T \\
0 &= \mathbf{v}\left(\mathbf{Q}_1^T\mathbf{u}\right)^T + \left(\mathbf{Q}_1^T\mathbf{u}\right)\mathbf{v}^T + \left(\mathbf{u}^T\mathbf{u}\right)\mathbf{v}\mathbf{v}^T \\
-\left(\mathbf{u}^T\mathbf{u}\right)\mathbf{v}\mathbf{v}^T &= \mathbf{v}\left(\mathbf{u}^T\mathbf{Q}_1\right) + \left(\mathbf{Q}_1^T\mathbf{u}\right)\mathbf{v}^T
\end{aligned}$$

Right multiplying both sides by $\mathbf{v}$ and using $\|\mathbf{v}\|_2 = 1$, we get:

$$
\begin{aligned}
-\left(\mathbf{u}^T\mathbf{u}\right)\mathbf{v} &= \left(\mathbf{u}^T\mathbf{Q}_1\mathbf{v}\right)\mathbf{v} + \mathbf{Q}_1^T\mathbf{u} \\
\mathbf{Q}_1^T\mathbf{u} &= -\left(\mathbf{u}^T\mathbf{u} + \mathbf{u}^T\mathbf{Q}_1\mathbf{v}\right)\mathbf{v} = \lambda\mathbf{v} \\
\mathbf{u} &= \lambda\mathbf{Q}_1\mathbf{v}, \quad \text{where } \lambda = -\left(\mathbf{u}^T\mathbf{u} + \mathbf{u}^T\mathbf{Q}_1\mathbf{v}\right)
\end{aligned} \tag{8.13}
$$

Substituting $\mathbf{u}$ using equation (8.13) in equation (8.10), we get:

$$
\begin{aligned}
\mathbf{Q}_2 - \mathbf{Q}_1 &= \lambda\mathbf{Q}_1\mathbf{v}\mathbf{v}^T \\
\mathbf{Q}_2 &= \mathbf{Q}_1\left(\mathbf{I} + \lambda\mathbf{v}\mathbf{v}^T\right)
\end{aligned}
$$

Since $\mathbf{Q}_2^T\mathbf{Q}_2 = \mathbf{I}$, we have:

$$
\begin{aligned}
\mathbf{Q}_2^T\mathbf{Q}_2 &= \left(\mathbf{Q}_1\left(\mathbf{I} + \lambda\mathbf{v}\mathbf{v}^T\right)\right)^T\mathbf{Q}_1\left(\mathbf{I} + \lambda\mathbf{v}\mathbf{v}^T\right) \\
\mathbf{Q}_2^T\mathbf{Q}_2 &= \left(\mathbf{I} + \lambda\mathbf{v}\mathbf{v}^T\right)\mathbf{Q}_1^T\mathbf{Q}_1\left(\mathbf{I} + \lambda\mathbf{v}\mathbf{v}^T\right) \\
\mathbf{I} &= \left(\mathbf{I} + \lambda\mathbf{v}\mathbf{v}^T\right)\left(\mathbf{I} + \lambda\mathbf{v}\mathbf{v}^T\right) \\
\mathbf{I} &= \mathbf{I} + 2\lambda\mathbf{v}\mathbf{v}^T + \lambda^2\mathbf{v}\mathbf{v}^T \\
&\implies \lambda = 0 \text{ or } \lambda = -2
\end{aligned}
$$

Since $\lambda = 0$ would imply $\mathbf{Q}_1 = \mathbf{Q}_2$ which is not allowed by the assumption of the Theorem that $\mathbf{Q}_1 \neq \mathbf{Q}_2$.

$\lambda = -2$ is the only possibility allowed.

This proves the other direction i.e:

$$\mathbf{Q}_2 = \mathbf{Q}_1 \left(\mathbf{I} - 2\mathbf{v}\mathbf{v}^T \right)$$

□

8.A.3 Proof of Corollary 2

Proof. Subsitute $\mathbf{Q}_1, \mathbf{v}$ as follows in Theorem 19.

$$\mathbf{Q}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{v} = \begin{bmatrix} +\sin(\theta/2) \\ -\cos(\theta/2) \end{bmatrix}$$

$$\mathbf{Q}_2 = \mathbf{I} - 2\mathbf{v}\mathbf{v}^T$$

$$\mathbf{Q}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - 2 \begin{bmatrix} \sin(\theta/2) \\ -\cos(\theta/2) \end{bmatrix} \begin{bmatrix} \sin(\theta/2) & -\cos(\theta/2) \end{bmatrix}$$

$$\mathbf{Q}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - 2 \begin{bmatrix} \sin^2(\theta/2) & -\sin(\theta/2)\cos(\theta/2) \\ -\sin(\theta/2)\cos(\theta/2) & \cos^2(\theta/2) \end{bmatrix}$$

$$\mathbf{Q}_2 = \begin{bmatrix} 1 - 2\sin^2(\theta/2) & 2\sin(\theta/2)\cos(\theta/2) \\ 2\sin(\theta/2)\cos(\theta/2) & 1 + 2\cos^2(\theta/2) \end{bmatrix}$$

$$\mathbf{Q}_2 = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(\theta) & -\cos(\theta) \end{bmatrix}$$

Theorem 19 then directly implies the corollary. □

8.A.4 Proof of Theorem 20

Theorem 20. *Given:* $0 \leq \theta_0 < \theta_1 \cdots < \theta_{2n} = 2\pi + \theta_0$ *such that* $\sum_{i=0}^{n-1}(\theta_{2i+1} - \theta_{2i}) = \pi$ *and* $\alpha_i = 2\sum_{j=0}^{i}\theta_{i-j}(-1)^j$*, The function* $\sigma_\Theta : \mathbb{R}^2 \to \mathbb{R}^2$ *is continuous, GNP and* 1*-Lipschitz where* $\Theta = [\theta_0, \theta_1, \ldots, \theta_{2n-1}]$ *(also called* 2D *Householder Activation of order* n*):*

$$\sigma_\Theta(z_1, z_2) = \begin{bmatrix} \cos\alpha_i & \sin\alpha_i \\ (-1)^i \sin\alpha_i & (-1)^{i+1}\cos\alpha_i \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \qquad \theta_i \leq \varphi < \theta_{i+1} \tag{8.14}$$

where $\varphi \in [\theta_0,\ \theta_{2n} = 2\pi + \theta_0)$ *and* $\cos(\varphi) = z_1/\sqrt{z_1^2 + z_2^2},\ \ \sin(\varphi) = z_2/\sqrt{z_1^2 + z_2^2}$.

Proof. We are given the following:

$$\sum_{i=0}^{n-1}(\theta_{2i+1} - \theta_{2i}) = \pi, \qquad \alpha_i = 2\sum_{j=0}^{i}\theta_{i-j}(-1)^j \tag{8.15}$$

Note that by definition (equation (8.14)), the function is linear for $\theta_i < \varphi < \theta_{i+1}$ and hence continuous.

Furthermore, since $\varphi \in [\theta_0,\ \theta_{2n})$, we proceed to prove continuity for the following two cases:

Case 1: $\theta_i - \epsilon < \varphi < \theta_i + \epsilon, \qquad \epsilon > 0,\ i \geq 1$

Case 2: $\theta_0 < \varphi < \theta_0 + \epsilon, \qquad \theta_{2n} - \epsilon < \varphi < \theta_{2n}, \quad \epsilon > 0$

Proof for Case 1:

Using equation (8.14), we know that σ_Θ realizes different linear functions for $\theta_i - \epsilon < \varphi < \theta_i$

and $\theta_i < \varphi < \theta_i + \epsilon$.

Thus, for σ_Θ to be continuous, we require that the two linear functions be the same at the boundary i.e $\varphi = \theta_i$.

We first write the input $(z_1,\ z_2)$ in terms of shifted polar coordinates i.e: $(r\cos(\varphi),\ r\sin(\varphi))$ where $r = \sqrt{z_1^2 + z_2^2}$ and $\cos\varphi = z_1/\sqrt{z_1^2 + z_2^2},\ \sin\varphi = z_2/\sqrt{z_1^2 + z_2^2}, \varphi \in [\theta_0, \theta_0 + 2\pi)$

Thus, the function for $\theta_i - \epsilon < \varphi < \theta_i$ is given by:

$$\sigma_\Theta\left(r\cos\varphi, r\sin\varphi\right) = \begin{bmatrix} \cos\alpha_{i-1} & \sin\alpha_{i-1} \\ (-1)^{i-1}\sin\alpha_{i-1} & (-1)^i\cos\alpha_{i-1} \end{bmatrix} \begin{bmatrix} r\cos\varphi \\ r\sin\varphi \end{bmatrix} \tag{8.16}$$

Similarly, the function for $\theta_i < \varphi < \theta_i + \epsilon$ is given by:

$$\sigma_\Theta\left(r\cos\varphi, r\sin\varphi\right) = \begin{bmatrix} \cos\alpha_{i} & \sin\alpha_{i} \\ (-1)^{i}\sin\alpha_{i} & (-1)^{i+1}\cos\alpha_{i} \end{bmatrix} \begin{bmatrix} r\cos\varphi \\ r\sin\varphi \end{bmatrix} \tag{8.17}$$

The difference in the function values at the boundary i.e $\varphi = \theta_i$, obtained by subtracting equations (8.17) and (8.16) is given as follows:

$$\begin{bmatrix} \cos\alpha_{i} & \sin\alpha_{i} \\ (-1)^{i}\sin\alpha_{i} & (-1)^{i+1}\cos\alpha_{i} \end{bmatrix} \begin{bmatrix} r\cos\theta_i \\ r\sin\theta_i \end{bmatrix} - \begin{bmatrix} \cos\alpha_{i-1} & \sin\alpha_{i-1} \\ (-1)^{i-1}\sin\alpha_{i-1} & (-1)^{i}\cos\alpha_{i-1} \end{bmatrix} \begin{bmatrix} r\cos\theta_i \\ r\sin\theta_i \end{bmatrix}$$

$$= r \begin{bmatrix} \cos\alpha_i - \cos\alpha_{i-1} & \sin\alpha_i - \sin\alpha_{i-1} \\ (-1)^i\sin\alpha_i - (-1)^{i-1}\sin\alpha_{i-1} & (-1)^{i+1}\cos\alpha_i - (-1)^i\cos\alpha_{i-1} \end{bmatrix} \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix}$$

$$= r \begin{bmatrix} \cos\alpha_i - \cos\alpha_{i-1} & \sin\alpha_i - \sin\alpha_{i-1} \\ (-1)^i(\sin\alpha_i + \sin\alpha_{i-1}) & (-1)^{i+1}(\cos\alpha_i + \cos\alpha_{i-1}) \end{bmatrix} \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix}$$

Using sum-to-product trigonometric identities, the above equals:

$$= r \begin{bmatrix} 2\sin\left(\frac{\alpha_{i-1}-\alpha_i}{2}\right)\sin\left(\frac{\alpha_{i-1}+\alpha_i}{2}\right) & 2\sin\left(\frac{\alpha_i-\alpha_{i-1}}{2}\right)\cos\left(\frac{\alpha_i+\alpha_{i-1}}{2}\right) \\ 2(-1)^i\sin\left(\frac{\alpha_i+\alpha_{i-1}}{2}\right)\cos\left(\frac{\alpha_i-\alpha_{i-1}}{2}\right) & 2(-1)^{i+1}\cos\left(\frac{\alpha_{i-1}-\alpha_i}{2}\right)\cos\left(\frac{\alpha_{i-1}+\alpha_i}{2}\right) \end{bmatrix} \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix}$$

$$= 2r \begin{bmatrix} \sin\left(\frac{\alpha_{i-1}-\alpha_i}{2}\right)\sin\left(\frac{\alpha_{i-1}+\alpha_i}{2}\right) & -\sin\left(\frac{\alpha_{i-1}-\alpha_i}{2}\right)\cos\left(\frac{\alpha_i+\alpha_{i-1}}{2}\right) \\ (-1)^i\sin\left(\frac{\alpha_i+\alpha_{i-1}}{2}\right)\cos\left(\frac{\alpha_i-\alpha_{i-1}}{2}\right) & (-1)^{i+1}\cos\left(\frac{\alpha_{i-1}-\alpha_i}{2}\right)\cos\left(\frac{\alpha_{i-1}+\alpha_i}{2}\right) \end{bmatrix} \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix}$$

$$= 2r \begin{bmatrix} \sin\left(\frac{\alpha_{i-1}-\alpha_i}{2}\right) \\ (-1)^i\cos\left(\frac{\alpha_{i-1}-\alpha_i}{2}\right) \end{bmatrix} \begin{bmatrix} \sin\left(\frac{\alpha_{i-1}+\alpha_i}{2}\right) & -\cos\left(\frac{\alpha_{i-1}+\alpha_i}{2}\right) \end{bmatrix} \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix}$$

Using equation (8.15), we directly have: $\alpha_i = 2\theta_i - \alpha_{i-1}$. Thus, the above equation reduces to:

$$= 2r \begin{bmatrix} \sin\left(\theta_i - \alpha_i\right) \\ (-1)^i\cos\left(\theta_i - \alpha_i\right) \end{bmatrix} \begin{bmatrix} \sin\left(\theta_i\right) & -\cos\left(\theta_i\right) \end{bmatrix} \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix} = \begin{bmatrix} 0. \\ 0. \end{bmatrix}$$

Hence, the linear functions given by equations (8.16) and (8.17) are equal at $\varphi = \theta_i$. This proves that the function is continuous for Case 1.

Proof for Case 2:

Using equation (8.14), we know that σ_Θ realizes different linear functions for $\theta_0 < \varphi < \theta_0 + \epsilon$ and $\theta_{2n} - \epsilon < \varphi < \theta_{2n}$.

Thus, for σ_Θ to be continuous, we require that the two linear functions be the same at the boundary i.e $\varphi = \theta_0$.

As before, we first write the input $(z_1,\ z_2)$ in terms of shifted polar coordinates i.e: $(r\cos(\varphi),\ r\sin(\varphi))$.

Thus, the function for $\theta_0 < \varphi < \theta_0 + \epsilon$ is given by:

$$\sigma_\Theta\left(r\cos\varphi, r\sin\varphi\right) = \begin{bmatrix} \cos\alpha_0 & \sin\alpha_0 \\ \sin\alpha_0 & -\cos\alpha_0 \end{bmatrix} \begin{bmatrix} r\cos\varphi \\ r\sin\varphi \end{bmatrix} \tag{8.18}$$

Similarly, the function for $\theta_{2n} - \epsilon < \phi < \theta_{2n}$ is given by:

$$\sigma_\Theta\left(r\cos\varphi, r\sin\varphi\right) = \begin{bmatrix} \cos\alpha_{2n-1} & \sin\alpha_{2n-1} \\ (-1)^i\sin\alpha_{2n-1} & (-1)^{i+1}\cos\alpha_{2n-1} \end{bmatrix} \begin{bmatrix} r\cos\varphi \\ r\sin\varphi \end{bmatrix} \tag{8.19}$$

Using equation (8.15), α_{2n-1} is given as follows:

$$\alpha_{2n-1} = 2\sum_{i=0}^{2n-1}\theta_{2n-1-i}(-1)^i = 2\sum_{i=0}^{n-1}(\theta_{2i+1} - \theta_{2i}) = 2\pi$$

Thus, equation (8.19) reduces to:

$$\sigma_\Theta\left(r\cos\varphi, r\sin\varphi\right) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} r\cos\varphi \\ r\sin\varphi \end{bmatrix} \tag{8.20}$$

The difference in the function values at the boundary i.e $\varphi = \theta_0$, obtained by subtracting

equations (8.20) and (8.18) is given as follows:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} r\cos\theta_0 \\ r\sin\theta_0 \end{bmatrix} - \begin{bmatrix} \cos\alpha_0 & \sin\alpha_0 \\ \sin\alpha_0 & -\cos\alpha_0 \end{bmatrix} \begin{bmatrix} r\cos\theta_0 \\ r\sin\theta_0 \end{bmatrix}$$
$$= r \begin{bmatrix} 1-\cos\alpha_0 & -\sin\alpha_0 \\ -\sin\alpha_0 & 1+\cos\alpha_0 \end{bmatrix} \begin{bmatrix} \cos\theta_0 \\ \sin\theta_0 \end{bmatrix}$$

Using the trigonometric identities: $1-\cos(\theta) = 2\sin^2(\theta/2)$, $1+\cos(\theta) = 2\cos^2(\theta/2)$ and $\sin(\theta) = 2\sin(\theta/2)\cos(\theta/2)$, we have:

$$= r \begin{bmatrix} 2\sin^2(\frac{\alpha_0}{2}) & -2\sin(\frac{\alpha_0}{2})\cos(\frac{\alpha_0}{2}) \\ -2\sin(\frac{\alpha_0}{2})\cos(\frac{\alpha_0}{2}) & 2\cos^2(\frac{\alpha_0}{2}) \end{bmatrix} \begin{bmatrix} \cos\theta_0 \\ \sin\theta_0 \end{bmatrix}$$
$$= 2r \begin{bmatrix} \sin(\frac{\alpha_0}{2}) \\ -\cos(\frac{\alpha_0}{2}) \end{bmatrix} \begin{bmatrix} \sin(\frac{\alpha_0}{2}) & -\cos(\frac{\alpha_0}{2}) \end{bmatrix} \begin{bmatrix} \cos\theta_0 \\ \sin\theta_0 \end{bmatrix}$$

Using equation (8.15), we have: $\alpha_0 = 2\theta_0$. Thus, the above equation reduces to:

$$= 2r \begin{bmatrix} \sin(\theta_0) \\ -\cos(\theta_0) \end{bmatrix} \begin{bmatrix} \sin(\theta_0) & -\cos(\theta_0) \end{bmatrix} \begin{bmatrix} \cos\theta_0 \\ \sin\theta_0 \end{bmatrix} = \begin{bmatrix} 0. \\ 0. \end{bmatrix}$$

Hence, the linear functions given by equations (8.18) and (8.20) are equal at $\varphi = \theta_0$.

This proves that the function is continuous for Case 2. □

Architecture	γ	Clean Acc.	Provable Robust Acc. ($\rho =$) 36/255	72/255	108/255	Increase (108/255)
LipConvnet-5	0.	75.82%	59.66%	42.78%	26.92%	_
	0.10	75.58%	59.74%	42.94%	28.04%	+0.94%
	0.25	75.54%	60.22%	43.98%	29.50%	+2.58%
	0.50	75.30%	**60.08%**	**45.36%**	**31.88%**	**+4.96%**
	0.75	74.14%	60.36%	46.08%	33.44%	+6.52%
	1.00	73.86%	60.30%	46.80%	34.60%	+7.68%

Table 8.4: Results for provable robustness against adversarial examples on the CIFAR-10 dataset for cross validation using 5000 held out samples from the training set.

8.B Selection of γ using cross validation

Using 5000 held out samples from CIFAR-10, we tested 6 different values of γ shown in Table 8.4 and selected $\gamma = 0.5$ because it resulted in less than 0.5% decrease in standard accuracy while 4.96% increase in provably robust accuracy. We used the LipConvnet-5 network with the $2D$ Householder activation function of order 2 i.e σ_θ.

8.C Differences between l_1 and l_2 certificate

By the equivalence of norms, we have the following result:

$$\mathbf{x} \in \mathbb{R}^d, \qquad \frac{1}{\sqrt{d}}\|\mathbf{x}\|_1 \leq \|\mathbf{x}\|_2 \leq \sqrt{d}\|\mathbf{x}\|_1 \tag{8.21}$$

Let us assume that we have an l_1 certificate for the input $\mathbf{x}$ so that the prediction of a neural network remains constant in a region of l_1 radius ρ_1 around the input $\mathbf{x}$ i.e:

$$\|\mathbf{x}^* - \mathbf{x}\|_1 \leq \rho_1 \tag{8.22}$$

We want to compute the l_2 certificate implied by the certificate given in equation (8.22). Let ρ_2 be the l_2 certificate so that:

$$\|\mathbf{x}^* - \mathbf{x}\|_2 \leq \rho_2 \implies \|\mathbf{x}^* - \mathbf{x}\|_1 \leq \rho_1$$

Using equation (8.21), we have the following set of equations:

$$\|\mathbf{x}^* - \mathbf{x}\|_2 \leq \rho_2 \implies \frac{1}{\sqrt{d}}\|\mathbf{x}^* - \mathbf{x}\|_1 \leq \rho_2 \implies \|\mathbf{x}^* - \mathbf{x}\|_1 \leq \sqrt{d}\rho_2$$

Using equation (8.22), we have the following:

$$\sqrt{d}\rho_2 \leq \rho_1 \implies \rho_2 \leq \frac{\rho_1}{\sqrt{d}}$$

Hence, the l_2 norm certificate induced by the l_1 norm certificate can be significantly smaller for high dimensional inputs. Using the l_1 certificate used in [249] i.e $\rho_1 = 4$ and $d = 32\sqrt{3}$ for CIFAR-10 and CIFAR-100, we get an implied certificate of $\rho_2 = 4/(32\sqrt{3}) = 0.07225$. In contrast, in this chapter we study l_2 robustness for much higher certificate values i.e $36/255 = 0.14118$.

8.D Verification that $\sigma_\theta(z_1, z_2)$ always lies on one side of the hyperplane

Consider the case: $z_1 \sin(\theta/2) - z_2 \cos(\theta/2) > 0$

In this case $\sigma_\theta(z_1, z_2) = (z_1, z_2)$ and the result follows directly.

Consider the other case: $z_1 \sin(\theta/2) - z_2 \cos(\theta/2) \leq 0$

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} z_1\cos\theta + z_2\sin\theta \\ z_1\sin\theta - z_2\cos\theta \end{bmatrix}$$

$$a_1 \sin(\theta/2) - a_2 \cos(\theta/2) = (z_1\cos\theta + z_2\sin\theta)\sin(\theta/2) - (z_1\sin\theta - z_2\cos\theta)\cos(\theta/2)$$

$$= z_1(\cos\theta\sin(\theta/2) - \sin\theta\cos(\theta/2)) + z_2(\sin\theta\sin(\theta/2) + \cos\theta\cos(\theta/2))$$

$$= -z_1\sin(\theta/2) + z_2\cos(\theta/2)$$

Since $z_1 \sin(\theta/2) - z_2 \cos(\theta/2) \leq 0$, we have $-z_1 \sin(\theta/2) + z_2 \cos(\theta/2) \geq 0$.

8.E Higher order Householder activation functions

We know that $\mathrm{MaxMin}(z_1, z_2) = (\max(z_1, z_2),\ \min(z_1, z_2))$ where $z_1, z_2 \in \mathbb{R}$. Because $\max(z_1, z_2) > \min(z_1, z_2)$, applying MaxMin again gives the same result i.e $\mathrm{MaxMin} \circ \mathrm{MaxMin} = \mathrm{MaxMin}$. Now consider the function σ_θ (discussed in the maintext, given below

for convenience):

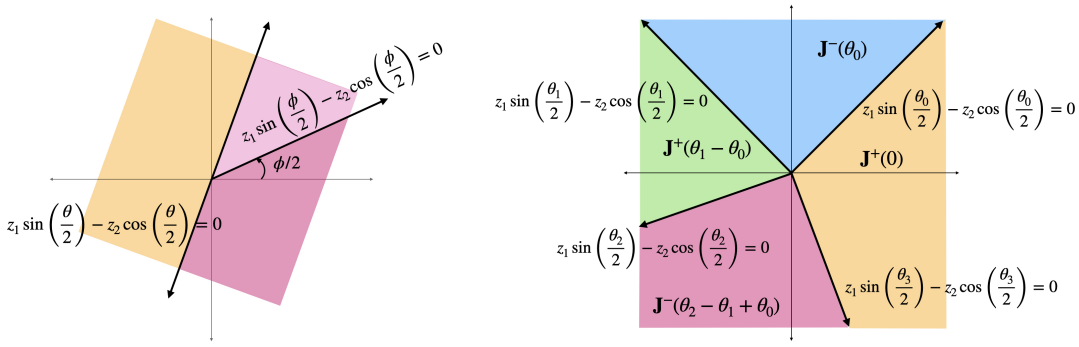
$$\sigma_\theta\left(z_1, z_2\right) = \begin{cases} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} & \text{if } \; z_1 \sin\left(\theta/2\right) - z_2 \cos\left(\theta/2\right) > 0 \\ \begin{bmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} & \text{if } \; z_1 \sin\left(\theta/2\right) - z_2 \cos\left(\theta/2\right) \leq 0 \end{cases}$$

From Figure 8.1, we observe that if $(u_1, u_2) = \sigma_\theta(z_1, z_2)$, then (u_1, u_2) always lies on the right side of the hyperplane (pink colored region). In other words, $u_1 \sin\left(\theta/2\right) - u_2 \cos\left(\theta/2\right) > 0 \;\; \forall \; z_1, z_2$ (proof in Supp. 8.D). This further implies that $\sigma_\theta \circ \sigma_\theta = \sigma_\theta$.

However from Figure 8.2a in the maintext, we observe that if we use a different angle ϕ where $\phi \neq \theta + 2n\pi$ for some $n \in \mathbb{Z}$, then $\sigma_\phi(u_1, u_2) \neq (u_1, u_2)$ for all (u_1, u_2) in the pink colored region ($u_1 \sin\left(\theta/2\right) - u_2 \cos\left(\theta/2\right) > 0$). This motivates us to construct the function $\sigma^{(n)} : \mathbb{R}^2 \to \mathbb{R}^2$ defined as follows:

$$\sigma^{(n)} = \underbrace{\sigma_\theta \circ \sigma_\theta \circ \sigma_\theta \ldots \circ \sigma_\theta}_{n \text{ times, } \theta\text{'s can be different}} \tag{8.23}$$

Clearly, $\sigma^{(n)}$ has a larger number of linear regions than σ_θ and thus expected to have more expressive power. However, a drawback of using $\sigma^{(n)}$ is that it requires a sequential application of σ_θ which can be expensive if the number of iterations n is large. To address this limitation, first observe that σ_θ realizes the same linear function for (z_1, z_2) and (cz_1, cz_2) when $c > 0$ i.e $\nabla_{(z_1, z_2)} \; \sigma_\theta = \nabla_{(cz_1, cz_2)} \; \sigma_\theta$. Since σ_θ is piecewise linear, $\sigma_\theta(cz_1, cz_2) = c\sigma_\theta(z_1, z_2)$. Thus, the input of the next function in the iteration is scaled by c as well and its linear function (or the



(a) The output of σ_θ always lies in the pink region. Applying σ_ϕ on this where $\phi \neq \theta + 2n\pi, n \in \mathbb{Z}$ further divides this region into two linear regions (light and original pink).

(b) In each colored region, the function is linear with the Jacobian mentioned. For the function to be continuous, we must have $\theta_3 - \theta_2 + \theta_1 - \theta_0 = \pi$. Thus, any 3 of $\{\theta_0, \theta_1, \theta_2, \theta_3\}$ can be chosen as learnable parameters.

Figure 8.2: Constructing Higher Order Householder activations ($\mathbf{J}^+$ and $\mathbf{J}^-$ defined in equation (8.2))

Jacobian) remains unchanged. By induction, same holds for all the subsequent iterations. By chain rule, the Jacobian of composition of functions is equal to the product of Jacobian of each individual function. Since the Jacobian of each function is unchanged on scaling by $c > 0$, the Jacobian $\nabla_{(z_1,z_2)}\, \sigma^{(n)}$ also remains unchanged: $\nabla_{(z_1,z_2)}\, \sigma^{(n)} = \nabla_{(cz_1,cz_2)}\, \sigma^{(n)}$. This suggests that it is possible to determine the Jacobian $\nabla_{(z_1,z_2)}\, \sigma^{(n)}$ for the input (z_1, z_2) by first converting to the polar coordinates $(\sqrt{z_1^2 + z_2^2},\ \varphi)$ and then using the phase angle φ alone (where $\cos(\varphi) = z_1/\sqrt{z_1^2 + z_2^2},\ \sin(\varphi) = z_2/\sqrt{z_1^2 + z_2^2}$).

This motivates us to construct another GNP piecewise linear activation that only depends on the phase of the input but unlike σ_θ, it is allowed to have more than 2 linear regions without requiring a sequential application. This construction is given in the following theorem (example in Figure 8.2b):

Theorem 8.E.1. *Given:* $0 \leq \theta_0 < \theta_1 \cdots < \theta_{2n} = 2\pi + \theta_0$ *such that* $\sum_{i=0}^{n-1}(\theta_{2i+1} - \theta_{2i}) = \pi$ *and* $\alpha_i = 2\sum_{j=0}^{i} \theta_{i-j}(-1)^j$, *The function* $\sigma_\Theta : \mathbb{R}^2 \to \mathbb{R}^2$ *is continuous, GNP and* 1*-Lipschitz where*

$\Theta = [\theta_0, \theta_1, \ldots, \theta_{2n-1}]$ *(also called Householder Activation of order* n *in* 2 *dimensions):*

$$\sigma_\Theta(z_1, z_2) = \begin{bmatrix} \cos\alpha_i & \sin\alpha_i \\ (-1)^i \sin\alpha_i & (-1)^{i+1}\cos\alpha_i \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \qquad \theta_i \leq \varphi < \theta_{i+1}$$

where $\varphi \in [\theta_0,\ \theta_{2n} = 2\pi + \theta_0)$ *and* $\cos(\varphi) = z_1/\sqrt{z_1^2 + z_2^2},\ \ \sin(\varphi) = z_2/\sqrt{z_1^2 + z_2^2}$.

Using the definition of α_i, α_{2n-1} can be computed as follows:

$$\alpha_{2n-1} = 2\sum_{j=0}^{2n-1} \theta_{2n-1-j}(-1)^j = 2\sum_{j=0}^{n-1} (\theta_{2n-1-2j} - \theta_{2n-2-2j}) = 2\sum_{j=0}^{n-1} (\theta_{2j+1} - \theta_{2j}) = 2\pi$$

$$\sigma_\Theta(z_1, z_2) = \begin{bmatrix} \cos\alpha_{2n-1} & \sin\alpha_{2n-1} \\ -\sin\alpha_{2n-1} & \cos\alpha_{2n-1} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = (z_1, z_2) \qquad \theta_{2n-1} \leq \varphi < \theta_{2n} (= \theta_0 + 2\pi)$$

By continuity, $\sigma_\Theta(z_1, z_2) = (z_1, z_2)$ for $\varphi = \theta_0$. Thus if we set $\theta_0 = 0$, σ_Θ is fixed to be identity when $\varphi = 0$ (or $z_2 = 0$). However, a learnable θ_0 offers the flexibility of choosing arbitrary intervals around the $\varphi = \theta_0$ to be the identity function (while of course allowing $\theta_0 = 0$). Since we can choose any interval of 2π for the phase angle, we choose $\varphi \in [\theta_0,\ \theta_{2n} = \theta_0 + 2\pi)$ instead of the usual $[0,\ 2\pi)$ to allow this possibility. We call $(\sqrt{z_1^2 + z_2^2},\ \varphi)$, the *shifted polar coordinates*.

Additionally, we make the following observations about Theorem 20. First, by construction σ_Θ has $2n$ linear regions. Second, since $\sum_{i=0}^{n-1}(\theta_{2i+1} - \theta_{2i}) = \pi$, the sum of angles subtended by linear regions with determinant -1 equals π. This in turn implies that sum of angles subtended by linear regions with determinant $+1$ must also equal π. Third, again using $\sum_{j=0}^{n-1}(\theta_{2j+1} - \theta_{2j}) = \pi$, we know that only $2n - 1$ of the $2n$ parameters in $[\theta_0,\ \theta_1,\ \theta_2,\ \ldots,\ \theta_{2n-1}]$ can be chosen

independently implying that σ_Θ has $2n-1$ learnable parameters. In contrast, $\sigma^{(n)}$ has only n learnable parameters. Fourth, when $n=1$, σ_Θ reduces to the original σ_θ activation.

Because every function of the form $\sigma^{(n)}$ (equation (8.23)) can have potentially 2^n linear regions, while σ_Θ has only $2n$ linear regions, σ_Θ cannot express every function of the form $\sigma^{(n)}$. The primary benefit of using σ_Θ is that it can be easily applied by first determining the angle φ (using shifted polar coordinates), the region $[\theta_i, \theta_{i+1})$ to which φ belongs and the Jacobian for this region. This requires 1 multiplication with the Jacobian instead of n required for $\sigma^{(n)}$.

8.F Extension to higher dimensions

We introduced Householder activation function of Order 1 in m dimensions in maintext Definition 9. However, it suffers from the limitation that it has only 2 linear regions thus limiting its expressive power. The construction given in Supp. Section 8.E allows more than 2 linear regions but is valid only for 2 dimensional inputs. This motivates us to construct Householder activations that depend on all the m components of input $\mathbf{z} \in \mathbb{R}^m,\ (m \geq 3)$ while allowing for more than 2 linear regions.

A straightforward way of constructing such an activation function is to apply an orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{m\times m}$, followed by dividing the output $\mathbf{Qz}$ into groups of size 2 each and then applying σ_θ to each group. However, since 1-Lipschitz neural networks involve multiplication with an orthogonal weight matrix followed by GNP activation anyway, this construction is trivial because it does not lead to additional gains in expressive power over using 2 dimensional σ_Θ activation functions.

Recall that the function $\sigma_{\mathbf{v}}$ is given by the following equation:

$$\sigma_{\mathbf{v}}(\mathbf{z}) = \begin{cases} \mathbf{z}, & \mathbf{v}^T\mathbf{z} > 0, \\ (\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)\mathbf{z}, & \mathbf{v}^T\mathbf{z} \leq 0. \end{cases} \tag{8.24}$$

By a similar analysis as for the 2-dimensional case (Figure 8.2 in maintext), a repeated application of $\sigma_{\mathbf{v}}$ leads to increased number of linear regions and thus higher expressive power. This motivates us to construct the function $\sigma^{(m,n)} : \mathbb{R}^m \to \mathbb{R}^m$ by applying the function $\sigma_{\mathbf{v}}$ (equation (8.24)) n times iteratively with different learnable parameter $\mathbf{v}$ at each iteration:

$$\sigma^{(m,n)} = \underbrace{\sigma_{\mathbf{v}} \circ \sigma_{\mathbf{v}} \circ \sigma_{\mathbf{v}} \ldots \circ \sigma_{\mathbf{v}}}_{n \text{ times, } \mathbf{v}\text{'s can be different}}$$

Since $\sigma_{\mathbf{v}}$ realizes the same linear function for both the inputs $\mathbf{z}$ and $c\mathbf{z}$ i.e $\nabla_{\mathbf{z}}\, \sigma_{\mathbf{v}} = \nabla_{c\mathbf{z}}\, \sigma_{\mathbf{v}}$ when $c > 0$, $\sigma^{(m,n)}$ satisfies this property as well. This suggests that it is possible to determine the Jacobian of $\sigma^{(m,n)}$ for the given input $\mathbf{z}$ by projecting $\mathbf{z}$ onto a unit sphere: $\mathbf{z}/\|\mathbf{z}\|_2$. Moreover, we want our constructed function to have at least $2n$ linear regions while requiring k iterations of $\sigma_{\mathbf{v}}$ where k is independent of n. This motivates the following open question:

Open Problem. *Can non-trivial order-n ($n > 1$) householder activation functions with $2n$ linear regions be constructed for m dimensional input ($m > 2$) using k iterations of $\sigma_{\mathbf{v}}$ where k is independent of n (but may depend on m)?*

8.F.1 Additional experiments

8.F.2 Additional experiments on CIFAR-10

The rows "BCOP", "Cayley" and "SOC (Baseline)" all use the MaxMin activation function. "HH" replaces MaxMin with $2\mathrm{D}$ Householder activation of order 1 (σ_θ), "+ CR" adds Certificate Regularization (CR) with $\gamma = 0.1$ (while using σ_θ as the activation function).

In Table 8.7, the row "SOC + $\mathrm{HH}^{(2)}$" uses Householder activation of order 2 (σ_Θ defined in Theorem 20), "+ CR" adds Certificate Regularization (CR) with $\gamma = 0.1$ (while using the HH activation of order 2 i.e σ_Θ as the activation function).

None of the results in Tables 8.5, 8.6 and 8.7 include Last Layer Normalization (LLN).

In Table 8.8, we evaluate the effect of using LLN. Due to the small number of classes in CIFAR-10, we do not observe significant gains.

Lip Conv net-	**Methods**	**Clean Acc.**	**Provable Robust Acc.** ($\rho =$) 36/255	72/255	108/255	**Increase** (108/255)
5	BCOP	74.25%	58.01%	40.34%	25.21%	-1.88%
	Cayley	72.37%	55.92%	38.65%	24.27%	-2.82%
	SOC (Baseline)	75.78%	59.18%	42.01%	27.09%	(+0%)
	+ HH	**76.30%**	60.12%	43.20%	27.39%	+0.30%
	+ CR	75.31%	**60.37%**	**45.62%**	**32.38%**	**+5.29%**
10	BCOP	74.47%	58.48%	40.77%	26.16%	-2.99%
	Cayley	74.30%	57.99%	40.75%	25.93%	-3.22%
	SOC (Baseline)	76.45%	60.86%	44.15%	29.15%	(+0%)
	+ HH	**76.86%**	61.52%	44.91%	29.90%	+0.75%
	+ CR	76.23%	**62.57%**	**47.70%**	**34.15%**	**+5.00%**
15	BCOP	73.86%	57.39%	39.33%	24.86%	-4.80%
	Cayley	71.92%	54.55%	37.67%	23.50%	-6.16%
	SOC (Baseline)	76.68%	61.36%	44.28%	29.66%	(+0%)
	+ HH	**77.41%**	61.92%	45.60%	31.10%	+1.44%
	+ CR	76.39%	**62.96%**	**48.47%**	**35.47%**	**+5.81%**
20	BCOP	69.84%	52.10%	34.75%	21.09%	-9.99%
	Cayley	68.87%	51.88%	35.56%	21.72%	-9.36%
	SOC (Baseline)	**76.90%**	61.87%	45.79%	31.08%	(+0%)
	+ HH	76.99%	61.76%	45.59%	30.99%	-0.09%
	+ CR	76.34%	**62.63%**	**48.69%**	**36.04%**	**+4.96%**

Table 8.5: Results for provable robustness on the CIFAR-10 dataset using shallow networks. None of these results include Last Layer Normalization (LLN).

Lip Conv net-	**Methods**	**Clean Acc.**	**Provable Robust Acc.** ($\rho =$) 36/255	72/255	108/255	**Increase** (108/255)
25	BCOP	68.47%	49.92%	31.99%	18.62%	-9.98%
	Cayley	64.00%	45.55%	29.24%	16.99%	-11.61%
	SOC (Baseline)	75.24%	60.17%	43.55%	28.60%	(+0%)
	+ HH	**76.37%**	61.50%	44.72%	29.83%	+1.23%
	+ CR	75.21%	**61.98%**	**47.93%**	**34.92%**	**+6.32%**
30	BCOP	64.11%	43.39%	25.02%	12.15%	-15.90%
	Cayley	58.83%	38.68%	22.07%	10.68%	-17.37%
	SOC (Baseline)	74.51%	59.06%	42.46%	28.05%	(+0%)
	+ HH	**75.25%**	59.90%	43.85%	29.35%	+1.30%
	+ CR	74.23%	**60.64%**	**46.51%**	**34.08%**	**+6.03%**
35	BCOP	63.05%	41.71%	23.30%	11.36%	-15.84%
	Cayley	53.55%	32.37%	16.18%	6.33%	-20.87%
	SOC (Baseline)	73.73%	58.50%	41.75%	27.20%	(+0%)
	+ HH	**75.44%**	61.05%	45.38%	30.85%	+3.65%
	+ CR	74.25%	**61.30%**	**47.60%**	**35.21%**	**+8.01%**
40	BCOP	60.17%	38.86%	21.20%	9.08%	-15.05%
	Cayley	51.26%	27.90%	12.06%	3.81%	-20.32%
	SOC (Baseline)	71.63%	54.39%	37.92%	24.13%	(+0%)
	+ HH	**73.24%**	58.12%	42.24%	28.48%	+4.35%
	+ CR	72.59%	**59.04%**	**44.92%**	**32.87%**	**+8.74%**

Table 8.6: Results for provable robustness against adversarial examples on the CIFAR-10 dataset. None of these results include Last Layer Normalization (LLN).

Lip Conv net-	Methods	Clean Acc.	Provable Robust Acc. ($\rho =$)			Increase
			36/255	72/255	108/255	(108/255)
5	**SOC + HH$^{(2)}$**	75.85%	59.66%	42.68%	27.44%	+0.35%
	+ CR	74.85%	60.56%	44.96%	31.98%	+4.59%
10	**SOC + HH$^{(2)}$**	76.80%	61.44%	44.91%	29.70%	+0.55%
	+ CR	76.30%	62.11%	47.85%	34.49%	+5.34%
15	**SOC + HH$^{(2)}$**	77.41%	62.21%	45.11%	30.49%	+0.83%
	+ CR	75.73%	62.21%	47.92%	35.26%	+5.60%
20	**SOC + HH$^{(2)}$**	76.69%	61.58%	45.39%	30.89%	-0.19%
	+ CR	75.72%	62.61%	48.30%	35.29%	+4.21%
25	**SOC + HH$^{(2)}$**	76.12%	61.24%	44.81%	29.63%	+1.03%
	+ CR	75.38%	61.94%	47.67%	34.22%	+5.62%
30	**SOC + HH$^{(2)}$**	75.09%	60.01%	44.22%	29.39%	+1.34%
	+ CR	74.88%	61.23%	46.63%	34.02%	+5.97%
35	**SOC + HH$^{(2)}$**	73.93%	58.61%	42.29%	28.47%	+1.27%
	+ CR	74.14%	60.72%	46.67%	34.64%	+7.44%
40	**SOC + HH$^{(2)}$**	70.90%	54.96%	38.94%	24.90%	+0.77%
	+ CR	72.28%	57.67%	43.00%	30.66%	+6.53%

Table 8.7: Results for provable robustness on CIFAR-10 using HH activation of Order 2 (σ_Θ). Increase (108/255) is calculated with respect to SOC Baseline (from Tables 8.5, 8.6). None of these results include Last Layer Normalization (LLN).

Lip Conv net-	**Methods**	**Clean Acc.**	**Provable Robust Acc.** ($\rho =$) (36/255)	(72/255)	(108/255)	**Increase** (Clean)
5	SOC (no LLN)	75.78%	59.18%	42.01%	27.09%	(+0%)
	SOC + LLN	**75.78%**	**59.58%**	**42.45%**	**27.20%**	**+0.00%**
10	SOC (no LLN)	76.45%	60.86%	44.15%	29.15%	(+0%)
	SOC + LLN	**76.69%**	**61.08%**	**44.04%**	**29.19%**	**+0.24%**
15	SOC (no LLN)	76.68%	61.36%	44.28%	29.66%	(+0%)
	SOC + LLN	**76.84%**	**61.94%**	**45.51%**	**30.28%**	**+0.16%**
20	SOC (no LLN)	**77.05%**	**61.87%**	**45.79%**	**31.08%**	**(+0%)**
	SOC + LLN	76.71%	61.44%	44.92%	30.19%	-0.34%
25	SOC (no LLN)	75.24%	60.17%	43.55%	28.60%	(+0%)
	SOC + LLN	**76.54%**	**61.21%**	**44.18%**	**29.47%**	**+1.30%**
30	SOC (no LLN)	**74.51%**	**59.06%**	**42.46%**	**28.05%**	**(+0%)**
	SOC + LLN	74.26%	58.97%	41.82%	26.93%	-0.25%
35	SOC (no LLN)	73.73%	58.50%	41.75%	27.20%	(+0%)
	SOC + LLN	**74.32%**	**59.05%**	**42.34%**	**28.14%**	**+0.59%**
40	SOC (no LLN)	71.63%	54.39%	37.92%	24.13%	(+0%)
	SOC + LLN	**74.03%**	**58.27%**	**41.75%**	**27.12%**	**+2.40%**

Table 8.8: Results for provable robustness on the CIFAR-10 dataset with and without LLN.

8.F.3 Additional experiments on CIFAR-100

In Table 8.9, we evaluate the effect of using Last Layer Normalization (LLN) on model performance across all models. We observe that there is a very significant gain in model performance.

In Tables 8.10 and 8.11, we compare the results using different convolution layers (namely "SOC", "BCOP" and "Cayley"). In Table 8.12, we evaluate the effect of using Householder activation of order 2 (σ_{Θ} defined in Theorem 20). All results in Tables 8.10, 8.11 and 8.12 include Last Layer Normalization (LLN) due to the strong results observed in Table 8.9.

In Table 8.12, the row "SOC + $\mathrm{HH}^{(2)}$" uses Householder activation of order 2 (σ_{Θ}, "+ CR" adds Certificate Regularization (CR) with $\gamma = 0.1$ (while using the HH activation of order 2 i.e σ_{Θ} as the activation function).

The rows "BCOP", "Cayley" and "SOC (Baseline)" all use the MaxMin activation function. "+ HH" replaces MaxMin with $2\mathrm{D}$ Householder activation of order 1 (σ_{θ}), "+ CR" adds Certificate Regularization (CR) with $\gamma = 0.1$ (while using σ_{θ} as the activation function).

Lip Conv net-	**Methods**	**Clean Acc.**	**Provable Robust Acc.** ($\rho =$) (36/255)	(72/255)	(108/255)	**Increase** (Clean)
5	SOC (no LLN)	42.71%	27.86%	17.45%	9.99%	(+0%)
	SOC + LLN	**45.86%**	**31.93%**	**21.17%**	**13.21%**	**+3.15%**
10	SOC (no LLN)	43.72%	29.39%	18.56%	11.16%	(+0%)
	SOC + LLN	**46.88%**	**33.32%**	**22.08%**	**13.87%**	**+3.16%**
15	SOC (no LLN)	42.92%	28.81%	17.93%	10.73%	(+0%)
	SOC + LLN	**47.72%**	**33.52%**	**21.89%**	**13.76%**	**+4.80%**
20	SOC (no LLN)	43.06%	29.34%	18.66%	11.20%	(+0%)
	SOC + LLN	**46.86%**	**33.48%**	**22.14%**	**14.10%**	**+3.80%**
25	SOC (no LLN)	43.37%	28.59%	18.18%	10.85%	(+0%)
	SOC + LLN	**46.32%**	**32.87%**	**21.53%**	**13.86%**	**+2.95%**
30	SOC (no LLN)	42.87%	28.74%	18.47%	11.21%	(+0%)
	SOC + LLN	**46.18%**	**32.82%**	**21.52%**	**13.52%**	**+3.31%**
35	SOC (no LLN)	42.42%	28.34%	18.10%	10.96%	(+0%)
	SOC + LLN	**45.22%**	**32.10%**	**21.28%**	**13.25%**	**+2.80%**
40	SOC (no LLN)	41.84%	28.00%	17.40%	10.28%	(+0%)
	SOC + LLN	**42.94%**	**29.71%**	**19.30%**	**11.99%**	**+1.10%**

Table 8.9: Results for provable robustness on the CIFAR-100 dataset with LLN and without LLN.

Lip Conv net-	Methods	Clean Acc.	Provable Robust Acc. ($\rho =$)			Increase
			36/255	72/255	108/255	(108/255)
5	BCOP	46.31%	31.55%	20.34%	12.52%	-0.69%
	Cayley	44.61%	31.01%	19.84%	12.43%	-0.78%
	SOC (Baseline)	45.86%	31.93%	21.17%	13.21%	(+0%)
	+ HH	**46.36%**	32.64%	21.19%	13.12%	-0.09%
	+ CR	45.82%	**32.99%**	**22.48%**	**14.79%**	**+1.58%**
10	BCOP	45.36%	31.71%	20.48%	12.40%	-1.47%
	Cayley	45.79%	31.91%	20.69%	12.78%	-1.09%
	SOC (Baseline)	46.88%	33.32%	22.08%	13.87%	(+0%)
	+ HH	**47.96%**	34.30%	22.35%	14.48%	+0.61%
	+ CR	47.07%	**34.53%**	**23.50%**	**15.66%**	**+1.79%**
15	BCOP	43.70%	30.11%	19.85%	12.29%	-1.47%
	Cayley	45.05%	31.60%	20.32%	12.93%	-0.83%
	SOC (Baseline)	47.72%	33.52%	21.89%	13.76%	(+0%)
	+ HH	**47.72%**	33.97%	22.45%	13.81%	+0.05%
	+ CR	47.61%	**34.54%**	**23.16%**	**15.09%**	**+1.33%**
20	BCOP	39.77%	27.20%	17.44%	10.49%	-3.61%
	Cayley	39.68%	26.93%	17.06%	10.48%	-3.62%
	SOC (Baseline)	46.86%	33.48%	22.14%	14.10%	(+0%)
	+ HH	47.71%	34.22%	22.93%	14.57%	+0.47%
	+ CR	**47.84%**	**34.77%**	**23.70%**	**15.84%**	**+1.74%**

Table 8.10: Results for provable robustness on the CIFAR-100 dataset using shallow networks. All of these results include Last Layer Normalization (LLN).

Lip Conv net-	Methods	Clean Acc.	Provable Robust Acc. ($\rho =$) 36/255	72/255	108/255	Increase (108/255)
25	BCOP	34.15%	21.57%	13.52%	7.97%	-5.89%
	Cayley	33.93%	21.93%	13.68%	8.19%	-5.67%
	SOC (Baseline)	46.32%	32.87%	21.53%	13.86%	(+0%)
	+ HH	**47.70%**	34.00%	22.67%	14.57%	+0.71%
	+ CR	46.87%	**34.09%**	**23.41%**	**15.61%**	**+1.75%**
30	BCOP	29.73%	18.69%	10.80%	6%	-7.52%
	Cayley	28.67%	18.05%	10.43%	6.09%	-7.43%
	SOC (Baseline)	46.18%	32.82%	21.52%	13.52%	(+0%)
	+ HH	46.80%	33.72%	22.70%	14.31%	+0.79%
	+ CR	**46.92%**	**34.17%**	**23.21%**	**15.84%**	**+2.32%**
35	BCOP	25.65%	14.88%	8.47%	4.30%	-8.95%
	Cayley	27.75%	16.37%	9.52%	5.40%	-7.85%
	SOC (Baseline)	45.22%	32.10%	21.28%	13.25%	(+0%)
	+ HH	46.21%	32.80%	21.55%	14.13%	+0.88%
	+ CR	**46.88%**	**33.64%**	**23.34%**	**15.73%**	**+2.48%**
40	BCOP	30.66%	18.68%	10.46%	5.92%	-6.07%
	Cayley	25.54%	14.91%	8.37%	4.40%	-7.59%
	SOC (Baseline)	42.94%	29.71%	19.30%	11.99%	(+0%)
	+ HH	**45.84%**	**32.79%**	21.98%	14.07%	+2.08%
	+ CR	45.03%	32.57%	**22.37%**	**14.76%**	**+2.77%**

Table 8.11: Results for provable robustness on the CIFAR-100 dataset using deeper networks. All of these results include Last Layer Normalization (LLN).

Lip Conv net-	Methods	Clean Acc.	Provable Robust Acc. ($\rho =$)			Increase
			36/255	72/255	108/255	(108/255)
5	**SOC + HH$^{(2)}$**	46.61%	32.50%	21.34%	13.22%	+0.01%
	+ CR	**46.69%**	**33.22%**	**22.34%**	**14.30%**	**+1.09%**
10	**SOC + HH$^{(2)}$**	47.47%	33.32%	21.84%	13.75%	-0.01%
	+ CR	**47.53%**	**34.52%**	**23.06%**	**15.07%**	**+1.31%**
15	**SOC + HH$^{(2)}$**	47.19%	33.67%	22.36%	13.78%	-0.09%
	+ CR	**47.22%**	**34.04%**	**22.98%**	**15.28%**	**+1.41%**
20	**SOC + HH$^{(2)}$**	**47.86%**	33.93%	22.44%	14.41%	+0.31%
	+ CR	47.54%	**34.32%**	**23.53%**	**15.54%**	**+1.44%**
25	**SOC + HH$^{(2)}$**	**47.86%**	33.97%	22.78%	14.59%	+0.73%
	+ CR	47.50%	**34.38%**	**23.92%**	**15.92%**	**+2.06%**
30	**SOC + HH$^{(2)}$**	46.23%	32.64%	21.95%	14.00%	+0.48%
	+ CR	**46.36%**	**33.20%**	**22.70%**	**14.85%**	**+1.33%**
35	**SOC + HH$^{(2)}$**	**46.06%**	32.35%	21.33%	13.65%	+0.40%
	+ CR	45.78%	**33.24%**	**22.39%**	**14.78%**	**+1.53%**
40	**SOC + HH$^{(2)}$**	43.81%	30.59%	20.08%	12.56%	+0.57%
	+ CR	**45.61%**	**32.50%**	**22.36%**	**14.84%**	**+2.85%**

Table 8.12: Results for provable robustness on CIFAR-100 using HH activation of Order 2 (σ_Θ). Increase (108/255) is calculated with respect to SOC Baseline (from Tables 8.10, 8.11). All of these results include Last Layer Normalization (LLN).

Chapter 9: Novel pooling layer, faster gradient computation and second-order robustness certification procedures for lipschitz networks

9.1 Introduction

The Lipschitz constant [1] of a neural network $f : \mathbb{R}^d \to \mathbb{R}^c$, denoted by $\mathrm{Lip}(f)$, controls the change in the output divided by change in the input (both changes measured in the l_2 norm). Previous work provides evidence that a small Lipschitz constant is useful for interpretable saliency maps [229], generalization bounds [205], Wasserstein distance estimation [230], adversarial robustness [15] and preventing gradient explosion during backpropagation [203]. Several prior works [196, 207] use heuristic methods to enforce Lipschitz constraints to successfully address problems such as stabilizing GAN training. However, these methods do not enforce a guaranteed lipschitz bound and it remains challenging to achieve strong results with provable guarantees.

Using the composition property i.e. $\mathrm{Lip}(g \circ h) \leq \mathrm{Lip}(g)\ \mathrm{Lip}(h)$, we can construct a 1-Lipschitz neural network by constraining each layer to be 1-Lipschitz. However, a key difficulty with this approach is that because a 1-Lipschitz layer can only reduce the gradient norm during backpropagation, for deeper networks, this results in small gradient values for layers closer to the input making training slow and difficult. To address this problem, [133] introduce Gradient Norm Preserving (GNP) architectures where each layer preserves the gradient norm during backpropa-

[1] In this chapter, we assume the Lipschitz constant under the l_2 norm.

SOC layers	Time/epoch (secs) Ours	Previous	Reduction	Standard accuracy Ours	Previous	Provable robust accuracy Ours	Previous
6	**25.34**	30.63	**-17.27%**	**79.36%**	76.68%	**67.13%**	60.09%
11	**37.11**	48.44	**-23.39%**	**79.57%**	77.73%	**66.75%**	62.82%
16	**49.27**	66.74	**-26.17%**	**79.44%**	77.78%	**66.99%**	62.75%
21	**61.59**	83.18	**-25.96%**	**79.13%**	77.50%	**66.45%**	63.31%
26	**71.51**	100.70	**-28.99%**	**79.19%**	77.18%	**66.28%**	62.46%
31	**84.00**	119.55	**-29.74%**	**78.64%**	74.43%	**66.05%**	59.65%
36	**95.06**	137.10	**-30.66%**	**78.57%**	72.73%	**65.94%**	57.18%
41	**106.01**	156.25	**-32.16%**	**78.41%**	71.33%	**65.51%**	55.74%

Table 9.1: Results on CIFAR-10 using faster gradients, projection pooling, CRC-Lip

gation. For 1-Lipschitz Convolutional Neural Networks (CNNs), this involves using orthogonal convolutions (convolution layers with an orthogonal Jacobian matrix) [232, 242, 243, 250, 251] and using a class of GNP activation functions called HouseHolder activations [252].

Among orthogonal convolutions, SOC [243] achieves state-of-the-art results on CIFAR-10, 100 [252]. SOC uses the following two mathematical properties to construct an orthogonal convolution: (a) the exponential of a skew symmetric matrix is orthogonal and (b) the matrix exponential can be computed using the exponential series. A drawback of using the exponential series is that it requires us to apply the convolution operation multiple times to achieve a reasonable approximation of an orthogonal matrix. Consequently, during backpropagation, computing the gradient with respect to the weights for a single SOC layer requires us to compute the weight gradient for each of these convolutions resulting in significant training overhead. In this chapter, we show that because the matrix is skew symmetric, using some approximations, the gradients with respect to weights can be computed in significantly reduced time with no loss in performance. This enables us to reduce the training time per epoch using SOC by $> 30\%$ for networks with large number of SOC layers (Results in Table 9.1).

Another limitation of 1-Lipschitz CNNs is that their performance is often significantly

below compared to that of standard CNNs. Recently, [252] introduced a procedure for certifying robustness by relaxing the orthogonality requirements of the last linear layer (i.e. the linear layer mapping penultimate neurons to class logits) achieving state of the art results on CIFAR-10, 100 [57]. Since MLPs are more expressive than linear layers, one would expect improved *standard accuracy* by replacing the last linear layer with them. However, the resulting networks are not 1-Lipschitz and achieving high robustness guarantees (*provable robust accuracy*) is difficult.

To certify robustness for these networks, note that since the mapping from input to penultimate layer is 1-Lipschitz, the robustness certificate for the penultimate output also provides a certificate for the input. Thus, we first replace the linear layer mapping penultimate output to logits with a 1-hidden layer MLP (Multi layer Perceptron) because such MLPs are easier to certify compared to deep MLPs. To certify robustness for the 1-hidden layer MLP, we use the Curvature-based Robustness Certificate or CRC [142]. To train MLP to have high robustness, we use a variant of adversarial training that only applies adversarial perturbations to the MLP (not the whole network). We call our certification procedure $\mathrm{CRC\text{-}Lip}$. This results in improved results for both the standard and provable robust accuracy across all network architectures on CIFAR-10 ($\geq 1.63\%,\ \geq 3.14\%$) and CIFAR-100 ($\geq 2.49\%,\ \geq 2.27\%$) . Tables 9.3 and 9.4 respectively.

While several works have attempted to construct novel and more expressive orthogonal convolution layers and GNP activation functions, current state-of-the-art 1-Lipschitz CNNs still use pooling layers by taking the $\max$ of different elements. In this chapter, we introduce a class of 1-Lipschitz pooling layers called *projection pooling* using the following mathematical property: Given a manifold $\mathcal{M} \subset \mathbb{R}^n$ and input $\mathbf{x} \in \mathbb{R}^n$, the function $d_{\mathcal{M}} : \mathbb{R}^n \to \mathbb{R}$ defined as the l_2 distance of $\mathbf{x}$ to $\mathcal{M}$ is 1-Lipschitz. Thus, to construct a pooling layer, we can first define a *learnable manifold* with parameters Θ (Example in Section 9.6). During training, for input

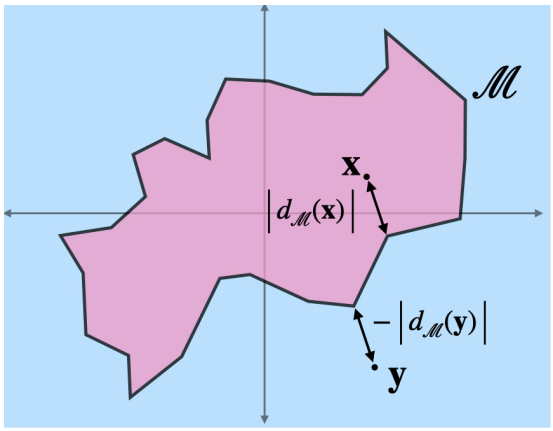


Figure 9.1: Example of 2D projection pooling using a piecewise linear curve

$\mathbf{x} \in \mathbb{R}^n$, the pooling layer simply outputs $d_{\mathcal{M}}(\mathbf{x}) \in \mathbb{R}$ as the output, resulting in decrease in input dimension by a factor of n. Moreover, since the output $d_{\mathcal{M}}(\mathbf{x})$ is also a function of Θ, Θ can be learned during training. However, solving for the distance $d_{\mathcal{M}}(\mathbf{x})$ can be difficult especially when $\mathcal{M}$ is a high-dimensional manifold.

To address this limitation, (a) we use 2D projection pooling layers that reduce the dimension by factor of 2 and (b) we construct these layers using piecewise linear curves for which the distance can be computed efficiently by computing the minimum distance to all line segments and the connecting points (Example in Supp. Figure 9.1). If the curve is closed and without self-intersections, we can also define a signed projection pooling for which the signs of the output for points inside and outside the region enclosed by the curve are different ($\mathbf{x}$ and $\mathbf{y}$ in Figure 9.1). This allows the subsequent layers to distinguish between the inputs inside and outside the region. In this chapter, we show some preliminary results using a simple 2D projection pooling layer (Section 9.6). We leave the problem of constructing high performance 2D projection pooling layers open for future research.

In summary, in this chapter, we make the following contributions:

- We introduce a method for faster computation of the weight gradient for SOC layers. For deeper networks, we observe reduction in training time per epoch by $> 30\%$ (Table 9.1).

- We introduce a certification procedure called $\mathrm{CRC\text{-}Lip}$ that replaces the last linear layer with a 1-hidden layer MLP and results in significantly improved standard and provable robust accuracy. For deeper networks ($\geq$ 35 layers), we observe improvements of $\geq 5.84\%, \geq 8.00\%$ in standard and $\geq 8.76\%, \geq 8.65\%$ in provable robust accuracy (l_2 radius $36/255$) on CIFAR-10 and CIFAR-100 respectively.

- We introduce a large class of 1-Lipschitz pooling layers called *projection pooling* using the mathematical property that the l_2 distance (denoted by $d_{\mathcal{M}}(\mathbf{x})$) of an input $\mathbf{x}$ to the manifold $\mathcal{M}$ is 1-Lipschitz.

- On CIFAR-10, across all architectures, we achieve the best standard and provable robust accuracy (at $36/255$) of $79.57, 67.13\%$ respectively (gain of $+1.79\%, +3.82\%$ from prior works). Similarly, on CIFAR-100, we achieve $51.84, 39.27\%$ ($+3.78\%, +4.75\%$ from prior works). These results establish new state-of-the-art results in the standard and provable robust accuracy on both datasets.

9.2 Related work

Provable defenses against adversarial examples: For a provably robust classifier, we can guarantee that its predictions remain constant within some region around the input. Most of the existing methods for provable robustness either bound the Lipschitz constant or use convex relaxations [141, 142, 144, 145, 146, 166, 182, 187, 236, 237, 238, 244, 245, 246, 247, 253, 254, 255, 256,

257, 258, 259, 260, 261, 262, 263, 264, 265, 266]. However, these methods are often not scalable to large neural networks while achieving high performance. In contrast, randomized smoothing [135, 136, 137, 190, 191, 192, 239, 240, 248, 267, 268] scales to large neural networks but is a *probabilistically certified defense*: certifying robustness with high probability requires generating a large number of noisy samples leading to high inference-time. In contrast, the defense we propose in this chapter is deterministic and not comparable to randomized smoothing.

Provably Lipschitz neural networks: The class of Gradient Norm Preserving (GNP) and 1-Lipschitz fully connected neural networks was first introduced by [133]. To design each layer to be GNP, they orthogonalize weight matrices and use a class of piecewise linear GNP activations called $\mathrm{GroupSort}$. Later, [252] proved that for any piecewise linear GNP function to be continuous, different Jacobian matrices in a neighborhood must change via householder transformations, implying that $\mathrm{GroupSort}$ is a special case of more general $\mathrm{HouseHolder}$ activations. Several previous works enforce Lipschitz constraints on convolution layers using spectral normalization, clipping or regularization [134, 214, 225, 228, 234]. However, these methods either enforce loose lipschitz bounds or do not scale to large networks. To ensure that the Lipschitz constraint on convolutional layers is tight, recent works construct convolution layers with an orthogonal Jacobian [232, 242, 243, 250, 251, 269]. These approaches avoid the aforementioned issues and allow training of large, provably 1-Lipschitz CNNs achieving state-of-the-art results for deterministic l_2 robustness.

9.3 Problem setup and Notation

For a vector $\mathbf{v}$, $\mathbf{v}_j$ denotes its jth element. For a matrix $\mathbf{A}$, $\mathbf{A}_{j,:}$ and $\mathbf{A}_{:,k}$ denote the jth row and k^{th} column respectively. Both $\mathbf{A}_{j,:}$ and $\mathbf{A}_{:,k}$ are assumed to be column vectors (thus $\mathbf{A}_{j,:}$ is the transpose of jth row of $\mathbf{A}$). $\mathbf{A}_{j,k}$ denotes the element in j^{th} row and k^{th} column of $\mathbf{A}$. $\mathbf{A}_{:j,:k}$ denotes the matrix containing the first j rows and k columns of $\mathbf{A}$. We define $\mathbf{A}_{:j} = \mathbf{A}_{:j,:}$ and $\mathbf{A}_{j:} = \mathbf{A}_{j:,:}$. Similar notation applies to higher order tensors. $\mathbf{I}$ denotes the identity matrix, $\mathbb{R}$ to denote the field of real numbers. We construct a 1-Lipschitz neural network, $f : \mathbb{R}^d \to \mathbb{R}^c$ (d is the input dimension, c is the number of classes) by composing 1-Lipschitz convolution layers and GNP activation functions. We often use the abbreviation $f_i - f_j : \mathbb{R}^d \to \mathbb{R}$ to denote the function so that:

$$(f_i - f_j)(\mathbf{x}) = f_i(\mathbf{x}) - f_j(\mathbf{x}), \qquad \forall\, \mathbf{x} \in \mathbb{R}^d$$

For a matrix $\mathbf{A} \in \mathbb{R}^{q\times r}$ and a tensor $\mathbf{B} \in \mathbb{R}^{p\times q\times r}$, $\overrightarrow{\mathbf{A}}$ denotes the vector constructed by stacking the rows of $\mathbf{A}$ and $\overrightarrow{\mathbf{B}}$ by stacking the vectors $\overrightarrow{\mathbf{B}_{j,:,:}},\ j \in [p-1]$ so that:

$$\left(\overrightarrow{\mathbf{A}}\right)^T = \left[\mathbf{A}_{0,:}^T\ ,\ \mathbf{A}_{1,:}^T\ ,\ \dots\ ,\ \mathbf{A}_{q-1,:}^T\right], \quad \left(\overrightarrow{\mathbf{B}}\right)^T = \left[\left(\overrightarrow{\mathbf{B}_{0,:,:}}\right)^T\ ,\ \left(\overrightarrow{\mathbf{B}_{1,:,:}}\right)^T\ ,\ \dots\ ,\ \left(\overrightarrow{\mathbf{B}_{p-1,:,:}}\right)^T\right]$$

For a 2D convolution filter, $\mathbf{L} \in \mathbb{R}^{p\times q\times r\times s}$ and input $\mathbf{X} \in \mathbb{R}^{q\times n\times n}$, we use $\mathbf{L} \star \mathbf{X} \in \mathbb{R}^{p\times n\times n}$ to denote the convolution of filter $\mathbf{L}$ with $\mathbf{X}$. We use the the notation $\mathbf{L} \star^i \mathbf{X} \triangleq \mathbf{L} \star^{i-1} (\mathbf{L} \star \mathbf{X})$ and $\mathbf{L} \star^0 \mathbf{X} = \mathbf{X}$. Unless specified, we assume zero padding and stride 1 in each direction.

9.4 Faster gradient computation for Skew Orthogonal Convolutions

Among the existing orthogonal convolution layers in the literature, Skew Orthogonal Convolutions (or SOC) by [243] achieves state-of-the-art results on CIFAR-10 and CIFAR-100 [252]. SOC first constructs a convolution filter $\mathbf{L}$ whose Jacobian is skew-symmetric. This is followed by a convolution exponential [209] operation. Since the exponential of a skew-symmetric matrix is orthogonal, the Jacobian of the resulting layer is an orthogonal matrix.

However, a drawback of this procedure is that convolution exponential requires multiple convolution operations per SOC layer to achieve a reasonable approximation of the orthogonal Jacobian. Consequently, if we use k convolution operations in the SOC layer during forward pass, we need to compute the gradient with respect to the weights $\mathbf{L}$ (called *convolution weight gradient*) per convolution operation (k times) which can lead to slower training time especially when the number of SOC layers is large. To address this limitation, we show that even if we use k convolutions in the forward pass of an SOC layer, the weight gradient for the layer can be computed using a *single convolution weight gradient* during backpropagation leading to significant reduction in training time.

For simplicity, let us first consider the case of an orthogonal fully connected layer (i.e. not convolution) with the same input and output size (n). Later, we will see that our analysis leads to improvements for orthogonal convolutional layers. Further, assume that the weights i.e. $\mathbf{A} \in \mathbb{R}^{n \times n}$ are skew-symmetric i.e. $\mathbf{A} = -\mathbf{A}^T$ and given the input $\mathbf{x} \in \mathbb{R}^n$, the output $\mathbf{z} \in \mathbb{R}^n$ is

computed as follows:

$$\mathbf{z} = \left(\sum_{i=0}^{k} \frac{\mathbf{A}^i}{i!} \right) \mathbf{x} + \mathbf{b}, \qquad \text{where} \quad \mathbf{A} = -\mathbf{A}^T \tag{9.1}$$

We approximate the exponential series: $\exp(\mathbf{A}) = \sum_{i=0}^{\infty} \mathbf{A}^i / i!$ using finite number of terms (k).

Forward pass ($\mathbf{z}$): To compute $\mathbf{z}$ during the forward pass, we use the following iterations:

$$\mathbf{u}^{(i)} = \begin{cases} \mathbf{x} & i = k-1 \\ \mathbf{x} + (\mathbf{A}\mathbf{u}^{(i+1)})/(i+1) & i \leq k-2 \end{cases} \tag{9.2}$$

It can be shown that $\mathbf{z} = \mathbf{u}^{(0)}$ (Supp. 9.B.2). During backpropagation, given the gradient of loss ℓ w.r.t. $\mathbf{z}$ i.e. $\nabla_{\mathbf{z}} \ell$, we want to compute $\nabla_{\mathbf{x}} \ell$ (input gradient) and $\nabla_{\mathbf{A}} \ell$ (weight gradient).

Input gradient ($\nabla_{\mathbf{x}} \ell$): To compute $\nabla_{\mathbf{x}} \ell$, observe that $\mathbf{z}$ is a linear function of $\mathbf{x}$. Thus, using the chain rule, skew-symmetricity ($\mathbf{A}^T = -\mathbf{A}$) and the property $\left(\mathbf{A}^i\right)^T = \left(\mathbf{A}^T\right)^i$, we have:

$$\nabla_{\mathbf{x}} \ell = \left(\sum_{i=0}^{k} \frac{\mathbf{A}^i}{i!} \right)^T (\nabla_{\mathbf{z}} \ell) = \left(\sum_{i=0}^{\infty} \frac{(-\mathbf{A})^i}{i!} \right) (\nabla_{\mathbf{z}} \ell)$$

We can again approximate the exponential series using the same number of terms as in the forward pass i.e. k. To compute the finite term approximation, we use the following iterations:

$$\mathbf{v}^{(i)} = \begin{cases} \nabla_{\mathbf{z}} \ell & i = k-1 \\ \nabla_{\mathbf{z}} \ell - (\mathbf{A}\mathbf{v}^{(i+1)})/(i+1) & i \leq k-2 \end{cases} \tag{9.3}$$

Similar to forward pass, it can be shown that $\nabla_{\mathbf{x}} \ell = \mathbf{v}^{(0)}$ (Supp. 9.B.2).

Weight gradient ($\nabla_{\mathbf{A}}\, \ell$): We first derive the exact expression for $\nabla_{\mathbf{A}}\, \ell$ in the Theorem below:

Theorem 21. *The gradient of the loss function ℓ w.r.t $\mathbf{A}$ i.e. $\nabla_{\mathbf{A}}\, \ell$ is given by:*

$$\nabla_{\mathbf{A}}\, \ell = -\sum_{i=1}^{k} \left(\left(\mathbf{A}^{i-1}\mathbf{x}\right) \left(\mathbf{v}^{(i)}\right)^T - \mathbf{v}^{(i)} \left(\mathbf{A}^{i-1}\mathbf{x}\right)^T \right) \tag{9.4}$$

where $\mathbf{v}^{(i)}$ is defined as in equation (9.3).

Note that the first outer product i.e. $\left(\mathbf{A}^{i-1}\mathbf{x}\right) \left(\mathbf{v}^{(i)}\right)^T$ and the second i.e. $\left(\mathbf{v}^{(i)}\right) \left(\mathbf{A}^{i-1}\mathbf{x}\right)^T$ are transpose of each other implying that each term in the summation is skew-symmetric. Although these outer products can be computed in a straightforward way for orthogonal *fully connected* layers, this is not the case for orthogonal *convolution* layers (SOC in this case). This is because, for a convolution filter $\mathbf{L} \in \mathbb{R}^{p \times q \times r \times s}$, the term analogous to $\mathbf{A}^{i-1}\mathbf{x}$ is a patch of size $q \times r \times s$ and that analogous to $\mathbf{v}^{(i)}$ is another patch of size $p \times 1 \times 1$ resulting in an outer product of the desired size $p \times q \times r \times s$ *per patch*. Thus, for SOC layers, each term inside the summation is computed by summing over the outer products for all such patches. For large input sizes, the number of such patches can often be large, making this computation expensive. To address this limitation, we use the following approximation:

$$\nabla_{\mathbf{A}}\, \ell \approx -\left(\mathbf{u}^{(1)} \left(\mathbf{v}^{(1)}\right)^T - \mathbf{v}^{(1)} \left(\mathbf{u}^{(1)}\right)^T \right) \tag{9.5}$$

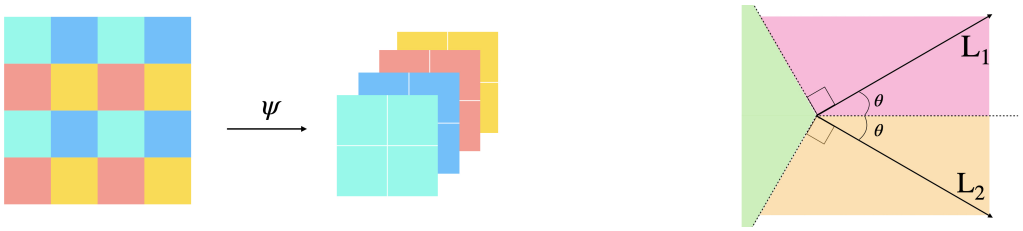
In Supp. 9.B.1, we show that the above approximation is principled because after subtracting the exact and approximation gradients (equations (9.4) and (9.5)) and simplifying, each term in the resulting series is divided by a large value in its denominator (≈ 0). This approximation is useful because in equation (9.5), the outer product needs to be computed once whereas in equation

(9.4), the outer products need to be computed k times. Also, both $\mathbf{u}^{(1)}$ and $\mathbf{v}^{(1)}$ are computed while computing $\mathbf{u}^{(0)} = \mathbf{z}$ during the forward pass and $\mathbf{v}^{(0)} = \nabla_{\mathbf{x}} \mathbf{z}$ during the backward pass using the recurrences in equations (9.2) and (9.3). Thus, we can simply store $\mathbf{u}^{(1)}, \mathbf{v}^{(1)}$ during the forward,backward pass respectively so that $\nabla_{\mathbf{A}} \ell$ can be computed directly using equation (9.5). In our experiments, we observe that this leads to significant reduction in training time with almost no drop in performance (Tables 9.1, 9.3, 9.4).

9.5 Curvature-based Robustness Certificate

A key property of 1-Lipschitz CNNs is that the output of each layer is 1-Lipschitz with respect to the input. Given an input $\mathbf{x} \in \mathbb{R}^d$, consider the penultimate output $g(\mathbf{x}) \in \mathbb{R}^m$ and logits $f(\mathbf{x}) \in \mathbb{R}^c$ for some 1-Lipschitz CNN. Existing robustness certificates [232, 252] rely on the *linearity* of the function from $g(\mathbf{x}) \to f(\mathbf{x})$. However, since MLPs have higher expressive power than linear functions [270], one way to improve performance could be to replace this mapping with MLPs. However, certifying robustness for deep MLPs is difficult.

To address these challenges, we first replace the mapping from $g(\mathbf{x}) \to f(\mathbf{x})$ with a 1-hidden layer MLP because they are easier to certify compared to deeper networks. Because computing exact certificates for ReLU networks is known to be NP-complete [271], we use the differentiable $\mathrm{Softplus}$ activation [226] to certify robustness. To certify robustness, we use the Curvature-based Robustness Certificate or CRC [142] because it provides exact certificates for a significant fraction of inputs for shallow MLPs. CRC is discussed in detail in Chapter 5. Let $g : \mathbb{R}^d \to \mathbb{R}^m$ be a 1-Lipschitz continuous function and $h : \mathbb{R}^m \to \mathbb{R}^c$ be a 1-hidden layer MLP such that $f = h \circ g$. Further, let l be the predicted class for input $\mathbf{x}$ i.e. $f_l(\mathbf{x}) \geq \max_{i \neq l} f_i(\mathbf{x})$.



(a) Rearrangement operation [243] (b) 2D projection pooling

Figure 9.2: Illustration of the rearrangement operation (left) and 2D projection pooling (right).

Since g is 1-Lipschitz, it can be shown that if h is provably robust in an l_2 radius R around input $g(\mathbf{x})$, then f is also provably robust in the l_2 radius R around $\mathbf{x}$. The resulting procedure is called CRC-Lip and is given in the following proposition:

Proposition 3. (CRC-Lip) *For input* $\mathbf{x}$ *such that* $f_l(\mathbf{x}) \geq \max_{i \neq l} f_i(\mathbf{x})$*, let* R *be the robustness certificate for* h *using input* $g(\mathbf{x})$*, then* R *is also the robustness certificate for the function* f*:*

$$\min_{i \neq l} \min_{h_l(\mathbf{y}^*)=h_i(\mathbf{y}^*)} \|\mathbf{y}^* - g(\mathbf{x})\|_2 \geq R \implies \min_{i \neq l} \min_{f_l(\mathbf{x}^*)=f_i(\mathbf{x}^*)} \|\mathbf{x}^* - \mathbf{x}\|_2 \geq R \tag{9.6}$$

Proof is in Supp. 9.A.2. In our experiments, we find that although replacing the linear layer with an MLP achieves high standard accuracy, directly using the above certificate often leads to very small robustness certificates and thus low certified robust accuracy. To address this problem, we introduce (a) an adversarial training procedure that *only* applies to the input of the MLP $g(\mathbf{x})$ (i.e. not the input of the neural network $\mathbf{x}$) and (b) curvature regularization to reduce the curvature of the MLP. It was shown in [243] that combining adversarial training with curvature regularization leads to significantly improved provable robust accuracy with small reduction in

standard accuracy. This results in the following loss function for training:

$$\min_{\Omega,\ \Psi} \quad \mathbb{E}_{(\mathbf{x},l)\sim\mathcal{D}}\left[\left(\max_{\|\mathbf{y}^*-g_\Psi(\mathbf{x})\|_2\leq\rho} \ell\left(h_\Omega(\mathbf{y}^*,l)\right)\right)+\gamma\,\mathcal{K}_h\right]$$

In the above equation, Ω denote the parameters of the MLP (i.e. h), Ψ are the parameters of the 1-Lipschitz function g, $\mathcal{K}_h$ is the bound on the curvature of the MLP, ρ denotes the l_2 perturbation radius and γ denotes the curvature regularization coefficient. The curvature bound $\mathcal{K}_h$ is the same as in [142]. Also, since we apply adversarial perturbations in the penultimate layer, we also need to backpropagate through this procedure to enable training of previous layers. To this end, we simply use an identity map (same gradient output as the gradient input) to backpropagate through the adversarial training procedure and find that it works well in practice, achieving significantly better results compared to the state-of-the-art. It is possible that a more principled method of backpropagation may lead to better results and we leave that avenue open for future research.

9.6 Projection pooling layers

In 1-Lipschitz CNNs, pooling is usually carried out as follows: given input $\mathbf{X} \in \mathbb{R}^{q\times r\times r}$ (r is even), we first use *rearrangement* [241] illustrated in Figure 9.2b to construct $\mathbf{X}' \in \mathbb{R}^{4q\times(r/2)\times(r/2)}$. Next, we apply an orthogonal convolution which gives an output of the same size $\mathbf{Z} \in \mathbb{R}^{4q\times(r/2)\times(r/2)}$ and divide it into two tensors of equal sizes (along the channel dimension) giving $\mathbf{Z}_{:2q} \in \mathbb{R}^{2q\times(r/2)\times(r/2)}$ and $\mathbf{Z}_{2q:} \in \mathbb{R}^{2q\times(r/2)\times(r/2)}$. We then define $\max(\mathbf{Z}_{:2q},\ \mathbf{Z}_{2q:})$ (or either one of $\mathbf{Z}_{:2q},\ \mathbf{Z}_{2q:}$) as the output of the pooling layer.

Although $\max$ is 1-Lipschitz, its expressive power is limited. For example, consider the 1-Lipschitz function $\|x,y\|_2=\sqrt{x^2+y^2}$. It is easy to see that if $x=y$, the error between $\|x,y\|_2$

and $\max(x, y)$ can be arbitrarily large for large values of x, y. To address such limitations, we construct expressive pooling layers using the following mathematical property:

Theorem 22. *Given $\mathbf{x} \in \mathbb{R}^n$ and manifold $\mathcal{M} \subset \mathbb{R}^n$, the distance function (in l_2 norm) is 1-Lipschitz:*

$$d_{\mathcal{M}}(\mathbf{x}) = \min_{\mathbf{x}^* \in \mathcal{M}} \|\mathbf{x}^* - \mathbf{x}\|_2 \implies |d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y})| \leq \|\mathbf{x} - \mathbf{y}\|_2 \tag{9.7}$$

The above theorem provides a powerful method for constructing a *learnable pooling layer* by selecting a *learnable manifold* $\mathcal{M}_\Theta \subset \mathbb{R}^r$ (Θ denotes the set of learnable parameters). To apply the pooling operation to an input $\mathbf{x} \in \mathbb{R}^n$, we simply output its l_2 distance to the manifold $\mathcal{M}_\Theta$ denoted by $d_{\mathcal{M}_\Theta}(\mathbf{x})$. Since $d_{\mathcal{M}_\Theta}(\mathbf{x}) \in \mathbb{R}$ while $\mathbf{x} \in \mathbb{R}^n$, this operation reduces the input size by a factor of n.

As an example, let $\mathcal{M}_\mathbf{u} = \{\mathbf{u}\}$. Here $\mathbf{u} \in \mathbb{R}^n$ is the learnable parameter and the distance function $d_{\mathcal{M}_\mathbf{u}}(\mathbf{x}) = \|\mathbf{x} - \mathbf{u}\|_2$. Even in this very simple case, for $\mathbf{u} = \mathbf{0}$, $d_{\mathcal{M}_\mathbf{u}}(\mathbf{x}) = \|\mathbf{x}\|_2$ and for $n = 2$, this function can learn to represent the function $\|x, y\|_2$ discussed earlier exactly. We also prove that a *signed* l_2 distance function can be defined when $\mathcal{M}$ satisfies certain properties:

Corollary 3. *If $\mathcal{M}$ is a connected manifold that divides $\mathbb{R}^n$ into two nonempty connected sets $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \cap \mathcal{B} = \phi$ and every path from $\mathbf{a} \in \mathcal{A}$ and $\mathbf{b} \in \mathcal{B}$ intersects a point on $\mathcal{M}$, then there exists a signed l_2 distance function with different signs in $\mathcal{A}$ and $\mathcal{B}$.*

Proofs of Theorems 22 and Corollary 3 are given in Supp. 9.A.3 and 9.A.4 respectively. In practice, computing $d_{\mathcal{M}}$ can be difficult for high-dimensional $\mathcal{M}$. To tackle this issue, we use 2D pooling layers where $\mathcal{M}$ is defined to be a piecewise linear curve in 2D. The l_2 distance can then

Output Size	Layer
$32\times32\times32$	Conv + MaxMin
$64\times16\times16$	LipBlock-n/5
$128\times8\times8$	LipBlock-n/5
$256\times4\times4$	LipBlock-n/5
$512\times2\times2$	LipBlock-n/5
$1024\times1\times1$	LipBlock-n/5
# of classes	Linear/MLP-1

(a) LipConvnet-n Architecture

Output Size	Layer	Repeats
$q \times r \times r$	Input	-
$q \times r \times r$	Conv + MaxMin	n/5 − 1
$4q \times (r/2) \times (r/2)$	Rearrange	1
$4q \times (r/2) \times (r/2)$	Conv	1
$2q \times (r/2) \times (r/2)$	Pooling	1

(b) LipBlock-n/5

Table 9.2: LipConvnet-n and LipBlock-n/5 architectures

be computed by finding the minimum distance to all the line segments and connecting points. We emphasize that even in this relatively simple case, $\mathcal{M}$ can have large number of parameters and $d_{\mathcal{M}}$ can still be efficient to compute because these individual distances can be computed in parallel. We use $\mathcal{M}_\theta$ defined below (illustrated in Figure 9.2b, lines $L_1,\ L_2$ correspond to $\phi = +\theta, -\theta$):

$$\mathcal{M}_\theta = \{R(\cos\phi,\ \sin\phi) : \quad R \geq 0,\ \phi \in \{+\theta, -\theta\}\}$$

In each colored region (Figure 9.2b), $d_{\mathcal{M}_\theta}$ can be computed using the following:

$$d_{\mathcal{M}_\theta}\left(R(\cos\alpha,\ \sin\alpha)\right) = \begin{cases} R\sin(\alpha-\theta), & 0 \leq \alpha \leq \theta + \pi/2 \\ -R\sin(\alpha+\theta), & 0 < -\alpha \leq \theta + \pi/2 \\ R, & \text{otherwise} \end{cases} \tag{9.8}$$

To apply projection pooling on $\mathbf{Z}_{:2q},\ \mathbf{Z}_{2q:}$ discussed previously, we output $d_{\mathcal{M}_\theta}\left(\mathbf{Z}_{:2q}, \mathbf{Z}_{2q:}\right)$.

9.7 Experiments

We perform experiments under the setting of provably robust image classification on CIFAR-10 and CIFAR-100 datasets. We use the $\mathrm{LipConvnet\text{-}}5, 10, 15, \ldots, 40$ architectures for comparison. We use SOC as the orthogonal convolution and MaxMin as the activation in all architectures. All experiments were performed using 1 NVIDIA GeForce RTX 2080 Ti GPU. All networks were trained for 200 epochs with initial learning rate of 0.1, dropped by a factor of 0.1 after 100 and 150 epochs. For adversarial training with curvature regularization, we use $\rho = 36/255$ (0.1411), $\gamma = 0.5$ for CIFAR-10 and $\rho = 0.2$, $\gamma = 0.75$ for CIFAR-100. We find that certifying robustness using $\mathrm{CRC\text{-}Lip}$ is computationally expensive for CIFAR-100 due to large number of classes. To address this issue, we only use classes with top-10 logits (instead of all 100 classes) for CIFAR-100 (Table 9.4). Since $\mathrm{CRC\text{-}Lip}$ requires us to solve a convex optimization, we consider a certificate to be valid if the input is correctly classified and gradient at the optimal solution is $\leq 10^{-6}$ (0 otherwise). All results are reported using the complete test sets of CIFAR-10 and CIFAR-100. We compare the provable robust accuracy using 3 different l_2 perturbation radii: $36/255$, $72/255$, $108/255$.

Table details: For the baseline ("Baseline" in Tables 9.3, 9.4), we use the standard $\max$ pooling with the certificate based on LLN [252] due to their superior performance over prior works. In Tables 9.3 and 9.4, for each architecture, "**+ Fast**" adds faster gradient computation, "**+ CRC**" replaces $\max$ pooling with projection pooling (equation (9.8)) and replaces the last linear layer with a 1-hidden layer MLP ($\mathrm{CRC\text{-}Lip}$ certificate) while also using faster gradients. For each architecture, the columns "Increase (Standard)" and "Increase (36/255)" denote the increase in standard and provable robust accuracy relative to "Baseline" standard and provable

LipConv net-	Methods	Clean Acc.	Provable Robust Accuracy 36/255	72/255	108/255	Increase (standard)	(36/255)
5	Baseline	76.68%	60.09%	42.89%	27.45%	_	_
	+ Fast	76.03%	60.12%	42.90%	27.91%	-0.65%	+0.03%
	+ Pool	77.53%	62.17%	44.94%	29.74%	+0.85%	+2.08%
	+ CRC	**79.36%**	**67.13%**	**52.49%**	**38.19%**	**+2.68%**	**+7.04%**
10	Baseline	77.73%	62.82%	45.62%	30.09%	_	_
	+ Fast	77.76%	62.42%	45.48%	30.38%	+0.03%	-0.40%
	+ Pool	77.31%	62.02%	46.04%	30.99%	-0.42%	-0.80%
	+ CRC	**79.57%**	**66.75%**	**53.17%**	**38.60%**	**+1.84%**	**+3.93%**
15	Baseline	77.78%	62.75%	46.34%	31.38%	_	_
	+ Fast	77.75%	62.52%	46.23%	31.19%	-0.03%	-0.23%
	+ Pool	77.75%	63.14%	47.01%	32.05%	-0.03%	+0.39%
	+ CRC	**79.44%**	**66.99%**	**52.56%**	**38.30%**	**+1.66%**	**+4.24%**
20	Baseline	77.50%	63.31%	46.42%	31.53%	_	_
	+ Fast	77.13%	62.05%	45.86%	31.13%	-0.37%	-1.26%
	+ Pool	77.61%	63.25%	47.10%	32.84%	+0.11%	-0.06%
	+ CRC	**79.13%**	**66.45%**	**52.45%**	**38.12%**	**+1.63%**	**+3.14%**
25	Baseline	77.18%	62.46%	45.78%	31.16%	_	_
	+ Fast	76.94%	61.91%	45.59%	30.69%	-0.24%	-0.55%
	+ Pool	75.44%	60.13%	45.02%	30.44%	-1.74%	-2.33%
	+ CRC	**79.19%**	**66.28%**	**51.74%**	**37.99%**	**+2.01%**	**+3.82%**
30	Baseline	74.43%	59.65%	43.76%	29.16%	_	_
	+ Fast	74.69%	58.84%	43.33%	28.93%	+0.26%	-0.81%
	+ Pool	74.12%	59.17%	42.76%	28.75%	-0.31%	-0.48%
	+ CRC	**78.64%**	**66.05%**	**51.31%**	**37.30%**	**+4.21%**	**+6.40%**
35	Baseline	72.73%	57.18%	42.08%	28.09%	_	_
	+ Fast	72.91%	57.58%	41.52%	27.37%	+0.18%	+0.40%
	+ Pool	72.84%	56.71%	40.41%	26.62%	+0.11%	-0.47%
	+ CRC	**78.57%**	**65.94%**	**52.04%**	**37.63%**	**+5.84%**	**+8.76%**
40	Baseline	71.33%	55.74%	39.32%	26.06%	_	_
	+ Fast	71.60%	56.15%	39.82%	25.63%	+0.27%	+0.41%
	+ Pool	70.19%	54.22%	37.51%	23.88%	-1.14%	-1.52%
	+ CRC	**78.41%**	**65.51%**	**51.32%**	**37.30%**	**+7.08%**	**+9.77%**

Table 9.3: Results for provable adversarial robustness on CIFAR-10

robust accuracy ($36/255$).

LipConvnet Architecture: We use a 1-Lipschitz CNN architecture called $\mathrm{LipConvnet\text{-}n}$ where n is a multiple of 5 and $\mathrm{n}+1$ is the total number of convolution layers. It consists of an initial SOC layer that expands the number of channels from 3 to 32. This is followed by 5 blocks

LipConv net-	Methods	Standard Acc.	Provable Robust Accuracy 36/255	72/255	108/255	Increase (standard)	(36/255)
5	Baseline	46.74%	32.45%	21.04%	12.88%	_	_
	+ Fast	46.18%	32.15%	20.79%	12.87%	-0.56%	-0.30%
	+ Pool	47.47%	33.51%	22.06%	13.73%	+0.73%	+1.06%
	+ CRC	**49.73%**	**36.05%**	**24.68%**	**16.31%**	**+2.99%**	**+3.60%**
10	Baseline	47.96%	34.33%	22.55%	14.23%	_	_
	+ Fast	47.24%	33.79%	22.38%	14.36%	-0.72%	-0.54%
	+ Pool	47.32%	34.01%	22.28%	14.04%	-0.64%	-0.32%
	+ CRC	**50.45%**	**36.60%**	**25.57%**	**16.93%**	**+2.49%**	**+2.27%**
15	Baseline	48.06%	34.52%	23.08%	14.70%	_	_
	+ Fast	47.97%	33.84%	22.66%	14.26%	-0.09%	-0.68%
	+ Pool	47.44%	34.12%	22.59%	14.48%	-0.62%	-0.40%
	+ CRC	**50.79%**	**37.50%**	**26.16%**	**17.27%**	**+2.73%**	**+2.98%**
20	Baseline	47.37%	33.99%	23.40%	14.69%	_	_
	+ Fast	46.41%	33.07%	22.06%	14.00%	-0.96%	-0.92%
	+ Pool	46.54%	32.75%	21.64%	13.22%	-0.83%	-1.24%
	+ CRC	**51.84%**	**38.54%**	**27.32%**	**18.53%**	**+4.47%**	**+4.55%**
25	Baseline	45.77%	32.08%	21.36%	13.64%	_	_
	+ Fast	45.28%	31.67%	20.69%	13.26%	-0.49%	-0.41%
	+ Pool	45.26%	31.87%	20.88%	13.65%	-0.51%	-0.21%
	+ CRC	**51.59%**	**39.27%**	**27.94%**	**19.06%**	**+5.82%**	**+7.19%**
30	Baseline	46.39%	33.08%	22.02%	13.77%	_	_
	+ Fast	45.86%	32.54%	21.18%	12.77%	-0.53%	-0.54%
	+ Pool	45.15%	31.53%	21.18%	13.14%	-1.24%	-1.55%
	+ CRC	**50.97%**	**38.77%**	**27.73%**	**19.28%**	**+4.58%**	**+5.69%**
35	Baseline	43.42%	30.36%	19.71%	12.66%	_	_
	+ Fast	42.78%	29.88%	19.73%	12.52%	-0.64%	-0.48%
	+ Pool	42.33%	28.65%	18.59%	11.75%	-1.09%	-1.71%
	+ CRC	**51.42%**	**39.01%**	**28.94%**	**20.29%**	**+8.00%**	**+8.65%**
40	Baseline	41.72%	28.53%	18.37%	11.49%	_	_
	+ Fast	42.07%	28.51%	18.86%	11.89%	+0.35%	-0.02%
	+ Pool	40.27%	27.09%	17.36%	10.62%	-1.45%	-1.44%
	+ CRC	**50.11%**	**38.69%**	**28.45%**	**20.05%**	**+8.39%**	**+10.16%**

Table 9.4: Results for provable robustness against adversarial examples on CIFAR-100.

that reduce spatial dimensions (height and width) by half while doubling the number of channels. The architecture is summarized in Tables 9.2a and 9.2b. The last layer outputs the class logits and is either a linear layer in which case we use LLN to certify robustness or a single-hidden layer MLP where we use CRC-Lip.

Lip Conv net-	Results on CIFAR-10 Time per epoch (s) Ours	Previous	Reduction	Results on CIFAR-100 Time per epoch (s) Ours	Previous	Reduction
5	**25.343**	30.633	**-17.27%**	**25.034**	31.877	**-21.47%**
10	**37.108**	48.436	**-23.39%**	**36.872**	49.446	**-25.43%**
15	**49.274**	66.741	**-26.17%**	**48.769**	68.071	**-28.36%**
20	**61.586**	83.183	**-25.96%**	**60.416**	85.007	**-28.93%**
25	**71.510**	100.700	**-28.99%**	**71.064**	101.933	**-30.28%**
30	**83.996**	119.549	**-29.74%**	**86.319**	122.774	**-29.69%**
35	**95.065**	137.100	**-30.66%**	**97.802**	140.606	**-30.44%**
40	**106.007**	156.254	**-32.16%**	**110.345**	157.503	**-29.94%**

Table 9.5: Reduction in time per epoch using fast gradient computation

Correcting the certificates: Since SOC is an approximation, the Lipschitz constant of each SOC layer can be slightly more than 1 and if we use a large number of SOC layers (e.g. 41 in LipConvnet-40), the Lipschitz constant of the full network ($\mathrm{Lip}(f)$) can be significantly larger than 1. To mitigate this issue, we take the following steps: (a) We use a large number of terms ($k = 15$) during test time to approximate the exponential which results in a small approximation error (using the error bound in [243]) and (b) We compute $\mathrm{Lip}(f)$ by multiplying the lipschitz constant of all SOC layers (using the power method) and then divide the certificate by $\mathrm{Lip}(f)$.

9.7.1 Results using Faster gradient computation

We show the reduction in training time per epoch in Table 9.5. In both datasets, we observe that for deeper networks (≥ 25 layers), the reduction in time per epoch is $\approx 30\%$. The corresponding standard and provable robust accuracy numbers are given in Tables 9.3 (CIFAR-10) and 9.4 (CIFAR-100) in the row "**+ Fast**". From the columns "Increase (Standard)" and "Increase (36/255)", we observe that the performance is similar to the baseline across all network architectures. For deeper networks: $\mathrm{LipConvnet}\text{-}35, 40$, we observe an increase in performance

for CIFAR-10 and small decrease ($< 0.64\%$) for CIFAR-100.

9.7.2 Results using projection pooling and CRC-Lip

We observe that using CRC and projection pooling (row "**+ CRC**") leads to significant improvements in performance across all $\mathrm{LipConvnet}$ architectures. On CIFAR-10, we observe significant improvements in both the standard ($\geq 1.63\%$, column "Increase (standard)") and provable robust accuracy ($\geq 3.14\%$, column "Increase (36/255)") across all architectures. On CIFAR-100, we also observe significant improvements in the standard ($\geq 2.49\%$) and provable robust accuracy ($\geq 2.27\%$). For deeper networks ($\mathrm{LipConvnet\text{-}35}, 40$), on CIFAR-10, we observe even more significant gains in the standard ($\geq 5.84\%$) and provable robust accuracy ($\geq 8.76\%$). Similarly on CIFAR-100, we observe gains of $\geq 8.00\%$ (standard) and $\geq 8.65\%$ (provable robust).

Our results establish a new state-of-the-art for both the standard and provable robust accuracy across all attack radii. In Table 9.3 (CIFAR-10), the best "Baseline" standard and provable robust accuracy values (at $36/255, 72/255, 108/255$) are 77.78% and $63.31\%,\ 46.42\%,\ 31.53\%$ respectively. The best "**+ CRC**" values are 79.57% and $67.13\%,\ 53.17\%,\ 38.60\%$. This results in improvements of $+1.79\%$ and $+3.82\%, +6.75\%, +7.07\%$ respectively. Similarly, in Table 9.4 (CIFAR-100), the best "Baseline" standard and provable robust values are 48.06% and $34.52\%, 23.40\%, 14.70\%$. The best '**+ CRC**" values are 51.84% $(+3.78\%)$ and 39.27% $(+4.75\%)$, 27.94% $(+4.54\%)$, 20.29% $(+5.59\%)$.

Supplementary Material

9.A Proofs

9.A.1 Proof of Theorem 21

Since $\mathbf{A}$ is skew symmetric, $\forall\ j, k,\ \ \mathbf{A}_{j,k} = -\mathbf{A}_{k,j}$.

Thus, we focus on computing $\nabla_{\mathbf{A}_{j,k}}\ \ell$ for $j > k$.

9.A.1.1 Preliminaries

Using equation (9.1) in the main text, we can rewrite $\mathbf{z}$ as follows:

$$\mathbf{z} = \mathbf{b} + \sum_{i=0}^{\infty} \frac{\mathbf{z}^{(i)}}{i!}, \quad \text{where } \ \mathbf{z}^{(i)} = \mathbf{A}^i\mathbf{x}$$

Using the chain rule, $\nabla_{\mathbf{A}_{j,k}}\ \ell$ can be computed as follows:

$$\nabla_{\mathbf{A}_{j,k}}\ \ell = \sum_{i=0}^{\infty} \left(\nabla_{\mathbf{A}_{j,k}}\ \mathbf{z}^{(i)}\right)^T \left(\nabla_{\mathbf{z}^{(i)}}\ \ell\right), \qquad \text{where} \quad \nabla_{\mathbf{z}^{(i)}}\ \ell = \frac{\nabla_{\mathbf{z}}\ \ell}{i!}$$

$$\implies \nabla_{\mathbf{A}_{j,k}}\ \ell = \sum_{i=0}^{\infty} \left(\nabla_{\mathbf{A}_{j,k}}\ \mathbf{z}^{(i)}\right)^T \left(\frac{\nabla_{\mathbf{z}}\ \ell}{i!}\right) = \left(\sum_{i=0}^{\infty} \frac{\nabla_{\mathbf{A}_{j,k}}\ \mathbf{z}^{(i)}}{i!}\right)^T \nabla_{\mathbf{z}}\ \ell$$

Moreover, since $\mathbf{z}^{(0)}$ is independent of $\mathbf{W}_{j,k}$, we have: $\nabla_{\mathbf{A}_{j,k}}\ \mathbf{z}^{(0)} = 0$ implying:

$$\nabla_{\mathbf{A}_{j,k}}\ \ell = \left(\sum_{i=1}^{\infty} \frac{\nabla_{\mathbf{A}_{j,k}}\ \mathbf{z}^{(i)}}{i!}\right)^T \nabla_{\mathbf{z}}\ \ell \tag{9.9}$$

To compute each $\nabla_{\mathbf{A}_{j,k}} \mathbf{z}^{(i)}$, consider a matrix $\mathbf{E}$ defined below:

$$\mathbf{E}_{p,q} = \begin{cases} +1 & p = j,\ q = k \\ -1 & p = k,\ q = j \\ 0 & otherwise \end{cases} \tag{9.10}$$

Now, let $\mathbf{e}^{(j)}$ denote a vector that satisfies:

$$(\mathbf{e}^{(j)})_l = \begin{cases} 1 & l = j \\ 0 & otherwise \end{cases} \tag{9.11}$$

Clearly, $\mathbf{E}$ can be rewritten as follows:

$$\mathbf{E} = \left(\mathbf{e}^{(j)}(\mathbf{e}^{(k)})^T - \mathbf{e}^{(k)}(\mathbf{e}^{(j)})^T\right) \tag{9.12}$$

9.A.1.2 Computing $\nabla_{\mathbf{A}_{j,k}} \mathbf{z}^{(i)}$

Since $\mathbf{z}^{(i)} = \mathbf{A}^i \mathbf{x}$, we can compute $\nabla_{\mathbf{A}_{j,k}} \mathbf{z}^{(i)}$ as follows:

$$\begin{aligned} \nabla_{\mathbf{A}_{j,k}} \mathbf{z}^{(i)} &= \lim_{\epsilon \to 0} \frac{(\mathbf{A} + \epsilon \mathbf{E})^i \mathbf{x} - \mathbf{A}^i \mathbf{x}}{\epsilon} = \lim_{\epsilon \to 0} \frac{(\mathbf{A} + \epsilon \mathbf{E})^i - \mathbf{A}^i}{\epsilon} \mathbf{x} \\ &= \lim_{\epsilon \to 0} \frac{\mathbf{A}^i + \epsilon \sum_{l=0}^{i-1} \left(\mathbf{A}^l\right) \mathbf{E} \left(\mathbf{A}^{i-l-1}\right) + \epsilon^2 (\text{higher order terms}) - \mathbf{A}^i}{\epsilon} \mathbf{x} \\ &= \sum_{l=0}^{i-1} \left(\mathbf{A}^l\right) \mathbf{E} \left(\mathbf{A}^{i-l-1}\right) \mathbf{x} \end{aligned} \tag{9.13}$$

9.A.1.3 Rewriting $\nabla_{\mathbf{A}_{j,k}} \ell$

Using equations (9.9) and (9.13), we can simplify $\nabla_{\mathbf{A}_{j,k}} \ell$ as follows:

$$
\begin{aligned}
\nabla_{\mathbf{A}_{j,k}} \ell &= \left(\sum_{i=1}^{\infty} \frac{\nabla_{\mathbf{A}_{j,k}} \mathbf{z}^{(i)}}{i!} \right)^T \nabla_{\mathbf{z}} \ell \\
&= \left(\sum_{i=1}^{\infty} \sum_{l=0}^{i-1} \frac{1}{i!} \left(\mathbf{A}^l \right) \mathbf{E} \left(\mathbf{A}^{i-l-1} \mathbf{x} \right) \right)^T \nabla_{\mathbf{z}} \ell \\
&= \left(\sum_{i=1}^{\infty} \left(\sum_{l=0}^{i-1} \frac{1}{i!} \left(\mathbf{A}^l \right) \mathbf{E} \left(\mathbf{A}^{i-l-1} \mathbf{x} \right) \right)^T \right) \nabla_{\mathbf{z}} \ell \\
&= \left(\sum_{i=1}^{\infty} \sum_{l=0}^{i-1} \frac{1}{i!} \left(\mathbf{A}^{i-l-1} \mathbf{x} \right)^T \mathbf{E}^T \left(\mathbf{A}^l \right)^T \right) \nabla_{\mathbf{z}} \ell \\
&= \left(\sum_{i=1}^{\infty} \sum_{l=0}^{i-1} \frac{1}{i!} \left(\mathbf{A}^{i-l-1} \mathbf{x} \right)^T \mathbf{E}^T \left(\mathbf{A}^T \right)^l \right) \nabla_{\mathbf{z}} \ell \\
&= \left(\sum_{i=1}^{\infty} \sum_{l=0}^{i-1} \frac{1}{i!} \left(\mathbf{A}^{i-l-1} \mathbf{x} \right)^T \mathbf{E}^T \left(-\mathbf{A} \right)^l \right) \nabla_{\mathbf{z}} \ell \\
&= \sum_{i \geq 1,\ l \leq i-1} \left(\frac{1}{i!} \left(\mathbf{A}^{i-l-1} \mathbf{x} \right)^T \mathbf{E}^T \left(\left(-\mathbf{A} \right)^l \nabla_{\mathbf{z}} \ell \right) \right) \\
&= \sum_{l \geq 0,\ i \geq l+1} \left(\frac{1}{i!} \left(\mathbf{A}^{i-l-1} \mathbf{x} \right)^T \mathbf{E}^T \left(\left(-\mathbf{A} \right)^l \nabla_{\mathbf{z}} \ell \right) \right)
\end{aligned}
$$

Rewriting $m = i - l - 1$, we get:

$$
\nabla_{\mathbf{A}_{j,k}} \ell = \sum_{l \geq 0,\ m \geq 0} \left(\frac{1}{(m+l+1)!} \left(\mathbf{A}^m \mathbf{x} \right)^T \mathbf{E}^T \left(\left(-\mathbf{A} \right)^l \nabla_{\mathbf{z}} \ell \right) \right)
$$

Note that the above equation can also be rewritten in the following equivalent forms:

$$\nabla_{\mathbf{A}_{j,k}} = \sum_{l=0}^{\infty} \left(\sum_{m=0}^{\infty} \frac{\mathbf{A}^m \mathbf{x}}{(m+l+1)!} \right)^T \mathbf{E}^T \left((-\mathbf{A})^l \nabla_{\mathbf{z}} \ell \right) \tag{9.14}$$

$$\nabla_{\mathbf{A}_{j,k}} = \sum_{m=0}^{\infty} (\mathbf{A}^m \mathbf{x})^T \mathbf{E}^T \left(\sum_{l=0}^{\infty} \frac{(-\mathbf{A})^l \nabla_{\mathbf{z}} \ell}{(m+l+1)!} \right) \tag{9.15}$$

Let us now define the following quantities:

$$\mathbf{u}^{(l)} = \sum_{m=0}^{\infty} \frac{\mathbf{A}^m \mathbf{x}}{(m+l)!} \tag{9.16}$$

$$\mathbf{v}^{(m)} = \sum_{l=0}^{\infty} \frac{(-\mathbf{A})^l \nabla_{\mathbf{z}} \ell}{(m+l)!} \tag{9.17}$$

Note that they are similar to equations (9.2) and (9.3) in the maintext.

Using equations (9.16) and (9.17), we can rewrite (9.14) and (9.15) as follows:

$$\nabla_{\mathbf{A}_{j,k}} = \sum_{l=1}^{\infty} \left(\mathbf{u}^{(l)} \right)^T \mathbf{E}^T \left((-\mathbf{A})^{l-1} \nabla_{\mathbf{z}} \ell \right) \tag{9.18}$$

$$\nabla_{\mathbf{A}_{j,k}} = \sum_{m=1}^{\infty} \left(\mathbf{A}^{m-1} \mathbf{x} \right)^T \mathbf{E}^T \mathbf{v}^{(m)} \tag{9.19}$$

Using the definition of $\mathbf{E}$ (equation (9.10)), we obtain the following equivalent expressions of $\nabla_{\mathbf{A}_{j,k}} \ell$:

$$\nabla_{\mathbf{A}_{j,k}} \ell = -\sum_{l=1}^{\infty} \left(\mathbf{u}_j^{(l)} \left((-\mathbf{A})^{l-1} \nabla_{\mathbf{z}} \ell \right)_k - \mathbf{u}_k^{(l)} \left((-\mathbf{A})^{l-1} \nabla_{\mathbf{z}} \ell \right)_j \right) \tag{9.20}$$

$$\nabla_{\mathbf{A}_{j,k}} \ell = -\sum_{m=1}^{\infty} \left(\left(\mathbf{A}^{m-1} \mathbf{x} \right)_j \mathbf{v}_k^{(m)} - \left(\mathbf{A}^{m-1} \mathbf{x} \right)_k \mathbf{v}_j^{(m)} \right) \tag{9.21}$$

Using the above equation, we obtain the following equivalent expressions for $\nabla\, \ell$:

$$\nabla_{\mathbf{A}}\, \ell = -\sum_{l=1}^{\infty} \left(\mathbf{u}^{(l)} \left((-\mathbf{A})^{l-1}\, \nabla_{\mathbf{z}}\, \ell \right)^T - \left((-\mathbf{A})^{l-1}\, \nabla_{\mathbf{z}}\, \ell \right) \left(\mathbf{u}^{(l)} \right)^T \right) \tag{9.22}$$

$$\nabla_{\mathbf{A}}\, \ell = -\sum_{m=1}^{\infty} \left(\left(\mathbf{A}^{m-1} \mathbf{x} \right) \left(\mathbf{v}^{(m)} \right)^T - \mathbf{v}^{(m)} \left(\mathbf{A}^{m-1} \mathbf{x} \right)^T \right) \tag{9.23}$$

9.A.2 Proof of Proposition 3

Consider the inequality given in equation (9.6):

$$\min_{i \neq l} \min_{h_l(\mathbf{y}^*) = h_i(\mathbf{y}^*)} \|\mathbf{y}^* - g(\mathbf{x})\|_2 \geq R$$

The above inequality says that for the prediction of h to be different from l for some input $\mathbf{y}^*$, the l_2 distance $\|\mathbf{y}^* - g(\mathbf{x})\|_2$ must be at least R.

We first take the contrapositive of this statement:

$$\|\mathbf{y}^* - g(\mathbf{x})\|_2 \leq R \implies \forall_i\, h_l(\mathbf{y}^*) \geq h_i(\mathbf{y}^*) \tag{9.24}$$

But since g is 1-Lipschitz, we have:

$$\|\mathbf{x}^* - \mathbf{x}\|_2 \leq R \implies \|g(\mathbf{x}^*) - g(\mathbf{x})\|_2 \leq R \tag{9.25}$$

Using both (9.24) and (9.25), we get:

$$\|\mathbf{x}^* - \mathbf{x}\|_2 \leq R \implies \forall_i\, h_l(g(\mathbf{x}^*)) \geq h_i(g(\mathbf{x}^*)) \implies \forall_i\, f_l(\mathbf{x}^*) \geq f_i(\mathbf{x}^*)$$

Thus, in an l_2 ball of radius R, prediction l remains unchanged and the function is provably robust.

9.A.3 Proof of Theorem 22

Let $\mathbf{x}^*,\ \mathbf{y}^* \in \mathcal{M}$ so that $d_{\mathcal{M}}(\mathbf{x})$ and $d_{\mathcal{M}}(\mathbf{y})$ satisfy:

$$d_{\mathcal{M}}(\mathbf{x}) = \|\mathbf{x}^* - \mathbf{x}\|_2 \qquad d_{\mathcal{M}}(\mathbf{y}) = \|\mathbf{y}^* - \mathbf{y}\|_2 \tag{9.26}$$

Since $\mathbf{x}^*$ and $\mathbf{y}^*$ minimize the l_2 distance of $\mathbf{x}$ and $\mathbf{y}$ respectively to $\mathcal{M}$, we have:

$$\|\mathbf{x}^* - \mathbf{x}\|_2 \leq \|\mathbf{y}^* - \mathbf{x}\|_2, \qquad \|\mathbf{y}^* - \mathbf{y}\|_2 \leq \|\mathbf{x}^* - \mathbf{y}\|_2 \tag{9.27}$$

Using the triangle inequality for the expressions $\|\mathbf{y}^* - \mathbf{x}\|_2$ and $\|\mathbf{x}^* - \mathbf{y}\|_2$, we have:

$$\|\mathbf{y}^* - \mathbf{x}\|_2 \leq \|\mathbf{y}^* - \mathbf{y}\|_2 + \|\mathbf{y} - \mathbf{x}\|_2, \qquad \|\mathbf{x}^* - \mathbf{y}\|_2 \leq \|\mathbf{x}^* - \mathbf{x}\|_2 + \|\mathbf{x} - \mathbf{y}\|_2 \tag{9.28}$$

Using inequalities (9.27) and (9.28), we get:

$$\|\mathbf{x}^* - \mathbf{x}\|_2 \leq \|\mathbf{y}^* - \mathbf{y}\|_2 + \|\mathbf{y} - \mathbf{x}\|_2, \qquad \|\mathbf{y}^* - \mathbf{y}\|_2 \leq \|\mathbf{x}^* - \mathbf{x}\|_2 + \|\mathbf{x} - \mathbf{y}\|_2 \tag{9.29}$$

But then using equation (9.26) and inequality (9.29), we get:

$$d_{\mathcal{M}}(\mathbf{x}) \leq d_{\mathcal{M}}(\mathbf{y}) + \|\mathbf{y}-\mathbf{x}\|_2, \qquad d_{\mathcal{M}}(\mathbf{y}) \leq d_{\mathcal{M}}(\mathbf{x}) + \|\mathbf{x}-\mathbf{y}\|_2$$

$$d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y}) \leq \|\mathbf{y}-\mathbf{x}\|_2, \qquad d_{\mathcal{M}}(\mathbf{y}) - d_{\mathcal{M}}(\mathbf{x}) \leq \|\mathbf{x}-\mathbf{y}\|_2$$

$$\implies |d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y})| \leq \|\mathbf{x}-\mathbf{y}\|_2$$

9.A.4 Proof of Corollary 3

We define a function $d_{\mathcal{M}}(\mathbf{x})$ that is (without loss of generality) positive in $\mathcal{A}$ and negative in $\mathcal{B}$:

$$d_{\mathcal{M}}(\mathbf{x}) = \begin{cases} \mathbf{x} \in \mathcal{A} & + \min_{\mathbf{x}^* \in \mathcal{M}} \|\mathbf{x}^* - \mathbf{x}\|_2 \\ \mathbf{x} \in \mathcal{B} & - \min_{\mathbf{x}^* \in \mathcal{M}} \|\mathbf{x}^* - \mathbf{x}\|_2 \end{cases}$$

Using Theorem 22, it is easy to see that within each set $\mathcal{A}$ and $\mathcal{B}$, $d_{\mathcal{M}}$ is 1-Lipschitz and continuous.

Similarly, because every line segment connecting $\mathcal{A}$ and $\mathcal{B}$ crosses through some point on the $\mathcal{M}$, it is evident that $d_{\mathcal{M}}$ is continuous on each path connecting $\mathcal{A}$ to $\mathcal{B}$.

So, we next prove that $d_{\mathcal{M}}$ is 1-Lipschitz if $\mathbf{x} \in \mathcal{A}$ and $\mathbf{y} \in \mathcal{B}$ i.e.:

$$|d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y})| \leq \|\mathbf{x}-\mathbf{y}\|_2$$

Let $\mathbf{p} \in \mathcal{M}$ be a point on the line segment connecting $\mathbf{x}$ and $\mathbf{y}$ such that:

$$\|\mathbf{x}-\mathbf{y}\|_2 = \|\mathbf{x}-\mathbf{p}\|_2 + \|\mathbf{p}-\mathbf{y}\|_2 \tag{9.30}$$

But then by the definitions of $d_{\mathcal{M}}(\mathbf{x})$ and $d_{\mathcal{M}}(\mathbf{y})$:

$$|d_{\mathcal{M}}(\mathbf{x})| \leq \|\mathbf{x} - \mathbf{p}\|_2, \qquad |d_{\mathcal{M}}(\mathbf{y})| \leq \|\mathbf{p} - \mathbf{y}\|_2 \tag{9.31}$$

Using equation (9.30) and inequality (9.31), we get:

$$\begin{aligned} &|d_{\mathcal{M}}(\mathbf{x})| + |d_{\mathcal{M}}(\mathbf{y})| \leq \|\mathbf{x} - \mathbf{p}\|_2 + \|\mathbf{p} - \mathbf{y}\|_2 = \|\mathbf{x} - \mathbf{y}\|_2 \\ \implies &|d_{\mathcal{M}}(\mathbf{x})| + |d_{\mathcal{M}}(\mathbf{y})| \leq \|\mathbf{x} - \mathbf{y}\|_2 \end{aligned} \tag{9.32}$$

Since $d_{\mathcal{M}}(\mathbf{x})$ and $d_{\mathcal{M}}(\mathbf{y})$ have opposite signs, we have:

$$|d_{\mathcal{M}}(\mathbf{x})| + |d_{\mathcal{M}}(\mathbf{y})| = |d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y})| \tag{9.33}$$

Using inequality (9.32) and equation (9.33), we prove:

$$|d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y})| \leq \|\mathbf{x} - \mathbf{y}\|_2$$

Hence, $|d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y})|$ is 1-Lipschitz.

9.B Additional analysis

9.B.1 Approximating the convolution weight gradient

9.B.1.1 Simplifying the expression for $\nabla_{\mathbf{A}_{j,k}} \ell$

Recall the expression for $\nabla_{\mathbf{A}_{j,k}} \ell$ using equations (9.9) and (9.13):

$$\nabla_{\mathbf{A}_{j,k}} \ell = \left(\sum_{i=1}^{\infty} \frac{\nabla_{\mathbf{A}_{j,k}} \mathbf{z}^{(i)}}{i!} \right)^T \nabla_{\mathbf{z}} \ell, \qquad \text{where } \nabla_{\mathbf{A}_{j,k}} \mathbf{z}^{(i)} = \sum_{l=0}^{i-1} \left(\mathbf{A}^l\right) \mathbf{E} \left(\mathbf{A}^{i-l-1}\right) \mathbf{x}$$

We replace $\nabla_{\mathbf{A}_{j,k}} \mathbf{z}^{(i)}$ in $\nabla_{\mathbf{A}_{j,k}} \ell$ and take $\mathbf{x}$ out of the transpose:

$$\begin{aligned}
\nabla_{\mathbf{A}_{j,k}} \ell &= \mathbf{x}^T \left(\sum_{i=1}^{\infty} \frac{1}{i!} \sum_{l=0}^{i-1} \left(\mathbf{A}^l\right) \mathbf{E} \left(\mathbf{A}^{i-l-1}\right) \right)^T \nabla_{\mathbf{z}} \ell \\
&= \mathbf{x}^T \left(\mathbf{E} + \frac{1}{2!} \left(\mathbf{EA} + \mathbf{AE}\right) + \frac{1}{3!} \left(\mathbf{EA}^2 + \mathbf{AEA} + \mathbf{A}^2\mathbf{E}\right) + \dots \right)^T \nabla_{\mathbf{z}} \ell
\end{aligned}$$

We simplify the above as follows:

$$\nabla_{\mathbf{A}_{j,k}} \ell \tag{9.34}$$

$$= \mathbf{x}^T \left(\mathbf{E} + \frac{1}{2!} \left(\mathbf{EA} + \mathbf{AE}\right) + \frac{1}{3!} \left(\mathbf{EA}^2 + \mathbf{A}^2\mathbf{E}\right) + \frac{1}{4!} \left(\mathbf{EA}^3 + \mathbf{A}^3\mathbf{E}\right) + \dots \right)^T \nabla_{\mathbf{z}} \ell \tag{9.35}$$

$$+ \mathbf{x}^T \mathbf{R}^T \left(\nabla_{\mathbf{z}} \ell\right) \tag{9.36}$$

where $\mathbf{R}$ is defined as follows:

$$\mathbf{R} = \frac{1}{3!} \left(\mathbf{AEA}\right) + \frac{1}{4!} \left(\mathbf{AEA}^2 + \mathbf{A}^2\mathbf{EA}\right) + \frac{1}{5!} \left(\mathbf{AEA}^3 + \mathbf{A}^2\mathbf{EA}^2 + \mathbf{A}^3\mathbf{EA}\right) + \dots \tag{9.37}$$

In the above simplification, $\mathbf{R}$ contains all terms of the form $\mathbf{A}^l\mathbf{E}\mathbf{A}^m$ where both $l \geq 1$ and $m \geq 1$.

9.B.1.2 Simplifying the expression for $\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T\mathbf{v}^{(1)}$

Recall that $\mathbf{u}^{(1)}, \mathbf{v}^{(1)}$ are given by (equations (9.16) and (9.17)):

$$\mathbf{u}^{(1)} = \left(\sum_{l=1}^{\infty} \frac{\mathbf{A}^{l-1}}{l!}\right)\mathbf{x}, \qquad \mathbf{v}^{(1)} = \left(\sum_{m=1}^{\infty} \frac{(-\mathbf{A})^{m-1}}{m!}\right)\nabla_{\mathbf{z}}\,\ell \tag{9.38}$$

We first replace $\mathbf{v}^{(1)}$ using the above in $\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T\mathbf{v}^{(1)}$:

$$\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T\mathbf{v}^{(1)} = \left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \left(\sum_{m=1}^{\infty} \frac{(-\mathbf{A})^{m-1}}{m!}\right)\nabla_{\mathbf{z}}\,\ell \tag{9.39}$$

Given 2 matrices $\mathbf{B}, \mathbf{C}$, we have $\left(\mathbf{B}^T\right)^i = (\mathbf{B}^i)^T$ and $(\mathbf{B}+\mathbf{C})^T = \mathbf{B}^T + \mathbf{C}^T$. Thus:

$$\left(\sum_{m=1}^{\infty} \frac{\mathbf{A}^{m-1}}{m!}\right)^T = \left(\sum_{m=1}^{\infty} \frac{(-\mathbf{A})^{m-1}}{m!}\right)$$

Replacing the above in equation (9.39), we get:

$$\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T\mathbf{v}^{(1)} = \left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \left(\sum_{m=1}^{\infty} \frac{\mathbf{A}^{m-1}}{m!}\right)^T \nabla_{\mathbf{z}}\,\ell$$

Given 3 matrices, $\mathbf{B}, \mathbf{C}, \mathbf{D}$, we have $\mathbf{D}^T\mathbf{C}^T\mathbf{B}^T = (\mathbf{B}\mathbf{C}\mathbf{D})^T$. Thus:

$$\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T\mathbf{v}^{(1)} = \left(\left(\sum_{m=1}^{\infty} \frac{\mathbf{A}^{m-1}}{m!}\right)\mathbf{E}\mathbf{u}^{(1)}\right)^T \nabla_{\mathbf{z}}\,\ell$$

Now, we replace $\mathbf{u}^{(1)}$ using equation (9.38) and take $\mathbf{x}$ out of the transpose:

$$\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \mathbf{v}^{(1)} = \mathbf{x}^T \left(\left(\sum_{m=1}^{\infty} \frac{\mathbf{A}^{m-1}}{m!} \right) \mathbf{E} \left(\sum_{l=1}^{\infty} \frac{\mathbf{A}^{l-1}}{l!} \right) \right)^T \nabla_{\mathbf{z}} \ell$$

Expanding the two series, we get:

$$\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \mathbf{v}^{(1)} = \mathbf{x}^T \left(\left(\mathbf{I} + \frac{\mathbf{A}}{2!} + \frac{\mathbf{A}^2}{3!} + \dots \right) \mathbf{E} \left(\mathbf{I} + \frac{\mathbf{A}}{2!} + \frac{\mathbf{A}^2}{3!} + \dots \right) \right)^T \nabla_{\mathbf{z}} \ell$$

We simplify the above as follows:

$$\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \mathbf{v}^{(1)} \tag{9.40}$$

$$= \mathbf{x}^T \left(\mathbf{E} + \frac{1}{2!} \left(\mathbf{EA} + \mathbf{AE} \right) + \frac{1}{3!} \left(\mathbf{EA}^2 + \mathbf{A}^2\mathbf{E} \right) + \frac{1}{4!} \left(\mathbf{EA}^3 + \mathbf{A}^3\mathbf{E} \right) + \dots \right)^T \nabla_{\mathbf{z}} \ell \tag{9.41}$$

$$+ \mathbf{x}^T \mathbf{S}^T \left(\nabla_{\mathbf{z}} \ell \right) \tag{9.42}$$

where $\mathbf{S}$ is defined as follows:

$$\mathbf{S} = \left(\frac{\mathbf{AEA}}{2!2!} \right) + \left(\frac{\mathbf{AEA}^2}{2!3!} + \frac{\mathbf{A}^2\mathbf{EA}}{3!2!} \right) + \left(\frac{\mathbf{AEA}^3}{2!4!} + \frac{\mathbf{A}^2\mathbf{EA}^2}{3!3!} + \frac{\mathbf{A}^3\mathbf{EA}}{4!2!} \right) + \dots \tag{9.43}$$

9.B.1.3 Comparing the two expressions

Comparing equations (9.34) and (9.40), the first part in the two are same.

The first part are the equations (9.35) and (9.41) respectively.

The two expressions only differ in the terms $\mathbf{x}^T\mathbf{R}^T \left(\nabla_{\mathbf{z}} \ell \right)$ and $\mathbf{x}^T\mathbf{S}^T \left(\nabla_{\mathbf{z}} \ell \right)$.

Note that in $\mathbf{R}$ (equation (9.37)), the term $\mathbf{A}^l\mathbf{EA}^m$ is multiplied by $1/(l+m+1)!$.

And in $\mathbf{S}$ (equation (9.43)), the term $\mathbf{A}^l\mathbf{E}\mathbf{A}^m$ is multiplied by $1/(l+1)!(m+1)!$.

Thus, since the quantities $\mathbf{R}$ and $\mathbf{S}$ are multiplied by factorial in the denominator.

For higher values of l or m, both the expressions converge to 0.

Moreover, $\mathbf{S}-\mathbf{R}$ for smaller values of l, m can be directly computed using equations (9.37) and (9.43):

$$\mathbf{S}-\mathbf{R}=\left(\frac{\mathbf{AEA}}{12}\right)+\left(\frac{\mathbf{AEA}^2}{24}+\frac{\mathbf{A}^2\mathbf{EA}}{24}\right)+\left(\frac{\mathbf{AEA}^3}{80}+\frac{\mathbf{A}^2\mathbf{EA}^2}{360/7}+\frac{\mathbf{A}^3\mathbf{EA}}{80}\right)+\dots \tag{9.44}$$

We can see that the denominator in the above terms are already very small even for $l+m=4$.

Thus, we simply assume $\mathbf{S}-\mathbf{R}\approx\mathbf{0}$.

This results in the following approximate expression for $\nabla_{\mathbf{A}_{j,k}}\ell$:

$$\nabla_{\mathbf{A}_{j,k}}\ell\approx\left(\mathbf{u}^{(1)}\right)^T\mathbf{E}^T\mathbf{v}^{(1)} \tag{9.45}$$

Using the definition of $\mathbf{E}$ (equation (9.10)), we obtain the following expression of $\nabla_{\mathbf{A}_{j,k}}\ell$:

$$\nabla_{\mathbf{A}_{j,k}}\ell\approx-(\mathbf{u}_j^{(1)}\mathbf{v}_k^{(1)}-\mathbf{u}_k^{(1)}\mathbf{v}_j^{(1)}) \tag{9.46}$$

Using equation (9.46), the gradient with respect to matrix $\mathbf{A}$ can be computed as follows:

$$\nabla_{\mathbf{A}}\ell\approx-\left(\mathbf{u}^{(1)}\left(\mathbf{v}^{(1)}\right)^T-\mathbf{v}^{(1)}\left(\mathbf{u}^{(1)}\right)^T\right) \tag{9.47}$$

9.B.2 Iterations for $\mathbf{z}$ and $\nabla_{\mathbf{z}}\,\ell$

Recall that we use the following iterations to compute $\mathbf{z}$ during the forward pass:

$$\mathbf{u}^{(i)} = \begin{cases} \mathbf{x} & i = k-1 \\ \mathbf{x} + \left(\mathbf{A}\mathbf{u}^{(i+1)}\right)/(i+1) & i \leq k-2 \end{cases}$$

We have to prove that $\forall\ k,\ \mathbf{u}^{(0)} = \mathbf{z}$ where:

$$\mathbf{z} = \sum_{i=0}^{k-1} \frac{\mathbf{A}^i\mathbf{x}}{i!}$$

To prove the same, we use induction over k.

For $k = 1$, $\mathbf{u}^{(0)} = \mathbf{z} = \mathbf{x}$ by definition.

For $k \geq 2$, we assume that the statement is true for $j \leq k-1$ and try to prove for k.

For k, we have the following expressions for $\mathbf{u}^{(k-1)}$ and $\mathbf{u}^{(k-2)}$:

$$\mathbf{u}^{(k-1)} = \mathbf{x}, \quad \mathbf{u}^{(k-2)} = \mathbf{x} + \mathbf{A}\mathbf{x}/(k-1)$$

Next, we apply the iteration on $\mathbf{u}^{(k-2)}$ till $\mathbf{u}^{(0)}$.

By induction, the first term i.e. $\mathbf{x}$ simply results in $\sum_{i=0}^{k-2}(\mathbf{A}^i\mathbf{x})/i!$.

The second term i.e. $\mathbf{A}\mathbf{x}/k$ in $(\mathbf{A}^{k-1}\mathbf{x})/(k-1)!$. Thus, we have:

$$\mathbf{u}^{(0)} = \left(\sum_{i=0}^{k-2} \frac{\mathbf{A}^i\mathbf{x}}{i!}\right) + \frac{\mathbf{A}^{k-1}\mathbf{x}}{(k-1)!} = \sum_{i=0}^{k-1} \frac{\mathbf{A}^i\mathbf{x}}{i!}$$

The same can proof follows for $\nabla_{\mathbf{z}}\, \ell$ except that we replace $\mathbf{A}$ with $-\mathbf{A}$.

Part IV

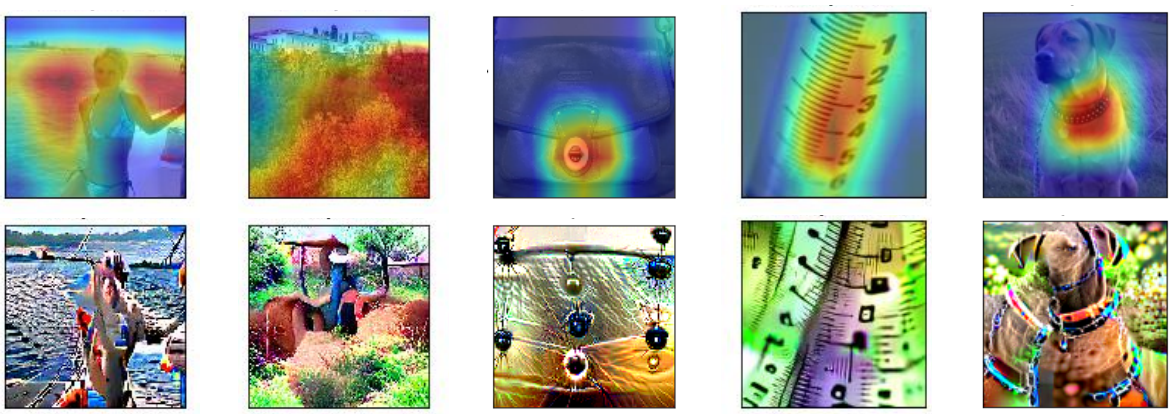
Understanding and mitigating model failures

Chapter 10: Understanding Failures of Deep Networks via Robust Feature Extraction

10.1 Introduction

It is critically important to understand the failure modes of machine learning (ML) systems, especially when they are employed in high-stakes applications. Aggregrate metrics in common use capture summary statistics on failure. While reporting overall performance is important, gaining an understanding of the specifics of failure is a core responsibility in the fielding of ML systems and components. For example, we need to understand situations where a self-driving car will fail to detect a pedestrian even when the system has high overall accuracy. Similarly, it is important to understand for which features misdiagnosis is most probable in chest x-rays even if the model has higher overall accuracy than humans. Such situation-specific insights can guide the iterative process of model development and debugging.

Model performance can be wildly non-uniform for different clusterings of instances and such heterogeneity is not reflected by standard metrics such as AUC or accuracy. For example, it was shown in [272] that a commercial model for emotion detection from facial expressions systematically failed for young children. Buolamwini et al.[273] found that gender detection in multiple commercial models had significantly higher error rates for women with darker skin



(a) class: maillot
feature: water
error: **+9.72%**

(b) class: monastery
feature: greenery
error: **+24.84%**

(c) class: purse
feature: buckle
error: **+10.94%**

(d) class: syringe
feature: markings
error: **+14.99%**

(e) class: rhodesian
feature: dog collar
error: **+10.91%**

Figure 10.1: Failure modes discovered using the proposed methodology for a standard Resnet-50 neural network trained on ImageNet. In the top row, red denotes the region that a specific feature is paying attention to. In the bottom row, we show the image generated by visually amplifying the same feature. We observe that, due to the presence of spurious features, the failure rate of the model increases significantly on the relevant class. Additional examples in Supp. Section 10.D.

tone. These examples highlight the importance of identifying natural clusters in the data with high failure rates. However, practical problems with these approaches still remain: (a) they require an expensive and time-consuming collection of metadata by humans, and (b) visual attributes that machine learning procedures pay attention to can be very different from the ones humans focus on (see Supp. Section 10.D).

To resolve these issues, we propose to leverage the internal representation of a robust model [20] to generate the metadata. The key property that makes robust representations useful is that the features can be visualized more easily than for a standard model [229, 274]. Our method, named Barlow[1], is inspired from Fault Tree Analysis [276] in safety engineering and uses robust representations as a building block.[2] We demonstrate the results on the ImageNet dataset [56] and find that it reveals two types of failures:

[1]In honor of perceptual psychologist, Horace Barlow [275].
[2]Code repository: https://github.com/singlasahil14/barlow

Spurious correlations: A spurious correlation is a feature that is causally unrelated to the desired class but is likely to co-occur with the same class in the training/test data. For example, food is likely to co-occur with plates. However, the absence of the food from a plate image should not result in misclassification (see examples in Figures 10.1a, 10.1b and 10.1e). A more formal discussion about spurious features is presented in Supp. Section 10.C.

Overemphasized features: An overemphasized feature is a feature that is causally related to the desired class but where the model gives *excessive* importance for classification, disregarding the other relevant features, and is unable to make a correct prediction when that feature is absent from the image. For example, a model may be likely to fail on a purse image if the buckle is absent (see Figures 10.1c and 10.1d).

While determining the type of failure (spurious vs. causal but overemphasized feature) and formulating mitigation steps remains a task that depends on human expertise, Barlow assists practitioners in this process by efficiently identifying and providing visualizations of failure modes, showing how input characteristics *correlate* with failures. For example, failures identified in Figure 10.1 suggest the following interventions: (a) add images of maillot/monastery with diverse backgrounds, Rhodesian Ridgebacks without collar (b) mask overemphasized features (buckle from purse, markings from syringe) in the training set.

In summary, we provide the following contributions:

1. An error analysis framework for discovering critical failure modes for a given model.
2. A feature extraction and visualization method based on robust model representations to enable humans to understand the semantics of a learned feature.
3. A large-scale crowdsourcing study to evaluate the effectiveness of the visualization technique

and the interpretability of robust feature representations.

4. A user study with engineers with experience using machine learning for vision tasks to evaluate the effectiveness of the methodology for model debugging.

10.2 Related work

Feature Visualization, Interpretability, and Robustness: For a trained neural network, feature visualization creates images to either (i) visualize the region that neurons are paying attention to using heatmaps, or (ii) maximize the activation of neurons that we are interested in and visualize the resulting image. Previous works [65, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287] provide evidence that the internal representations of a neural network can capture important semantic concepts. For example, the network dissection method [287, 288] visualizes hidden network units by showing regions that are most activated for that unit and correlates the activation maps with dense human-labeled concepts. In our studies, we observe that activation maps alone are not sufficient for narrowing down the feature concept and that methods for maximizing neuron activation produce noisy visualizations with changes not visible to the human eye (see Supp. Section 10.F). Similar limitations are also discussed by Olah et al.[289] in the context of understanding the interpretability of feature visualization methods for standard models. Recent work [20, 229, 290] shows that saliency maps are qualitatively more interpretable for adversarially trained models (when compared to standard training) and align well with human perception. Moreover, Engstrom et al.[274] showed that, for robust models (in contrast to standard models), optimizing an image directly to maximize a certain neuron helps with visualizing the respective relevant learned features. We note that disentangled representations

[291, 292, 293, 294, 295, 296] based on VAEs [297] can also be used for feature extraction. However, it is difficult to scale these methods to large-scale, rich datasets such as ImageNet.

Failure explanation: In recent years, there has been increasing interest in the understandability of predictions made by machine-learned models [65, 66, 68, 283, 298]. Most of these efforts have focused on *local* explanations, where decisions about single images are inspected [274, 283, 299, 300, 301, 302, 303, 304]. Examining numerous local explanations for debugging can however be time-consuming. Thus, we focus on identifying major failure modes across the entire dataset and presenting them in a useful way to guide model improvement by actions as fixing the data distribution and performing data augmentation. For this, we build upon prior work that discovers generalizable failure modes based on metadata or features [305, 306, 307, 308]. These methods operate typically on tabular data where features are already available [306, 307], on language data with query-definable text operators [308], or on image data where features are collected from intermediate multi-component output and crowdsourcing [305]. In this work, we focus on identifying failure modes for images but with features extracted in an automated manner from learned robust representations.

10.3 Background and method overview

Let $h : \mathbf{x} \rightarrow y$ be a trained neural network that classifies an input image x to one of the classes $y \in Y$. For a cluster of images C in an overall benchmark S (i.e., $C \subset S$), we use the following definitions to quantify failures:

Definition 10. *The error rate of a cluster is the fraction of images in the cluster that are misclas-*

sified:

$$\mathrm{ER}(C) = \frac{\sum_{\mathbf{x} \in C} \mathbb{1}_{h(\mathbf{x}) \neq y}(\mathbf{x})}{|C|}$$

Definition 11. *The error coverage of a cluster is the fraction of all errors in the benchmark that fall in the cluster:*

$$\mathrm{EC}(C) = \frac{\sum_{\mathbf{x} \in C} \mathbb{1}_{h(\mathbf{x}) \neq y}(\mathbf{x})}{\sum_{\mathbf{x} \in S} \mathbb{1}_{h(\mathbf{x}) \neq y}(\mathbf{x})}$$

Definition 12. *The base error rate is the error rate for the whole benchmark, treating it as one cluster:*

$$\mathrm{BER} = \mathrm{ER}(S)$$

We seek to describe failures in a benchmark S with low-dimensional rules using a reduced set of human-interpretable features. For example, for an image recognition model that detects traffic lights, we want to generate failure explanations, such as *"The error rate for detecting traffic lights is 20% higher when an image is captured in rainy weather and low light."*. The failure explanation in this case would be `weather=rainy` $\wedge$ `light_intensity=low`. Such rules slice the data into clusters for which we can report metrics as in Definitions 10 and 11. Ideally, we would like to find clusters that jointly have large error rates and that cover a significant fraction of all errors in the benchmark. These criteria ensure that the explanatory rules will be of sufficient importance and generality. Note that the purpose of these explanations is not to predict failure but rather to provide actionable guidance to engineers via a small set of rules about failures and

their indicators. Figure 10.2 depicts the end-to-end Barlow workflow.

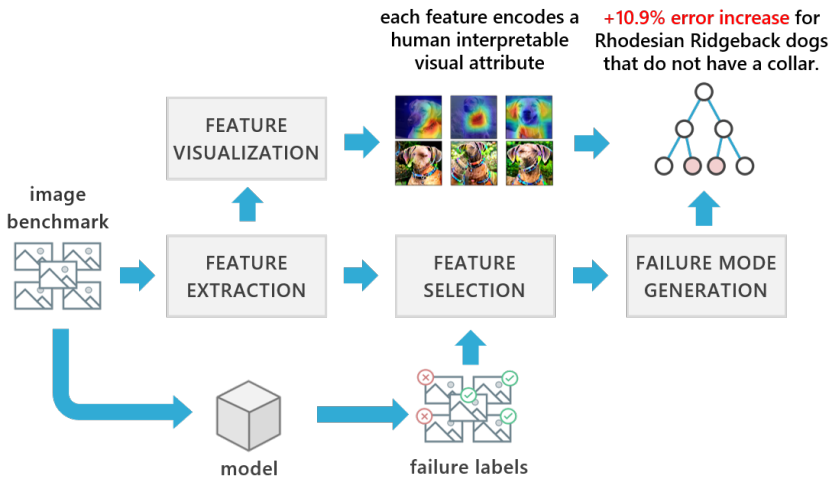


Figure 10.2: Barlow error analysis workflow. *Feature extraction* and *visualization* are described in Section 10.4. *Feature selection* and *failure mode generation* in Section 10.5.

10.4 Feature extraction and visualization

For each image in the set S, we first construct feature representation F as a vector such that each element of the vector encodes some human-interpretable visual attribute. For this purpose, we use the penultimate layer (i.e., the layer adjacent to the logits layer) of a pretrained robust neural network to extract this feature vector. The robust model is trained via objective (10.1) for l_2 robustness.

$$\min_{\theta} \quad \mathbb{E}_{(x,y)\sim\mathbb{D}} \left[\max_{\|x'-x\|_2 \leq \rho} \ell(h_\theta(x'), y) \right] \tag{10.1}$$

Tsipras et al.[229] show that for robust models, saliency maps are more interpretable than standard models and align well with human perception. Engstrom et al.[274] show that direct maximization of neurons of robust models is sufficient to generate recognizable visual attributes.

Such output stands in contrast to the opacity of the visualizations of features generated from standard models for which, even with regularization, visualized features are rarely human interpretable [289] (examples in Supp. Section 10.F). In Sections 10.8 and 10.9, we validate the earlier findings about human interpretability via performing studies with human subjects.

Given the findings on the interpretability of robust models, for feature extraction, we use an adversarially trained Resnet-50 model [1] pretrained on ImageNet (using $\rho = 3$). We note that we can perform feature extraction using a robust model even when the model under inspection is not robust. In this case, we can consider the extracted features as attributes of the data and not necessarily as part of the representation employed within the model.

We next describe our methodology for visualizing the neuron in the representation layer of a robust model:

Most activating images: A common approach to visualizing a neuron's sensitivity is to search through the given set of images to find the top-k instances that maximally activate the desired neuron. A challenge with this approach is that it does not identify the specific attributes of images that are responsible for the activation [289]. For example, consider the images in the top row of Figure 10.3 where it is not clear whether the neuron is focused on sky, water or ground.

Heatmaps: To address the aforementioned problem, we propose to use heatmaps based on CAM [282] (second row in Figure 10.3), as an additional signal for visualization. For a Resnet-50 model, the representation layer is computed via a global average pooling operation to the tensor in the previous layer. We use the index of the desired neuron to retrieve the slice of the tensor from the previous layer at the same index (we refer to this as the *feature map*). Next, we normalize the feature map between 0 and 1 and resize it to match the input size. Details are in Supp. Section 11.A.2. Figure 10.3 shows how heatmaps can help resolve the ambiguity.

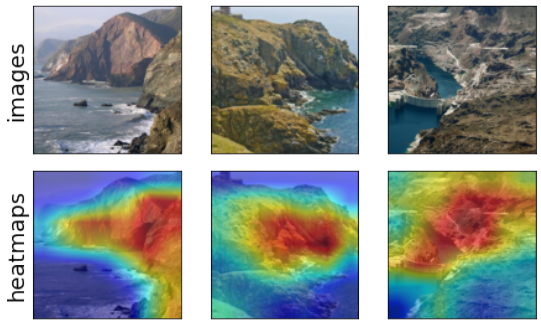


Figure 10.3: Illustration of heatmaps. From the most activating images (top row) for a neuron, it is not clear if the focus is on sky, water, or ground. Heatmaps (bottom row) resolve the ambiguity. The neuron appears to focus on ground.

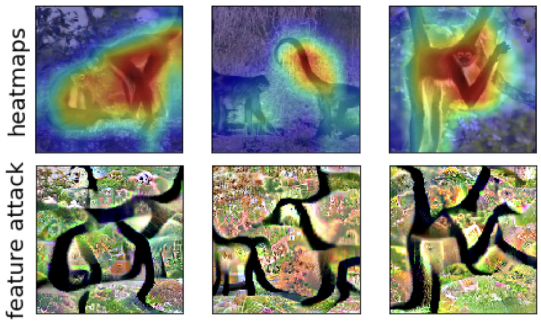


Figure 10.4: Illustration of feature attack. Heatmaps focus on different attributes in different images: body, tail, and face (left to right). Feature Attack (bottom row) resolves the ambiguity. The neuron appears to focus on the limbs and tail.

Feature Attack: In some cases, heatmaps may not be sufficient to resolve ambiguity. For example, in Figure 10.4, it is not clear from the heatmaps whether the neuron is focused on the body, tail, or face of the monkey. Hence, we propose to maximize the neuron in the representation layer with respect to the original image (based on Engstrom et al.[274]). We observe that the limbs and tail (on top of green background) are amplified and this resolves the ambiguity. We call the resulting visualization, a *feature attack*. More details are provided in Supp. Section 11.A.3.

We use the top-6 most activating images, corresponding heatmaps, and feature attack images to generate visualizations of the desired feature. We showcase the importance of these visualizations for interpretability via a study detailed in Section 10.8.

10.5 Failure mode generation

We start with a set of features extracted from images using a robust model. Then, we explore the failure modes of a neural network (not necessarily the robust model) identified via consideration of the ground-truth labels. Our goal is to identify clusters with high error rates. In pursuit of

failure analyses that can be understood with ease, we generate decision trees trained to predict failures (see Figure 10.2), determined by checking whether the model prediction is equal to the image label.

We note that there are challenges with relying on decision trees for explanations. Large numbers of features can make finding a good split difficult due to the curse of dimensionality. From a usability perspective, failures for different classes can vary greatly and practitioners may be interested in understanding them separately [305]. Moreover, psychological studies provide evidence that people can hold in memory and understand only a limited number of chunks of information at the same time [309, 310, 311]. These challenges motivate designs for Barlow and other human-in-the-loop methodologies that avoid complex representations, such as large trees. Thus, we take the following steps:

Class-based failure explanations: We focus analyses and results on two types of class-based groupings: (i) *prediction groupings* and (ii) *label groupings*. For example, *prediction groupings* for the class "goldfish", contain all images for which the model under inspection predicts "goldfish". A failure analysis for this group enables us to understand false positives. *Label groupings* for the class "goldfish", contain all images for which the ground truth in ImageNet is "goldfish" regardless of the model's prediction. A failure analysis for this class provides insights about false negatives.

Feature selection: To reduce dimensionality, we compute the mutual information between each feature and failure labels in the group, and select the top-20 features, sorted by mutual information values. Mutual information estimates the information gained via the feature on the failure label variable in addition to its prior (i.e., the base error rate). Using mutual information for feature selection aligns with our goal of building decision trees with the features, as the same metric is

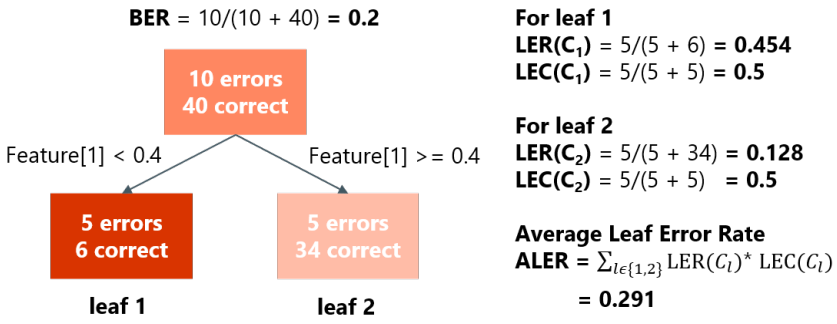


Figure 10.5: Illustration of ALER metric. For both leaf nodes, precision and recall are 0 since $\mathrm{ER} < 0.5$ however leaf 1 is more important for failure mode discovery.

used to estimate the value of splits in the tree.

Truncation and rule generation: All trees are truncated to a low depth (≤ 3). We choose to explain failure modes by using paths in the tree as summary rules that have an error coverage higher than a given threshold τ and an error rate of at least $\mathrm{BER} + \delta$. Both δ and τ are adjustable. This procedure is described in Supp. Algorithm 6.

10.5.1 Automatic evaluation of decision tree

Traditional metrics for evaluating the quality of a decision tree for a classification problem, such as accuracy, precision, and recall, are insufficient when the model is used for description and explanation. Ideally we would want to consider failure modes that include all possible failures in a benchmark. This goal is challenging because of (i) incompleteness in the feature set, (ii) difficulties in finding failure modes that generalize well for many examples at the same time, and, most importantly, (iii) certain failures may happen very rarely in the benchmark. Therefore, we focus on the explanatory properties of failure analyses.

For a given leaf node l, we can use definitions 10 and 11 to define its *leaf error rate* $\mathrm{ER}(C_l)$, and *leaf error coverage* $\mathrm{EC}(C_l)$. C_l denotes the cluster of data that falls into leaf l. For a decision

tree T, we then compute the following metric as the *average leaf error rate* (ALER).

Definition 13. ALER *of a tree T is the average error rate across all leaves weighed by the respective error coverage.*

$$\text{ALER}(T) = \sum_{l \in \ leaves(T)} \text{ER}(C_l) \times \text{EC}(C_l)$$

Per the definition of the leaf error coverage, we have $\sum_{l \in \ leaves(T)} \text{ER}(C_l) = 1$ because all failure instances will be covered by exactly one leaf.

As an example, consider the simple 1-depth decision tree given in Figure 10.5. Since the leaf error rate is less than 0.5 for both leaf nodes, tree precision and tree recall are both zero. However, since the error rate of leaf 1 is significantly higher than the base error rate ($0.454 >> 0.2$), leaf 1 is important as a failure mode description (and more predictive than leaf 2), which is exactly the notion we seek to capture for describing high concentrations of error.

The key property that makes ALER useful is that if it is equal to some quantity q, then there is at least one leaf in the decision tree with leaf error rate greater than q:

Proposition 4. *For a tree T with $\text{ALER}(T) = q$, there exists at least one leaf l, with leaf error rate $\text{ER}(C_l) \geq q$.*

The proposition follows from the fact that all weights (i.e., leaf error coverage) are less than or equal to 1. For the decision tree in Figure 10.5, $\text{BER} = 0.291$, which is greater than the base error rate 0.2 by a margin of 0.091. This signals the presence of a leaf with error rate of at least 0.291 which we know is leaf 1, $\text{ER}(C_1) = 0.454$.

Since the root node of the tree already comes with a prior on the error rate (BER), we are

interested in how much more value the discovered failure modes in the tree add when compared to the root. Thus, for the automated evaluation we use $\mathrm{ALER} - \mathrm{BER}$ to measure the increase in error and discrepancy that the tree explains.

This metric also suggests that leaves with higher value of $\mathrm{ER}(C_l) \times \mathrm{EC}(C_l)$ contribute more to ALER and are thus more important for explaining failures. This leads to the following metric for ranking nodes for failure explanation:

Definition 14. *The Importance Value i.e* $\mathrm{IV}(C_l)$ *of a leaf node* l *in a tree* T *is defined as:*

$$\mathrm{IV}(C_l) = \mathrm{ER}(C_l) \times \mathrm{EC}(C_l)$$

10.6 Failure modes discovered by Barlow

We now describe some failure modes discovered by Barlow when analyzing a standard Resnet-50 model, using a robust Resnet-50 model for feature extraction (both trained on ImageNet). We use the ImageNet training set (instead of the validation set) for failure analysis due to the larger number of image instances per class (1300 vs. 50 in the validation set). Note that, from a methodology perspective, practitioners may apply the Barlow workflow to any benchmark with ample data and failures, but the failure modes that they discover may vary depending on the nature of the benchmark. For ease of exposition, all decision trees have depth one. We selected the leaf node with highest $\mathrm{IV}(C_l)$ and visualized the feature responsible for the split. More examples of failure modes are in Supp. Section 10.D.1 (using a standard model) and Section 10.D.2 (using a robust model).

Label grouping: In Figure 10.6, the top row shows the most highly activating images for a feature

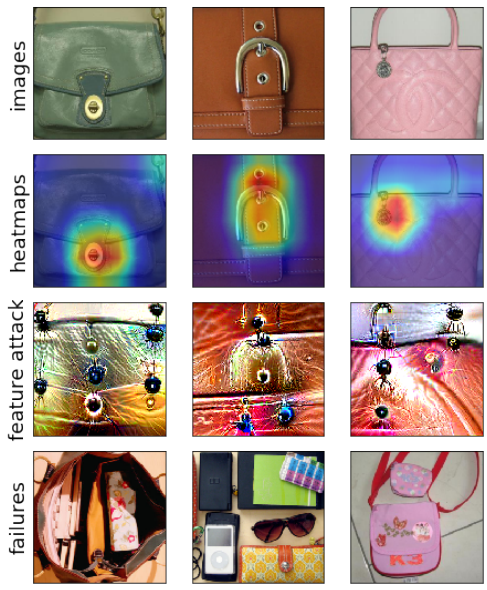


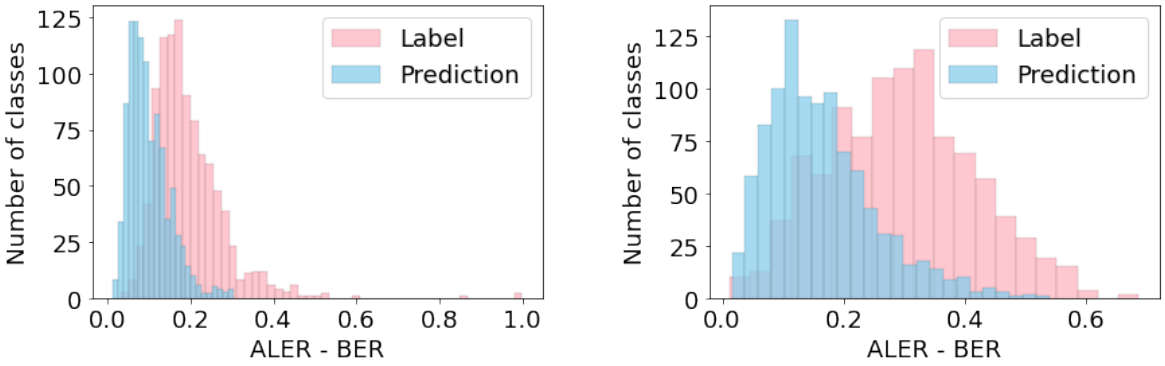
Figure 10.6: For images with **label purse**, BER = 0.3085. When feature[1456] $<$ 0.3641, error rate increases to 0.4179 (increase of **+10.94%**) and a fraction of 0.6409 of all failures are in this node. ALER = 0.3433.



Figure 10.7: For images with **prediction Rhodesian Ridgeback**, BER = 0.2093. When feature[1634] $<$ 1.3779, error rate increases to 0.3184 (increase of **+10.91%**) and a fraction of 0.4561 of all failures are in this node. ALER = 0.2337.

identified important for failure. The heatmap and feature attack provide strong evidence that the feature is paying attention to the buckle on purses. The bottom row shows randomly selected failures in this leaf node. We do not observe a buckle in any of these images even though they have ground-truth labels of purse indicating the feature (while correlated, not causal) is important for making correct predictions. In other words, it appears that, whenever the purse image does not include a buckle, it is more difficult for the model to predict it correctly as a purse (error increases by 10.94%) and the prediction is more likely to be a false negative.

Prediction grouping: Figure 10.7 displays an example where the heatmap and feature attack provide strong evidence that the feature indicates a dog collar. In the bottom row (i.e randomly selected failure images), we do not observe a dog collar in any of these images even though the model predicts a Rhodesian Ridgeback for all of them. This indicates that the feature (correlated,



(a) Standard model

(b) Robust model

Figure 10.8: Comparison between grouping strategies using a decision tree of depth 3.

not causal) is important for making correct predictions. In other words, whenever the model predicts Rhodesian Ridgeback, and the image does not contain a dog collar, the prediction is more likely to be a false positive (error increases by 10.91%).

10.7 Experiments with automated evaluation

We now report on a study of factors that influence the effectiveness of error analysis: decision tree depth, robustness of model, and grouping strategy. We train decision trees with depths of 1 and 3 for each model and grouping strategy. For evaluating a decision tree, we use the metric $\text{ALER} - \text{BER}$ as defined in Section 10.5.1. We also select the leaf with highest importance value $\text{IV}(C_l)$ for each decision tree and evaluate whether the cluster of data in this leaf satisfies the two conditions: $\text{ER}(C_l) > \text{BER} + \delta$ and $\text{EC}(C_l) > \tau$, with $\delta = 0.1$ and $\tau = 0.2$. In Supp. Table 10.1, we report for each model, grouping strategy, and tree depth the fraction of such *valid leaves* across all 1000 classes that satisfy these conditions.

In summary, we make the following observations:

- Grouping by ground-truth labels results in better decision trees (by $\text{ALER} - \text{BER}$ score)

as compared to prediction grouping for both standard and robust models (Figure 10.8), and also for decision trees with different depths.

- Failure explanation for a robust model results in significantly better score compared to the standard model for both grouping strategies and depths of decision tree (See Supp. Figures 10.14, 10.17).

- Increasing depth significantly improves the score for both models as shown in Supp. Figures 10.15 and 10.18.

- For the standard model, on all trees (except prediction grouping and depth=1 where it is 0.211) the fraction of classes with at least one valid leaf is at least 0.596. For the robust model, on all trees the fraction of classes with at least one valid leaf is at least 0.787.

10.8 Crowd study on feature interpretability

To understand the effectiveness of feature visualizations, we conducted a crowdsourcing study using Amazon Mechanical Turk. For both grouping strategies (prediction and label), we selected the top, middle, and bottom 20 classes (total 60 x 2) based on the error rate of the robust Resnet-50 model and selected the top 10 features with the highest mutual information on failure, resulting in 1200 visualizations and 5971 human answers. In total, we had 312 unique workers, each completing an average of 19.14 tasks. All visualizations were evaluated by five workers, except 29 for which we only acquired four assessments. Workers were paid $0.5 per hit, with an average salary of $12 per hour. Anonymized data from this study is available in the associated code repository.

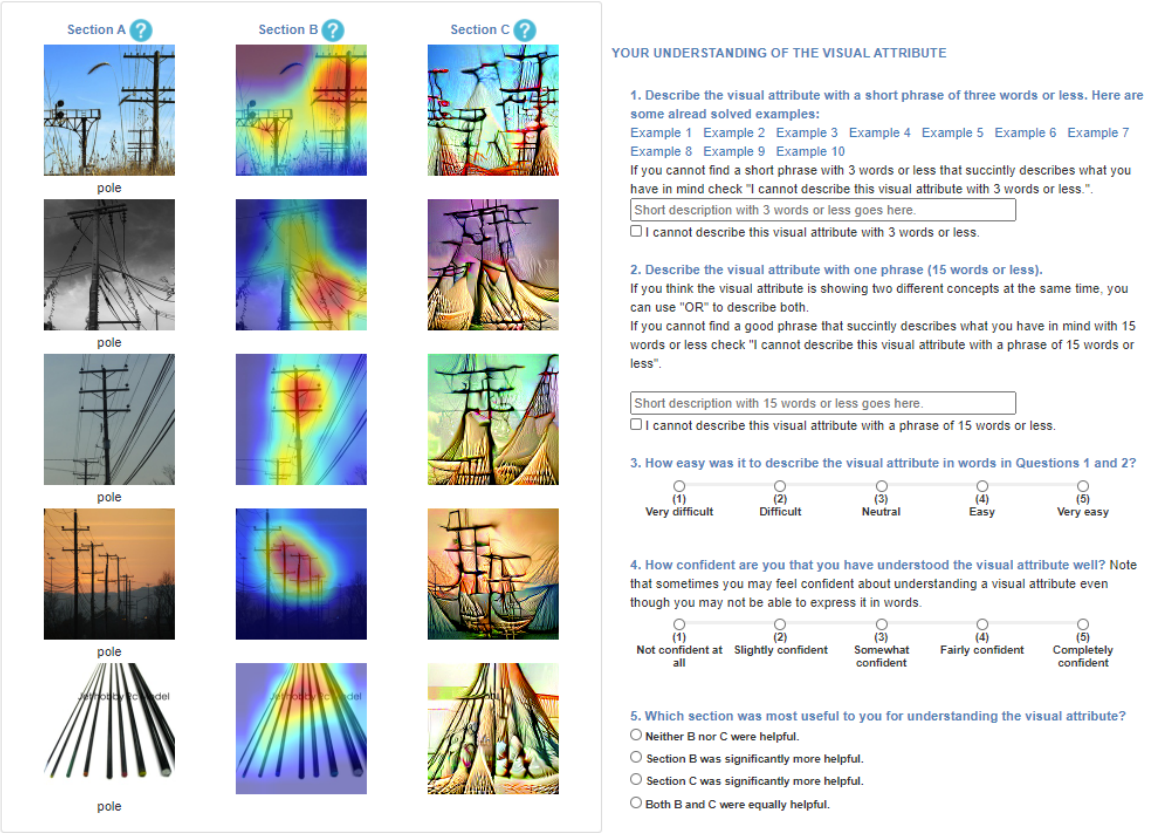
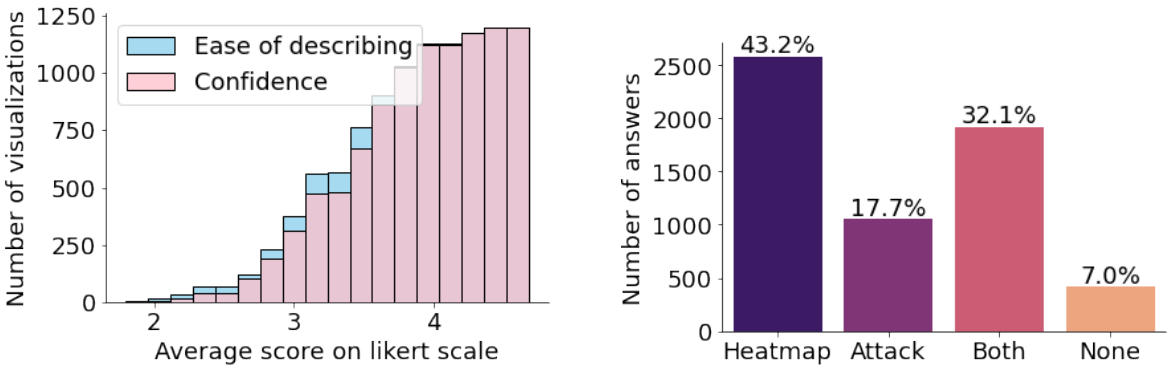


Figure 10.9: Amazon Mechanical Turk questionnaire.

For each visualization, workers were shown three sections: A (most activating images), B (heatmaps), and C (feature attack), and asked to answer the set of questions in Figure 10.9. The questions were designed with two goals: (1) to collect human-generated feature descriptions, and (2) to evaluate the ease with which workers can understand and describe the features.

Figure 10.10a shows the cumulative distribution of answers to Questions 3 (Ease of understanding) and 4 (Confidence). We observe that workers are able to describe visualizations with confidence and ease (average likert scale > 3 for both) for a large number of features (> 800). Figure 10.10b shows workers' preferences of heatmaps versus feature attacks (Question 5). In response to a question about the most useful views, 43.18% answers (out of 5971) report Section B (heatmap) as most useful, 17.67% report Section C (feature attack), and 32.14% report both



(a) Questions 3 and 4

(b) Question 5

Figure 10.10: Cumulative distribution of answers to Questions 3, 4, 5 in the crowd study.

Section B and C. Only 7% of the answers reported None. These results provide evidence that heatmaps are most valuable in contributing to the understanding of the meaning of features, and that feature attack visualizations are also valuable individually and together with heatmaps. Importantly, only 0.92% of all 1200 features received None as the majority vote from all 5 workers. This provides further evidence that the methodology is effective in explaining a large number of features. Note that since crowd workers do not routinely visualize and debug models using such visualizations, our results are likely to be a lower bound on the true effectiveness of the visualizations for engineers.

10.9 Study with machine learning practitioners

To evaluate the usefulness of Barlow for error analysis and debugging, we conducted user studies with 14 ML practitioners at Microsoft. Participants were recruited via four mailing lists on the topics of "Machine Learning" and "Computer Vision". Participation requirements included previous experience in applying machine learning on vision tasks. A summary of roles and years of experience is shown in Table 10.2 in the Supp. Section 10.E.

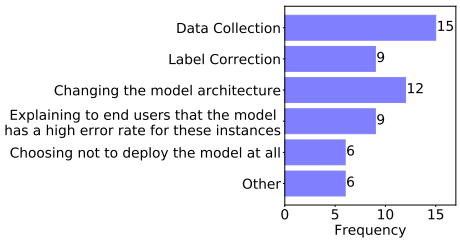


Figure 10.11: Answers of practitioners on identifying action items for mitigating failures discovered by Barlow.

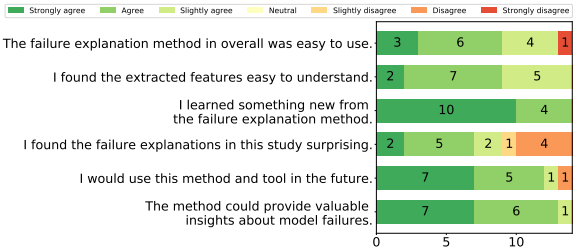


Figure 10.12: Agreement scores of participants on general evaluation of Barlow.

Study Protocol: Each study lasted one hour and started with a description of the Barlow workflow and terminology as shown in Figure 10.2. We asked participants to imagine facing a situation as follows:

"*We will conduct error analysis for a classification model (robust Resnet-50) trained on ImageNet. We ask you to imagine that you are part of a team that will deploy this in production and wants to understand where the model is incorrect and identify action items upon failures.*"

All participants inspected two groupings. The first grouping was randomly assigned from a set of five pre-selected groupings where the most important features for failure explanation were rated as "easy to describe" by crowd workers to facilitate onboarding (exact assignment in Table 10.3, Supp. Section 10.E). The second grouping was selected randomly. In each case, participants were presented with a decision tree (of depth 2) describing the failure modes. For each node, they could see the error rate and coverage, the instances in the node, and the visualization of the feature responsible for the split. To collect feedback on their experience, we asked the following questions:

Q1: *Can you describe the cluster of images defined by this feature?* All participants were able to describe the features of at least one of the two groupings they inspected. 10 out of 14 participants were able to describe features of both groupings. The four participants who could not describe the

features of one grouping faced one or both of the following challenges: the feature represented two or more concepts (e.g., sky and wires), or participants had a difficult time understanding the sensitivity of the feature's visual appearance to its activation value. In such cases, they preferred to see more than five examples for refining their hypothesis.

Q2: *Is the feature necessary for the task or do you think it is a spurious correlation?* Participants were able to identify several surprising spurious correlations (e.g., "*This feature looks like a hair detector, should not be used for Seat Belts.*" - P12, "*Seems like the model can do well only when the photos of Water Jugs are taken either professionally or on a clean background.*" - P17). Similarly, they could identify when the failure modes were related to necessary but not sufficient features (e.g., "*The model is focusing on the face of the dog. We might also need to make the model look at the legs, hair length, hair color, etc.*" - P7).

Q3: *What action items would you take for mitigating the errors for this class?* Overall, participants reported that the identified failure explanations gave them ideas about how to continue with mitigating the errors, given sufficient time and resources. Data collection was the most popular action item (Figure 10.11) either for mitigating lack of data diversity or addressing statistical biases (e.g. "*I would check if most Tiger Cats in the training data have green background and if so, add more diverse photos.*" - P5). Changes in model architecture were most often related to increasing the capacity of the model (e.g., "*The model is using the same features for detecting zebra patterns and spatulas, perhaps it's best to use more than one feature for this.*" - P5).

User satisfaction scores: The study ended with a survey on whether participants agreed with six statements, completed in private (Figure 10.12). We observe overall enthusiasm about the usefulness of the workflow and its ability to surface important failures (e.g. "*The approach can help make DNNs more explainable, making such analysis mandatory before deploying a*

solution. A step further could be to take a similar test with humans to ensure the model has learned explainable features, before we ship the model." - P7).

10.10 Conclusion and future work

We described proposed methods and a workflow aimed at providing insights about critical failures of machine learning models by using features extracted from robust representations. Beyond developing and refining the approach, we performed two sets of studies with human subjects, focusing on the interpretability of features and on the usefulness of the error analysis in realistic settings. The studies provide evidence that practitioners can effectively leverage the methods to interpret the features and to identify most significant clusters in data with errors due to issues, such as systematic spurious correlations. We believe that the error analyses and visualizations embodied in Barlow show promise for supporting cycles of iterative improvement that can identify and mitigate failures without requiring expensive manual metadata collection.

Directions for refining Barlow include extending the range of actions suggested automatically to practitioners and providing richer interactive capabilities. Interactive explorations can include hypothesis testing via the consideration of counterfactuals that make specific, understandable changes to the data and models. Another enabling direction centers on extending the current correlational analyses to causal studies. Extensions that establish causal influences on failure modes will be valuable for debugging non-robust models with robust features.

Supplementary material

10.A Failure mode generation

We describe our procedure for generating failure modes in Algorithm 6. The algorithm can take as an input any cluster of image data C. In our experiments, the clusters were defined via image grouping by label and model prediction. However, practitioners may choose to apply the same procedure to clusters of images defined in other ways such as for example pairs of classes that are often confused with each other or unions of prediction and label groupings for the same class.

Algorithm 6 Failure mode generation procedure

procedure FAILURE_MODES(features F, model h, image cluster C, number_of_features k, tree_parameters A, error_rate_threshold δ, error_coverage_threshold τ)
  $L \leftarrow \emptyset$
  $\text{BER} \leftarrow \text{ER}(C)$
  $E(x) \leftarrow \begin{cases} 0 & h(x) = y \\ 1 & h(x) \neq y \end{cases} \quad \forall (x, y) \in C$
  $F^* \leftarrow \emptyset$
  while $|F^*| < k$ **do**
    $F^* \leftarrow F^* \cup \arg\max_{f \in F \setminus F^*} \text{MI}(E; f)$ ▷ MI calculates mutual information
  end while
  $T \leftarrow$ train_decision_tree(F^*, E, A)
  for all $l \in T$ **do**
    if $(\text{ER}(C_l) > \text{BER} + \delta)$ and $(\text{EC}(C_l) > \tau)$ **then**
      $L = L \cup \{l\}$
    end if
  end for
  return L
end procedure

10.B Automatic evaluation of decision tree

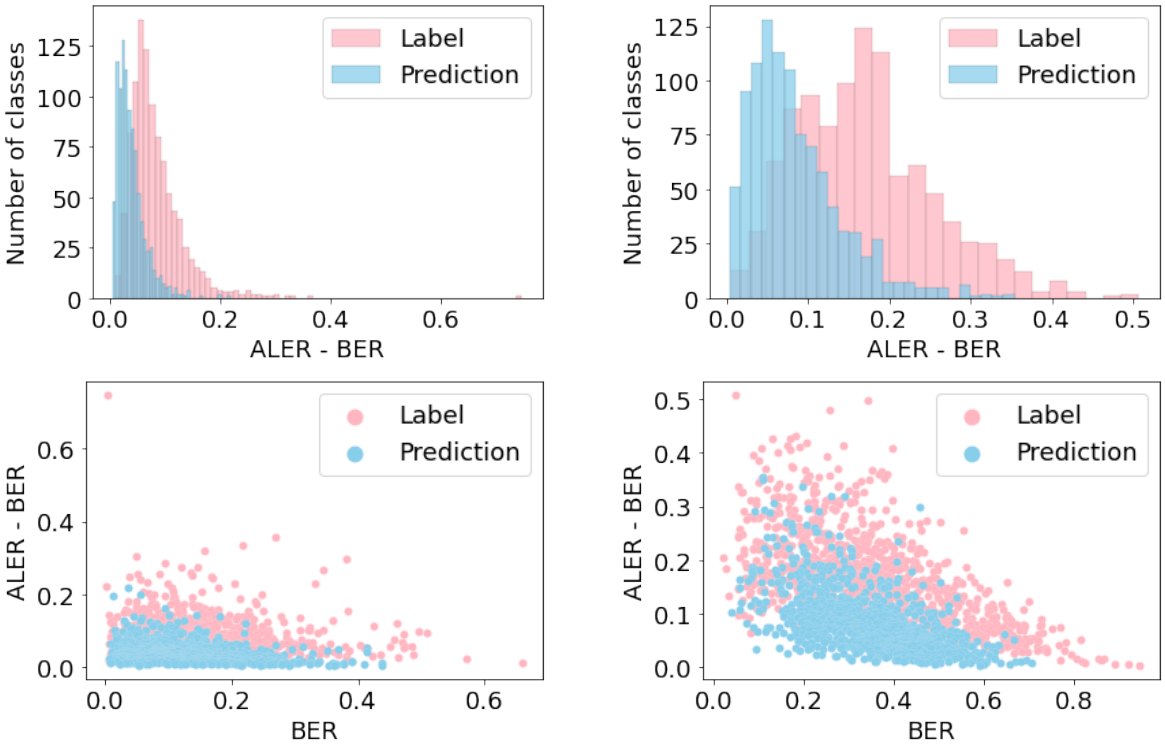
We now report on a study of factors that influence the effectiveness of error analysis: decision tree depth, robustness of model, grouping strategy. We train decision trees with depths of 1 and 3 for each model and grouping strategy. For evaluating a decision tree, we use the metric $\mathrm{ALER} - \mathrm{BER}$ as defined in Maintext Section 10.5.1, Definition 13. We also select the leaf with highest importance value $\mathrm{IV}(C_l)$ for each decision tree (Maintext Definition 14) and evaluate whether the cluster of data in this leaf satisfies the two conditions: $\mathrm{ER}(C_l) > \mathrm{BER} + \delta$ and $\mathrm{EC}(C_l) > \tau$, with $\delta = 0.1$ and $\tau = 0.2$. In Table 10.1, we report for each model, grouping strategy, and tree depth the fraction of such *valid leaves* across all 1000 classes that satisfy these conditions.

We make the following observations:

- Grouping by ground-truth labels results in better decision trees (by $\mathrm{ALER} - \mathrm{BER}$ score) compared to prediction grouping for both standard and robust models and also for decision trees with different depths. This is true even when BER is similar (See Figures 10.13 and 10.16).
- Failure explanation for a robust model results in significantly better score compared to standard model for both grouping strategies and depths of decision tree. This is again true, even when BER is similar (See Figures 10.14 and 10.17). While this observation is intuitive, given that that the extracted features come from the robust model, it serves as an additional motivation for employing robust models in practice. The evaluation shows that such models might simplify the debugging and error analysis processes.

Model	Depth	Grouping	Fraction
Standard	1	Label	0.596
Standard	1	Prediction	0.211
Standard	3	Label	0.900
Standard	3	Prediction	0.649
Robust	1	Label	0.977
Robust	1	Prediction	0.787
Robust	3	Label	0.899
Robust	3	Prediction	0.804

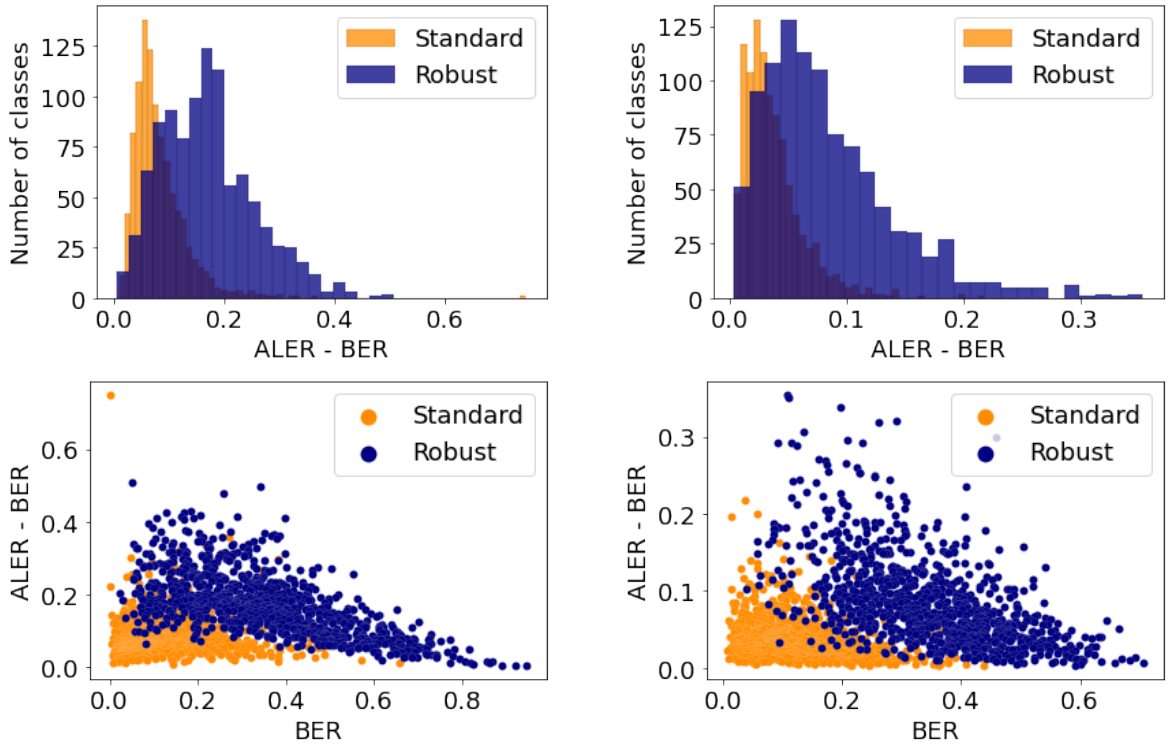
Table 10.1: For each model, grouping strategy and decision tree depth we report the fraction of *valid leaves* across all 1000 classes, i.e the leaf nodes that satisfy these two conditions: $\mathrm{ER}(C_l) > \mathrm{BER} + \delta$ and $\mathrm{EC}(C_l) > \tau$, with $\delta = 0.1$ and $\tau = 0.2$ in the last column. Semantically, these would be all leaves with an error increase of at least 10% that cover 20% of the failures or more.



(a) Standard model

(b) Robust model

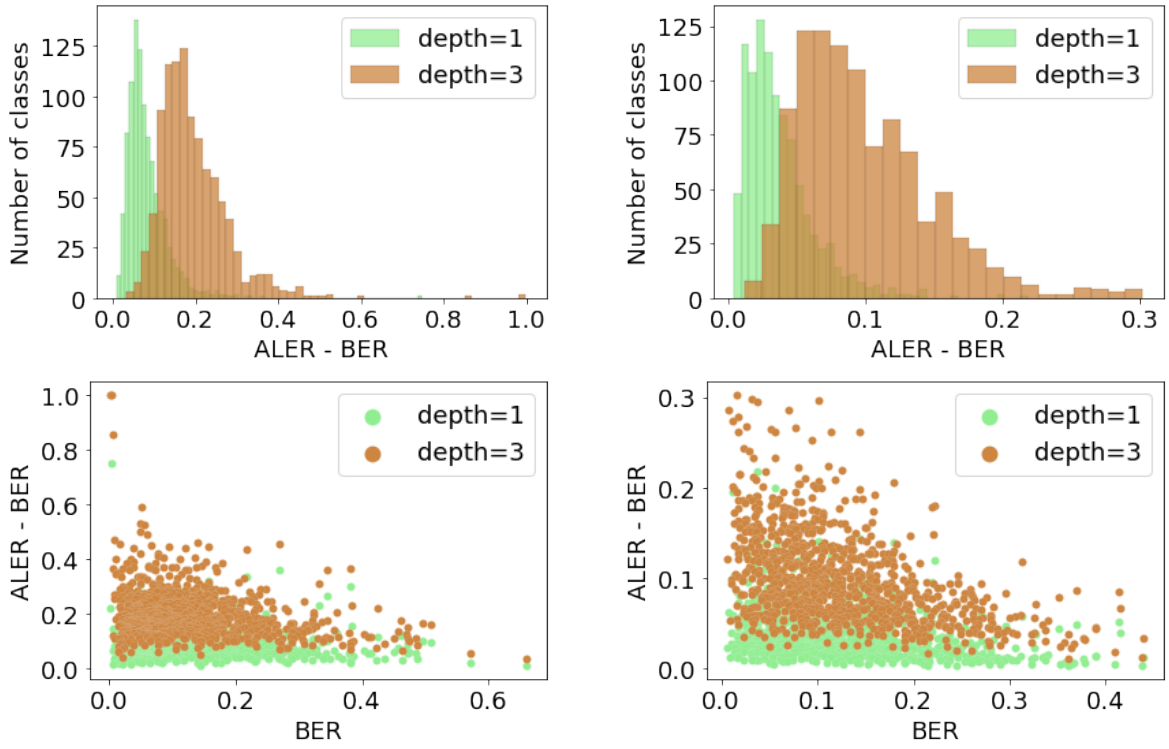
Figure 10.13: Comparison between grouping strategies using a decision tree of depth 1.



(a) Label grouping

(b) Prediction grouping

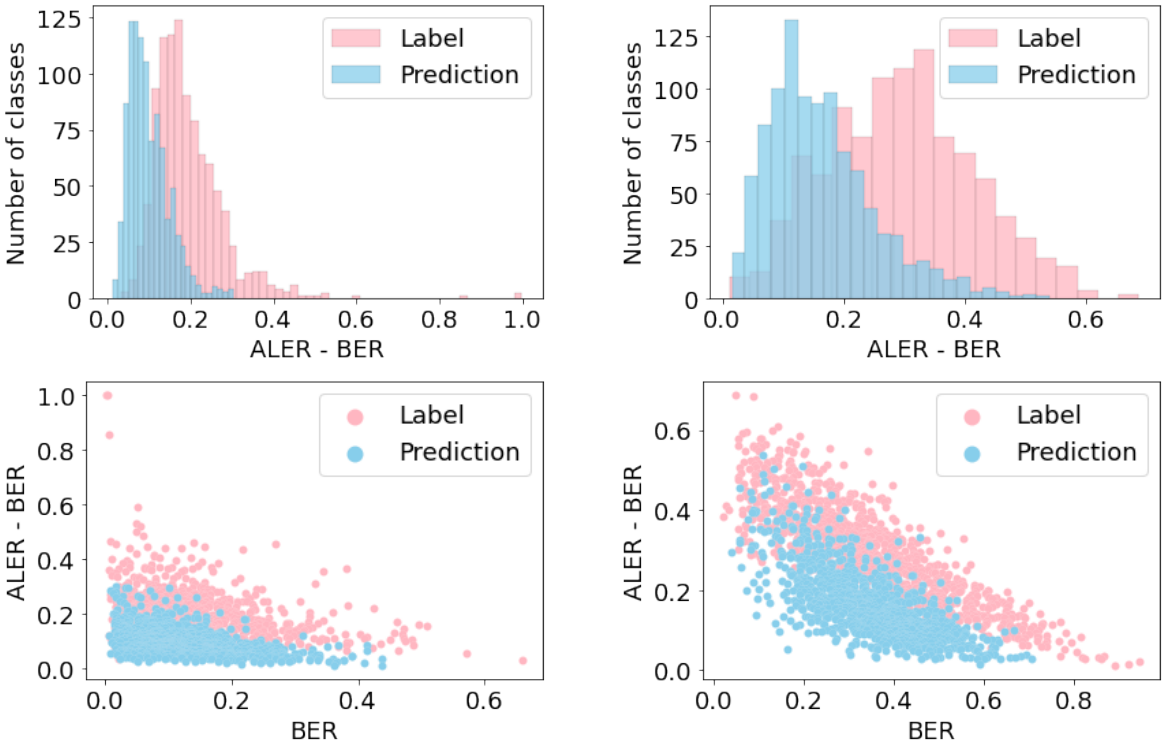
Figure 10.14: Comparison between standard and robust models using a decision tree of depth 1.



(a) Label grouping

(b) Prediction grouping

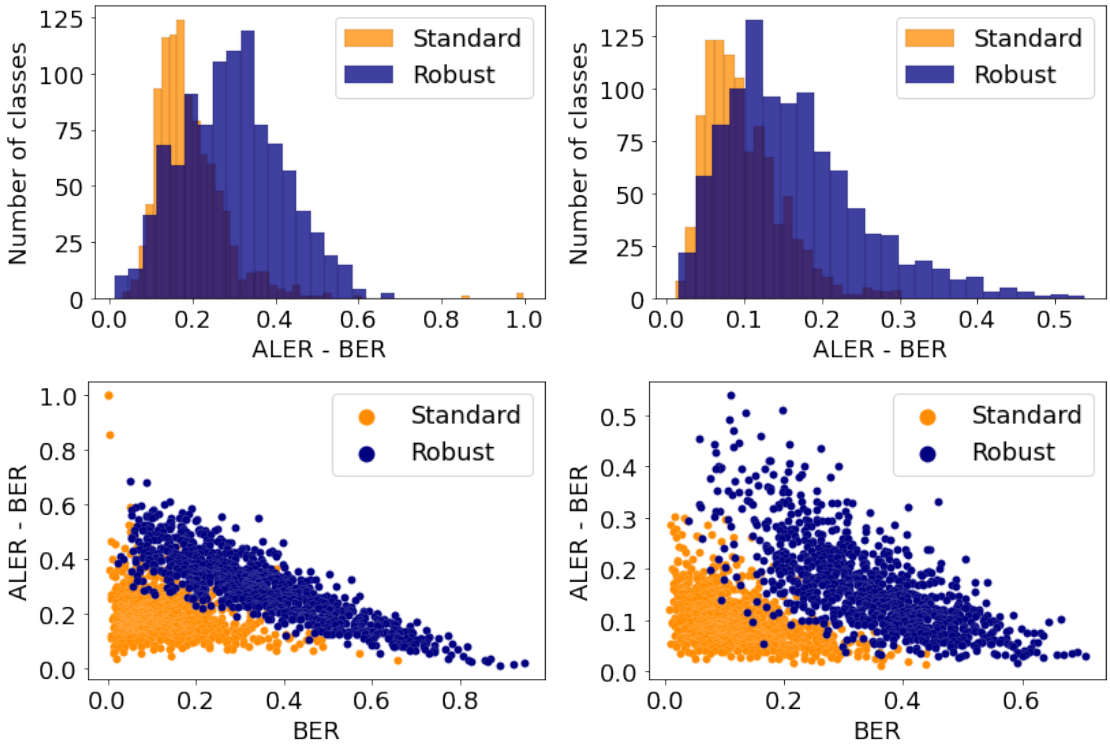
Figure 10.15: Comparison between decision trees of different depths using a standard model.



(a) Standard model

(b) Robust model

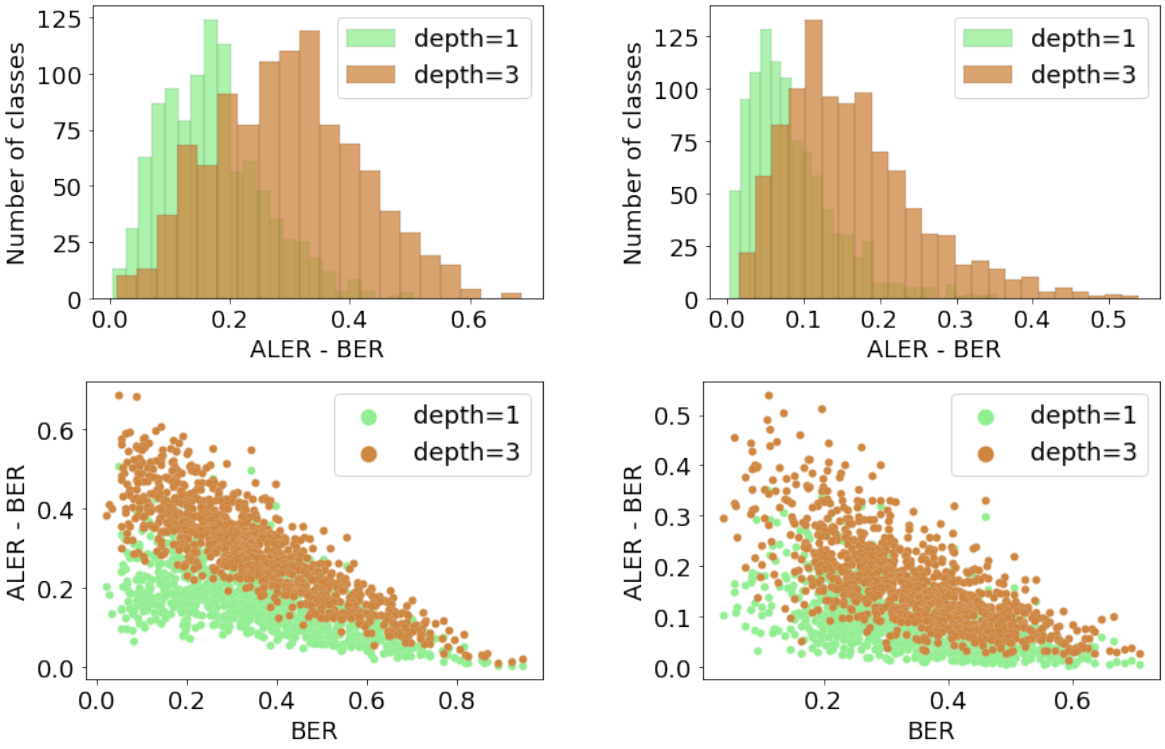
Figure 10.16: Comparison between grouping strategies using a decision tree of depth 3.



(a) Label grouping

(b) Prediction grouping

Figure 10.17: Comparison between standard and robust models using a decision tree of depth 3.



(a) Label grouping

(b) Prediction grouping

Figure 10.18: Comparison between decision trees of different depths using a robust model.

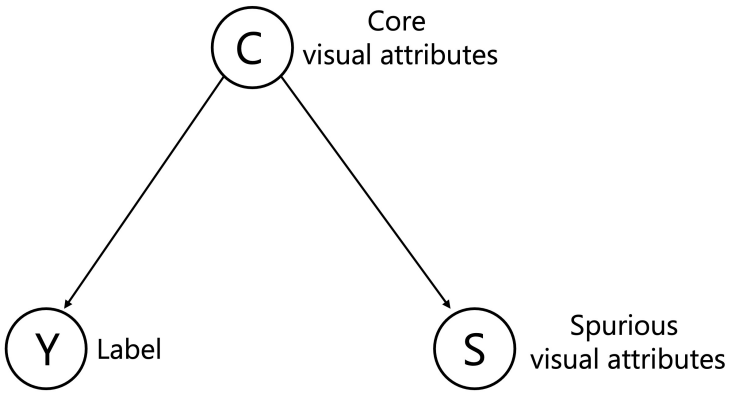


Figure 10.19: A simple graphical model relationship between core, spurious and label variables.

10.C Formal Definitions of Core and Spurious visual attributes

We can define core and spurious visual attributes formally through a simple graphical model depicted in Figure 10.19 (also explained in [312, 313, 314]) Let us consider Y, C and S as label, core and spurious random variables respectively. This graphical model indicates that $\mathrm{Y} \perp \mathrm{S} \mid \mathrm{C}$ i.e. Y and S are conditionally independent given C. This graphical model can be viewed through the lens of the Reichenbach's Common Cause Principle which indicates that since there is a correlation (and not causation) relationship between S and Y, they have a common cause that renders them conditionally independent.

In an image classification problem, mathematically defining sets S and C can be quite challenging. Thus, we use the following practical definitions:

- **Core Attributes** are the set of visual features that are part of the main object (i.e. labelled with Y).
- **Spurious Attributes** are the ones that are likely to *co-occur* with the object, but not a part of it (such as food, vase, flowers in Figure 11.1).

With these definitions, suppose C represents features of the bird; S represents features of tree branches and Y is the bird label, then Y is independent of S given C i.e. if the bird is present, the label is bird irrespective of whether tree branches are present or not.

We note that our proposed approach might not be effective for discovering *non-spatial* spurious signals such as gender/race that are difficult to visualize using heatmaps and feature attacks. Another limitation of our method is that we define something as spurious if it is not a part of the main object. Consider the class beach. For this class, water is not a part of the beach and we find that participants in our user studies label it as a spurious feature. However, it is clearly not spurious because if water is not present, we would classify it as desert. Finally, our definitions do not consider potential spurious features that can be on the main objects. However, we note that our proposed methodologies can be potentially extended by incorporating application-specific notions of spurious/core features.

10.D Failure modes discovered by Barlow

In this section, we describe several failure modes discovered by Barlow. For experiments in subsection 10.D.1, we analyze the errors of a standard Resnet-50 model for failure analysis and for subsection 10.D.2, we inspect a robust Resnet-50 model. In both cases, we use a robust Resnet-50 model for feature extraction. All models were pretrained on ImageNet. We use the ImageNet training set (instead of the validation set) for failure analysis due to the larger number of instances and failures. For ease of exposition, all decision trees have depth one. We select the leaf node with highest Importance Value (i.e IV as defined in Definition 14) for visualizing the failure mode. Since the tree has depth one, we can visualize the one feature that defines this leaf node.

All feature visualizations are organized as follows. The topmost row shows the most activating images. The second row shows the heatmaps The third row shows feature attack images. Finally, the bottom row shows randomly selected failure examples in the leaf node.

For all tables, BER denotes the Base Error Rate, ER denotes Error Rate, EC denotes Error Coverage for the leaf with highest Importance Value and ALER denotes Average Leaf Error Rate.

10.D.1 Failure explanation for a standard model

10.D.1.1 Grouping by label

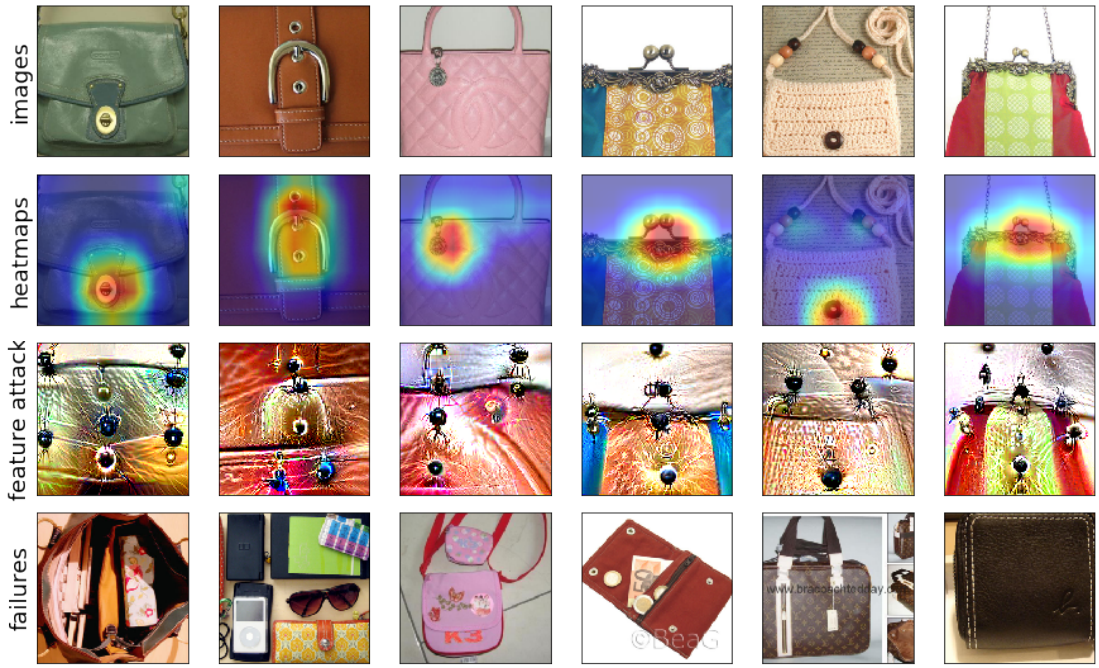


Figure 10.20: Visualization of feature[1456]. For images with **label purse**, when feature$[1456] < 0.3641$, error rate increases to 0.4179 (**+10.94%**).

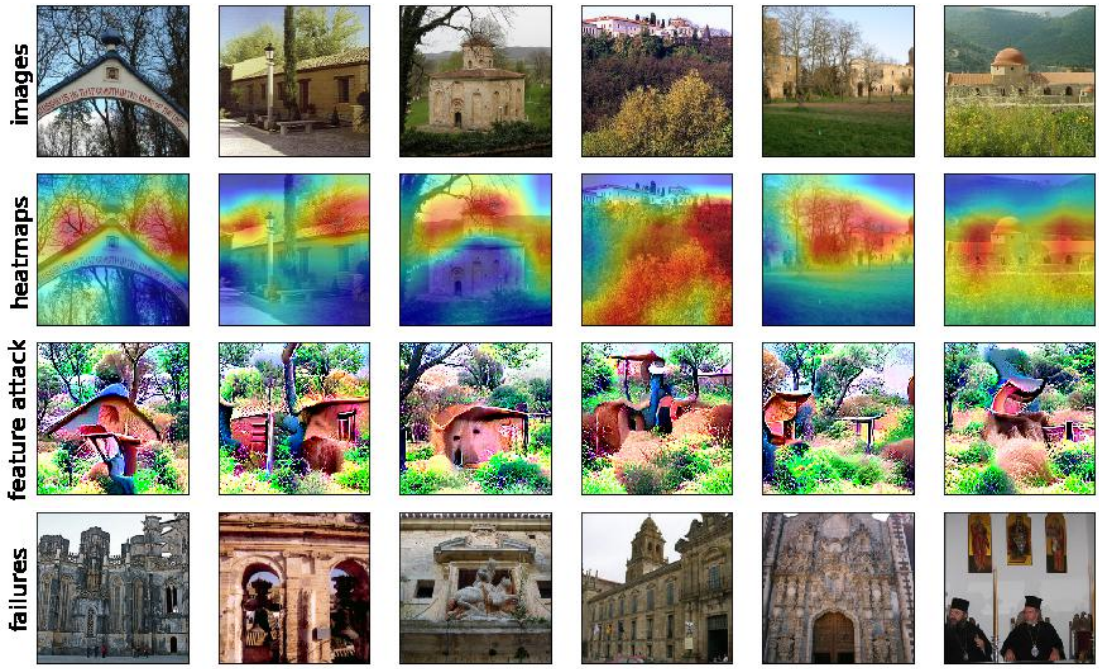


Figure 10.21: Visualization of feature$[995]$. For images with **label monastery**, when feature$[995] < 0.1428$, error rate increases to 0.6345 (**+24.84%**).

10.D.1.2 Grouping by prediction

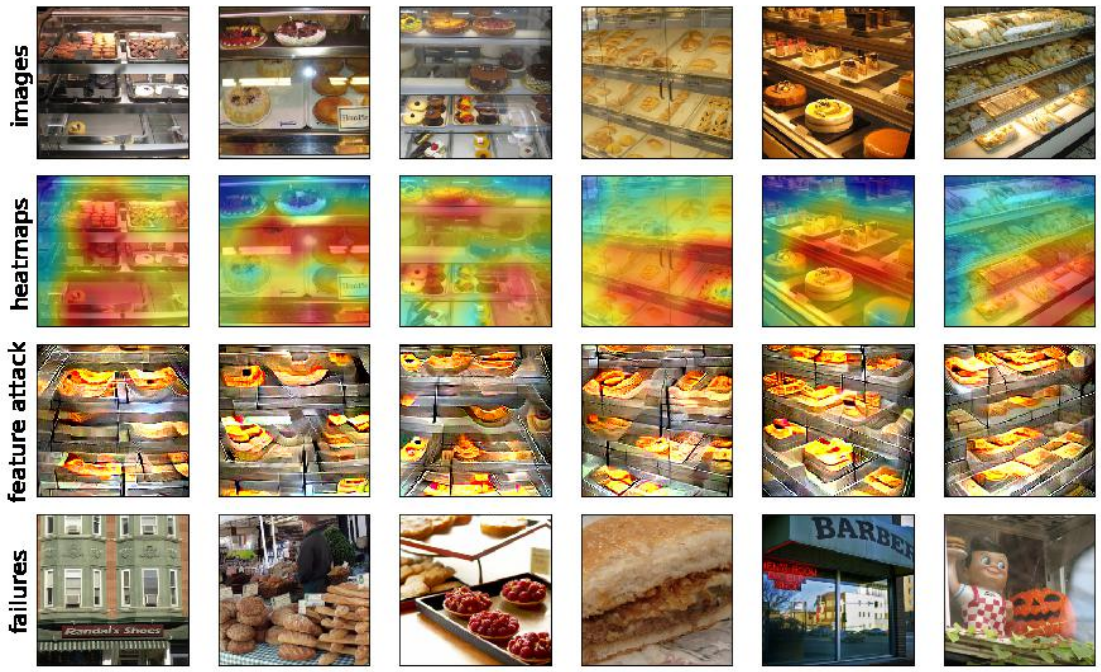


Figure 10.22: Visualization of feature[443]. For images with **prediction bakery**, when feature$[443] < 1.1382$, error rate increases to 0.3875 (**+11.80%**).

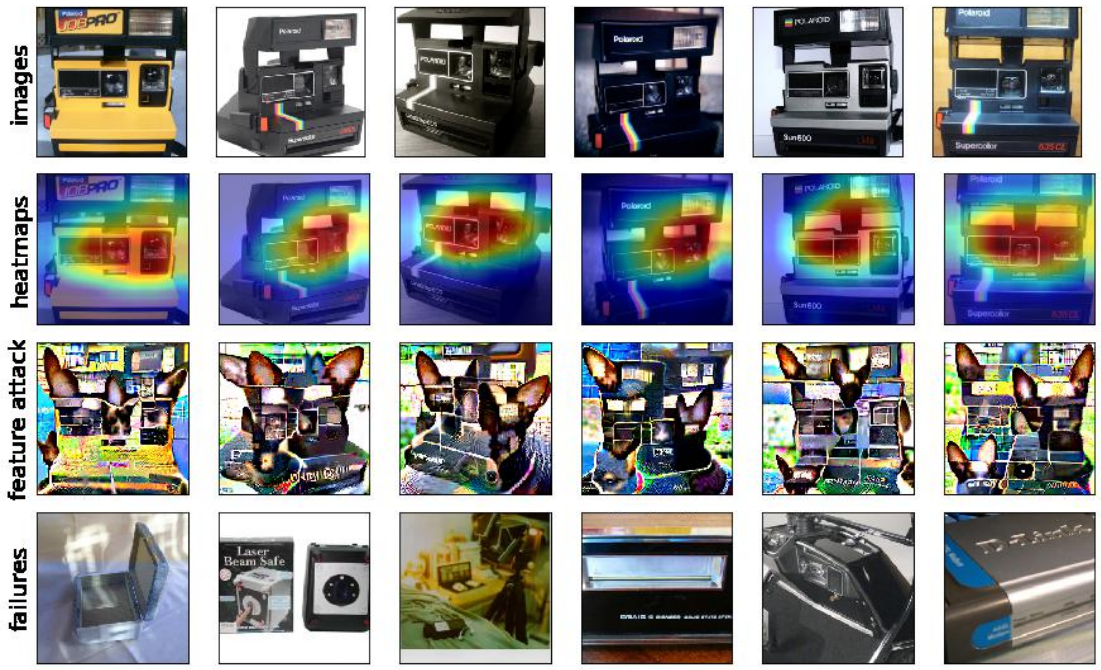


Figure 10.23: Visualization of feature[793]. For images with **prediction polaroid camera**, when feature$[793] < 0.8166$, error rate increases to 0.2713 (**+15.72%**).

10.D.2 Failure explanation for a robust model

10.D.2.1 Grouping by label

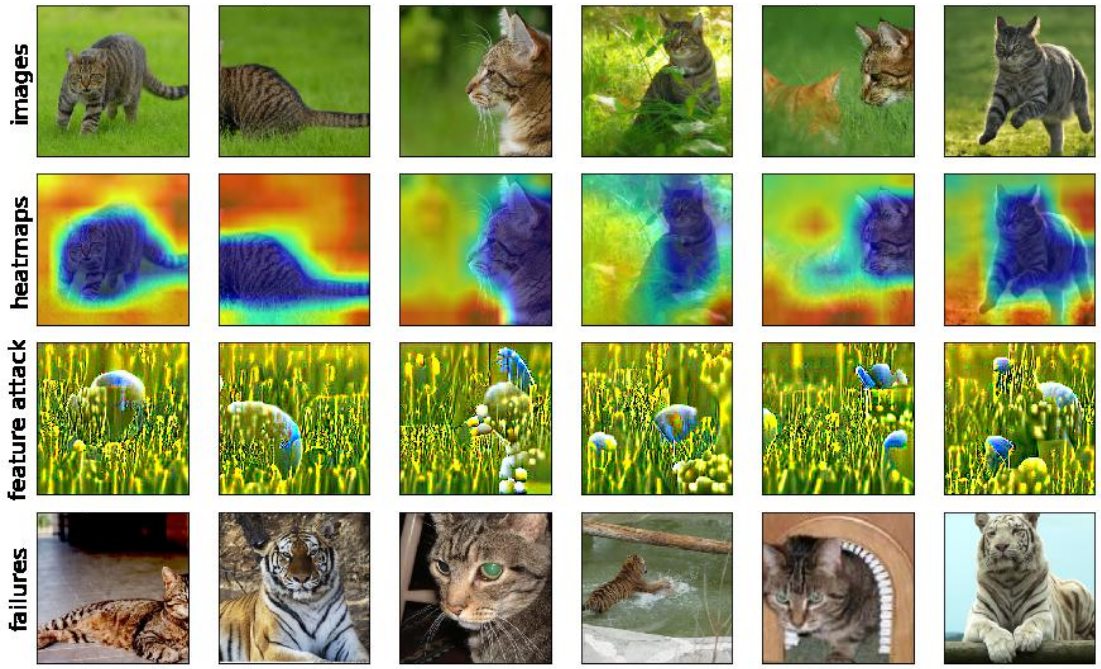


Figure 10.24: Visualization of feature[1864]. For images with **label tiger cat**, when feature$[1864] < 0.4673$, error rate increases to 0.8786 (**+10.71%**).



Figure 10.25: Visualization of feature[380]. For images with **label lighter**, when feature$[380] < 1.2961$, error rate increases to 0.8608 (**+13.23%**).

10.D.2.2 Grouping by prediction

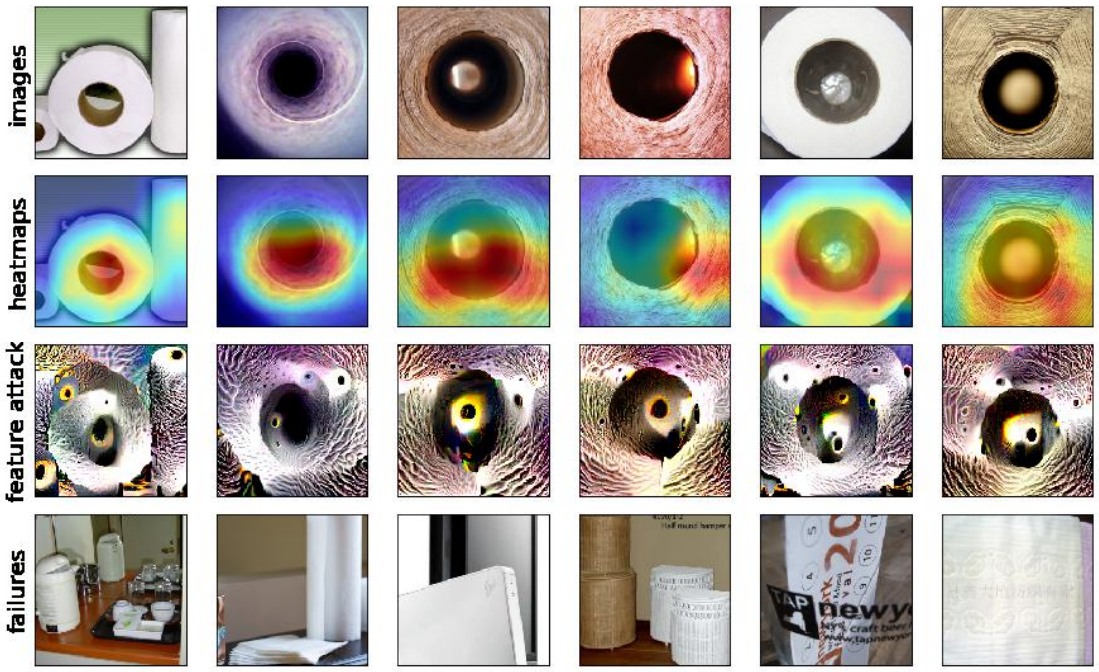


Figure 10.26: Visualization of feature[1412]. For images with **prediction paper towel**, when feature$[1412] < 1.2665$, error rate increases to 0.7825 (**+15.15%**).

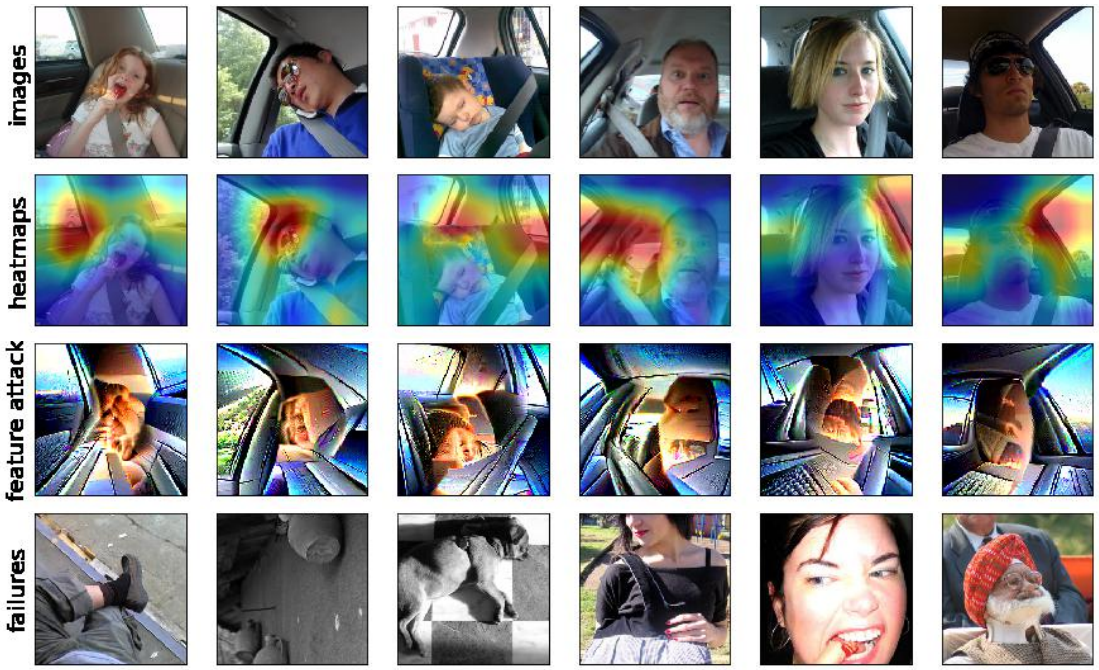


Figure 10.27: Visualization of feature[1493]. For images with **prediction seat belt**, when feature$[1493] < 1.0624$, error rate increases to 0.7402 (**+14.19%**).

10.E User study with ML practitioners

Role	**Participants**
ML Engineer	5 [P2, P4, P5, P11, P18]
Applied Scientist	2 [P9, P12]
Researcher	4 [P1, P7, P16, P17]
Data Scientist	3 [P10, P20, P21]
Experience in ML	**Participants**
1 - 2 years	1 [P2]
2 - 5 years	4 [P5, P10, P11, P20]
5 - 10 years	5 [P4, P7, P16, P17, P18]
$>$ 10 years	4 [P1, P9, P12, P21]

Table 10.2: Distribution of roles and years of experience in Machine Learning among ML practitioners in the study.

Class id	**Class name**	**Grouping**	**Top-1 Error**	**Participants**
424	Barbershop	prediction	68.32%	3 [P10, P18, P20]
703	Park Bench	label	33.31%	3 [P9, P11, P17]
785	Seat Belt	label	33.23%	4 [P2, P4, P12, P21]
820	Steam Locomotive	label	6.69%	1 [P1]
282	Tiger Cat	label	77.15%	3 [P5, P7, P16]

Table 10.3: Distribution of the first class groupings among machine-learning practitioners. The five examples contained features that were considered as "easy to describe" by Mturk workers to facilitate onboarding. The second class grouping was instead assigned randomly from the set of 120 class groupings that were part of the MTurk study. (Top-1 Error is for Robust Resnet-50)

10.F Comparison between the interpretations of a robust and non-robust model

To compare the interpretations of a robust model with a non-robust model, we analyzed the failures of top-5 classes with highest number of failures in the non-robust model (using grouping

by label). The feature visualizations for the 5 classes and the respective most important feature for failure explanation are given below. We observe that using a robust model for feature extraction and visualization leads to significantly more interpretable visualizations qualitatively. While we did not conduct quantitative comparisons with humans studies (robust vs. non-robust features) we encourage future research in this space that may exclusively focus in describing such differences at a larger extent.

10.F.1 Class name: water jug

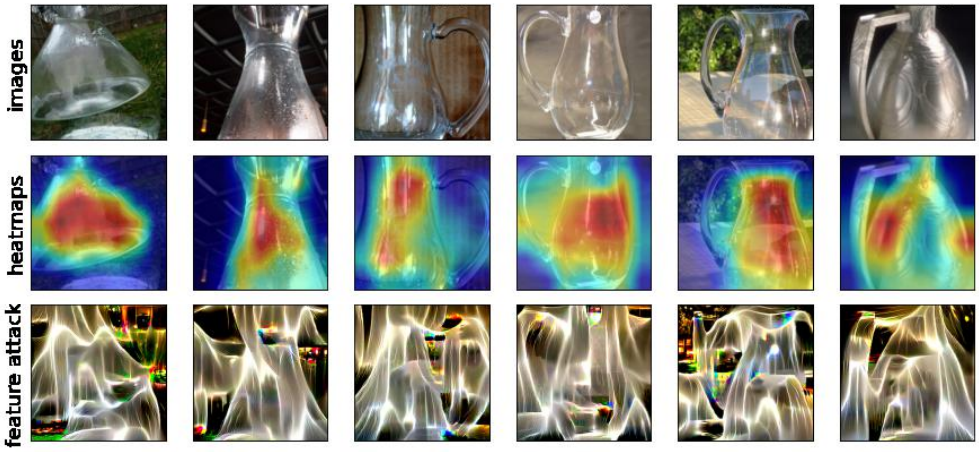


Figure 10.28: Feature visualization **using a robust model**. Visualization of feature[1725], class[899] (water jug) and label grouping.

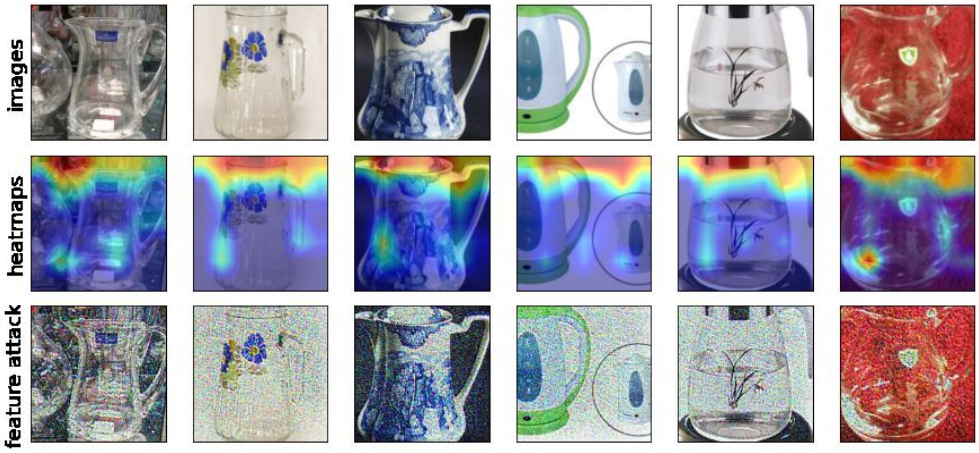


Figure 10.29: Feature visualization **using a non-robust model (standard model)**. Visualization of feature[1357], class[899] (water jug) and label grouping.

10.F.2 Class name: horned viper

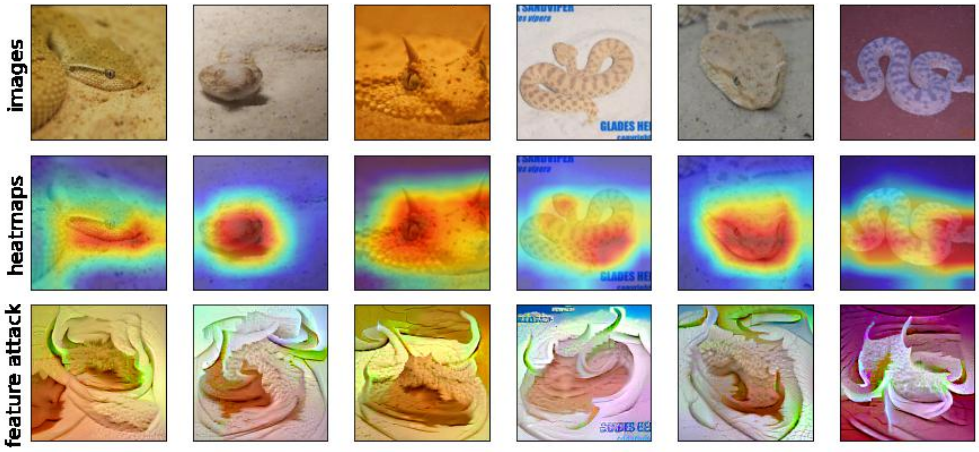


Figure 10.30: Feature visualization **using a robust model**. Visualization of feature[54], class[66] (horned viper) and label grouping.

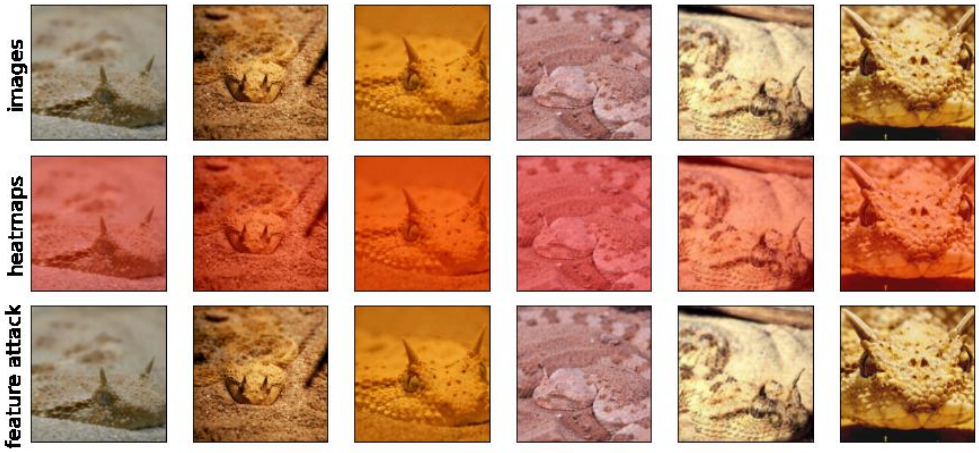


Figure 10.31: Feature visualization **using a non-robust model**. Visualization of feature[378], class[66] (horned viper) and label grouping.

10.F.3 Class name: tiger cat

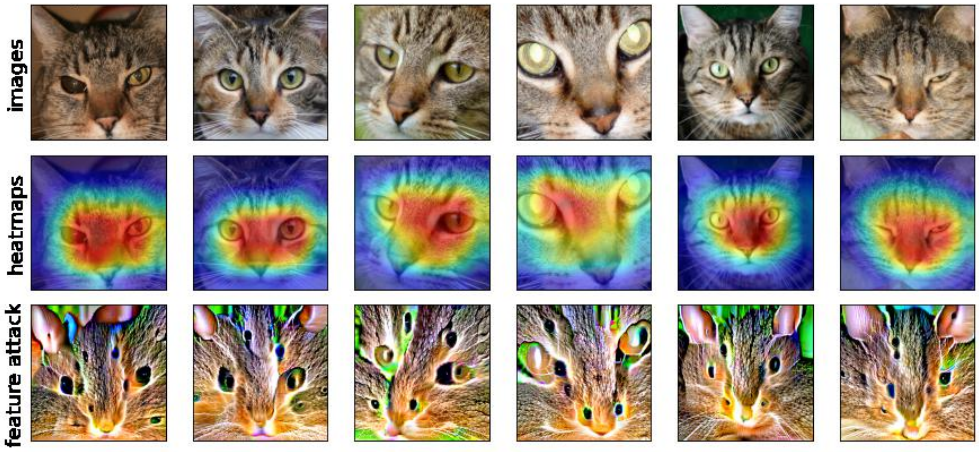


Figure 10.32: Feature visualization **using a robust model**. Visualization of feature[544], class[282] (tiger cat) and label grouping.

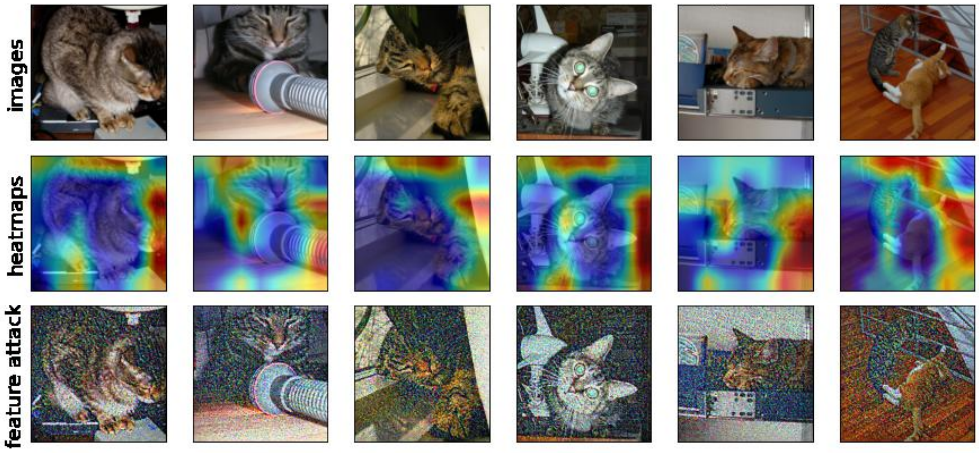


Figure 10.33: Feature visualization **using a non-robust model**. Visualization of feature[1075], class[282] (tiger cat) and label grouping.

10.F.4 Class name: tape player

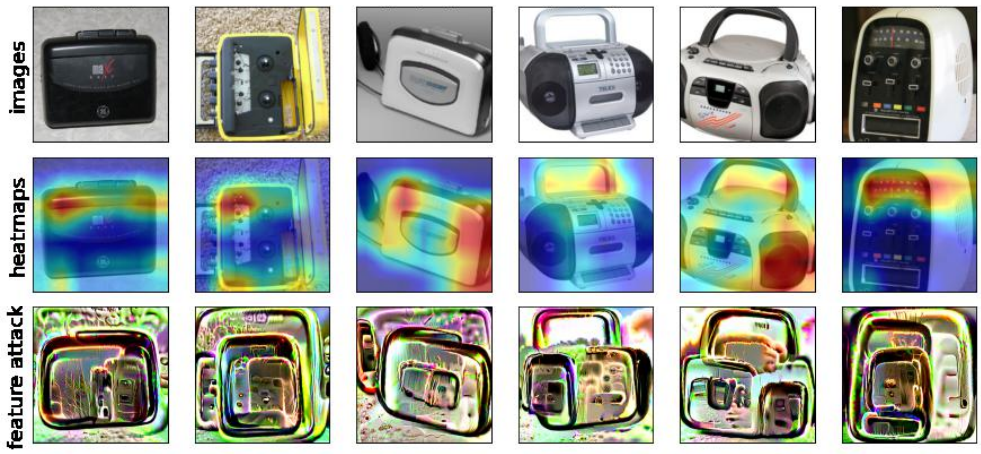


Figure 10.34: Feature visualization **using a robust model**. Visualization of feature[1751], class[848] (tape player) and label grouping.

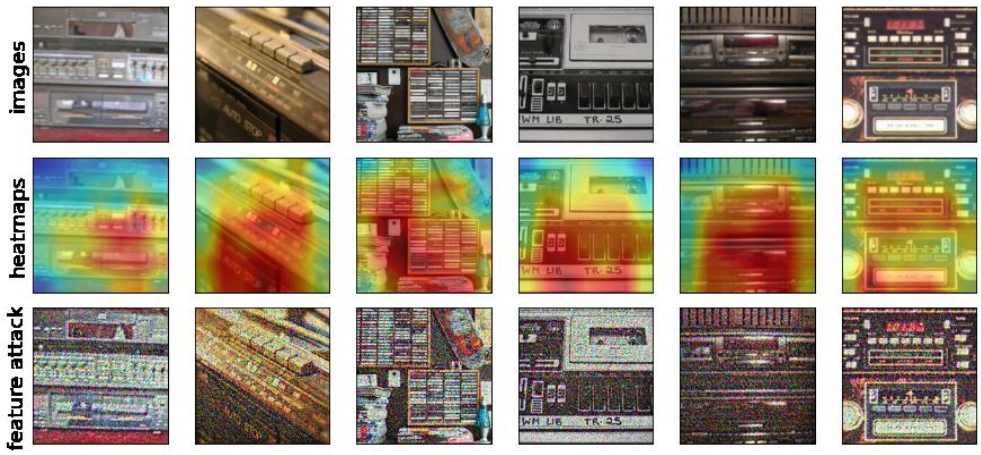


Figure 10.35: Feature visualization **using a non-robust model**. Visualization of feature[935], class[848] (tape player) and label grouping.

10.F.5 Class name: overskirt

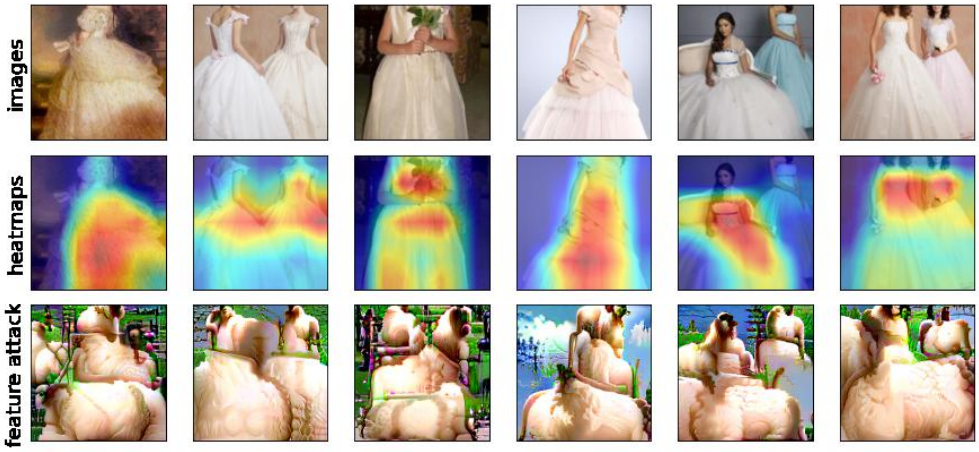


Figure 10.36: Feature visualization **using a robust model**. Visualization of feature[343], class[689] (overskirt) and label grouping.

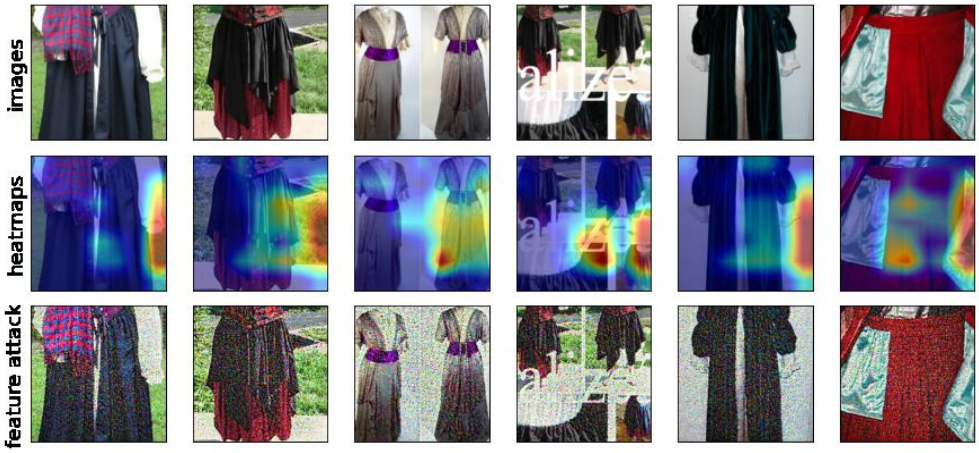


Figure 10.37: Feature visualization **using a non-robust model**. Visualization of feature[1405], class[689] (overskirt) and label grouping.

Chapter 11: Salient ImageNet: How to discover spurious features in deep learning?



(a) **class:** band aid, **spurious feature**: fingers, **-41.54%** (b) **class:** space bar, **spurious feature**: keys, **-46.15%** (c) **class:** plate, **spurious feature**: food, **-32.31%** (d) butterfly, **spurious feature**: flowers, **-21.54%** (e) potter's wheel, **spurious feature**: vase, **-21.54%**

Figure 11.1: Examples of spurious features discovered using our framework for a Standard Resnet-50 model. In the top row, red color highlights regions with spurious features. The bottom row shows images generated by visually amplifying these features. Adding a small amount of gaussian noise (with $\sigma = 0.25$) to spurious regions (red regions) significantly reduces the model accuracy (shown in dark red in captions) for a subset of 65 images with the shown class labels.

11.1 Introduction

The growing use of deep learning systems in sensitive applications such as medicine, autonomous driving, law enforcement and finance raises concerns about their trustworthiness and reliability

in the real world. A key reason for the lack of reliability of deep models is their reliance on spurious input features (i.e., features that are not essential to the true label) in their inferences. For example, a convolutional neural network (CNN) trained to classify camels from cows associates green pastures with cows and fails to classify pictures of cows in sandy beaches correctly since most of the images of cows had green pastures in the training set [16, 315]. Similarly, [316] discovered that skin-lesion detectors use spurious visual artifacts for making predictions. The list of such examples goes on [317, 318].

Most of the prior work on discovering spurious features [305, 306, 307] requires humans to first come up with a possible list of spurious features. This is often followed by an expensive human-guided labeling of visual attributes which is not scalable for datasets with a large number of classes and samples such as ImageNet. In some cases, this may even require the construction of new datasets with unusual visual attributes to validate the hypothesis (such as cows in sandy beaches discussed previously). To address these limitations, recent works [318, 319] use the neurons of robust models as visual attribute detectors thereby circumventing the need for manual annotations. However, their applicability is either limited to robust models which achieve low clean accuracy or to models that achieve low accuracy on the training dataset. We discuss limitations of these methods in more detail in Section 11.2.

In this chapter, we introduce a *general* methodology to discover a subset of spurious and core (non-spurious) visual attributes used in model inferences and localize them on a large number of images with minimal human supervision. Our work builds upon the prior observations [274] that for a robust model, neurons in the penultimate layer (called *neural features*) often correspond to human-interpretable visual attributes. These visual attributes can be inferred by visualizing the *heatmaps* (e.g. via CAM, [282]) that highlight the importance of image pixels

for those neural features (top row in Figure 11.1). Alternatively, one can amplify these visual attributes in images by maximizing the corresponding neural feature values via a procedure called the *feature attack* ([274, 318], bottom row in Figure 11.1).

Our framework (shown in Figure 11.2) is based on this key idea: to identify spurious or core *visual attributes* used for predicting the class i, we identify spurious or core *neural features* via limited human supervision. We define a core visual attribute as an attribute for class i that is always a part of the object definition and spurious otherwise. To annotate a neural feature as core or spurious, we show only the top-5 images (with a predicted label of i) that maximally activate that neural feature to Mechanical Turk workers. We then show that these neural feature annotations *generalize* extremely well to top-k (with $k \gg 5$) images with label i that maximally activate that neural feature. For example, on Imagenet, in Section 11.3.4, we show a generalization accuracy of 95% for $k = 65$ (i.e., a 13 fold generalization). Thus, by using neural features and their corresponding neural activation maps (heatmaps), with *limited* human supervision, we can identify a large subset of samples that contain the relevant spurious or core visual attributes. The neural activation maps for these images can then be used as soft segmentation masks for these visual attributes. We emphasize that the usual method of obtaining such a set would require the manual detection and segmentation of a visual attribute across all images in the training set and is therefore not scalable.

We apply our proposed methodology to the Imagenet dataset: we conducted a Mechanical Turk study using 232 classes of Imagenet and 5 neural features per class. For each neural feature, we obtained its annotation as either core (non-spurious) or spurious for the label. Out of the $232 \times 5 = 1,160$ neural features, 160 were deemed to be spurious by the workers. For 93 classes, at least one spurious neural feature was discovered. Next, for each (class=i, feature=j)

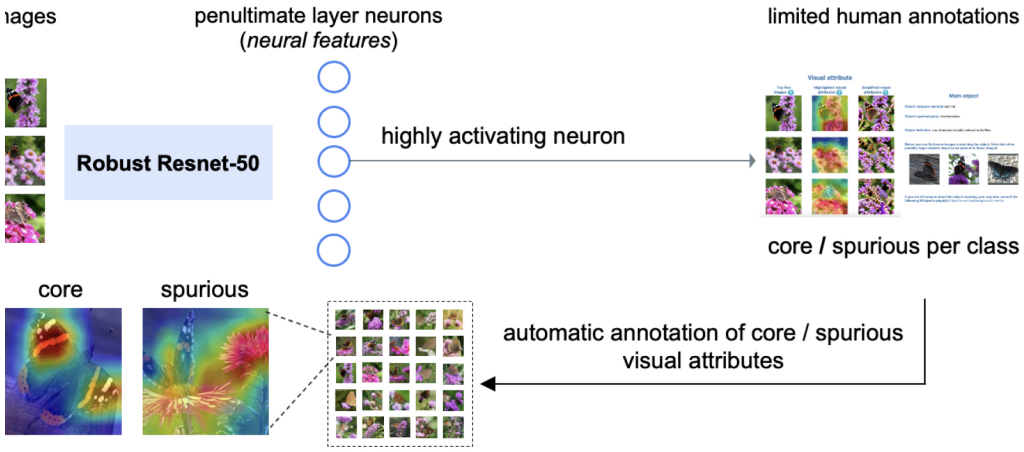


Figure 11.2: Our framework for constructing the *Salient Imagenet* dataset. We annotate *neural features* as core (non-spurious) or spurious using minimal human supervision. We next show that these annotations *generalize* to many more new samples. We highlight core and spurious regions on the input samples using activation maps of these neural features.

pair, we obtained 65 images with class i showing highest activations of the neural feature j and computed their neural activation maps. The union of these images is what we call the *Salient Imagenet* dataset. The dataset contains $52,521$ images with around 226 images per class on average. Each instance in the dataset is of the form $(\mathbf{x}, y, \mathcal{M}^c, \mathcal{M}^s)$ where y is the ground truth class and $\mathcal{M}^s, \mathcal{M}^c$ represent the set of spurious and core masks, respectively (obtained from neural activation maps of neural features). This dataset can be used to test the sensitivity of *any pretrained model* to different image features by corrupting images using their relevant masks and observing the drop in the accuracy.

Ideally, we expect our trained models to show a low drop in accuracy for corruptions in spurious regions and a high drop for that of the core regions. However, using a standard Resnet-50 model, we discover multiple classes for which the model shows a trend contradictory to our expectations: a significantly higher *spurious drop* compared to the *core drop*. As an example, for the class drake (Fig. 11.5), we discover that corrupting any of spurious features results in a significantly higher drop ($> 40\%$) compared to corrupting any of core features ($< 5\%$). We show

many more examples in Supp. 11.K. For various standard models (Resnet-50, Wide-Resnet-50-2, Efficientnet-b4, Efficientnet-b7), we evaluate their accuracy drops due to corruptions in spurious or core regions at varying degrees of corruption noise and find no significant differences between their spurious or core drops suggesting that none of the models significantly differentiates between core and spurious visual attributes in their predictions (Fig. 11.6). In summary, our methodology has the following steps:

- **Step 1:** We select neural features as penultimate-layer neurons of a pretrained robust network and visualize them using a few highly activating samples (Section 11.3.2).
- **Step 2:** Using Mechanical Turk, we annotate neural features as core or spurious (Section 11.3.3).
- **Step 3:** Using neural feature annotations, we automatically annotate core and spurious visual attributes on many more samples without any human supervision (Section 11.3.3).
- **Step 4:** Using another Mechanical Turk study, we show that our methodology generalizes extremely well to identify core/spurious visual attributes on new samples (Section 11.3.4).
- **Step 5:** Applying steps 1-4 to Imagenet, we develop a new dataset called *Salient Imagenet* whose samples, in addition to the class label, are annotated by core and/or spurious masks (Section 11.4).
- **Step 6:** Using such a richly annotated dataset, we assess *core/spurious accuracy* of standard models by adding small noise to spurious/core visual attributes, respectively (Section 11.5).

To the best of our knowledge, this is the first work that introduces a general methodology to discover core and spurious features with limited human supervision by leveraging neurons in the penultimate layer of a robust model as interpretable visual feature detectors. Also, this is the first work that releases a version of Imagenet with annotated spurious and core image features. We believe that this chapter opens future research directions to leverage richly annotated datasets to develop deep models that mainly rely on core and meaningful features in their inferences.

11.2 Related work

Robustness and Interpretability: In recent years, there have been many efforts towards post-hoc interpretability techniques for trained neural networks. Most of these efforts have focused on *local explanations* where decisions about single images are inspected [277, 278, 279, 280, 281, 284, 285, 286, 288, 298, 301, 302, 303, 304]. These include saliency maps [65, 66, 68, 193], class activation maps [282, 283, 287, 320, 321], surrogate models to interpret local decision boundaries such as LIME [322], methods to maximize neural activation values by optimizing input images [323, 324] and finding influential inputs [325]. However, recent work suggests that class activation or saliency maps may not be sufficient to narrow down visual attributes and methods for maximizing neural activations often produce noisy visualizations [284, 289] with changes imperceptible to humans. To address these limitations, recent works [229, 274] show that for robust [20] or adversarially trained models (in contrast to standard models), activation maps are qualitatively more interpretable and optimizing an image directly to maximize a certain neuron produces human perceptible changes in the images which can be useful for visualizing the learned features.

Failure explanation: Most of the prior works on failure explanation either operate on tabular data where interpretable features are readily available [306, 307], language data where domain-specific textual queries can be easily generated [308, 326], images where visual attributes can be collected using crowdsourcing [305, 327] or photorealistic simulation [328]. However, collecting visual attributes via crowdsourcing can be expensive for large datasets. Moreover, these methods require humans to hypothesize about the possible failures and one could miss critical failure modes when the visual attributes used by model are different from the ones hypothesized by humans.

To address these limitations, recent works [318, 319] use the neurons of robust models as the visual attribute detectors for discovering failure modes, thus avoiding the need for crowdsourcing. However, [319] cannot be used to analyze the failures of standard (non-robust) models which achieve significantly higher accuracy and are more widely deployed in practice. Moreover, to discover the spurious feature, they ask MTurk workers to identify the common visual attribute across multiple images without highlighting any region of interest. However, different images may have multiple common attributes even if they come from different classes and this approach may not be useful in such cases. Barlow [318] learns decision trees using misclassified instances to discover leaf nodes with high failure concentration. However, the instances could be correctly classified due to some spurious feature (e.g., background) and Barlow will discover no failure modes in such cases. In contrast, our approach circumvents these limitations and discovers the spurious features for standard models even when they achieve high accuracy.

11.3 A general framework for discovering Spurious Features

Consider an image classification problem $\mathcal{X} \rightarrow \mathcal{Y}$ where the goal is to predict the ground truth label $y \in \mathcal{Y}$ for inputs $\mathbf{x} \in \mathcal{X}$. For each class $i \in \mathcal{Y}$, we want to identify and localize a set of *core or spurious visual attributes* in the training set that can be used by neural networks to predict i:

- **Core Attributes** are the set of visual features that are always a part of the object definition.
- **Spurious Attributes** are the ones that are likely to *co-occur* with the object, but not a part of it (such as food, vase, flowers in Figure 11.1).

A more formal discussion of these abstract definitions can be found in Supp. 10.C. Note that localizing these visual attributes on a large number of images requires significant human annotations and can be very expensive. In this chapter, we propose a general methodology to tackle this issue.

11.3.1 Notation and Definitions

For a trained neural network, the set of activations in the penultimate layer (adjacent to logits) is what we call the *neural feature vector*. Each element of this vector is called a *neural feature*. For an input image and a neural feature, we can obtain the *Neural Activation Map* or *NAM*, similar to CAM [282], that provides a soft segmentation mask for highly activating pixels (for the neural feature) in the image (details in Supp. 11.A.1). The corresponding *heatmap* can then be obtained by overlaying the NAM on top of the image so that the red region highlights the highly activating pixels (Supp. 11.A.2). The *feature attack* [318] is generated by optimizing the input image to increase the values of the desired neural feature (Supp. 10.C).

11.3.2 Extracting, visualizing and selecting neural features

Prior works [274, 318] provide evidence that neural features of a robust model encode human interpretable visual attributes. Building on these works, to identify spurious or core visual attributes used for predicting class i, we identify neural features encoding these attributes. For each image in the Imagenet training set [56], we extract the neural feature vector using a robust Resnet-50 model. The robust model was adversarially trained using the l_2 threat model of radius 3. Each neural feature is visualized using the following techniques:

- **Most activating images**: By selecting 5 images that maximally activate the neural feature, the common visual pattern in these images can be interpreted as the visual attribute encoded in it.
- **Heatmaps (NAMs)**: When there are multiple common visual attributes, heatmaps highlight the region of interest in images and can be useful to disambiguate; e.g., in Figure 11.3 (left panels), heatmap disambiguates that the focus of the neural feature is on the ground (not water or sky).
- **Feature attack**: In some cases, heatmaps may highlight multiple visual attributes. In such cases, the feature attack can be useful to disambiguate; e.g., in Figure 11.3 (right panels), the feature attack indicates that the focus of the neural feature is on the wires (not cougar).

However, the neural feature vector can have a large size (2048 for the Resnet-50 used in this chapter) and visualizing all of these features per class to determine whether they are core or spurious can be difficult. Thus, we select a small subset of these features that are highly predictive of class i and annotate them as core or spurious for i. Prior work [318] selects this

subset by considering the top-k neural features with the highest mutual information (MI) between the neural feature and the *model failure* (a binary variable that is 1 if the model misclassifies $\mathbf{x}$ and 0 otherwise). However, the model could be classifying all the images correctly using some spurious features. In this case, the MI value between the (spurious) neural feature and the model failure will be small and thus this approach will fail to discover such critical failure modes.

In this chapter, we select a subset of neural features *without* using the model failure variable. To do so, we first select a subset of images (from the training set) on which the robust model predicts the class i. We compute the mean of neural feature vectors across all images in this subset denoted by $\overline{\mathbf{r}}(i)$. From the weight matrix $\mathbf{w}$ of the last linear layer of the robust model, we extract the ith row $\mathbf{w}_{i,:}$ that maps the neural feature vector to the logit for the class i. Next, we compute the hadamard product $\overline{\mathbf{r}}(i) \odot \mathbf{w}_{i,:}$. Intuitively, the jth element of this vector $(\overline{\mathbf{r}}(i) \odot \mathbf{w}_{i,:})_j$ captures the mean contribution of neural feature j for predicting the class i. This procedure leads to the following definition:

Definition 15. *The Neural Feature Importance of feature j for class i is defined as:* $\mathrm{IV}_{i,j} = (\overline{\mathbf{r}}(i) \odot \mathbf{w}_{i,:})_j$. *For class i, the neural feature with k^{th} highest IV is said to have the feature rank k.*

We thus select the neural features with the highest-5 importance values (defined above) per class.

11.3.3 Mechanical Turk study for discovering spurious and core features

To determine whether a neural feature j is core or spurious for class i, we conducted a crowd study using Mechanical Turk (MTurk). We show the MTurk workers two panels: One panel

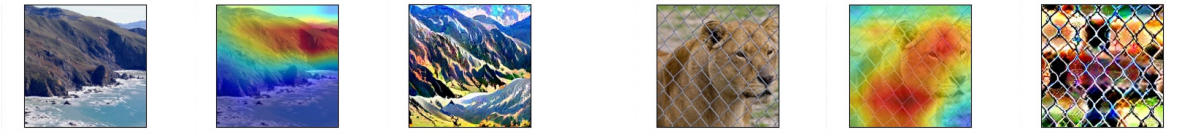


Figure 11.3: On the left, the heatmap suffices to explain that the focus is on ground. But on the right, the heatmap covers both cougar and wires. Feature attack clarifies that the focus is on wires.

visualizes the feature j and the other describes the class i. To visualize j, we show the 5 most activating images (with predicted class i), their heatmaps highlighting the important pixels for j, and the feature attack visualizations [274]. For the class i, we show the object names [329], object supercategory (from [330]), object definition and the wikipedia links. We also show 3 images of the object from the Imagenet validation set. We then ask the workers to determine whether the visual attribute is a part of the main object, some separate objects or the background. They were also required to provide reasons for their answers and rate their confidence on a likert scale from 1 to 5. The design for this study is shown in Figure 11.9 in Supp. 11.C. Each of these tasks (also called Human Intelligence Tasks or HITs) were evaluated by 5 workers. The HITs for which the majority of workers selected either separate objects or background as their responses were deemed to be spurious and the ones with main object as the majority answer were deemed to be core.

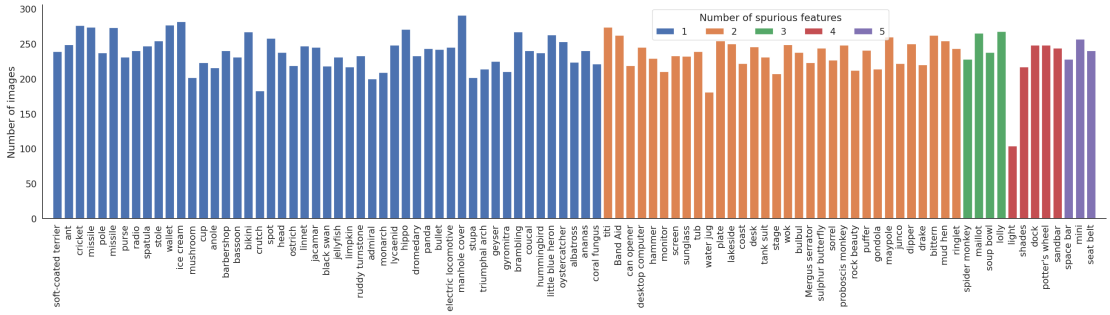
Because conducting a study for all 1000 Imagenet classes can be expensive, we selected a smaller subset of 232 classes (denoted by $\mathcal{T}$) by taking a union of 50 classes with the highest and 50 with the lowest accuracy across multiple models (details in Supp. 11.B). For each class, we selected 5 neural features with the highest feature importance values resulting in $232 \times 5 = 1160$ HITs. For 160 HITs, workers deemed the feature to be spurious for the label. Thus, for each class $i \in \mathcal{T}$, we obtain a set of core and spurious neural features denoted by $\mathcal{C}(i)$ and $\mathcal{S}(i)$, respectively.

11.3.4 Generalization of visual attributes encoded in neural features

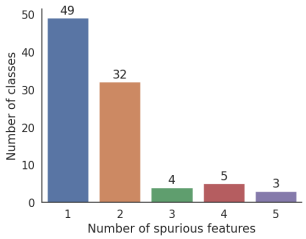
We show that the visual attribute inferred for some neural feature j (via visualizing the top-5 images with predicted class i as discussed in the previous section) *generalizes* extremely well to top-k (with $k \gg 5$) images with the ground truth label i that maximally activate feature j. That is, we can obtain a large set of images containing the visual attribute (encoded in neural feature j) by only inspecting a small set of images. To obtain the desired set, denoted by $\mathcal{D}(i, j)$, we first select training images with label i (Imagenet contains ≈ 1300 images per label). Next, we select $k = 65$ images (5% of 1300) from this set with the highest activations of the neural feature j. We use the Neural Activation Maps (NAMs) for these images as soft segmentation masks to highlight the desired visual attributes.

To validate that the NAMs focus on the desired visual attributes, we conducted another Mechanical Turk study. From the set $\mathcal{D}(i, j)$, we selected 5 images with the highest and 5 with the lowest activations of feature j. We show the workers two panels: The first panel shows images with the highest 5 activations (along with their heatmaps) and the second shows images with the lowest 5 activations (and heatmaps). We then ask them to determine if the focus of heatmaps in both visualizations was on the same visual attribute, different attributes or if either of the two visualizations was unclear. For each spurious feature (160 total), we obtained answers from 5 workers each. For 152 visualizations (95%), majority of the workers selected *same* as the answer, thereby validating that the NAMs focus on the same visual attribute. The design for the study is shown in Figure 11.10 in Supp. 11.E.

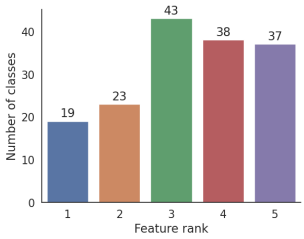
We emphasize that although using $k = 65$ significantly increases the size of the set (i.e., a 13 fold increase over the manually annotated set), the value of k can be changed depending on



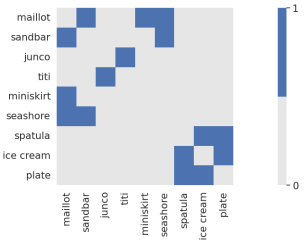
(a) Number of images for different classes with at least 1 spurious feature



(b) Number of classes vs. number of spurious features



(c) Number of classes with spurious feature at various ranks



(d) Number of common spurious neural features between classes

Figure 11.4: In the top row, we show number of images for different classes with at least 1 spurious feature in the Salient Imagenet dataset. Bottom row shows various plots using the dataset.

the number of training images with the desired properties. One can quickly search over various possible values of k by visualizing the 5 images (and their heatmaps) with the highest and lowest activations of the relevant neural feature (in the set of k images) until they both focus on the desired visual attribute.

11.4 The Salient Imagenet dataset

Recall that for each class $i \in \mathcal{T}$, we obtain a set of core and spurious neural features denoted by $\mathcal{C}(i)$ and $\mathcal{S}(i)$, respectively (Section 11.3.3). Then, using the generalization property (Section 11.3.4), we obtain the set $\mathcal{D}(i,j)$ for each $i \in \mathcal{T}$ and $j \in \mathcal{C}(i) \cup \mathcal{S}(i)$. Each $\mathcal{D}(i,j)$ contains 65 images along with the NAM (per image) acting as the soft segmentation mask for the visual attribute encoded in feature j. We note these soft segmentation masks overlap with the desired

visual attribute and may not cover it completely. However, since we corrupt these regions using Gaussian noise which *preserves the content* of the original images (discussed in Section 11.5), they can be useful for evaluating the model sensitivity to various spurious/core visual attributes. If $j \in \mathcal{C}(i)$, these are called *core masks*, otherwise *spurious masks*. The union of all these datasets $\mathcal{D}(i, j)$ is what we call the *Salient Imagenet* dataset. Each instance in the dataset is of the form $(\mathbf{x}, y, \mathcal{M}^c, \mathcal{M}^s)$ where y is the ground truth label while $\mathcal{M}^s/\mathcal{M}^c$ represent the set of spurious/core masks for the image $\mathbf{x}$. The dataset and anonymized Mechanical Turk study results are also available at the associated github repository.

The dataset contains $52,521$ images with around 226 images per class on average. From the MTurk study (Section 11.3.3), 160 features were discovered to be spurious and $1,000$ to be core for relevant classes. We visualize several spurious features (59 total) in Supp. 11.I. Examples of background and foreground spurious features are in Supp. 11.I.1 and 11.I.2, respectively. For 93 classes, we discover at least one spurious feature (the number of images per class shown in Fig. 11.4a). We show a histogram of the number of classes vs. the number of spurious features in Fig. 11.4b. For 3 classes (namely space bar, miniskirt and seatbelt) all 5 features were found to be spurious. This finding suggests that in safety critical applications, it can be unsafe to just use the test accuracy to assess the model performance because the model may be highly accurate on the benchmark for *all the wrong reasons*. We plot the histogram of the number of classes with spurious feature at rank k (from 1 to 5) in Fig. 11.4c. We discover that across all feature ranks, a significant number of spurious features are discovered (minimum 19) and a larger number of spurious features are discovered at higher feature ranks (4 and 5). This suggests that inspecting a larger number of features per class (i.e., > 5) may be necessary for a more thorough analysis of spurious features. In Fig. 11.4d, we show the number of shared spurious features between

different classes (diagonal is zeroed out for brevity). By visualizing common neural features, we diagnose that tree branches are a confusing visual attribute between titi and junco (Supp. 11.J.1); food between plate and icecream (Supp. 11.J.2).

11.5 Evaluating deep models using the Salient Imagenet dataset

In this section, we use the Salient Imagenet dataset to evaluate the performance of several pretrained models on Imagenet. In particular, each set $\mathcal{D}(i, j)$ can be used to test the sensitivity of *any trained model* to the visual attribute encoded with feature j for predicting the class i. We can either blackout (fill with black color) or corrupt (using some noise model) the region containing the attribute (using the soft segmentation masks) for all images in this set and evaluate the accuracy of the model on these corrupted images. However, because the existing Imagenet pretrained models have not explicitly been trained on images with black regions, these images can be characterized as out of distribution of the training set. Thus, it may be difficult to ascertain whether changes in accuracy are due to the distribution shift or the removal of spurious features. Instead, we choose to corrupt the region of interest using Gaussian noise to preserve the content of the original images (which can be useful when the soft segmentation masks are inaccurate). Such Gaussian noise is also known to occur in the real world due to sensor or electronic circuit noise. Since the segmentation masks we derive are soft (i.e., their values lie between 0 and 1, not binary), we use these masks to control the degree of the Gaussian noise corruption across the original image. That is, given image $\mathbf{x}$ and mask $\mathbf{m}$, we add noise $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ using:

$$\mathbf{x} + \sigma\left(\mathbf{z} \odot \mathbf{m}\right), \tag{11.1}$$

where σ is a hyperparameter that can be used to control the degree of corruption in the image. The above equation ensures that regions where the mask $\mathbf{m}$ has values close to 1 are corrupted by noise with the desired standard deviation σ and the ones with values close to 0 suffer little changes.

11.5.1 Core and Spurious accuracy

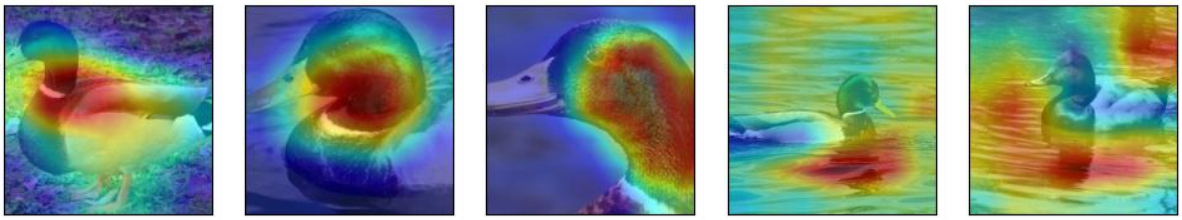
We now introduce concepts of *core* and *spurious* accuracy to evaluate deep models. Informally, the core accuracy is the accuracy of the model only due to the core features of the images and the spurious accuracy is due to the spurious regions. To compute the core accuracy for class i, we first obtain the sets $\mathcal{D}(i,j)$. For some image $\mathbf{x} \in \mathcal{D}(i,j)$, let $\mathbf{m}^{(j)}$ be the mask obtained for some feature j. Next, we take the union of all these sets for $j \in \mathcal{S}(i)$ denoted by $\mathcal{DS}(i)$. Note that for some images in the union, we may have multiple spurious masks but to evaluate the core accuracy, we want to obtain a single mask per image that covers all the spurious attributes. Thus, for the image $\mathbf{x}$, the (p,q)-th element of the single spurious mask $\mathbf{s}$ (or, the single core mask $\mathbf{c}$), is computed by taking the element-wise max over all the relevant masks:

$$\mathbf{s}_{p,q} = \max_{j \in \mathcal{S}(i),\ \mathbf{x} \in \mathcal{D}(i,j)} \mathbf{m}^{(j)}_{p,q}, \qquad \mathbf{c}_{p,q} = \max_{j \in \mathcal{C}(i),\ \mathbf{x} \in \mathcal{D}(i,j)} \mathbf{m}^{(j)}_{p,q} \tag{11.2}$$

Definition 16. ***(Core Accuracy)*** *We define the Core Accuracy or* $\mathrm{acc}^{(\mathcal{C})}$ *as follows:*

$$\mathrm{acc}^{(\mathcal{C})} = \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \mathrm{acc}^{(\mathcal{C})}(i), \quad \mathrm{acc}^{(\mathcal{C})}(i) = \frac{1}{|\mathcal{DS}(i)|} \sum_{\mathbf{x} \in \mathcal{DS}(i)} \mathbb{1}\left(h\left(\mathbf{x} + \sigma\left(\mathbf{z} \odot \mathbf{s}\right)\right) = y\right)$$

We acknowledge that our definition of core accuracy is *incomplete* because the set of spu-



(a) Core, **-1.5%** (b) Core, **-4.6%** (c) Core, **-0%** (d) Spur. **-41.5%** (e) Spur. **-46.1%**

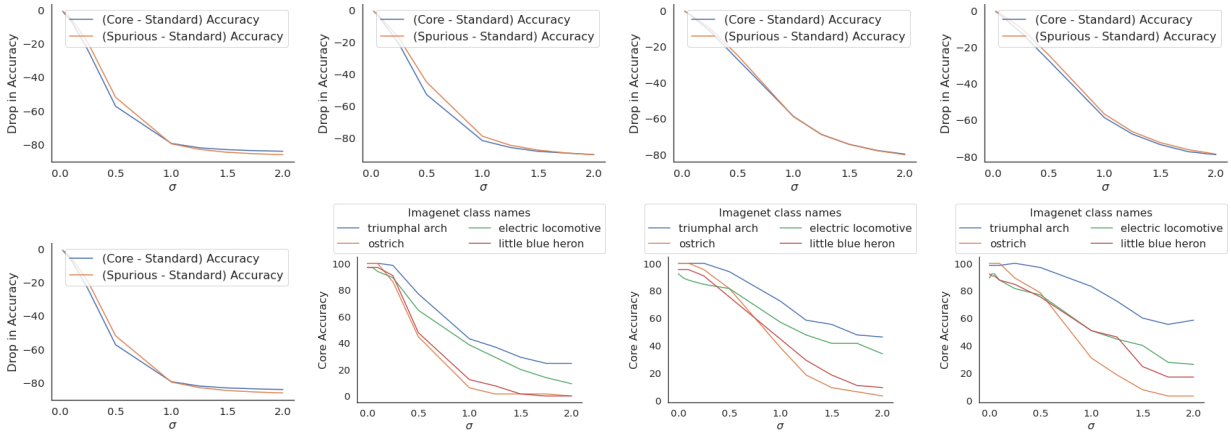
Figure 11.5: Each image denotes the heatmap for different neural features for the class drake. Using a standard Resnet-50 model, (clean accuracy of 95.4%), we observe a drop of *at least* 41.5% by adding gaussian noise with $\sigma = 0.25$ to spurious masks while a drop of *at most* 4.6% for core masks. This shows that the model heavily relies on spurious features in its predictions for this class.

rious visual attributes discovered using our framework may not cover *all* spurious attributes in the dataset. Finally, note that the defined core accuracy is a function of the noise parameter (σ) used to corrupt spurious regions. The Spurious Accuracy or $\text{acc}^{(\mathcal{S})}(i)$ can be computed similarly by replacing $\mathcal{DS}(i)$ with $\mathcal{DC}(i) = \cup_{j \in \mathcal{C}(i)} \mathcal{D}(i,j)$, and $\mathbf{s}$ with $\mathbf{c}$ in (11.2). Observe that $\mathcal{DS}(i) \neq \mathcal{DC}(i)$ in general because $\mathcal{D}(i,j)$ may contain different sets of images for different j's. Thus, the standard accuracy of the model (i.e. without adding any noise) on the two sets pf $\mathcal{DS}(i)$ and $\mathcal{DC}(i)$ can be different. We want our trained models to show a low degradation in performance when noise is added to spurious regions (i.e. high core accuracy) and a high degradation when corrupting core regions (i.e. low spurious accuracy).

11.5.2 Results from evaluating deep models on the Salient Imagenet dataset

In this section, we test the sensitivity of several models to various spurious/core features.

Testing model sensitivity to spurious features. We test the sensitivity of a standard Resnet-50 model to different spurious and core features (one feature at a time) by corrupting images from the datasets $\mathcal{D}(i,j)$ and evaluating the drop in model accuracy. We use $\sigma = 0.25$



(a) Resnet-50 (b) Wide-Resnet-50-2 (c) Efficientnet-b4 (d) Efficientnet-b7

Figure 11.6: In the top row, we observe a similar drop in accuracy irrespective of whether noise is added to spurious or core regions suggesting that trained models do not differentiate between the spurious and core regions for predicting the desired class. In the bottom row, we show core accuracy vs. noise parameter σ for several classes. In particular, for both classes "triumphal arch" and "ostrich", the model has a standard accuracy of 100% (at $\sigma = 0$). However, the core accuracy for triumphal arch is high even at $\sigma = 2.0$ ($\approx 40\%$) while that for ostrich is almost 0%.

((11.1)) because we observe that it preserves the content of the images so we would expect the model prediction to remain unchanged. In Figure 11.1, we show multiple examples of spurious features discovered using this procedure. In Figure 11.5, we show a class (namely drake) on which the model has a high clean accuracy (95.4%). However, it exhibits a large drop in performance when any of the spurious regions (e.g. water) are corrupted while a small drop is observed when any of the core regions (e.g. bird's head) are corrupted. This highlights that **the model relies on mostly spurious features in its predictions** for this class. Out of the 93 classes with at least one spurious feature, we discovered 34 classes such that the accuracy drop due to some spurious feature was at least 20% higher than due to some core feature. In Supp. 11.K, we show visualizations of core and spurious features for 15 of these classes. On 11 of these 15 classes, the model has $> 95\%$ accuracy. Since the l_2 norm of noise perturbations can be different for different features (e.g. due to different l_2 norms of their activation maps), we also

evaluate accuracy drops due to corrupting spurious/core features when the mean l_2 perturbations are similar and find similar trends (details in Supp. 11.K).

Comparing core and spurious accuracy of trained models. Next, we compute the core and spurious accuracy of four standard (i.e. not adversarially trained) pretrained models namely, Resnet-50 [1], Wide Resnet-50-2 [161], Efficientnet-b4 and Efficientnet-b7 [331]. In Figure 11.6 (top row), we plot the drop in accuracy when noise is added to spurious regions (i.e., core accuracy - standard accuracy) and also to core regions (i.e., spurious accuracy - standard accuracy). We observe a similar drop in performance for all values of σ and for all trained models. This suggests that trained models do not differentiate between spurious and core regions of the images for predicting an object class. In some sense, this is expected because in the standard Empirical Risk Minimization (ERM) paradigm, models have never explicitly been trained to make such differentiation, suggesting that providing additional supervision during training (for example, segmenting core/spurious regions as provided in Salient Imagenet or diversifying the training dataset) may be essential to train models that achieve high core accuracy. In Figure 11.6 (bottom row), we plot the core accuracy for 4 different classes as the noise level σ increases. In particular, consider the two classes "triumphal arch" and "ostrich" that both have standard accuracy of 100% (at $\sigma = 0$). However, the core accuracy for triumphal arch is high even at $\sigma = 2.0$ ($\approx 40\%$) while that for ostrich is almost 0%. This provides further evidence that **the standard accuracy alone is not a reliable measure for model performance in the real world** because the core accuracy for two classes with the same standard accuracy can be very different. We believe having richly annotated training datasets such as Salient Imagenet can lead to training reliable deep models that mainly rely on core and informative features in their predictions.

Supplementary Material

11.A Visualizing the neural features of a robust model

11.A.1 Neural Activation Map (NAM)

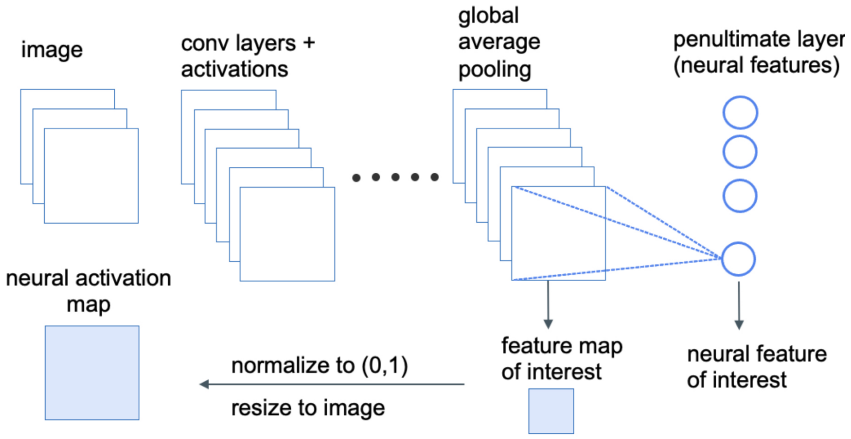


Figure 11.7: Neural activation map generation

Figure 11.7 describes the Neural Activation Map generation procedure. To obtain the neural activation map for feature j, we select the feature map from the output of the tensor of the previous layer (i.e the layer before the global average pooling operation). Next, we simply normalize the feature map between 0 and 1 and resize the feature map to match the image size, giving the neural activation map.

11.A.2 Heatmap

Heatmap can be generated by first converting the neural activation map (which is grayscale) to an RGB image (using the jet colormap). This is followed by overlaying the jet colormap on top of the original image using the following few lines of code:

```
import cv2


def compute_heatmap(img, fam):
    hm = cv2.applyColorMap(np.uint8(255 * nam),
        cv2.COLORMAP_JET)
    hm = np.float32(hm) / 255
    hm = hm + img
    hm = hm / np.max(hm)
    return hm
```

11.A.3 Feature attack

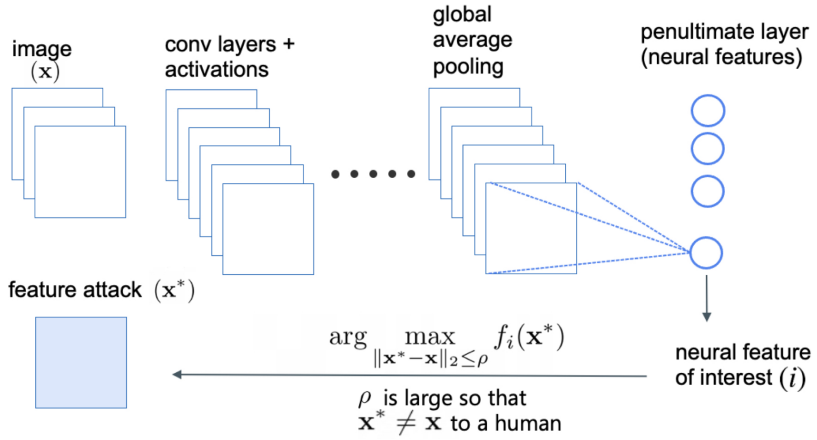


Figure 11.8: Feature attack generation

In Figure 11.8, we illustrate the procedure for the feature attack. We select the feature we are interested in and simply optimize the image to maximize its value to generate the visualization. ρ is a hyperparameter used to control the amount of change allowed in the image. For

optimization, we use gradient ascent with step size = 40, number of iterations = 25 and $\rho = 500$.

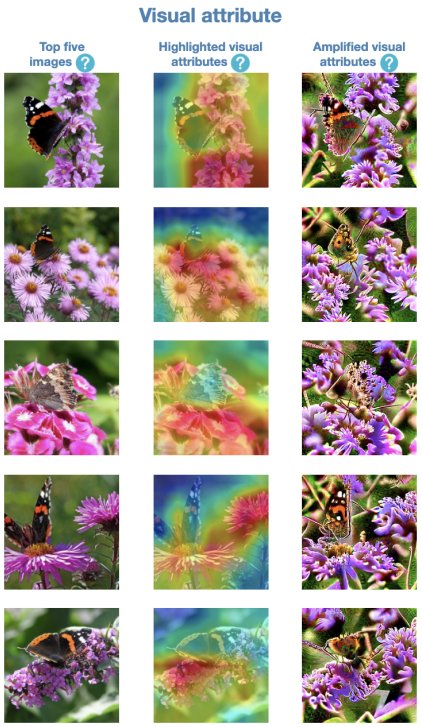
11.B Selecting the classes for discovering spurious features

Because conducting a Mechanical Turk (MTurk) study for discovering the spurious features for all 1000 classes of Imagenet can be expensive, we selected a smaller subset of classes as follows. Using some pretrained neural network h, for each class i in Imagenet, we obtain groups of images with the label i (called label grouping) and prediction i (i.e h predicts i, called prediction grouping) giving 2000 groups. For each group, we compute their accuracy using the network h. For each grouping (label/prediction), we selected 50 classes with the highest and 50 with the lowest accuracy giving 100 classes per grouping and take the union. We used two pretrained neural networks: standard and robust Resnet-50 resulting in total 232 classes.

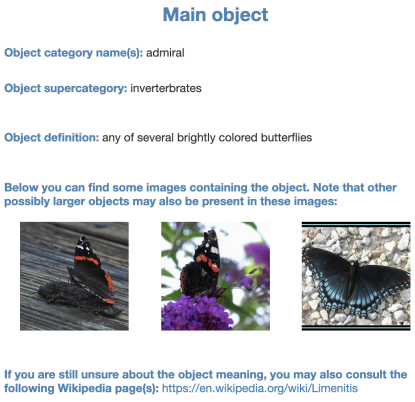
11.C Mechanical Turk study for discovering spurious features

The design for the Mechanical Turk study is shown in Figure 11.9. We showed the workers two panels side by side. The left panel visualizes a neuron (from a robust resnet-50 model trained on Imagenet) and the right panel describes an object class from Imagenet.

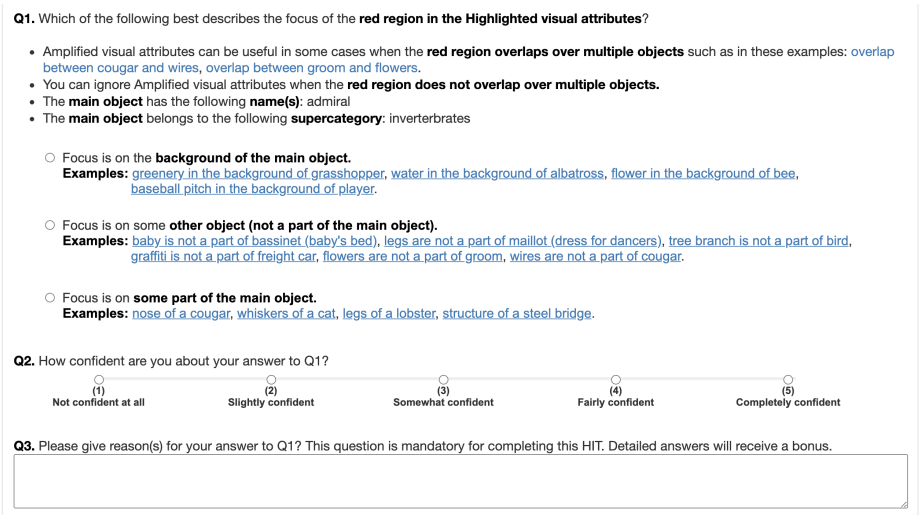
The left panel visualizing the neuron is shown in Figure 11.9a. To visualize the neuron, we first select the subset of images for which the robust model predicts the object class (given on the right). We show three sections: Top five images (five images with highest activation values of the neuron in the subset), Highlighted visual attributes (heatmaps computed using the neural activation maps as described in Section 11.A.2) and Amplified visual attributes (feature attack computed by optimizing the top five images to maximize the neuron (Section 11.A.3).



(a) Visual attribute



(b) Main object



(c) Questionnaire

Figure 11.9: Mechanical Turk study for discovering spurious features

The right panel describing the object class is shown in Figure 11.9b. We show the object category name (also called synset) from the wordnet heirarchy [329], object supercategory (from

[330]), object definition (also called gloss) and the relevant wikipedia links describing the object. We also show 3 images of the object from the Imagenet validation set.

The questionnaire is shown in Figure 11.9c. We ask the workers to determine whether they think the visual attribute (given on the left) is a part of the main object (given on the right), some separate object or the background of the main object. We also ask the workers to provide reasons for their answers and rate their confidence on a likert scale from 1 to 5. The visualizations for which majority of workers selected either separate object or background as the answer were deemed to be spurious. In total, we had 205 unique workers, each completing 5.66 tasks (on average). Workers were paid \$0.1 per HIT, with an average salary of \$8 per hour.

11.D Quality control

Only allowed workers with at least 95% approval rates and minimum of 1000 HITs were allowed to complete our tasks. Additionally, short and generic answers to Q3 were rejected.

11.E Mechanical Turk study for validating heatmaps

The design for the Mechanical Turk study is shown in Figure 11.10. Given a subset of the training set (constructed as described in Section 11.3.4), our goal is to validate whether the heatmaps focus on the same visual attribute for all images in the subset.

Recall that this subset was constructed for each spurious feature discovered using the previous crowd study (Section 11.C). For each spurious feature, we already know both the object class and the neuron index of the robust model. We construct the subset of training set by first selecting the subset of images for which the *label is the object class*. Note that this is different

from the previous crowd study where the prediction of the robust model is the object class. Next, we select the top-65 images from this subset where the desired neuron has the highest activation values.

We showed the workers two panels side by side, each panel visualizing the same neuron. The left panel visualizing the neuron is shown in Figure 11.10a. The visualization shows the images with the highest-5 neural activations in the subset and corresponding heatmaps. The right panel visualizing the neuron is shown in Figure 11.10b. That shows the images with the lowest-5 neural activations in the subset and corresponding heatmaps.

The questionnaire is shown in Figure 11.10c. We ask the workers to determine whether they think the focus of the heatmap is on the same object (in both the left and right panels), different objects or whether they think the visualization in either of the sections is unclear . They were given the option to choose from these four options: same, different, Section A is unclear and Section B is unclear. Same as in the previous study (Section 11.C), we ask the workers to provide reasons for their answers and rate their confidence on a likert scale from 1 to 5.

The visualizations for which majority of workers selected same as the answer were deemed to be validated i.e for this subset of 65 images, we assume that the neural activation maps focus on the same visual attribute.

11.F Discovering spurious visual attributes in other image datasets using our framework

In this chapter, we consider the problem of classification on Imagenet and introduce a methodology to discover core/spurious features. Our framework can be applied to discover the spurious/-

core visual attributes of other image classification datasets using the following steps:

(i) Train a robust model using the l_2 norm of some large radii (we use 3 in this chapter).

(ii) To discover spurious visual attributes for some class i in the dataset, select the subset of images from the training dataset on which the robust model predicts i.

(iii) Select neural features from the penultimate layer of the robust model with the k highest importance values (as defined in Section 15). We use $k = 5$ in this chapter. We note that other layers of the network may also provide informative neural features in some applications.

(iv) Visualize each neural feature using m highly activating images, their heatmaps and feature attack (Section 11.3.2). We use $m = 5$ in this chapter. Note that both k and m may be changed depending on the problem at hand.

(v) Create a Mechanical Turk HIT per (class=i, neural feature=j) pair such that the HIT contains both the visualization for feature j and also the details about the class i (similar to Figure 11.9) and obtain worker annotations.

The HITs for which the majority of the workers annotate the neural feature to be spurious for class are deemed to be the spurious visual attributes. Then, simply one can use any Heatmap methods such as CAM to automatically highlight core and spurious masks on many other samples in the dataset. At this point, we recommend another MTurk study to validate the quality of highlighted core and spurious masks on new samples.

11.G Analyzing the reasons provided by Mechanical Turk workers

Understanding the reasoning behind the answers of MTurk workers is a general and challenging problem in all crowd studies. To understand the reasons behind the answers provided by our MTurk workers, we have taken the following steps: (i) workers were required to give reasons for their answers to complete each task and (ii) they were explicitly instructed to use heatmaps (called highlighted visual attributes in the study) for their answers and use feature attack visualization (called amplified visual attributes in the study) for their answers when the heatmaps are unclear. For the crowd study for discovering spurious features (Section 11.3.3), we have observed more than 1200 mentions of the word "highlighted" and only 29 mentions of the word "amplified" suggesting that the workers use highlighted visual attributes (heatmaps) significantly more than the amplified visual attributes (feature attack). Some example descriptions provided by workers are given below in Figures 11.11 and 11.12.



(a) **Section A**: Images with the highest activation values for the neuron of interest in the subset.



(b) **Section B**: Images with the lowest activation values for the neuron of interest in the subset.

Q1. Which of the following best describes the focus of the **red region in the Highlighted visual attributes**?

- In both Section A and Section B, focus is on the **same visual attribute.**
- In Section B, focus is on **different visual attribute** compared to Section A.
- The focus of red region is **unclear in Section A.**
- The focus of red region is **unclear in Section B.**

Q2. How confident are you about your answer to Q1?

(1) Not confident at all | (2) Slightly confident | (3) Somewhat confident | (4) Fairly confident | (5) Completely confident

Q3. Describe the visual attributes in Section A and Section B for your answer to Q1? This question is mandatory for completing this HIT. Detailed answers will receive a bonus.

(c) Questionnaire.

Figure 11.10: Mechanical Turk study for validating heatmaps



Figure 11.11: Visualization of feature **625** for class **barn spider** (class index: **73**).
Reasons for the answers provided by workers (all workers answered main object): (i) The red region in the Highlighted visual attributes focuses on the leg of the barn spider that is the main object, (ii) Legs of spider are mainly focused. (iii) Focus on spider, (iv) The red is on the legs of the barn spider, (v) Focus is on another part of the main object, such as the barn spider's legs.

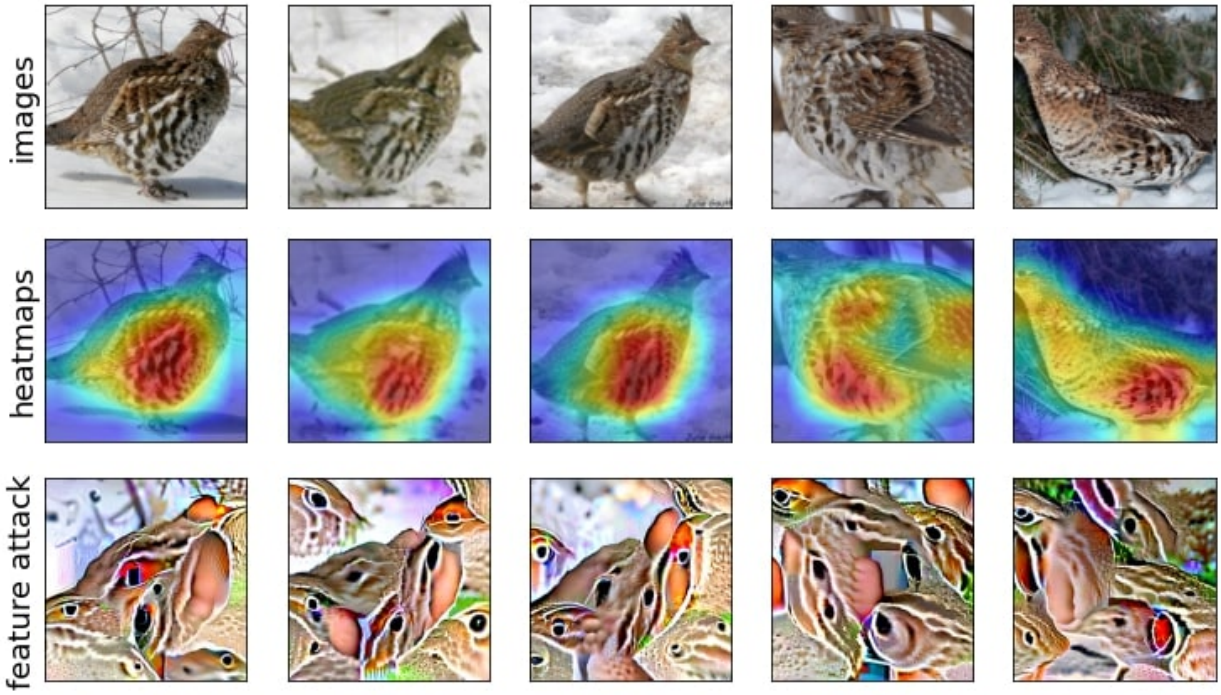


Figure 11.12: Visualization of feature **957** for class **ruffed grouse** (class index: **82**).
Reasons for the answers provided by workers (all workers answered main object): (i) The red region in the Highlighted visual attributes focuses on the feathers of the ruffed grouse that is the main object, (ii) focus is highlighting the bird's wing, (iii) The focus is on the feathers of the bird, (iv) Focus is on the bird, (v) All of the red parts are within the object

11.H Additional results from evaluating deep models on the Salient Imagenet dataset

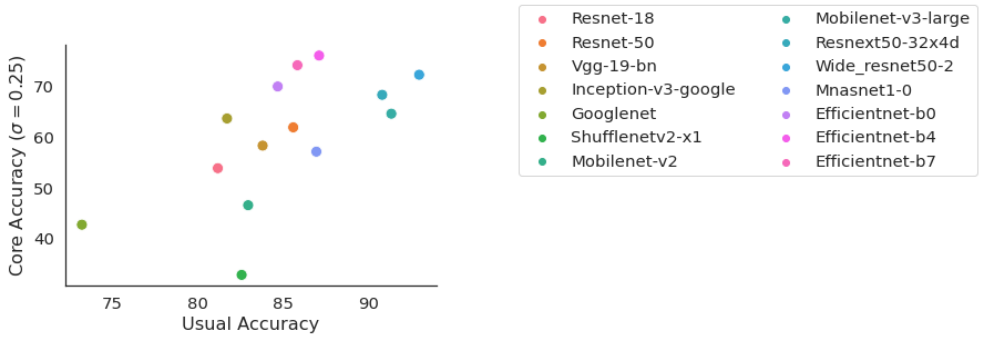
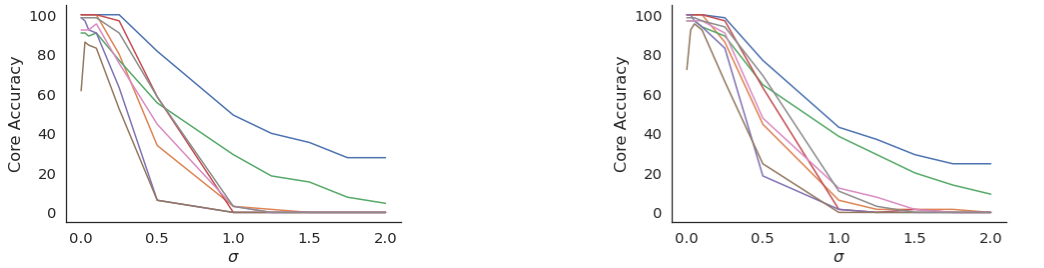
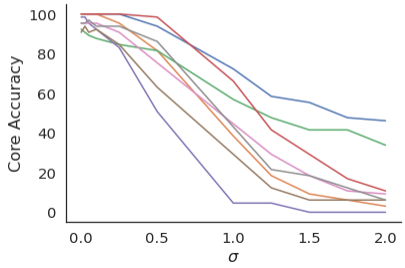


Figure 11.13: Comparing between the core accuracy (using $\sigma = 0.25$) and usual accuracy of different Imagenet trained models.

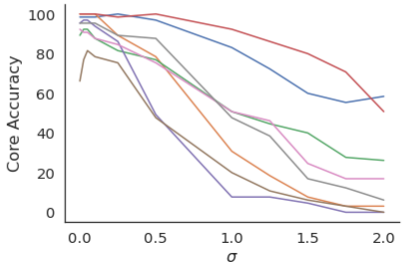


(a) Resnet-50

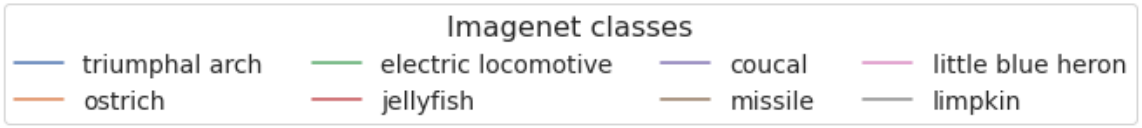
(b) Wide-Resnet-50



(c) Efficientnet-b4



(d) Efficientnet-b7



(e) Legend

Figure 11.14: We plot the core accuracy for different classes as noise level σ increases. For some classes, core accuracy is high even at large σ, but for most it decreases rapidly (e.g., triumphal arch, jellyfish on Efficientnet-b7).

11.I Examples of spurious features

11.I.1 Background spurious features

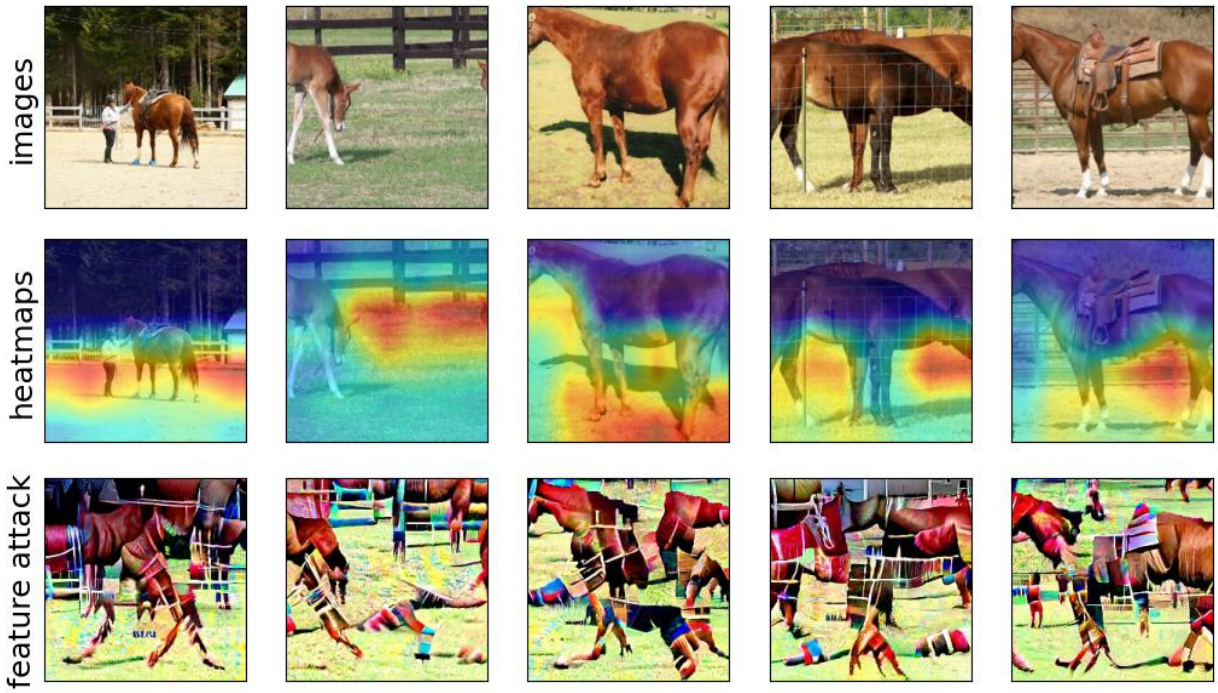


Figure 11.15: Visualization of feature **230** for class **sorrel** (class index: **339**).
Train accuracy using Standard Resnet-50: **99.385**%.
Drop in accuracy when gaussian noise is added to the highlighted (red) regions: **-12.308**%.

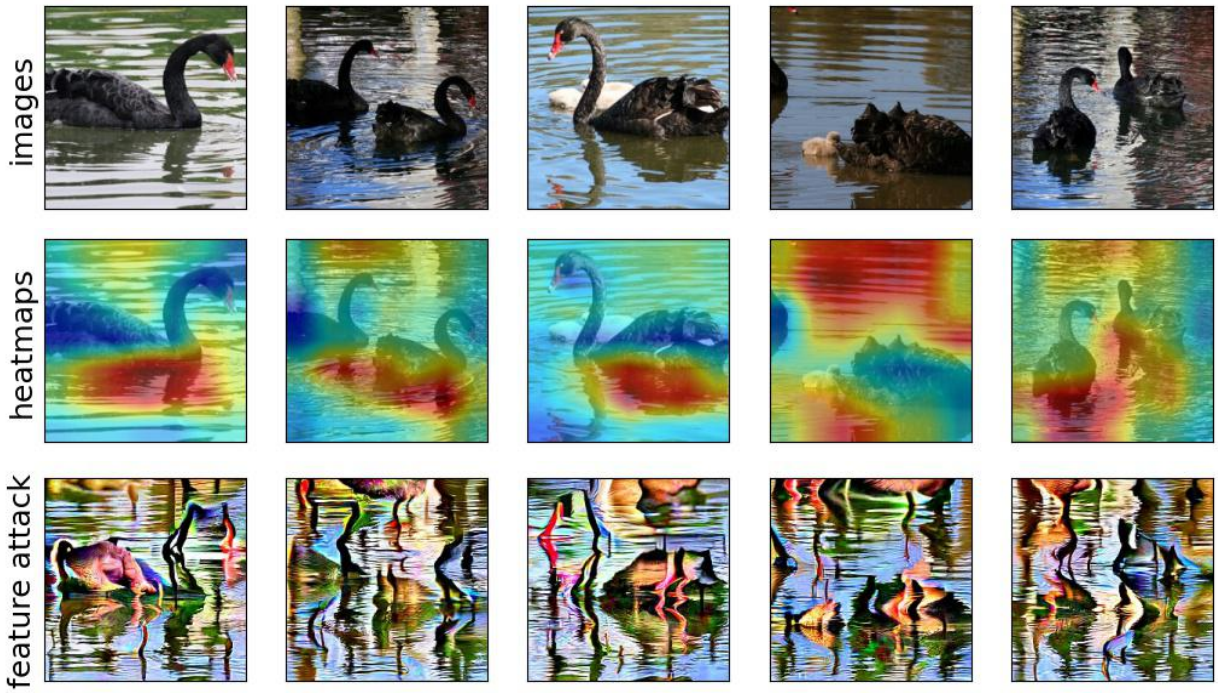


Figure 11.16: Visualization of feature **925** for class **black swan** (class index: **100**).
Train accuracy using Standard Resnet-50: **98.769**%.
Drop in accuracy when gaussian noise is added to the highlighted (red) regions: **-4.615**%.

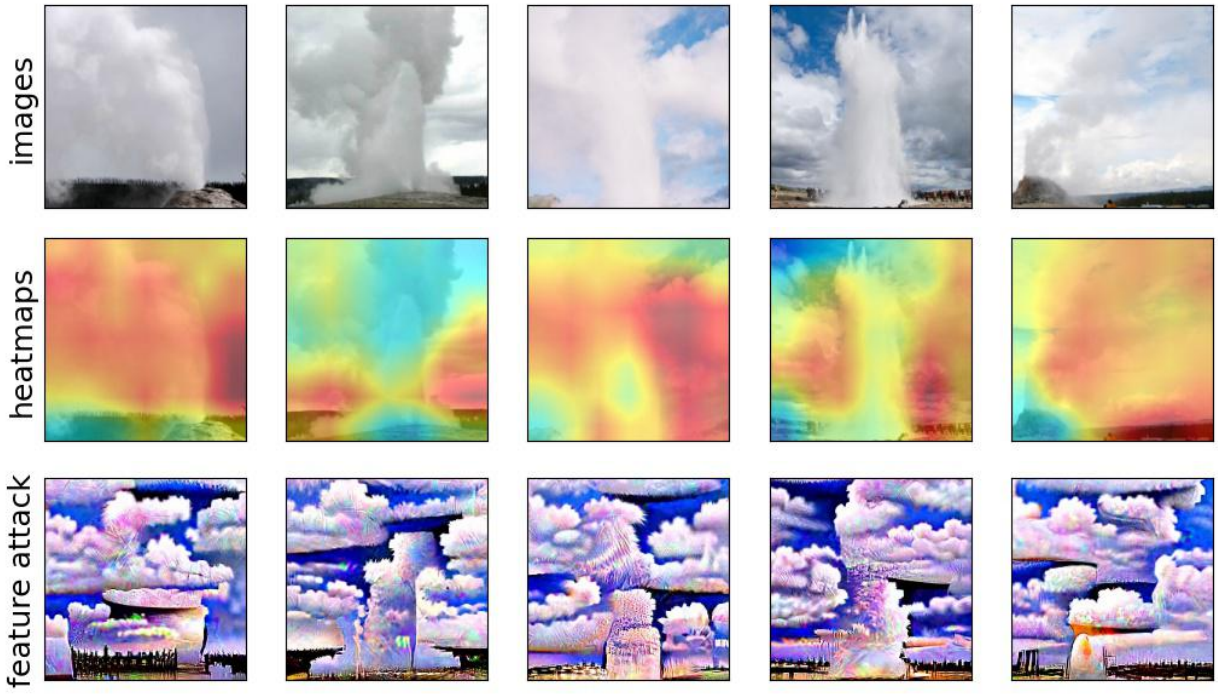


Figure 11.17: Visualization of feature **223** for class **geyser** (class index: **974**).
Train accuracy using Standard Resnet-50: **97.923**%.
Drop in accuracy when gaussian noise is added to the highlighted (red) regions: **-20.0**%.

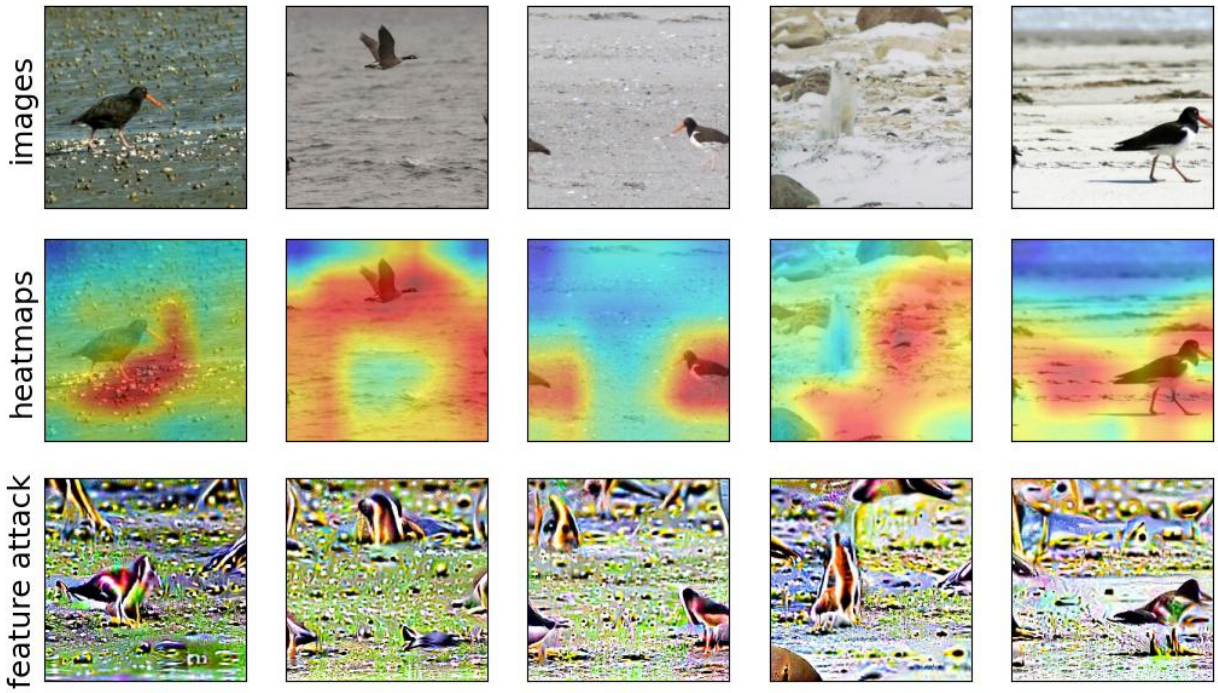


Figure 11.18: Visualization of feature **1468** for class **oystercatcher** (class index: **143**).
Train accuracy using Standard Resnet-50: **97.769**%.
Drop in accuracy when gaussian noise is added to the highlighted (red) regions: **-21.539**%.

11.I.2 Foreground spurious features

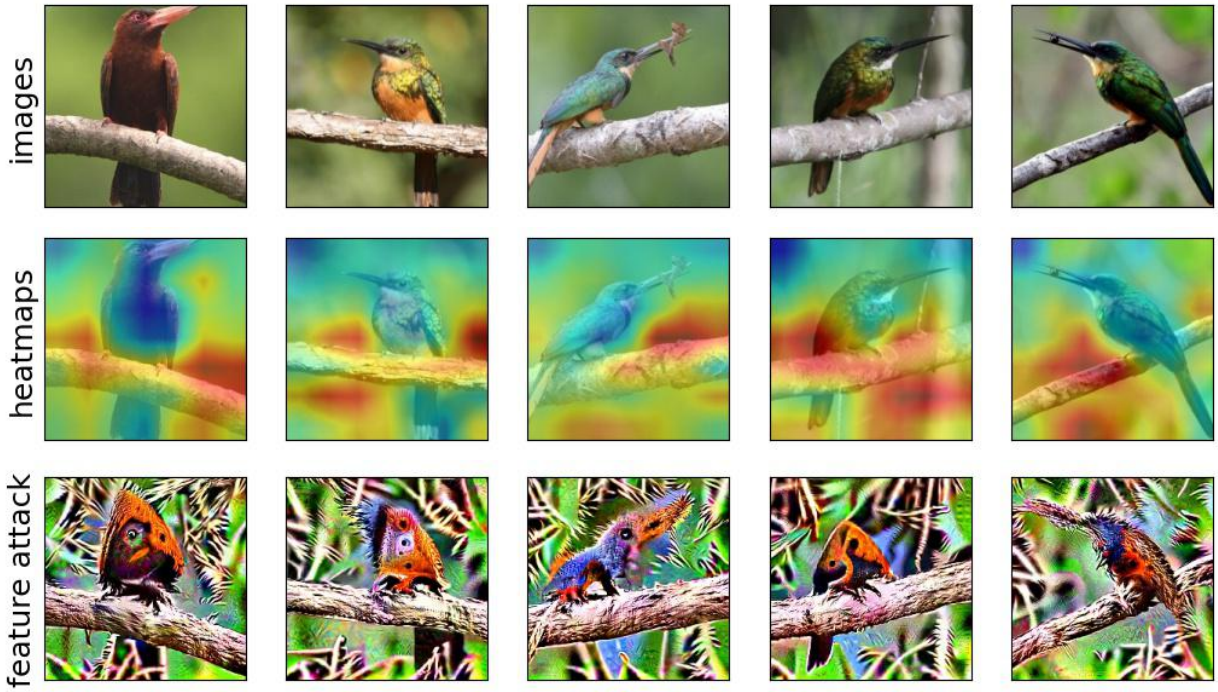


Figure 11.19: Visualization of feature **1556** for class **jacamar** (class index: **95**).
Train accuracy using Standard Resnet-50: **97.000**%.
Drop in accuracy when gaussian noise is added to the highlighted (red) regions: **-21.538**%.



Figure 11.20: Visualization of feature **189** for class **potter's wheel** (class index: **739**).
Train accuracy using Standard Resnet-50: **95.615**%.
Drop in accuracy when gaussian noise is added to the highlighted (red) regions: **-21.538**%.

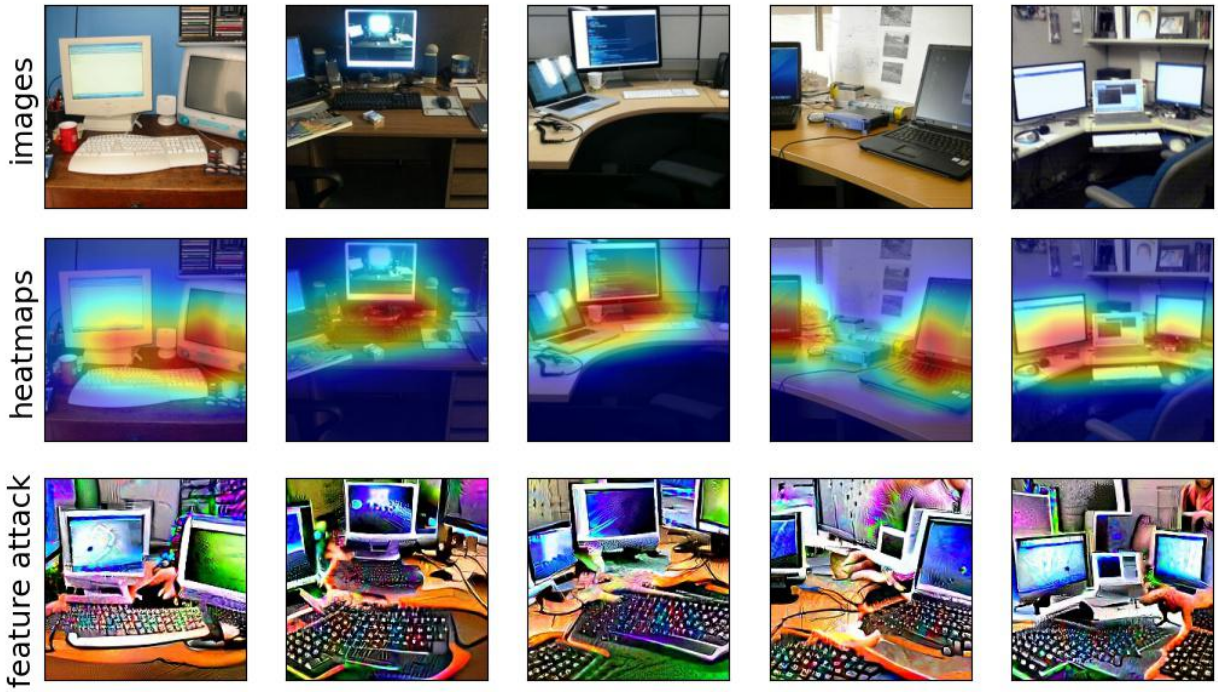


Figure 11.21: Visualization of feature **1832** for class **desk** (class index: **526**).
Train accuracy using Standard Resnet-50: **69.769**%.
Drop in accuracy when gaussian noise is added to the highlighted (red) regions: **-7.692**%.

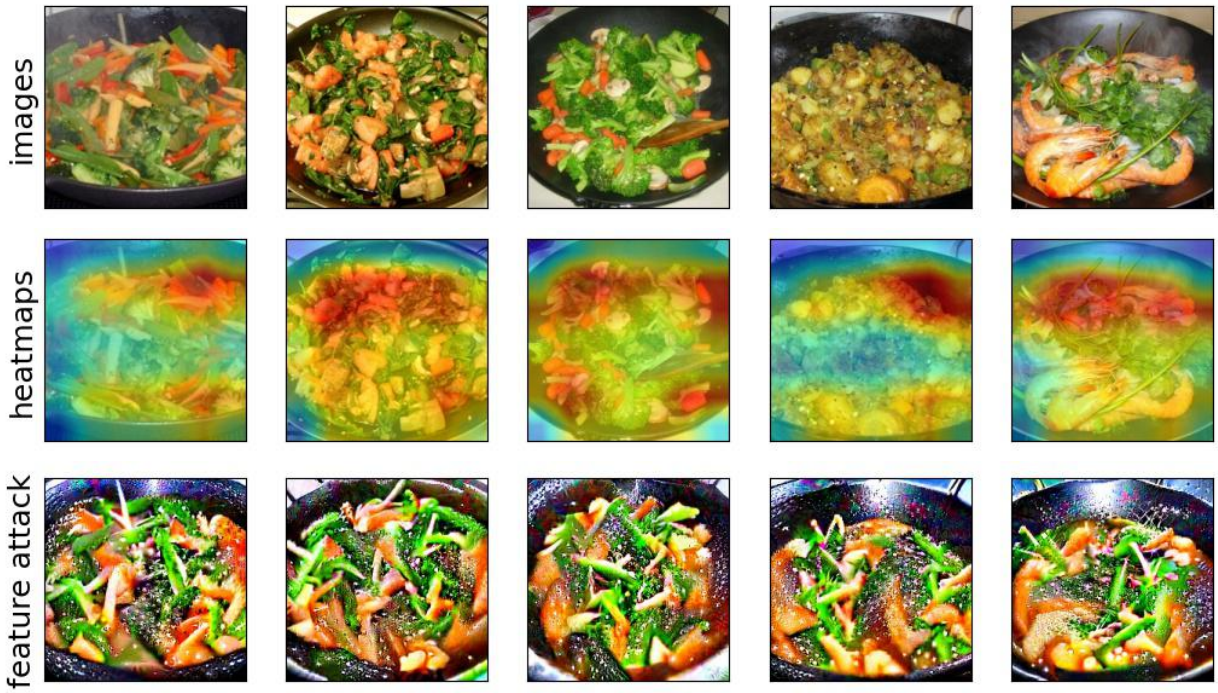


Figure 11.22: Visualization of feature **895** for class **wok** (class index: **909**).
Train accuracy using Standard Resnet-50: **71.077**%.
Drop in accuracy when gaussian noise is added to the highlighted (red) regions: **-46.154**%.

11.J Diagnosing the confusing visual attribute between different classes

11.J.1 Feature index: 1541



Figure 11.23: Visualization for class **junco** (class index: **13**).

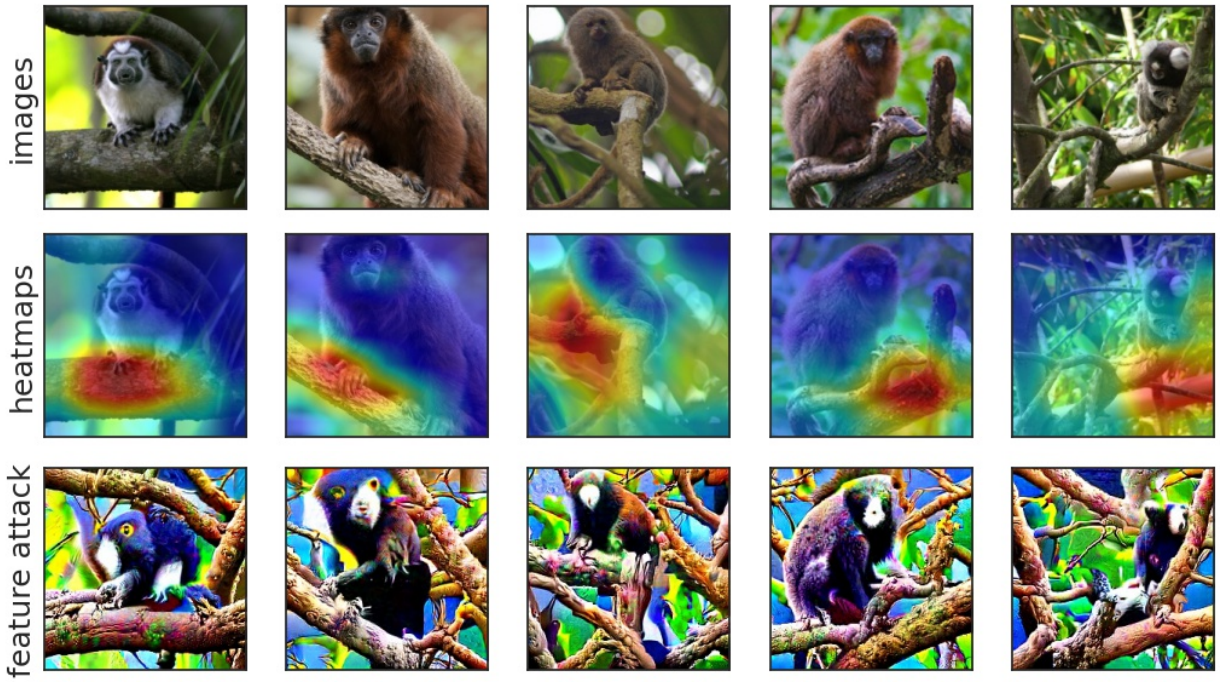


Figure 11.24: Visualization for class **titi** (class index: **380**).

11.J.2 Feature index: 2025



Figure 11.25: Visualization for class **plate** (class index: **923**).

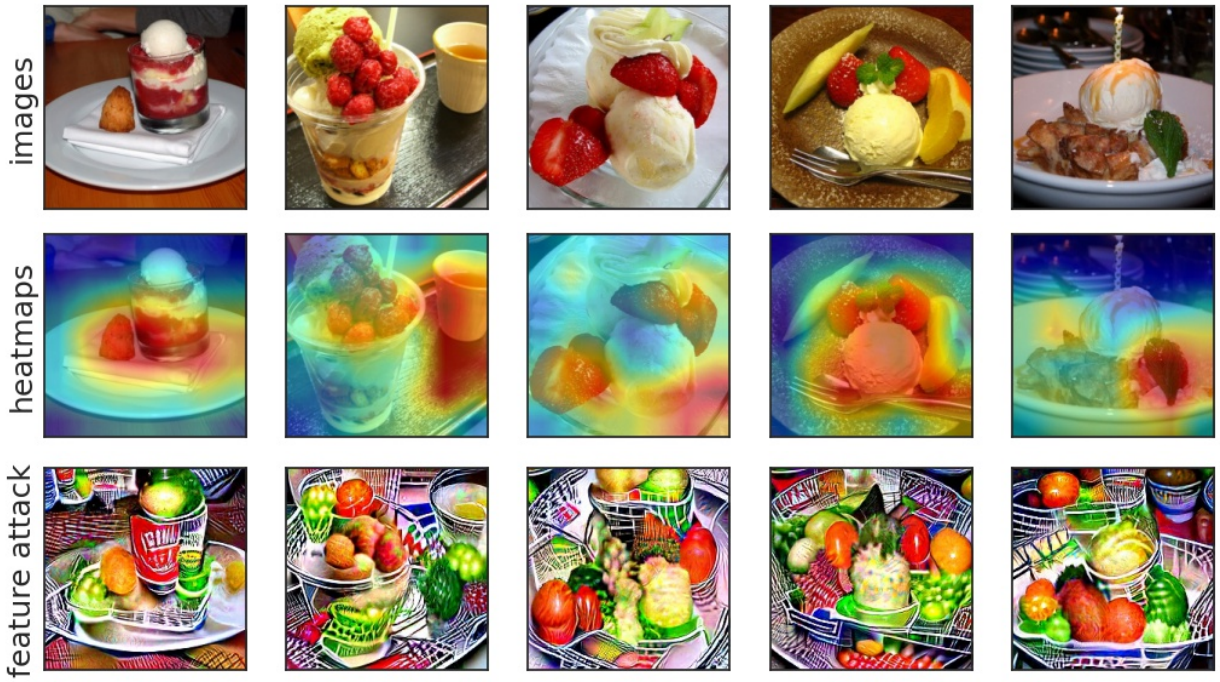


Figure 11.26: Visualization for class **icecream** (class index: **928**).

11.K Comparing between the accuracy drops due to spurious and core neural features

For some class i, we want to test the sensitivity of a trained model to the visual attribute encoded in the feature j for predicting the class i. To this end, we add Gaussian noise to highly activating regions for images in the dataset $\mathcal{D}(i, j)$ and evaluate the drop in the accuracy of the model being inspected for failure. In particular, we show the drop in the accuracy for different classes by perturbing both core and spurious neural features using Gaussian noise with $\sigma = 0.25$.

However, for some of the classes, given a σ value, the mean of noise perturbation magnitudes imposed on images in the dataset $\mathcal{D}(i, j)$ (referred to as "mean ℓ_2 perturbation") can be different for different features. This is because the activation maps of different neural features can have different l_2 norms. Thus, for such classes, we also compute accuracy drops using multiple values of σ: $[0.30, 0.35, \ldots, 0.60]$. Then, for the features whose mean l_2 perturbations were smaller, we choose the smallest value of σ (referred to as σ_s) that results in a mean l_2 perturbation similar to (or even slightly higher than) that of the other neural feature (that initially had higher mean l_2 perturbation). Thus, for each neural feature, we show the accuracy drops using both $\sigma = 0.25$ and σ_s in the figure captions in this section.

11.K.1 Class name: Ostrich, Train accuracy (Standard Resnet-50): 98.615%

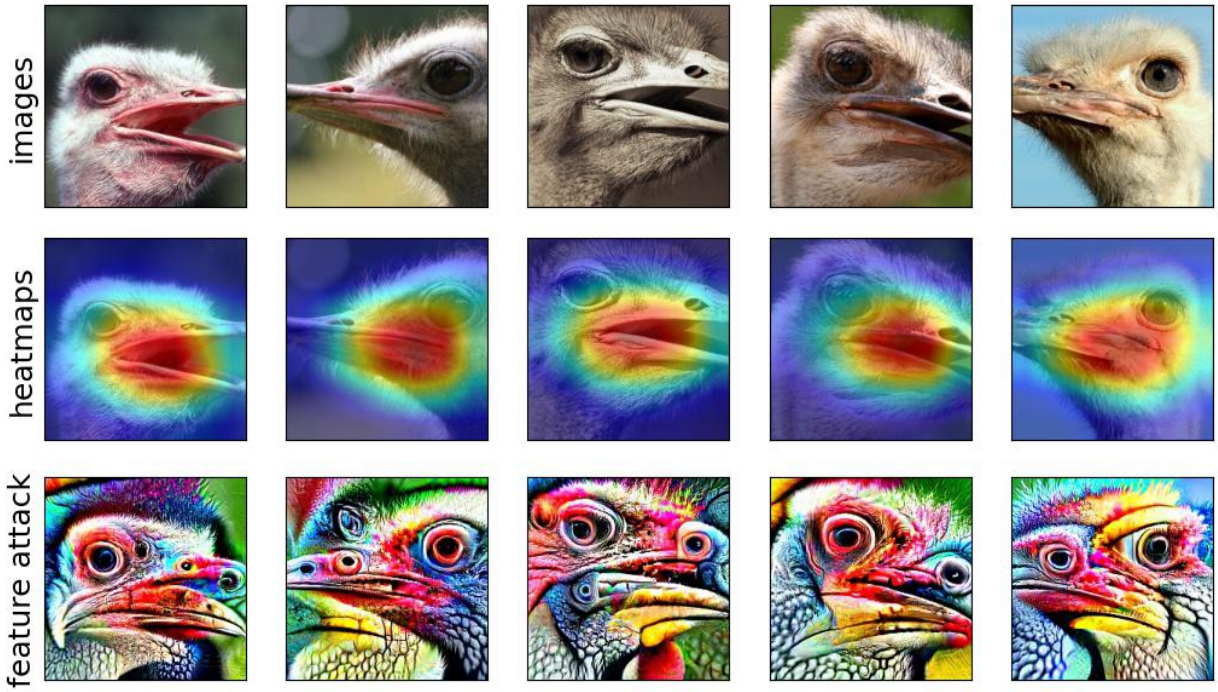


Figure 11.27: Visualization of feature **1964** for class **ostrich** (class index: **9**).
Feature label as annotated by Mechanical Turk workers: **core**.
Using $\sigma = 0.25$, drop in accuracy: **0.0**%, mean l_2 perturbation: **35.782**.
Using $\sigma = 0.35$, drop in accuracy: **-6.154**%, mean l_2 perturbation: **47.815**.

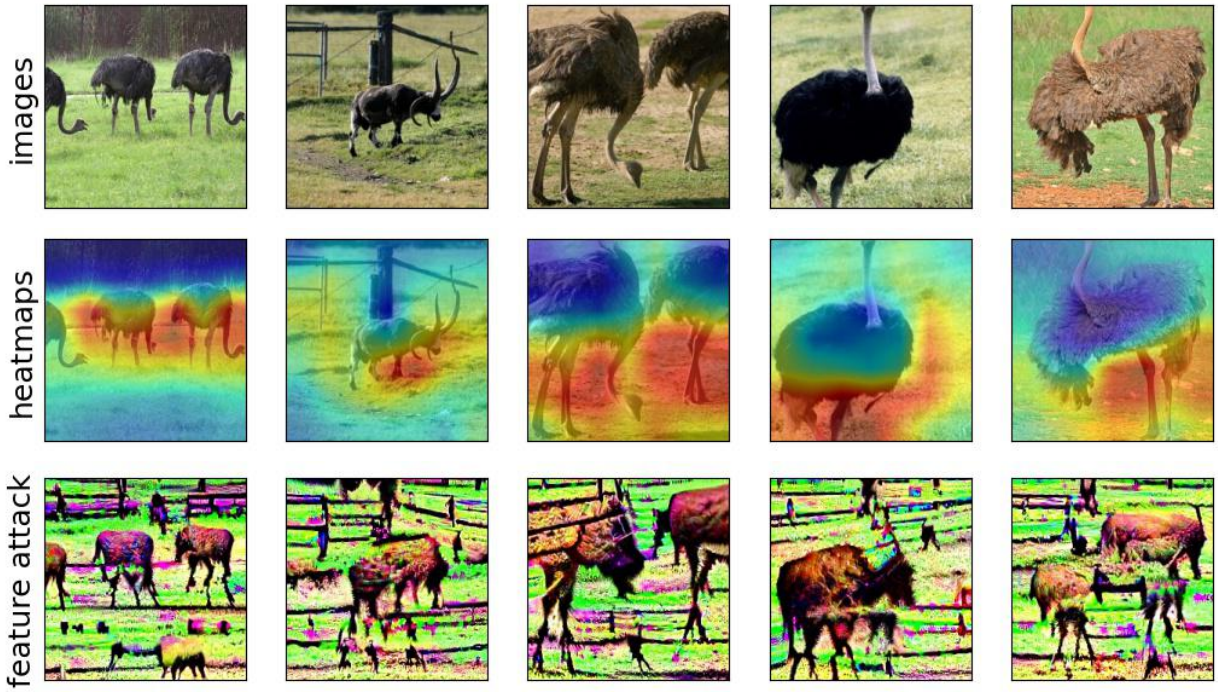


Figure 11.28: Visualization of feature **63** for class **ostrich** (class index: **9**).
Feature label as annotated by Mechanical Turk workers: **spurious**.
Using $\sigma = 0.25$, drop in accuracy: **-20.0**%, mean l_2 perturbation: **47.653**.
Using $\sigma = 0.35$, drop in accuracy: **-32.308**%, mean l_2 perturbation: **63.911**.

11.K.2 Class name: Brambling, Train accuracy (Standard Resnet-50): 95.615%

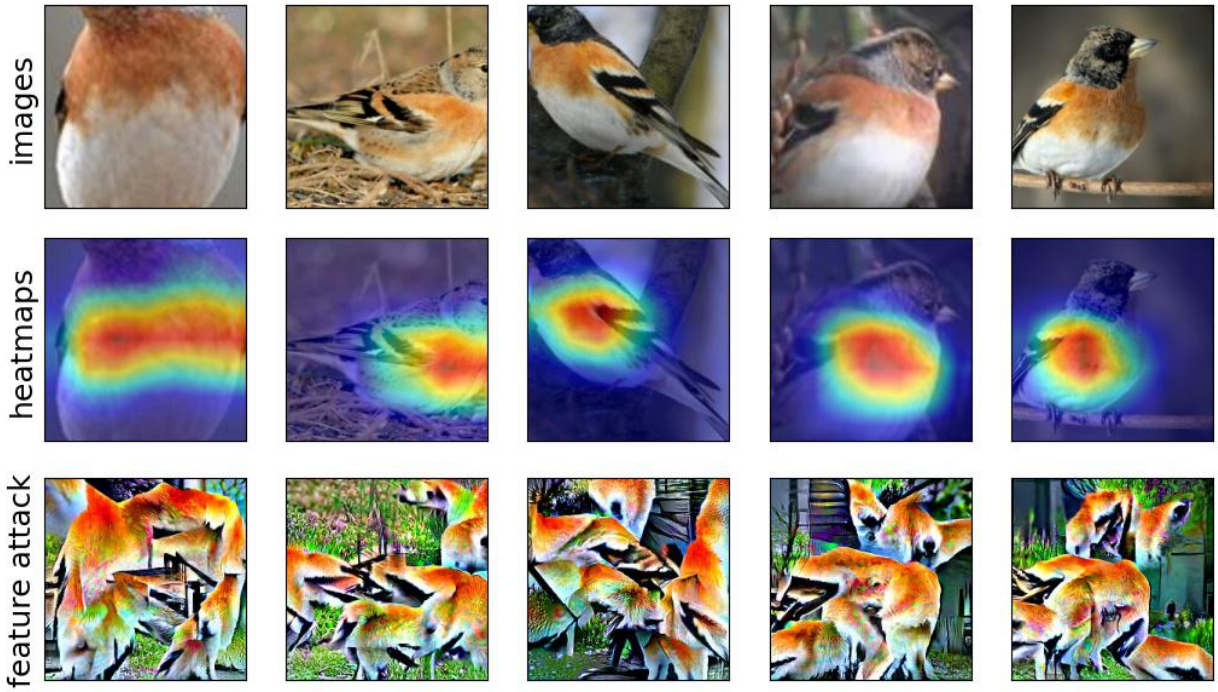


Figure 11.29: Visualization of feature **457** for class **brambling** (class index: **10**).
Feature label as annotated by Mechanical Turk workers: **core**.
Using $\sigma = 0.25$, drop in accuracy: **-1.538**%, mean l_2 perturbation: **25.337**.
Using $\sigma = 0.6$, drop in accuracy: **-21.538**%, mean l_2 perturbation: **51.371**.

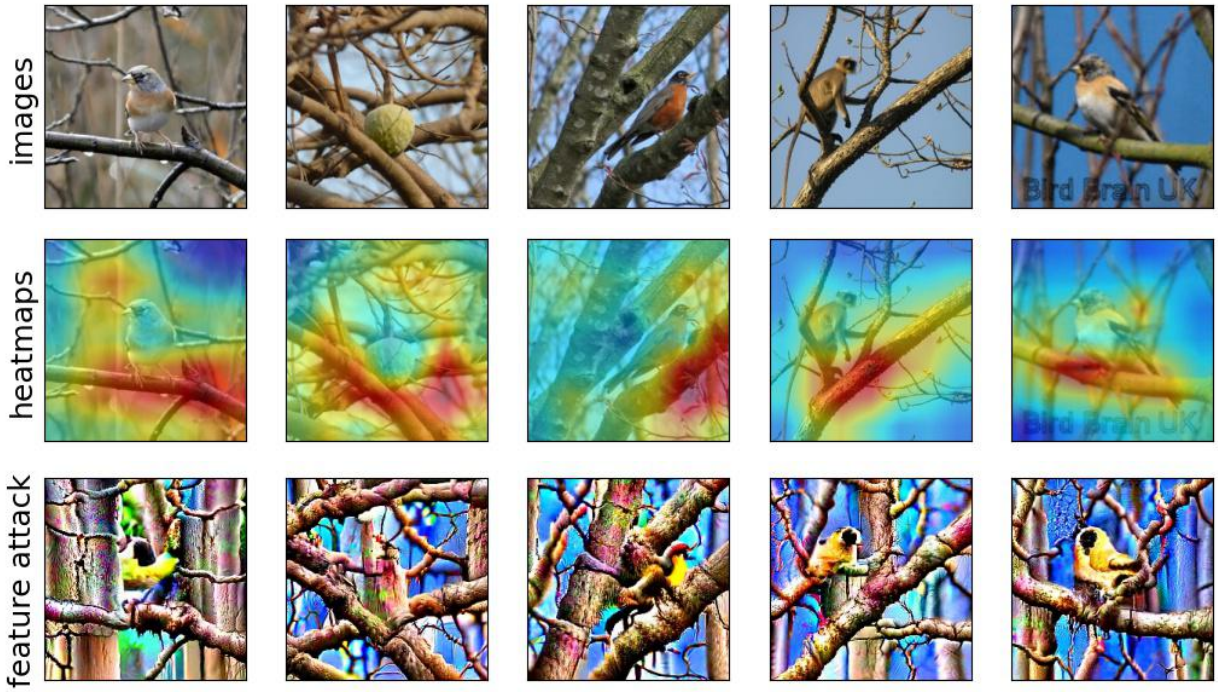


Figure 11.30: Visualization of feature **371** for class **brambling** (class index: **10**).
Feature label as annotated by Mechanical Turk workers: **spurious**.
Using $\sigma = 0.25$, drop in accuracy: **-33.846**%, mean l_2 perturbation: **50.046**.
Using $\sigma = 0.6$, drop in accuracy: **-93.846**%, mean l_2 perturbation: **100.689**.

11.K.3 Class name: Drake, Train accuracy (Standard Resnet-50): 95.462%

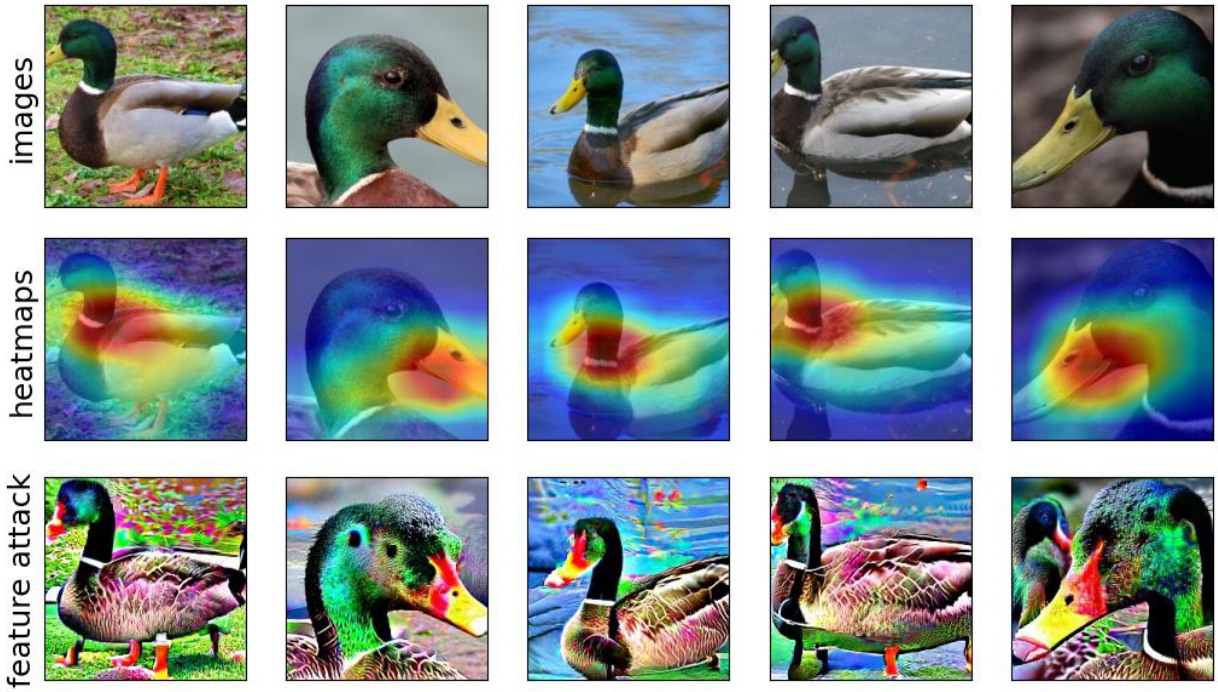


Figure 11.31: Visualization of feature **1113** for class **drake** (class index: **97**).
Feature label as annotated by Mechanical Turk workers: **core**.
Using $\sigma = 0.25$, drop in accuracy: **-1.538**%, mean l_2 perturbation: **36.558**.
Using $\sigma = 0.45$, drop in accuracy: **-16.923**%, mean l_2 perturbation: **60.362**.

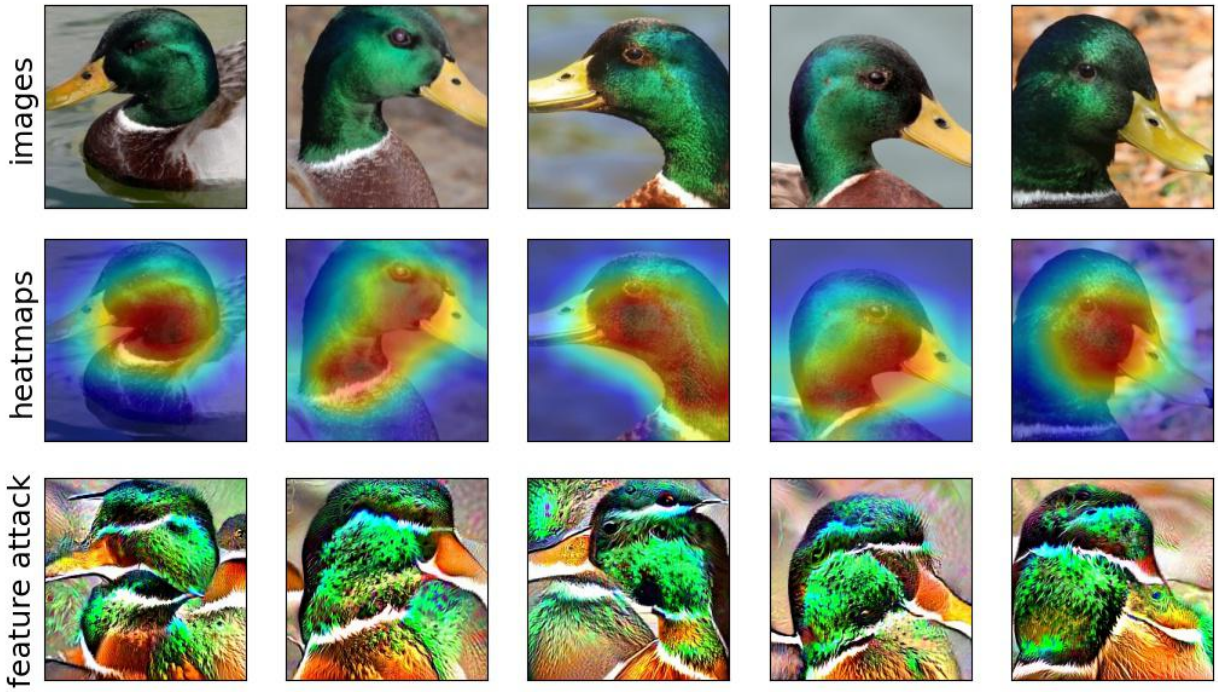


Figure 11.32: Visualization of feature **1339** for class **drake** (class index: **97**).
Feature label as annotated by Mechanical Turk workers: **core**.
Using $\sigma = 0.25$, drop in accuracy: **-4.615**%, mean l_2 perturbation: **32.946**.
Using $\sigma = 0.45$, drop in accuracy: **-16.923**%, mean l_2 perturbation: **54.731**.

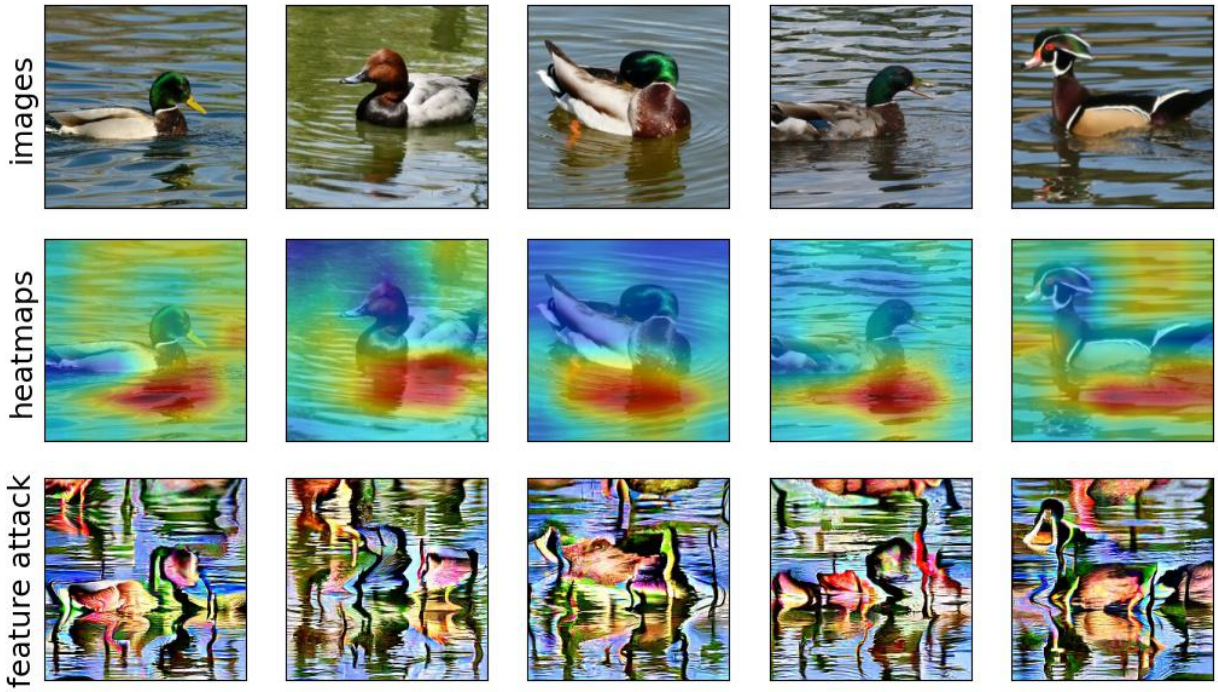


Figure 11.33: Visualization of feature **925** for class **drake** (class index: **97**).
Feature label as annotated by Mechanical Turk workers: **spurious**.
Using $\sigma = 0.25$, drop in accuracy: **-41.539**%, mean l_2 perturbation: **47.465**.
Using $\sigma = 0.45$, drop in accuracy: **-72.308**%, mean l_2 perturbation: **78.273**.

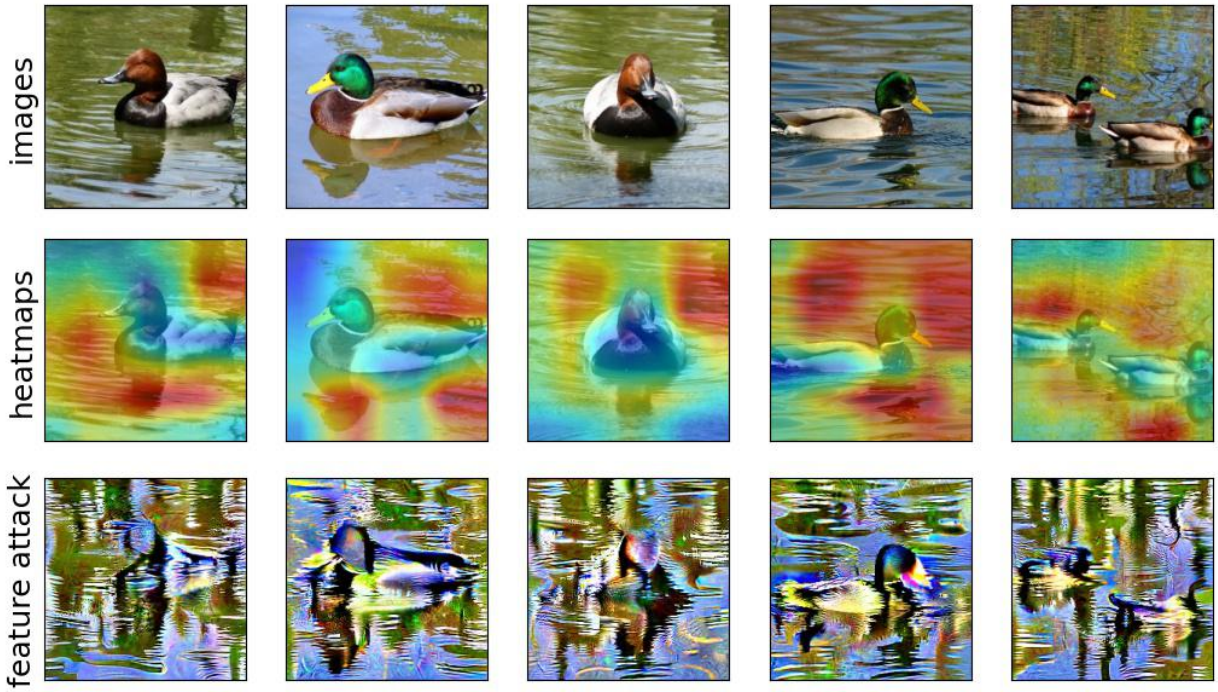


Figure 11.34: Visualization of feature **341** for class **drake** (class index: **97**).
Feature label as annotated by Mechanical Turk workers: **spurious**.
Using $\sigma = 0.25$, drop in accuracy: **-46.154**%, mean l_2 perturbation: **55.205**.
Using $\sigma = 0.45$, drop in accuracy: **-86.154**%, mean l_2 perturbation: **89.644**.

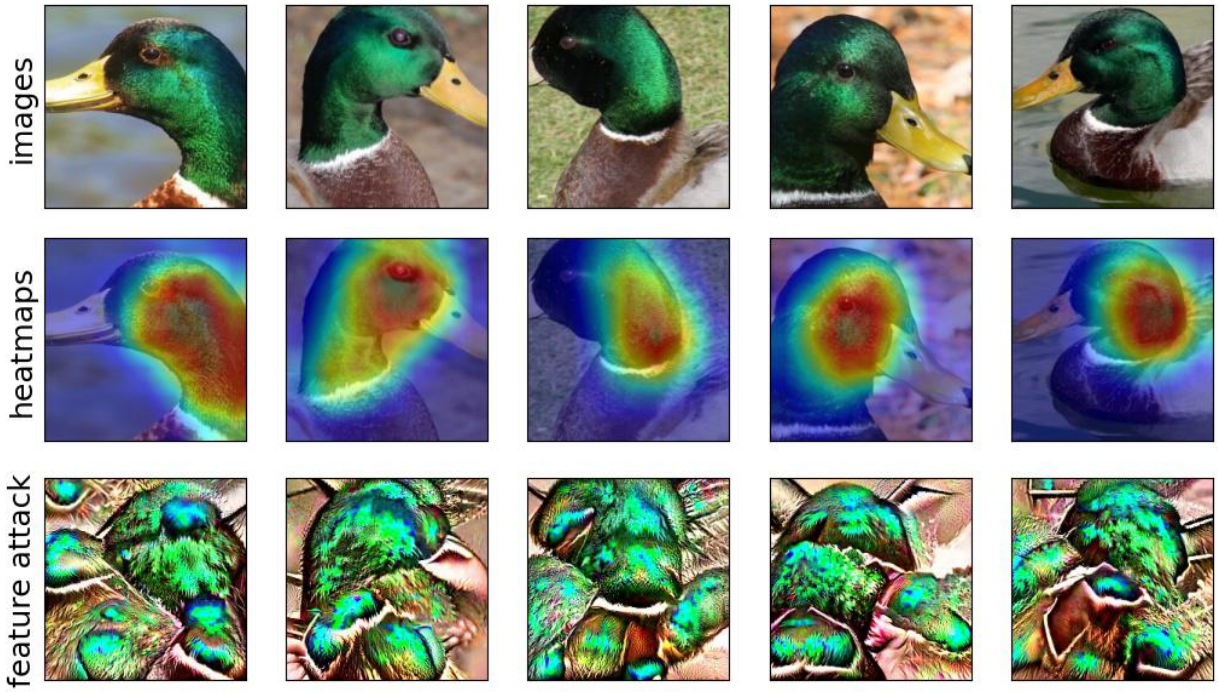


Figure 11.35: Visualization of feature **736** for class **drake** (class index: **97**).
Feature label as annotated by Mechanical Turk workers: **core**.
Using $\sigma = 0.25$, drop in accuracy: **0.0**%, mean l_2 perturbation: **28.878**.
Using $\sigma = 0.45$, drop in accuracy: **-6.154**%, mean l_2 perturbation: **47.661**.
Using $\sigma = 0.55$, drop in accuracy: **-7.692**%, mean l_2 perturbation: **55.554**.

Chapter 12: Hard ImageNet: Segmentations for Objects with Strong Spurious Cues

12.1 Introduction

Deep learning based image classifiers are effective but brittle to distribution shift, leading to serious issues when models are deployed to sensitive applications like medicine [17, 18]. The reliance of models on spurious features, which are predictive of class labels in training data but are irrelevant to the true labeling function, contributes to reduced distributional robustness. The lack of interpretability of deep models is a bottleneck to understanding causes and degree to which models rely on spurious features, especially on real data. Some datasets have been proposed where spurious features are manually annotated or injected into synthetic [315, 332] or application-specific [333, 334] datasets, though insights from these benchmarks may have limited transferability to more general domains.

Recently, [335] apply a mostly automated procedure for neural network interpretability [318] to discover and annotate core (i.e. essential to the class label) and spurious features at scale. [336] expand this analysis to softly segment core input regions in nearly all of the ubiquitous benchmark dataset, ImageNet [56]. Their method leverages the activation maps of neural features that highly contribute to the activation of a given class logit and detect core



Figure 12.1: *Hard ImageNet* classes. Object segmentation masks are overlaid in purple.

features for that class. However, for 15 classes, all of the inspected features were spurious, making the core-feature-based segmentation scheme infeasible, while suggesting the presence of uniquely strong spurious cues within this subset of ImageNet.

In this chapter, we first gather object segmentations for these 15 classes, forming the *Hard ImageNet* dataset. We then curate an evaluation suite of ablation, noise, and saliency based analyses that leverage object segmentations to assess the degree to which models rely on spurious features. With this benchmark, we show that models rely on spurious features much more for Hard ImageNet objects than more typical objects, as represented by the RIVAL10 dataset [337], who's classes appear in multiple standard benchmark datasets (ImageNet, CIFAR10 and STL10 [57, 338]). We then investigate properties of Hard ImageNet objects, observing that they are significantly smaller and less centered, likely leading to models favoring spurious features. Further,

we complement our instance-wise object segmentations by ranking each image within a class based on the strength of spurious cues present, leading to clearer visualizations of the specific spurious features involved. The image rankings also open the door to last-layer retraining of models on data where the presence and absence of spurious features is balanced, an efficient new technique for reducing spurious feature reliance [339].

With Hard ImageNet, we go beyond single-label annotations to shed new insights on the data and model conditions that lead to increased spurious feature reliance in deep image classifiers. We hope our evaluation suite and challenging dataset can inspire new and improved training procedures that result in models that classify *for the right reasons*, towards more reliable and trustworthy AI.

12.2 Related work

The tendency of models to rely on spurious features due to simplicity bias [340] or shortcut learning [341] has garnered much attention, due to its consequences on fairness and generalization, leading to the proposal of many algorithms to mitigate spurious dependencies [332, 342, 343, 344, 345, 346]. However, in comprehensive evaluations, the gains of these methods over empirical risk minimization are often marginal or inconsistent [347, 348]. Recently, a simpler approach for reducing spurious feature reliance has been proposed, where only a final linear layer is retrained over data that balances the presence of spurious cues, thus encouraging models to downweight spurious features, which are less predictive of class labels in the balanced set [339]. A bottleneck to this approach, along with many of the other mentioned algorithms, is that they require knowledge of the degree to which spurious cues are present in each sample; the

image rankings of our dataset may address this need.

Several datasets have been created to aid in studying spurious feature use, including many with synthetically-injected spurious correlations [315, 332, 349]. A noteworthy example is ImageNet-9, which swaps backgrounds across classes to assess background sensitivity, using analyses similar to our ablation studies [350]. While insightful, the use of simple classification tasks and unnatural data in these synthetic sets may limit their impact in practice. A number of real-world datasets have also been proposed [333, 334, 351, 352], though they are usually very specific to the domain of their application.

Recent work appeals to additional annotations to assess model sensitivity to various input regions via noise-based analyses [318, 337]. Notably, [336] annotate core and spurious input regions for nearly all of ImageNet in a mostly automated procedure, leveraging neural feature visualizations [318] and the improved interpretability of an adversarially robust network [229], resulting in *Salient ImageNet-1M*, a large-scale, natural, and general dataset for evaluating and improving model reliance on spurious features. Namely, they used activation maps of robust neural features corresponding to core features as soft segmentations of core regions (see Supp. Section 12.C). However, for 15 classes, all neural features inspected were spurious, making their segmentation framework inapplicable. Arguably, these classes are the most interesting, as the underlying robust network used for segmentation relied on spurious features the most for these classes, indicating that this subset of data contains the strongest spurious cues. We manually collect object segmentations for these 15 classes, design a suite of evaluation metrics, and study the properties of this data that leads to heightened spurious feature reliance.

12.3 Hard ImageNet Dataset

12.3.1 Overview

Our dataset, *Hard ImageNet*, consists of images from 15 ImageNet [56] synsets. These classes are *Dog Sled, Howler Monkey, Seat Belt, Ski, Sunglasses, Swimming Cap, Balance Beam, Gymnastic Horizontal Bar, Patio, Hockey Puck, Miniskirt, Keyboard Space Bar, Volleyball, Baseball Player,* and *Snorkel*, selected because for each class, all features annotated in [336] were spurious with respect to the class. Specifically, [336] inspect the five neural features (i.e. nodes in the penultimate layer) of an ℓ_2 adversarially trained ResNet50 per class that contribute most to the activation of the corresponding logit. Each class-feature pair is annotated as *core* or *spurious*, based on if the detected feature is essential to the class label. Over all classes, the vast majority of class-feature pairs inspected were deemed *core*. Thus, having all five most important features serving *spurious* roles suggests that each Hard ImageNet class uniquely has very strong spurious cues that models favor over core features.

We collect object segmentation masks for these 15 classes, resulting in a dataset with $19,097$ training samples and 750 validation samples. We maintain the same train/validation split as ImageNet. Figure 12.1 visualizes an example and its object segmentation for each Hard ImageNet class. Additionally, we provide class-wise rankings for each image in the Hard ImageNet training set based on the strength of spurious cues present in the image. We base these rankings on the activations of the annotated neural features for each class (see Section 12.6 for more details).

Impact First, Hard ImageNet segmentations complete the *Salient ImageNet* dataset, which al-

lows for the exploration of going beyond single-label supervision in training ImageNet models to predict *for the right reasons*. That is, with object segmentations available during training, models can be guided to rely more on core features than spurious ones, leading to improved generalization to domains where spurious correlations are broken. Moreover, Hard ImageNet annotations shed insight on the conditions for models and data under which spurious features are favored: through our suite of evaluation metrics (see Section 12.4), the effect of training procedures and model architectures on spurious feature reliance can be compared, while analysis of segmentation masks reveal how the shape, size, and location of objects in images affects model sensitivity to the object (see Section 12.5).

12.3.2 Collection Procedure

We collect Hard ImageNet object segmentations over Amazon Mechanical Turk. To ensure quality, annotations are collected in many phases. First, workers must pass a *qualification exam*, consisting of segmenting one sample from each class. All workers take the same qualification exam, and only workers who achieve an intersection-over-union (IoU) score of at least 0.65 pass. For reference, any IoU greater than 0.5 is considered successful in the Pascal VOC object detection evaluation [353]. Workers meeting this bar demonstrate proficiency with the Mechanical Turk platform and sufficient understanding of the relevant objects to be segmented. Then, workers sign a *informative consent form*, in which we explain the nature of and purpose for their work, and also share answers to the qualification exam and tips for improved segmentations so to correct for common errors. Finally, qualified and consenting workers move on to the full data collection stage, where we release batches in

rounds, monitoring quality with *attention checks* (randomly inserted images with ground truth segmentations) and updating class-specific tips based on errors observed on attention checks. We place attention checks with a frequency of 10% for the validation split and 5% for the training split. The average IoU score on attention checks was 0.76. Workers with very low quality on attention checks were removed from the study, and their segmentations were redone in later rounds. Also, each image in the validation set is annotated five times, with annotations consolidated via a pixel-wise majority vote to correct for errors in any individual segmentation. Figure 12.2 shows the IoU scores of each validation set segmentation to the consolidated final validation masks. The vast majority of segmentations are highly agreed upon, with a small minority having near zero agreement, indicating that the extra rounds of annotation had a meaningful corrective impact.

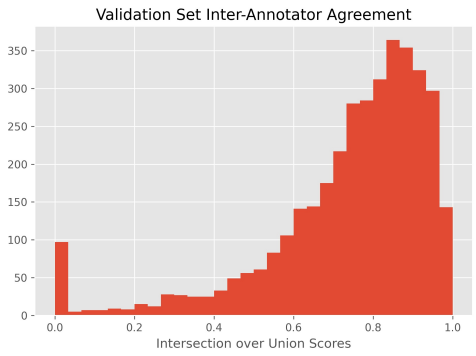


Figure 12.2: Agreement of individual segmentations with final averaged segmentation for Hard ImageNet validation split.

Throughout our data collection, we strive to create a transparent and collaborative working environment. In the consent/information stage, we invite feedback and explicitly encourage our workers to directly contact us with questions. Moreover, we reward workers with fair salaries (estimated to be $\$12$ to $\$16$ an hour depending on annotation speed) and bonuses (between 5% and 7% depending on number of annotations) for continued work. We also invite workers who previously partook in RIVAL10 [337] annotation, as their experience from a similar task leads to improved annotation quality, and rewarding loyalty improves worker well being.

12.4 Benchmarking Spurious Feature Reliance

We compile a suite of evaluation metrics that use object segmentations for assessing image classifier reliance on spurious features. We focus this section on demonstrating the significantly higher degree of spurious feature reliance for models on Hard ImageNet compared to other more typical data. As a baseline, we use RIVAL10 [337], which is comprised of images from 20 ImageNet synsets, selected so that each synset corresponds to a class from CIFAR10 [57] (i.e. two synsets per CIFAR10 class). Importantly, RIVAL10 includes an object segmentation per image, collected in a similar fashion to Hard ImageNet's annotations. We consider RIVAL10 data to be more typical of what is commonly studied in image classification, as many or all of the classes also appear in the standard benchmark datasets CIFAR10 and STL10. We note that we use samples in RIVAL10 that were originally in ImageNet's validation split, as opposed to using RIVAL10's own train/test split, which leaks ImageNet training samples to their test split. For Hard ImageNet, we also evaluate over images originally in ImageNet's validation split, which corresponds to Hard ImageNet's validation split.

Across our evaluations, we use a convolutional neural network (ResNet50 [1]) and a vision transformer (small DeiT [354]) of roughly equal size, both pretrained on ImageNet. To evaluate these models 'off the shelf', we map samples from Hard ImageNet and RIVAL10 to their original ImageNet class indices. Thus, for RIVAL10 evaluation off the shelf, we actually consider 20 target classes, and so we refer to this as RIVAL20. We also include finetuned versions where a final linear layer is trained for each model atop representations from the corresponding fixed feature encoder. For RIVAL10, we finetune for both 10-way and 20-way classification. While the emphasis of this section is *data*, in practice our evaluation metrics are intended for comparison

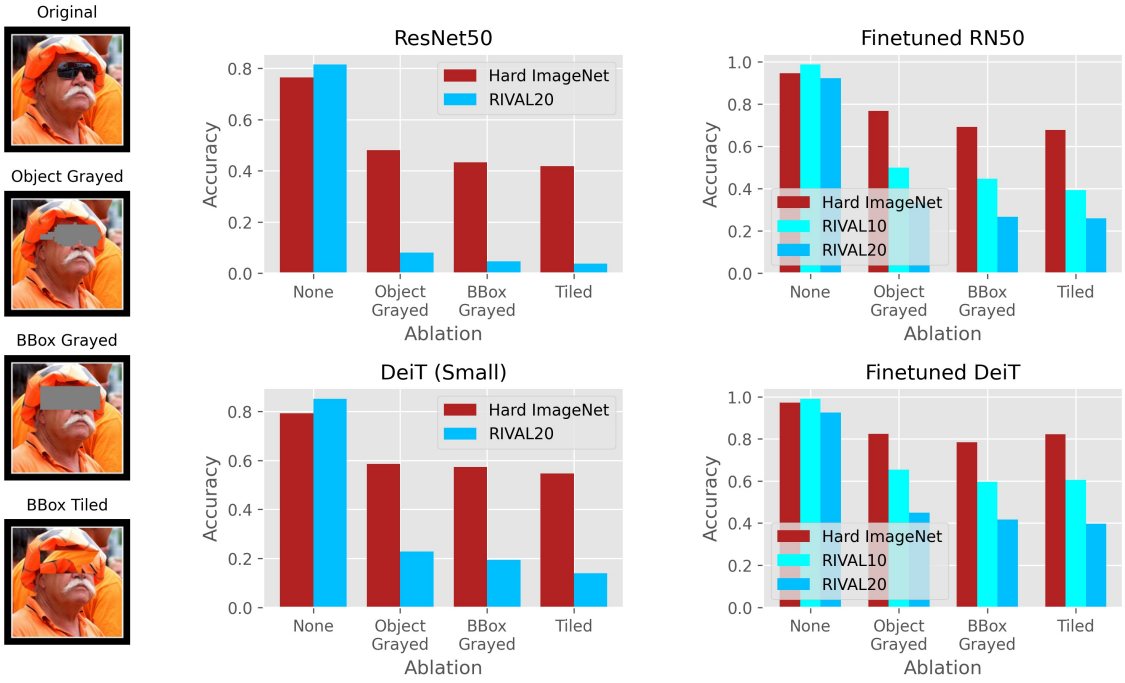


Figure 12.3: Accuracy under ablation. Accuracy drops much less when Hard ImageNet objects are ablated than when RIVAL10 objects are ablated. Example ablations for class *sunglasses* shown left.

of *models* of diverse types.

12.4.1 Ablation

Object segmentation masks allow for removal of the object corresponding to the class label for any sample. Classifying an ablated image to its original class indicates that the model uses background information as it does not require the object's presence to predict its class. While these ablated images are out of distribution, accuracy drop due to ablation can still inform the degree to which the model relies on spurious background features. We consider three types of ablation. First, we replace all object pixels with 0.5, graying out the object. Second, we replace all pixels in the *bounding box* of the object with gray, thus also removing object shape information. Finally, we replace the bounding box of the object with a tile adjacent to the object in the image.

In cases where a tile is smaller than the bounding box, we repeat the tile to fill the region, as in [350]. Note that all ablations replace each instance of an object in an image separately (i.e. we consider multiple boxes bounding each instance as opposed to one box containing all instances) so to retain as much background as possible.

Figure 12.3 shows that ablation leads to a much smaller accuracy drop for Hard ImageNet objects than for RIVAL10 objects, in both the finetuned and off the shelf case, and for all types of ablation. In fact, for an off the shelf DeiT (i.e. performing 1000-way classification), Hard ImageNet images are predicted to their class over 50% of the time under all ablations, despite the fact that the object essential to the class label is actually missing, rendering the predictions inaccurate (see Supp. Section 12.D). This is several times larger than the rate for RIVAL20. In finetuned models, accuracy under ablation is still significantly larger for Hard ImageNet, though this gap is smaller. We note that random guessing yields much higher accuracy in these cases because the number of classes is multiple orders of magnitude smaller. Because number of classes varies across the three datasets considered, direct comparison is more challenging, but Hard ImageNet accuracy is still substantially larger than RIVAL10 ablated accuracy, where random guessing achieves 5% higher accuracy than on Hard ImageNet.

12.4.2 Relative Foreground Sensitivity

We now turn to the noise-based metric *relative foreground sensitivity* (RFS), introduced with RIVAL10 [337]. To compute RFS, equal amounts of Gaussian noise is added to foreground and background regions. Then, the gap in accuracy drops (i.e. how much more accuracy drops due to foreground noise than background noise) is normalized to allow for comparison across

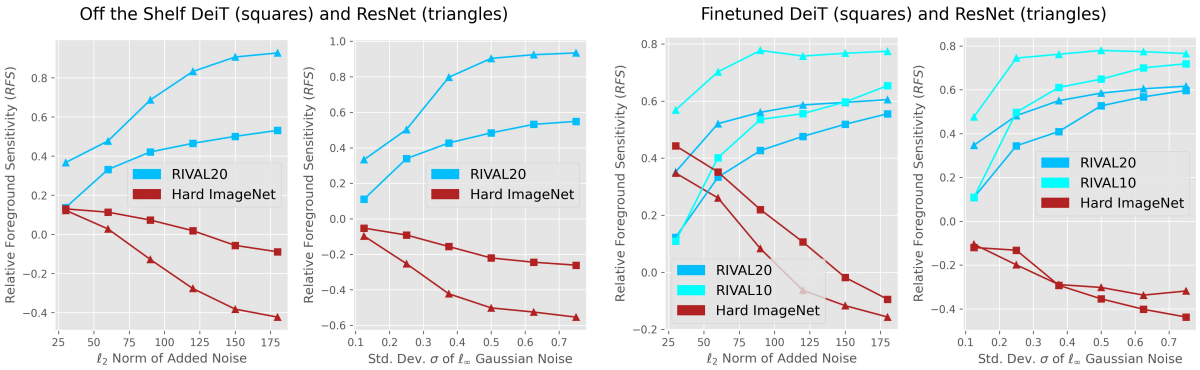


Figure 12.4: Relative Foreground Sensitivity (RFS) for DeiT and ResNet, off the shelf (left) and finetuned (right). RFS for Hard ImageNet is significantly lower, and often negative, indicating greater sensitivity to background noise than foreground noise.

models with varying noise robustness, as derived extensively in [337] and [336]. Higher RFS scores entail greater sensitivity to noise in foregrounds relative to backgrounds. We consider adding both ℓ_∞ and ℓ_2 noise, where in the latter case, we fix the ℓ_2 norm of noise added in both backgrounds and foregrounds. These types of noise have opposite size biases, as ℓ_2 noise results in a higher ℓ_∞ norm of noise is smaller regions, while ℓ_∞ noise results in a higher ℓ_2 norm of noise in larger regions.

In figure 12.4, we see that for all models and across noise levels, RFS is substantially lower when evaluated on Hard ImageNet than on RIVAL10 or RIVAL20. In fact, RFS **is negative in many cases**, indicating that noise in the background reduces accuracy more than noise in the foreground. For finetuned models at low levels of ℓ_2 normalized noise, RFS is comparable across datasets, but otherwise, models appear to be significantly more sensitive to noise in backgrounds than they are to the actual object in Hard ImageNet, suggesting that models use object information far less than surrounding context. Arguably, despite achieving high accuracy, this result suggests that models are not actually learning to detect Hard ImageNet objects, instead relying on spurious background cues.



Figure 12.5: GradCAMs of Hard ImageNet instances with poor saliency alignment for finetuned ResNet50 (left three) and finetuned DeiT (right two). Spurious features of *gymnast, dogs, hockey stick and ice, legs* and *keys* for each class respectively are highlighted.

12.4.3 Saliency Alignment

Our third metric makes use of the interpretability method GradCAM [283], which assigns a saliency score to each input pixel, proxying its importance to the prediction. While GradCAMs are qualitative, we perform a quantitative analysis by computing the alignment of GradCAMs to objects. Specifically, we compute IoU of object segmentations and GradCAMs (binarized with a threshold of 0.5) as in [337]. Figure 12.6 shows IoU scores across models are significantly lower for Hard ImageNet objects than RIVAL10 objects, indicating that backgrounds are more salient to Hard ImageNet classification.

With saliency alignment, we can also automatically extract instances with low IoU scores to reveal specific spurious background cues. Some examples are visualized in figure 12.5. Here, we see that other features present in the image but non-essential to the class label are arguably more prominent and easier to recognize than the actual object. For example, the keys on a keyboard are more noticeable than the spacebar below, as they take up more of the image and contain more detail than the uniformly colored spacebar. Further, the location of the objects in these examples makes them appear as though they are in the background, as they reside along the edges of the

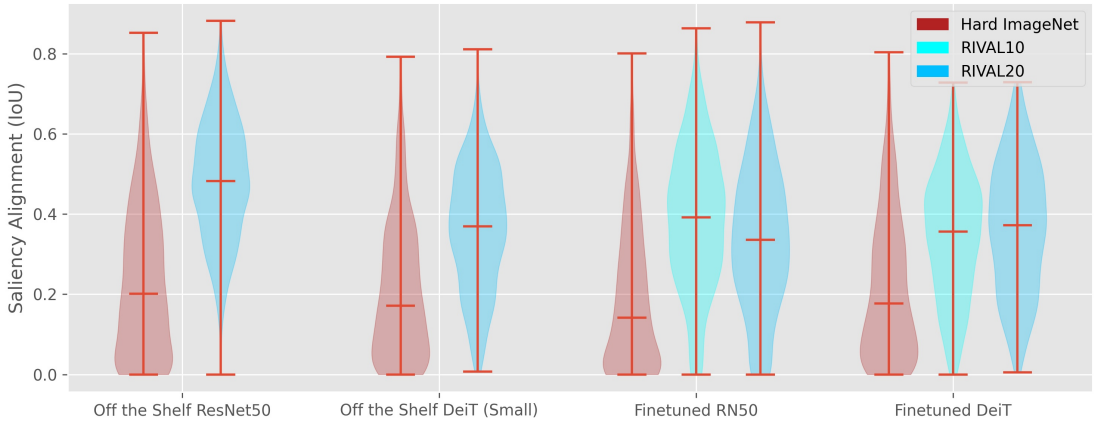


Figure 12.6: Alignment of GradCAM saliencies to object masks. Low alignment indicates more saliency is placed in the background. Across all models, saliency alignment is lowest for Hard ImageNet.

image, as opposed to the center. We investigate the size and location of Hard ImageNet objects in more detail in the following section.

12.5 Hard ImageNet Objects are Smaller and Less Centered

Figure 12.8 shows that compared to RIVAL10, Hard ImageNet objects occupy substantially less space both in their original images and after the standard ImageNet test time augmentation (consisting of resizing and center cropping). Some reasons for this are that Hard ImageNet objects are i. simply smaller (*hockey puck, swimming cap*), ii. oddly shaped so that square crops include lots of background (*spacebar, ski*), iii. often co-occurring with much larger spurious features (*snorkel, balance beam*).

Interestingly, the standard test time augmentation only slightly increases the percent of the image the object occupies, unlike for RIVAL10 objects, which see a larger increase. The lack of a center bias in object locations likely contributes to the reduced benefit of augmentation, as a center crop would not amplify non-centered objects. In fact, we find that this augmentation often

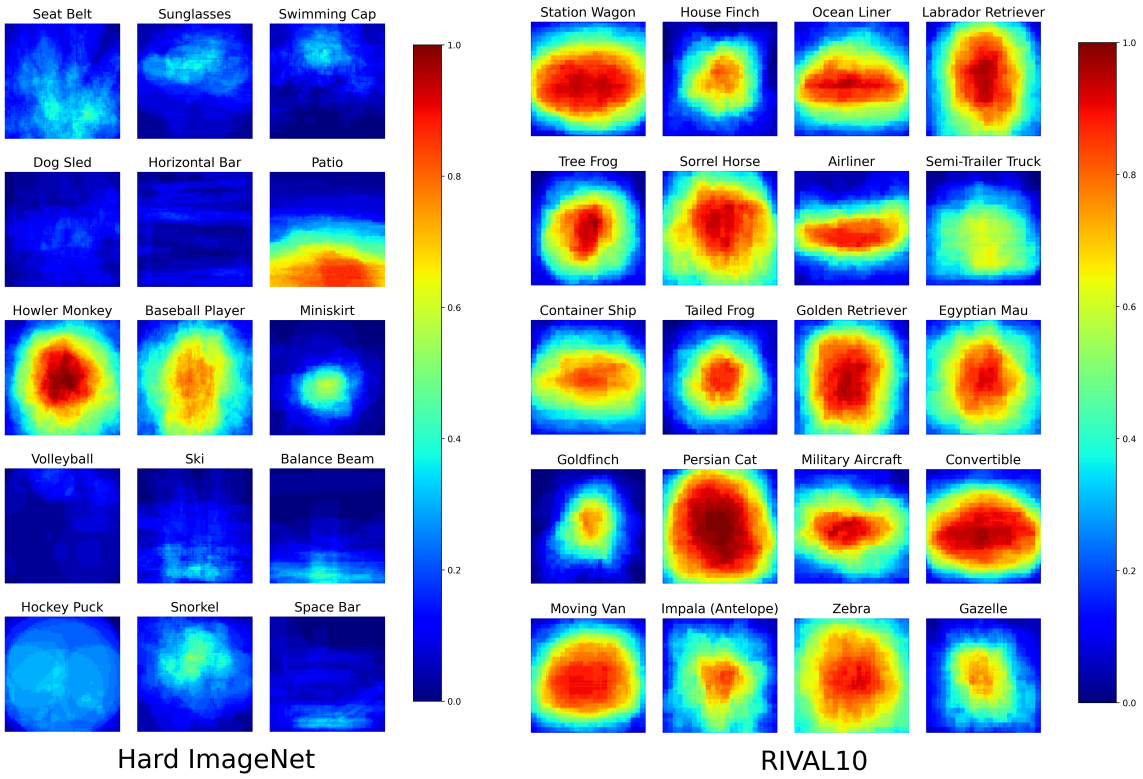


Figure 12.7: Average object position for Hard ImageNet and RIVAL10. Hard ImageNet objects occupy less space and are often not centered, unlike RIVAL10 (and most other ImageNet) objects.

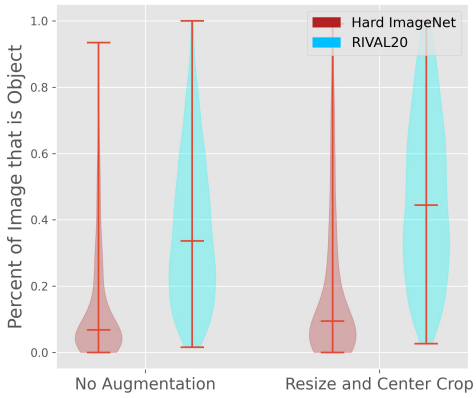


Figure 12.8: Hard ImageNet objects occupy less space in images than RIVAL10 objects both before/after standard augmentation.

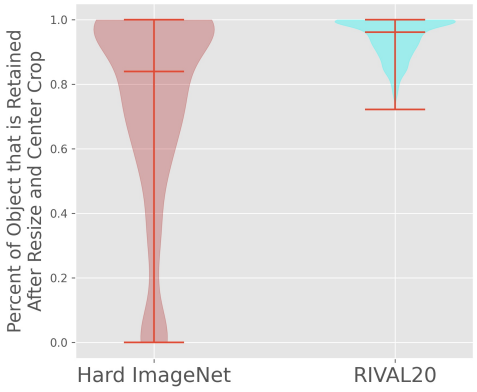


Figure 12.9: Hard ImageNet objects are often substantially cropped out by the standard ImageNet test time augmentation.

removes large portions of Hard ImageNet objects, as shown in Figure 12.9. While in the worst case at most 25% of a RIVAL10 object is lost due to the standard augmentation, much more of the object is lost in Hard ImageNet, including cases where **standard augmentation completely removes Hard ImageNet objects**.

We visualize the class-wise average of object segmentations after standard augmentation in figure 12.7. Indeed, Hard ImageNet objects are smaller and less centered than RIVAL10 objects. For *howler monkey* and *baseball player*, which are relatively larger and more centered, we conjecture that the increased spurious feature reliance is due to easily recognizable spurious cues. Namely, howler monkeys are usually photographed in dense foliage with many leaves and branches. For baseball players, the uniform green grass and brown dirt of the baseball field is likely very easy for a deep network to detect. While patios are not centered, they do take up lots of space, but they often co-occur with patio furniture, which models may rely on more so than the patio itself. Finally, we note the smaller size of the finches in RIVAL10; RIVAL10's original analysis observed greater spurious feature reliance in the bird class, further suggesting that small

object size leads to greater model use of backgrounds.

12.6 Ranking Images by Spurious Feature Presence

To complement Hard ImageNet segmentation masks, we annotate each image in the training set with a class-wise rank corresponding to the strength of spurious features present in the image. We determine these ranks by appealing to neural features (nodes in the penultimate layer) of the ℓ_2 adversarially trained ResNet50 used to construct the Salient ImageNet dataset [335, 336]. Per class, we first rank all images by feature value on each of the five annotated features (which are all spurious for Hard ImageNet classes). We then sort the average of these ranks to obtain our final image rankings.

Figure 12.10 shows the highest and lowest ranked images for classes *ski, snorkel*, and *howler monkey*. Samples with low spurious rank are classified with significantly lower accuracy. Also, viewing the high and low ranked images reveals the strong spurious cues present in Hard ImageNet. For example, models appear to rely on *snow* and *skiers* to classify *skis*. Similarly, the absence of *people* and *water* for *snorkels* and *trees* for *howler monkey* reduces spurious ranking and classification accuracy. Viewpoint may also be a spurious feature for *howler monkey*, as highly ranked images are all taken from below, with the sky and trees as the backdrop. Viewpoint, along with backgrounds, was observed to be a spurious correlation that contributes to large accuracy drop when broken in ObjectNet [355].

While image ranking facilitates the interpretation of spurious features naturally (i.e. without using visualizations that make artificial changes to the image, which were observed to be less effective in [356]), they can also be used to reduce spurious feature reliance. Namely,

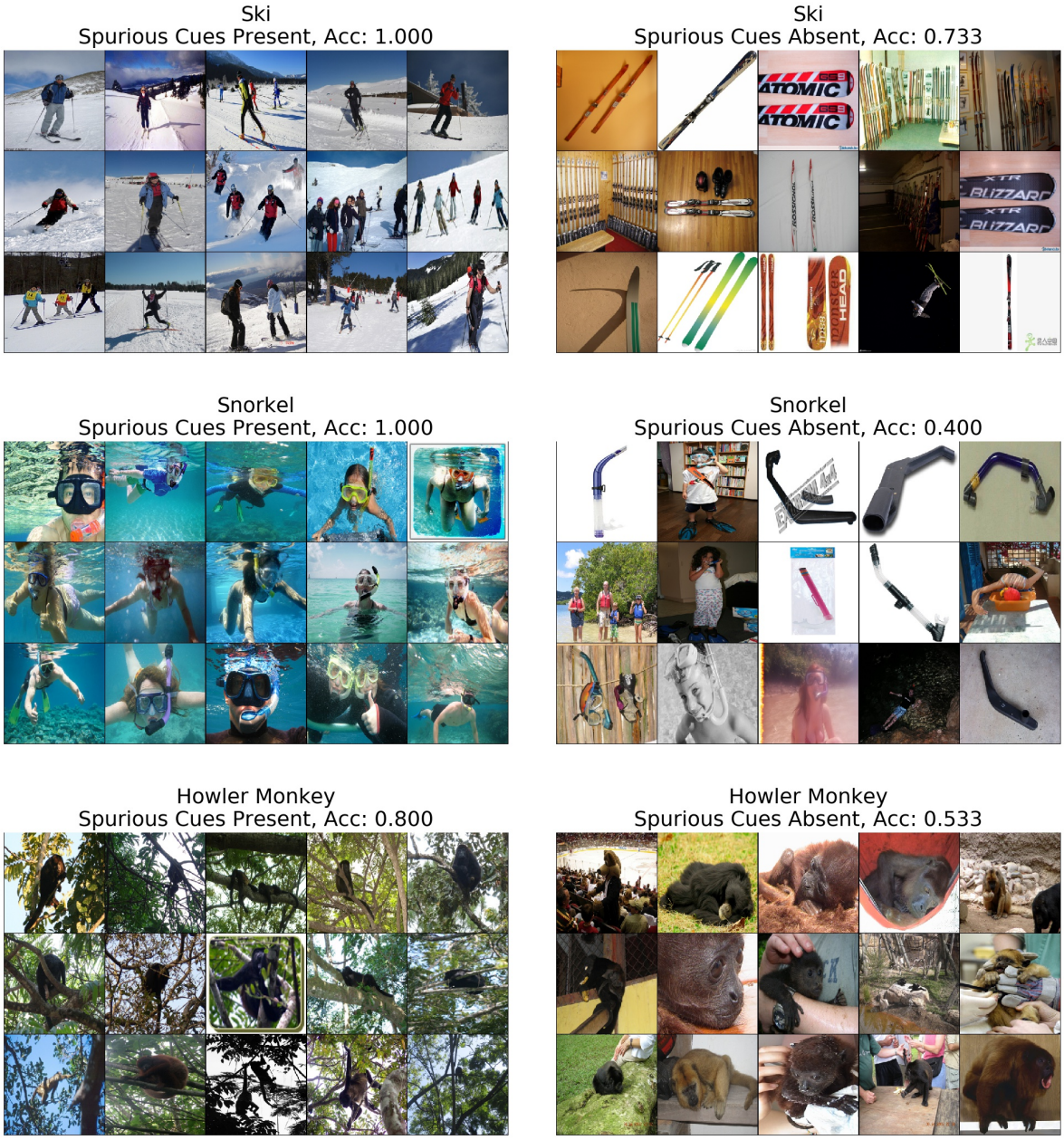


Figure 12.10: Examples of high (left) and low (right) spurious feature presence, based on Hard ImageNet rankings. Accuracies shown are for a standardly trained ResNet50.

retraining a linear layer atop fixed encoders on data that balances the presence and absence of spurious features has been found to greatly improve performance on instances where spurious correlations are broken [339]. Spurious image ranking allows for the construction of these balanced datasets, explored in the following section.

12.7 Improving Models with Hard ImageNet Annotations

We now explore baseline methods to harness Hard ImageNet's annotations for improved model classification (i.e. with reduced reliance on spurious features). We focus our study on finetuned models pretrained on ImageNet, using ResNet50 and DeiT (Small) as in Section 12.4. We keep features fixed during finetuning, only optimizing the parameters of a new final layer for the 15-way Hard ImageNet classification.

We employ two approaches for mitigating spurious feature reliance. [336] propose **Core Risk Minimization** (CoRM) as an alternative to ERM when segmentations of core (i.e. not spurious) regions are available; Hard ImageNet's object segmentations fulfill this prerequisite. Specifically, the objective of CoRM is to minimize classification error over the distribution of images *with noise applied to non-core regions*, so that the optimal classifier predicts correctly even when spurious features are corrupted. In that work, *random noising*, where small amounts of Gaussian noise are added to non-core regions with probability $p = 0.5$, and *saliency regularization*, where the ℓ_2 norm of the gradient on non-core pixels is added to the classification loss, were applied in tandem to improve relative core sensitivity (an analagous metric to RFS). [339] propose **Deep Feature Reweighting** (DFR), in which retraining a final linear layer using a *balanced dataset* reduces spurious feature reliance. The balanced dataset consists of a subset of the

Method		Ablation Accuracies (↓)				RFS (↑)		Saliency (↑)
CoRM	DFR	None (↑)	Gray	Gray BBox	Tile	$\sigma = 0.25$	$\sigma = 0.5$	IoU
Finetuned DeiT (Small)								
✗	✗	**96.79**	84.22	80.48	81.15	−0.19	−0.35	20.90
✓	✗	96.39	**81.02**	78.74	80.75	**0.02**	**−0.19**	21.57
✗	✓	96.66	81.28	**77.01**	77.94	−0.20	−0.33	21.63
✓	✓	96.52	82.35	**77.01**	**77.81**	−0.10	−0.29	**21.99**
Finetuned ResNet50								
✗	✗	94.25	75.94	69.39	67.38	−0.18	**−0.27**	18.44
✓	✗	92.91	76.20	69.12	68.32	**−0.08**	**−0.27**	**20.43**
✗	✓	**94.39**	73.53	67.51	66.71	−0.27	−0.35	18.39
✓	✓	91.31	**72.59**	**63.64**	**63.90**	−0.23	−0.31	20.35

Table 12.1: Final layer retraining improves faithful learning on Hard ImageNet. Results shown for entire benchmark under two different training approaches: i) Core Risk Minimization (CoRM) via *random background noising* and *saliency regularization*, and ii) deep feature reweighting (DFR) using a *spurious-balanced training subset*. We also report results for the combination of the two approaches and ordinary finetuning (as a baseline) under two architectures. Relative Foreground Sensitivity (RFS) is evaluated under two ℓ_∞ noise levels, indicated by σ. Saliency refers to *saliency alignment* as measured by intersection over union (IoU).

training data containing an equal portion of samples with and samples without spurious features, essentially breaking spurious correlations that impede generalization to minority groups. Using Hard ImageNet's image rankings, we extract the top and bottom 100 images for each class to form the spurious-balanced subset.

Table 12.1 shows that these two methods can considerably reduce model reliance on spurious features, improving numbers across all metrics in our benchmark. Between the two approaches, CoRM appears to lead to more improvement in saliency alignment and RFS, while DFR yields beter results for ablation. Combining CoRM and DFR leads to even better performance with respect to accuracies under ablation. While improvements are at times small, we note that in these experiments, the vast majority of model parameters are left unchanged, as we

only train a new final layer. We leave the door open to new approaches for improving the *faithful* learning of Hard ImageNet objects, including training models from scratch.

12.8 Discussion

Our benchmark brings a new perspective to classification, as we not only seek models that predict accurately, but also predict for the right reasons. Assessing model performance using accuracy alone can obscure key misconceptions held by models, which may only become apparent when models are deployed to new domains at test time. Moreover, design decisions such as training strategy and architecture may affect the degree to which spurious features are relied upon, as observed in [337]; this dataset and accompanying benchmark can reveal these model differences. Finally, we emphasize the need to understand model behavior under "bad" data; that is, images where the object of interest is not centered or large, unlike most cases. With models becoming increasingly data hungry, it is inevitable that some portion of the data will not capture objects in ideal conditions. Further, certain objects simply are not well suited to be captured prominently (i.e. large and centered) in square photos. Figuring out how to learn to recognize objects from these suboptimal data conditions will be an important challenge to extend the impressive performance of deep classifiers from standard datasets to many more realistic settings.

With Hard ImageNet, the community can evaluate the capacity of any ImageNet trained model to faithfully learn challenging objects, and also explore how going beyond single class label annotations can lead to improved image classifiers. While segmentation masks are expensive to collect, procedures that are much more automated already exist [336], and we envision newer ones are likely to emerge with time. Also, the procedure with which we ranked images

was largely automated, indicating that these types of annotations are by no means prohibitively expensive. We hope Hard ImageNet can lead to new perspectives on both training and evaluation paradigms for image classification.

Supplementary material

12.A Distinguishing Hard ImageNet from Related Challenge Datasets

Our work is inspired by other challenge datasets that focus on improving deep classifiers by aggregating edge cases where usually strong performance falters. We highlight two datasets in particular: ObjectNet [355] and ImageNet-A [357]. Both of these datasets include samples where spurious correlations are broken, leading to dramatically lower accuracy. Further, these datasets consist of clean images, as opposed to other challenge sets that make synthetic changes [332, 350].

We now outline some key distinctions between Hard ImageNet and these datasets. First, ObjectNet and ImageNet-A only contain test sets. We include a training set in Hard ImageNet because the central goal of our work is for the community to develop new algorithms that can learn to recognize objects without relying on spurious cues, even when the spurious signals are very strong in the training data. To this end, we also introduce two new forms of annotation (object segmentation and image ranking), with hopes of challenging the community to explore training paradigms beyond single-label supervision. Lastly, model performance on ObjectNet and ImageNet-A is evaluated using *accuracy*. In contrast, we present three alternative evaluation metrics leveraging the richer annotations of Hard ImageNet. In spirit, our evaluation is orthogonal to the traditional metric of accuracy, as we shift the focus from *what* models predict to *how* they predict. We believe that the reliability and trustworthiness of deep models hinges on their use of appropriate reasoning structures. That is, if a model predicts correctly but for the wrong reasons, the model may act erratically when deployed.

We greatly value the inspiring work of these earlier challenge datasets and recognize the similarities of their work to our contribution, though we believe that Hard ImageNet may open the door to understanding deep classifier performance, specifically with respect to spurious feature reliance, in a new light.

12.B Evaluation of Additional Pretrained Models

Model	Ablation Accuracies (↓)				RFS (↑)	Saliency (↑)
	None (↑)	Gray	Gray BBox	Tile	$\sigma = 0.25$	IoU
Hard ImageNet						
Swin	80.59	61.19	59.97	59.30	−0.01	4.28
Convit	79.92	60.11	59.97	55.93	−0.12	22.37
Densenet161	57.68	37.06	30.05	29.65	−0.26	18.10
Vgg16	71.83	46.63	41.37	42.86	−0.53	16.80
RIVAL20						
Swin	86.96	30.13	25.18	18.50	0.44	6.64
Convit	85.74	21.64	25.28	16.68	0.31	35.89
Densenet161	78.67	7.58	3.03	2.73	0.40	43.95
Vgg16	76.74	6.88	3.03	3.44	0.80	30.08
Hard ImageNet - RIVAL20						
Swin	−6.36	31.05	34.80	40.80	−0.45	−2.36
Convit	−5.82	38.47	34.69	39.25	−0.43	−13.53
Densenet161	−20.98	29.48	27.02	26.92	−0.67	−25.85
Vgg16	−4.91	39.76	38.34	39.42	−1.32	−13.27

Table 12.2: Evaluation of additional pretrained models (no finetuning). All models have higher accuracies under ablation, lower RFS scores, and lower saliency alignment on Hard ImageNet than RIVAL20.

In addition to the transformer and convolutional neural networks (DeiT and ResNet50) explored in the main text, we evaluate four other deep classifiers. Namely, we investigate Swin Transformer[358], ConViT[359], DenseNet161[360], and VGG16 [62]. As seen in table 12.2,

our results on new models corroborate the findings of the main text. Specifically, we see that across models, classifying Hard ImageNet objects leads to higher accuracy under ablation, lower RFS scores, and lower saliency alignment, compared to classifying RIVAL20 objects. This implies that certain properties inherent to the data in Hard ImageNet makes it far more challenging to learn to classify without heavily relying on spurious cues.

12.C Overview of Salient ImageNet

We refer readers to [336] for all details related to the Salient ImageNet-1M data and collection procedure. For completeness, we offer brief discussion of the methods relevant to this chapter. Namely, we elaborate on the way in which class-feature pairs were annotated as core or spurious (i.e. a neural feature was annotated as detecting input regions that were spurious with respect to the given class label). Recall that the motivation for closer inspection of Hard ImageNet classes was that all class-feature pairs for Hard ImageNet were annotated as spurious.

Salient ImageNet annotations first correspond to labeling 5 neural features as core or spurious for each of the 1000 ImageNet classes, resulting in 5000 class-feature pair binary annotations (core or spurious). Neural feature refers to the nodes in the penultimate layer of a deep classifier. Specifically, the neural features of an ℓ_2 adversarially trained ResNet50 were inspected because adversarially robust models have been observed to be more interpretable. For each class, the five neural features annotated were those that contribute the most to the logit of the given class. The average contribution of a neural feature to a class can easily be computed as the product of the average feature activation and the weight of the linear layer connecting the feature to the class logit.

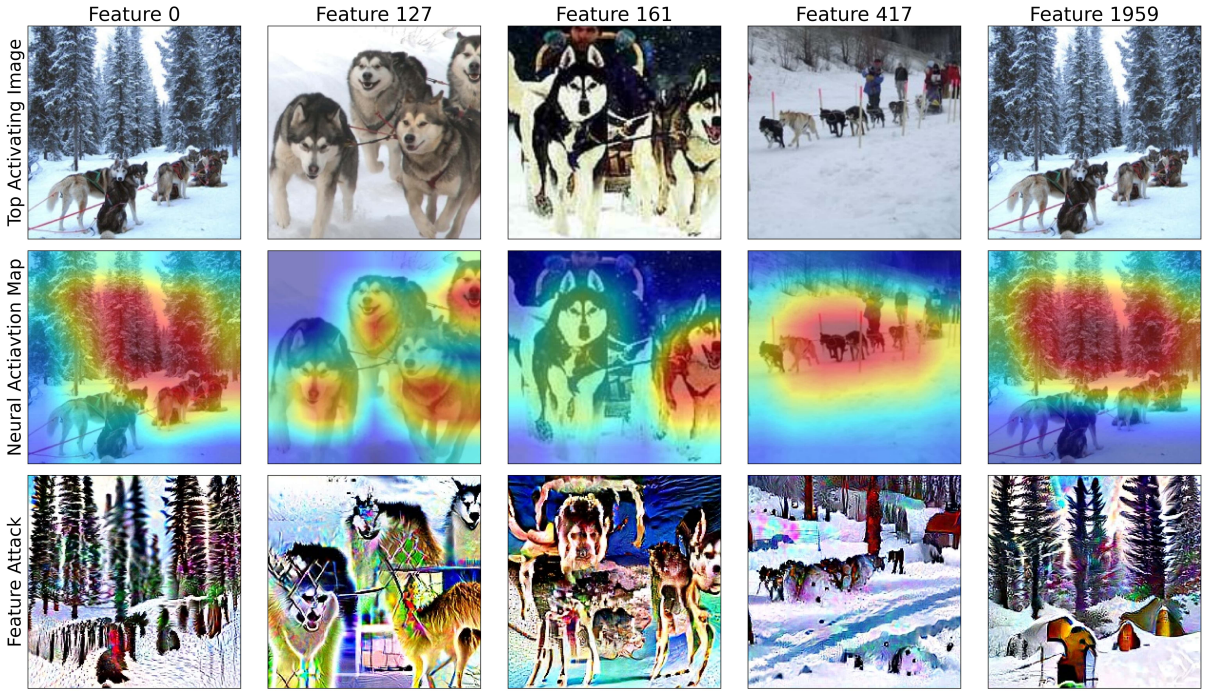


Figure 12.11: Example visualizations of the five most important neural features for the class **Dog Sled** used in Salient ImageNet annotation. From left to right, the features may be described as focusing on *trees, dogs, dogs, dogs in snow,* and *trees*.

Of the 5000 class-feature pairs annotated, 4370 (87.4%) were deemed to be core, signifying that in most cases, the model effectively learned to use the appropriate features in classification. For 342 classes, at least one feature was annotated as spurious. However, only a small minority (the 15 classes comprising Hard ImageNet) had all five features annotated as spurious. This motivated our hypothesis that there were inherent properties of the data in Hard ImageNet that leads standard supervised classification training algorithms to result in models that rely on spurious cues. We do not claim that the classes in Hard ImageNet have the strongest spurious cues, nor do we claim that models do not rely on spurious cues for classes outside of Hard ImageNet; only that strong spurious cues exist in Hard ImageNet, and studying this data can yields insights related to causes and solutions for image classifier reliance on spurious features.

There are three key visualization techniques applied in order to reveal the function a neural

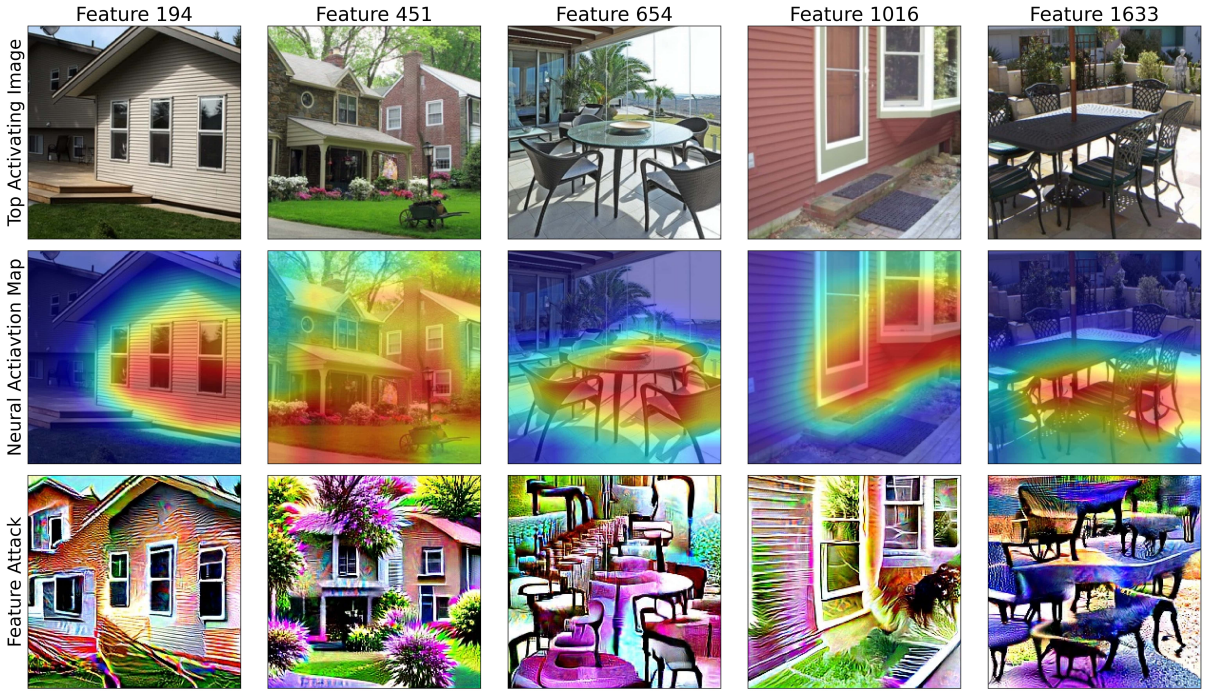


Figure 12.12: Example visualizations of the five most important neural features for the class **Patio** used in Salient ImageNet annotation. From left to right, the features may be described as focusing on *windows, house, furniture, window jambs,* and *patio chairs*.

feature serves over images from some class: natural images that highly activate the feature, ii. neural activation maps which highlight the input region responsible for the neural feature activation, and iii. feature attacks that optimize the input image to amplify feature activation. These visualizations are shown for the top five activating images per class-feature pair to five human annotators, who each vote to describe the focus of the feature as either on the main object (core) or a separate object or the background (spurious). The final annotation of the class feature pair is determined by majority vote.

We show the top activating image, its neural activation map, and a feature attack performed on it for each of the five features annotated for the Dog Sled and Patio classes in Figures 12.11 and 12.12 respectively. Visualizations for all Hard ImageNet classes (as well as the rest of ImageNet) can be viewed here: `www.salient-imagenet.cs.umd.edu`.

Lastly, we note that Salient ImageNet-1M also consists of soft segmentations masks for the objects for all images, *except* for those belonging to Hard ImageNet classes. This discrepancy is because the soft segmentation masks are constructed from the neural activation maps of core features. Thus, since Hard ImageNet classes have no annotated core features, Salient ImageNet-1M lacks segmentations for those classes. Therefore, the segmentations collected for the Hard ImageNet dataset effectively complete Salient ImageNet-1M. These masks can be potentially leveraged to train more reliable models, though this is an open research problem with little existing work, since annotations of this kind (segmentations) have not been prevalent for classification at scale until these recent works.

12.D Inspecting Prediction Confidence on Ablated Images

One may argue that it is unreasonable to fault a model for classifying an ablated image to its original class, particularly when it is not an option to predict some other more suitable class. After all, the model simply returns probabilities that an image belongs to each class, and chooses the class that is most likely. A similar metric would be to inspect prediction confidence instead of accuracy. This way, we no longer directly fault a model for still predicting the original class, but still reward the calibration of a model. That is, it may be more reasonable to hope a model at least predicts an ablated image to the true class with far less confidence.

We explore this related metric in this section so to validate our ablation analyses using the more canonical (though potentially slightly more problematic) accuracy measure. Quite simply, we aggregate prediction confidences of the true class (not the predicted class) for each ablated image, and view the average confidence. As figure 12.13 shows, we very closely corroborate the

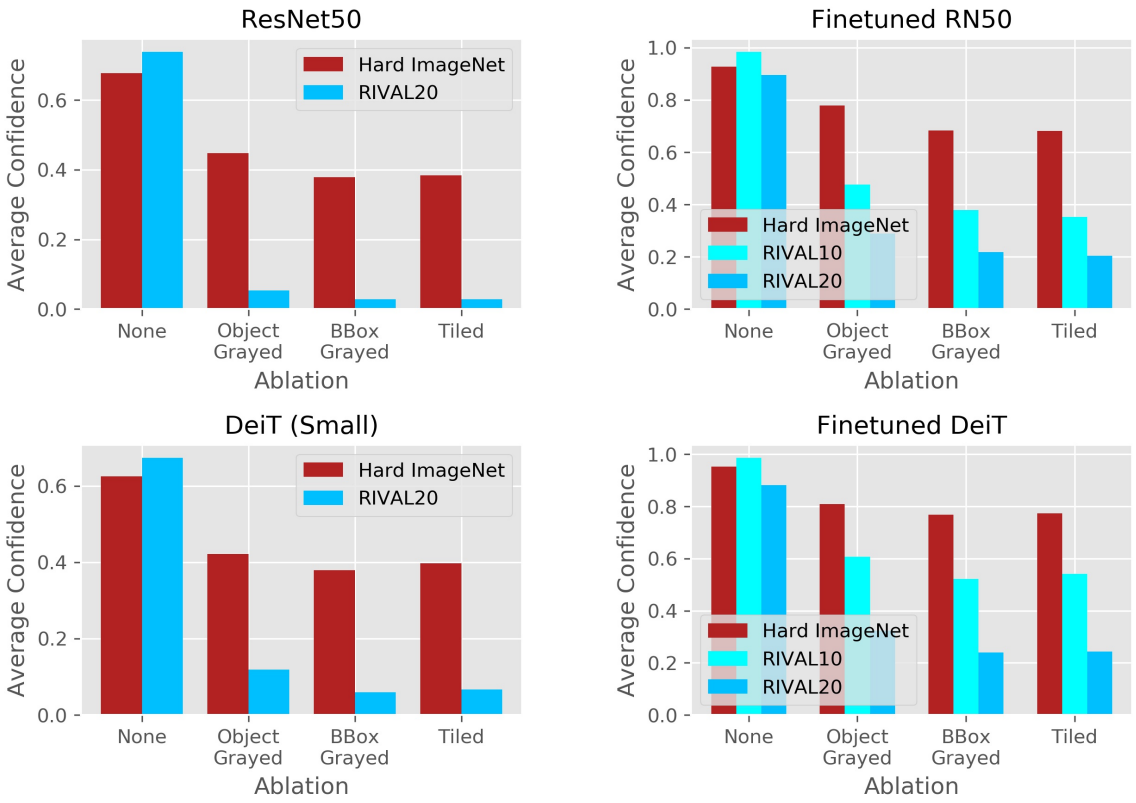


Figure 12.13: Probability (confidence) of true class under ablation. Confidence drops much less when Hard ImageNet objects are ablated than when RIVAL10 objects are ablated, exactly as observed for accuracy under ablation.

findings obtained when inspecting accuracy under ablation.

While using accuracy under ablation directly may be imperfect, we find that it is an intuitive measure that may be more easily interpretable than our noise or saliency based metrics. Furthermore, accuracy is the standard evaluation metric for classification, and is highly correlated with true class prediction confidence, which as detailed above, reveals analogous findings and is less affected by the fact that desired classification behavior on ablated images is unclear. Thus, we present accuracy in the main text, though we argue that either accuracy or true class prediction confidence under ablation can be used in practice. We provide implementations for both metrics.

12.E Datasheet

We now share more detail on our dataset, following the *Datasheets for Datasets* protocol [361]. Access all code and data at the following link: `mmoayeri.github.io/HardImageNet`.

12.E.1 Motivation

Hard ImageNet was created to assess and improve image classifier capacity to learn to objects that commonly occur with strong spurious cues. We hypothesized that despite high classification accuracy, models were incorrectly learning the objects corresponding to Hard ImageNet classes. Going beyond single class-label annotations allowed for quantitative demonstration of this undesirable (and otherwise undetectable) behavior, as well as opening the door to new ways of improving models on these challenging objects. The dataset was created by academics (namely from the University of Maryland) for academic purposes, leveraging crowd annotations through Amazon's Mechanical Turk platform. Data collection was funded by an AWS Machine Learning

Research Award.

12.E.2 Composition

Each instance consists of an image with a label and a binary segmentation mask corresponding to the class object. Instances either fall in the training or validation split, which is consistent with ImageNet's split. Images in the training split additionally are ranked within their respective class by the strength of spurious cues present, as determined in an automated procedure leveraging the neural feature annotations of Salient ImageNet-1M [336]. Instances in the validation split are unlikely to be noisy, as they consolidate five separate repetitions of annotations, while training set segmentations may be noisier, though quality is generally ensured via qualification exams and attention checks. Generally, the dataset does not relate to people, though many images do contain people (in fact, they are a common spurious cue). It is unlikely that the data can be used to identify any individuals are subpopulations, and we note that these pitfalls are inherited from the standard benchmark dataset ImageNet, from where Hard ImageNet images are drawn. Nonetheless, we advise caution in using and sharing images containing faces or otherwise prominently displaying people; we attempted to avoid including such images in our figures to the best of our ability.

12.E.3 Collection Process

Images were drawn directly from ImageNet. Segmentations were collected via Amazon Mechanical Turk. Image rankings were computed by inspecting the activations of particular neural features in a ℓ_2 adversarially trained ResNet50 using an attack budget of $\epsilon = 3$ (see [318] for more detail). Segmentations were validated in the sense that quality was monitored via attention

checks, where annotators consistently achieved high IoUs with ground truth segmentations (average IoU of 0.76). Validation set segmentations were validated across one another, by having five separate annotation rounds and taking a pixel-wise majority to obtain final segmentations. The people involved in data collection were the first author and roughly 50 crowdworkers. The crowdworkers were paid 0.2 per segmentation, amounting to about $\$12$ to $\$16$ per hour. Workers were also eligible for bonuses of $\$1, \$3$ or $\$7$ for submitting $100, 250,$ or 500 segmentations respectively. Workers could collect a maximum of two bonuses: one for training images and one for validation images. We did not obtain IRB approval as our data annotation does not constitute human subject research as defined in federal regulation 45 CFR 46.102. Specifically, because we do not ask the human annotators information about themselves, they are not technically human subjects. We consulted our institution's IRB to confirm that our study is exempt from approval. Nonetheless, we closely followed principles learned from the history of human subject research, such as providing informed consent (see 12.15), ensuring the rights of the participants, anonymizing responses, keeping work entirely transparent, voluntary, and justly compensated. We strived to uphold the highest ethical standards in our procedures and actively maintain a healthy environment for our workers. Annotations were collected in a two week span.

12.E.4 Preprocessing/cleaning/labeling

The only data cleaning performed was the consolidation of multiple rounds of annotations for validation set segmentations via pixel-wise majority vote.

12.E.5 Uses

The dataset has not been used yet. We have developed a suite of evaluation metrics and demonstrated the utility of the dataset to improve model performance. We hope new training methods can be developed leveraging the richer annotations of our dataset (relative to standard single label classification datasets). The dataset is not intended to replace large scale diverse datasets (e.g. ImageNet), but instead focuses on the specific subproblem of faithful learning despite strong spurious cues. Data and evaluation code will be publicly available and accessible at `mmoayeri.github.io/HardImageNet`.

12.E.6 Distribution

The dataset will be publicly distributed immediately upon submission. There are no limitations on use of this data.

12.E.7 Maintenance

The authors will maintain the dataset website and answer any question regarding usage. We encourage questions to be asked via GitHub, though the authors can be contacted directly. The primary author can be emailed at mmoayeri@umd.edu. There are no current plans to release new versions to this dataset, though if that does occur, old versions will remain archived.

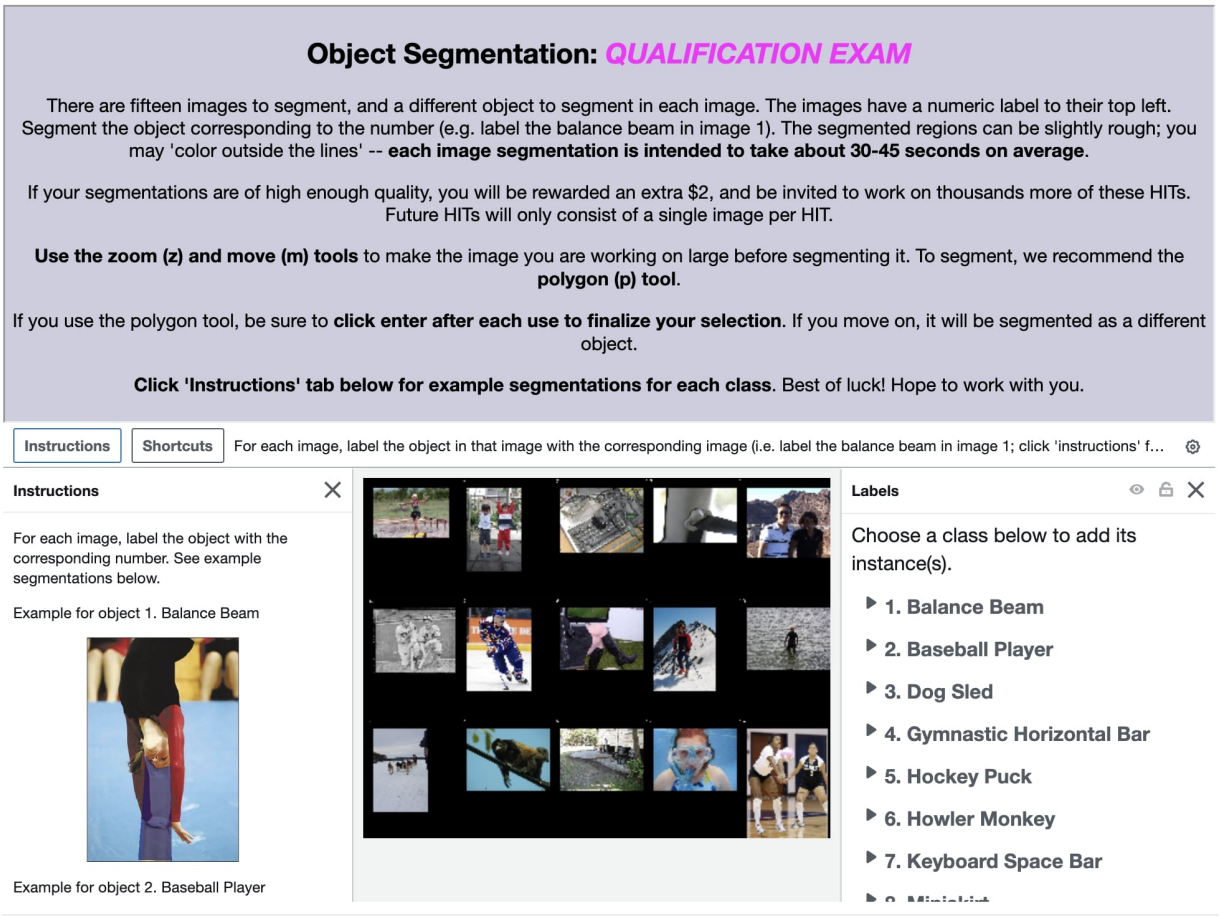
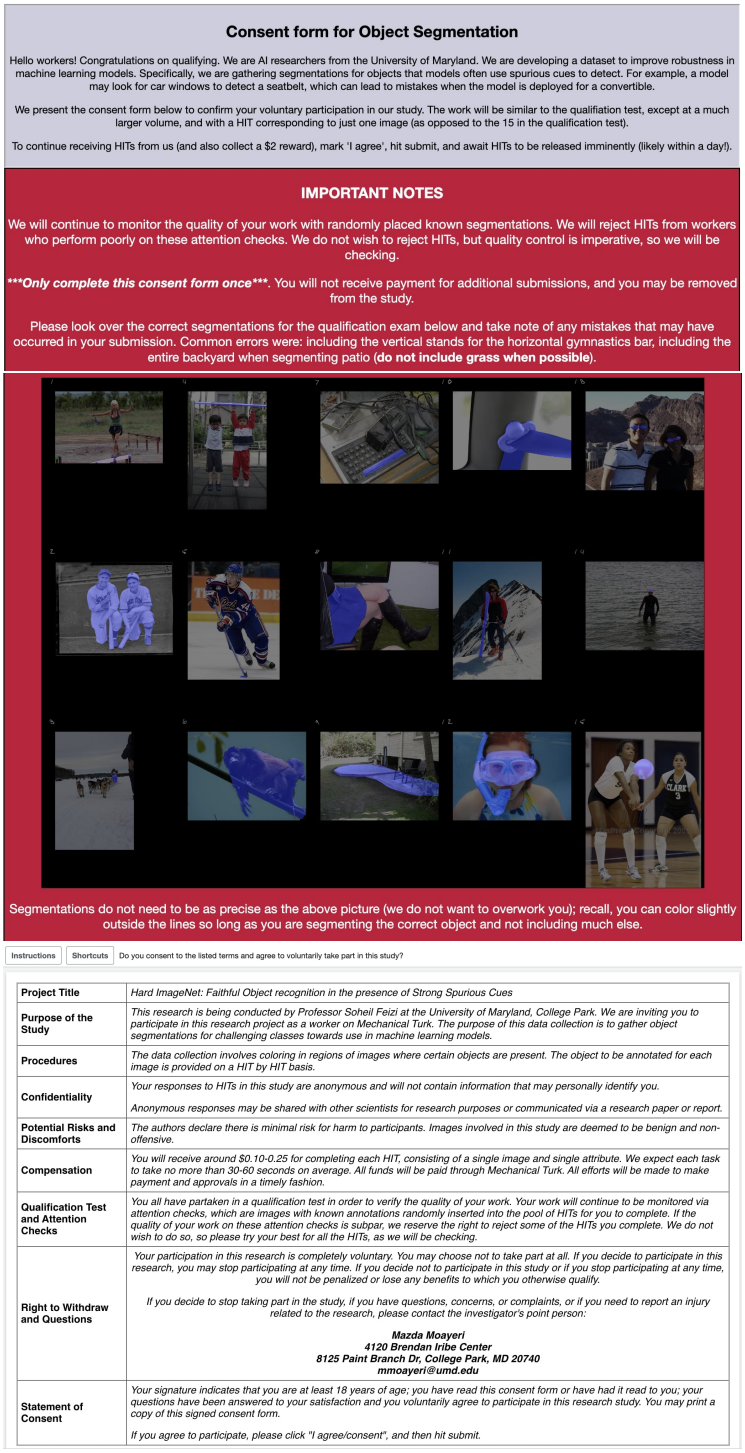


Figure 12.14: Qualification exam.

12.F Mechanical Turk Forms

We now provide screenshots for all Amazon Mechanical Turk Forms used to facilitate data collection in our study. For full transparency, we leave copies these forms up on the free analog of Mechanical Turk so that any interested parties can view and familiarize themselves with the annotation platform. The forms are listed at the following link: `https://workersandbox.mturk.com/requesters/ATCCTSC7WNN97/projects`. Figures 12.14, 12.15, and 12.16 show the forms for the qualification exam, information/consent phase, and full data collection respectively.



Consent form for Object Segmentation

Hello workers! Congratulations on qualifying. We are AI researchers from the University of Maryland. We are developing a dataset to improve robustness in machine learning models. Specifically, we are gathering segmentations for objects that models often use spurious cues to detect. For example, a model may look for car windows to detect a seatbelt, which can lead to mistakes when the model is deployed for a convertible.

We present the consent form below to confirm your voluntary participation in our study. The work will be similar to the qualifiation test, except at a much larger volume, and with a HIT corresponding to just one image (as opposed to the 15 in the qualification test).

To continue receiving HITs from us (and also collect a $2 reward), mark 'I agree', hit submit, and await HITs to be released imminently (likely within a day!).

IMPORTANT NOTES

We will continue to monitor the quality of your work with randomly placed known segmentations. We will reject HITs from workers who perform poorly on these attention checks. We do not wish to reject HITs, but quality control is imperative, so we will be checking.

******Only complete this consent form once******. You will not receive payment for additional submissions, and you may be removed from the study.

Please look over the correct segmentations for the qualification exam below and take note of any mistakes that may have occurred in your submission. Common errors were: including the vertical stands for the horizontal gymnastics bar, including the entire backyard when segmenting patio (**do not include grass when possible**).

Segmentations do not need to be as precise as the above picture (we do not want to overwork you); recall, you can color slightly outside the lines so long as you are segmenting the correct object and not including much else.

Instructions | Shortcuts | Do you consent to the listed terms and agree to voluntarily take part in this study?

Project Title	*Hard ImageNet: Faithful Object recognition in the presence of Strong Spurious Cues*
Purpose of the Study	*This research is being conducted by Professor Soheil Feizi at the University of Maryland, College Park. We are inviting you to participate in this research project as a worker on Mechanical Turk. The purpose of this data collection is to gather object segmentations for challenging classes towards use in machine learning models.*
Procedures	*The data collection involves coloring in regions of images where certain objects are present. The object to be annotated for each image is provided on a HIT by HIT basis.*
Confidentiality	*Your responses to HITs in this study are anonymous and will not contain information that may personally identify you.* *Anonymous responses may be shared with other scientists for research purposes or communicated via a research paper or report.*
Potential Risks and Discomforts	*The authors declare there is minimal risk for harm to participants. Images involved in this study are deemed to be benign and non-offensive.*
Compensation	*You will receive around $0.10-0.25 for completing each HIT, consisting of a single image and single attribute. We expect each task to take no more than 30-60 seconds on average. All funds will be paid through Mechanical Turk. All efforts will be made to make payment and approvals in a timely fashion.*
Qualification Test and Attention Checks	*You all have partaken in a qualification test in order to verify the quality of your work. Your work will continue to be monitored via attention checks, which are images with known annotations randomly inserted into the pool of HITs for you to complete. If the quality of your work on these attention checks is subpar, we reserve the right to reject some of the HITs you complete. We do not wish to do so, so please try your best for all the HITs, as we will be checking.*
Right to Withdraw and Questions	*Your participation in this research is completely voluntary. You may choose not to take part at all. If you decide to participate in this research, you may stop participating at any time. If you decide not to participate in this study or if you stop participating at any time, you will not be penalized or lose any benefits to which you otherwise qualify.* *If you decide to stop taking part in the study, if you have questions, concerns, or complaints, or if you need to report an injury related to the research, please contact the investigator's point person:* ***Mazda Moayeri*** ***4120 Brendan Iribe Center*** ***8125 Paint Branch Dr, College Park, MD 20740*** ***mmoayeri@umd.edu***
Statement of Consent	*Your signature indicates that you are at least 18 years of age; you have read this consent form or have had it read to you; your questions have been answered to your satisfaction and you voluntarily agree to participate in this research study. You may print a copy of this signed consent form.* *If you agree to participate, please click "I agree/consent", and then hit submit.*

Figure 12.15: Consent Form. Workers who passed the qualification exam then moved on to sign a consent form, where the purpose of their work was explained, common mistakes were corrected, and an extra payment was awarded. This phase is intended to create an active dialogue between data annotators and collectors. We received many inquiries and enthusiastic bits of feedback.



Figure 12.16: Example task for full data collection phase, only accessible to workers who passed the qualification exam and signed the consent form.

Chapter 13: Spuriosity Rankings: Sorting Data for Spurious Correlation Robustness

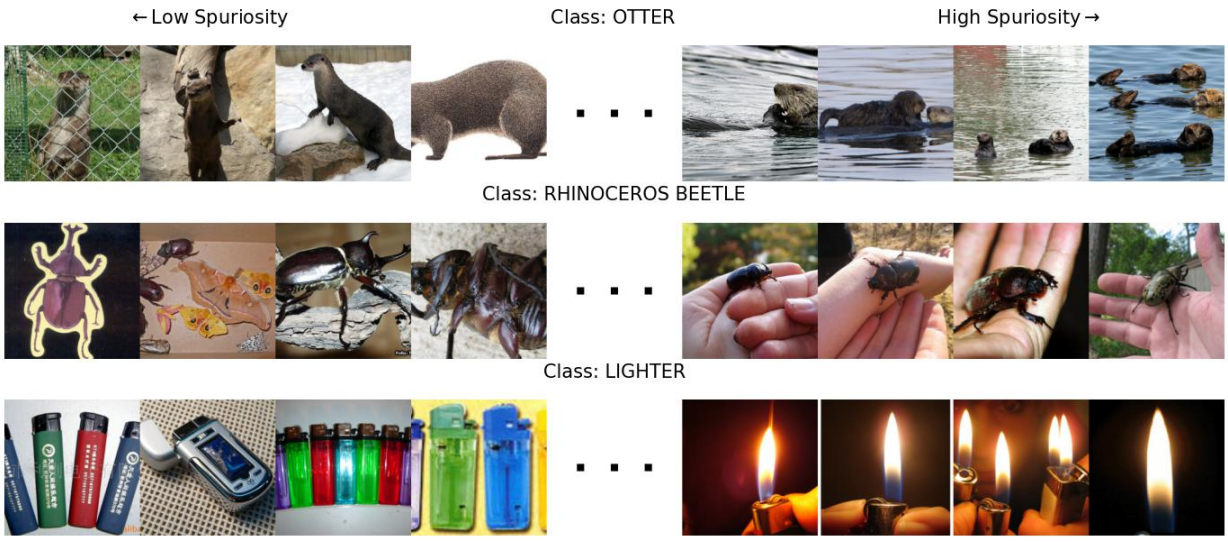


Figure 13.1: Spuriosity rankings sort images based on the presence of spurious cues. The rankings reveal hidden spurious features like *rippling water, human hands,* and *flame in the dark* (right; top to bottom) and identify minority subgroups where spurious cues are absent (left).

13.1 Introduction

The vast size of modern deep networks and their training sets obscures the data patterns they rely on and the degree to which they do so. Moreover, deep models are prone to depending on spurious features [362, 363, 364], which are predictive of a class within the training distribution but irrelevant to the true labelling function. Spurious feature dependence hinders robust

generalization (e.g. medical systems trained in one hospital may fail when deployed to others [17, 18]) and can reduce fairness (e.g. facial recognition and recidivism prediction systems can become biased to race [273, 365]).

Many algorithmic solutions have been proposed to mitigate spurious feature dependence [315, 332, 339, 343], though they often require some supervision about whether the spurious feature is present or not (typically in the form of a binary label). However, re-annotating data is expensive and necessitates knowledge of some single spurious cues a model may rely on apriori. Discovering spurious cues is non-trivial, and further, binary labelling of the presence of an attribute may be misleading, as an attribute can be more prevalent in one sample than another. Furthermore, the spurious correlation problem is crucially class-dependent. As shown in Figure 13.2, the concept *pine trees* is spurious to the class *ski*, but core to the class *lakeshore*, since they are a part of the lakeshore. Thus, existing paradigms of adding binary annotation of a single class-agnostic spurious feature may be incomplete.

Instead, a procedure to reveal when and to what degree spurious features (specific to each class) are present would provide richer signal to existing methods for spurious correlation robustness. We propose to *rank images within their classes* with respect to *spuriosity* (i.e. presence of cues spurious to the class). This ranking provides natural interpretation of the spurious features by presenting them in context (i.e. in highly ranked images) and illuminating minority subpopulations where the features are absent (i.e. lowly ranked images). For example, in Figure 13.1, we can immediately identify spurious features (i.e. water for an otter) and minority subgroups with spurious correlations broken (e.g. otters in rare contexts like snow or grass) by inspecting images with highest and lowest spuriosity ranks respectively.

It has been shown that neural features (i.e. nodes in the penultimate node of a trained

deep network), particularly those that contribute highly to a specific class logit, encode concepts relevant to the class [319]. Thus, the activation of a neural feature can serve as an automated proxy for the degree to which a concept is present. We use robust neural features due to their enhanced interpretability [335], and after annotating 5000 class-feature dependencies using minimal human supervision, we discover that at least 357 classes of ImageNet contain spurious cues that models may rely upon. Then, per class, we rank both training and validation images by their spuriosity using activations of robust neural features deemed spurious for the class. Indeed, visual inspection of the highest and lowest ranked images confirms that using neural features as proxies to concepts that humans deem as spurious is effective, as images with the highest and lowest spuriosity differ markedly in the non-core features present (see Figure 13.1).

With spuriosity rankings, we first assess the reliance of 89 models with diverse architectures (attention, conv, and mlp-based) and training methods (adversarial, vision + language, SSL, etc) on spurious cues using our *spurious gap* metric (i.e. drop in accuracy on highest vs. lowest ranked images). We observe CLIP [366] and adversarially trained [20] models to be the least and most biased respectively to images with high spuriosity, consistent with prior work [337, 367] and thus further validating our rankings. *All models* underperform on images with low spuriosity, stressing the hidden bias of models to spurious cues is still pervasive. More striking, though, is the variance in spurious gap *per class*, which is $3\times$ as high as the variance across models.

A common belief in the community is that model accuracy drops when spurious correlations are broken in the test time. In our experiments, we observe this for many classes, reflecting that test-time spurious cues often help *clarify* the class. However, some classes actually have higher test-time accuracy on images *without* spurious cues. Here, the spurious features are *confusing* because they are also relied upon, more strongly, by other classes. For example, in predict-

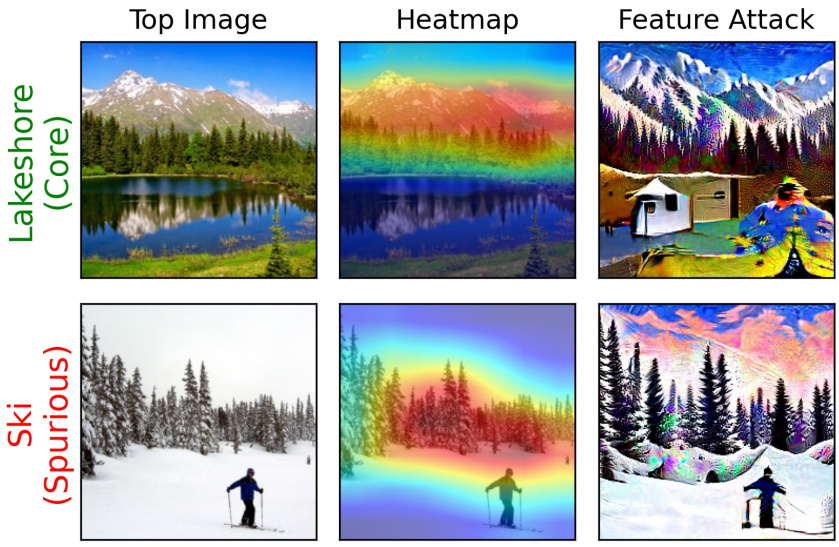
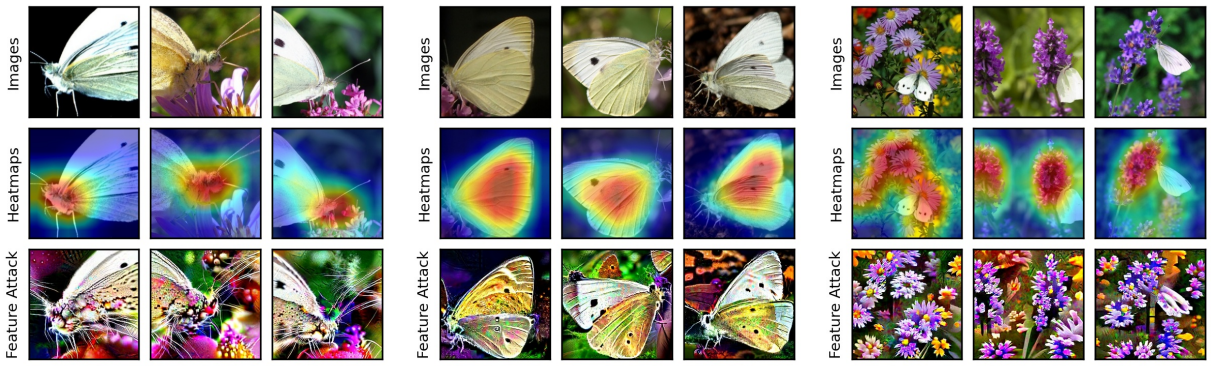


Figure 13.2: A feature (e.g. mountain pine trees) can be core or spurious *depending on the class*, as shown through our complementary visualizations: (**left**) top activating images, (**middle**) heatmaps showing regions activating the feature in red, and (**right**) feature attacks that perturb the image to visually amplify the feature.

ing the class *car wheel*, a model may look for the body of a car. However, if the car body is very prominent, the model will incorrectly predict a *car* class as opposed to *car wheel*. Indeed, models are on average 54.2% less accurate on *car wheel* images with high spuriosity. **The existence of both clarifying and confusing spurious features underscores the crucial class dependence and previously overlooked nuance in the spurious correlation problem.**

We then train linear heads on subsets of data determined by our ranking. Training on images with low spuriosity closes the spurious gap, removing the bias towards the spurious feature. Further, this comes at no drop in out-of-distribution accuracy, suggesting that in-distribution accuracy gains due to spurious feature reliance offer an inflated sense of model performance. Conversely, including low spuriosity images may be vital for distributional robustness.

We further probe the role of core and spurious features for classification in and out of distribution with a second metric that assesses the reliability of a neural feature directly. We find



(a) **Core** feature: antennae, hair (b) **Core** feature: wing (c) **Spurious** feat.: purple flower

Figure 13.3: Examples of visualizations used to annotate important neural features as core or spurious for *small white butterfly* class.

that core features are generally more reliable, but for some classes, surprisingly only spurious features are reliable, highlighting yet another complexity in which spurious features can actually help classification, even under distribution shift.

Finally, we extend our robust-neural-feature-based ranking framework to new domains with *only retraining a linear layer* (i.e. the simplest transfer learning setting), expanding its use cases to low data regimes where adversarial training would be unstable. The lightweight extension of our framework detects designated spurious attributes for Waterbirds and Celeb-A classification tasks. Furthermore, **we uncover a previously unobserved racial bias in the Celeb-A hair classification task, demonstrating our framework's ability to identify hidden spurious features.**

13.2 Related Work

Spurious correlation benchmarks often only consider a small set of predefined spurious attributes fixed across classes [332, 333, 347]. Real-world benchmarks usually are very specific to a narrow task [334], as opposed to our benchmark which seeks to understand the role of spurious features in the general and ubiquitous ImageNet dataset [56]. To study the related distributional

robustness problem for ImageNet models, it is standard to assess accuracy on variants like ImageNet-R, -Sketch, and ObjectNet [348, 355, 368], which present objects in altered rendition, as sketches, or in household settings (instead of natural contexts) respectively, as we do in Section 13.5. Other approaches to measuring sensitivities require synthetic data [315, 350, 369] or corrupting spurious regions, which undesirably takes models out of distribution [337, 370]. In contrast, our benchmark operates only on natural images within the validation set, using spuriosity to obtain subsets with spurious cues strongly present and absent.

Natural image-based interpretability is typically used to explain inferences, based on examples from the corpus [371, 372], or on generated examples intended to visually reflect a traversal between concepts [373], similar to how our rankings traverse from strong spuriosity to minority examples with spurious cues absent, though we do not require generative models. Other related approaches are influence functions [325] and data models [374], though they are computationally expensive and the former can be fragile [375].

Mitigating spurious correlation dependencies often involves altering the training objective to learn more invariant features [315, 344] or directly optimize worst group accuracy [332, 342, 376]. Many methods require at least some extra supervision with respect to non-class-label attributes [377]. Retraining approaches either focus on correcting errors [343, 378] or using balanced data (i.e. dominant spurious cues balanced with minority samples) [339]; our rankings would directly assist in constructing such balanced subsets.

Neural features have been examined to interpret models [289, 379]. Robust neural features have also been examined and observed to have enhanced interpretability [318, 319]. Most notably, [335] introduces a framework based on robust neural features for discovering core and spurious data patterns with minimal human supervision. We apply this framework to empirically

compute spuriosity ranks at scale. Key contributions of spuriosity rankings to this framework include enabling assessment of feature dependence without relying on corruption (which precludes comparison across models), identifying minority subpopulations, quantifying feature reliability, and extending the framework to new domains without requiring adversarial training.

13.3 Defining and Computing Spuriosity

Models often rely on many data patterns to recognize each class. Suppose we identify a set $\mathcal{S}(i)$ of cues predictive of class label i in a spurious way. These cues can be abstract (e.g. *sand* helps predict *fiddler crab*), and their presence in an image may not be binary (e.g. there is more *sand* at a beach than in a terrarium). We define *spuriosity* as follows.

Definition 17. ***Spuriosity (conceptual)****: The average strength of spurious cues $\mathcal{S}(i)$ for a sample x from class i.*

The 'strength' of a spurious cue is purely qualitative. However, we can quantitatively proxy this notion via activations of penultimate nodes (called *neural features*) in trained classifiers. Neural features have been observed to respond to stimuli corresponding to concepts related to the classes the model is trained to predict [289]. Further, since classifiers are composed of linear layers operating over neural features, it is easy to select features that contribute the most to a class logit by inspecting linear layer weights, indicating that the cues they detect pertain to the class.

Specifically, we use *robust neural features* (i.e. from an adversarially trained network), because of their enhanced interpretability. Namely, as shown in [335], one can reliably generate heatmaps highlighting input regions activating a robust neural feature, and even perturb an input image to visually amplify the cue that the feature detects. Combined with showing the highest-

ranked images for a cue, annotation of a robust neural feature's function as core or spurious becomes feasible. Figure 13.3 exemplifies these visualizations for three cues relevant to the *small white butterfly* class, including a spurious cue (purple flowers).

We apply the feature discovery framework preliminary introduced in [335] at a much larger scale to collect class-wise spurious dependencies across ImageNet. We annotate 5000 class-feature dependencies (namely the five features per class contributing most to the corresponding class logit), of which 630 are spurious over 357 classes. We will maintain a web interface for easy viewing of our analysis, offering a direct visual look into the patterns a neural network sees and uses across ImageNet. We also verify that heatmaps for annotated features localize the same cue in images whose activation on the feature is in the top 20th percentile for the class, successfully validating 95.3% of annotated class-feature pairs (see Supp. for details). Thus, we generate $325,000$ feature segmentations across ImageNet as a bonus; we characterize model sensitivity to these regions in the Supp.. Crucially, the validation confirms that sorting by feature activation is effective in gathering instances where a feature is present.

Thus, we can now *compute* spuriosity empirically, leveraging the annotated pairs of classes and robust neural features by simply assessing activation on the features. We use pre-relu activations to better distinguish low-activating images. Formally, letting $\mathbf{r}_j(x)$ denote the activation of image x on the jth robust neural feature, and $\tilde{\mathcal{S}}(i)$ consisting of the set of robust neural features annotated as spurious for class i, we can re-define spuriosity as follows.

Definition 18. ***Spuriosity (empirical)*** *of x for class i, with μ_{ij} and σ_{ij} denoting the mean and standard deviation of activations on feature j over class i, is* $\frac{1}{|\tilde{\mathcal{S}}(i)|} \sum_{j \in \tilde{\mathcal{S}}(i)} \frac{\mathbf{r}_j(x) - \mu_{ij}}{\sigma_{ij}}$.

Defined empirically, we can now compute the spuriosity of all images, and rank them ac-

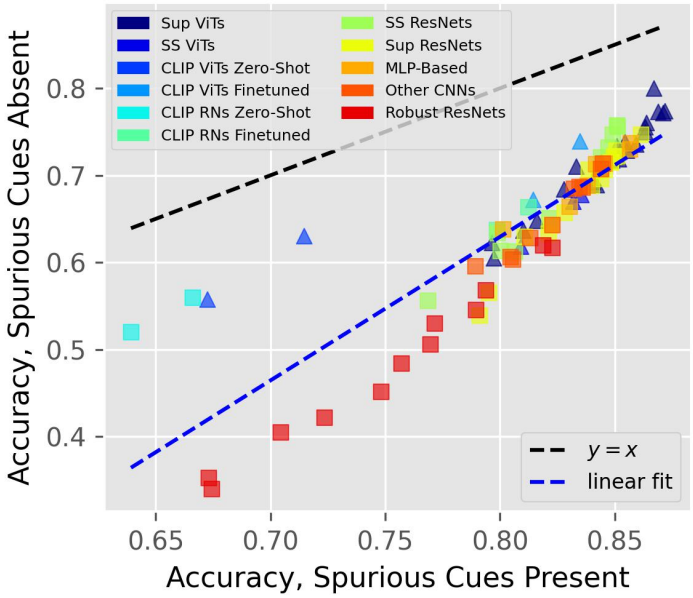


Figure 13.4: Model accuracy with spurious cues most/least present (i.e. accuracy on top-10 highest/lowest spurious ranked validation images respectively). While zero-shot CLIP models exhibit the most *effective robustness* and adversarially trained models exhibit the least, most other models behave roughly similarly.

cordingly. Note that the ranking must be within a class, as the spurious features one considers depends on the class. Thus, our spuriosity rankings consist of ordering each image within its class (and data split) based on its spuriosity. As shown in Figure 13.1, our ranks organize images into groups that visually differ in spurious features. This suggests that while we use the neural features of a specific network in computing spuriosity empirically, the patterns we measure capture semantic concepts that pertain to the data, and thus affect all models.

13.4 Assessing Spurious Feature Reliance

Using spuriosity rankings, we define ***spurious gap***, a metric to assess the reliance of a model on spurious features. Spurious gap is simply the difference in accuracy over ImageNet validation images with the ten highest and ten lowest spuriosity ranks per class. A high spurious gap means

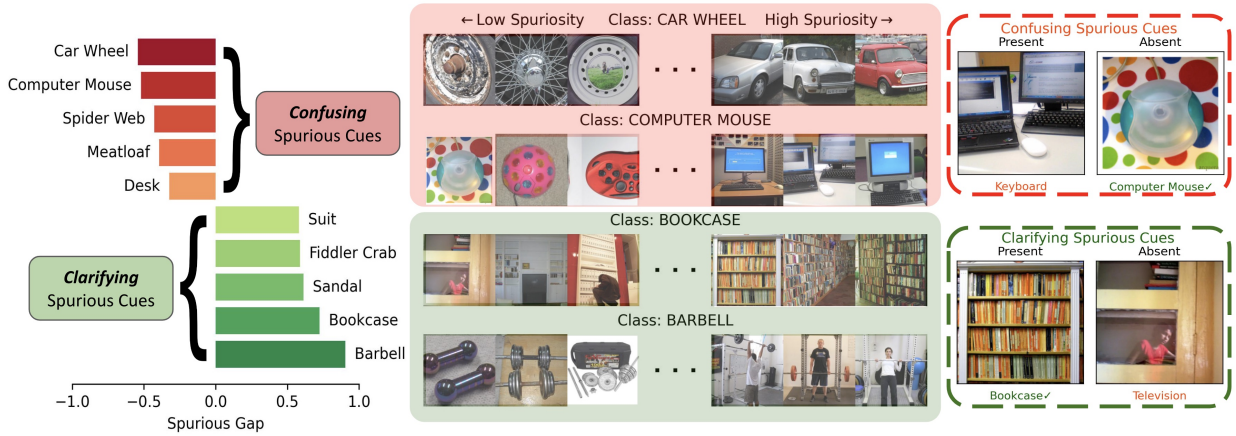


Figure 13.5: The effect of spurious features depends strongly on the feature and class. Spurious features can be clarifying: it is easier to recognize a bookcase when there are books on it. Spurious features can also be confusing when they are relied upon (often more strongly) by other classes, leading to worse performance when the cue is strongly present.

that models perform worse when spurious features are absent, indicating reliance on those features. We note since only 357 classes had at least one of the five most relevant neural features annotated as spurious, spurious gap is evaluated over this subset of classes. Again, this reflects the data dependency of the spurious correlation problem: for some classes, all annotated features are observed to be core (as desired), while for others, they are all spurious.

With the spurious gap metric, we perform a large-scale evaluation of 89 diverse pretrained models (details in Supp.). Figure 13.4 shows accuracy on the highest spurious ranked images (x-axis) vs. accuracy on the lowest spurious ranked images (y-axis) over all 357 classes. For all models, accuracy is higher when spurious features are present, matching the prevailing thought that the drawback of spurious feature reliance is that models underperform when spurious correlations are broken. As in [380], we observe a strong correlation between the two accuracy measures. However, some models stray from this linear trend, exhibiting 'effective robustness' by performing significantly better on images with spurious cues absent than the linear trend predicts (e.g. zero-shot CLIP) or the opposite (e.g. adversarially trained models). Prior work

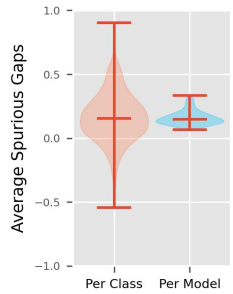


Figure 13.6: Spurious gap varies much more across data classes than models.

corroborates our findings on the robustness (and lack thereof) of CLIP and adversarially trained models [366, 367], thus validating our metric.

We now turn our attention to *class-wise* spurious gaps (i.e. we average spurious gaps per class over all models, instead of averaging over all classes per model). We find that spurious gap varies far more across classes than across models (Figure 13.6). Specifically, the sample variance is $3.1\times$ larger for spurious gaps across classes than across models, suggesting that the spurious correlation problem pertains more to data than to models. We remark that averaging over classes completely obscures this rich class-wise signal, as is standard practice; we encourage closer inspection of class-dependencies in other problems as well.

Digging deeper into the observed class variance, we visualize classes with extreme spurious gaps in figure 13.5. Usually, model performance drops when spurious cues are removed at the test time, as is typically expected, indicating that the spurious cues help *clarify* the object class (e.g. it is easier to recognize a *bookcase* when *books* are present). However, surprisingly, for some classes, models actually perform significantly worse when spurious cues are present. We attribute this to *confusing* spurious cues, where the cue is also relied upon in predicting other classes. For example, it may be useful to recognize the body of a car when predicting the *car wheel* class. However, a *car body* feature would naturally be relevant in predicting car classes, and if that

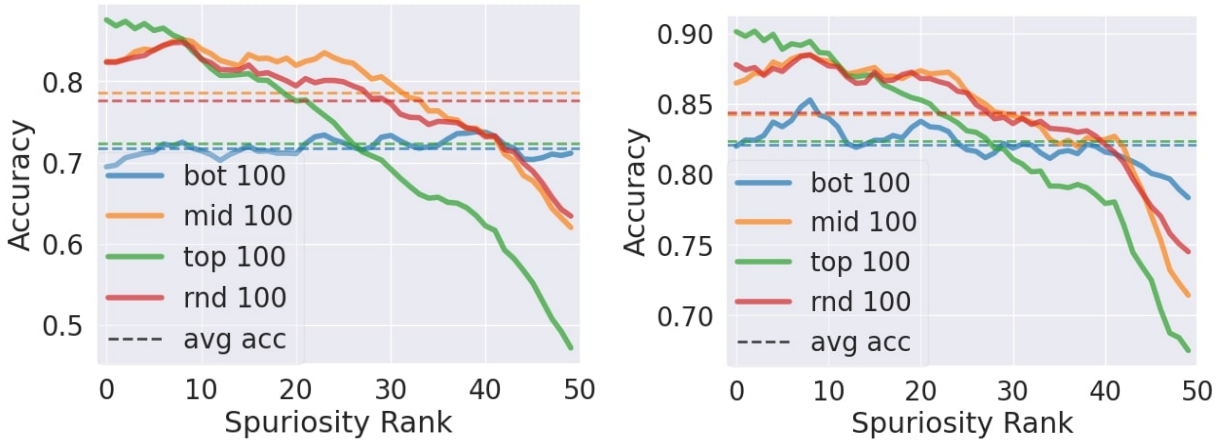
feature is strongly present, a model would more often wrongly predict a car. In summary, the effect of relying on spurious features is nuanced and strongly dependent on the features, the class, and how other classes also use the feature. Given the strong class-dependence on how spurious feature affect model performance, class-agnostic approaches to improving spurious correlation robustness may be doomed to fail.

13.5 Closing Spurious Gaps

Previously, we focused on spuriosity rankings of validation samples. Now, we use spuriosity rankings of the training set, so to close spurious gaps and assess the impact of spurious cues on OOD generalization.

Setup. Given a pretrained model (ResNet-50, DeiT-small), we simply retrain its last layer (i.e. the linear classification head) from scratch, similar to [339], on different subsets of the training data that may or may not depend on spuriosity rankings: *bot k*, the k samples from each class with the lowest spuriosity rank; *mid k*, the k samples from each class with closest ranks to the median spuriosity rank; *top k*, the k samples from each class with the highest spuriosity rank; *rnd k*, k randomly selected samples from each class. For evaluation on ImageNet, ImageNet-R, ImageNet-Sketch, and ObjectNet, we evaluate only accuracy on samples corresponding to classes with at least one annotated spurious feature.

Reassessing Standard Accuracy and the Toxic Side of Spuriosity. In Figure 13.7. we present the accuracy of the classification heads (trained using 100 training samples per class) on validation samples with varying spuriosity ranks. Firstly, the spurious cues in the training set actually increase the standard accuracy as indicated by the dashed, horizontal lines, where *mid*



(a) ResNet-50 (b) DeiT-small

Figure 13.7: The accuracy of classification heads trained on different subsets of the training data with varying spuriosity ranks. The dased, horizontal lines correspond to the standard accuracy of different classification heads. For better readability, curves are smoothed using a sliding window of size 5 over spuriosity ranks.

100, top 100, rnd 100 offer higher standard accuracy than *bot 100*, the classification heads learned from samples without strong spurious cues.

Does this mean spuriosity is favored? No. As in Figure 13.7, training on samples with spurious cues results in degradation of accuracy on samples without strong spurious cues (i.e. the lowly ranked validation samples). In this sense, the top-ranked training samples with the most spuriosity can be viewed as a form of poisoned samples, including them in training is associated with clear drops of average performance on a particular subset, i.e. the samples with few spurious cues. Meanwhile, the classification heads trained on only the lowest ranked training samples offer consistent performance for validation samples across different spuriosity ranks (i.e. flat curves in Figure 13.7). While this comes at a cost of in-distribution accuracy, others may argue reducing disparity in performance across groups may be preferred to maximizing overall accuracy; such ideas are prevalent in the fairness community [381].

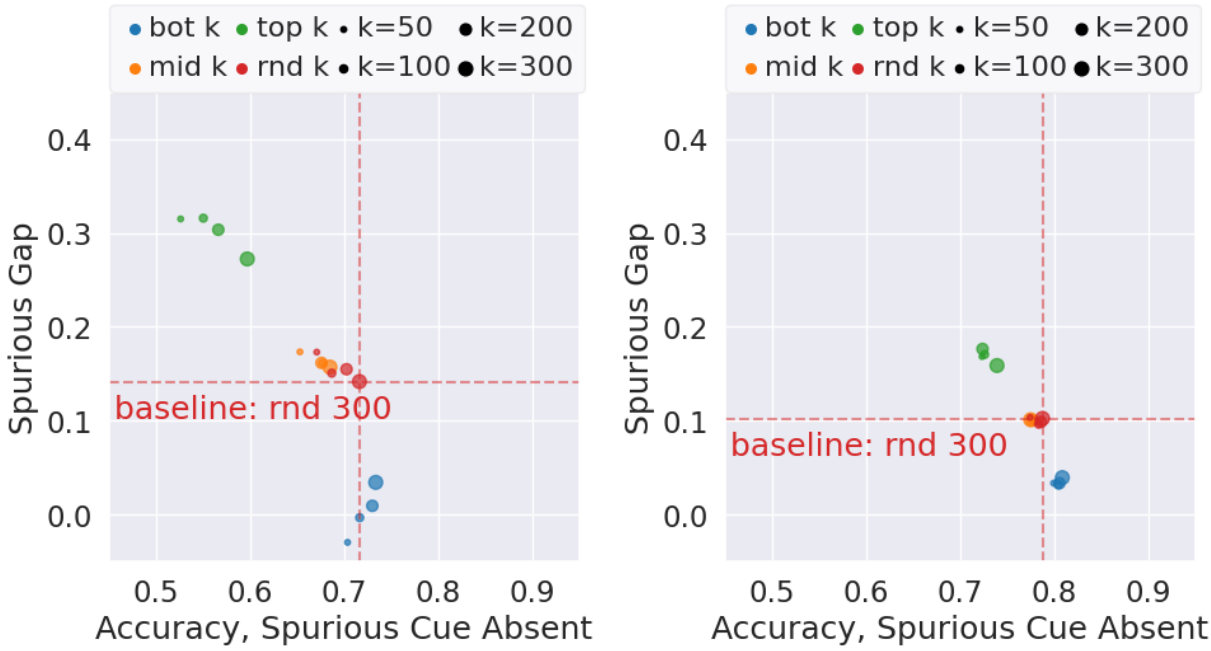
Closing Spurious Gaps and Improving Effective Robustness. In Figure 13.8, we present

both spurious gaps and accuracy on validation samples with few spurious cues (i.e. the 10 lowest ranked validation samples from each class) for classification heads learned from different subsets of data with varying sizes. Training linear heads specifically on the lowest-ranked samples effectively reduces spurious gaps with improved accuracy assuming few spurious cues.

In Figure 13.9, we present the evaluation of these classification heads on OOD benchmarks, including ImageNet-R, ImageNet-Sketch, and ObjectNet. Here we adapt the notion of effective robustness[380, 382], which, given a pool of base models, fits a line in the 2d-plane of OOD accuracy v.s. standard accuracy and measures the robustness of each model with its vertical distance to the fitted line. Intuitively, it measures how robust a model is compared to the linear trend of the given pool of base models (as in section 13.4). We include the linear fits respectively for ResNet-50 and DeiT-small in Figure 13.9a and the average effective robustness for classification heads trained on different types of subsets in Figure 13.9b. Re-training the linear classification head specifically on lowly ranked samples offers higher effective robustness than all other baselines.

13.6 Feature Reliability

Typically, the perceived danger of model reliance on spurious features is that performance drops when the spurious cues are no longer present; spurious gap exactly measures the performance drop. However, our analysis revealed a second danger of relying on spurious features, as they may pertain to multiple classes, leading to model confusion when the feature is strongly present. In fact, this vulnerability may also occur with core features that are shared among many classes. To characterize this issue, we propose another metric called *feature reliability*, which compares average activation on some feature for samples correctly or incorrectly classified to a specific



(a) ResNet-50

(b) DeiT-small

Figure 13.8: Spurious gaps and accuracy on validation samples with few spurious cues (i.e. the 10 lowest ranked validation samples from each class) for classification heads learned from different subsets of data with different sizes.

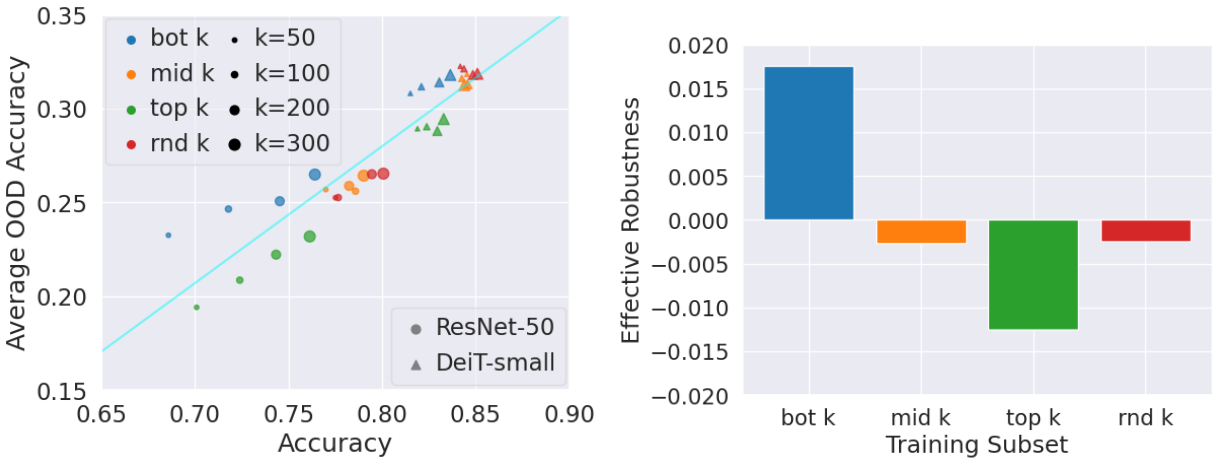


Figure 13.9: (**left**) Average OOD accuracy (over ImageNet-R, ImageNet-Sketch, and ObjectNet) vs ImageNet accuracy for classification heads learned from different subsets of data with varying sizes, along with linear fits over all classification heads; (**right**) Average effective robustness by classification head training subset.

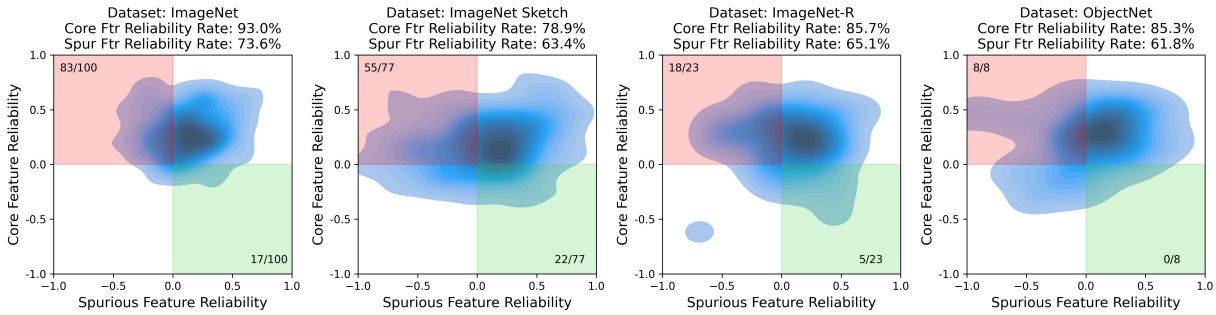


Figure 13.10: Kernel density estimations of the distribution of core and spurious feature reliability per class, over various datasets. We find that (i) core and spurious features are generally reliable, though core features are more reliable, (ii) both features become less reliable under distribution shift (iii) while rare, there exist classes where *only* spurious features are reliable, suggesting that removing spurious features in a data-agnostic fashion may deteriorate robust classification for certain classes.

class.

Formally, given a model, we define the reliability of a neural feature j to some class i as follows. Let $\mathcal{CC}(i)$ and $\mathcal{MC}(i)$ denote samples correctly and incorrectly predicted to class i respectively, so that $\mathcal{CC}(i) \cup \mathcal{MC}(i)$ is the set of all samples predicted to class i. W.l.o.g., let $\overline{\mathcal{CC}(i)_j}$ denote the average activation on feature j for samples in $\mathcal{CC}(i)$. We then define the reliability of feature j for class i as the relative difference between $\overline{\mathcal{CC}(i)_j}$ and $\overline{\mathcal{MC}(i)_j}$.

$$\textbf{FtrRel}(i,j) = \frac{\overline{\mathcal{CC}(i)_j} - \overline{\mathcal{MC}(i)_j}}{\max(\overline{\mathcal{CC}(i)_j}, \overline{\mathcal{MC}(i)_j})}$$

This metric lies between 1 and -1, with a negative score indicating that misclassified samples to class i activate feature j more than correctly classified samples. Thus, $\textbf{FtrRel}(i,j) < 0$ indicates feature j is unreliable for class i, as relying upon j to predict i leads to mistakes (i.e. a sample is wrongly predicted to a class because it activates a relied upon feature more than usual, increasing the class logit value). We analyze the reliability of neural features for a model adversarially trained on ImageNet. All linear head weights for this model on features we analyze

are positive, so higher activation on a feature does indeed translates to a greater contribution to the class logit. We focus on this model because we have identified its important neural features per class and annotated their function as core or spurious. Thus, we can use this model as a case study to compare the reliability of spurious features to core ones. We note that our metric is model-agnostic, and while this analysis is specific to a single model, we argue that the findings may extend to other models, as we observe diverse models to have strongly correlated feature sensitivities (see Supp.).

Namely, we focus on classes with both core and spurious features annotated. We compute feature reliability for all annotated features on ImageNet, as well as multiple OOD variants (ImageNet-R, -Sketch, and ObjectNet). We define core/spurious feature reliability for a class as the average reliability of all core/spurious features for that class. Figure 13.10 shows the joint distribution of core and spurious feature reliability across classes. First, we observe that both sets of features are generally reliable, though *core features are more reliable than spurious ones*. Also, under distribution shift, both feature sets become less reliable.

We now focus on classes where only one set of features is unreliable. Here, we can think of the unreliable set of features as responsible for errors: samples mistakenly predicted to i do so because they fool one set of features, activating them even higher than true instances of the class. In the majority of these cases, core features are reliable while spurious ones are not. However, surprisingly, there are classes where the opposite is true. Thus, if one were to remove spurious dependencies entirely, performance would degrade significantly on this subset of classes. Further, we see the fraction of classes with only reliable spurious features can be even higher on OOD datasets (28.6% for ImageNet-R vs. 17% for ImageNet).

Figure 13.11 shows examples of both cases: core features for *lawn mower* are unreliable

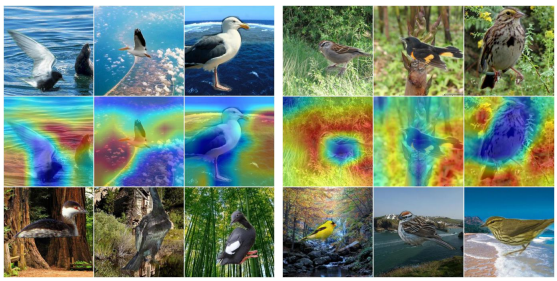


Figure 13.11: Samples with unreliable spurious but reliable core features for ObjectNet (**left**), and vice versa for ImageNet-R (**right**).

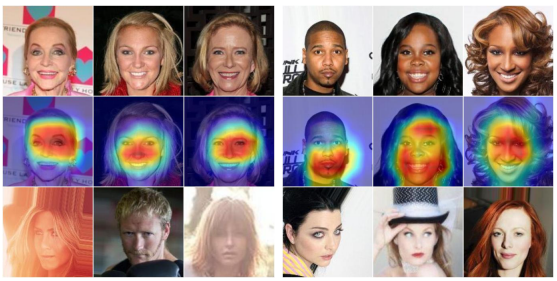
in ImageNet-R, where the lawn mowers are plastic toys instead of real metal ones, but the green grass spurious cue remains reliable, enabling correct prediction. Conversely, the keyboard feature is unreliable for the *computer mouse* class in ObjectNet, leading to a false positive prediction for a banana placed by a keyboard as a computer mouse. Recall *computer mouse* had a highly negative spurious gap, reflecting that its spurious features were confusing. Indeed, there is a direct connection between feature reliability and spurious gap in that they both highlight how spurious cues can be *clarifying* or *confusing*, though the former metric focuses on false positive errors, while the latter emphasizes true negative errors.

13.7 Beyond ImageNet: Lightweight Feature Discovery in New Domains

While our analysis focuses on ImageNet, we now show our methods extend beyond it, even with minimal additional training. Our framework for finding relevant data patterns relies on robust neural features, which have enhanced interpretability properties. However, adversarial training



(a) **Background bias** in Waterbird vs. Landbird Classification. The core features pertain to the bird, but above, the focus is on background.



(b) **Racial bias** in Celeb-A Blond vs. Brown Hair Color Classification. The core features pertain to the hair, but above, the focus is on skin.

Figure 13.12: Our framework can detect spurious features in *unseen data*. We detect designated spurious attributes like background correlation in the Waterbirds benchmark (left), as well as new spurious dependencies such as racial bias in Celeb-A Hair Classification (right). Presented above are highly activating images (top row), corresponding heatmaps (middle), and lowly activating images (bottom) for spurious features observed for the classes *waterbird, landbird, blond hair, brown hair* (left to right).

can be challenging, particularly when data is limited. We propose to simply fit a linear layer for the new task atop fixed features from an adversarially robust feature encoder pretrained on ImageNet; prior work shows transfer learning on robust features can be very effective [383]. We then perform the usual process of identifying important robust neural features per class and visualizing them via inspecting the highest-ranked and the lowest-ranked natural images, as well as heatmaps and/or feature attacks for highly ranked images.

We demonstrate the feature discovery on two spurious correlation benchmarks: Waterbirds and Celeb-A Hair Color classification. In Figure 13.12, we show some of the discovered patterns. For waterbirds, we find neural features that focus entirely on backgrounds; an important feature for predicting the waterbirds class is a water background. The lowest-ranked images within a class for these features often contain the opposite background (land instead of water), thus revealing the minority subgroup (waterbirds on land). For Celeb-A, we observe features that focus on faces, making them spurious for the hair classification task. Further, race appears to

be homogeneous amongst the top-ranked images, and opposite in the bottom-ranked images. This suggests that a model may spuriously associate brown skin with the brown hair class, likely leading to failures for blond-haired brown-skinned people.

Supplementary Material

13.A Impact and Limitations

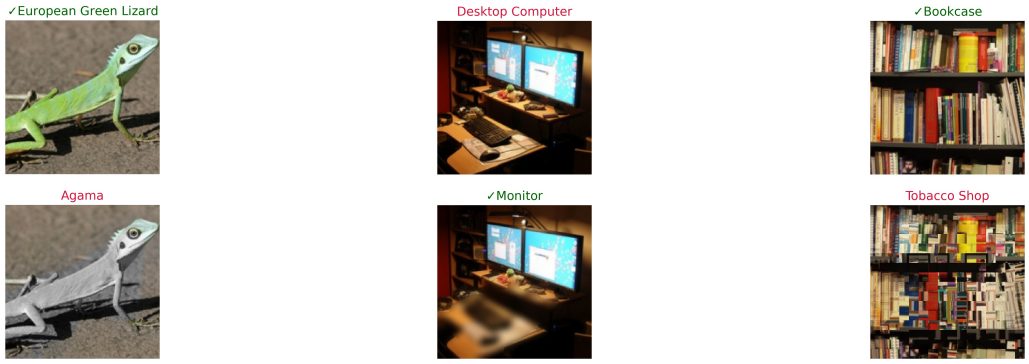
We hope our work will allow for better understanding of how spurious feature reliance affects deep models, leading to improved handling of algorithmic fairness and model robustness, which are both related to the spurious correlation problem. We open source all code, and modify a key framework of our method to vastly increase its usability for people with fewer computational or data resources. We also create a website for easier viewing of our analysis, making our entire annotation process completely transparent. We obtain verbal approval [1] from our institution's IRB for our study, and pay workers above minimum wage. Key limitations are that our method does not detect all spurious features (only a subset), and uses a single model to detect and measure them, though we argue that the features we detect are very relevant and likely relied upon by all models; we elaborate on this point in Section 13.F.2.

13.B Characterizing Feature Sensitivities

Using visualizations from our robust neural feature annotation, we obtain soft segmentations for features in input space that activate a specific neural feature. Namely, the Neural Activation Map for a neural feature can serve as a soft segmentation for the feature. Using a Mechanical Turk human study, we validate that neural activation maps for images in the top 20th percentile within its class for activation of the neural feature segment the same input pattern.

We then use targeted corruptions to gain insight on the information within an input region

[1] We were informed that formal approval is not needed because we do not obtain information *about* our crowd worker, and thus our study does not constitute human-subject research.



(a) Gray (Color) (b) Blur (Texture) (c) Patch Rotate (Shape)

Figure 13.13: Examples of features that are particularly sensitive to color, texture, and shape corruptions. Original images (**top**) and corrupted images (**bottom**) shown, along with model predictions. In one case, corrupting the spurious feature of *keyboard on desk* for class *monitor* actually improves prediction.

that models rely on most. The core premise of our argument is that degradation in model performance due to corruption of some information can be a proxy of the importance of the corrupted information. Indeed, similar arguments were made in [335, 337], where adding Gaussian noise to specific regions was employed to assess core/spurious or foreground/background sensitivity respectively. However, in our analysis, we use carefully chosen corruptions so to remove one aspect of primitive information (i.e. color, shape, or texture) without disrupting the others. Namely, our corruptions are graying out image regions, blurring out image regions, or patching an image region and randomly rotating each patch by a multiple of 90 degrees. Graying out an image removes color, blurring out destroys local (i.e. textural) details, and random patch rotation breaks longer edges, hence disrupting shape information.

We apply these targeted corruptions to segmented feature regions for all 5000 annotated class-feature pairs. Specifically, we focus on the 65 images per class where a feature is activated most highly (i.e. roughly top 5th percentile), since the feature is most prominent in these images. Also, for spurious features, we include a filtering step to avoid corrupting core regions. We do

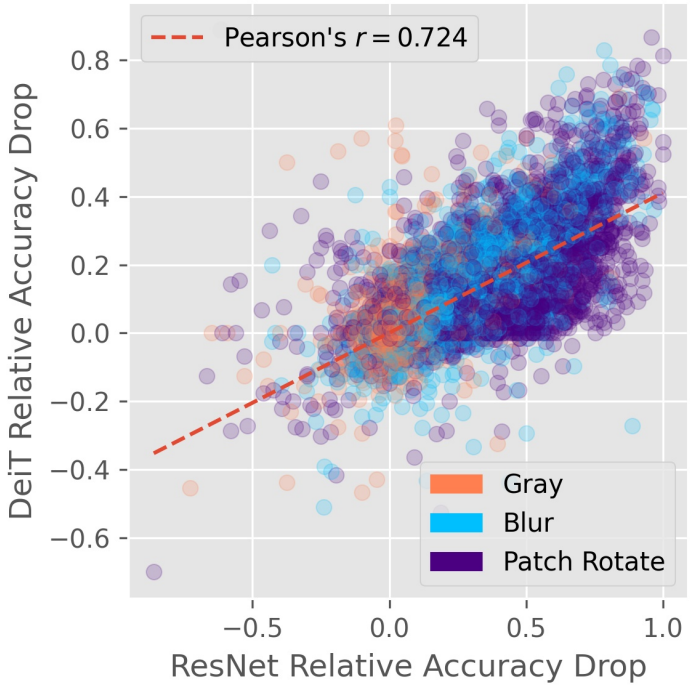


Figure 13.14: Sensitivities of each class-feature pair to specific corruptions for ResNet-50 and DeiT-Small. The two distinct models rely on features in the *same* way.

this by removing any overlap of the spurious feature segmentation with the consolidated core mask [335], which is a pixel-wise maximum of soft segmentations for any relevant core features. We track model accuracy on samples with and without each corruption applied, using change in accuracy as a measure of sensitivity.

Figure 13.13 shows examples of features that are the most sensitive (i.e. highest change in accuracy) to each of the three corruption types, visualizing the corruption and its affect on model prediction. In the left panel, we see that the color information in the core feature of *lizard body* is crucial for successful prediction of the class *European Green Lizard* (matching intuition). In the middle panel, we see a case where the spurious feature of *keyboard on desk* hurts classification. Namely, blurring out the keyboard and desk leads to more accurate prediction of the *desktop computer*. We note that while the shape and color information is retained, it appears that the

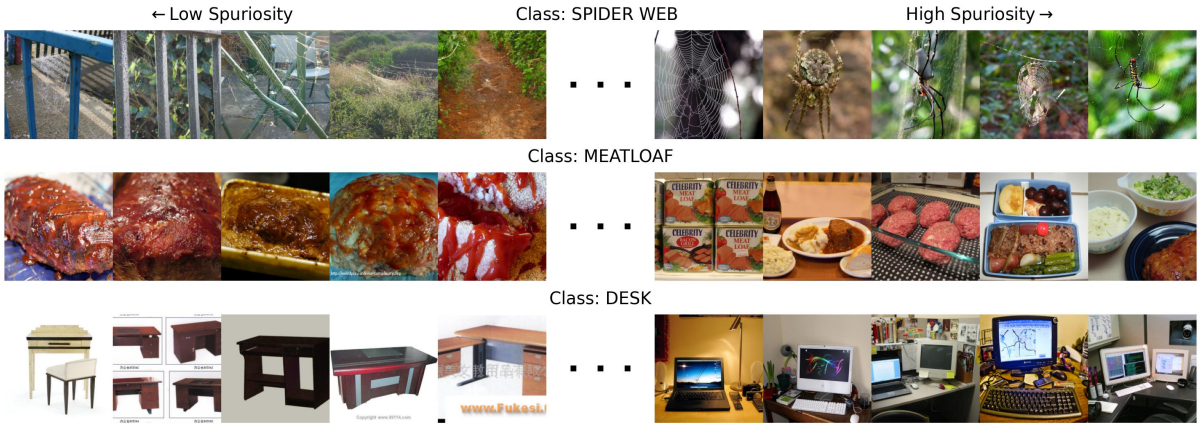


Figure 13.15: Images with highest and lowest spuriosity for classes with the lowest spurious gaps (aside from the two shown in Figure 13.5). The spurious dependencies for these classes are *confusing* because when the spurious features are prominent, models are less accurate. These features include *blurry background often with spider in foreground, plates or containers*, and *computer equipment* for classes *spider web, meatloaf*, and *desk* respectively.

local details in the texture of the keyboard and desk contribute more to confusing the model and leading to misclassification. Lastly, we revisit the *books* on a *bookcase* spurious feature, observed to be *clarifying* (i.e. models classify *bookcases* much more accurately when the spurious feature is present). Here, we observe that the shape of the books is crucial, as rotating patches leads to prediction of *tobacco shop*.

We conduct this analysis using two pretrained models of roughly equal size: ResNet50 and small DeiT. Figure 13.14 shows the relative accuracy drops incurred by performing each of the three corruptions for all 5000 annotated class-feature pairs on both models. We find that the two models have strongly correlated sensitivities ($r = 0.724$). The least correlated sensitivities appear to be for the patch-rotate corruption, where the transformer has nearly zero accuracy drop for many cases where the ResNet experiences greater accuracy drop. This is most likely due to the unique robustness of vision transformers to various patch transformations like permutation, rotation, and ablation [384, 385]. Despite this, the strong correlation between feature sensitivities



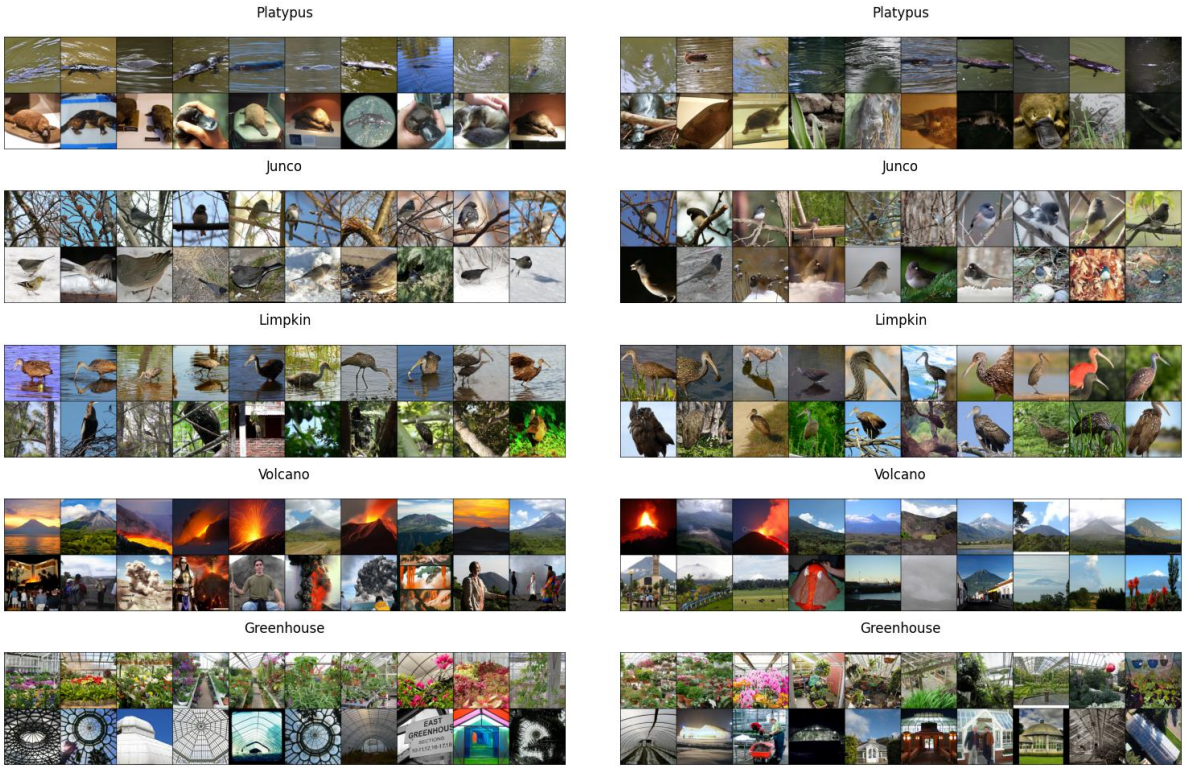
Figure 13.16: Images with highest and lowest spuriosity for classes with the highest spurious gaps (aside from the two shown in Figure 13.5). The spurious dependencies for these classes are *clarifying* because when the spurious features are prominent, models are more accurate. These features include *white collar with tie, sand*, and *aerial viewpoint, often with feet* for classes *suit, fiddler crab*, and *sandal* respectively.

for two diverse models suggests that model behavior is determined much more by the data it operates over than the specific decisions made during training. In other words, the features that models rely on and the ways in which they rely on them may be far more a function of data than a function of the model. If this conjecture holds, it would warrant greater inspection of the role of data in various open problems in modern deep learning (robust generalization, efficient learning, etc).

13.C More Examples of Spuriosity Rankings

To further validate our spuriosity rankings, we now present more examples of images ranked by spuriosity. While this validation is qualitative, we argue that it is sufficient since we are attempting to proxy a fundamentally qualitative notion.

Namely, we show the top ten and bottom ten images by spuriosity ranking. First, we show more examples of the most confusing spurious dependencies (i.e. negative spurious gap; figure



(a) Training Split

(b) Validation Split

Figure 13.17: Images with highest and lowest spuriosity for five randomly selected classes (top/bottom row in each chunk corresponds to images with highest/lowest spuriosity respectively). The spurious features depicted are *lake water, branches, water, sky with clouds or lava*, and *plants*. The spurious features are easier to see in the training split than in the validation split, because there are roughly $25\times$ more samples to choose from. Nonetheless, the validation images still seem to be organized so that the top-10 images contain the spurious feature much more than the bottom-10, indicating that our spuriosity ranks generalize to the validation split.

13.15) and the most clarifying spurious dependencies (i.e. positive spurious gap; figure 13.16). Then, we also present images organized by spuriosity ranking for five randomly selected classes. Here, we show images both ranked in the training and validation split to demonstrate that even though trends are clearer when ranking training images (because there are $\sim 25\times$ more of them to choose from), the underlying semantic concepts by which images are ranked still generalize to the validation split.

13.D Full Pretrained Model Evaluation Results

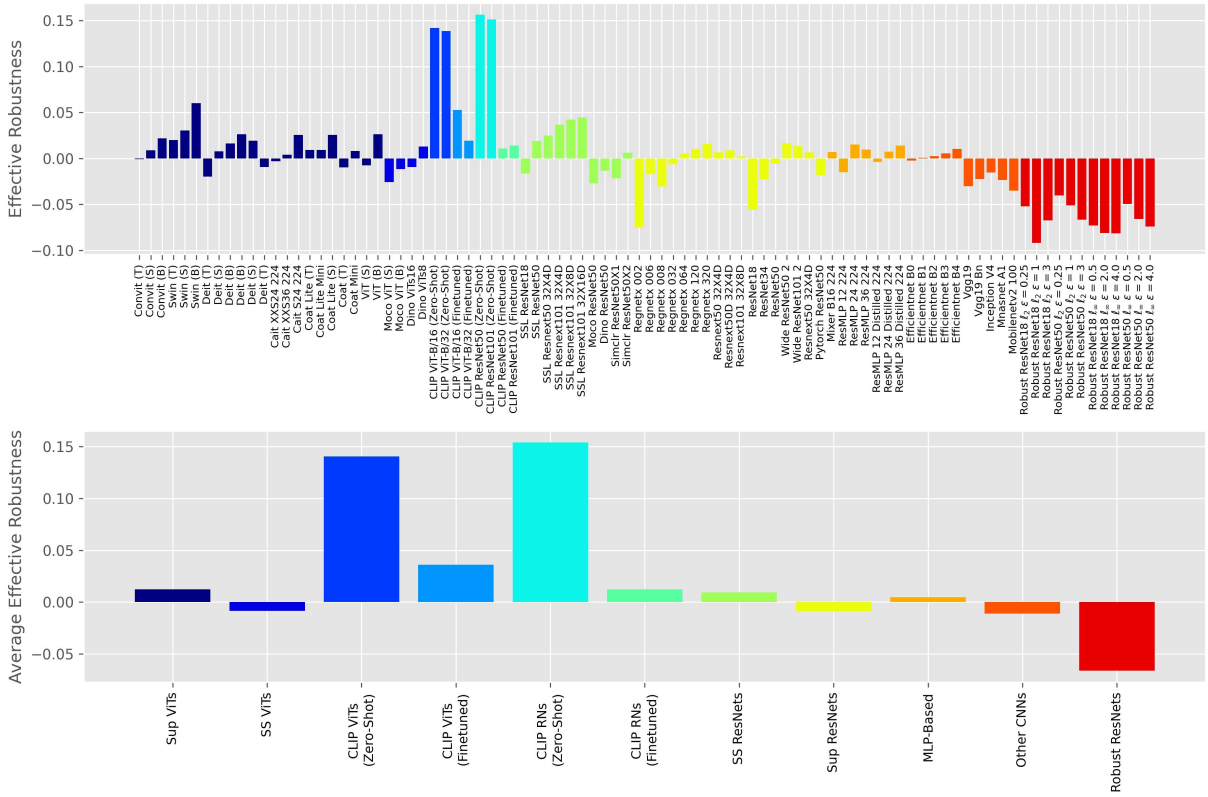


Figure 13.18: Effective robustness for all 89 models considered (**top**), along with effective robustness averaged over model category (**bottom**).

We now provide greater detail for the pretrained model evaluation presented in Section 13.4. We evaluate 89 models spanning diverse architectures and training procedures. Namely,

the models we consider fall into the following categories, with category nicknames in parenthesis: supervised vision transformers (Sup ViTs), self-supervised vision transformers (SS ViTS), CLIP vision transformers (CLIP ViTs, both Zero-Shot and Finetuned), CLIP ResNets (CLIP RNs), self- or semi-supervised ResNets (SS ResNets), supervised ResNets (Sup ResNets), mixer and ResMLP models (MLP-Based), various other convolutional networks (Other CNNs), and adversarially trained ResNets (Robust ResNets). The vast of majority of pretrained weights are obtained from the timm library [386], while others come directly from their original respective repositories [366, 383, 387, 388]. Figure 13.18 shows effective robustness for each model, as well as the average effective robustness per category.

We measure effective robustness by first fitting a line to predict accuracy on the 10 validation images per class with the *least* spuriosity as a function of accuracy on the 10 validation images per class with the *most* spuriosity. Let acc_{top} and acc_{bot} denote accuracy on images with the most and least spuriosity respectively (i.e. top and bottom with respect to spuriosity rankings). We refer to the line of best fit as $\beta : \mathbb{R} \to \mathbb{R}$. Effective robustness for a model with accuracies on top and bottom spuriosity ranked images $\text{acc}_{\text{top}}, \text{acc}_{\text{bot}}$ respectively is simply $\text{acc}_{\text{bot}} - \beta(\text{acc}_{\text{top}})$. In other words, we measure how high above the line of best fit a model's marker falls in figure 13.4.

We note that this differs slightly from the originally presented notion of effective robustness in [380], where a log-linear fit is performed. Moreover, instead of predicting out-of-distribution accuracy based on in-distribution accuracy, we compare accuracy on two carefully chosen subsets of in-distribution data. We argue that we capture subsets where subpopulation shift has occurred, specifically with respect to common spurious cues for each class.

Our findings show that some models are more effectively robust than others. Namely,

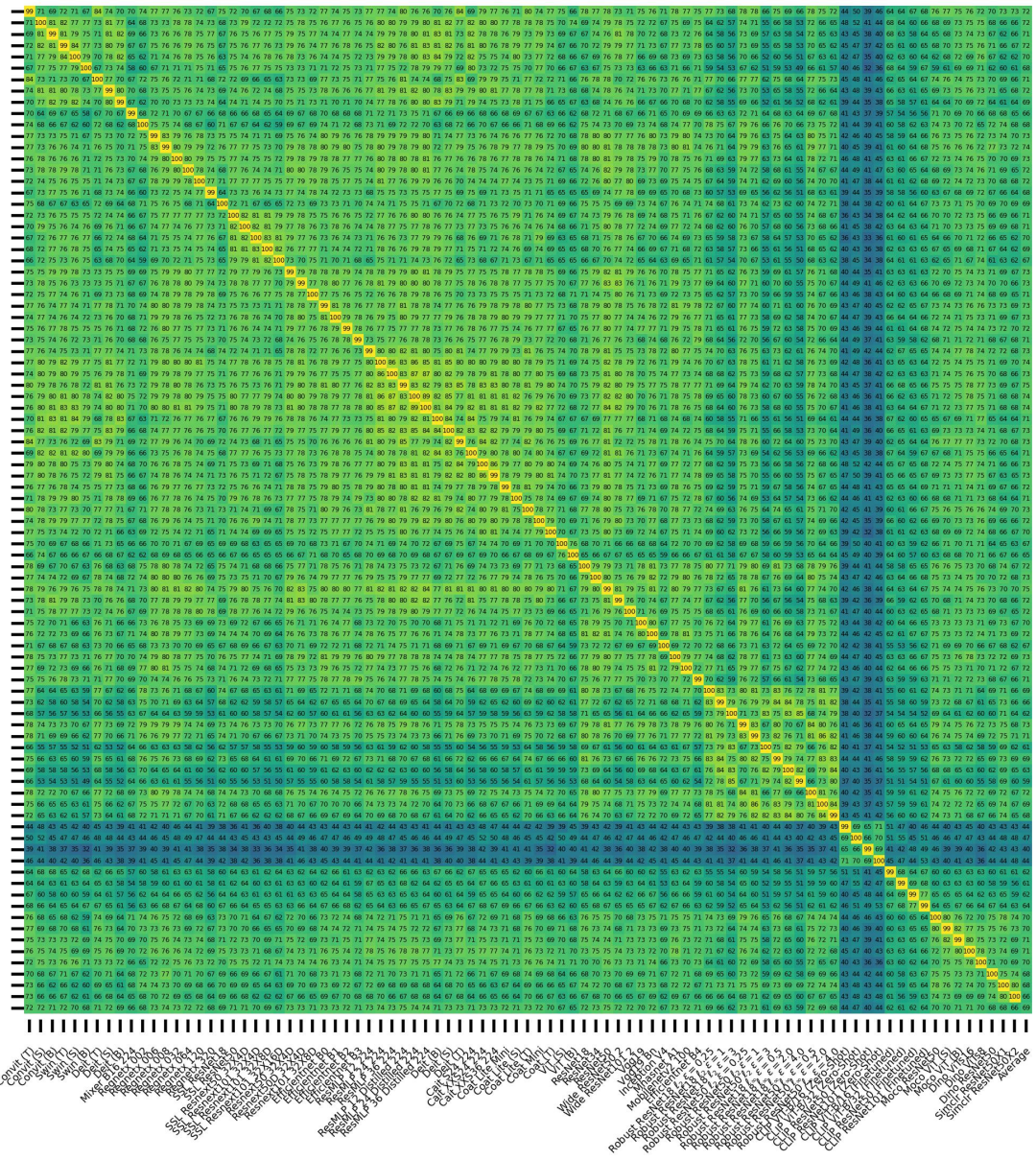


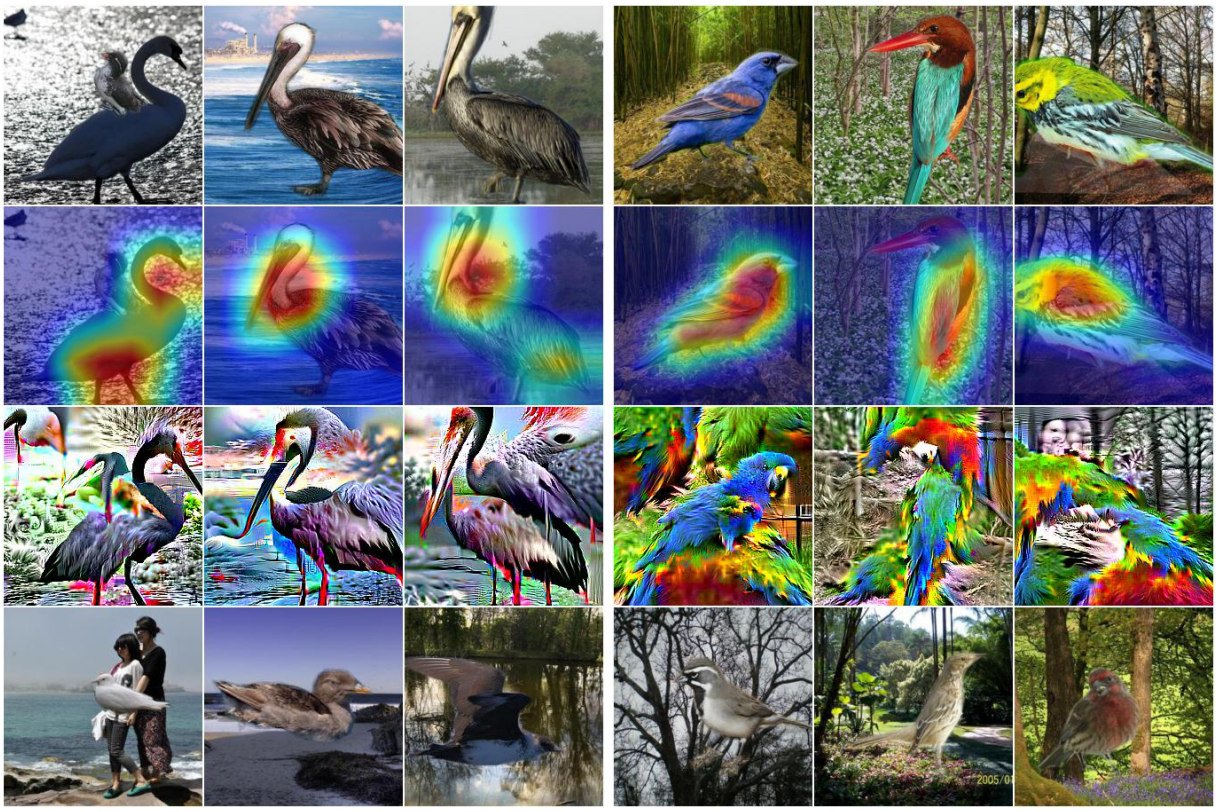
Figure 13.19: Correlation of per-class spurious gaps for all pairs of models. Numbers shown are Pearson's r correlation coefficients times 100. On average, a pair of diverse models have strongly correlated class-wise spurious gaps, with $r = 0.69$. Notably, zero-shot CLIP models have the worst correlation with other models, suggesting that their perception is fundamentally different.

zero-shot CLIP models have much higher accuracy on images with lowest spuriosity than their performance on the images with highest spuriosity would predict. After finetuning a linear head using ImageNet on top of the fixed CLIP image encoder, we see effective robustness drops significantly. We note that both these results were observed using independent distributional robustness measures in [389]. At the other extreme, adversarially trained ResNets have the lowest effective robustness, a result carefully studied in [367].

Aside from these two extremes, most trends are muted. We believe the stronger signal exists in per-class spurious gaps, as shown in figure 13.6, suggesting that while models generally behave similarly on the same inputs, their behavior (i.e. with respect to spurious feature dependence) varies dramatically across inputs. Indeed, we find that per-class spurious gaps correlate strongly between pairs of models in our evaluation suite. Figure 13.19 visualizes these correlations, with an average correlation of $r = 0.69$ being observed. We thus recommend closer attention to be paid to the individual data classes and potential spurious features *per class* a model is to be deployed over.

13.E Additional Discovered Features for Waterbirds and Celeb-A

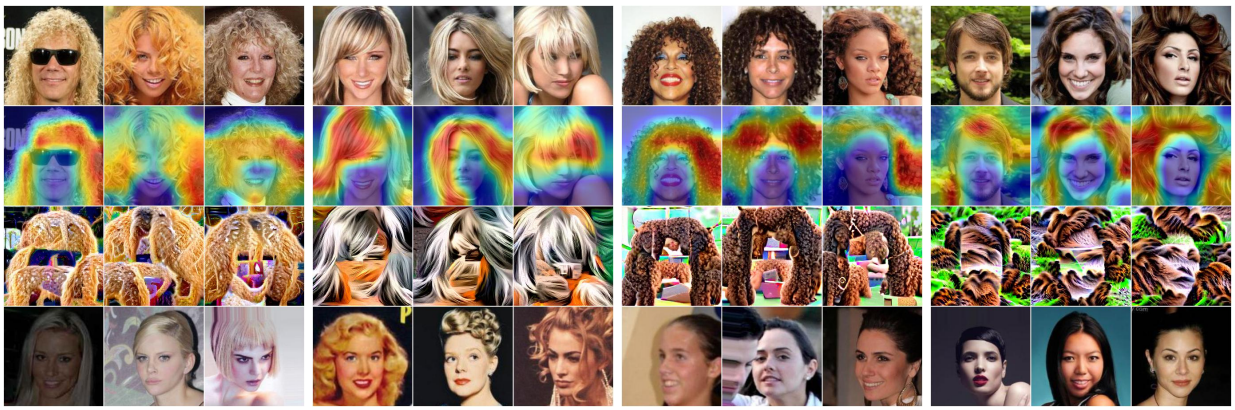
We now present additional features identified via our lightweight extension of the feature discovery framework of [335]. Specifically, we show core features, which are relied upon more frequently than spurious features, which we show in Figure 13.12. We observe robust neural features to respond to specific patterns shared among subpopulations within the broader classes. Surprisingly, these robust neural features were not at all trained on images from the downstream tasks. Yet, the patterns they respond to are semantically meaningful for the downstream task. We



(a) Waterbird core feature. Focus is on elongated and curved necks.

(b) Landbird core feature. Focus is on feathers with bright colors.

Figure 13.20: Core features discovered for the Waterbirds waterbird vs. landbird classification task. We identify features that occur in specific subpopulations of the broader classes. Low activating images reveal subpopulations where the feature is absent. The rows show (from top to bottom): highly activating images, corresponding heatmaps and feature attacks, and lowly activating images.



(a) Focus is on blond curly hair.

(b) Focus is on blond straight hair.

(c) Focus is on brown curly hair.

(d) Focus is on wavy brown hair.

Figure 13.21: Core features discovered for the Celeb-A blond vs. brown hair classification task. We identify features corresponding to different types of hair within the same broader class of hair color. The rows show (from top to bottom): highly activating images, corresponding heatmaps and feature attacks, and lowly activating images.

attribute this to the diversity in ImageNet, which leads to learning many general visual patterns that are manifested in more specific contexts for a vast number of downstream tasks.

13.F Robust Neural Feature Annotation Details

Our work uses the feature discovery framework introduced in [335], also known as Salient ImageNet. We now i. review the framework, ii. discuss its merits and potential limitations, and iii. detail our expansion/modifications to it.

13.F.1 Salient ImageNet Framework

The steps of the Salient ImageNet Framework are:

1. Adversarially train a model for your classification task.

2. Identify important neural features per-class based on feature importance, which is simply

the product of the average feature activation and the weight of linear head connecting feature to class logit. Feature importance explicitly computes the average contribution of a feature to a class logit for samples in the class.

3. Select the top-k (i.e. $k = 5$) activating instances of a class for a selected feature and additionally generate heatmaps (via overlaying the Neural Activation Map on the original image) and feature attacks (via adversarially attacking the original image using gradients from the adversarially robust model so to maximize activation on the feature).

4. Have humans inspect the $3 \times 5 = 15$ visualizations to determine if the focus of the neural feature is on the main object (i.e. pertaining to the class and hence *core*) or on the background or a separate object (hence making the feature *spurious*).

In [335], the annotated robust neural features were primarily used to softly segment image regions containing core and spurious features at scale. In addition to the Mechanical Turk study performed to annotate neural features, a second study was conducted to confirm that neural activation maps for annotated features softly segment the same input features across the top 5th percentile of samples within the relevant class. With this confirmation, they computed neural activation maps for all of these images and used these as feature segmentation maps. To assess reliance on core and spurious features, they would inspect drop in accuracy due to adding small amounts of Gaussian noise to these regions.

13.F.2 On Feature Discovery Using Only One Model

The key idea of the Salient ImageNet framework was to use neural features of an adversarially trained network as automatic 'concept' detectors, and further to use their activation maps as soft

segmentations. One may argue then that the features discovered are biased to the underlying model used. However, we contend that the primary patterns that models rely on are generally the same. Note that the existence of patterns predictive of a class is determined by the data distribution and not the model. While it is true that models may learn to rely on different patterns and may also vary the degree to which they rely on the same patterns, we contest that the most predictive patterns will be detected and relied upon by nearly all models. In our own experiments, we find that the class-wise spurious gaps are highly correlated for any two pairs of models, with average correlation of $r^2 = 0.69$ (see figure 13.19). We also observe that the accuracy drops experienced due to targeted corruption of input regions where certain features reside for a ResNet and a vision transformer are highly correlated too ($r^2 = 0.724$) (see figure 13.14). These results suggest that models are sensitive to the same data patterns and even rely on the different pieces of information (i.e. color, shape, texture) captured in the data patterns in the same ways.

To be perfectly clear, we do not claim to have captured *all* spurious (or core) dependencies, mainly because we only inspect the 5 most important robust neural features per class. We acknowledge that another reason for certain dependencies to be overlooked is because we use a single model to detect and measure the presence of concepts. However, this only occurs for patterns that other models are sensitive to but the adversarially trained model is not. The main patterns that fit this description are adversarial ones, which contain no semantic meaning and thus arguably *should* be overlooked for our purposes. In other work, adversarialy trained models were found to have increased reliance on spurious features [367]; this result was also corroborated in our spurious gap analysis. Thus, if anything, it seems like using an adversarially trained model may overlook core features, favoring spurious ones. Seeing as the goal of our analysis is to detect and assess spurious feature dependencies, we believe using robust neural features from an adver-

sarially trained model is warranted, especially given the enhanced interpretability of this model, which allows for reliable heatmaps and feature attacks.

13.F.3 Our Contributions

We now detail the novel contributions we make to the Salient ImageNet framework and analysis from [335]. The most crucial contribution is the notion that to interpret and improve spurious correlation robustness, one can seek to simply reorganize the data they already have, as opposed to collecting more samples or additional instance-wise supervision (i.e. segmentations, manual annotations beyond class-label) that would require fundamental changes the training strategy. Using spuriosity rankings, we gain new and surprising insight into the nuanced ways that models rely on spurious features. Namely, we find that contrary to common belief, models can perform worse when spurious features are present than when they are absent (i.e. because these spurious features are depended on for multiple classes and thus are *confusing*). Our analysis on training new heads on curated subsets of data (based on spuriosity rankings) shed insight on how valuable samples are with respect to accuracy across distribution shifts, highlighting that spurious feature reliance does not to add to (and can even hurt) OOD accuracy, while also causing disparate performance within the training distribution, raising fairness concerns.

Our second analysis based on our new *feature reliability* metric also yields curious and complementary findings. We note that this metric can extend to any model; we choose to use it on the adversarially trained model we annotated because we can then compare spurious and core neural class-feature dependencies. Here, we can quantitatively answer questions like 'are core features more reliable than spurious ones?' (yes), 'are they always more reliable?' (no,

surprisingly), and 'how does reliability change over distribution shift?' (varies by class).

Finally, we make a crucial adjustment to the feature discovery framework of [335] that removes its most costly and at times prohibitive step (adversarial training). Observing that the framework can be applied by simply fitting a linear layer atop a frozen adversarially robust feature encoder vastly increases the usability of this framework, cutting the time and data required to perform feature discovery by huge margins.

Additionally, we believe our rehaul of evaluating spurious feature reliance corrects crucial shortcomings of the evaluation framework of [335]. First, the previous framework required taking images out of distribution by corrupting core and spurious regions. The choice and strength of corruption introduced hyperparameters that we observed the final metric to be undesirably sensitive to. Further, because diverse models have varying general robustness to the corruption applied, the previous evaluation procedure precluded comparisons across models. In contrast, our method focuses on finding subsets of natural images with vastly different degrees of spurious cue presence. Since we only look at accuracy over in-distribution images, comparison across models is completely appropriate. Moreover, we account for the strong correlation between accuracy and distributional robustness by using the notion of effective robustness adapted from [380] to control for the varying accuracies of the models we consider. Further, our method evaluates models on the *same* sets of data, while the previous method did not, instead using images where a given feature was more present. Finally, we observe that segmentations for spurious features (used to measure sensitivity to those features via inspecting drop in accuracy due to noising the region) would often include core features, thus conflating the sensitivity of core and spurious features in their metric. Nonetheless, the work of [335] is absolutely instrumental in enabling our analysis, though our methods and findings are markedly different and more extensive.

Other contributions that are less novel but still impactful include massively expanding the scale of the feature annotation analysis and creating a web interface to view the discovered dependencies. Specifically, we annotate over $4\times$ more classes, yielding interpretation of feature dependencies for all 1000 classes in ImageNet. Given that this benchmark is ubiquitous, we believe a deep dissection of what one model perceives as important patterns in ImageNet can be extremely informative in better understanding how machine perception differs from human perception. We also perform the MTurk study to demonstrate the generalizability of our feature annotations for the top 20th percentile of images based on feature activation as opposed to the top 5th percentile, leading to reliable feature soft segmentations for many more images. Lastly, we create a website[2], which, while simply a user-interface, will greatly increase ease of viewing and transparency into our analysis, as we show all the visualizations used to perform our core/spurious neural feature annotations. We present screencaps of pages from the website in Figure 13.22 and Figure 13.23. As a web-interface, our work is now accessible to people who do not code or lack the resources to generate our visualizations on their own devices.

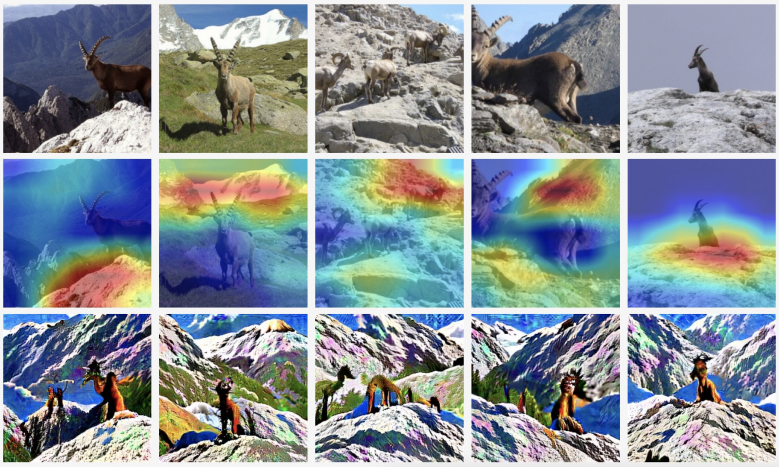
[2]We do not include the link to the website to maintain anonymity. Nonetheless, the website is already complete.

Class: 350 (Alpine Ibex), Feature: 389

This feature is *spurious* for class *Alpine Ibex*. It is the third most important feature for class *Alpine Ibex*.

Below, we display the five Alpine Ibex images that most highly activate the feature, along with the Neural Activation Map (heatmap) and Feature Attack for each image. These are the same as the images shown to crowdworkers to annotate this class-feature pair.

We additionally provide the full responses of the five crowdworkers that annotated this pair, as well as links to other related pairs (i.e. same class or same feature).



MTURK WORKER ANNOTATIONS

We now show the responses from five crowdworkers used to annotate this class-feature pair as *core* or *spurious*. Workers first denote if the focus of the image is on the main object (*core*), or a separate or background object (*spurious*). They also mark their confidence on a scale of 1 to 5, and provide written explanations for their annotation.

Main Focus	Confidence	Explanation
Main Object	4	Focus is on some part of the main object ibex, capra ibex leg and head are focus
Background	3	The main object is capra ibex but the focus is on background.
Background	5	Focus is on mountain in the background of capra ibex
Main Object	3	Red focus is on ibex.
Separate Object	5	Focus is on mountain which is not part of ibex

Votes for core: 2, Votes for spurious: 3. Feature is thus annotated as *spurious* for the class.

RELATED CLASS-FEATURE PAIRS

IMPORTANT FEATURES FOR *ALPINE IBEX*

There are 3 features annotated as *core* and 2 features annotated as *spurious* for *Alpine Ibex*. Core features include 462, 1207, 1526. Spurious features include 389, 820.

CLASSES WHERE FEATURE 389 IS AMONG MOST IMPORTANT (I.E. TOP-5)

Feature 389 is *core* for the following classes: Mountain, Valley, Volcano.
Feature 389 is *spurious* for the following classes: Marmot, Alpine Ibex, Ski.

Figure 13.22: Example of a landing page for one class-feature dependency. We present all visualizations shown to MTurk workers to annotate the feature as cor or spurious, as well as the annotations and explanations given by the MTurk workers. Further, we provide links to pages for other features relevant to the class, and other classes that also rely on the feature.

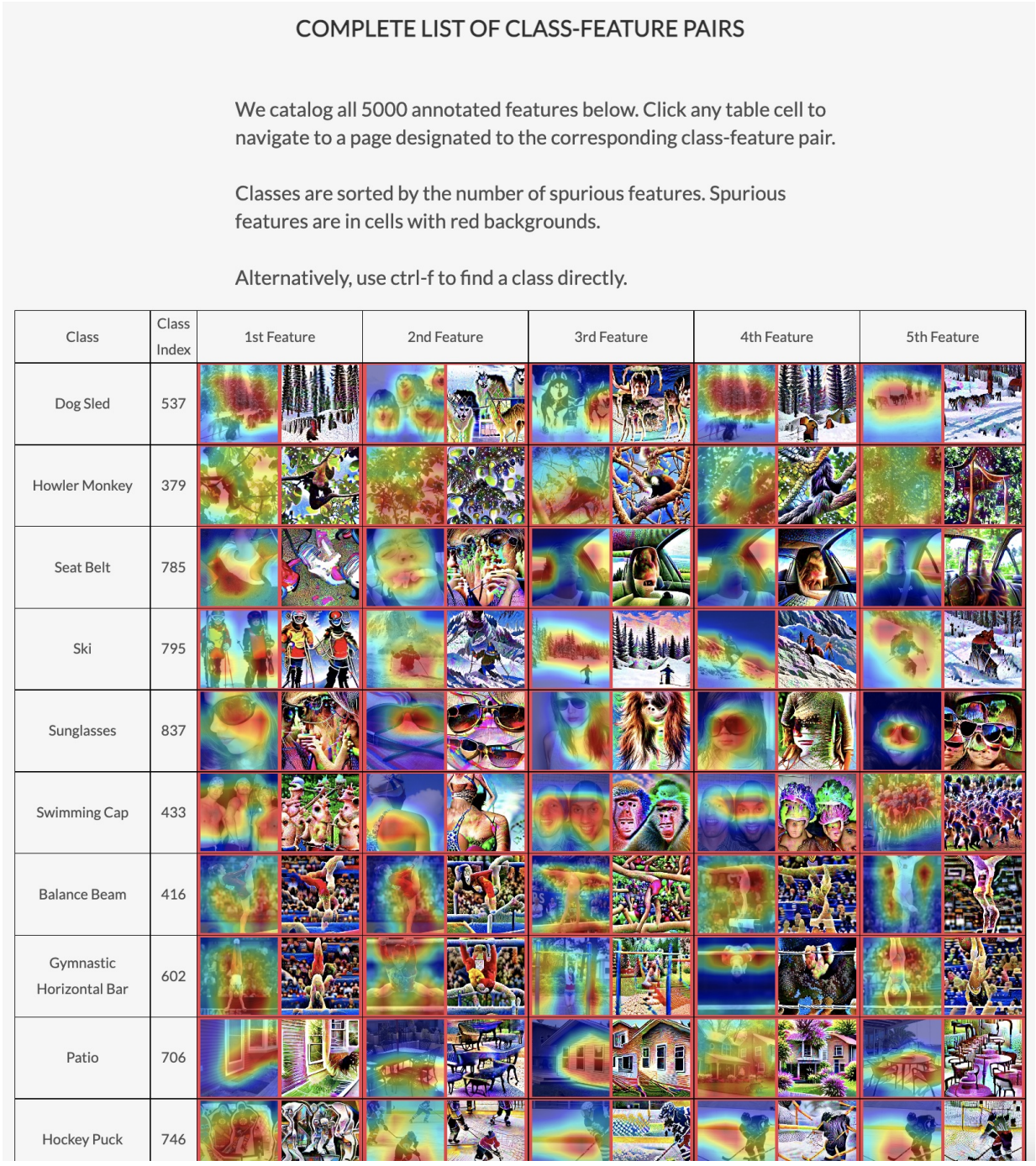


Figure 13.23: Screencap of a main table from our web-interface for accessing all 5000 class-feature dependencies. One can simply search for a class by its name or ImageNet class index to see the five features deemed most important for (i.e. contributing the most to the logit of) the class. Each image then links to an individual landing page designated for the class-feature pair (see Figure 13.22 for an example).

Chapter 14: Data-Centric Debugging: mitigating model failures via targeted data collection

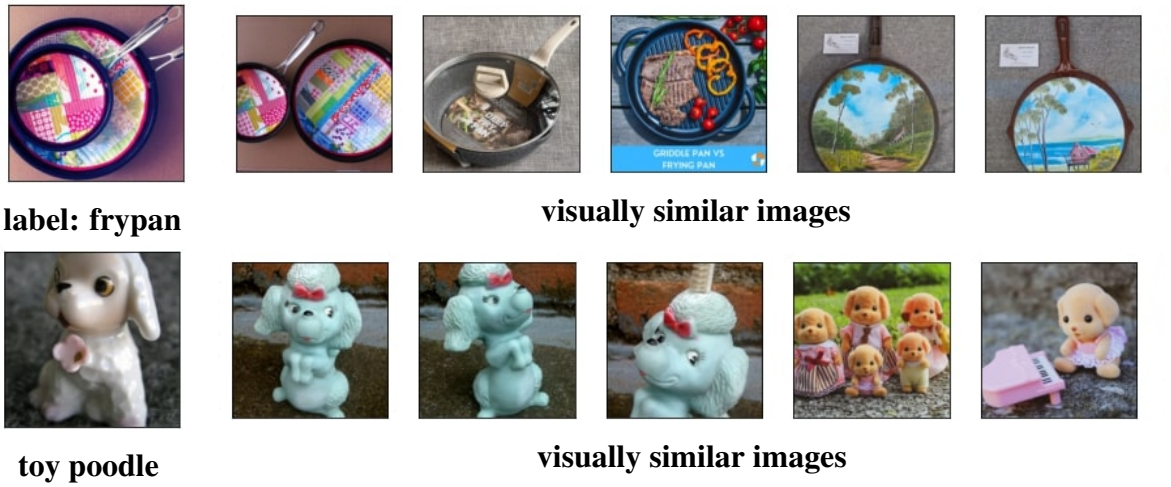


Figure 14.1: We introduce a method for debugging model failures by discovering "visually similar" images from the web. Left: sample images on which 20 ImageNet trained models with near state-of-the-art accuracy make incorrect predictions (label below each image). Our proposed debugging methodology can fix such failure modes. Right: "visually similar" images from the pool images $\mathcal{F}$.

14.1 Introduction

As machine learning systems are increasingly being deployed in the real world, understanding and mitigating their failure modes becomes critical to ensure that models work reliably in different deployment settings. For example, in medical applications, it is common to train a model using data from a few hospitals, and then deploy it more broadly to hospitals outside the training

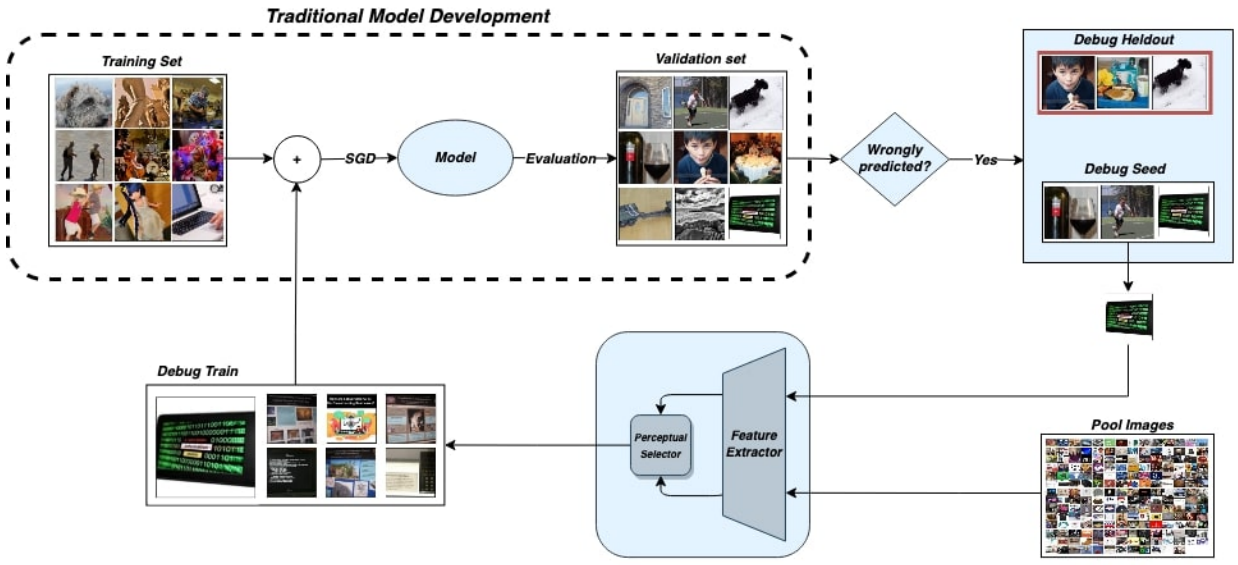


Figure 14.2: Our framework (Data-Centric Debugging) for model debugging via targeted image retrieval. We want to improve model performance on an error distribution $\mathcal{E}$ (while maintaining accuracy on the test sets) using a sample $\mathcal{E}_{sample}$ (from $\mathcal{E}$) and a weakly labeled dataset $\mathcal{F}$. We divide the set $\mathcal{E}_{sample}$ into two disjoint sets: $\mathcal{E}_{seed}$ and $\mathcal{E}_{heldout}$. Using the samples in $\mathcal{E}_{seed}$, we want to add several "visually similar" images (selected from $\mathcal{F}$) to the training set, such that the model performance improves on $\mathcal{E}_{heldout}$.

set[17]. In such cases, we may want to identify the hospital systems on which the model fails and feed more training data from those systems into the model to improve its performance.

Most of the prior works in the literature focus on mitigating a small set of failure modes identified by a human-in-the-loop [390, 391]. This can involve collecting new datasets with objects in uncommon settings [59, 348, 352, 355, 357] (e.g. frog in snow, ship indoors) which can be time consuming and expensive. Moreover, in several cases, humans might not be adequately aware of the undesirable failure modes and even if they are, collecting large number of images in the desired deployment scenarios might not always be feasible.

In this chapter, we focus on the image classification problem $\mathcal{X} \rightarrow \mathcal{Y}$ where the goal is to predict the ground truth label $y \in \mathcal{Y}$ for input $\mathbf{x} \in \mathcal{X}$. In the traditional model development framework, we have a training set, a validation set and a test set. The training set is used to

train the model, the validation set is used to evaluate and improve the model performance during development and the test set is used to report a final metric for the model performance.

In this setting, we consider an error distribution $\mathcal{E}$ representing a deployment scenario where a trained model fails i.e. the model makes incorrect predictions on every sample from $\mathcal{E}$. We have access to a small set of images $\mathcal{E}_{sample}$ from $\mathcal{E}$ and it is prohibitively expensive to obtain more samples from $\mathcal{E}$. Our goal is to improve model performance on $\mathcal{E}$ while maintaining performance on the existing test set(s). One naive approach could be to add $\mathcal{E}_{sample}$ to the training set. However, the model might overfit to $\mathcal{E}_{sample}$ and fail to generalize to novel samples from $\mathcal{E}$. Thus, the traditional development framework can be ineffective when sampling a large amount of data from $\mathcal{E}$ is infeasible.

In this chapter, we propose a new formulation where in addition to the training data and $\mathcal{E}_{sample}$, we have access to a large weakly-labeled (i.e., very noisily labeled) pool of images denoted by $\mathcal{F}$. Here, $\mathcal{F}$ could be obtained from Flickr, Commoncrawl [392] or any suitable data source. We collect $\mathcal{F}$ using Flickr search (Section 14.4.2) and carry out a filtration step to ensure that the images in $\mathcal{F}$ are "significantly different" from the test set (Section 14.4.3). Because of the noisy labeling, we find that naively adding the complete set $\mathcal{F}$ to the training set can hurt model performance. For example, we observe -13.67% drop in accuracy (Table 14.4). Thus, we want to select a few samples from $\mathcal{F}$ *without human supervision* to improve model performance on $\mathcal{E}$.

Intuitively, by selecting several images from $\mathcal{F}$ that are "visually similar" to the images in $\mathcal{E}_{sample}$, we would expect a broader coverage of $\mathcal{E}$ resulting in improved model performance compared to say, only adding $\mathcal{E}_{sample}$. However, because $\mathcal{F}$ can be large, identifying such similar images can be difficult. Previous works on similarity matching such as LPIPS [41] compute very

high dimensional image embeddings and use the l_2 distance between them as the "perceptual similarity distance". However, their high dimensionality makes them infeasible for large-scale visual search problems (discussed in Section 14.4.1).

Moreover, even if we discover the similar images, we may achieve improved results on $\mathcal{E}_{sample}$ due to some patterns specific to the $\mathcal{E}_{sample}$ images. For example, an image in $\mathcal{E}_{sample}$ contains some pattern that is similar to the patterns of some different class and similarity matching may discover images from the other class. As a result, the model may achieve improvements on $\mathcal{E}_{sample}$ and still fail to generalize to new samples from $\mathcal{E}$. Thus, to ensure that a model revision improves performance over the distribution $\mathcal{E}$ as opposed to simply the observed instances $\mathcal{E}_{sample}$, a careful framework for model debugging is required.

To address these challenges, we introduce **Data-Centric Debugging (DCD)**, illustrated in Figure 14.2: a framework for targeted data collection to mitigate failure modes of deep models and faithfully assess model performance on the error distribution. To retrieve visually similar images, we use the ℓ_2 distance in the feature space (penultimate layer activations) of a deep network. Compared to LPIPS [41] (state of the art for measuring perceptually similarity), our embeddings have several orders of magnitude lower dimensionality and require vastly less memory and processing time to retrieve similar images. We find that our method reduces compute and storage requirements by **99.58%** (Table 14.2).

To faithfully evaluate model performance, we divide the set $\mathcal{E}_{sample}$ into two disjoint sets namely, $\mathcal{E}_{seed}$ and $\mathcal{E}_{heldout}$. We refer to them as debug-seed set and debug-heldout sets respectively. We want to use the set $\mathcal{E}_{seed}$ for discovering visually similar images such that the model performance improves on $\mathcal{E}_{heldout}$. That is, we only use $\mathcal{E}_{seed}$ (not $\mathcal{E}_{heldout}$) for visual similarity matching and evaluate model performance on $\mathcal{E}_{heldout}$. Because $\mathcal{E}_{heldout}$ is disjoint from $\mathcal{E}_{seed}$

but from the same distribution $\mathcal{E}$, an improved performance would suggest that the model is not overfitting to the images in $\mathcal{E}_{seed}$ and can generalize to novel samples from $\mathcal{E}$. Thus, model performance on $\mathcal{E}_{heldout}$ is a more faithful evaluation metric for $\mathcal{E}$.

We apply our proposed framework on the ImageNet [56] classification task. We first select 160 ImageNet classes on which 20 highly accurate ImageNet trained models achieve low accuracy (details in Appendix 14.A.1). From these classes, we select the incorrectly classified samples from the ImageNet-V2 dataset as the $\mathcal{E}_{sample}$ set. Next, we divide $\mathcal{E}_{sample}$ into the $\mathcal{E}_{seed}$ and $\mathcal{E}_{heldout}$ sets (Section 14.4.4). For an image $\mathbf{x} \in \mathcal{E}_{seed}$ with label i, we can either select visually similar images from the subset of $\mathcal{F}$ with weak label i (denoted by $\mathcal{F}(i)$) or from the complete set $\mathcal{F}$ thereby discarding the weak labels. In the latter case, we can assign the label i to selected images. We find that selecting from the complete set often leads to images that are similar to $\mathbf{x}$, but from a different class, thereby contaminating the dataset with wrongly labeled images. We illustrate this in Figure 14.5. Thus, we select similar images from the subset $\mathcal{F}(i)$.

In Section 14.5, we experiment with several different models for extracting image embeddings for visual similarity matching namely, Standard Resnet-50, Robust Resnet-50, DINO ViT-S/8 and DINO ViT-S/16 [387]. Our experiments (Table 14.4) suggest that DINO models are significantly better at discovering similar images compared to Resnets.

In Table 14.1, we compare our method against the "Original" model trained using standard ImageNet, "Random": trained using randomly selected images from subsets $\mathcal{F}(i)$, "Complete": trained on the full $\mathcal{F}$ dataset (with class re-weighting so that weights assigned to classes are same as in ImageNet). For our results ("DINO"), we used DINO ViT S/8 for similarity matching. We observe that our method achieves the best results on Heldout set: (29.62%), significantly outperforming both complete (23.07%) and random (20.17%). We also achieve the best results

Debug-Train	Accuracy on different sets							
	incorrectly class.		subset of 160 classes			all 1000 classes		
	Seed	Heldout	MFreq	Compl.	INet	MFreq	Compl.	INet
Original	0%	0%	35.56%	56.89%	62.89%	63.70%	76.12%	76.47%
Complete	18.53%	23.09%	37.56%	53.84%	49.22%	61.70%	72.93%	75.07%
Random	16.78%	20.17%	41.56%	58.81%	63.91%	63.77%	75.21%	76.37%
DINO	36.28%	**29.62%**	54.06%	**63.85%**	**64.62%**	65.28%	**76.42%**	76.54%

Table 14.1: Results using DINO ViT-S/8 in our framework. "INet" denotes the ImageNet test set, "MFreq" denotes the ImageNet-V2[393] MatchedFrequency set, "Compl." (Complement) denotes the set of all ImageNet-V2 images excluding "MFreq" images.

on several 160 class ImageNet subsets. Moreover, "Complete" results in large accuracy drop on INet (160 classes) from 62.89% to 49.22% (-13.67%) whereas with our method, the accuracy improves to 64.62% (+1.73%). These results highlight that our proposed framework is effective in mitigating model failures.

In summary, we make the following contributions:

1. We proposed **DCD**, a framework for mitigating model failures via data-centric debugging. In contrast to the traditional model development using training/validation/test splits, we construct debug seed/train/heldout datasets to systematically improve failure modes of the model.

2. We use the ℓ_2 distance in the feature space for efficiently retrieving perceptually similar images to a reference image from the large pool dataset $\mathcal{F}$. Our experiments suggest that DINO models are significantly better at discovering similar images compared to Resnets.

3. Using our framework, we achieve 29.62% accuracy on the debug-heldout set, compared to 0%, 23.09%, 20.17% for the baseline models. We also achieve significant improvements: 63.85% vs 58.81% for runner-up (+5.04%) on the ImageNet-V2 subset ("Complement" in

	LPIPS	**Ours**
Time (secs)	1147.58	**4.81**
Space (GBs)	1806.74	**7.63**
# of dims	484992	**2048**

Table 14.2: Comparison between LPIPS and our method for finding similar images from a set of 1 million images. Using our method, leads to a **99.58% reduction in both time and space.**

Table 14.1).

14.2 Related work

Dataset Design: Several previous works [1, 55, 394] use gigantic amounts of training data to achieve high performance. For example, models such as Basic [395] and CLIP [366] use web-scale datasets of sizes 6.6 billion and 400 million images respectively. The availability of large weakly labeled web data combined with self-supervision methods [396, 397, 398, 399, 400] has made it easier to train such models. However, recent work [401, 402] suggests that adding targeted training images may be more effective.

Debugging and Explainability: Most of the existing works on explainability of deep networks focus on inspecting the decisions for a single image [277, 278, 279, 280, 281, 284, 285, 286, 288, 298, 301, 302, 303, 304]. These include saliency maps [65, 68, 193], activation maps [282, 283, 287, 320, 321], removing image patches [403], interpreting local decision boundaries [322], finding influential [325] or counterfactual inputs [299, 323, 324]. However, examining several single-image explanations for debugging can be time-consuming. Therefore, several works focus on identifying failure modes across a large set of images[305, 306, 307, 308, 318, 319, 335, 404] or introducing datasets to stress test model performance on images with the main objects in uncommon or rare settings [352, 355, 357, 370]. Another class of works focus on making edits

to the model to modify its predictions on specific inputs [390, 391, 405].

Visual Similarity metrics: Several prior similarity metrics have been proposed in the literature [40, 49, 406]. The state-of-the-art LPIPS metric [41] concatenates intermediate features of a deep network to compute the embeddings.

14.3 Notation

Let set $\mathcal{A}$ consist of (image, label) pairs: $(\mathbf{x}, y) \in \mathcal{A}$, and $\mathcal{A}(i)$ denote the images with label i:

$$\mathcal{A}(i) = \{\mathbf{x} : (\mathbf{x}, i) \in \mathcal{A}\}$$

Given two sets: $\mathcal{A}$ and $\mathcal{B}$, we use: $(\mathcal{A} - \mathcal{B})(i)$ to denote $\mathcal{A}(i) - \mathcal{B}(i)$. We use $|\mathcal{A}|$ to denote the cardinality (number of elements) of $\mathcal{A}$, $[n]$ to denote the set: $[0, 1, \ldots, n-1]$ and $\|\mathbf{z}\|$ for the l_2 norm of vector $\mathbf{z}$. For image $\mathbf{x}$, $\Phi(\mathbf{x})$ denotes the penultimate layer output of the model Φ. We use $\mathcal{Y}$ to denote the set of all 1000 ImageNet classes.

14.4 Framework for model debugging

Consider the image classification problem $\mathcal{X} \rightarrow \mathcal{Y}$ where we want to predict the ground truth label $y \in \mathcal{Y}$ for input $\mathbf{x} \in \mathcal{X}$. Given a trained model, we have an *error distribution* $\mathcal{E}$ of incorrectly classified images i.e. every image sampled from $\mathcal{E}$ is misclassified by the model. We have access to a set of samples $\mathcal{E}_{sample}$ from $\mathcal{E}$ and it is very expensive to draw more samples. Here, $\mathcal{E}$ represents the deployment scenario where we we want to improve model performance. For example, we may be interested in images with people of color, specific gender or distribution

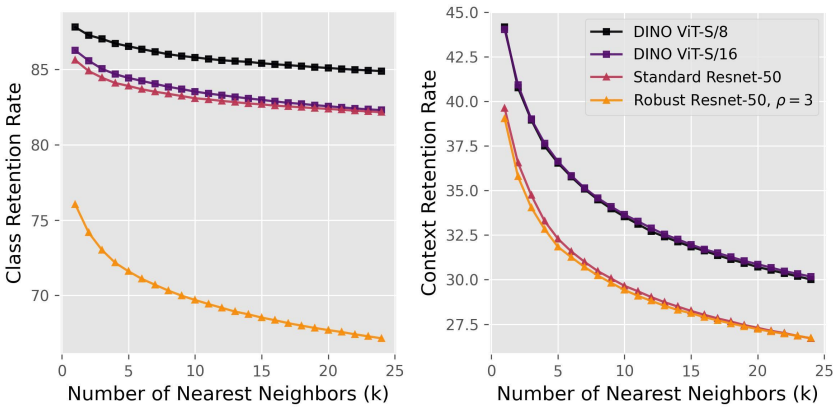


Figure 14.3: Class and context retention for nearest-neigbhor retrieval on the FOCUS dataset, using different models Φ. DINO ViTs are superior for both class andd context retention.

shift (e.g., people wearing masks during COVID-19), etc.

We also assume that we have access to a large pool of weakly labeled images (noisy labels) denoted by $\mathcal{F}$. Here, $\mathcal{F}$ can be obtained using any suitable data source depending on the problem. Using $\mathcal{F}$, we want to improve performance on $\mathcal{E}$ while maintaining on the desired test set(s).

One naive method could be to add the complete set $\mathcal{F}$ to the training set. However, because the labels in $\mathcal{F}$ can be very noisy, this can reduce the quality of the dataset and hurt model performance. Thus, we want to improve performance on $\mathcal{E}$ by selecting new training images from $\mathcal{F}$ while maintaining model performance on the desired test set(s).

Intuitively, we would expect an improved performance on $\mathcal{E}$ by selecting several images from $\mathcal{F}$ that are "visually similar" to $\mathcal{E}_{sample}$ images. However, identifying similar images from a large dataset can be difficult. Moreover, even if we successfully discover such images, we may see an improved performance on $\mathcal{E}_{sample}$ due to some patterns specific to $\mathcal{E}_{sample}$ images that may fail to generalize to new images from $\mathcal{E}$. For example, the pattern in one image may be a strong match for another image from different class (see Figure 14.5). Thus, we want an evaluation procedure that reflects the true model performance on $\mathcal{E}$ because the performance on $\mathcal{E}_{sample}$ may

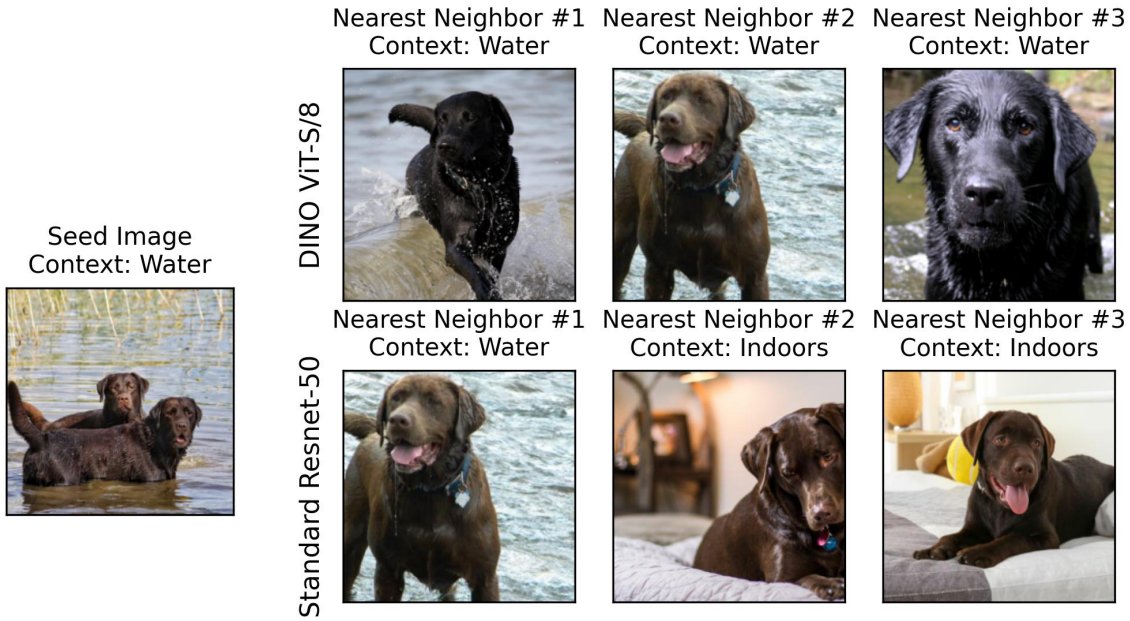


Figure 14.4: Example where nearest neighbor ($k = 3$) search on FOCUS dataset using DINO ViT-S/8 features retains context, while Standard Resnet-50 only does so in one of three cases.

be an overestimate of the same.

To address these challenges, we introduce **Data-Centric Debugging (DCD)**, illustrated in Figure 14.2: a framework for targeted data collection to mitigate failure modes of deep models and faithfully assess model performance on the error distribution. In the next subsections, we discuss the building blocks of our framework.

14.4.1 Visual similarity matching at scale

From the set $\mathcal{F}$, we would like to identify images that are "visually similar" to the images in $\mathcal{E}_{sample}$ and add to the training set as we would expect such images to be most effective for improving model performance. However, finding visually similar images to a reference image from a large pool of images can be difficult. Previous work on similarity matching, LPIPS [41] uses image embeddings that are very high dimensional. For example, for an image of size $224 \times$

224×3 (150528 dimensions), the LPIPS embedding using a pretrained AlexNet [55] model is of size $484,992$ (≈ 3.22 times the size of the original image). This leads to significant challenges because storing the LPIPS embedding takes a large amount of space and computing the LPIPS distance of a large number of images from some reference image can be very time-consuming.

In this chapter, we propose to use the penultimate layer output of a deep model Φ as the image embedding which is very compact (2048 for Resnet-50). Given two images $\mathbf{x}, \mathbf{z}$, we use the squared l_2 distance ($\|\Phi(\mathbf{x}) - \Phi(\mathbf{z})\|^2$) in this space as the visual similarity distance to discover visually similar images. In Table 14.2, we show that compared to LPIPS, our embedding leads to 99.58% reduction in both time and storage space, thereby making it feasible to obtain visually similar images from a large pool of images.

We experiment with four pretrained models Φ for computing these distances: Standard Resnet-50, Robust Resnet-50, DINO ViT-S/16 and DINO ViT-S/8 [387] (Appendix 14.A.3). To better inform the choice of the model Φ, we conduct an experiment on the FOCUS dataset [352]. FOCUS consists of common objects in various settings, leading to *two* labels per sample: one label denoting class (bird, plane, etc), and the other denoting context (snow, night, indoors, etc). We obtain features for every sample in FOCUS using various backbones, and then obtain $k = 25$ nearest neighbors per sample in the feature space of each model. We then compute the percent of neighbors amongst the top $k' \leq k$ that retain (i) object class and (ii) image context.

Figure 14.3 visualizes the results. We observe that the retention rate for class is far higher than for context. Retaining both class and context is important for our use case as the set $\mathcal{E}_{sample}$ may consist of instances of a class in an uncommon context and we would want to select more examples of that object in that same context. We find that DINO vision transformers are superior in *both* class and context retention compared to Resnet models. Specifically, DINO ViT-S/8

achieves the best class and context retention. This unique property makes DINO transformers a prime candidate for visual similarity computation and targeted image retrieval. In Figure 14.4, we show an example where the 3 nearest neighbors using DINO ViT-S/8 features retain context, but Standard Resnet-50 features do not.

14.4.2 Collecting large pool of images from the web

We want to collect a large pool of images from the web and identify images that are visually similar to the images in $\mathcal{E}_{sample}$. Since collecting a large number of images for all 1000 ImageNet classes can be time consuming, we first select 160 classes (denoted by $\mathcal{T}$) on which 20 highly accurate ImageNet trained models achieve low accuracy (see Appendix 14.A.1). For each class $i \in \mathcal{T}$, we obtain their synset (set of synonyms). For example, in the synset {'junco', 'snowbird'}, 'junco' and 'snowbird' are synonyms. For each synonym in the synset, we perform a Flickr search and collect the URLs of the first 30,000 images in the search results. After collecting URLs for all classes in $\mathcal{T}$, we remove URLs that were common across multiple classes. This results in a weakly labeled dataset (denoted by $\bar{\mathcal{F}}$) consisting of 952,951 images across 160 classes. Note that $\bar{\mathcal{F}}$ is weakly labeled because all images in the search results may not contain the relevant object in the search term.

14.4.3 Removing images visually similar to test sets

Since the model performance on test set can be trivially improved by adding images from the test set to the training set, it is critically important to ensure that the new images added to the training set are "sufficiently different" from the test set images. To this end, we introduce a filtration step



Figure 14.5: left: wrongly predicted image with label **king snake**, right: similar images obtained using DINO ViT-S/8 model with two different methods. Selecting images without using any weak labels outputs samples where the main object, in this case, king snake, is absent, which can introduce wrongly labeled images in the dataset. More examples are in Appendix 14.B.6.

based on the criteria that the newly added images should be at least as different from test set images as they are between the ImageNet train/test sets (see Appendix 14.A.4). Thus for each class $i \in \mathcal{T}$, we first compute a threshold visual similarity distance $\tau(i)$ using the ImageNet dataset.

Let $\mathcal{U}$ denote the union of all test sets that we want to evaluate our model on. This includes the seed set, heldout set, and all test sets. We select the images $\mathbf{x} \in \bar{\mathcal{F}}(i)$ that have visual similarity distance $> \tau(i)$, from all images $\mathbf{z} \in \mathcal{U}(i)$. The new dataset constructed by selecting such images is denoted by $\mathcal{F}$. We select "visually similar" from $\mathcal{F}$ to prevent images identical to the test set from being selected.

14.4.4 Debug-seed and -heldout sets

We divide the set $\mathcal{E}_{sample}$ into two disjoint sets: $\mathcal{E}_{seed}$ and $\mathcal{E}_{heldout}$. Using $\mathcal{E}_{seed}$, we want to add images from $\mathcal{F}$ to the training set that result in improved model performance on $\mathcal{E}_{heldout}$. We stress that $\mathcal{E}_{heldout}$ is never used for data collection. The intuition here is that since we are only using $\mathcal{E}_{seed}$ to collect new images and $\mathcal{E}_{heldout}$ is from the same distribution $\mathcal{E}$, an improved performance

Dataset	Class subset	# of images
Seed	160 classes ($\mathcal{T}$)	1,031
Heldout		719
MFreq	160 classes ($\mathcal{T}$)	1,600
	1000 classes ($\mathcal{Y}$)	10,000
Comple-ment	160 classes ($\mathcal{T}$)	1,668
	1000 classes ($\mathcal{Y}$)	10,683

Table 14.3: Dataset sizes. Seed and Heldout are for single-model.

would suggest that the model is learning relevant concepts (not overfitting to $\mathcal{E}_{seed}$). This leads to the following definitions:

- **Debug-Seed Set**: set of images ($\mathcal{E}_{seed}$) sampled from $\mathcal{E}$ used to collect new training data to improve performance
- **Debug-Heldout Set**: set of images ($\mathcal{E}_{heldout}$) sampled from $\mathcal{E}$ disjoint from $\mathcal{E}_{seed}$ that is used to evaluate performance of model trained on images collected using $\mathcal{E}_{seed}$.

We remark that this is similar to the validation/test setup in model development. We construct the debug-seed and heldout sets in two settings, (a) single-model: images incorrectly classified by a single model (We use a Standard Resnet-50) and (b) multiple-model: images incorrectly classified by 20 highly accurate models (see Appendix 14.A.2). In both these settings, we use the 160 class subset ($\mathcal{T}$) for which we obtained Flickr images (Section 14.4.2).

We use the ImageNet-V2 dataset [393] to sample the seed $\mathcal{E}_{seed}$ and heldout $\mathcal{E}_{heldout}$ sets. ImageNet-V2 consists of three (non-disjoint) sets namely, (a) MatchedFrequency, (b) Threshold0.7, (c) TopImages. We observe that models achieve the lowest accuracy on MatchedFrequency (or "MFreq") set [393]. Thus, we use the incorrectly classified images from this set to construct the set $\mathcal{E}_{seed}$. This ensures that $\mathcal{E}_{seed}$ has a large size. We want the heldout set $\mathcal{E}_{heldout}$

to be disjoint from $\mathcal{E}_{seed}$. Thus, we take the union of all the three sets and remove the "MFreq" images from the union to define the "Complement" set.

In the single-model setting, we construct the seed set ($\mathcal{E}_{seed}$) by selecting images from "MFreq" with labels in $\mathcal{T}$ that were incorrectly classified by the Resnet-50 model. For the heldout set ($\mathcal{E}_{heldout}$), we select the incorrectly classified images from the "Complement" set, again from the 160 classes in $\mathcal{T}$. The sizes of these datasets are in Table 14.3.

In the multiple-model setting, the procedure is similar except that we select images from 160 classes that were incorrectly classified by each the 20 models.

14.4.5 Debug-train and -validation sets

We use the images in $\mathcal{E}_{seed}$ to select new images from $\mathcal{F}$ and add them to the training set. We may also want to validate that upon training the model on these new images, the performance improves on images visually similar to $\mathcal{E}_{seed}$. Thus, we define the debug-train and debug-validation sets:

- **Debug-Train Set**: set of images selected from $\mathcal{F}$ and added to the training set to improve model performance.
- **Debug-Validation (De-Val) Set**: set of images selected from $\mathcal{F}$ (and disjoint from debug-train set) to validate that the model performance improves on images visually similar to the images in $\mathcal{E}_{seed}$.

For each image $\mathbf{x} \in \mathcal{E}_{seed}(i)$, the de-val should contain a set of few (say k) images visually similar to $\mathbf{x}$ with label i. We may construct this set by selecting k images with the smallest visual similarity distance to $\mathbf{x}$ from $\mathcal{F}(i)$. However, for two images $\mathbf{x}, \mathbf{z} \in \mathcal{E}_{seed}(i)$, the sets of k images may overlap. Thus, some seed images may have fewer (than k) images included and not be well

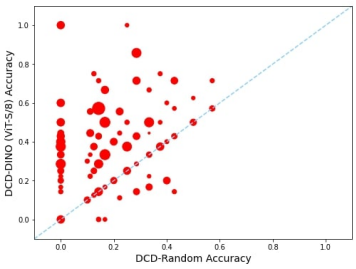
Debug-Train method	Accuracy on different sets							
	incorrectly class.		subset of 160 classes			all 1000 classes		
	Seed	Heldout	MFreq	Compl.	INet	MFreq	Compl.	INet
Original	0%	0%	35.56%	56.89%	62.89%	63.70%	76.12%	76.47%
Complete	18.53%	23.09%	37.56%	53.84%	49.22%	61.70%	72.93%	75.07%
Random	16.78%	20.17%	41.56%	58.81%	63.91%	63.77%	75.21%	76.37%
(ViT-S/8)	36.28%	**29.62%**	54.06%	**63.85%**	**64.62%**	65.28%	**76.42%**	76.54%
(ViT-S/16)	**36.76%**	26.98%	**55.00%**	62.83%	64.11%	**65.69%**	75.38%	76.41%
(Standard)	32.39%	28.09%	51.75%	63.31%	64.57%	65.62%	75.96%	**76.70%**
(Robust)	33.07%	26.84%	51.5%	62.29%	62.27%	65.04%	75.70%	76.50%

Table 14.4: Results for *single-model* seed and heldout sets. "INet" denotes the ImageNet test set, "MFreq" denotes the ImageNet-V2[393] MatchedFrequency set, "Compl." denotes the Complement set i.e. all ImageNet-V2 images excluding "MFreq" images.

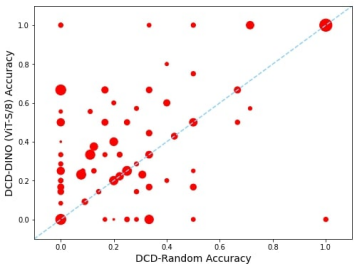
represented. To address this limitation, we use an algorithm (in Appendix 14.A.5) that removes the selected images on the fly and avoids overlaps. The resulting de-val set is denoted by $\mathcal{V}$. We construct the debug-train set using a similar procedure. Because we want debug-train set to be disjoint from the debug-val set ($\mathcal{V}$), we find visually similar images from the subset: $(\mathcal{F} - \mathcal{V})(i)$. The procedure is same as the de-val set except that we use: $(\mathcal{F} - \mathcal{V})$ instead of $\mathcal{F}$. We first construct the de-val set using $k = 4$ followed by the debug-train set using $k = 46$.

14.5 Experiments

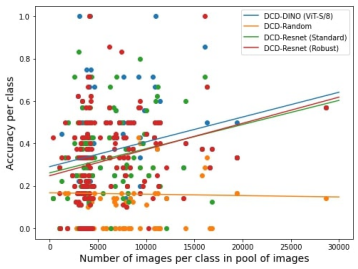
In this Section, we discuss results using the single-model seed and heldout sets (Section 14.4.4). Results for multiple-model are similar and discussed in Appendix 14.B.1. We evaluate our proposed method on two criteria: the improvement in accuracy on the heldout-debug set and the accuracy drop on the ImageNet, MFreq and Complement test sets. For each of these test sets, we evaluate on both 160 class subset and all 1000 classes. We use the Resnet-50 architecture for training all models. Each model was trained for 90 epochs over eight GPUs (RTX 2080 Ti). We



(a) accuracy for 160 classes for DCD-DINO (Y-axis) and -Random (X-axis) on seed set



(b) accuracy (160 classes) for DCD-DINO (Y) and -Random (X) on heldout set



(c) heldout set accuracy for each class $i \in \mathcal{T}$ (Y-axis) as $|\mathcal{F}(i)|$ increases (X-axis)

Figure 14.6: In (a) and (b), we plot DCD-DINO accuracy (Y-axis) and DCD-Random (X-axis) for all 160 classes. The size of each dot is proportional to the number of images in $\mathcal{F}$ for that class ($|\mathcal{F}(i)|$) and dashed line represents $Y = X$. In (c), we plot the accuracy per class on the heldout set as $|\mathcal{F}(i)|$ increases. We also show the corresponding linear regression lines.

use the composer library for training all models to reduce training time [407].

14.5.1 Baseline models

We compare against three baseline models:

- **Original**: trained on the ImageNet training set (no additional training images are added)
- **Complete**: We add the complete dataset $\mathcal{F}$ of size $952,022$ (Section 14.4.3). Since this leads to a disproportionately large number of images from the 160 classes ($\mathcal{T}$) in $\mathcal{F}$, we assign weights $(w(i))$ to these classes so that the total weight for each class is the same as in ImageNet:

$$\forall\, i \in \mathcal{T},\;\; w(i) \times |\mathcal{I}_{train}(i) \cup \mathcal{F}(i)| = |\mathcal{I}_{train}(i)|$$

- **Random**: For each class $i \in \mathcal{T}$, we randomly select $46 \times |\mathcal{E}_{seed}(i)|$ images (without replacement) from $\mathcal{F}(i)$. Note that this set has the same size as our debug-train set. This ensures a fair comparison across models. From Table 14.3, the dataset has size $1031 \times 46 = 47426$.

14.5.2 Table details

In Table 14.4, we show results on different test sets namely:

- **MFreq**: only the ImageNet-V2 MatchedFrequency set
- **Complement**: all ImageNet-V2 images except Mfreq
- **INet**: standard ImageNet test set.

Details for "MFreq" and "Complement" are in Section 14.4.4.

In "Accuracy on different sets", we show the model accuracy on various subsets of the test sets. In "160 classes", we show the accuracy on images from classes in $\mathcal{T}$ and "1000 classes": from all 1000 classes in ImageNet. In "incorrectly class.", accuracy on images (from 160 class subset) that are incorrectly classified by the "original" model.

In the last four rows, "Debug-Train method" denotes the model Φ used for computing visual similarity distances. "ViT-S/8" and "ViT-S/16" denote the models trained using DINO ViT-S/8 and ViT-S/16 features for similarity matching. "Standard" and "Robust" denote the models trained using Standard and Robust Resnet-50 features.

14.5.3 Discussion

adding the complete set: While one may believe that noisy data is better than no data, we observe that naively adding noisy data ($\mathcal{F}$) has diminishing returns. In "Complete", 952,022 extra images are added while in "Random" only 47,426. Even though we add 20 $\times$ more images in "Complete", accuracy on seed and heldout are only marginally better. In fact, under other metrics, such as "160 classes" (INet), accuracy for "Complete" is 49.22% significantly below

"Original" 62.89% (-13.67%) and "Random" 63.91% (-14.69%). This suggests that naively adding large amounts of noisy data can hurt model performance.

comparing models: We want to compare the quality of image embeddings for visual similarity matching obtained using different models. We observe that both DINO models achieve significantly higher accuracy on "Seed" compared to the Resnet models: DINO ViT-S/8 achieves 36.28% compared to 33.07% for Robust Resnet-50. This provides evidence that DINO models are better suited for similarity matching. Similar trend is also observed for the Heldout set. However, between Standard and Robust Resnet-50, the results on "Seed" are comparable suggesting that adversarial robustness is not critical for similarity matching.

comparing baselines and our method: We observe that DINO "ViT-S/8" achieves significantly improved results compared to both "Complete" and "Random". On "Heldout", we achieve 29.62% compared to 23.09% for the next best i.e. gain of 6.53%. On the "Complement" set (160 classes), we achieve 63.85%: gain of 5.04% compared to 58.81% for the next best model. On the 1000 classes sets, we achieve slightly improved results on all sets: 76.42% on "Complement", compared to 76.12% (+0.3%). Similarly on INet, we achieve 76.54% similar to 76.47%. The model performance is maintained on all the test sets.

accuracy on different classes: We now compare the seed and heldout accuracy for different classes $i \in \mathcal{T}$ as $|\mathcal{F}(i)|$ varies. In both Figures 14.6a and 14.6b, we observe that "DINO" achieves better accuracy on most classes comparing to "Random" (as most points lie above the dashed y=x line). Also, "DINO" achieves better accuracy for classes with larger amount of data (larger red dots). In Figure 14.6c, we compare accuracy per class as $|\mathcal{F}(i)|$ increases (x-axis) for four different models: DCD-Resnet (Standard and Robust), DCD-DINO (ViT-S/8) and DCD-Random. We see that as $|\mathcal{F}(i)|$ increases, we achieve better accuracy (according to the linear regression lines)

for all methods except DCD-Random. For DCD-Random, there is a slight accuracy reduction for large $|\mathcal{F}(i)|$. Our results provide evidence that by obtaining large amounts of weakly-labeled data and adding selected images to the training sets, we can achieve significantly improved results.

Supplementary material

14.A Implementation details

14.A.1 Selecting 160 classes for model debugging

We first selected 30 models with different architectures from the timm library. From these 30 models, we selected 20 models that achieved highest accuracy on the ImageNet-V2 set. The 20 models were: swin_base_patch4_window7_224, swin_small_patch4_window7_224, convit_base, deit_base_patch16_224, convit_small, swin_tiny_patch4_window7_224, resnet50d, mixnet_xl, seresnet50, deit_small_patch16_224, resnext50_32x4d, efficientnet_b4, resnet50, efficientnet_b3, wide_resnet101_2, efficientnet_b0, vit_base_patch16_224, resnet34, mnasnet_a1, vit_small_patch16_224.

ImageNet-V2 consists of 3 test sets namely: MatchedFrequency, TopImages and Threshold0.7. We used the MatchedFrequency version because models achieve the lowest accuracy on this set (making it suitable for debugging). Next, we selected classes with at least 3 images on which all 20 selected models were inaccurate on the MatchedFrequency set.

This resulted in total 160 classes.

14.A.2 Set of models used in the multiple-model setting

We used the same models as in Section 14.A.1 except that the Resnet-50 model included was not from the timm library but another Resnet-50 trained from scratch by us. We used the same Resnet-50 as the one used in the single-model setting as this makes the comparison between multiple-model and single-model settings easier. This again resulted in total 20 models.

14.A.3 Models used for visual similarity matching

We experiment with four pretrained models Φ for computing these distances: Standard Resnet-50, Robust Resnet-50, DINO ViT-S/16 and DINO ViT-S/8 [387]. Here, Standard Resnet-50 model is the original trained model we are trying to debug. We use Robust Resnet-50 because adversarially robust models have the unique property that if you try to optimize two visually different images to minimize the visual similarity distance between them (using the robust model as the feature extractor), the resulting images look visually very similar. The same is not true for standard models as shown in [274]. This suggests that robust models may be better suited for visual similarity matching and we investigate this in the chapter. We selected the DINO ViT-S/16 and ViT-S/8 models because the penultimate layer features for these models are known to be good kNN classifiers [387] achieving 74.5% and 78.3% accuracy on ImageNet respectively.

14.A.4 Removing images "visually similar" to test-sets

Since the model performance on test set can be trivially improved by adding images from the test set to the training set, it is critically important to ensure that the new images added to the training set are "sufficiently different" from the test set images. To this end, we rely on the representation of a Robust Resnet-50 model Φ_r (for two images $\mathbf{x}, \mathbf{z}$, $\|\Phi_r(\mathbf{x}) - \Phi_r(\mathbf{z})\|^2$ is the visual similarity metric). Using this metric, we argue that for each class, the newly added images should be at least as different from test set images as they are between the ImageNet train/test sets.

Let $\mathcal{I}_{train}$ and $\mathcal{I}_{test}$ denote the ImageNet train and test sets. Thus, for each label $i \in \mathcal{T}$, we

first compute a threshold visual similarity distance $\tau(i)$ using ImageNet as follows:

$$\tau(i) = \min \|\Phi_r(\mathbf{x}) - \Phi_r(\mathbf{z})\|^2$$

$$\text{where } \mathbf{x} \in \mathcal{I}_{train}(i), \quad \mathbf{z} \in \mathcal{I}_{test}(i)$$

For each class i, $\tau(i)$ captures the minimum perceptual distance that should exist between the train and test images for class i. Let $\mathcal{U}$ denote the union of all test sets that we want to evaluate our model on. This includes the seed set, heldout set and every test set on which we want to maintain model performance. We want to select the images $\mathbf{x} \in \mathcal{F}(i)$ that have visual distance $> \tau(i)$, from all images $\mathbf{z} \in \mathcal{U}(i)$. To this end, we construct a new dataset $\mathcal{F}$ as follows:

$$\mathcal{F}(i) = \{\mathbf{x} \in \bar{\mathcal{F}}(i) : \min_{\mathbf{z} \in \mathcal{U}(i)} \|\Phi_r(\mathbf{x}) - \Phi_r(\mathbf{z})\|^2 > \tau(i)\}$$

We use the Robust Resnet-50 model for removing similar images because for a standard model, using gradient based adversarial attacks, it is possible to construct pairs of images that look exactly the same to the human eye, yet map to different representations. However, for a robust model, such adversarial attacks lead to changes that are visible to the human eye and the resulting pairs of images are visually different. Thus, even if $\|\Phi(\mathbf{x}) - \Phi(\mathbf{z})\|^2$ is very large for a standard model, $\mathbf{x}$ and $\mathbf{z}$ may still look identical to a human. This is further illustrated in Figure 14.7. Thus suggests that using a robust model leads to a more reliable metric for removing images that are similar to some reference image.



(a) original image (b) standard model (c) robust model

Figure 14.7: (a) initial image (denoted by $\mathbf{x}$). We solve $\mathbf{z} = \arg\max_{\mathbf{z}:\|\mathbf{z}-\mathbf{x}\|\leq\rho} \|\Phi(\mathbf{z}) - \Phi(\mathbf{x})\|^2$ using Φ as Standard Resnet-50 in (b) and Robust Resnet-50 in (c). In (b), $\mathbf{x}$ and $\mathbf{z}$ look identical to a human eye. In (c), $\mathbf{x}$ and $\mathbf{z}$ are visually very different.

14.A.5 Algorithm for constructing debug-validation and debug-train sets

To avoid overlaps when selecting "visually similar" images per image, we use the below procedure:

- For each $\mathbf{x} \in \mathcal{E}_{seed}(i)$, we initialize the set: $\mathcal{N}[\mathbf{x}] = \{\}$.
- For all pairs $\mathbf{x} \in \mathcal{E}_{seed}(i),\ \mathbf{z} \in \mathcal{F}(i)$, we compute the distance $\|\Phi(\mathbf{x}) - \Phi(\mathbf{z})\|^2$. This gives the matrix D.
- Let $\mathbf{x}^*,\ \mathbf{z}^*$ be the pair with the minimum distance in D. We add $\mathbf{z}^*$ to $\mathcal{N}[\mathbf{x}^*]$ and set $\mathrm{D}[:, \mathbf{z}^*] = \infty$ (this prevents $\mathbf{z}^*$ from being selected again, thus no overlaps).
- If the number of elements in $\mathcal{N}[\mathbf{x}^*]$ equals to k, we set $\mathrm{D}[\mathbf{x}^*, :] = \infty$ (this prevents $\mathbf{x}^*$ from being selected again if $\mathcal{N}[\mathbf{x}^*]$ contains k elements).
- We repeat this procedure until $\forall\ \mathbf{x}, \mathcal{N}[\mathbf{x}]$ is of size k.

This is equivalent to Algorithm 7 with $\mathcal{Q} = \mathcal{E}_{seed}(i), \mathcal{W} = \mathcal{F}(i)$. The complete de-val set is given by:

$$\mathcal{V}(i) = \cup_{\mathbf{x} \in \mathcal{E}_{seed}(i)} \mathcal{N}[\mathbf{x}] \tag{14.1}$$

Similarly, the debug-train procedure is constructed using the inputs $\mathcal{Q} = \mathcal{E}_{seed}(i), \mathcal{W} = (\mathcal{F} - \mathcal{V})(i)$ to Algorithm 7.

Algorithm 7 Finding disjoint sets of similar images

procedure DISJOINT_SETS(seed set $\mathcal{Q}$, , web data: $\mathcal{W}$, count: k)
  for all $\mathbf{x} \in \mathcal{Q}$ **do**
    $\mathcal{N}[\mathbf{x}] \leftarrow \{\}$
    for all $\mathbf{z} \in \mathcal{W}$ **do**
      $\mathrm{D}[\mathbf{x}, \mathbf{z}] \leftarrow \|\Phi(\mathbf{x}) - \Phi(\mathbf{z})\|^2$
    end for
  end for
  while $\exists\, \mathbf{x} : \ |\mathcal{N}[\mathbf{x}]| < k$ **do**
    $\mathbf{x}, \mathbf{z} = \arg\min \mathrm{D}$
    $\mathcal{N}[\mathbf{x}] \leftarrow \mathcal{N}[\mathbf{x}] \cup \mathbf{z}$
    $\mathrm{D}[:, \mathbf{z}] \leftarrow \infty$
    if $|\mathcal{N}[\mathbf{x}]| == k$ **then**
      $\mathrm{D}[\mathbf{x}, :] \leftarrow \infty$
    end if
  end while
end procedure

14.B Additional results

14.B.1 Results in the multiple-model setting

The seed and heldout sets contain 563 and 264 images respectively.

Debug-Train method	Accuracy on different sets							
	incorrectly class.		subset of 160 classes			all 1000 classes		
	Seed	Heldout	MFreq	Compl.	INet	MFreq	Compl.	INet
Original	0%	0%	35.56%	56.89%	62.89%	63.70%	76.12%	76.47%
Complete	8.88%	7.95%	37.56%	53.84%	49.22%	61.70%	72.93%	75.07%
Random	2.66%	3.03%	40.31%	59.35%	63.95%	63.76%	75.96%	76.60%
ViT-S/8	19.89%	**8.33%**	47.31%	60.25%	63.74%	65.32%	75.81%	76.61%
ViT-S/16	**21.67%**	4.92%	**47.38%**	60.43%	63.94%	64.93%	76.03%	76.49%
Standard	17.58%	7.57%	46.56%	**61.27%**	**64.34%**	64.69%	**76.23%**	**76.63%**
Robust	15.98%	6.06%	45.31%	60.43%	64.00%	64.95%	75.80%	76.58%

Table 14.5: Results for *multiple-model* seed and heldout sets. "INet" denotes the ImageNet test set, "MFreq" denotes the ImageNet-V2[393] MatchedFrequency set, "Compl." (Complement) denotes the set of all ImageNet-V2 images excluding "MFreq" images.

14.B.2 Table details

In Table 14.5, the columns "Seed" and "Heldout" show the accuracy on images in "Mfreq" and "Complement" sets respectively from the 160 classes ($\mathcal{T}$) that were incorrectly classified by the all 20 models (14.A.2). Note that these are equivalent to the seed and heldout sets discussed in Section 14.4.4.

In the last four rows, "Debug-Train method" denotes the model Φ used for computing visual similarity distances. DCD-DINO (ViT-S/8) and (ViT-S/16) denote the models trained using DINO ViT-S/8 and ViT-S/16 models. DCD-Resnet (Standard) and (Robust) denote the models trained using Standard and Robust Resnet-50 models.

14.B.3 Discussion

For the "Debug-Train method": DCD-DINO (ViT-S/8), in the column "incorrectly classified" achieves the highest accuracy on Heldout: 8.33%. Notably, on heldout, the accuracy is only

slightly higher than "DCD-Complete" (0.38%). However, "DCD-Complete" shows 13.67% drop in performance on ImageNet (160 class subset). Although the accuracy of DCD-Random is similar to original model on "INet (160)", it performs considerable worse compared to DCD-DINO (ViT-S/8) on Heldout: we achieve 8.33% significantly better the "DCD-Random" method (3.03% i.e. gain of 5.3%).

While one may believe that the images misclassified by 20 ImageNet trained models would have multiple objects or may be mislabeled. We find that this is not the case for several images from both the seed and heldout-debug sets. We show images correctly classified by our model and misclassified by 20 ImageNet trained models in Appendix 14.B.4 (for the seed-debug set) and Appendix 14.B.5 (for the heldout-debug set).

14.B.4 Examples of images from the seed-debug set

Figures 14.8, 14.9, 14.10 and 14.11 show several images from the seed-debug set on which we obtain correct predictions.

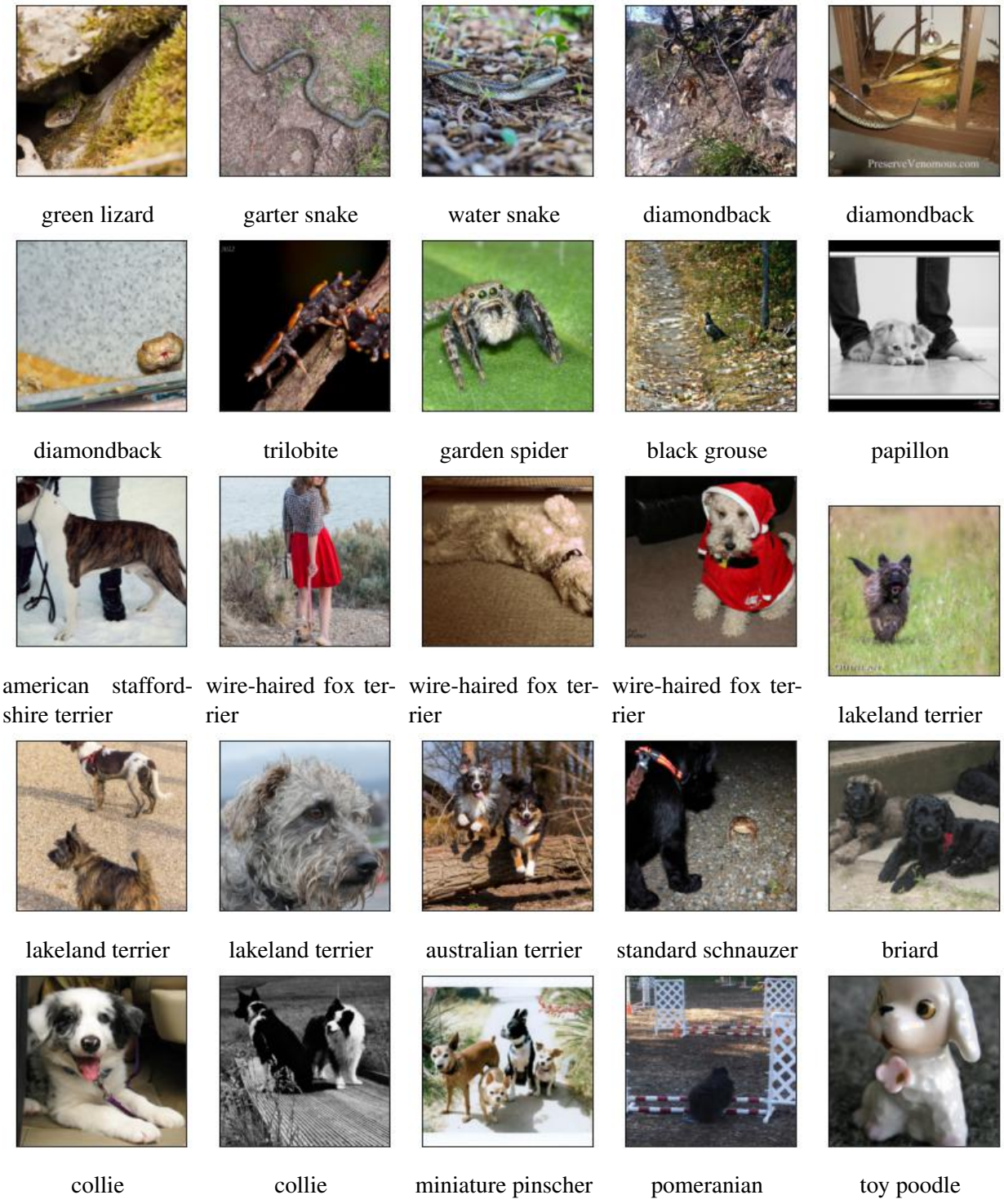


Figure 14.8: Examples of images (from the seed-debug set) on which 20 highly accurate ImageNet trained models predict incorrectly (ground truth label below each image). However, models trained using our framework make correct predictions on all of them.

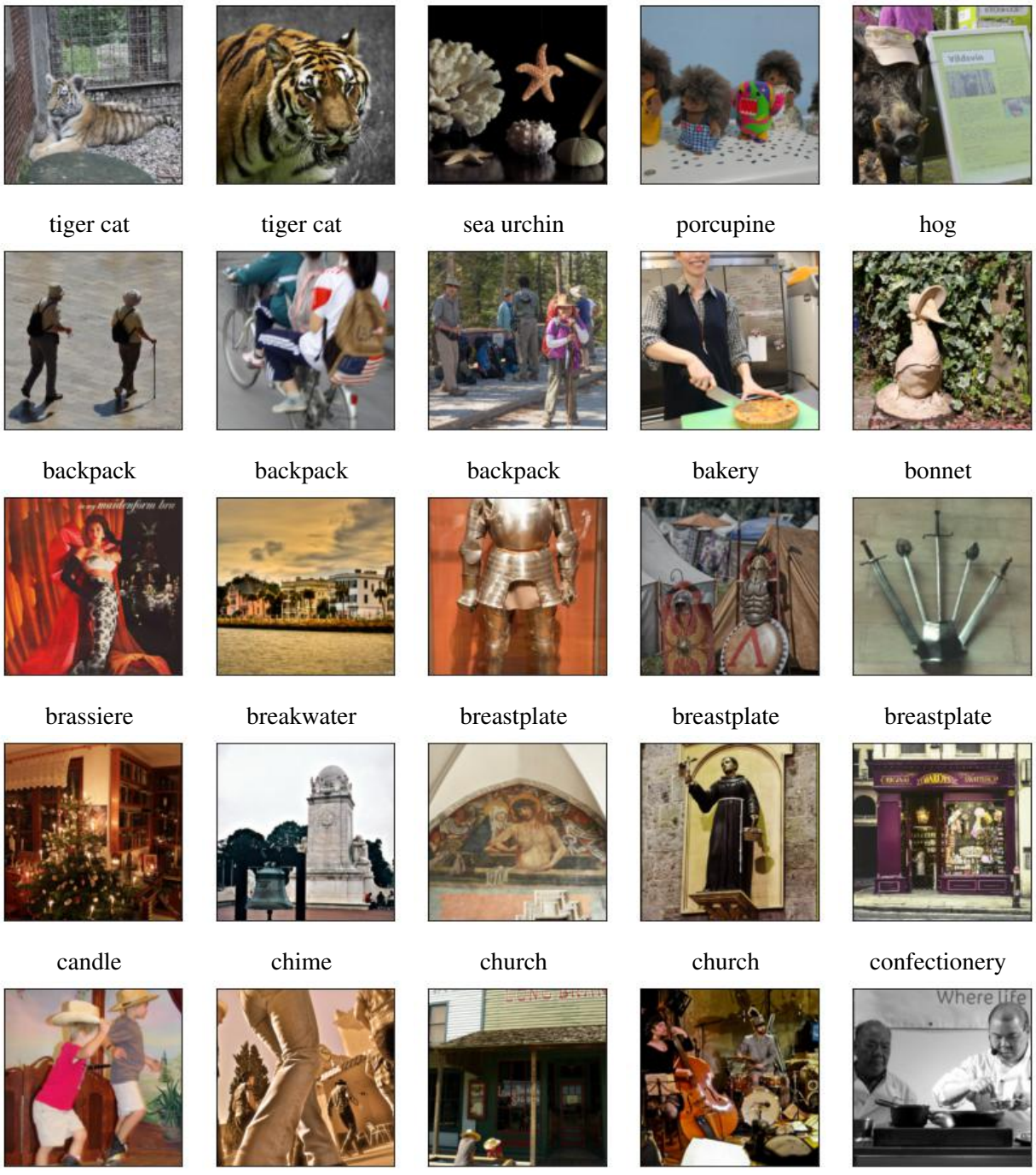


Figure 14.9: Examples of images (from the seed-debug set) on which 20 highly accurate ImageNet trained models predict incorrectly (ground truth label below each image). However, models trained using our framework make correct predictions on all of them.

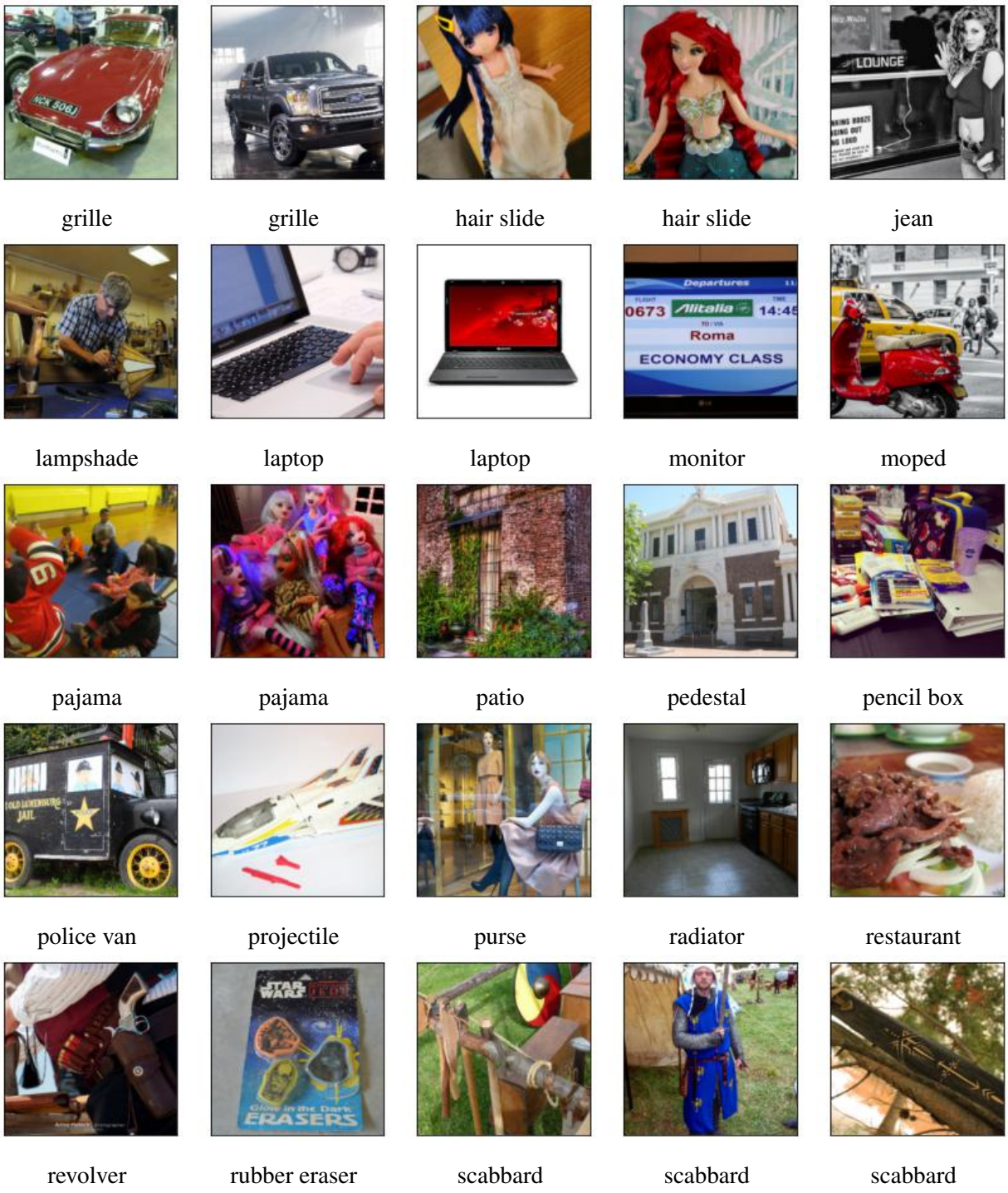


Figure 14.10: Examples of images (from the seed-debug set) on which 20 highly accurate ImageNet trained models predict incorrectly (ground truth label below each image). However, models trained using our framework make correct predictions on all of them.



Figure 14.11: Examples of images (from the seed-debug set) on which 20 highly accurate ImageNet trained models predict incorrectly (ground truth label below each image). However, models trained using our framework make correct predictions on all of them.

14.B.5 Examples of images from the heldout-debug set

Figure 14.12 shows images several from the heldout-debug set on which we obtain correct predictions.



Figure 14.12: Examples of images (from the heldout-debug set) on which 20 highly accurate ImageNet trained models predict incorrectly (ground truth label below each image). However, models trained using our framework make correct predictions on all of them.

14.B.6 Comparing between images discovered using weak labels and no labels

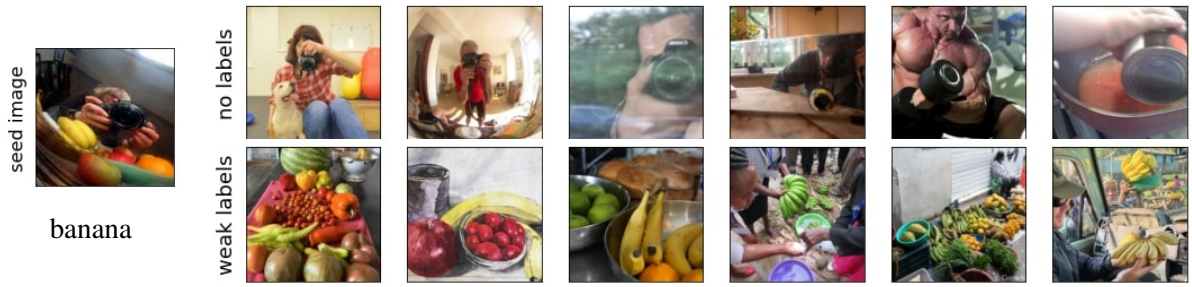


Figure 14.13: left: image with label **banana**, right: similar images obtained using the two different methods

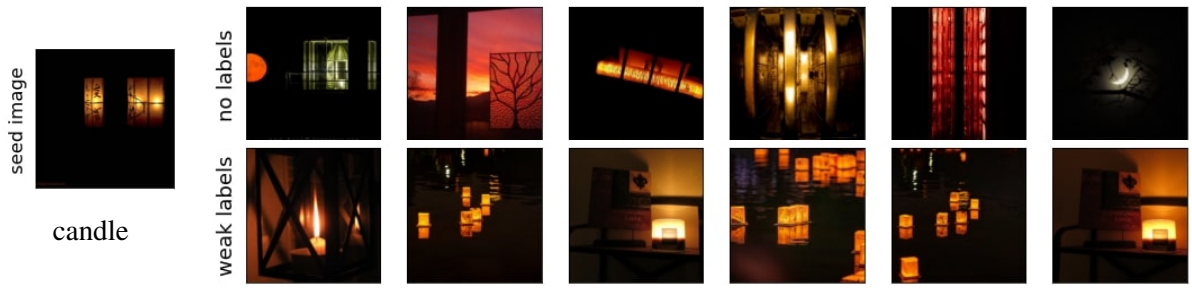


Figure 14.14: left: image with label **candle**, right: similar images obtained using the two different methods

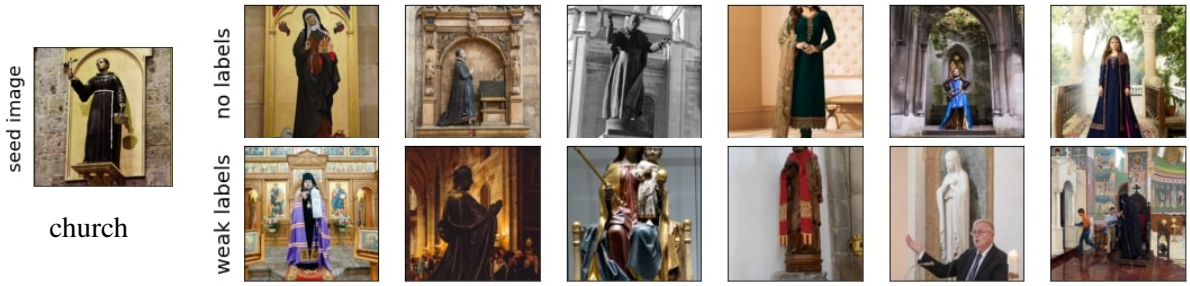


Figure 14.15: left: image with label **church**, right: similar images obtained using the two different methods



Figure 14.16: left: image with label **church**, right: similar images obtained using the two different methods

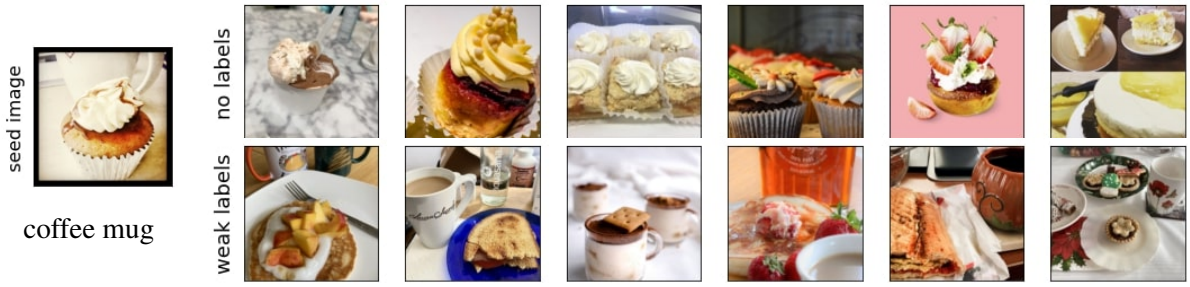


Figure 14.17: left: image with label **coffee mug**, right: similar images obtained using the two different methods



Figure 14.18: left: image with label **frypan**, right: similar images obtained using the two different methods



Figure 14.19: left: image with label **ice cream**, right: similar images obtained using the two different methods

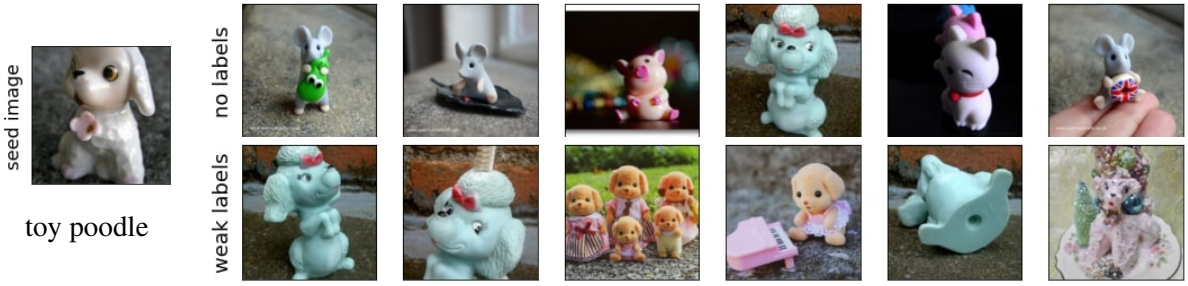


Figure 14.20: left: image with label **toy poodle**, right: similar images obtained using the two different methods



Figure 14.21: left: image with label **wine bottle**, right: similar images obtained using the two different methods

Part V

Conclusion

Chapter 15: Conclusions and Future work

15.0.1 Summary

In this dissertation, we presented several techniques for improving robustness against adversarial examples and identifying/mitigating the failure modes for deep learning systems.

In the first part, we presented an algorithm for making neural networks robust against unforeseen adversarial attack threat models. Our proposed method leads to significantly improved results against union of several threat models despite not being trained on any of them.

In the second part, we analyzed the usefulness of second-order information for deep learning interpretability and robustness. We showed that second-order information can be useful for saliency maps when the model confidence in the predicted class is low. We then showed that using smooth activation functions with low curvature values leads to significantly reduced robust overfitting. Using the second-order information, we introduced a procedure of robustness certification for deep networks that achieves state-of-the-art results.

In the third part, we presented several building blocks for Lipschitz neural networks: spectral bounds for convolution layers, orthogonal convolutions, activations with orthogonal jacobian, lipschitz pooling layers and robustness certification procedures for these networks. Our proposed methods establish new state-of-the-art far exceeding the prior works.

In the fourth part, we introduced the idea of using neurons of adversarially robust models

as visual attribute detectors. To discover the visual attribute encoded in neurons, we introduced a method for visualizing them. Using these visualizations, we presented several methods for discovering failure modes and analyzing the sensitivity of trained models to spurious features. Our analysis reveals several ImageNet classes that are highly vulnerable to spurious features. For these classes, we introduce the Hard ImageNet dataset consisting of segmentations of the object for all train/test images. Finally, we presented a method for mitigating model failure modes by discovering images from the web that are visually similar to the failure mode images.

15.0.2 Future Directions

Expressive 1-Lipschitz networks: Lipschitz networks are constructed by designing each layer to be 1-Lipschitz and enjoy several desirable properties such as provable adversarial robustness, stable training, interpretable gradients etc. However, the performance of current 1-Lipschitz networks significantly lags behind other models such as Resnet, ViTs etc. Thus, there remains a significant performance gap and reducing this is an important direction of future research.

Provable robustness for general threat models: The Lipschitz networks presented in the work focus on certifying model robustness against l_2 threat models. However, models can often be vulnerable to more general threat models such as l_∞, JPEG, RecolorAdv attacks etc. Certifying provable robustness against such threat models is an open problem for future research.

Robustness against spurious features: Our analysis reveals that several ImageNet classes such as seat belt (which occupies few pixels in the image) or space bar (which occurs with keyboard keys) etc. are highly susceptible to spurious features. We show that even if we remove the main object from the image, the model continues to predict the object suggesting that predictions are

heavily based on spurious signals. Further improving model robustness against spurious features while minimally reducing the clean accuracy is an important direction of future research.

Dataset design: In Chapter 14, we show that carefully selecting few images from a large set of images and adding them to the training set can be far more effective for improving model performance on the desired distribution compared to indiscriminately adding the complete set. Our results suggest that carefully designing the training set to reflect the situations likely to be encountered during deployment can be much more effective for obtaining high performant models (compared to adding all images available) and this is an important direction of future research.

Debugging models by fine-tuning: Our proposed debugging framework (in Chapter 14) requires re-training a model from scratch to improve performance on the failure mode distribution. Improving model performance on the distribution via fine-tuning instead of re-training from scratch is an important problem for future research.

Extension to other modalities: The problem of spurious features and failure modes is ubiquitous and exists in several other domains such as audio, text and multimodal settings [408]. Coming up with visualization methods that enable efficient discovery of failure modes, spurious features for these modalities and mitigating these problems is an important direction of future research.

Bibliography

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[2] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, Los Alamitos, CA, USA, jun 2015. IEEE Computer Society.

[3] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[4] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016. cite arxiv:1609.03499.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[6] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[9] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[10] Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[11] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

[12] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17(1):1334–1373, jan 2016.

[13] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[14] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *J. Artif. Int. Res.*, 61(1):215–289, jan 2018.

[15] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[16] Sara Beery, Grant Van Horn, and Pietro Perona. Recognition in terra incognita. *CoRR*, abs/1807.04975, 2018.

[17] John R. Zech, Marcus A. Badgeley, Manway Liu, Anthony Beardsworth Costa, Joseph J. Titano, and Eric Karl Oermann. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study. *PLoS Medicine*, 15, 2018.

[18] Alex J. DeGrave, Joseph D. Janizek, and Su-In Lee. Ai for radiographic covid-19 detection selects shortcuts over signal. *Nature Machine Intelligence*, 3(7):610–619, 2021.

[19] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1625–1634, 2018.

[20] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[21] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion Attacks against Machine Learning at Test Time. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, Lecture Notes in Computer Science, pages 387–402, Berlin, Heidelberg, 2013. Springer.

[22] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *ArXiv*, abs/1607.02533, 2016.

[23] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1369–1378, 2017.

[24] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[25] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. Theoretically principled trade-off between robustness and accuracy. In *ICML*, 2019.

[26] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A Rotation and a Translation Suffice: Fooling CNNs with Simple Transformations. *arXiv preprint arXiv:1712.02779*, 2017.

[27] Eric Wong, Frank Schmidt, and Zico Kolter. Wasserstein adversarial examples via projected Sinkhorn iterations. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6808–6817. PMLR, 09–15 Jun 2019.

[28] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples. In *International Conference on Learning Representations*, 2018.

[29] Hossein Hosseini and Radha Poovendran. Semantic Adversarial Examples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1614–1619, 2018.

[30] Cassidy Laidlaw and Soheil Feizi. Functional Adversarial Attacks. In *NeurIPS*, 2019.

[31] Anand Bhattad, Min Jin Chong, Kaizhao Liang, Bo Li, and D. A. Forsyth. Unrestricted adversarial examples via semantic manipulation. In *International Conference on Learning Representations*, 2020.

[32] Yang Song, Rui Shu, Nate Kushman, and Stefano Ermon. Constructing Unrestricted Adversarial Examples with Generative Models. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 8322–8333. Curran Associates Inc., 2018.

[33] Xiaohui Zeng, Chenxi Liu, Yu-Siang Wang, Weichao Qiu, Lingxi Xie, Yu-Wing Tai, Chi Keung Tang, and Alan L. Yuille. Adversarial Attacks Beyond the Image Space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019. arXiv: 1711.07183.

[34] Matt Jordan, Naren Manoj, Surbhi Goel, and Alexandros G. Dimakis. Quantifying perceptual distortion of adversarial examples, 2019.

[35] Pratyush Maini, Eric Wong, and J. Zico Kolter. Adversarial robustness against the union of multiple perturbation models. In *International Conference on Machine Learning*, 2020.

[36] Florian Tramer and Dan Boneh. Adversarial Training and Robustness for Multiple Perturbations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 5866–5876. Curran Associates, Inc., 2019.

[37] Daniel Kang, Yi Sun, Dan Hendrycks, Tom Brown, and Jacob Steinhardt. Testing Robustness Against Unforeseen Adversaries. *arXiv preprint arXiv:1908.08016*, 2019.

[38] T. B. Brown, N. Carlini, C. Zhang, C. Olsson, P. Christiano, and I. Goodfellow. Unrestricted adversarial examples. *arXiv preprint arXiv:1809.08352*, 2018.

[39] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial Patch. *arXiv:1712.09665 [cs]*, May 2018. arXiv: 1712.09665.

[40] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004. Conference Name: IEEE Transactions on Image Processing.

[41] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018.

[42] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2016.

[43] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *International Conference on Learning Representations*, 2017.

[44] Mahmood Sharif, Lujo Bauer, and Michael K. Reiter. On the suitability of lp-norms for creating and preventing adversarial examples. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1686–16868, 2018.

[45] Isaac Dunn, Thomas F. Melham, and Daniel Kroening. Semantic adversarial perturbations using learnt representations. *ArXiv*, abs/2001.11055, 2020.

[46] Qiuling Xu, Guanhong Tao, Siyuan Cheng, Lin Tan, and Xiangyu Zhang. Towards Feature Space Adversarial Attack. *arXiv:2004.12385 [cs, eess]*, April 2020. arXiv: 2004.12385.

[47] Charles Jin and Martin Rinard. Manifold Regularization for Locally Stable Deep Neural Networks. March 2020.

[48] David Stutz, Matthias Hein, and Bernt Schiele. Confidence-calibrated adversarial training: Generalizing to unseen attacks. *Proceedings of the International Conference on Machine Learning ICML*, 2020.

[49] Z. Wang, E.P. Simoncelli, and A.C. Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, volume 2, pages 1398–1402 Vol.2, November 2003.

[50] Mehul P. Sampat, Zhou Wang, Shalini Gupta, Alan Conrad Bovik, and Mia K. Markey. Complex Wavelet Structural Similarity: A New Image Similarity Index. *IEEE Transactions on Image Processing*, 18(11):2385–2401, November 2009. Conference Name: IEEE Transactions on Image Processing.

[51] Rafał Mantiuk, Kil Joong Kim, Allan G. Rempel, and Wolfgang Heidrich. HDR-VDP-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions. *ACM Transactions on Graphics*, 30(4):40:1–40:14, July 2011.

[52] Zhou Wang and Eero P. Simoncelli. Maximum differentiation (MAD) competition: A methodology for comparing computational models of perceptual quantities. *Journal of Vision*, 8(12):8–8, September 2008. Publisher: The Association for Research in Vision and Ophthalmology.

[53] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *ECCV*, 2018.

[54] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60:84–90, 2012.

[56] J. Deng, Wei Dong, R. Socher, L. Li, K. Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[57] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009.

[58] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, 2020.

[59] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*, 2019.

[60] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[61] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, and Dimitris Tsipras. Robustness (python library), 2019.

[62] K Simonyan and A Zisserman. Very deep convolutional networks for large-scale image recognition. pages 1–14. Computational and Biological Learning Society, 2015.

[63] Yogesh Balaji, Tom Goldstein, and Judy Hoffman. Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets. *arXiv:1910.08051 [cs, stat]*, October 2019. arXiv: 1910.08051.

[64] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *ICLR*, 2018.

[65] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Workshop at International Conference on Learning Representations*, 2014.

[66] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *ICML*, 2017.

[67] Jiwei Li, Will Monroe, and Daniel Jurafsky. Understanding neural networks through representation erasure. *arXiv preprint arXiv: 1612.08220*, 2016.

[68] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. In *ICML Workshop on Visualization for Deep Learning*, 2017.

[69] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[70] Julius Adebayo, Justin Gilmer, Ian Goodfellow, and Been Kim. Local explanation methods for deep neural networks lack sensitivity to parameter values. 2018.

[71] Weili Nie, Yang Zhang, and Ankit B. Patel. A theoretical explanation for perplexing behaviors of backpropagation-based visualizations. In *International Conference on Machine Learning*, 2018.

[72] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2016.

[73] Emmanuel J Candes and Terence Tao. Decoding by linear programming. *IEEE transactions on information theory*, 2005.

[74] David L. Donoho. Compressed sensing. In *IEEE Transactions on Information Theory*, 2006.

[75] Robert Tibshirani. Regression shrinkage and selection via the lasso. In *Journal of the Royal Statistical Society*, 1996.

[76] Neal Parikh and Stephen Boyd. Proximal algorithms. *Found. Trends Optim.*, 1(3):127–239, January 2014.

[77] Barak A. Pearlmutter. Fast exact multiplication by the hessian. In *Neural Computation*, 1994.

[78] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[79] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C. Courville, and Yoshua Bengio. Maxout networks. In *International Conference on Machine Learning*, 2013.

[80] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. 2014. cite arxiv:1409.0575Comment: 43 pages, 16 figures. v3 includes additional comparisons with PASCAL VOC (per-category comparisons in Table 3, distribution of localization difficulty in Fig 16), a list of queries used for obtaining object detection images (Appendix C), and some additional references.

[81] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[82] Zhewei Yao, Amir Gholami, Qi Lei, Kurt Keutzer, and Michael W Mahoney. Hessian-based analysis of large batch training and robustness to adversaries. *Neural Information Processing Systems*, 2018.

[83] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[84] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2009.

[85] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least Angle Regression. *The Annals of Statistics*, June 2004.

[86] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3145–3153. PMLR, 06–11 Aug 2017.

[87] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, Wojciech Samek, and Óscar Déniz Suárez. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. In *PloS one*, 2015.

[88] Pieter-Jan Kindermans, Kristof Schütt, Klaus-Robert Müller, and Sven Dähne. Investigating the influence of noise and distractors on the interpretation of neural networks. In *NIPS Workshop on Interpretable Machine Learning in Complex Systems*, 2016.

[89] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[90] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T. Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un)reliability of saliency methods. *Neural Information Processing Systems*, 2018.

[91] Garvesh Raskutti, Martin J Wainwright, and Bin Yu. Restricted eigenvalue properties for correlated gaussian designs. *Journal of Machine Learning Research*, 11(Aug):2241–2259, 2010.

[92] Peter J Bickel, Ya'acov Ritov, Alexandre B Tsybakov, et al. Simultaneous analysis of lasso and dantzig selector. *The Annals of Statistics*, 2009.

[93] Emmanuel Candes, Terence Tao, et al. The dantzig selector: Statistical estimation when p is much larger than n. *The Annals of Statistics*, 2007.

[94] Vasu Singla, Songwei Ge, Basri Ronen, and David Jacobs. Shift invariance can reduce adversarial robustness. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 1858–1871. Curran Associates, Inc., 2021.

[95] Cassidy Laidlaw, Sahil Singla, and Soheil Feizi. Perceptual adversarial robustness: Defense against unseen threat models. In *International Conference on Learning Representations*, 2021.

[96] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597, Los Alamitos, CA, USA, may 2016. IEEE Computer Society.

[97] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *International Conference on Learning Representations*, 2018.

[98] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*, 2018.

[99] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 5019–5031, 2018.

[100] Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In *International Conference on Machine Learning*, pages 8093–8104. PMLR, 2020.

[101] Farzan Farnia, Jesse Zhang, and David Tse. Generalizable adversarial training via spectral normalization. In *International Conference on Learning Representations*, 2019.

[102] Tianlong Chen, Sijia Liu, Shiyu Chang, Yu Cheng, Lisa Amini, and Zhangyang Wang. Adversarial robustness: From self-supervised pre-training to fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 699–708, 2020.

[103] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[104] Cihang Xie, Mingxing Tan, Boqing Gong, Alan Yuille, and Quoc V Le. Smooth adversarial training. *arXiv preprint arXiv:2006.14536*, 2020.

[105] Sven Gowal, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. Uncovering the limits of adversarial training against norm-bounded adversarial examples. *arXiv preprint arXiv:2010.03593*, 2020.

[106] Tianyu Pang, Xiao Yang, Yinpeng Dong, Hang Su, and Jun Zhu. Bag of tricks for adversarial training. In *International Conference on Learning Representations*, 2021.

[107] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[108] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations*, 2020.

[109] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 501–509, 2019.

[110] Tianyu Pang, Xiao Yang, Yinpeng Dong, Taufik Xu, Jun Zhu, and Hang Su. Boosting adversarial training with hypersphere embedding. In *NeurIPS*, 2020.

[111] Jingfeng Zhang, Xilie Xu, Bo Han, Gang Niu, Lizhen Cui, Masashi Sugiyama, and Mohan Kankanhalli. Attacks which do not kill training make adversarial learning stronger. In *International Conference on Machine Learning*, pages 11278–11287. PMLR, 2020.

[112] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[113] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[114] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*, 2020.

[115] Ali Shafahi, Mahyar Najibi, Zheng Xu, John Dickerson, Larry S Davis, and Tom Goldstein. Universal adversarial training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5636–5643, 2020.

[116] Philipp Benz, Chaoning Zhang, Adil Karjauv, and In So Kweon. Universal adversarial training with class-wise perturbations. In *2021 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2021.

[117] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. In *International Conference on Learning Representations*, 2018.

[118] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. In *International Conference on Learning Representations*, 2018.

[119] Guneet S. Dhillon, Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, and Animashree Anandkumar. Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations*, 2018.

[120] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 369–385, 2018.

[121] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *International Conference on Learning Representations*, 2017.

[122] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[123] Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*, 2018.

[124] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1633–1645. Curran Associates, Inc., 2020.

[125] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pages 3–14, New York, NY, USA, 2017. ACM.

[126] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2019.

[127] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *CoRR*, abs/1706.07351, 2017.

[128] Matteo Fischetti and Jason Jo. Deep neural networks as 0-1 mixed integer linear programs: A feasibility study. *CoRR*, abs/1712.06174, 2017.

[129] Rudy Bunel, Ilker Turkaslan, Philip H.S. Torr, Pushmeet Kohli, and M. Pawan Kumar. A unified view of piecewise linear neural network verification. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 4795–4804, Red Hook, NY, USA, 2018. Curran Associates Inc.

[130] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *CAV*, 2017.

[131] Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis*, 2017.

[132] Xiaowei Huang, Marta Z. Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. *ArXiv*, abs/1610.06940, 2016.

[133] Cem Anil, James Lucas, and Roger B. Grosse. Sorting out lipschitz function approximation. In *ICML*, 2018.

[134] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 110(2):393–416, 2021.

[135] Jeremy M. Cohen, Elan Rosenfeld, and J. Zico Kolter. Certified adversarial robustness via randomized smoothing. In *ICML*, 2019.

[136] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and S. K. K. Jana. Certified robustness to adversarial examples with differential privacy. In *IEEE S&P 2019*, 2018.

[137] Alexander Levine, Sahil Singla, and Soheil Feizi. Certifiably robust interpretation in deep learning, 2019.

[138] Alexander Levine and Soheil Feizi. (de)randomized smoothing for certifiable defense against patch attacks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6465–6475. Curran Associates, Inc., 2020.

[139] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2018.

[140] Matthew Mirman, Timon Gehr, and Martin T. Vechev. Differentiable abstract interpretation for provably robust neural networks. In *ICML*, 2018.

[141] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019.

[142] Sahil Singla and Soheil Feizi. Second-order provable defenses against adversarial attacks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8981–8991. PMLR, 13–18 Jul 2020.

[143] Sven Gowal, Krishnamurthy (Dj) Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. Scalable verified training for provably robust image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[144] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[145] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5286–5295, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[146] Eric Wong, Frank R. Schmidt, Jan Hendrik Metzen, and J. Zico Kolter. Scaling provable adversarial defenses. In *NeurIPS*, 2018.

[147] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[148] Tianlong Chen, Zhenyu Zhang, Sijia Liu, Shiyu Chang, and Zhangyang Wang. Robust overfitting may be mitigated by properly learned smoothening. In *International Conference on Learning Representations*, 2021.

[149] Li Yuan, Francis EH Tay, Guilin Li, Tao Wang, and Jiashi Feng. Revisiting knowledge distillation via label smoothing regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3903–3911, 2020.

[150] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In Amir Globerson and Ricardo Silva, editors, *UAI*, pages 876–885. AUAI Press, 2018.

[151] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. In *NeurIPS*, 2020.

[152] Saehyung Lee, Hyungyu Lee, and Sungroh Yoon. Adversarial vertex mixup: Toward better adversarially robust generalization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[153] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. Exploring generalization in deep learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5949–5958, Red Hook, NY, USA, 2017. Curran Associates Inc.

[154] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1729–1739, Red Hook, NY, USA, 2017. Curran Associates Inc.

[155] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[156] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[157] Swalpa Kumar Roy, Suvojit Manna, Shiv Ram Dubey, and Bidyut Baran Chaudhuri. Lisht: Non-parametric linearly scaled hyperbolic tangent activation function for neural networks. *arXiv preprint arXiv:1901.05894*, 2019.

[158] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[159] Diganta Misra. Mish: A self regularized non-monotonic activation function. In *British Machine Vision Conference*, 2020.

[160] Seyed Mohsen Moosavi Dezfooli, Alhussein Fawzi, Jonathan Uesato, and Pascal Frossard. Robustness via curvature regularization, and vice versa. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[161] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016.

[162] Garrett Bingham and Risto Miikkulainen. Discovering parametric activation functions. *CoRR*, abs/2006.03179, 2020.

[163] Preetum Nakkiran, Prayaag Venkat, Sham M. Kakade, and Tengyu Ma. Optimal regularization can mitigate double descent. In *International Conference on Learning Representations*, 2021.

[164] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. *CoRR*, abs/1602.02697, 2016.

[165] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 4939–4948. Curran Associates, Inc., 2018.

[166] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane S. Boning, Inderjit S. Dhillon, and Luca Daniel. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning (ICML)*, July 2018.

[167] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations*, 2020.

[168] Chongli Qin, James Martens, Sven Gowal, Dilip Krishnan, Krishnamurthy Dvijotham, Alhussein Fawzi, Soham De, Robert Stanforth, and Pushmeet Kohli. Adversarial robustness through local linearization. In *NeurIPS*, 2019.

[169] Jonathan Peck, Joris Roels, Bart Goossens, and Yvan Saeys. Lower bounds on the robustness to adversarial perturbations. In *NIPS*, 2017.

[170] Huan Zhang, Pengchuan Zhang, and Cho-Jui Hsieh. Recurjac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications. In *AAAI*, 2018.

[171] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2263–2273. Curran Associates, Inc., 2017.

[172] Jonathan Uesato, Brendan O'Donoghue, Pushmeet Kohli, and Aäron van den Oord. Adversarial risk and the dangers of evaluating against weak attacks. In *ICML*, 2018.

[173] Anish Athalye and Nicholas Carlini. On the robustness of the cvpr 2018 white-box adversarial example defenses. *ArXiv*, abs/1804.03286, 2018.

[174] Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. *ArXiv*, abs/1705.01320, 2017.

[175] Nicholas Carlini, Guy Katz, Clark E. Barrett, and David L. Dill. Provably minimally-distorted adversarial examples. 2017.

[176] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *ATVA*, 2017.

[177] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *ArXiv*, abs/1706.07351, 2017.

[178] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NFM*, 2018.

[179] Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23:296–309, 2018.

[180] Shiqi Wang, Yizheng Chen, Ahmed Abdou, and S. K. K. Jana. Mixtrain: Scalable training of verifiably robust neural networks. 2018.

[181] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and S. K. K. Jana. Efficient formal safety analysis of neural networks. In *NeurIPS*, 2018.

[182] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *NeurIPS*, 2018.

[183] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018.

[184] Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. Training verified learners with learned verifiers. *ArXiv*, abs/1805.10265, 2018.

[185] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A. Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *UAI*, 2018.

[186] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. In *International Conference on Artificial Intelligence and Statistics*, 2019.

[187] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Puschel, and Martin T. Vechev. Fast and effective robustness certification. In *NeurIPS*, 2018.

[188] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2018.

[189] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In *European Conference on Computer Vision*, 2018.

[190] Xiaoyu Cao and Neil Zhenqiang Gong. Mitigating evasion attacks to deep neural networks via region-based classification. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, ACSAC 2017, page 278–287, New York, NY, USA, 2017. Association for Computing Machinery.

[191] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Certified adversarial robustness with additive noise. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlche Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 9464–9474. Curran Associates, Inc., 2019.

[192] Hadi Salman, Jerry Li, Ilya Razenshteyn, Pengchuan Zhang, Huan Zhang, Sebastien Bubeck, and Greg Yang. Provably robust deep learning via adversarially trained smoothed classifiers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alche Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 11292–11303. Curran Associates, Inc., 2019.

[193] Sahil Singla, Eric Wallace, Shi Feng, and Soheil Feizi. Understanding impacts of high-order loss approximations and features in deep learning interpretation. In *ICML*, 2019.

[194] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George J Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[195] Kevin Scaman and Aladin Virmaux. Lipschitz regularity of deep neural networks: Analysis and efficient estimation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 3839–3848, Red Hook, NY, USA, 2018. Curran Associates Inc.

[196] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

[197] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[198] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[199] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 4939–4948. Curran Associates, Inc., 2018.

[200] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. 1991.

[201] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

[202] Gunter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[203] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of CNNs: How to train 10,000-layer vanilla convolutional neural networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5393–5402, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[204] Peter L. Bartlett, Dylan J. Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6241–6250, USA, 2017. Curran Associates Inc.

[205] Philip M. Long and Hanie Sedghi. Generalization bounds for deep convolutional neural networks. In *International Conference on Learning Representations*, 2020.

[206] Martin Arjovsky, Soumith Chintala, and Leon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[207] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[208] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders. In *International Conference on Learning Representations*, 2018.

[209] Emiel Hoogeboom, Victor Garcia Satorras, Jakub M. Tomczak, and Max Welling. The convolution exponential and generalized sylvester flows. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[210] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 854–863. JMLR.org, 2017.

[211] H. Drucker and Y. Le Cun. Improving generalization performance using double backpropagation. *Trans. Neur. Netw.*, 3(6):991–997, November 1992.

[212] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. *ArXiv*, abs/1705.10941, 2017.

[213] Takeru Miyato, Shin ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41:1979–1993, 2017.

[214] Hanie Sedghi, Vineet Gupta, and Philip M. Long. The singular values of convolutional layers. In *International Conference on Learning Representations*, 2019.

[215] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[216] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[217] Ernest Ryu, Jialin Liu, Sicheng Wang, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. Plug-and-play methods provably converge with properly trained denoisers. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5546–5557, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[218] Adel Bibi, Bernard Ghanem, Vladlen Koltun, and Rene Ranftl. Deep layers as stochastic solvers. In *International Conference on Learning Representations*, 2019.

[219] Haoming Jiang, Zhehui Chen, Minshuo Chen, Feng Liu, Dingding Wang, and Tuo Zhao. On computation and generalization of generative adversarial networks under spectrum control. In *International Conference on Learning Representations*, 2019.

[220] Akhilan Boopathy, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In *AAAI*, Jan 2019.

[221] Robert M. Gray. Toeplitz and circulant matrices: A review. *Commun. Inf. Theory*, 2(3):155–239, August 2005.

[222] Z. Zhu and M. B. Wakin. On the asymptotic equivalence of circulant and toeplitz matrices. *IEEE Transactions on Information Theory*, 63(5):2975–2992, May 2017.

[223] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *NIPS*, 1991.

[224] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[225] Yusuke Tsuzuku, I. Sato, and Masashi Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *NeurIPS*, 2018.

[226] Charles Dugas, Yoshua Bengio, François Belisle, Claude Nadeau, and Rene Garcia. Incorporating second-order functional knowledge for better option pricing. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, page 451–457, Cambridge, MA, USA, 2000. MIT Press.

[227] Peter L. Bartlett, Dylan J. Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6241–6250, USA, 2017. Curran Associates Inc.

[228] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann N. Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 854–863. PMLR, 2017.

[229] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *ICLR*, 2018.

[230] Cedric Villani. Optimal transport, old and new, 2008.

[231] Gabriel Peyré and Marco Cuturi. Computational optimal transport, 2018.

[232] Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger Grosse, and Jörn-Henrik Jacobsen. Preventing gradient attenuation in lipschitz constrained convolutional networks. *Conference on Neural Information Processing Systems*, 2019.

[233] Sahil Singla and Soheil Feizi. Fantastic four: Differentiable and efficient bounds on singular values of convolution layers. In *International Conference on Learning Representations*, 2021.

[234] Haifeng Qian and Mark N. Wegman. L2-nonexpansive neural networks. In *International Conference on Learning Representations*, 2019.

[235] A. Edelman, T. A. Arias, and S. Smith. The geometry of algorithms with orthogonality constraints. *SIAM J. Matrix Anal. Appl.*, 20:303–353, 1998.

[236] Huan Zhang, Pengchuan Zhang, and Cho-Jui Hsieh. Recurjac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications. In *AAAI Conference on Artificial Intelligence (AAAI), arXiv preprint arXiv:1810.11783*, dec 2019.

[237] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems (NIPS), arXiv preprint arXiv:1811.00866*, dec 2018.

[238] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. *AISTATS 2019*, 2019.

[239] Xuanqing Liu, Minhao Cheng, Huan Zhang, and C. Hsieh. Towards robust neural networks via random self-ensemble. In *ECCV*, 2018.

[240] Aounon Kumar, Alexander Levine, Tom Goldstein, and Soheil Feizi. Curse of dimensionality on randomized smoothing for certifiable robustness. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5458–5467. PMLR, 13–18 Jul 2020.

[241] Jörn-Henrik Jacobsen, Arnold W.M. Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks. In *International Conference on Learning Representations*, 2018.

[242] Asher Trockman and J Zico Kolter. Orthogonalizing convolutional layers with the cayley transform. In *International Conference on Learning Representations*, 2021.

[243] Sahil Singla and Soheil Feizi. Skew orthogonal convolutions. In *ICML*, 2021.

[244] Bohang Zhang, Tianle Cai, Zhou Lu, Di He, and Liwei Wang. Towards certifying linfinity robustness using neural networks with linfinity-dist neurons. In *ICML*, 2021.

[245] Bohang Zhang, Du Jiang, Di He, and Liwei Wang. Boosting the certified robustness of l-infinity distance nets. In *International Conference on Learning Representations*, 2022.

[246] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[247] Yujia Huang, Huan Zhang, Yuanyuan Shi, J Zico Kolter, and Anima Anandkumar. Training certifiably robust neural networks with efficient local lipschitz bounds. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[248] Aounon Kumar, Alexander Levine, Soheil Feizi, and Tom Goldstein. Certifying confidence via randomized smoothing. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5165–5177. Curran Associates, Inc., 2020.

[249] Alexander Levine and Soheil Feizi. Improved, deterministic smoothing for l1 certified robustness. In *ICML*, 2021.

[250] Tan Yu, Jun Li, YUNFENG CAI, and Ping Li. Constructing orthogonal convolutions in an explicit manner. In *International Conference on Learning Representations*, 2022.

[251] Bobak Kiani, Randall Balestriero, Yann Lecun, and Seth Lloyd. projunn: efficient method for training deep networks with unitary matrices, 2022.

[252] Sahil Singla, Surbhi Singla, and Soheil Feizi. Improved deterministic l2 robustness on CIFAR-10 and CIFAR-100. In *International Conference on Learning Representations*, 2022.

[253] Gagandeep Singh, Markus Püschel, and Martin Vechev. Fast polyhedra abstract domain. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, page 46–59, New York, NY, USA, 2017. Association for Computing Machinery.

[254] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[255] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the single neuron convex barrier for neural network certification. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[256] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Robustness certification with refinement. In *International Conference on Learning Representations*, 2019.

[257] Krishnamurthy Dvijotham*, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *Proceedings of the Thirty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-18)*, Corvallis, Oregon, 2018. AUAI Press.

[258] Sven Gowal, Krishnamurthy Dvijotham*, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy A. Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. In *ICCV*, volume 9, pages –, 2019.

[259] Krishnamurthy (Dj) Dvijotham*, Aditi Raghunathan, Jonathan Uesato, Sumanth Dathathri, Alexey Kurakin, Ian Goodfellow, Pushmeet Kohli, Jacob Steinhardt, and Percy

Liang. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. In *Advances in Neural Information Processing Systems*, pages –, 2020.

[260] Jingyue Lu and M. Pawan Kumar. Neural network branching for neural network verification. In *International Conference on Learning Representations*, 2020.

[261] Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. Branch and bound for piecewise linear neural network verification. *J. Mach. Learn. Res.*, 21:42:1–42:39, 2020.

[262] Klas Leino, Zifan Wang, and Matt Fredrikson. Globally-robust neural networks. In *International Conference on Machine Learning (ICML)*, 2021.

[263] Klas Leino and Matt Fredrikson. Relaxing local robustness. In *Neural Information Processing Systems (NIPS)*, 2021.

[264] Christoph Müller, François Serre, Gagandeep Singh, Markus Püschel, and Martin Vechev. Scaling polyhedral neural network verification on gpus. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 733–746, 2021.

[265] Harkirat Singh, M. Pawan Kumar, Philip Torr, and Krishnamurthy Dj Dvijotham. Overcoming the convex barrier for simplex inputs. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[266] Alessandro De Palma, Harkirat Behl, Rudy R Bunel, Philip Torr, and M. Pawan Kumar. Scaling the convex barrier with active sets. In *International Conference on Learning Representations*, 2021.

[267] Hadi Salman, Mingjie Sun, Greg Yang, Ashish Kapoor, and J. Zico Kolter. Denoised smoothing: A provable defense for pretrained classifiers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[268] Hadi Salman, Saachi Jain, Eric Wong, and Aleksander Madry. Certified patch robustness via smoothed vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15137–15147, June 2022.

[269] Jiahao Su, Wonmin Byeon, and Furong Huang. Scaling-up diverse orthogonal convolutional networks with a paraunitary framework. In *ICML*, 2022.

[270] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989.

[271] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifiable distributional robustness with principled adversarial training. In *International Conference on Learning Representations*, 2018.

[272] Ayanna Howard, Cha Zhang, and Eric Horvitz. Addressing bias in machine learning algorithms: A pilot study on emotion recognition for intelligent systems. In *IEEE Workshop on Advanced Robotics and its Social Impacts*, 2017.

[273] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *FAT*, 2018.

[274] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Brandon Tran, and Aleksander Madry. Learning perceptually-aligned representations via adversarial robustness. *ArXiv*, abs/1906.00945, 2019.

[275] Horace Barlow. Single units and sensation: A neuron doctrine for perceptual psychology. *Perception*, 1972.

[276] W. S. Lee, D. L. Grosh, F. A. Tillman, and C. H. Lie. Fault tree analysis, methods, and applications: A review. *IEEE Transactions on Reliability*, R-34(3):194–203, Aug 1985.

[277] Matthew D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

[278] Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. In *ICML Deep Learning Workshop*, 2016.

[279] Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[280] Aravindh Mahendran and A. Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 120:233–255, 2016.

[281] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. In *ICML Workshop on Visualization for Deep Learning*, 2016.

[282] B. Zhou, A. Khosla, Lapedriza. A., A. Oliva, and A. Torralba. Learning Deep Features for Discriminative Localization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[283] R. R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, D. Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128:336–359, 2019.

[284] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. https://distill.pub/2018/building-blocks.

[285] Pascal Sturmfels, Scott Lundberg, and Su-In Lee. Visualizing the impact of feature attribution baselines. *Distill*, 2020. https://distill.pub/2020/attribution-baselines.

[286] Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. Activation atlas. *Distill*, 2019. https://distill.pub/2019/activation-atlas.

[287] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Agata Lapedriza, Bolei Zhou, and Antonio Torralba. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 117(48):30071–30078, 2020.

[288] Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba. Interpreting deep visual representations via network dissection. *IEEE transactions on pattern analysis and machine intelligence*, 41(9):2131–2145, 2018.

[289] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[290] Zeyuan Allen-Zhu and Yuanzhi Li. Feature purification: How adversarial training performs robust deep learning. *arXiv preprint arXiv:2005.10190*, 2020.

[291] Tejas D. Kulkarni, Will Whitney, Pushmeet Kohli, and Joshua B. Tenenbaum. Deep convolutional inverse graphics network. In *NIPS*, 2015.

[292] I. Higgins, Loïc Matthey, A. Pal, C. Burges, Xavier Glorot, M. Botvinick, S. Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

[293] Christopher P. Burges, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-vae. In *NeurIPS*, 2017.

[294] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. ICML, 2018.

[295] Ricky T. Q. Chen, Xuechen Li, Roger Grosse, and David Duvenaud. Isolating sources of disentanglement in variational autoencoders. In *NeurIPS*, 2018.

[296] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *ICML*, 2019.

[297] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[298] Julius Adebayo, J. Gilmer, M. Muelly, Ian J. Goodfellow, M. Hardt, and Been Kim. Sanity checks for saliency maps. In *NeurIPS*, 2018.

[299] Yash Goyal, Ziyan Wu, Jan Ernst, Dhruv Batra, Devi Parikh, and Stefan Lee. Counterfactual visual explanations. In *ICML*, 2019.

[300] Yash Goyal, Amir Feder, U. Shalit, and Been Kim. Explaining classifiers with causal concept effect (cace). *ArXiv*, abs/1907.07165, 2019.

[301] Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. Explaining image classifiers by counterfactual generation. In *International Conference on Learning Representations*, 2019.

[302] Matthew O'Shaughnessy, Gregory Canal, Marissa Connor, Mark Davenport, and Christopher Rozell. Generative causal explanations of black-box classifiers. In *NeurIPS*, 2019.

[303] Sahil Verma, John Dickerson, and Keegan Hines. Counterfactual explanations for machine learning: A review, 2020.

[304] Chih-Kuan Yeh, Cheng-Yu Hsieh, Arun Sai Suggala, David I. Inouye, and Pradeep D. Ravikumar. On the (in)fidelity and sensitivity of explanations. In *NeurIPS*, 2019.

[305] Besmira Nushi, Ece Kamar, and Eric Horvitz. Towards accountable AI: hybrid human-machine analyses for characterizing system failure. In Yiling Chen and Gabriella Kazai, editors, *Proceedings of the Sixth AAAI Conference on Human Computation and Crowdsourcing, HCOMP*, pages 126–135. AAAI Press, 2018.

[306] Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David S Ebert. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE transactions on visualization and computer graphics*, 25(1):364–373, 2018.

[307] Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. Slice finder: Automated data slicing for model validation. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1550–1553. IEEE, 2019.

[308] Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel S Weld. Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 747–763, 2019.

[309] George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

[310] Klaus Oberauer and Laura Hein. Attention to information in working memory. *Current Directions in Psychological Science*, 21(3):164–169, 2012.

[311] Nelson Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and brain sciences*, 24(1):87–114, 2001.

[312] Fereshte Khani and Percy Liang. Removing spurious features can hurt accuracy and affect groups disproportionately. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 196–205, New York, NY, USA, 2021. Association for Computing Machinery.

[313] Alessandro Achille and Stefano Soatto. Emergence of invariance and disentanglement in deep representations. *2018 Information Theory and Applications Workshop (ITA)*, pages 1–9, 2017.

[314] David Alvarez-Melis and Tommi S. Jaakkola. Towards robust interpretability with self-explaining neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 7786–7795, Red Hook, NY, USA, 2018. Curran Associates Inc.

[315] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization, 2020.

[316] Alceu Bissoto, E. Valle, and S. Avila. Debiasing skin lesion datasets and models? not so fast. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3192–3201, 2020.

[317] Pim de Haan, Dinesh Jayaraman, and Sergey Levine. Causal confusion in imitation learning. *CoRR*, abs/1905.11979, 2019.

[318] Sahil Singla, Besmira Nushi, Shital Shah, Ece Kamar, and Eric Horvitz. Understanding failures of deep networks via robust feature extraction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[319] Eric Wong, Shibani Santurkar, and Aleksander Madry. Leveraging sparse linear layers for debuggable deep networks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11205–11216. PMLR, 18–24 Jul 2021.

[320] Aya Abdelsalam Ismail, Mohamed K. Gunady, Luiz Pessoa, Héctor Corrada Bravo, and Soheil Feizi. Input-cell attention reduces vanishing saliency of recurrent neural networks. In *NeurIPS*, 2019.

[321] Aya Abdelsalam Ismail, Mohamed K. Gunady, Héctor Corrada Bravo, and Soheil Feizi. Benchmarking deep learning interpretability in time series predictions. In *NeurIPS*, 2020.

[322] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery.

[323] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, 2015.

[324] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5188–5196, 2015.

[325] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1885–1894. JMLR.org, 2017.

[326] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of nlp models with checklist. In *Association for Computational Linguistics (ACL)*, 2020.

[327] Kai Yuanqing Xiao, Logan Engstrom, Andrew Ilyas, and Aleksander Madry. Noise or signal: The role of image backgrounds in object recognition. In *International Conference on Learning Representations*, 2021.

[328] Guillaume Leclerc, Hadi Salman, Andrew Ilyas, Sai Vemprala, Logan Engstrom, Vibhav Vineet, Kai Xiao, Pengchuan Zhang, Shibani Santurkar, Greg Yang, Ashish Kapoor, and Aleksander Madry. 3db: A framework for debugging computer vision models. In *Arxiv preprint arXiv:2106.03805*, 2021.

[329] George A. Miller. Wordnet: A lexical database for english. *COMMUNICATIONS OF THE ACM*, 38:39–41, 1995.

[330] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Andrew Ilyas, and Aleksander Madry. From imagenet to image classification: Contextualizing progress on benchmarks. In *International Conference on Machine Learning*, 2020.

[331] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019.

[332] Shiori Sagawa*, Pang Wei Koh*, Tatsunori B. Hashimoto, and Percy Liang. Distributionally robust neural networks. In *International Conference on Learning Representations*, 2020.

[333] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[334] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton Earnshaw, Imran Haque, Sara M Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. Wilds: A benchmark of in-the-wild distribution shifts. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5637–5664. PMLR, 18–24 Jul 2021.

[335] Sahil Singla and Soheil Feizi. Salient imagenet: How to discover spurious features in deep learning? In *International Conference on Learning Representations*, 2022.

[336] Sahil Singla, Mazda Moayeri, and Soheil Feizi. Core risk minimization using salient imagenet, 2022.

[337] Mazda Moayeri, Phillip Pope, Yogesh Balaji, and Soheil Feizi. A comprehensive study of image classification model sensitivity to foregrounds, backgrounds, and visual attributes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[338] Adam Coates, A. Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011.

[339] Anonymous. Last layer re-training is sufficient for robustness to spurious correlations. In *Submitted to The Eleventh International Conference on Learning Representations*, 2023. under review.

[340] Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9573–9585. Curran Associates, Inc., 2020.

[341] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2020.

[342] Weihua Hu, Gang Niu, Issei Sato, and Masashi Sugiyama. Does distributionally robust supervised learning give robust classifiers? In *ICML*, 2018.

[343] Evan Zheran Liu, Behzad Haghgoo, Annie S. Chen, Aditi Raghunathan, Pang Wei Koh, Shiori Sagawa, Percy Liang, and Chelsea Finn. Just train twice: Improving group robustness without training group information. In *ICML*, 2021.

[344] Alexander Robey, George J. Pappas, and Hamed Hassani. Model-based domain generalization. In *NeurIPS*, 2021.

[345] Mengye Ren, Wenyuan Zeng, Binh Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *ICML*, 2018.

[346] J. Peters, Peter Buhlmann, and Nicolai Meinshausen. Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 78, 2015.

[347] Nanyang Ye, Kaican Li, Haoyue Bai, Runpeng Yu, Lanqing Hong, Fengwei Zhou, Zhenguo Li, and Jun Zhu. Ood-bench: Quantifying and understanding two dimensions of out-of-distribution generalization. 2021.

[348] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Lixuan Zhu, Samyak Parajuli, Mike Guo, Dawn Xiaodong Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8320–8329, 2021.

[349] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness, 2018.

[350] Kai Yuanqing Xiao, Logan Engstrom, Andrew Ilyas, and Aleksander Madry. Noise or signal: The role of image backgrounds in object recognition. In *International Conference on Learning Representations*, 2021.

[351] Yue He, Zheyan Shen, and Peng Cui. Towards non-iid image classification: A dataset and baselines. *Pattern Recognition*, 110:107383, 2021.

[352] Priyatham Kattakinda and Soheil Feizi. FOCUS: Familiar objects in common and uncommon settings. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 10825–10847. PMLR, 17–23 Jul 2022.

[353] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 2009.

[354] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers and distillation through attention. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR, 18–24 Jul 2021.

[355] Andrei Barbu, David Mayo, Julian Alverio, William Luo, Christopher Wang, Dan Gutfreund, Joshua B. Tenenbaum, and Boris Katz. Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. In *NeurIPS*, 2019.

[356] Judy Borowski, Roland Simon Zimmermann, Judith Schepers, Robert Geirhos, Thomas S. A. Wallis, Matthias Bethge, and Wieland Brendel. Exemplary natural images explain {cnn} activations better than state-of-the-art feature visualization. In *International Conference on Learning Representations*, 2021.

[357] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. *CVPR*, 2021.

[358] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9992–10002, 2021.

[359] Stéphane d'Ascoli, Hugo Touvron, Matthew L. Leavitt, Ari S. Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In *ICML*, 2021.

[360] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[361] Timnit Gebru, Jamie H. Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna M. Wallach, Hal Daumé, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64:86 – 92, 2021.

[362] Rakshith Shetty, Bernt Schiele, and Mario Fritz. Not using the car to see the sidewalk — quantifying and controlling the effects of context in classification and segmentation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8210–8218, 2019.

[363] Amir Rosenfeld, Richard S. Zemel, and John K. Tsotsos. The elephant in the room. *ArXiv*, abs/1808.03305, 2018.

[364] Michael A. Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, and Anh M Nguyen. Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4840–4849, 2019.

[365] Alexandra Chouldechova. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big data*, 5 2:153–163, 2017.

[366] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[367] Mazda Moayeri, Kiarash Banihashem, and Soheil Feizi. Explicit tradeoffs between adversarial and natural distributional robustness. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[368] Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. Learning robust global representations by penalizing local predictive power. In *Advances in Neural Information Processing Systems*, pages 10506–10518, 2019.

[369] Josip Djolonga, Jessica Yung, Michael Tschannen, Rob Romijnders, Lucas Beyer, Alexander Kolesnikov, Joan Puigcerver, Matthias Minderer, Alexander D'Amour, Dan I. Moldovan, Sylvan Gelly, Neil Houlsby, Xiaohua Zhai, and Mario Lucic. On robustness and transferability of convolutional neural networks. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16453–16463, 2021.

[370] Mazda Moayeri, Sahil Singla, and Soheil Feizi. Hard imagenet: Segmentations for objects with strong spurious cues. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[371] Chaofan Chen, Oscar Li, Alina Jade Barnett, Jonathan Su, and Cynthia Rudin. This looks like that: deep learning for interpretable image recognition. *ArXiv*, abs/1806.10574, 2019.

[372] Jonathan Crabbe, Zhaozhi Qian, Fergus Imrie, and Mihaela van der Schaar. Explaining latent representations with a corpus of examples. In *NeurIPS*, 2021.

[373] Asma Ghandeharioun, Been Kim, Chun-Liang Li, Brendan Jou, Brian Eoff, and Rosalind Picard. DISSECT: Disentangled simultaneous explanations via concept traversals. In *International Conference on Learning Representations*, 2022.

[374] Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data. In *ICML*, 2022.

[375] Samyadeep Basu, Phil Pope, and Soheil Feizi. Influence functions in deep learning are fragile. In *International Conference on Learning Representations*, 2021.

[376] Jingzhao Zhang, Aditya Krishna Menon, Andreas Veit, Srinadh Bhojanapalli, Sanjiv Kumar, and Suvrit Sra. Coping with label shift via distributionally robust optimisation. In *International Conference on Learning Representations*, 2021.

[377] Junhyun Nam, Jaehyung Kim, Jaeho Lee, and Jinwoo Shin. Spread spurious attribute: Improving worst-group accuracy with spurious attribute estimation. In *International Conference on Learning Representations*, 2022.

[378] Michael Zhang, Nimit Sharad Sohoni, Hongyang R. Zhang, Chelsea Finn, and Christopher R'e. Correct-n-contrast: A contrastive approach for improving robustness to spurious correlations. In *ICML*, 2022.

[379] Evan Hernandez, Sarah Schwettmann, David Bau, Teona Bagashvili, Antonio Torralba, and Jacob Andreas. Natural language descriptions of deep features. In *International Conference on Learning Representations*, 2022.

[380] John Miller, Rohan Taori, Aditi Raghunathan, Shiori Sagawa, Pang Wei Koh, Vaishaal Shankar, Percy Liang, Yair Carmon, and Ludwig Schmidt. Accuracy on the line: on the strong correlation between out-of-distribution and in-distribution generalization. *ArXiv*, abs/2107.04649, 2021.

[381] Tatsunori B. Hashimoto, Megha Srivastava, Hongseok Namkoong, and Percy Liang. Fairness without demographics in repeated loss minimization. In *ICML*, 2018.

[382] Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. Measuring robustness to natural distribution shifts in image classification. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18583–18599. Curran Associates, Inc., 2020.

[383] Hadi Salman, Andrew Ilyas, Logan Engstrom, Ashish Kapoor, and Aleksander Madry. Do adversarially robust imagenet models transfer better? In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3533–3545. Curran Associates, Inc., 2020.

[384] Muzammal Naseer, Kanchana Ranasinghe, Salman Hameed Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Intriguing properties of vision transformers. In *Neural Information Processing Systems*, 2021.

[385] Yao Qin, Chiyuan Zhang, Ting Chen, Balaji Lakshminarayanan, Alex Beutel, and Xuezhi Wang. Understanding and improving robustness of visiontransformers through patch-based negativeaugmentation. In *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*, 2021.

[386] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[387] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[388] Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9620–9629, 2021.

[389] Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7959–7971, June 2022.

[390] Shibani Santurkar, Dimitris Tsipras, Mahalaxmi Elango, David Bau, Antonio Torralba, and Aleksander Madry. Editing a classifier by rewriting its prediction rules. *Advances in Neural Information Processing Systems*, 34:23359–23373, 2021.

[391] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*, 2021.

[392] Common crawl. `https://commoncrawl.org/`. Accessed: 2022-05-18.

[393] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet classifiers generalize to ImageNet? In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[394] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[395] Hieu Pham, Zihang Dai, Golnaz Ghiasi, Hanxiao Liu, Adams Wei Yu, Minh-Thang Luong, Mingxing Tan, and Quoc V Le. Combined scaling for zero-shot transfer learning. *arXiv preprint arXiv:2111.10050*, 2021.

[396] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[397] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[398] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[399] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[400] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022.

[401] Thao Nguyen, Gabriel Ilharco, Mitchell Wortsman, Sewoong Oh, and Ludwig Schmidt. Quality not quantity: On the interaction between dataset design and robustness of clip. *arXiv preprint arXiv:2208.05516*, 2022.

[402] Jiachen Yang, Zhuo Zhang, Yicheng Gong, Shukun Ma, Xiaolan Guo, Yue Yang, Shuai Xiao, Jiabao Wen, Yang Li, Xinbo Gao, et al. Do deep neural networks always perform better when eating more data? *arXiv preprint arXiv:2205.15187*, 2022.

[403] Saachi Jain, Hadi Salman, Eric Wong, Pengchuan Zhang, Vibhav Vineet, Sai Vemprala, and Aleksander Madry. Missingness bias in model debugging. In *International Conference on Learning Representations*, 2022.

[404] Guillaume Leclerc, Hadi Salman, Andrew Ilyas, Sai Vemprala, Logan Engstrom, Vibhav Vineet, Kai Xiao, Pengchuan Zhang, Shibani Santurkar, Greg Yang, et al. 3db: A framework for debugging computer vision models. *arXiv preprint arXiv:2106.03805*, 2021.

[405] Nicola De Cao, Wilker Aziz, and Ivan Titov. Editing factual knowledge in language models. 2021.

[406] Lin Zhang, Lei Zhang, Xuanqin Mou, and David Zhang. Fsim: A feature similarity index for image quality assessment. *IEEE Transactions on Image Processing*, 20(8):2378–2386, 2011.

[407] The Mosaic ML Team. composer. `https://github.com/mosaicml/composer/`, 2021.

[408] Ramprasaath R. Selvaraju, Purva Tendulkar, Devi Parikh, Eric Horvitz, Marco Tulio Ribeiro, Besmira Nushi, and Ece Kamar. Squinting at vqa models: Introspecting vqa models with sub-questions. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10000–10008, 2020.