Towards ML Models That We Can Deploy Confidently

by

Hadi Salman

B.S., Mathematics, American University of Beirut (2016)
B.E., MechE, American University of Beirut (2016)
M.S., Robotics, Carnegie Mellon University (2018)

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

September 2023

©2023 Hadi Salman. All rights reserved.
The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Hadi Salman
Department of Electrical Engineering and Computer Science
August 28, 2023

Certified by: Aleksander Mądry
Cadence Design Systems Professor of Computing
Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Towards ML Models That We Can Deploy Confidently

by
Hadi Salman

Submitted to the Department of Electrical Engineering and Computer Science
on August 28, 2023, in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy

Abstract

As machine learning (ML) systems are deployed in the real world, the reliability and trustworthiness of these systems become an even more salient challenge. This thesis aims to address this challenge through two key thrusts: (1) making ML models more trustworthy by leveraging what has been perceived solely as a weakness of ML model—adversarial perturbations, and (2) exploring the underpinnings of reliable ML deployment.

Specifically, in the first thrust, we focus on adversarial perturbations, which constitute a well-known threat to integrity of ML models, and show how to build ML models that are robust to so-called adversarial patches. We then show that adversarial perturbations can be repurposed to not just be a weakness of ML models but rather to bolster these models' resilience and reliability. To this end, we leverage these perturbations to, first, develop a way to create objects that are easier for ML models to recognize, then to devise a way to safeguard images against unwanted AI-powered alterations, and finally to improve transfer learning performance.

The second thrust of this thesis revolves around ML model interpretability and debugging so as to ensure safety, equitability, and unbiased decision-making of ML systems. In particular, we investigate methods for building ML models that are more debuggable and provide tools for diagnosing their failure modes. We then study how data affects model behavior, identify unexpected ways in which data might introduce biases into ML models, particularly in the context of transfer learning. Finally, we put forth a data-based framework for studying transfer learning which can help us discover problematic biases inherited from pretraining data.

Thesis Supervisor: Aleksander Mądry
Title: Cadence Design Systems Professor of Computing

Acknowledgements

First and foremost, I want to express my gratitude to God for the endless blessings and guidance throughout my life. Alhamdulillah.

I extend a heartfelt thank you to my advisor, Aleksander Mądry. Your guidance and friendship have been indispensable in shaping my academic and career paths. I am extremely lucky to have had the chance to be one of your students and learn from you. I want to also thank Antonio Torralba and Costis Daskalakis for serving on my committee, and for their valuable insights and discussions.

To all my lab mates, both past and present—Alaa, Andrew, Aspen, Ben, Brian, Debbie, Dimitris, Eric, Guillaume, Hedi, Harshay, Josh, Kai, Kristian, Logan, Saachi, Sam, Sara, Shibani—thank you for making MadryLab a remarkable space for intellectual growth, camaraderie, and even some much-needed levity. Your collective spirit has been invaluable. I am really lucky that I got the chance to work with you all.

I must also acknowledge my professors, friends, and staff at the American University of Beirut. Your unwavering support throughout my undergraduate years was foundational. My gratitude extends to everyone at Carnegie Mellon University for making my time there enriching and enjoyable. Special thanks to everyone at Microsoft Research for the life-changing experience during the few years I spent there.

To my circle of Lebanese friends in Boston—Ali, Rasha, Lama, Ali, Kareem, Hussein, Ibrahim, Aya, Samia, Ahmad, Mohamad Ali, Maryam, Zeina, Hiba, Diala, Wael, Molly, Mohamad, Rola—your friendship has been my sanctuary in a foreign land, making Boston feel like a second home.

A warm thank you to my extended family in Lebanon for their consistent support, as well as to my wife's family for embracing me as one of their own and standing beside me throughout this journey. Special thanks to my parents, my brother Ali, and my sisters Mariam and Fatima; thank you for the unwavering and unconditional love and support throughout my life.

Lastly, to my incredible wife, Alaa: Your amazing support and love have been my compass through the challenging and rewarding terrain of this PhD journey. I'm eagerly looking forward to the next chapter of our lives together. With all my love, thank you.

Contents

Introduction **19**

I Adversarial perturbations and better ML **29**

1 Building practical certifiably robust classifiers against adversarial patches **31**
1.1 Certified patch defense with smoothing & transformers . . . 32
1.1.1 Preliminaries . . . 33
1.1.2 Smoothed vision transformers . . . 36
1.2 Improving certified and standard accuracies with ViTs . . . 36
1.2.1 ViTs outperform ResNets on image ablations. . . . 37
1.2.2 Ablation size matters . . . 39
1.3 Faster inference with ViTs . . . 40
1.3.1 Dropping masked tokens . . . 40
1.3.2 Empirical speed-up for smoothed ViTs . . . 41

2 Improving transfer learning via adversarial perturbations **43**
2.1 Background on Transfer Learning . . . 44
2.2 Motivation: Fixed-Feature Transfer Learning . . . 45
2.3 Adversarial Robustness and Full-Network Fine Tuning . . . 48
2.4 Analysis and Discussion . . . 49
2.4.1 ImageNet accuracy and transfer performance . . . 50
2.4.2 Robust models improve with width . . . 51
2.4.3 Optimal robustness levels for downstream tasks . . . 53
2.4.4 Comparing adversarial robustness to texture robustness . . . 54

3 Unadversarial examples: Designing objects for robust vision **55**
3.1 Motivation and approach . . . 57
3.1.1 Leveraging more controlled vision settings . . . 57
3.1.2 Unadversarial examples . . . 58

3.1.3 Constructing unadversarial objects . . . 59
3.2 Experimental evaluation . . . 61
3.2.1 Access model and baselines . . . 61
3.2.2 Clean data and synthetic corruptions . . . 62
3.2.3 Classification in 3D simulation . . . 64
3.2.4 Localization for (simulated) drone landing . . . 66
3.2.5 Physical-world unadversarial examples . . . 66

4 Raising the cost of malicious AI-based manipulation 69
4.1 Preliminaries . . . 70
4.1.1 Diffusion Models . . . 70
4.1.2 Adversarial Attacks . . . 73
4.2 Adversarially Attacking Latent Diffusion Models . . . 73
4.3 Results . . . 75
4.3.1 Qualitative Results . . . 76
4.3.2 Quantitative Results . . . 77
4.4 A Techno-Policy Approach to Mitigation of AI-Powered Editing . . . 79

II Understanding the underpinnings of reliable ML deployment 81

5 Model debugging and the missingness bias 83
5.1 Missingness . . . 85
5.1.1 Missingness bias . . . 86
5.1.2 A more natural form of missingness via vision transformers . . . 87
5.2 The impacts of missingness bias . . . 88
5.3 Missingness bias in practice: a case study on LIME . . . 91

6 Debugging computer vision models with 3DB 95
6.1 Designing *3DB* . . . 97
6.2 Debugging and analyzing models with *3DB* . . . 99
6.2.1 Sensitivity to image backgrounds . . . 99
6.2.2 Texture-shape bias . . . 102
6.2.3 Orientation and scale dependence . . . 104
6.2.4 Case study: using *3DB* to dive deeper . . . 106
6.3 Physical realism . . . 106
6.4 Extensibility . . . 108

7 When does bias transfer in transfer learning? 113
- 7.1 Biases Can Transfer . . . 114
- 7.2 Exploring the Landscape of Bias Transfer . . . 116
  - 7.2.1 Bias consistently transfers in the fixed-feature transfer setting . . . 117
  - 7.2.2 Factors mitigating bias transfer . . . 118
- 7.3 Bias Transfer Beyond Backdoor Attacks . . . 119
  - 7.3.1 Transferring co-occurrence biases in object recognition . . . 120
  - 7.3.2 Transferring gender bias in facial recognition . . . 120
- 7.4 Bias Transfer in the Wild . . . 124

8 A data-based framework for studying transfer learning 127
- 8.1 A Data-Based Framework for Studying Transfer Learning . . . 128
- 8.2 Identifying the Most Influential Classes of the Source Dataset . . . 130
- 8.3 Probing the Impact of the Source Dataset on Transfer Learning . . . 131
  - 8.3.1 Capability 1: Extracting target subpopulations . . . 133
  - 8.3.2 Capability 2: Debugging the failures of a transferred model . . . 135
  - 8.3.3 Capability 3: Detecting data leakage and misleading source examples 136

Appendix 157

A Additional details for Chapter 1 157
- A.1 Experimental setup . . . 157
  - A.1.1 Models and architectures . . . 157
  - A.1.2 Datasets . . . 157
  - A.1.3 Training parameters . . . 158
  - A.1.4 Compute and timing experiments . . . 158
  - A.1.5 Example ablations . . . 158
  - A.1.6 Differences in setup from Levine and Feizi [LF20a] . . . 159
- A.2 Ablation sweeps . . . 160
  - A.2.1 Train-time ablation . . . 160
  - A.2.2 Test-time ablations . . . 161
- A.3 Dropping tokens for ViTs . . . 161
  - A.3.1 Computational complexity of ViTs with dropped tokens . . . 162
  - A.3.2 Effect of dropping tokens on speed . . . 163
  - A.3.3 Effect of dropping tokens on performance . . . 164
- A.4 Strided ablations . . . 164

A.4.1 Certification thresholds for strided ablation sets 165
A.4.2 Performance under strided ablations 165
A.5 Block smoothing 166
A.5.1 Practical inference speeds for block smoothing 166
A.6 Extended experimental results 169

B Additional details for Chapter 2 171
B.1 Experimental Setup 171
B.1.1 Pretrained ImageNet models 171
B.1.2 ImageNet transfer to classification datasets 172
B.1.3 Unifying dataset scale 174
B.1.4 Replicate our results 174
B.2 Transfer Learning with ℓ_∞-robust ImageNet models 175
B.3 Object Detection and Instance Segmentation 175
B.4 Background on Adversarially Robust Models 177
B.5 Omitted Figures 179
B.5.1 Full-network Transfer: additional results to Figure 2.5 179
B.5.2 Varying architecture: additional results to Table 2.2 180
B.5.3 Unified scale: additional results to Figure 2.7 181
B.5.4 Stylized ImageNet Transfer: additional results to Figure 2.8b 181
B.5.5 Effect of width: additional results to Figure 2.6 182
B.6 Detailed Numerical Results 183
B.6.1 Fixed-feature transfer to classification tasks (Fig. 2.5) 183
B.6.2 Full-network transfer to classification tasks (Fig. 2.3) 184
B.6.3 Unifying dataset scale 185

C Additional details for Chapter 3 187
C.1 3D Simulation Details 187
C.1.1 Overview of AirSim 187
C.1.2 3D Boosters Classification Experiment 187
C.1.3 Drone Landing Experiment 189
C.2 Experimental Setup 191
C.2.1 Pretrained vision models we evaluate 191
C.2.2 Unadversarial patch/texture training details 191
C.2.3 Details of the physical world experiment 192
C.2.4 Datasets 192

C.2.5 Compute . . . 193
C.2.6 Replicate our results . . . 193
C.3 Omitted Results . . . 193
C.3.1 Corruption benchmark main results: additional results to Figure 3.3b 194
C.3.2 Baselines . . . 195

D Additional details for Chapter 4 **199**
D.1 Experimental Setup . . . 199
D.1.1 Details of the diffusion model we used . . . 199
D.1.2 Our attacks details . . . 199
D.2 Extended Background for Diffusion Models . . . 200
D.3 Additional Results . . . 202
D.3.1 Additional quantitative results . . . 202
D.3.2 Generating Image Variations using Textual Prompts . . . 203
D.3.3 Image Editing via Inpainting . . . 204

E Additional details for Chapter 5 **213**
E.1 Experimental details. . . . 213
E.1.1 Models and architectures . . . 213
E.1.2 Training Details . . . 213
E.1.3 Experimental Details for Section 5.2 . . . 214
E.1.4 Experimental Details for Section 5.3 . . . 215
E.2 Implementing missingness by dropping tokens in vision transformers . . . 216
E.3 Additional experiments (Section 5.2) . . . 217
E.3.1 Additional examples of the bias (Similar to Figure 5.2). . . . 217
E.3.2 Bias for removing patches in various orders . . . 218
E.3.3 Results for different architectures . . . 219
E.3.4 Results for different missingness approximations . . . 220
E.3.5 Using differently sized patches . . . 222
E.3.6 Using superpixels instead of patches . . . 224
E.3.7 Comparison of dropping tokens vs blacking out pixels for ViTs . . . 224
E.4 Additional experiments (Section 5.3) . . . 225
E.4.1 Examples of LIME . . . 225
E.4.2 Top-k ablation test with superpixels. . . . 227
E.4.3 Effects of Missingness Bias on Learned Masks . . . 227
E.5 Other Datasets . . . 229

E.5.1 MS-COCO . . . 229
E.5.2 CIFAR-10 . . . 229
E.6 Relationship to ROAR . . . 231
E.6.1 Overview on ROAR . . . 231
E.6.2 ViTs do not require retraining . . . 231

F Additional details for Chapter 6 **233**
F.1 Experiment dashboard . . . 233
F.2 iPhone app . . . 234
F.3 Controls . . . 234
F.4 Additional experiments details . . . 236
F.4.1 Sensitivity to image backgrounds (Section 6.2.1) . . . 236
F.4.2 Texture-shape bias (section 6.2.2) . . . 237
F.4.3 Orientation and scale dependence (Section 6.2.3) . . . 237
F.4.4 3D models heatmaps (Figure 6.12) . . . 237
F.4.5 Case study: using 3DB to dive deeper (Section 6.2.4) . . . 237
F.4.6 Physical realism (Section 6.3) . . . 238
F.5 Omitted figures . . . 239

G Additional details for Chapter 7 **241**
G.1 Experimental Setup . . . 241
G.1.1 ImageNet Models . . . 241
G.1.2 Transfer details from ImageNet to downstream image classification . 241
G.1.3 Compute and training time . . . 242
G.1.4 Varying architectures . . . 243
G.1.5 The effect of weight decay in full-network transfer learning . . . 244
G.1.6 Clean accuracies for experiments of Section 7.2 . . . 246
G.1.7 Comparison with models trained from scratch (Additional results to Section 7.2) . . . 246
G.1.8 MS-COCO . . . 248
G.1.9 CelebA . . . 248
G.2 ImageNet Biases . . . 251
G.2.1 Chainlink fence bias. . . . 251
G.2.2 Hat bias. . . . 254
G.2.3 Tennis ball bias. . . . 255

H Additional details for Chapter 8 **257**

H.1 Experimental Setup 257
H.1.1 ImageNet Models 257
H.1.2 ImageNet transfer to classification datasets 257
H.1.3 Compute and training time. 258
H.1.4 Handpicked baseline details 259
H.1.5 Convergence Analysis 259
H.2 Variants of Computing Influences 261
H.2.1 Variations of targets for computing transfer influences 261
H.3 Full Counterfactual Experiment 263
H.4 Adapting our Framework to Compute the Effect of Every Source Datapoint on Transfer Learning 267
H.5 Omitted Results 269
H.5.1 Per-class influencers 269
H.5.2 More examples of extracted subpopulations from the target dataset . 271
H.5.3 More examples of transfer of shape and texture feature 272
H.5.4 More examples of debugging mistakes of transfer model 273
H.5.5 Do Influences Transfer? 274
H.6 Further Convergence Analysis 276

List of Figures

1 An example of an adversarial example 20

1.1 Example of column ablations for derandomized smoothing 33
1.2 Illustration of the smoothed vision transformer 33
1.3 Accuracies on column-ablated images for models 38
1.4 Certified accuracies for ViTs and ResNets as patch size varies 38
1.5 The effect of ablation size at inference time for the smoothed classifiers . . . 39
1.6 Average time for forward pass of smoothed ViTs 41

2.1 Difference betweeen adversarially robust and standard representations . . . 47
2.2 Overview of fixed-feature transfer learning results for robust models 48
2.3 Overview of full-network transfer learning results for robust models 49
2.4 Object detection with robust backbones 50
2.5 Fixed-feature transfer accuracies of robust ImageNet models 51
2.6 Effect of width on robust transfer 52
2.7 Effect of dataset scale on fixed-feature robust transfer learning 54
2.8 Comparison between standard, stylized and robust ImageNet models. . . . 54

3.1 Overview of unadversarial objects 56
3.2 Two methods for constructing unadverasarial objects 60
3.3 Example ImageNet images with unadversarial patches 62
3.4 Unadversarial examples improve OOD accuracy on ImageNet-C 62
3.5 Classifier relies on both object and unadversarial patch for predictions . . . 64
3.6 The unadversarial jet example 65
3.7 Additional unadversarial objects 65
3.8 Drone landing on unadversarial landing pad 66
3.9 Unadversarial examples in the physical world 67

4.1 Overview of our image immunization framework 70
4.2 Overview of diffusion models' capabilities 72
4.3 Overview of encoder and diffusion attacks 74

4.4 Encoder attack immunization example 76
4.5 Photo-guarding photos with encoder and diffusion attacks 77
4.6 Image-prompt similarity after immunization 79

5.1 Importance of missingness in model debugging 85
5.2 Qualitative illustration of missingness bias 86
5.3 Quantitative illustration of missingness bias 89
5.4 Prediction change under patch ablations 89
5.5 Prediction change under patch ablations with retraining 89
5.6 WordNet similarity for assessing missingness bias 90
5.7 Effect of missingness bias on LIME explanations 92
5.8 Jaccard similarity for assessing the missingness bias 92
5.9 LIME explanations on ViTs and ResNets 93
5.10 LIME explanations on ViTs and ResNets with missingness augmentation . . 94

6.1 Examples of vulnerabilities of computer vision systems 96
6.2 *3DB* allows users to realistically compose transformations 96
6.3 An overview of the *3DB* workflow 99
6.4 Visualization of model accuracy per object and per environment 101
6.5 Coffee mug 3D model rendered in different environments 102
6.6 Best and worst environments for the coffee mug 102
6.7 Relation between the complexity of a background and its average accuracy 103
6.8 Using *3DB* to study the composition of zoom and background changes . . . 103
6.9 Generating texture vs. shape cue-conflict with *3DB* 103
6.10 The effect of modifying texture on accuracy 103
6.11 What is more important - texture or shape? 104
6.12 Analyzing model sensitivity to pose via heatmaps 105
6.13 Per-object analysis of the effect of orientation and zoom 105
6.14 The liquid inside a coffee mug determines whether or not it is a coffee mug 107
6.15 Comparison between images from *3DB* and their real-world counterparts . 109
6.16 Examples of how to extend *3DB* 109

7.1 Bias transfer in the fixed-feature setting in the backdoor example 116
7.2 The effect of bias strength on bias transfer 117
7.3 Bias transfer in full-network finetuning setting 118
7.4 Effect of target dataset debiasing on bias transfer 119
7.5 MS-COCO bias transfer experiment 121

7.6 CelebA bias transfer experiment 121
7.7 The "chainlink fence" ImageNet bias 122
7.8 The "tennis ball" ImageNet bias 123

8.1 Top positive and negative ImageNet influencing classes 131
8.2 Transfer accuracy as we remove negative influencing ImageNet classes . . . 132
8.3 Top class influences for CIFAR-10 "bird" class 133
8.4 Projecting source labels onto the target datset 133
8.5 Finding subpopulations using class influences 134
8.6 Class influences for debugging ML models 135
8.7 Detecting data leakage between source and target datasets 136

A.1 Example ablations that we use in this chapter. 159
A.2 Train-time ablation for smoothed ViTs 160
A.3 Test-time ablations for smoothed ViTs 161
A.4 Effect of dropping tokens on derandomized smoothing speed 164
A.5 Effect of dropping tokens on derandomized smoothing performance 164
A.6 Effect of strided ablations on derandomized smoothing 165
A.7 Average inference time to compute with block smoothing 167
A.8 Strided block smoothing on ImageNet varying ablation size 168
A.9 Strided block smoothing on ImageNet for the best ablation size 168

B.1 Object detection additional results with robust backbones 176
B.2 An example of an adversarial attack 178
B.3 Full-network transfer accuracies from standard vs. robust models 179
B.4 Effect of unifying dataset scale on robust full-network transfer 181
B.5 Comparison between standard, stylized, and robust transfer models 181
B.6 Additional results on the effect of width for robust transfer 182

C.1 Various AirSim environment for testing unadeversarial examples 188
C.2 Physical unadversarial examples in various poses 193
C.3 Detailed plots for 2D unadversarial examples ImageNet ResNet-18 194
C.4 Detailed plots for 2D unadversarial examples ImageNet ResNet-50 194
C.5 QR-Code boosted ImageNet results under various corruptions. 195
C.6 Best training example baseline for unadversarial examples 196
C.7 Best training example baseline vs random example baseline 197
C.8 Predefined pattern baseline for unadversarial examples 198

D.1 Immunization against generating prompt-guided image variations. 203
D.2 Immunization against image editing via prompt-guided inpainting. 204

E.1 Additional qualitative examples demonstrating missingness bias 217
E.2 Full experiments of removing patches to study missingness bias 218
E.3 Missingness bias for ViT-T vs ResNet-18 219
E.4 Missingness bias for ViT-S vs ResNet-50 219
E.5 Missingness bias for ViT-S vs InceptionV3 220
E.6 Missingness bias for ViT-S vs VGG16 220
E.7 Various baseline color for approximating missingness 221
E.8 Blur as a missingness approximation 222
E.9 Effect of using various patch sizes for masking pixels 223
E.10 Using SLIC superpixels instead of patches 224
E.11 Dropping tokens vs masking pixels as missingness approximations 225
E.12 Examples of LIME explanations 226
E.13 Top K ablation test using superpixels 227
E.14 Effect of missingess bias on learned masks in model debugging 228
E.15 Missingness bias in MS-COCO 230
E.16 Missingness bias in CIFAR-10 230

F.1 The *3DB* dashboard used for data exploration. 233
F.2 The iOS app used to recreate real-world versions of render *3DB* images . . . 235
F.3 Picture of studio used for the real-world experiments 236
F.4 Spherical objects have different sensitivity to object heading and tilt 239
F.5 Additional plots of the mug liquid experiment 239
F.6 Additional figures of the texture swap experiment 240

G.1 Backdoor bias transfer on various architectures 243
G.2 Effect of weight decay on bias transfer 244
G.3 The clean accuracy of transfer models as we vary weight decay 245
G.4 The clean accuracies of backdoor biased models we used 247
G.5 Additional baseline: training from scratch and measuring bias transfer . . . 247
G.6 CelebA Experiment bias transfer full results 250
G.7 The chainlink fence bias in ImageNet 251
G.8 The chainlink fence bias transfers to Birdsnap 252
G.9 The chainlink fence bias transfers to Flowers 252
G.10 The chainlink fence bias transfers to Food 252

G.11 The chainlink fence bias transfers to SUN397 253
G.12 The hat bias in ImageNet 254
G.13 The hat bias transfers to CIFAR-10 254
G.14 The tennis ball bias in ImageNet 255
G.15 The tennis ball bias transfers to CIFAR-100 255
G.16 The tennis ball bias transfers to Aircraft 256
G.17 The tennis ball bias transfers to Birdsnap 256
G.18 The tennis ball bias transfers to SUN397 256

H.1 Detailed counterfactual experiments with more models 260
H.2 Standard deviation of the class influences as number of models varies 260
H.3 The effect of influence targets on counterfactual experiments 261
H.4 Datamodels vs influences counterfactual experiment 262
H.5 ImageNet class influencers for all CIFAR-10 classes 269
H.6 ImageNet class influencers for all CIFAR-10 classes (Continued) 270
H.7 Most positively influenced CIFAR-10 samples by ImageNet classes 271
H.8 Most highly influenced CIFAR-10 samples by ImageNet classes 272
H.9 More examples of debugging transfer mistakes through our framework . . 273
H.10 Influence transfer across datasets 275
H.11 Influence transfer across architectures 276
H.12 Effect of number of models on influence estimation (rank correlation) 277
H.13 Effect of number of models on influence estimation (FDR) 278

List of Tables

1.1 Summary of our ImageNet results and comparisons to other baselines . . . 35
1.2 Summary of our CIFAR-10 results and comparisons to other baselines . . . 37
1.3 Multiplicative speed up of smoothed ViT over smoothed ResNet 42

2.1 Transfer performance of robust and standard ImageNet models 44
2.2 Source and target accuracies for a fixed robustness level 52

4.1 Effect of immunization on image quality after manipulation 78

6.1 Baseline accuracy of a standard pre-trained model on *3DB*-rendered objects 100

A.1 A collection of neural network architectures we use in this chapter. 157
A.2 Extended summary of CIFAR-10 smoothing results 169
A.3 Standard accuracies of regular and smoothed models 169
A.4 Extended summary of ImageNet smoothing results 170

B.1 Clean accuracies of ℓ_∞-robust ImageNet classifiers. 171
B.2 Clean accuracies of standard and ℓ_2-robust ImageNet classifiers 172
B.3 Classification datasets used in this chapter. 173
B.4 Transfer Accuracy of standard vs ℓ_∞-robust ImageNet models 175
B.5 Additional source and target accuracies for a fixed robustness level 180
B.6 Fixed-feature transfer detailed numerical results 183
B.7 Full-network transfer detailed numerical results 184
B.8 Fixed-feature transfer on 32x32 downsampled datasets. 185
B.9 Full-network transfer on 32x32 downsampled datasets 186

D.1 Hyperparameters used for the Stable Diffusion model. 199
D.2 Hyperparameters used for the adversarial attacks. 200
D.3 Effect of immunization on image quality after manipulation 202

E.1 Neural network architectures we used in this chapter 213
E.2 A collection of neural network architectures we use in this chapter. 214

G.1 Image classification benchmarks used in this chapter 242
G.2 The synthetic datasets we create from MS-COCO for testing bias transfer . . 248
G.3 Hyperparameters used for training on the MS-COCO dataset 248
G.4 The synthetic source datasets we create from CelebA 249
G.5 The synthetic target datasets we create from CelebA 249
G.6 Hyperparameters used for training on the CelebA datasets 249

H.1 Image classification datasets used in this chapter. 258

Introduction

Over the past decade, machine learning (ML) has fueled remarkable advancements in various fields such as computer vision [KSH12], natural language processing [VSP+17; DCL+19], and speech recognition [GMH13; BZM+20; ZQP+20]. The widespread utilization of ML in diverse fields accentuates the critical necessity to thoroughly evaluate its reliability, trustworthiness, and deployability in real-world systems. As machine learning continues to spread into complex and ever-changing areas, making sure it works reliably and fairly is crucial.

Despite the substantial achievements attributed to ML models, they are not devoid of shortcomings. Interestingly, these models are fragile, inadvertently aligning with superficial patterns that perpetuate existing biases within the data they are trained on. A clear illustration of this brittleness is the phenomenon of *adversarial examples* [BCM+13; SZS+14], where imperceptible perturbations to images can disrupt ML models leading to erroneous classifications (cf. Figure 1).

Adversarial examples represent merely one manifestation of the broader issue of ML models' lack of robustness particularly when exposed to *distribution shift*, where the data distribution at test time diverges from that at training time. This casts doubts on the readiness of ML for real-world deployment, accentuating the need for dependable systems capable of withstanding dynamic, real-world conditions. Consequently, the following critical question arises:

How can we confidently and responsibly deploy machine learning in the wild?

This thesis advances this overarching challenge via two major thrusts, both working towards addressing some of the most critical issues facing real-world deployment of machine learning. These two thrusts are:

Adversarial perturbations and better ML. The development of new ML models often involves optimization on static benchmarks, which can be quite different from the scenarios these models face during deployment. This discrepancy necessitates the creation of models

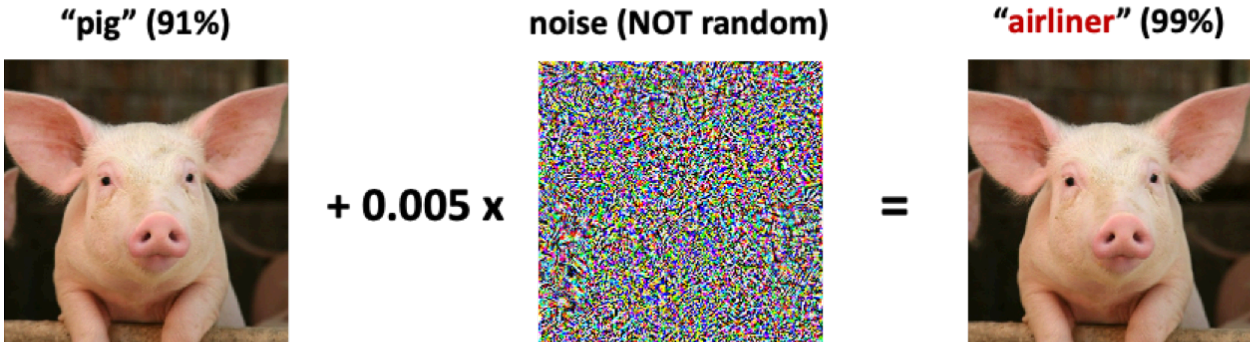


Figure 1: Making nearly invisible changes (adversarial perturbations) to an image of a "pig" can lead an otherwise highly accurate classifier to incorrectly identify it as an "airliner." This phenomenon is known as an adversarial example.

that are robust and reliable, especially for high-stakes applications where prediction accuracy is crucial. The first part of this thesis addresses this need, with a focus on worst-case distribution shifts manifested by adversarial perturbations. We first show how to build models robust to these perturbations, and demonstrate how, in addition to being safer to deploy from a security perspective, such robust models generalize better when used for downstream tasks (e.g., in transfer learning). We then switch gears to show how we can utilize these (seemingly bad) adversarial perturbations to (1) create of *robust objects* that can be easily recognized by ML models under distribution shift, and (2) protect images against unwanted AI manipulation, both of which also aid in making ML deployment more reliable and trustworthy.

Understanding the underpinnings of reliable ML deployment. Real-world ML deployment requires more than robust and reliable models; it demands a deep understanding of models' decision-making processes for safety, equity, and bias detection. The second part of this thesis focuses on deciphering and troubleshooting ML models. It explores (1) building debuggable ML models, (2) developing tools for detecting and understanding how ML models fail, and (3) investigating how data can unexpectedly bias and affect ML models. This multifaceted approach seeks to enhance our comprehension and control over ML models in practical applications.

In the subsequent sections of this introduction, we present a summary of each of these primary areas of focus, delineating our principal concepts and findings, and aligning them with the respective parts of the thesis.

Part I: Adversarial perturbations and better ML

The evolving landscape of ML has witnessed both incredible achievements and unique challenges. As the applications of ML models grow in complexity and criticality, understanding and addressing vulnerabilities becomes an imperative part of model development. In the first part of this thesis, we delve into the intricate relationship between adversarial perturbations (as depicted in Figure 1) and the improvement of ML models. We not only explore how adversarial examples can pose severe threats to system reliability but also uncover how they can be turned into valuable assets for understanding and improving machine learning systems.

In particular, we first demonstrate how to create robust models that can withstand adversarial perturbations, a theme we delve into in Chapter 1. We then demonstrate in Chapter 2 how, in addition to being safer to deploy from a security perspective, such robust models generalize better when used for downstream tasks in particular in transfer learning settings. Finally, we shift our focus towards showcasing how these perturbations, often perceived as harmful, can be repurposed to enhance various facets of ML. This includes the creation of objects that are more easily recognizable by ML models (explained in Chapter 3), and the protection of images from unwanted alterations by generative ML models (covered in Chapter 4).

Building adversarially robust ML models

Adversarial examples pose serious threats on the reliability and security of ML models. Indeed, they can be used to attack ML models in the wild, leading to erroneous predictions. This is especially concerning in safety-critical applications such as autonomous driving, where a small perturbation to a stop sign can lead to a car ignoring this sign. This motivates the need for building models that are robust to adversarial perturbations.

Most of my pre-doctoral work has focused on building models that are robust to *imperceptible* adversarial perturbations [SLR+19; SYZ+19; SSY+20]. However, in practice, we usually expect to deal with more physically-realizable adversarial perturbations that might not be imperceptible. This class of perturbations, known as adversarial patches [BMR+18], are characterized by arbitrary changes contained within a small, contiguous region of the input (e.g., a small square). Adversarial patches are particularly interesting types of adversarial examples due to how easy it is to generate and deploy them. They also capture the essence of a range of maliciously designed physical objects such as adversarial glasses [SBB+16], stickers/graffiti [EEF+18a], and adversarial clothing [WLD+20]. To

defend against such threats, we commence this thesis in Chapter 1 and put forth *smoothed vision transformers* [SJW+21], which are vision transformers (ViTs [DBK+21]) that are built to be certifiably robust against adversarial patches. These models achieve non-precedent robustness against adversarial patches while remaining a viable alternative for standard non-robust models thanks to maintaining competitive standard accuracy and inference speeds. We thus demonstrate that, contrary to general expectation, certified robustness does not need to come at a high price of deployment (i.e., bad performance on benign data and slow inference).

Adversarial perturbations beyond model security

Although adversarial examples are at first glance just a threat to the robustness and reliability of ML systems, one can also view them as symptomatic of a much deeper problem: a fundamental *misalignment* between humans and ML models [IST+19]. The fact that an adversary can make a tiny perturbation to parts of an image that humans do not consider important, and yet the model totally changes classification, indicates that the model makes its decision very differently from how humans do. Indeed, Ilyas et al. [IST+19] show that ML models fundamentally rely on features that humans do not. Their results indicate that while such features are extremely brittle, they are still useful in classification. The relationship this work uncovers between adversarial perturbations and the mechanisms underlying model decisions paved the way for exploring the role of adversarial robustness as a *prior* for aligning ML models with human perception [EIS+19b; STT+19; SIE+20]. So how can this observation be used to improve ML models?

Robustness prior improves transfer learning. In Chapter 2, we find that this robustness prior allows ML models to learn significantly better feature representations. Specifically, we investigate the benefits of such robust representations in the context of transfer learning (where one fine-tunes models from pre-trained weights to obtain better performance on a given task than could be achieved with random initialization). We find that despite having lower accuracy on the original task, robustly trained models yield higher accuracies on downstream tasks spanning image classification, object detection, and semantic segmentation. The reliability and breadth of tasks for which robust models outperform standard models is evidence that robust representations can be more meaningful than those of standardly trained models.

Building (unadversarial) objects that are easily recognized by ML models. The fact that ML models and humans approach decision-making differently motivated us to further ask the questions: given that ML models rely on (non-robust) features that humans do not use, can we leverage these features to improve our ML models? Specifically, can we design the world to be robustly perceived by our ML models? Indeed, in Chapter 3, we introduce "unadversarial examples,"—objects specifically designed (leveraging adversarial perturbations) to be robustly recognized by ML models.

In particular, we proposed a new approach to image recognition in the face of unforeseen corruptions or distribution shifts. This approach is rooted in a reconsideration of the problem setup itself. Specifically, we observe that in many situations, a system designer actually controls, to some extent, the inputs that are fed into their model. For example, a drone operator seeking to train a landing pad detector can modify the surface of the landing pad; and, a roboticist training a perception model to recognize a small set of custom objects can slightly alter the texture or design of these objects. Indeed, a similar insight motivates QR codes, which are patterns explicitly designed to encode easily recoverable bits in photographs.

We find that such control over inputs can be leveraged to drastically improve our ability to tackle computer vision tasks. In particular, it allows us to turn the reliance of modern vision systems on non-robust features from a weakness into a strength. Instead of optimizing inputs to mislead models (e.g., as in adversarial examples), we can alter inputs to reinforce correct behavior, yielding to unadversarial examples or robust objects. Indeed, we show that even a simple gradient-based algorithm can successfully construct unadversarial examples in a variety of vision settings and demonstrate that, by optimizing objects for vision systems (rather than vice-versa), we can significantly improve both in-distribution performance and robustness to unforeseen data shifts and corruptions. This was our first demonstration that adversarial perturbations can be used to improve ML models, and not just to attack them.

Defending against malicious AI-powered image editing. Another venue where we show that adversarial perturbations can be useful is for protecting images against unwanted manipulation by generative AI models. Indeed, recently large diffusion models such as DALL·E 2 [RDN+22] and Stable Diffusion [RBL+22] have emerged with impressive capabilities to produce high-quality photorealistic images. However, the ease of use of these models has raised concerns about their potential abuse, e.g., by creating inappropriate or harmful digital content. In Chapter 4, we propose an approach that aims to protect people against malicious AI-powered image editing. At the core of our approach is

the idea of image *immunization*—that is, making a specific image resistant to AI-powered manipulation by adding (imperceptible) adversarial perturbation to it. This perturbation would disrupt the operation of a diffusion model, forcing the edits it performs to be unrealistic. This is as in the previous chapter, motivated by the fact that ML models heavily rely on non-robust features in their predictions. Thus, we aim to utilize these non-robust features to immunize images against malicious AI-powered manipulation.

Part II: Understanding the underpinnings of reliable ML deployment

In the first part of this thesis, we explored how we can improve the reliability of ML models by making them more robust, performant, and trustworthy. However, even with highly performant and robust ML models, the complexities of real-world deployment demand more than resilience. It's essential to thoroughly examine the decision-making process of these models, whether for ensuring safety, maintaining equity, or identifying underlying biases in data and algorithms. Indeed, we should not deploy ML models in safety-critical settings until we have a more precise characterization of how they work, and a clear understanding of when they would fail.

To this end, the second part of this thesis aims to develop a rigorous foundation for explaining how, why, and in what settings modern machine learning systems succeed or fail.

Building interpretable ML models. Interpretability is critical in nearly any deployment situation. Indeed, it is essential for identifying biases in models, finding and ameliorating potential failure modes, uncovering potential negative externalities of a model's operation in the real world, and—more generally—ensuring that the underlying process behind decision-making is well-aligned with how we would like models to make decisions.

A key primitive in interpretability is the ability to remove features from the input of ML models, sometimes referred to as *missingness*. Indeed, by comparing the model's output with and without specific features, we can infer what parts of the input led to a specific outcome, as done in various interpretability methods [SLL20; STY17; ACÖ+17; RSG16a]. However, there is a problem: removing features from inputs is not always straightforward. Indeed, removing a feature from an image usually requires approximating missingness by replacing those pixel values with something else, e.g., black color. However, these approximations tend not to be perfect [SLL20].

In chapter 5, we investigate how the above missingness approximations can result in what we call *missingness bias*, hindering our ability to properly interpret ML models. We then show how transformer-based architectures can enable a more natural implementation of missingness, allowing us to side-step this bias, and leading to more interpretable ML models.

Developing model debugging tools. Even if we build interpretable ML models, it is important to properly evaluate and diagnose failure modes for these models before deploying them in the real world. In Chapter 6, we introduce 3DB, a framework for automatically identifying and analyzing the failure modes of computer vision models. Specifically, we integrate a 3D simulator into a robustness analysis pipeline to render realistic scenes that can be used to stress-test computer vision system. 3DB is general enough to enable users to, with little-to-no effort, evaluate the robustness of ML models to pose, background, and texture bias, among others.

Such a tool is important for testing the ML model in isolation. However, in most scenarios, the ML model is just a component of a larger system, and this requires more complex techniques as we explore in the next chapters.

Investigating how data can impact the full ML pipeline. In practice, the process of creating an ML system is not merely confined to training a model on a dataset and evaluating it through benchmarks. ML models often grow and change over time, forming integral components of more complex systems. This complexity underscores the necessity of devising tools that can accurately detect and diagnose failure points within the complete pipeline. For instance, ML models are often crafted using transfer learning, drawing from pre-existing models or source datasets. This method raises pertinent questions: Are there underlying issues or biases embedded in the transfer process? And if so, do these challenges propagate into the subsequent, downstream models?

Indeed, in Chapter 7, we show that biases in pretrained models can (and do) transfer to downstream tasks—a phenomenon that we refer to as *bias transfer*. For example, consider a facial recognition system that has been pretrained on some large-scale dataset (which contains a racial or gender bias), and then fine-tuned on a curated downstream dataset where all genders or races are *equally represented*. Our work shows that the racial or gender bias shows up in the resulting model, even though this models was finetuned on a curated and de-biased dataset.

This phenomenon raises more flags on the equitability and fairness of our deployed ML models, and begs for methods to detect and remedy these limitations. So we need de-

bugging tools that work across the ML pipeline, from pretraining all the way to finetuning and testing stages. In Chapter 8, we present a framework for pinpointing the impact of the source dataset on the downstream predictions in transfer learning. This framework draws inspiration from techniques such as influence functions [FZ20] and datamodels [IPE+22] and enables us, in particular, to automatically identify source datapoints that—positively or negatively—impact transfer learning performance. Using this framework, we are able to detect datapoints in the source dataset that are detrimental to the downstream performance on the target task, debug failure modes originating due to transfer learning, and surface pathologies such as source-target data leakage and misleading or mislabelled source datapoints, which we would not be able to do by applying debugging techniques (e.g. 3DB alone) to the ML model itself without considering the full pipeline starting from pretraining.

Outlook: Towards confident ML deployment

Deployable machine learning (ML) continues to pose significant challenges, as complexities arise with the evolution and increased capabilities of models, such as Large Language Models (LLMs). Though this thesis sheds light on certain aspects of deployable ML, there remains an extensive path to achieving confident ML deployment. Further research is required in this field, and some pressing unanswered questions are highlighted below.

Reassessing adversarial examples for LLMs. Despite efforts to enhance ML model robustness against adversarial examples, it is still a difficult open problem. The focus has recently shifted towards average case robustness for general distribution shifts, given the difficulty of addressing the worst-case robustness problem. However, with the advancements in LLMs, it has become vital to revisit adversarial examples. For instance, how can we strengthen LLMs against jailbreaks [WHS23; ZWK+23], a particular type of adversarial example affecting LLMs? As LLMs grow more potent and encompass various modalities, such as images, videos, and audio, jailbreaks become more challenging to prevent [CNC+23].

Furthermore, the solution may not only lie in fortifying ML models but also in strengthening the entire ML pipeline. The current research often overlooks the context in which ML models operate within complex systems. How would the robustness assessment change if an ML model were a part of a complex system? Are content moderation techniques effective against jailbreaks, and how can they be improved? These questions have become exceedingly relevant with LLMs' increasing societal impact.

Creating human-aligned representations. The existence of adversarial examples illustrates the substantial differences in decision-making between ML models and humans. The goal is to align models with human-like feature utilization, thus eliminating adversarial vulnerabilities. How can models be guided to use the "right" features? While adversarial robustness has shown some success in aligning with human representations, this area remains an open challenge. There may be a need for entirely new techniques and methodologies in the quest for human-aligned learning.

Exploring unadversarial examples beyond computer vision. This thesis has explored unadversarial examples as a method to enhance ML model robustness and reliability in image classification. However, this concept can be extended to other domains such as object detection, image segmentation, and beyond. In fields like natural language processing, speech recognition, and tabular data processing, unadversarial examples could play a critical role. Could unadversarial speech signals or text be synthesized to facilitate future recognition? Connecting this with watermarking techniques, especially in detecting fake generated content by LLMs and diffusion models, presents another exciting avenue of exploration.

Debugging ML models as part of a comprehensive system. This thesis has demonstrated how biases can permeate the ML pipeline, emphasizing the need for debugging the entire process to identify failure modes. The focus was specifically on failure modes originating either from the model itself, or from pretrained models that it was built on top of. What other essential components of ML systems might similarly introduce biases or failures? There is a growing necessity to develop specialized tools for comprehensive debugging within the broader context of ML deployment.

Thesis organization

We now describe how the rest of the thesis is organized.
Chapter 1 presents our smoothed vision transformers which achieve non-precedent certified robustness while maintaining competitive inference speeds and clean accuracies, making them viable alternatives to their standard counterparts. The material presented in this chapter is based on joint work with Saachi Jain, Eric Wong, and Aleksander Mądry [SJW+21].
Chapter 2 shows that adversarial robustness can improve transfer learning. The material presented in this chapter is based on joint work with Andrew Ilyas, Logan Engstrom,

Ashish Kapoor, and Aleksander Mądry [SIE+20].

Chapter 3 shows that adversarial perturbations can be used to design objects that are robustly recognized by ML models. The material presented in this chapter is based on joint work with Andrew Ilyas, Logan Engstrom, Sai Vemprala, Ashish Kapoor, and Aleksander Mądry [SIE+21].

Chapter 4 shows that adversarial perturbations can be used to protect images against malicious AI-powered manipulation. The material presented in this chapter is based on joint work with Alaa Khaddaj, Guillaume Leclerc, Andrew Ilyas, and Aleksander Mądry [SKL+23].

Chapter 5 presents the missingness bias, and shows that vision transformers naturally mitigate this bias. The material presented in this chapter is based on joint work with Saachi Jain, Eric Wong, Pengchuan Zhang, Vibhav Vineet, Sai Vemprala, and Aleksander Mądry [JSW+22].

Chapter 6 presents 3DB, a framework for debugging computer vision models. We use this framework to automate discovery of ML model biases and vulnerabilities. The material presented in this chapter is based on joint work with Logan Engstrom, Andrew Ilyas, Guillaume Leclerc, Hadi Salman, Sai Vemprala, Vibhav Vineet, Pengchuan Zhang, Shibani Santurkar, Greg Yang, Ashish Kapoor, and Aleksander Mądry [LSI+21].

Chapter 7 shows that biases can transfer from source datasets to downstream tasks in transfer learning, and demonstrates techniques to mitigate this problem. The material presented in this chapter is based on joint work with Saachi Jain, Andrew Ilyas, Logan Engstrom, Eric Wong, and Aleksander Mądry [SJI+22].

Chapter 8 presents a framework for pinpointing the impact of the source dataset on the downstream predictions in transfer learning. The material presented in this chapter is based on joint work with Saachi Jain, Alaa Khaddaj, Eric Wong, Sung Min Park, and Aleksander Mądry [JSK+22].

Part I

Adversarial perturbations and better ML

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Building practical certifiably robust classifiers against adversarial patches

High-stakes scenarios warrant the development of certifiably robust models that are *guaranteed* to be robust to a set of transformations. These techniques are beginning to find applications in real-world settings, such as verifying that aircraft controllers behave safely in the presence of approaching airplanes [JK19], and ensuring the stability of automotive systems to sensor noise [WSS+20].

In this chapter, we study robustness in the context of adversarial patches—a broad class of arbitrary changes contained within a small, contiguous region. Adversarial patches capture the essence of a range of maliciously designed physical objects such as adversarial glasses [SBB+16], stickers/graffiti [EEF+18a], and clothing [WLD+20]. Researchers have used adversarial patches to fool image classifiers [BMR+18], manipulate object detectors [LK19; HSS+20], and disrupt optical flow estimation [RJG+19].

Adversarial patch defenses can be tricky to evaluate—recent work broke several empirical defenses [BMV18; Hay18; NKP19] with stronger adaptive attacks [TCB+20; CNA+20]. This motivated *certified* defenses, which deliver provably robust models without having to rely on an empirical evaluation. However, certified guarantees tend to be modest and come at a cost: poor standard accuracy and slower inference times [LF20b; LF20a; ZYM+20; XBS+21]. For example, a top-performing, recently proposed method reduces standard accuracy by 30% and increases inference time by two orders of magnitude, while certifying only 13.9% robust accuracy on ImageNet against patches that take up 2% of the image [LF20a]. These drawbacks are commonly accepted as the cost of certification, but severely limit the applicability of certified defenses. Does certified robustness really need to come at such a high price?

Our contributions

In this chapter, we demonstrate how to leverage vision transformers (ViTs) [DBK+21] to create certified patch defenses that achieve significantly higher robustness guarantees than prior work. Moreover, we show that certified patch defenses with ViTs can actually maintain standard accuracy and inference times comparable to standard (non-robust) models. At its core, our methodology exploits the token-based nature of attention modules used in ViTs to gracefully handle the ablated images used in certified patch defenses. Specifically, we demonstrate the following:

Improved guarantees via smoothed vision transformers. We find that using ViTs as the backbone of the derandomized smoothing defense [LF20a] enables significantly improved certified patch robustness. Indeed, this change alone boosts certified accuracy by up to 13% on ImageNet, and 5% on CIFAR-10 over similarly sized ResNets.

Standard accuracy comparable to that of standard architecures. We demonstrate that ViTs enable certified defenses with standard accuracies comparable to that of standard, non-robust models. In particular, our largest ViT improves state-of-the-art certified robustness on ImageNet while maintaining standard accuracy that is similar to that of a non-robust ResNet (>70%).

Faster inference. We modify the ViT architecture to drop unnecessary tokens, and reduce the smoothing process to pass over mostly redundant computation. These changes turn out to vastly speed up inference time for our smoothed ViTs. In our framework, a forward pass on ImageNet becomes up to two orders of magnitude faster than that of prior certified defenses, and is close in speed to a standard (non-robust) ResNet.

1.1 Certified patch defense with smoothing & transformers

Smoothing methods are a general class of certified defenses that combine the predictions of a classifier over many variations of an input to create predictions that are certifiably robust [CRK19; LF20b]. One such method that obtains robustness to adversarial patches is derandomized smoothing [LF20a], which aggregates a classifier's predictions on various *image ablations* that mask most of the image out.

These approaches typically use CNNs, a common default model for computer vision tasks, to evaluate the image ablations. The starting point of our approach is to ask: are

convolutional architectures the right tool for this task? The crux of our methodology is to leverage vision transformers, which we demonstrate are more capable of gracefully handling the image ablations that arise in derandomized smoothing.

1.1.1 Preliminaries

Image ablations. Image ablations are variations of an image where all but a small portion of the image is masked out [LF20a]. For example, a column ablation masks the entire image except for a column of a fixed width (see Figure 1.1 for an example). We focus primarily on column ablations and explore the more general block ablation in Appendix A.5.

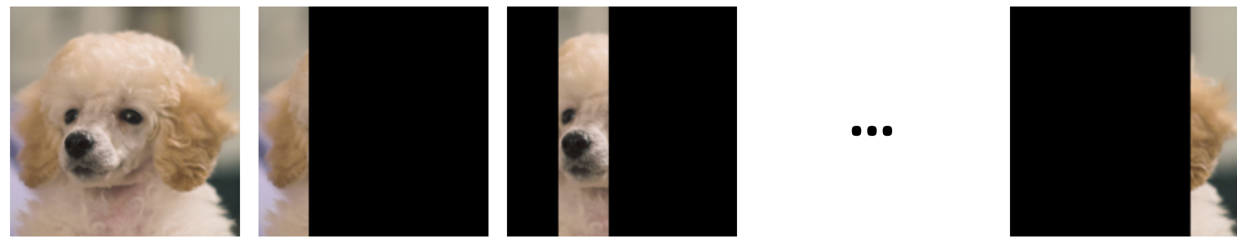


Figure 1.1: Examples of column ablations for the left-most image with column width 19px.

For a input $h \times w$ sized image $\mathbf{x}$, we denote by $\mathcal{S}_b(\mathbf{x})$ the set of all possible column ablations of width b. A column ablation can start at any position and wrap around the image, so there are w total ablations in $\mathcal{S}_b(\mathbf{x})$.

Derandomized smoothing. Derandomized smoothing [LF20a] is a popular approach for certified patch defenses that constructs a *smoothed classifier* comprising of two main

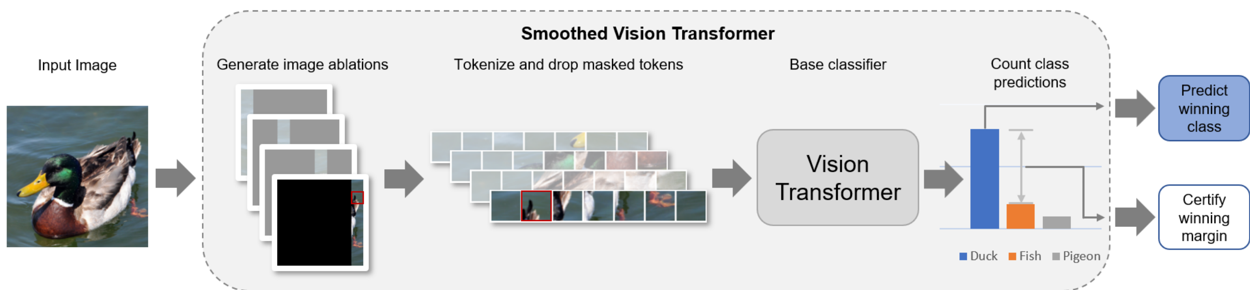


Figure 1.2: Illustration of the smoothed vision transformer. For a given image, we first generate a set of ablations. We encode each ablation into tokens, and drop fully masked tokens. The remaining tokens for each ablation are then fed into a vision transformer, which predicts a class label for each ablation. We predict the class with the most predictions over all the ablations, and use the margin to the second-place class for robustness certification.

components: (1) a *base classifier*, and (2) a set of image ablations used to smooth the base classifier. Then, the resulting smoothed classifier returns the most frequent prediction of the base classifier over the ablation set $\mathcal{S}_b(\mathbf{x})$. Specifically, for an input image $\mathbf{x}$, ablation set $\mathcal{S}_b(\mathbf{x})$, and a base classifier f, a smoothed classifier g is defined as:

$$g(\mathbf{x}) = \arg\max_c n_c(\mathbf{x}) \tag{1.1}$$

where

$$n_c(\mathbf{x}) = \sum_{\mathbf{x}' \in \mathcal{S}_b(\mathbf{x})} \mathbb{I}\{f(\mathbf{x}') = c\}$$

denotes the number of image ablations that were classified as class c. We refer to the fraction of images that the smoothed classifier correctly classifies as *standard accuracy*.

A smoothed classifier is *certifiably robust* for an input image if the number of ablations for the most frequent class exceeds the second most frequent class by a large enough margin. Intuitively, a large margin makes it impossible for an adversarial patch to change the prediction of a smoothed classifier since a patch can only affect a limited number of ablations.

Specifically, let Δ be the maximum number of ablations in the ablation set $S_b(\mathbf{x})$ that an adversarial patch can simultaneously intersect (e.g., for column ablations of size b, an $m \times m$ patch can intersect with at most $\Delta = m + b - 1$ ablations). Then, a smoothed classifier is certifiably robust on an input $\mathbf{x}$ if it is the case that for the predicted class c:

$$n_c(\mathbf{x}) > \max_{c' \neq c} n_{c'}(\mathbf{x}) + 2\Delta. \tag{1.2}$$

If this threshold is met, the most frequent class is guaranteed to not change even if an adversarial patch compromises every ablation it intersects. We denote the fraction of predictions by the smooth classifier that are both correct and certifiably robust (according to Equation 1.2) as *certified accuracy*.

Vision transformers. A key component of our approach is the vision transformer (ViT) architecture [DBK+21]. In contrast to convolutional architecures, ViTs use self-attention layers instead of convolutional layers as their primary building block and are inspired by the success of self-attention in natural language processing [VSP+17]. ViTs process images in three main stages:

1. *Tokenization:* The ViTs split the image into $p \times p$ patches. Each patch is then embedded into a positionally encoded *token*.

Table 1.1: Summary of our ImageNet results and comparisons to certified patch defenses from the literature: Clipped Bagnet (CBG), Derandomized Smoothing (DS), and PatchGuard (PG). Time refers to the inference time for a batch of 1024 images, b is the ablation size, and s is the ablation stride. An extended version is in Appendix A.6.

Standard and Certified Accuracy on ImageNet (%)					
	Standard	1% pixels	2% pixels	3% pixels	Time (sec)
Baselines					
Standard ResNet-50	76.1	—	—	—	0.67
WRN-101-2	78.85	—	—	—	3.1
ViT-S	79.90	—	—	—	0.4
ViT-B	81.80	—	—	—	0.95
CBN [ZYM+20]	49.5	13.4	7.1	3.1	3.05
DS [LF20a][a]	44.4	17.7	14.0	11.2	149.5
PG [XBS+21][b]	55.1[b]	32.3[b]	26.0[b]	19.7[b]	3.05
Smoothed models					
ResNet-50 (b = 19)	51.5	22.8	18.3	15.3	149.5
ViT-S (b = 19)	**63.5**	**36.8**	**31.6**	**27.9**	**14.0**
WRN-101-2 (b = 19)	61.4	33.3	28.1	24.1	694.5
ViT-B (b = 19)	69.3	**43.8**	**38.3**	**34.3**	31.5
ViT-B (b = 37)	**73.2**	43.0	38.2	34.1	58.7
ViT-B (b = 19, s = 10)	68.3	36.9	36.9	31.4	**3.2**

2. *Self-Attention:* The set of tokens are then passed through a series of multi-headed self-attention layers [VSP+17].

3. *Classification head:* The resulting representation is fed into a fully connected layer to make predictions for classification.

[a] We found that ResNets could achieve a significantly higher certified accuracy than was reported by Levine and Feizi [LF20a] if we use early stopping-based model selection. We elaborate further in Appendix A.1.

[b] The PatchGuard defense uses a specific mask size that guarantees robustness to patches smaller than the mask, and provides no guarantees for larger patches. In this table, we report their best results: each patch size corresponds to a separate model that achieves 0% certified accuracy against larger patches. Comparisons across the individual models can be found in Appendix A.6.

1.1.2 Smoothed vision transformers

Two central properties of vision transformers make ViTs particularly appealing for processing the image ablations that arise in derandomized smoothing. Firstly, unlike CNNs, ViTs process images as sets of tokens. ViTs thus have the natural capability to simply drop unnecessary tokens from the input and "ignore" large regions of the image, which can greatly speed up the processing of image ablations.

Moreover, unlike convolutions which operate locally, the self-attention mechanism in ViTs shares information *globally* at every layer [VSP+17]. Thus, one would expect ViTs to be better suited for classifying image ablations, as they can dynamically attend to the small, unmasked region. In contrast, a CNN must gradually build up its receptive field over multiple layers and process masked-out pixels.

Guided by these intuitions, our methodology leverages the ViT architecture as the base classifier for processing the image ablations used in derandomized smoothing. We first demonstrate that these *smoothed vision transformers* enable substantially improved robustness guarantees, without losing much standard accuracy (Section 1.2). We then modify the ViT architecture and smoothing procedure to drastically speed up the cost of inference of a smoothed ViT (Section 1.3). We present an overview of our approach in Figure 1.2.

Setup. We focus primarily on the column smoothing setting and defer block smoothing results to Appendix A.5. We consider the CIFAR-10 [Kri09] and ImageNet [DDS+09] datasets, and perform our analysis on three sizes of vision transformers—ViT-Tiny (ViT-T), ViT-Small (ViT-S), and ViT-Base (ViT-B) models [Wig19; DBK+21]. We compare to residual networks of similar size—ResNet-18, ResNet-50 [HZR+16], and Wide ResNet-101-2 [ZK16], respectively. Further details of our experimental setup are in Appendix A.1.

1.2 Improving certified and standard accuracies with ViTs

Recall that even though certified patch defenses can guarantee robustness to patch attacks, this robustness typically does not come for free. Indeed, certified patch defenses tend to have substantially lower standard accuracy when compared to typical (non-robust) models, while delivering a fairly limited degree of (certified) robustness.

In this section, we show how to use ViTs to substantially improve both standard and certified accuracies for certified patch defenses. To this end, we first empirically demonstrate that ViTs are a more suitable architecture than traditional convolutional

Table 1.2: Summary of our CIFAR-10 results and comparisons to certified patch defenses from the literature: Clipped Bagnet (CBG), Derandomized Smoothing (DS), and PatchGuard (PG). Here, b is the column ablation size out of 32 pixels. An extended version is in Appendix A.6.

Standard and Certified Accuracy on CIFAR-10 (%)			
	Standard	2×2	4×4
Baselines			
CBN [ZYM+20]	84.2	44.2	9.3
DS [LF20a][a]	83.9	68.9	56.2
PG [XBS+21][b]	84.7[b]	69.2[b]	57.7[b]
Smoothed models			
ResNet-50 (b = 4)	86.4	71.6	59.0
ViT-S (b = 4)	**88.4**	**75.0**	**63.8**
WRN-101-2 (b = 4)	88.2	73.9	62.0
ViT-B (b = 4)	**90.8**	**78.1**	**67.6**

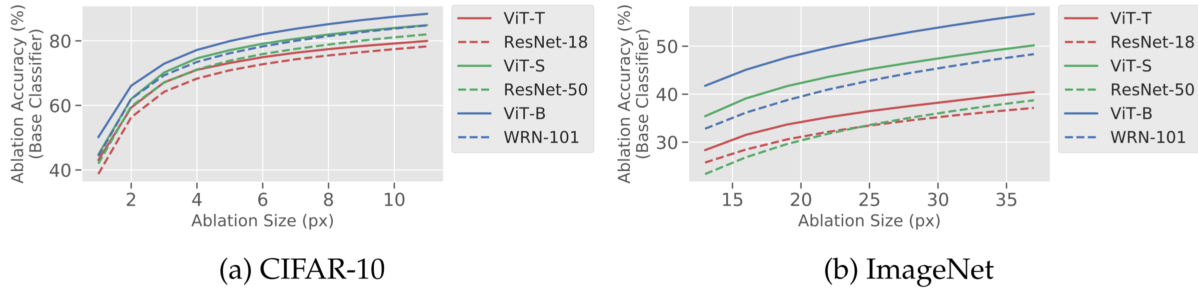
networks for classifying the image ablations used in derandomized smoothing (Section 1.2.1). Specifically, this change in architecture alone yields models with significantly improved standard and certified accuracies. We then show how a careful selection of smoothing parameters can enable smoothed ViTs to have even higher standard accuracies that are comparable to typical (non-robust) models, without sacrificing much certified performance (Section 1.2.2).

Our ImageNet and CIFAR-10 results are summarized in Table 1.1 and Table 1.2, respectively. We further include the inference time to evaluate a batch of images, using the modifications described in Section 1.3. See Appendix A.6 for extended tables covering a wider range of experiments.

1.2.1 ViTs outperform ResNets on image ablations.

We first isolate the effect of using a ViT instead of a ResNet as the base classifier for derandomized smoothing. Specifically, we keep all smoothing parameters fixed and only vary the base classifier. Following Levine and Feizi [LF20a], we use column ablations of width $b = 4$ for CIFAR-10 and $b = 19$ for ImageNet for both training and certification.

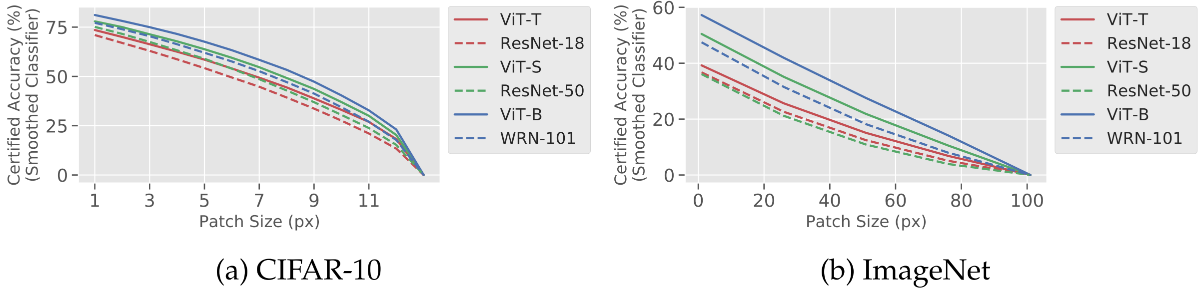
Ablation accuracy. The performance of derandomized smoothing entirely depends on whether the base classifier can accurately classify ablated images. We thus measure the



(a) CIFAR-10

(b) ImageNet

Figure 1.3: Accuracies on column-ablated images for models on CIFAR-10 and ImageNet. The models were trained on column ablations of width $b = 19$ for ImageNet and $b = 4$ for CIFAR-10, and evaluated on a range of ablation sizes. ViTs outperform ResNets on image ablations by a sizeable margin.



(a) CIFAR-10

(b) ImageNet

Figure 1.4: Certified accuracies for ViT and ResNet models on CIFAR-10 and ImageNet for various adversarial patch sizes. Certification was performed using a fixed ablation of size $b = 4$ for CIFAR-10 and $b = 19$ for ImageNet (as in [LF20a]).

accuracy of ViTs and ResNets at classifying column ablated images across a range of evaluation ablation sizes as shown in Figure 1.3. We find that ViTs are significantly more accurate on these ablations than comparably sized ResNets. For example, on ImageNet, ViT-S has up to 12% higher accuracy on ablations than ResNet-50.

Certified patch robustness. We next measure the effect of improved ablation accuracy on certified accuracy. We find that using a ViT as the base classifier in derandomized smoothing substantially boosts certified accuracy compared to ResNets across a range of model sizes and adversarial patch sizes, as shown in Figure 1.4. For example, against 32×32 adversarial patches on ImageNet (2% of the image), a smoothed ViT-S improves certified accuracy by 14% over a smoothed ResNet-50, while the larger ViT-B reaches a certified accuracy of 39%—well above the highest reported baseline of 26% [XBS+21][1].

[1]The highest reported certified accuracy in the literature for this patch size on ImageNet is 26% from PatchGuard [XBS+21]. However, this defense uses a masking technique that is optimized for this particular patch size, and achieves 0% certified accuracy against larger patches.

Standard accuracy. We further find that smoothed ViTs can mitigate the precipitous drop in standard accuracy observed in previously proposed certified defenses, particularly so for larger architectures and datasets. Indeed, the smoothed ViT-B remains 69% accurate on ImageNet—14.2% higher standard accuracy than that of the best performing prior work (Table 1.1). A full comparison between the performance of smoothed models and their non-robust counterparts can be found in Appendix A.6.

1.2.2 Ablation size matters

In the previous section, we fixed the width of column ablations at $b = 19$ for derandomized smoothing on ImageNet, following [LF20a]. We now demonstrate that properly choosing the ablation size can improve the standard accuracy even further—by 4% on ImageNet—without sacrificing certified performance.

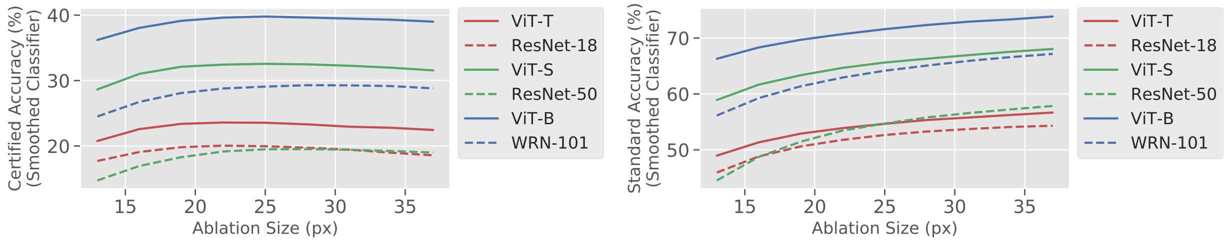


Figure 1.5: Certified (left) and standard (right) accuracies for a collection of smoothed models trained with a fixed ablation size $b = 19$ on ImageNet, and evaluated with varying ablation sizes. Certified accuracy remains stable across a range of ablation sizes, while standard accuracy substantially improves with larger ablations.

Specifically, we take ImageNet models trained on column ablations with width $b = 19$, and change the smoothing procedure to use a different width at *test* time. We report the resulting standard and certified accuracies in Figure 1.5, and defer additional experiments on changing the ablation size during training to Appendix A.2.1.

Although Levine and Feizi [LF20a] found a steep trade-off between certified and standard accuracy in CIFAR-10 (which we verify in Appendix A.2.2), we find this to not be the case for ImageNet for either CNNs or ViTs. We can thus substantially increase the ablation size to improve standard accuracy *without* significantly dropping certified performance as shown in Figure 1.5. For example, increasing the width of column ablations to $b = 37$ improves the standard accuracy of the smoothed ViT-B model by nearly 4% to 73% while maintaining a 38% certified accuracy against 32×32 patches. In addition to being 12% higher than the standard accuracy of the best performing prior work, this model's standard accuracy is only 3% lower than that of a *non-robust* ResNet-50.

Thus, using smoothed ViTs, we can achieve state-of-the-art certified robustness to patch attacks in the ImageNet setting while attaining standard accuracies that are more comparable to those of non-robust ResNets.

1.3 Faster inference with ViTs

Derandomized smoothing with column ablations is an expensive operation, especially for large images. Indeed, an image with $h \times w$ pixels has w column ablations, so the forward pass of smoothed model is w times slower than a normal forward pass—*two orders of magnitude* slower on ImageNet.

To address this, we first modify the ViT architecture to avoid unnecessary computation on masked pixels (Section 1.3.1). We then demonstrate that reducing the number of ablations via striding offers further speed up (Section 1.3.2). These two (complementary) modifications vastly improve the inference time for smoothed ViTs, making them comparable in speed to standard (non-robust) convolutional architectures.

1.3.1 Dropping masked tokens

Recall that the first operation in a ViT is to split and encode the input image as a set of *tokens*, where each token corresponds to a patch in the image. However, for image ablations, a large number of these tokens correspond to fully masked regions of the image.

Our strategy is to pass only the *subset* of tokens that contain an unmasked part of the original image, thus avoiding computation on fully masked tokens. Specifically, given an image ablation, we alter the ViT architecture to do the following steps:

1. Positionally encode the entire ablated image into a set of tokens.
2. Drop any tokens that correspond to a *fully* masked region of the input.
3. Pass the remaining tokens through the self-attention layers.

As one would expect, since the positional encoding maintains the spatial information of the remaining tokens, the ViT's accuracy on image ablations barely changes when we drop the fully masked tokens. We defer a detailed analysis of this phenomenon, along with a formal description of the token-dropping procedure to Appendix A.3.

Computational complexity. We now provide an informal summary of the computational complexity of this procedure, and defer a formal asymptotic analysis to Appendix A.3.1. After tokenization, the bulk of a ViT consists of two main operation types:

- *Attention operators*, which have costs that scale quadratically with the number of tokens but linearly in the hidden dimension.
- *Fully-connected operators*, which have costs that scale linearly with the number of tokens but quadratically in the hidden dimension.

Reducing the number of tokens thus directly reduces the cost of attention and fully connected operators at a quadratic and linear rate, respectively. For a small number of tokens, the linear scaling from the fully-connected operators tends to dominate. The cost of processing column ablations thus scales linearly with the width of the column, which we empirically validate in Figure 1.6. Further details about how we time these models can be found in Appendix A.1.4.

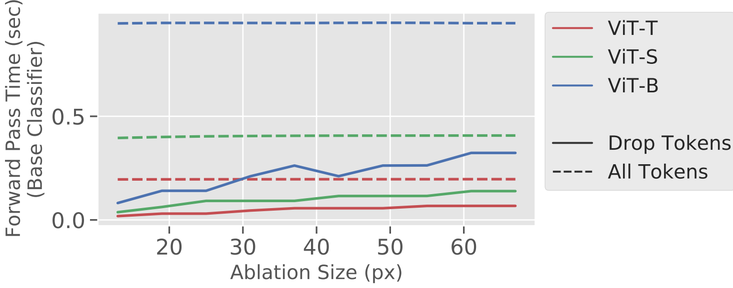


Figure 1.6: The average time to compute a forward pass for ViTs on 1024 column ablated images with varying ablation sizes, with and without dropping masked tokens. The cost of processing a full image without dropping masked tokens corresponds to the maximum ablation size $b = 224$.

1.3.2 Empirical speed-up for smoothed ViTs

Smoothed classifiers must process a large number of image ablations in order to make predictions and certify robustness. Consequently, using our ViT (with dropped tokens) as the base classifier for derandomized smoothing directly speeds up inference time. In this section, we explore how much faster smoothed ViTs are in practice.

We first measure the number of images per second that smoothed ViTs and smoothed ResNets can process. We use column ablations of size $b = 19$ on ImageNet, following Levine and Feizi [LF20a]. In Table 1.3 that describes our results, we find speedups of

Table 1.3: Multiplicative speed up of inference for a smoothed ViT with dropped tokens over a smoothed ResNet, measured over a batch of 1024 images with $b = 19$.

	ResNet-18	ResNet-50	WRN-101
ViT-T	**5.85x**	21.96x	101.99x
ViT-S	2.85x	**10.68x**	49.62x
ViT-B	1.26x	4.75x	**22.04x**

5-22x for smoothed ViTs over smoothed ResNets of similar size, with larger architectures showing greater gains. Notably, using our largest ViT (ViT-B) as the base classifier is 1.25x faster than using a ResNet-18, despite being 8x larger in parameter count. Dropping masked tokens thus substantially speeds up inference time for smoothed ViTs, to the point where using a large ViT is comparable in speed to using a small ResNet.

Strided ablations. We now consider a complementary means of speeding up smoothed classifiers: directly reducing the size of the ablation set via *strided* ablations. Specifically, instead of using every possible ablation, we can subsample every s-th ablation for a given stride s. Striding can reduce the total number of ablations (and consequently speed up inference) by a factor of s, *without* substantially hurting standard or certified accuracy (Table 1.1). We study this in more detail in Appendix A.4.

Strided ablations, in conjunction with the dropped tokens optimization from Section 1.3.1, lead to smoothed ViTs having inference times comparable to standard (non-robust) models. For example, when using stride $s = 10$ and dropping masked tokens, a smoothed ViT-S is only 2x slower than a single inference step of a standard ResNet-50, while a smoothed ViT-B is only 5x slower. We report the inference time of these models, along with their standard and certified accuracies, in Table 1.1.

Chapter 2

Improving transfer learning via adversarial perturbations

In the previous chapter, we laid the groundwork by demonstrating how to create models robust to (a specific class of) adversarial perturbations. As we move forward, this chapter will extend that discussion, showing how such robustness is not just a security asset but also a crucial factor in improving *transfer learning* [DJV+14; SAS+14].

The relevance of this discussion is underpinned by the prevailing role that transfer learning plays in many practical settings where there is insufficient data or compute. Broadly, transfer learning refers to any machine learning algorithm that leverages information from one ("source") task to better solve another ("target") task. A prototypical transfer learning pipeline in computer vision (and the focus of our work) starts with a model trained on the ImageNet-1K dataset [DDS+09; RDS+15], and then refines this model for the target task.

Though the exact underpinnings of transfer learning are not fully understood, recent work has identified factors that make pre-trained ImageNet models amenable to transfer learning. For example, [HAE16; KBZ+19] investigate the effect of the source dataset; Kornblith et al. [KSL19] find that pre-trained models with higher ImageNet accuracy also tend to transfer better; Azizpour et al. [ARS+15] observe that increasing depth improves transfer more than increasing width.

Our contributions. In this work, we identify another factor that affects transfer learning performance: adversarial robustness [BCM+13; SZS+14]. We find that despite being less accurate on ImageNet, adversarially robust neural networks match or improve on the transfer performance of their standard counterparts. We first establish this trend in the "fixed-feature" setting, in which one trains a linear classifier on top of features extracted

Table 2.1: Transfer performance of robust and standard ImageNet models on 12 downstream classification tasks. For each transfer learning paradigm, we report accuracy averaged over ten random trials of standard and robust models. We used a grid search (using a disjoint set of random seets) to find the best hyperparameters, architecture, and (for robust models) robustness level ε. In each column, we bold the entry with the higher average accuracy; if the accuracy difference is significant (as judged by a 95% CI two-tailed Welch's t-test [WEL47]) we bold only the higher entry, otherwise (if the test is inconclusive) we bold both.

		Dataset											
Mode	**Model**	Aircraft	Birdsnap	CIFAR-10	CIFAR-100	Caltech-101	Caltech-256	Cars	DTD	Flowers	Food	Pets	SUN397
Fixed-feature	**Robust**	**44.24**	**50.75**	**95.50**	**81.16**	**92.54**	**85.16**	**51.35**	**70.38**	**92.05**	**69.32**	**92.08**	**58.80**
	Standard	38.52	48.40	81.29	60.08	90.01	82.87	44.54	**70.32**	91.83	65.73	91.92	56.02
Full-network	**Robust**	**86.26**	**76.41**	**98.70**	**89.22**	**95.67**	**87.92**	**91.37**	**77.05**	**96.94**	**89.10**	**94.36**	**64.97**
	Standard	**86.19**	75.90	97.72	86.20	94.85	86.54	**91.37**	76.11	**97.13**	88.61	**94.43**	63.90

from a pre-trained network. Then, we show that this trend carries forward to the more complex "full-network" transfer setting, in which the pre-trained model is entirely fine-tuned on the relevant downstream task. We carry out our study on a suite of image classification tasks (summarized in Table 2.1), object detection, and instance segmentation.

Our results are consistent with (and in fact, add to) recent hypotheses suggesting that adversarial robustness leads to improved feature representations [EIS+19b; AL20]. Still, future work is needed to confirm or refute such hypotheses, and more broadly, to understand what properties of pre-trained models are important for transfer learning.

2.1 Background on Transfer Learning

A number of works study transfer learning with CNNs [DJV+14; CSV+14; SAS+14; ARS+15]. Indeed, transfer learning has been studied in varied domains including medical imaging [MGM18], language modeling [CK18], and various object detection and segmentation related tasks [RHG+15; DLH+16; HRS+17; CPK+17]. In terms of methods, others [AGM14; CSV+14; GDD+14; YCB+14; ARS+15; LRM15; HAE16; CMB+16] show that fine-tuning typically outperforms frozen feature-based methods. As discussed throughout this chapter, several prior works [ARS+15; HAE16; KSL19; ZSS+18; KBZ+19; SSS+17; MGR+18; YCB+14] have investigated factors improving or otherwise affecting transfer learning performance. Recently proposed methods have achieved state-of-the-art performance on downstream tasks by scaling up transfer learning techniques [HCB+18;

KBZ+19].

On the adversarial robustness front, many works—both empirical (e.g., [MMS+18; MMK+18; BGH19; ZYJ+19]) and certified (e.g., [LAG+19; WZC+18; WK18; RSL18; CRK19; SLR+19; YDH+20])—significantly increase model resilience to adversarial examples [BCM+13; SZS+14]. A growing body of research has studied the *features* learned by these robust networks and suggested that they improve upon those learned by standard networks (cf. [IST+19; EIS+19b; STT+19; AL20; KSJ19; KCL19] and references). On the other hand, prior studies have also identified theoretical and empirical tradeoffs between standard accuracy and adversarial robustness [TSE+19; BPR19; SZC+18; RXY+19]. At the intersection of robustness and transfer learning, Shafahi et al. [SSZ+19] investigate transfer learning for increasing downstream-task adversarial robustness (rather than downstream accuracy, as in this work). Aggarwal et al. [ASK+20] find that adversarially trained models perform better at downstream zero-shot learning tasks and weakly-supervised object localization. Finally, concurrent to our work, [UKE+20] also study the transfer performance of adversarially robust networks. Our studies reach similar conclusions and are otherwise complementary: here we study a larger set of downstream datasets and tasks and analyze the effects of model accuracy, model width, and data resolution; Utrera et al. [UKE+20] study the effects of training duration, dataset size, and also introduce an influence function-based analysis [KL17] to study the representations of robust networks.

2.2 Motivation: Fixed-Feature Transfer Learning

In one of the most basic variants of transfer learning, one uses the source model as a feature extractor for the target dataset, then trains a simple (often linear) model on the resulting features. In our setting, this corresponds to first passing each image in the target dataset through a pre-trained ImageNet classifier, and then using the outputs from the penultimate layer as the image's feature representation. Prior work has demonstrated that applying this "fixed-feature" transfer learning approach yields accurate classifiers for a variety of vision tasks and often out-performs task-specific handcrafted features [SAS+14]. However, we still do not completely understand the factors driving transfer learning performance.

How can we improve transfer learning? Conventional wisdom and evidence from prior work [CSV+14; SZ15; KSL19; HRS+17] suggest that accuracy on the source dataset is a strong indicator of performance on downstream tasks. In particular, Kornblith et al. [KSL19] find that pre-trained ImageNet models with higher accuracy yield better fixed-feature transfer learning results.

Still, it is unclear if improving ImageNet accuracy is the only way to improve performance. After all, the behaviour of fixed-feature transfer is governed by models' learned representations, which are not fully described by source-dataset accuracy. These representations are, in turn, controlled by the *priors* that we put on them during training. For example, the use of architectural components [UVL17], alternative loss functions [MIM+18], and data augmentation [VM01] have all been found to put distinct priors on the features extracted by classifiers.

The adversarial robustness prior. In this work, we turn our attention to another prior: *adversarial robustness*. Adversarial robustness refers to a model's invariance to small (often imperceptible) perturbations of its inputs. Robustness is typically induced at training time by replacing the standard empirical risk minimization objective with a robust optimization objective [MMS+18]:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim D} \left[\mathcal{L}(x,y;\theta)\right] \implies \min_{\theta} \mathbb{E}_{(x,y)\sim D} \left[\max_{\|\delta\|_2 \leq \varepsilon} \mathcal{L}(x+\delta,y;\theta)\right], \tag{2.1}$$

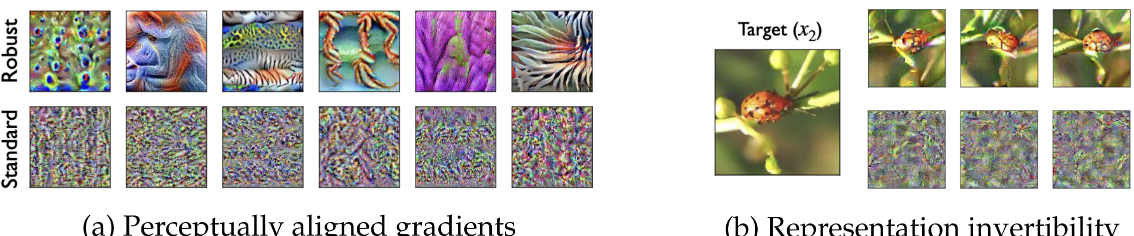
where ε is a hyperparameter governing how invariant the resulting "adversarially robust model" (more briefly, "robust model") should be. In short, this objective asks the model to minimize risk on the training datapoints while also being locally stable in the (radius-ε) neighbourhood around each of these points. (A more detailed primer on adversarial robustness is given in Appendix B.4.)

Adversarial robustness was originally studied in the context of machine learning security [BCM+13; BR18; CW17] as a method for improving models' resilience to adversarial examples [GSS15; MMS+18]. However, a recent line of work has studied adversarially robust models in their own right, casting (2.1) as a prior on learned feature representations [EIS+19b; IST+19; JBZ+19; ZZ19].

Should adversarial robustness help fixed-feature transfer? It is, a priori, unclear what to expect from an "adversarial robustness prior" in terms of transfer learning. On one hand, robustness to adversarial examples may seem somewhat tangential to transfer performance. In fact, adversarially robust models are known to be significantly less accurate than their standard counterparts [TSE+19; SZC+18; RXY+19; Nak19], suggesting that using adversarially robust feature representations should hurt transfer performance.

On the other hand, recent work has found that the feature representations of robust models carry several advantages over those of standard models. For example, adversarially robust representations typically have better-behaved gradients [TSE+19; STT+19;

ZZ19; KCL19] and thus facilitate regularization-free feature visualization [EIS+19b] (cf. Figure 2.1a). Robust representations are also approximately invertible [EIS+19b], meaning that unlike for standard models [MV15; DB16], an image can be approximately reconstructed directly from its robust representation (cf. Figure 2.1b). More broadly, Engstrom et al. [EIS+19b] hypothesize that by forcing networks to be invariant to signals that humans are also invariant to, the robust training objective leads to feature representations that are more similar to what humans use. This suggests, in turn, that adversarial robustness might be a desirable prior from the point of view of transfer learning.



(a) Perceptually aligned gradients

(b) Representation invertibility

Figure 2.1: Adversarially robust (top) and standard (bottom) representations: robust representations allow (a) feature visualization without regularization; (b) approximate image inversion by minimizing distance in representation space. Figures reproduced from Engstrom et al. [EIS+19b].

Experiments. To resolve these two conflicting hypotheses, we use a test bed of 12 standard transfer learning datasets (all the datasets considered in [KSL19] as well as Caltech-256 [GHP07]) to evaluate fixed-feature transfer on standard and adversarially robust ImageNet models. We considere four ResNet-based architectures (ResNet-{18,50}, WideResNet-50-x{2,4}), and train models with varying robustness levels ε for each architecture (for the full experimental setup, see Appendix B.1).

In Figure 2.2, we compare the downstream transfer accuracy of a standard model to that of the best robust model with the same architecture (grid searching over ε[1]). The results indicate that robust networks consistently extract better features for transfer learning than standard networks—this effect is most pronounced on Aircraft, CIFAR-10, CIFAR-100, Food, SUN397, and Caltech-101. Due to computational constraints, we could not train WideResNet-50-4x models at the same number of robustness levels ε, so a coarser grid was used. It is thus likely that a finer grid search over ε would further improve results (we discuss the role of ε in more detail in Section 2.4.3).

[1]To ensure a fair comparison (i.e., that the gains observed are not an artifact of training many random robust models), we first use a set of random seeds to select the best ε level, and then calculate the perfomance for just that ε using a separate set of random seeds.

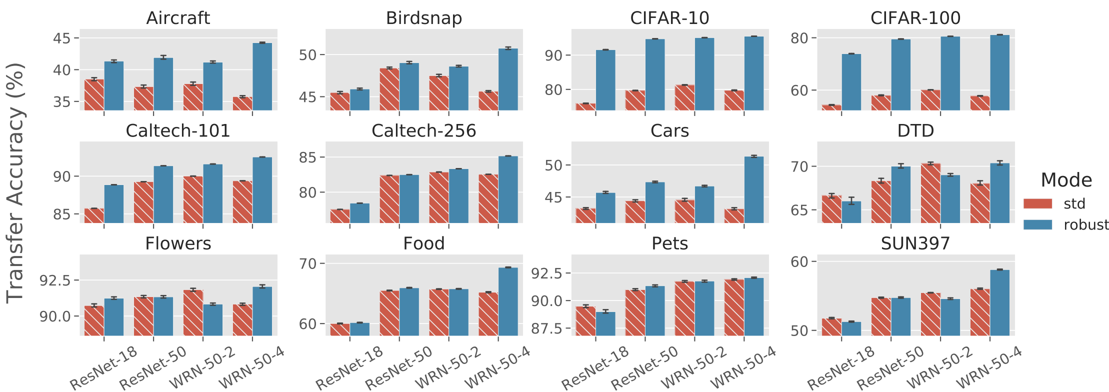


Figure 2.2: **Fixed-feature** transfer learning results using standard and robust models for the 12 downstream image classification tasks considered. Following [KSL19], we record re-weighted accuracy for the unbalanced datasets, and raw accuracy for the others (cf. Appendix B.1). Error bars denote one standard deviation computed over ten random trials.

2.3 Adversarial Robustness and Full-Network Fine Tuning

A more expensive but often better-performing transfer learning method uses the pre-trained model as a weight initialization rather than as a feature extractor. In this "full-network" transfer learning setting, we update all of the weights of the pre-trained model (via gradient descent) to minimize loss on the target task. Kornblith et al. [KSL19] find that for standard models, performance on full-network transfer learning is highly correlated with performance on fixed-feature transfer learning. Therefore, we might hope that the findings of the last section (i.e., that adversarially robust models transfer better) also carry over to this setting. To resolve this conjecture, we consider three applications of full-network transfer learning: image classification (i.e., the tasks considered in Section 2.2), object detection, and instance segmentation.

Downstream image classification We first recreate the setup of Section 2.2: we perform full-network transfer learning to adapt the robust and non-robust pre-trained ImageNet models to the same set of 12 downstream classification tasks. The hyperparameters for training were found via grid search (cf. Appendix B.1). Our findings are shown in Figure 2.3—just as in fixed-feature transfer learning, robust models match or improve on standard models in terms of transfer learning performance.

Object detection and instance segmentation It is standard practice in data-scarce object detection or instance segmentation tasks to initialize earlier model layers with weights from ImageNet-trained classification networks. We study the benefits of using robustly

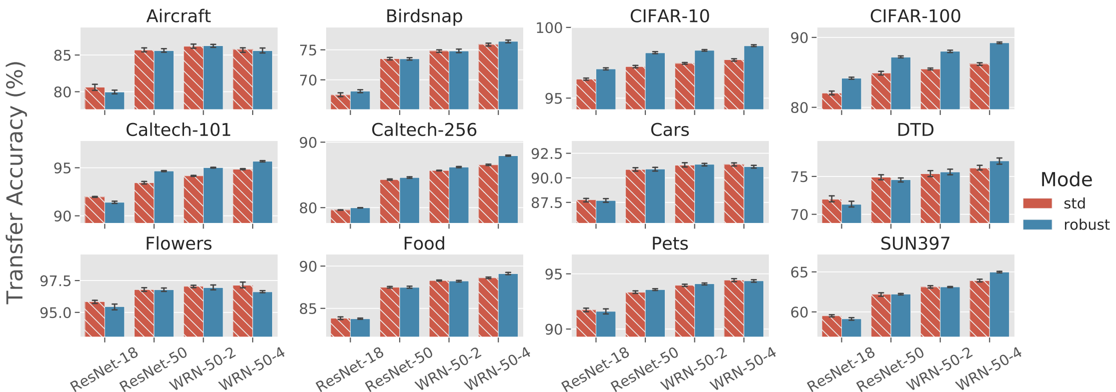


Figure 2.3: **Full-network** transfer learning results using standard and robust models for the 12 downstream image classification tasks considered. Following [KSL19], we record re-weighted accuracy for the unbalanced datasets, and raw accuracy for the others (cf. Appendix B.1). Error bars denote one standard deviation computed with ten random trials.

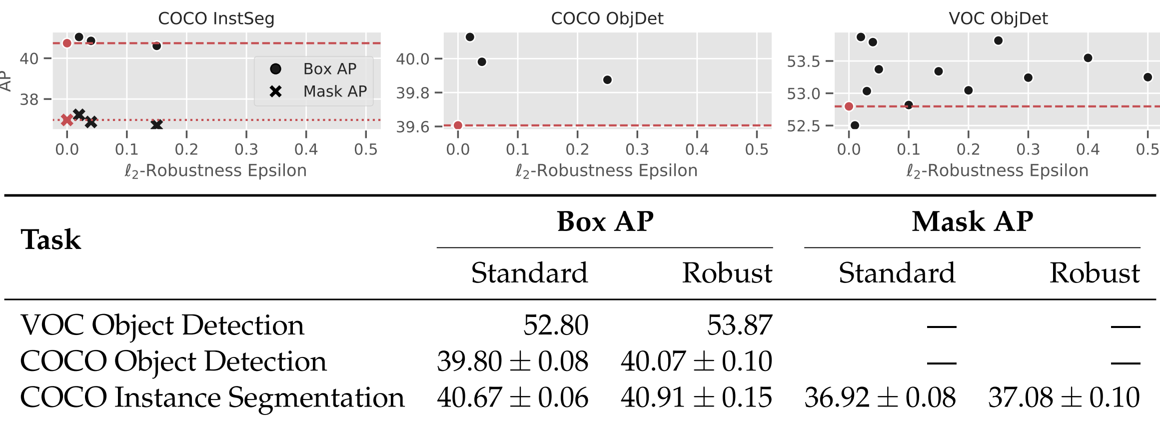
trained networks to initialize object detection and instance segmentation models, and find that adversarially robust networks consistently outperform standard networks.

We evaluate with benchmarks in both object detection (PASCAL Visual Object Classes (VOC) [EVW+10] and Microsoft COCO [LMB+14]) and instance segmentation (Microsoft COCO). We train systems using default models and hyperparameter configurations from the Detectron2 [WKM+19] framework (i.e., we do not perform any additional hyperparameter search). Appendix B.3 describes further experimental details and more results.

We first study object detection. We train Faster R-CNN FPN [LDG+17] models with varying ResNet-50 backbone initializations. For VOC, we initialize with one standard network, and twelve adversarially robust networks with different values of ε. For COCO, we only train with three adversarially robust models (due to computational constraints). For instance segmentation, we train Mask R-CNN FPN models [HGD+17] while varying ResNet-50 backbone initialization. We train three models using adversarially robust initializations, and one model from a standardly trained ResNet-50. Figure 2.4 summarizes our findings: the best robust backbone initializations outperform standard models.

2.4 Analysis and Discussion

Our results from the previous section indicate that robust models match or improve on the transfer learning performance of standard ones. In this section, we take a closer look at the similarities and differences in transfer learning between robust networks and standard networks.



Task	Box AP		Mask AP	
	Standard	Robust	Standard	Robust
VOC Object Detection	52.80	53.87	—	—
COCO Object Detection	39.80 ± 0.08	40.07 ± 0.10	—	—
COCO Instance Segmentation	40.67 ± 0.06	40.91 ± 0.15	36.92 ± 0.08	37.08 ± 0.10

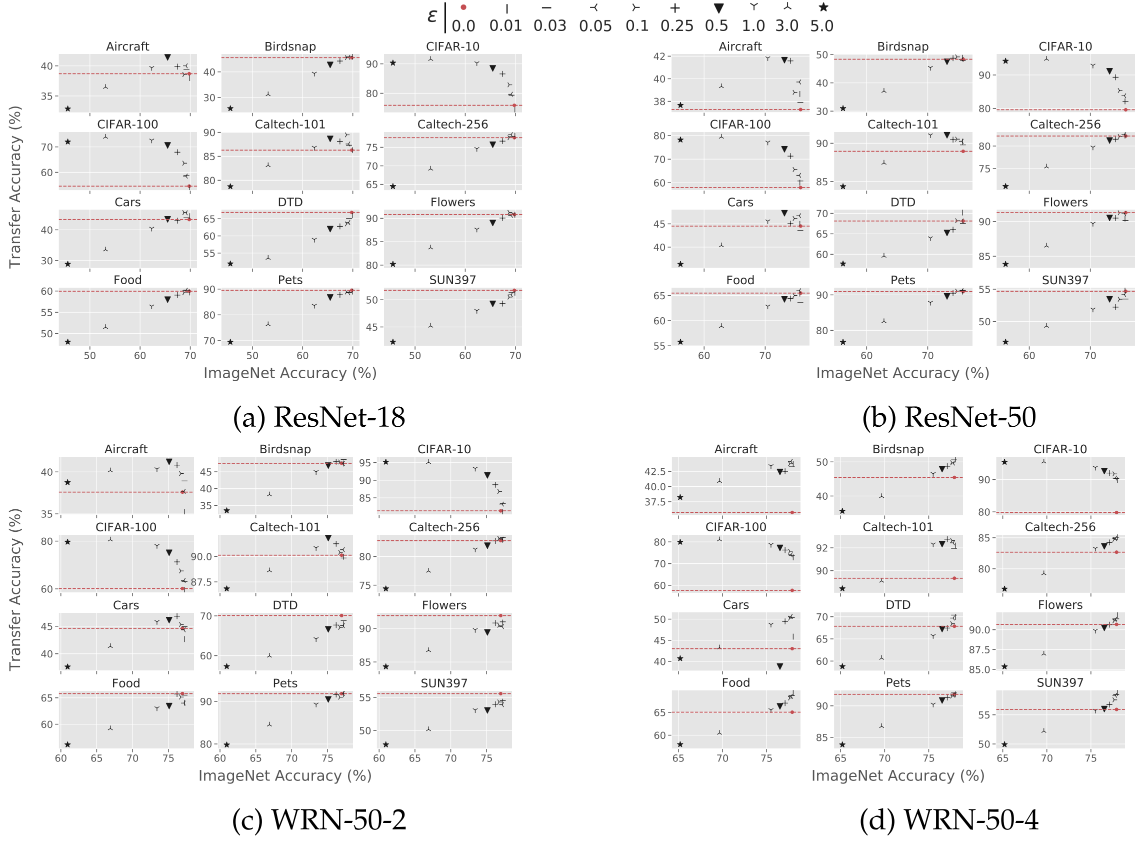
Figure 2.4: AP of instance segmentation and object detection models with backbones initialized with ε-robust models before training. Robust backbones generally lead to better AP, and the best robust backbone always outperforms the standardly trained backbone for every task. COCO results averaged over four runs due to computational constraints; $\pm$ represents standard deviation.

2.4.1 ImageNet accuracy and transfer performance

In Section 2.2, we discussed a potential tension between the desirable properties of robust network representations (which we conjectured would improve transfer performance) and the decreased accuracy of the corresponding models (which, as prior work has established, should hurt transfer performance). We hypothesize that robustness and accuracy have counteracting yet separate effects: that is, higher accuracy improves transfer learning for a fixed level of robustness, and higher robustness improves transfer learning for a fixed level of accuracy.

To test this hypothesis, we first study the relationship between ImageNet accuracy and transfer accuracy for each of the robust models that we trained. Under our hypothesis, we should expect to see a deviation from the direct linear accuracy-transfer relation observed by [KSL19], due to the confounding factor of varying robustness. The results (cf. Figure 2.5; similar results for full-network transfer in Appendix B.5) support this. Indeed, we find that the previously observed linear relationship between accuracy and transfer performance is often violated once robustness aspect comes into play.

In even more direct support of our hypothesis (i.e., that robustness and ImageNet accuracy have opposing yet separate effects on transfer), we find that when the robustness level is held fixed, the accuracy-transfer correlation observed by prior works for standard models actually holds for robust models too. Specifically, we train highly robust ($\varepsilon = 3$)—and thus less accurate—models with six different architectures, and compared ImageNet



(a) ResNet-18 (b) ResNet-50

(c) WRN-50-2 (d) WRN-50-4

Figure 2.5: **Fixed-feature** transfer accuracies of standard and robust ImageNet models to various image classification datasets. The linear relationship between ImageNet and transfer accuracies does not hold. Full numerical results (i.e., in tabular form) are available in Appendix B.6.

accuracy against transfer learning performance. Table 2.2 shows that for these models improving ImageNet accuracy improves transfer performance at around the same rate as (and with higher R^2 correlation than) standard models.

These observations suggest that transfer learning performance can be further improved by applying known techniques that increase the accuracy of robust models (e.g. [BGH19; CRS+19]). More broadly, our findings also indicate that accuracy is not a sufficient measure of feature quality or versatility. Understanding why robust networks transfer particularly well remains an open problem, likely relating to prior work that analyses the features these networks use [EIS+19b; SSZ+19; AL20].

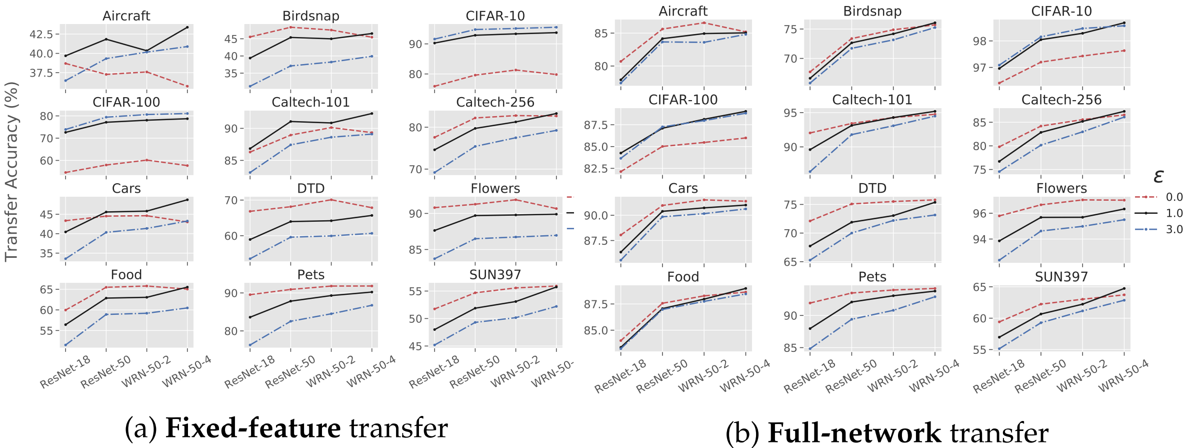
2.4.2 Robust models improve with width

Our experiments also reveal a contrast between robust and standard models in how their transfer performance scales with model width. Azizpour et al. [ARS+15], find that

Table 2.2: Source (ImageNet) and target (CIFAR-10) accuracies, fixing robustness (ε) but varying architecture. When robustness is controlled for, ImageNet accuracy is highly predictive of transfer performance. Similar trends for other datasets are shown in Appendix B.5.

		Architecture (see details in Appendix B.1.1)						
Robustness	**Dataset**	A	B	C	D	E	F	R^2
Std ($\varepsilon = 0$)	ImageNet	77.37	77.32	73.66	65.26	64.25	60.97	—
	CIFAR-10	97.84	97.47	96.08	95.86	95.82	95.55	0.79
Adv ($\varepsilon = 3$)	ImageNet	66.12	65.92	56.78	50.05	42.87	41.03	—
	CIFAR-10	98.67	98.22	97.27	96.91	96.23	95.99	0.97

although increasing network depth improves transfer performance, increasing width hurts it. Our results corroborate this trend for standard networks, but indicate that it does *not* hold for robust networks, at least in the regime of widths tested. Indeed, Figure 2.6 plots results for the three widths of ResNet-50 studied here (x1, x2, and x4), along with a ResNet-18 for reference: as width increases, transfer performance plateaus and decreases for standard models, but continues to steadily grow for robust models. This suggests that scaling network width may further increase the transfer performance gain of robust networks over the standard ones. (This increase comes, however, at a higher computational cost.)



(a) **Fixed-feature** transfer

(b) **Full-network** transfer

Figure 2.6: Varying width and model robustness while transfer learning from ImageNet to various datasets. Generally, as width increases, transfer learning accuracies of standard models generally plateau or level off while those of robust models steadily increase. More values of ε are in Appendix B.5.

2.4.3 Optimal robustness levels for downstream tasks

We observe that although the best robust models often outperform the best standard models, the optimal choice of robustness parameter ε varies widely between datasets. For example, when transferring to CIFAR-10 and CIFAR-100, the optimal ε values were 3.0 and 1.0, respectively. In contrast, smaller values of ε (smaller by an order of magnitude) tend to work better for the rest of the datasets.

One possible explanation for this variability in the optimal choice of ε might relate to dataset granularity. We hypothesize that on datasets where leveraging finer-grained features are necessary (i.e., where there is less norm-separation between classes in the input space), the most effective values of ε will be much smaller than for a dataset where leveraging more coarse-grained features suffices. To illustrate this, consider a binary classification task consisting of image-label pairs (x, y), where the correct class for an image $y \in \{0, 1\}$ is determined by a single pixel, i.e., $x_{0,0} = \delta \cdot y$, and $x_{i,j} = 0$, otherwise. We would expect transferring a standard model onto this dataset to yield perfect accuracy regardless of δ, since the dataset is perfectly separable. On the other hand, a robust model is trained to be invariant to perturbations of norm ε—thus, if $\delta < \varepsilon$, the dataset will not appear separable to the standard model and so we expect transfer to be less successful. So, the smaller the δ (i.e., the larger the "fine grained-ness" of the dataset), the smaller the ε must be for successful transfer.

Unifying dataset scale. We now present evidence in support of our above hypothesis. Although we lack a quantitative notion of granularity (in reality, features are not simply singular pixels), we consider image resolution as a crude proxy. Since we scale target datasets to match ImageNet dimensions, each pixel in a low-resolution dataset (e.g., CIFAR-10) image translates into several pixels in transfer, thus inflating datasets' separability. Drawing from this observation, we attempt to calibrate the granularities of the 12 image classification datasets used in this work, by first downscaling all the images to the size of CIFAR-10 (32×32), and then upscaling them to ImageNet size once more. We then repeat the fixed-feature regression experiments from prior sections, plotting the results in Figure 2.7 (similar results for full-network transfer are presented in Appendix B.5). After controlling for original dataset dimension, the datasets' epsilon vs. transfer accuracy curves all behave almost identically to CIFAR-10 and CIFAR-100 ones. Note that while this experimental data supports our hypothesis, we do not take the evidence as an ultimate one and further exploration is needed to reach definitive conclusions.

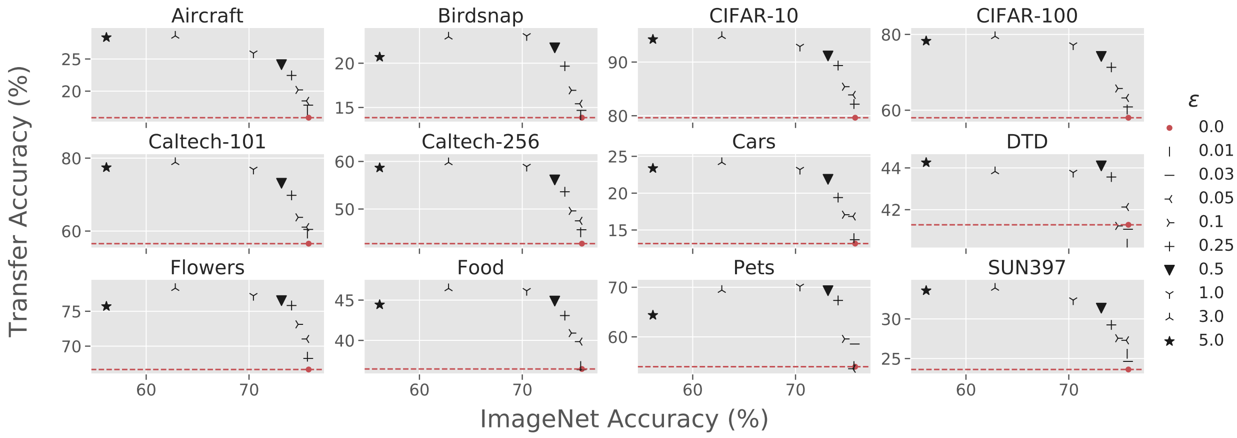
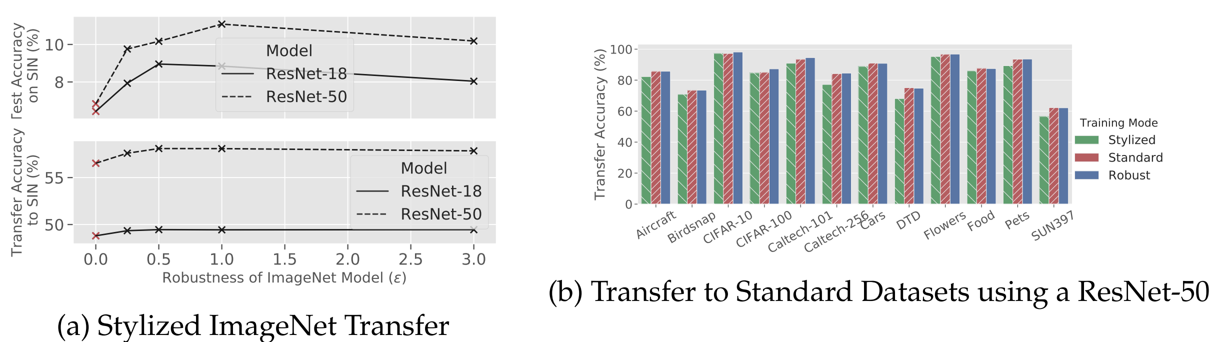


Figure 2.7: **Fixed-feature** transfer accuracies of various datasets that are down-scaled to 32×32 before being up-scaled again to ImageNet scale and used for transfer learning. The accuracy curves are closely aligned, unlike those of Figure 2.5, which illustrates the same experiment without downscaling.

2.4.4 Comparing adversarial robustness to texture robustness



(a) Stylized ImageNet Transfer

(b) Transfer to Standard Datasets using a ResNet-50

Figure 2.8: We compare standard, stylized and robust ImageNet models on standard transfer tasks (and to stylized ImageNet).

We now investigate the effects of adversarial robustness on transfer learning performance in comparison to other invariances commonly imposed on deep neural networks. Specifically, we consider texture-invariant [GRM+19] models, i.e., models trained on the Stylized ImageNet (SIN) [GRM+19] dataset. Figure 2.8b shows that transfer learning from adversarially robust models outperforms transfer learning from texture-invariant models on all considered datasets.

Finally, we use the SIN dataset to further re-inforce the benefits conferred by adversarial robustness. Figure 2.8a top shows that robust models outperform standard imagenet models when evaluated (top) or fine-tuned (bottom) on Stylized-ImageNet.

Chapter 3

Unadversarial examples: Designing objects for robust vision

Up to this point, we have largely viewed adversarial examples as a vulnerability that compromises the reliability of ML systems. However, a different perspective suggests that these examples actually highlight a deeper issue: a fundamental misalignment between humans and ML models. Indeed, Ilyas et al. [IST+19] show that ML models fundamentally rely on *non-robust features* that humans do not. Their results indicate that while such features are extremely brittle, they are still useful in classification. In this chapter and the next one, we demonstrate that we can leverage these—seemingly bad—non robust features to enhance the trustworthiness of ML models. We initiate this line work by illustrating how to employ these non-robust features in designing objects that ML models can easily and robustly recognize, even under distribution shift.

Indeed, performing reliably on unseen or shifting data distributions is a difficult challenge for modern computer vision systems. For example, slight rotations and translations of images suffice to reduce the accuracy of state-of-the-art classifiers [ETT+19; ALG+19; KMF18]. Similarly, models that attain near human-level performance on benchmarks exhibit significantly degraded performance when faced with even mild image corruptions and transformations [HD19; KSH+19]. In fact, when an adversary is allowed to modify inputs directly, standard vision models can be manipulated into predicting arbitrary outputs (cf. *adversarial examples* [BCM+13; SZS+14]). While robustness interventions and additional training data can improve out-of-distribution behavior, they do not fully close the gap between model performance on standard heldout data and on corrupted/otherwise unfamiliar data [TDS+20; HBM+20]. The situation is worse still when test-time distribution is under- or mis-specified, which occurs commonly in practice.

How can we change this state of affairs? We propose a new approach to image recognition in the face of unforeseen corruptions or distribution shifts. This approach is rooted in a reconsideration of the problem setup itself. Specifically, we observe that in many situations, a system designer actually controls, to some extent, the inputs that are fed into that model. For example, a drone operator seeking to train a landing pad detector can modify the surface of the landing pad; and, a roboticist training a perception model to recognize a small set of custom objects can slightly alter the texture or design of these objects.

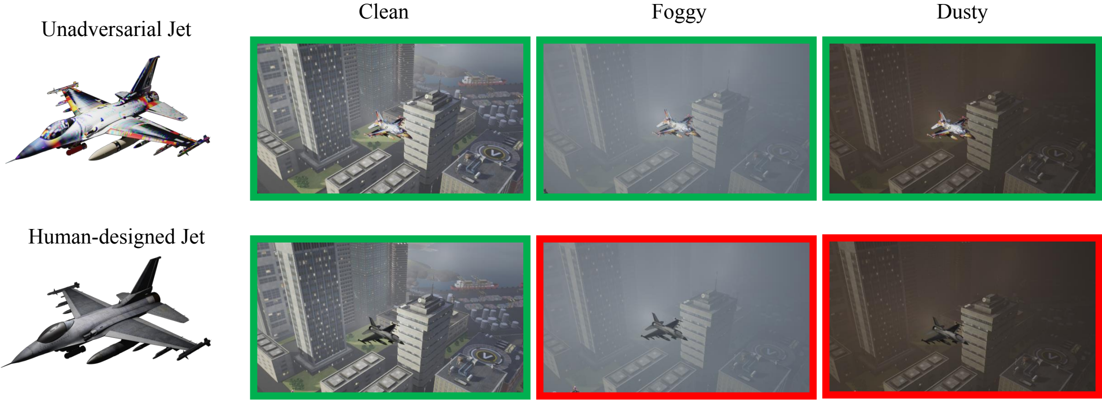


Figure 3.1: We demonstrate that optimizing objects (e.g., the pictured jet) for pre-trained neural networks can boost performance and robustness on computer vision tasks. Here, we show an example of classifying an unadversarial jet and a standard jet using a pretrained ImageNet model. The model correctly classifies the unadversarial jet even under bad weather conditions (e.g., foggy or dusty), whereas it fails to correctly classify the standard jet.

We find that such control over inputs can be leveraged to drastically improve our ability to tackle computer vision tasks. In particular, it allows us to turn the input-sensitivity of modern vision systems from a weakness into a strength. Instead of optimizing inputs to *mislead* models (e.g., as in adversarial examples), we can alter inputs to *reinforce* correct behavior, yielding what we refer to as "unadversarial examples." Indeed, we show that even a simple gradient-based algorithm can successfully construct unadversarial examples in a variety of vision settings and demonstrate that, by optimizing objects for vision systems (rather than vice-versa), we can significantly improve both in-distribution performance *and* robustness to unforeseen data shifts and corruptions.

We demonstrate the efficacy of our methods on both standard benchmarks (CIFAR, ImageNet) and robustness-based benchmarks (ImageNet-C, CIFAR-C) while also comparing them to a broad set of baselines (e.g., QR codes or heuristically designed patches). To further highlight the practicality of our framework, we (a) extend our methods to designing

the texture of three-dimensional objects (rather than patches); (b) deploy unadversarial examples in a simulated drone setting; and (c) ensure that the performance improvement yielded by the objects we design actually transfer to the physical world.

3.1 Motivation and approach

While vision models tend to perform well on held-out data drawn from the same distribution as the training data, out-of-distribution inputs can severely degrade this performance. For example, models behave unreliably under distribution shifts induced by new data collection procedures [RRS+19; EIS+20; TE11], synthetic corruptions [HD19; KSH+19], spatial transformations [ETT+19; FF15], as well as under other types of shift.

Given a fixed type of distribution shift, a standard approach to increasing model robustness is to explicitly train on or regularize with data from the corresponding anticipated test distribution [KSH+19]. For example, Engstrom et al. [ETT+19] find that vision models trained on worst-case rotations and translations end up being fairly robust to rotation and translation-based distribution shifts. However, this approach is not without shortcomings—for example, Kang et al. [KSH+19] find that training CIFAR classification models that are robust to JPEG-compression in this manner requires a significant sacrifice in natural accuracy. Recent works make similar observations in the context of other distribution shift mechanisms like ℓ_p adversaries [TSE+19; SZC+18; RXY+19] or texture swapping [GRM+19].

These observations give rise to a more general question: given that performing reliably in the face of constrained, well-specified distribution shifts is already a difficult challenge, how can we attain robustness to broad, unforeseen distribution shifts?

3.1.1 Leveraging more controlled vision settings

Consider the vision tasks of detecting a landing pad from a drone, or classifying manufacturing components from a factory robot. In both these tasks, reliable in-distribution performance is a necessity; still, a number of possible distribution shifts may occur at deployment time. For example, the drone might approach the landing pad at an atypical angle, or have a view obstructed by snow, smoke, or rain. Similarly, the factory robot may encounter objects in unfamiliar poses, or could be equipped with only a low-quality/noisy camera.

At first glance, dealing with these issues seems to require tackling the difficult problem of general distribution shift robustness discussed earlier in this section. However, there

is in fact a critical distinction between the scenarios considered above and vision tasks in their full generality. In particular, in these scenarios and many others, the system designer has control over the physical objects that the model operates on. For instance, the designer of the drone's landing algorithm could paint the landing pad bright yellow. A machine learning model trained to detect this custom landing pad might then be more effective than a model trained to detect a standard grey pad, especially in low-visibility conditions. Still, the particular choice to paint the landing pad yellow is rather ad hoc, and likely rooted in the way *humans* recognize objects. Meanwhile, an abundance of prior work (e.g., [JBZ+19; GRM+19; JLT18; IST+19]) demonstrates that humans and machine learning models tend to use different sets of features to make their decisions. This suggests that rather than relying on human priors, we should instead be asking: *how can we build objects that are easily detectable by machine learning models?*

3.1.2 Unadversarial examples

The task of making inputs *less* recognizable by computer vision systems has been a focus of research in *adversarial examples*. Adversarial examples are small, carefully constructed perturbations to natural images that can induce arbitrary (mis)behavior from machine learning models [BCM+13; SZS+14]. These perturbations are typically constructed as the result of an optimization problem that maximizes the loss of a machine learning model with respect to the input, i.e., by solving the optimization problem

$$\delta_{adv} = \arg\max_{\delta \in \Delta} L(f_\theta(x+\delta), y), \tag{3.1}$$

where f_θ is a parameterized model (e.g., a neural network with weights θ); x is a natural input; y is the corresponding correct label; L is the loss function used to train θ (e.g., cross-entropy loss) and Δ is a class of permissible perturbations (e.g., norm-bounded perturbations: $\Delta = \{\delta : \|\delta\|_p \leq \epsilon\}$ for some small $\epsilon > 0$). Adversarial perturbations are typically crafted via projected gradient descent (PGD) [Nes03] in input space, a standard iterative first-order optimization method—prior work in adversarial examples has shown that even a few iterations of PGD suffice to completely change the prediction of many state-of-the-art machine learning systems [MMS+18].

From adversarial examples to unadversarial objects. The goal of this work is to modify the design of objects so that they are more easily recognizable by computer vision systems. If we could specify every pixel of every image that a model encounters at test time, we

could draw on the effectiveness of adversarial examples, and construct image perturbations (using PGD) that *minimize* the loss of the system, e.g.,

$$\delta_{unadv} = \arg\min_{\delta \in \Delta} L(\theta; x + \delta, y). \tag{3.2}$$

In our setting of interest, however, having such fine-grained access to the test inputs is unrealistic (presumably, if we had precise control over every pixel in the input, we could just directly encode the ground-truth label directly in it). Instead, we have *limited* control over some physical objects; these objects are in turn captured within images, affected by many signals that are out of our control, such as camera artifacts, weather effects, or background scenery.

It turns out that we can still draw on techniques from adversarial examples research in this limited-control setting. Specifically, a recent line of work [KGB17; SBB+16; EEF+18a; AEI+18] concerns itself with constructing *robust adversarial examples* [AEI+18], i.e., physically realizable objects that act as adversarial examples when introduced into a scene in any one of a variety of ways. For example, Sharif et al. [SBB+16] design glasses frames that cause facial recognition models to misclassify faces, Athalye et al. [AEI+18] design custom-textured 3D models that are misclassified by state-of-the-art ImageNet classifiers from many angles and viewpoints, and [BMR+18] design adversarial patches: stickers that can be placed anywhere on objects causing them to be misclassified. In this chapter, we leverage the techniques developed in the above line of work to construct robust un-adversarial objects—physically realizable *objects* optimized to minimize (rather than maximize) the loss of a target classifier. In the next section, we will more concretely discuss our methods for generating unadversarial objects, then outline our evaluation setup.

3.1.3 Constructing unadversarial objects

In the previous section, we identified a class of scenarios where a system designer can, to some extent, control the objects that a machine learning system model operates on. In these settings, we motivated so-called *unadversarial examples* as a potential way to boost models' overall performance and robustness to distribution shifts. In this section, we present and illustrate two concrete algorithms for constructing unadversarial examples: unadversarial patches and unadversarial textures. In the former, we design a sticker or "patch" [BMR+18] that can be placed on the object; in the latter, we design the 3D texture of the object (in a similar manner to the texture-based adversarial examples of Athalye et al. [AEI+18]). Example results from both techniques are shown in Figure 3.2. For simplicity,

(a) An example unadversarial patch designed for the "tiger" class.

(b) An example unadversarial texture designed for a jet 3D mesh (class "warplane") and applied to rendered city backgrounds.

Figure 3.2: Examples of the two considered methods for constructing unadversarial objects.

we will assume that the task being performed is image classification, but the techniques are directly applicable to other tasks as well. In all cases, we require access to a pre-trained model for the dataset of interest.

Unadversarial patches. To train unadversarial patches (cf. Figure 3.2a), in addition to the pre-trained model, we require sample access to image-label pairs from the dataset of interest. At each iteration, we sample an image-label pair (x, y) from a training set, and place the patch corresponding to class y onto the image with random orientation and position[1]. Since placing the patch is an affine transformation, after each iteration we can compute the gradient of the model's loss with respect to the pixels in the patch, and take a negative gradient step on the patch parameters. The algorithm terminates when the model's loss on sticker-boosted images plateaus, or after a fixed number of iterations.

Unadversarial textures. To train unadversarial *textures* (cf. Figure 3.2b), we do not require sample access to the dataset, but instead a set of 3D meshes for each class of objects that we would like to augment, as well as a set of background images that we can use to simulate sampling a scene (these can be images from the dataset of interest, solid-color backgrounds, random patterns, etc.).

For each 3D mesh, our goal is to optimize a 2D texture which improves classifier performance when mapped onto the mesh. At each iteration, we sample a mesh and a random background; we then use a 3D renderer (Mitsuba [NVZ+19]) to map the corresponding texture onto the mesh. We overlay the rendering onto a random background image, and then feed the resulting composed image into the pre-trained classifier, with the label being that of the sampled 3D mesh. Since rendering is typically non-differentiable, we use a linear approximation of the rendering process (cf. Athalye et al. [AEI+18]) in order to compute (this time approximate) gradients of the model's loss with respect to the utilized

[1] We allow the patch to be placed anywhere as a matter of convenience: ideally we would only be applying the patch onto the main object itself, but this would require bounding box data that we do not have for most classification datasets.

texture. From there, we apply the same SGD algorithm as we did for the patch case.

3.2 Experimental evaluation

In order to determine the effectiveness of our proposed framework, we evaluate against a suite of computer vision tasks. Below, we first provide some detail on the precise access model and baselines that will be considered. We then briefly outline the experimental setup of each task, and show that unadversarial objects consistently improve the performance and robustness of the vision systems tested. For a more detailed account of each experimental setup, see Appendix C.2.

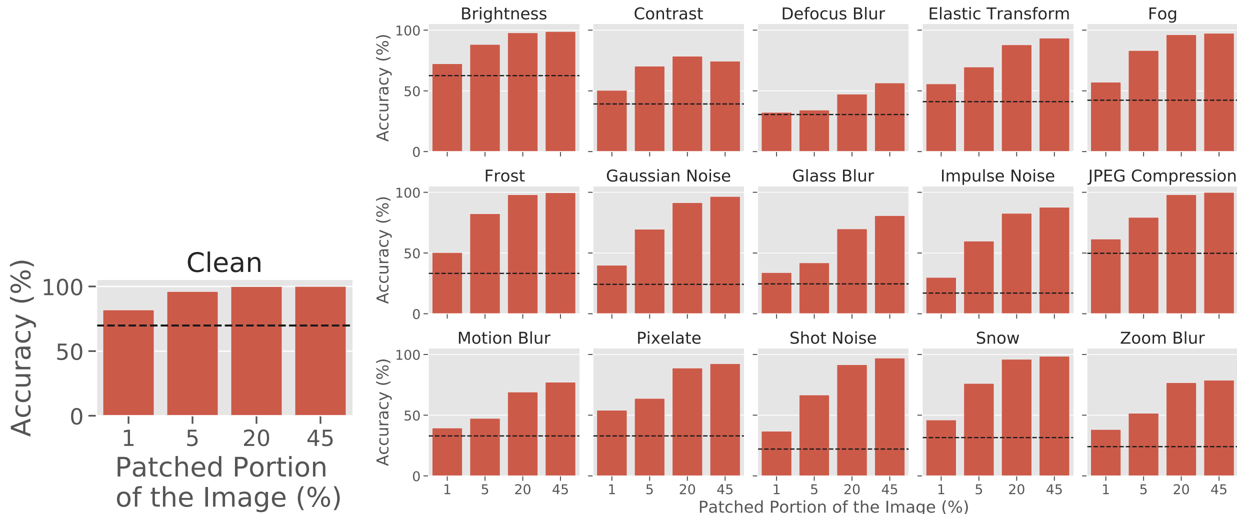
3.2.1 Access model and baselines

In many of the settings discussed thus far, a system designer can alter the objects being recognized but is *not allowed* to alter the the classifier itself. That is, we are optimizing unadversarial objects for a fixed (pre-trained) model. For instance, a road engineer may wish to design road signs that are easier to recognize for autonomous vehicles, without being able to train or alter the machine learning models that operate the vehicles. Similarly, a roboticist might want to design a landing pad that works better for a commercial (pre-trained) drone. We will refer to this setting as the *fixed-model* setting. On the other hand, sometimes the same entity is able to train both the model and the transformations (the discussed factory robot example may fall into this category, for example). In this "free-model" setting, one may be able to boost performance by *co-designing* the machine learning model and the objects of interest.

In this work, we will focus on designing unadversarial examples in the fixed-model setting, for the sake of both simplicity and applicability. In particular, any valid algorithm under a fixed-model assumption is also a valid algorithm under the co-design assumption (but the converse is not true). This leaves the task of leveraging even more control in joint optimization settings as a potential avenue for future work.

Baselines. Since we consider the fixed-model setting throughout our chapter, the only truly comparable baselines are those which do not alter the model being trained. Nonetheless, in order to fully contextualize our results, we will also consider a few baselines that fall outside of our intended access model (e.g., QR codes). A notable disadvantage of these baselines (that we do not explicitly demonstrate below) is that if the "unadversarial signal" (e.g., the QR code) is occluded or removed, the entire system fails; this is in contrast to the

Figure 3.3: Clean (left) and corresponding corrupted (right) ImageNet images augmented with an unadversarial patch—we use such images to evaluate the efficacy of unadversarial patches in Section 3.2.2.



(a) Performance on ImageNet

(b) Performance on synthetically corrupted data (ImageNet-C)

Figure 3.4: Accuracy on (a) clean ImageNet images and (b) synthetically corrupted ImageNet-C images as a function of patch size (given as a percentage of image area). In (b), each bar denotes the average accuracy over the five severities in ImageNet-C, and the horizontal dashed lines report the accuracy on the original (non-patched) datasets. Unadversarial patches consistently boost performance for both clean and corrupted images, with accuracy monotonically increasing with patch size. The patches were trained without any corruptions or non-standard data augmentation in-the-loop (we train with the same augmentations that the pre-trained model itself was trained with).

fixed-model setting, where the pre-trained model is able to recognize objects without any unadversarial signals and is instead "boosted" by their presence.

3.2.2 Clean data and synthetic corruptions

We first test whether unadversarial examples improve the performance of image classifiers on benchmark datasets. Using the algorithm described in Section 3.1.3, we construct unadversarial patches of varying size for pre-trained ResNet-50 classifiers on the CIFAR [Kri09] and ImageNet [RDS+15] datasets. For evaluation, we add these patches at random

positions, scales, and orientations to validation set images (see Appendix C.2 for the exact protocol). As shown in Figure 3.3a, the pre-trained ImageNet classifier is consistently more accurate on the augmented ImageNet images. For example, an unadversarial patch 20 times smaller than ImageNet images boosts accuracy by 26.3% (analogous results for CIFAR are given in Appendix C.3).

Robustness to synthetic corruptions. Next, we use the CIFAR-C and ImageNet-C datasets [HD19] (consisting of the CIFAR and ImageNet validation sets corrupted in 15 systematic ways) to see whether the addition of unadversarial patches to images confers any corruption robustness.

We use the same patches and evaluation protocol that we used when looking at clean data (to ensure a fair evaluation, we apply corruptions to boosted images only *after* the unadversarial patches have been applied). As a consequence, at test time neither model nor patch has been exposed to any image corruptions beyond standard data augmentation. As a result, this experiment tests the ability for unadversarially boosted images to withstand completely unforeseen corruptions; we also avoid any potential biases from training on (and thus "overfitting" to [KSH+19]) a specific type of corruption. The results (cf. Figure 3.3b for ImageNet and Appendix C.3 for CIFAR) indicate that unadversarial patches do improve performance across corruption types; for example, applying an unadversarial patch 5% the size of a standard ImageNet image boosts accuracy by an average of 31.7% points across corruptions [2].

The model does not ignore the image in the presence of unadversarial patches. Recall from our discussion of the fixed-model setting in Section 3.2.1 that an advantage of designing unadversarial objects without changing the model is that the model still works in the absence of the unadversarial signal. We now briefly explore the case where the model is exposed to an unadversarial signal for the *wrong class*. Ideally, we would want the patch to only assist/boost the signal from the original image—in particular, we do not want the patch to make the model totally ignore the contents of the image itself. Thus, in cases where the signal from the image and the patch conflict, we would like the classifier to predict according to the features present in the image more frequently than the class encoded in the unadversarial patch.

[2] Since the original corruption benchmarks proposed by [HD19] are only available as pre-computed JPEGs (for which we cannot apply a patch pre-corruption) or CPU-based Python image operations (which were prohibitively slow), we re-implemented all 15 corruptions as batched GPU operations; we verified that model accuracies on our corruptions mirrored the original CPU counterparts (i.e., within 1% accuracy). For more details about our reimplementation, see our code release.

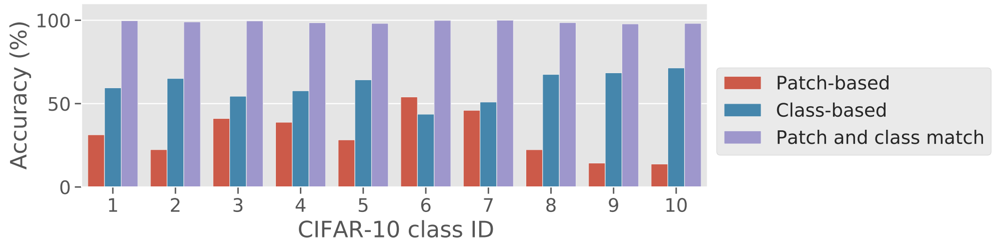


Figure 3.5: The accuracy of a pretrained ResNet-50 on boosted CIFAR-10 when the image and the unadversarial patch used to boost the image have: (1) same class, (2) conflicting classes. For the conflicting classes setting, we report the accuracies based on the class of the patch (red bars), and those based on the class of the image (blue bars). When there is a conflicting signal between the patch and the image, the model relies more on the image.

Indeed, this turns out to be the case. Figure 3.5 shows that on CIFAR-10, when the signal from an unadversarial patch and the image itself conflict, the model predicts according to the patch only 31.2% of the time on average, and according to the image 60.3% of the time (the accuracy of the model when the patch and image agree is 98.93%.)

At first glance, this result may seem to be at odds with the near-perfect effectiveness of *adversarial* patches [BMR+18]. However, the phenomenon we observe here can be tied to the subtle difference between the way we train our unadversarial patches and the way one trains targeted adversarial patches. In the former, we overlay each patch exclusively onto images from its respective class—thus, unadversarial patches are never optimized to be effective when overlaid on a different class. In the latter, however, adversarial patches are optimized to maximize confidence in a particular class on all possible backgrounds, making the patch dominant even when overlaid on an image from a different class.

Baselines. We also compare our results to a variety of natural baselines; the most relevant of these is the "best loss image patch," where we use the minimum-loss training image in place of a patch. We compare with this baseline to ensure that our method is doing something beyond this naive way to add signal to an image. The results are shown in Appendix C.3, along with comparisons to less sophisticated baselines, such as QR Codes and predefined random Gaussian noise patches.

3.2.3 Classification in 3D simulation

We now test unadversarial examples in a more practical setting: recognizing 3D objects in a high-fidelity simulator. We collect meshes corresponding to four ImageNet classes: "warplane," "minibus," "container ship," and "trailer truck," from `sketchfab.com`. We

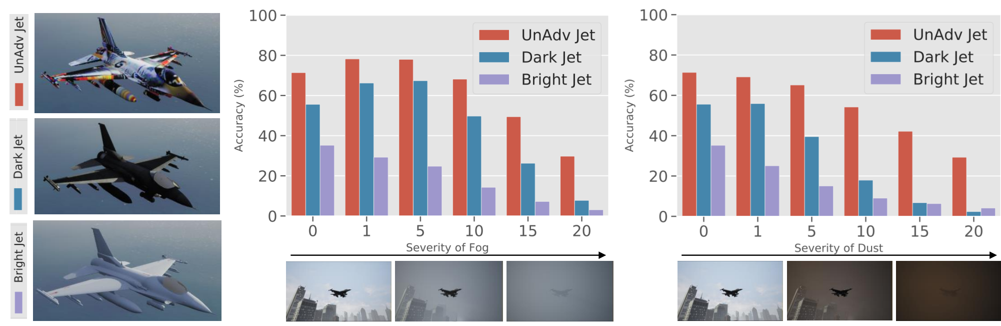
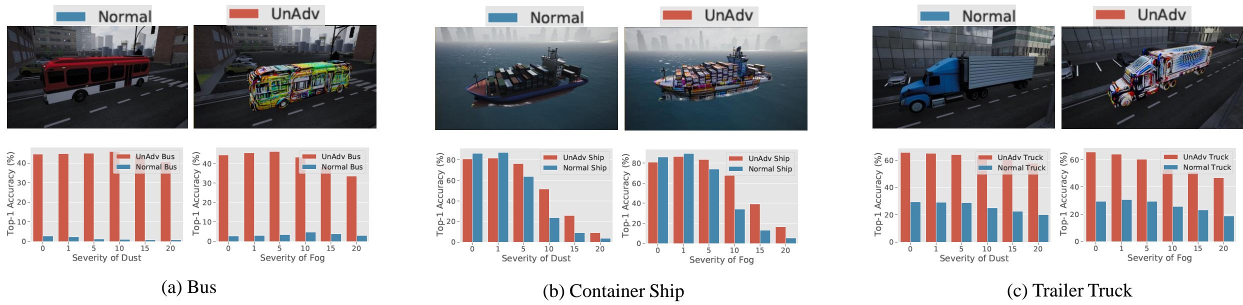


Figure 3.6: The jet unadversarial example task. We show example conditions under which we evaluate the objects, along with aggregate statistics for how well an ImageNet classifier classifies the objects in different conditions. We find that the classifiers perform consistently better on the unadversarial jet texture over the standard jet texture in both standard and distributionally shifted conditions. We also give a baseline of a white jet with a lighter texture because of the poorly visibility inherent in the simulator; we find it performed worse than even the standard jet.



(a) Bus (b) Container Ship (c) Trailer Truck

Figure 3.7: Additional examples reporting aggregate statistics for how well an ImageNet classifier classifies various objects in different conditions. Again, we find that the classifiers perform consistently better on the unadversarial objects texture over the standard objects.

generate a texture for each object using the unadversarial texture algorithm of Section 3.1.3, using the ImageNet validation set as the set of backgrounds for the algorithm, and a pre-trained ResNet-50 as the classifier.

To evaluate the resulting textures, we import each mesh into Microsoft AirSim, a high-fidelity three-dimensional simulator; we then test pre-trained ImageNet models' ability to recognize each object with and without the unadversarial texture applied in a variety of surroundings. We also test each texture's robustness to more realistic weather corruptions (snow and fog) built directly into the simulator (rather than applied as a post-processing step). We provide further detail on AirSim and our usage of it in Appendix C.1. Examples of the images used to evaluate the unadversarial textures, as well as our main results for one of the meshes are shown in Figure 3.6. We find that in both standard and adverse weather conditions, the model consistently performs better on the unadversarial texture

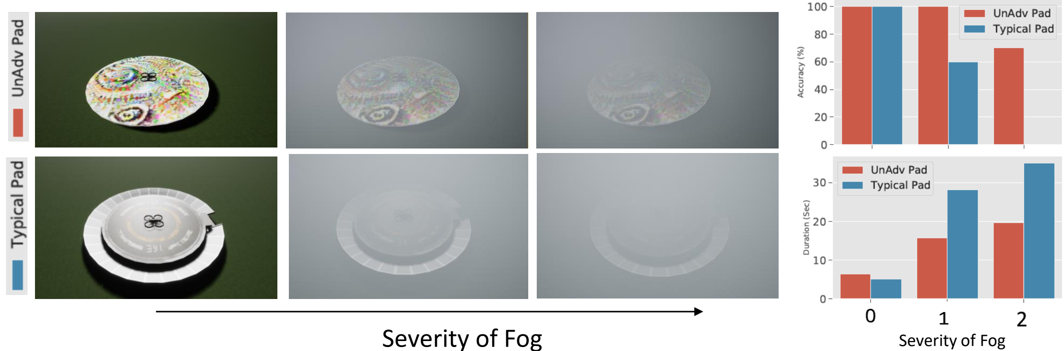


Figure 3.8: Drone landing task. On the left we show the unadversarial versus standard landing pads. On the right we show the results for the task when both the standard and unadversarial landing pads are used. We find that the drone consistently takes less time to land, and has a higher chance of landing correctly, when detecting the unadversarial landing pad.

than on the original. We present similar results for the other three meshes in Figure 3.7.

3.2.4 Localization for (simulated) drone landing

We then assess whether unadversarial examples can help outside of the classification setting. Again using AirSim, we set up a drone landing task with a perception module that receives as input an axis-aligned aerial image of a landing pad, and is tasked with outputing an estimate of the camera's (x, y)-position relative to the pad. While this task is quite basic, we are particularly interested in studying performance in the presence of heavy (simulated) weather-based corruptions. The drone is equipped with a pretrained regression model that localizes the landing pad (described in detail in Appendix C.1). We optimize an unadversarial texture for the surface of the landing pad to best help the drone's regression model in localization. Figure 3.8 shows an example of the landing pad localization task, along with the performance of the unadversarial landing pad compared to the standard pad. The drone landing on the unadversarial pad consistently lands both more reliably.

3.2.5 Physical-world unadversarial examples

Finally, we move out of simulation and test whether the unadversarial patches that we generate can survive naturally-arising distribution shift from effects such as real lighting, camera artifacts, and printing imperfections. We use four household objects (a toy racecar, miniature plane, coffeepot, and eggnog container), and print out (on a standard InkJet

Class	No Patch	Patch
“racer”	22%	83%
“eggnog”	22%	44%
“coffee pot”	39%	56%
“warplane”	67%	83%

(a) Accuracy of pre-trained ResNet-18 on photographs of real world objects with and without patches.

Airplane with patch (83% accuracy)

Car with patch (86% accuracy)

Airplane without patch (67% accuracy)

Car without patch (27% accuracy)

(b) Example photos of the “warplane” and “racer” physical objects taken with (top) and without (bottom) an unadversarial patch.

Figure 3.9: Physical-world experiments. We take pictures of objects at diverse orientations while varying the presence of a patch on the object. Note that we don’t do any additional data augmentation on the patches, which are the same used in our previous ImageNet benchmark experiment.

printer) the adversarial patch corresponding to the label of each object. We take pictures of the toy with and without the patch taped on using an ordinary cellphone camera, and count the number of poses for which the toy is correctly classified by a pre-trained ImageNet classifier. Our results are in Table 3.9a, and examples of patches are in Figure 3.9b. Classifying both patched and unpatched images over a diverse set of poses, we find that the adversarial patches consistently improve performance even at uncommon object orientations.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Raising the cost of malicious AI-based manipulation

Large diffusion models such as DALL·E 2 [RDN+22] and Stable Diffusion [RBL+22] are known for their ability to produce high-quality photorealistic images, and can be used for a variety of image synthesis and editing tasks. However, the ease of use of these models has raised concerns about their potential abuse, e.g., by creating inappropriate or harmful digital content. For example, a malevolent actor might download photos of people posted online and edit them maliciously using an off-the-shelf diffusion model (as in Figure 4.1 top).

How can we address these concerns? First, it is important to recognize that it is, in some sense, impossible to completely eliminate such malicious image editing. Indeed, even without diffusion models in the picture, malevolent actors can still use tools such as Photoshop to manipulate existing images, or even synthesize fake ones entirely from scratch. The key new problem that large generative models introduce is that these actors can now create realistic edited images with *ease*, i.e., without the need for specialized skills or expensive equipment. This realization motivates us to ask:

How can we raise the cost of malicious (AI-powered) image manipulation?

In this chapter, we put forth an approach that aims to alter the economics of AI-powered image editing. At the core of our approach is the idea of image *immunization*—that is, making a specific image resistant to AI-powered manipulation by adding a carefully crafted (imperceptible) perturbation to it. This perturbation would disrupt the operation of a diffusion model, forcing the edits it performs to be unrealistic (see Figure 4.1). In this paradigm, people can thus continue to share their (immunized) images as usual, while getting a layer of protection against undesirable manipulation.

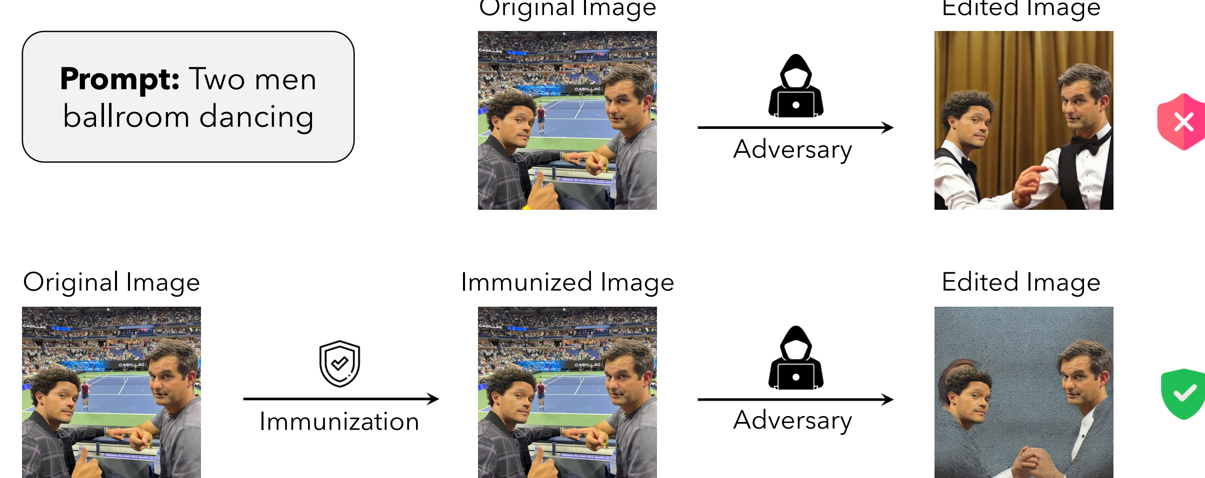


Figure 4.1: *Overview of our framework*. An adversary seeks to modify an image found online. The adversary describes via a textual prompt the desired changes and then uses a diffusion model to generate a realistic image that matches the prompt (top). By immunizing the original image before the adversary can access it, we disrupt their ability to successfully perform such edits (bottom).

We demonstrate how one can craft such imperceptible perturbations for large-scale diffusion models and show that they can indeed prevent realistic image editing. We then discuss in Section 4.4 complementary technical and policy components needed to make our approach fully effective and practical. Finally, this presents another venue (besides designing robust objects in the previous chapter) where adversarial perturbations turn out to be useful.

4.1 Preliminaries

We start by providing an overview of diffusion models as well as of the key concept we will leverage: adversarial attacks.

4.1.1 Diffusion Models

Diffusion models have emerged recently as powerful tools for generating realistic images [SWM+15; HJA20]. These models excel especially at generating and editing images using textual prompts, and currently surpass other image generative models such as GANs [GPM+14] in terms of the quality of produced images.

Diffusion process. At their core, diffusion models employ a stochastic differential process called the *diffusion process* [SWM+15]. This process allows us to view the task of (approximate) sampling from a distribution of real images $q(\cdot)$ as a series of *denoising* problems. More precisely, given a sample $\mathbf{x}_0 \sim q(\cdot)$, the diffusion process incrementally adds noise to generate samples $\mathbf{x}_1, \ldots, \mathbf{x}_T$ for T steps, where $\mathbf{x}_{t+1} = a_t \mathbf{x}_t + b_t \boldsymbol{\varepsilon}_t$, and $\boldsymbol{\varepsilon}_t$ is sampled from a Gaussian distribution[1]. Note that, as a result, the sample $\mathbf{x}_T$ starts to follow a standard normal distribution $\mathcal{N}(0, \mathbf{I})$ when $T \to \infty$. Now, if we reverse this process and are able to sample $\mathbf{x}_t$ given $\mathbf{x}_{t+1}$, i.e., *denoise* $\mathbf{x}_{t+1}$, we can ultimately generate new samples from $q(\cdot)$. This is done by simply starting from $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$ (which corresponds to T being sufficiently large), and iteratively denoising these samples for T steps, to produce a new image $\tilde{\mathbf{x}} \sim q(\cdot)$.

The element we need to implement this process is thus to learn a neural network $\boldsymbol{\varepsilon}_\theta$ that "predicts" given $\mathbf{x}_{t+1}$ the noise $\boldsymbol{\varepsilon}_t$ added to $\mathbf{x}_t$ at each time step t. Consequently, this *denoising model* $\boldsymbol{\varepsilon}_\theta$ is trained to minimize the following loss function:

$$\mathcal{L}(\theta) = \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\varepsilon} \sim \mathcal{N}(0,1)} \left[\| \boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_\theta(\mathbf{x}_{t+1}, t) \|_2^2 \right], \tag{4.1}$$

where t is sampled uniformly over the T time steps. We defer discussion of details to Appendix D.2 and refer the reader to [Wen21] for a more in-depth treatment of diffusion models.

Latent diffusion models (LDMs). Our focus will be on a specific class of diffusion models called the *latent diffusion models* (LDMs) [RBL+22][2]. These models apply the diffusion process described above in the *latent space* instead of the input (image) space. As it turned out, this change enables more efficient training and faster inference, while maintaining high quality generated samples.

Training an LDM is similar to training a standard diffusion model and differs mainly in one aspect. Specifically, to train an LDM, the input image $\mathbf{x}_0$ is first mapped to its latent representation $\mathbf{z}_0 = \mathcal{E}(\mathbf{x}_0)$, where $\mathcal{E}$ is a given encoder. The diffusion process then continues as before (just in the *latent* space) by incrementally adding noise to generate samples $\mathbf{z}_1, \ldots, \mathbf{z}_T$ for T steps, where $\mathbf{z}_{t+1} = a_t \mathbf{z}_t + b_t \boldsymbol{\varepsilon}_t$, and $\boldsymbol{\varepsilon}_t$ is sampled from a Gaussian distribution. Finally, the denoising network $\boldsymbol{\varepsilon}_\theta$ is then learned analogously to as before but, again, now in the latent space, by minimizing the following loss function:

[1] Here, a_t and b_t are the parameters of the distribution $q(\mathbf{x}_{t+1} | \mathbf{x}_t)$. Details are provided in Appendix D.2.

[2] Our methodology can be adjusted to other diffusion models. Our focus on LDMs is motivated by the fact that all popular open-sourced diffusion models are of this type.

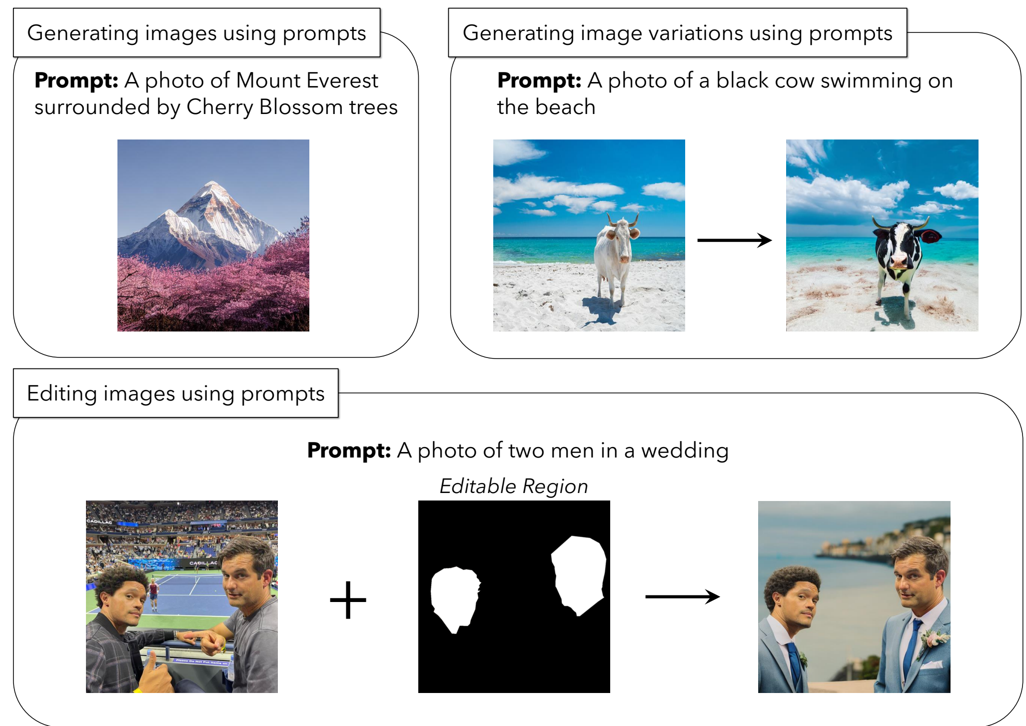


Figure 4.2: Diffusion models offer various capabilities, such as (1) generating images using textual prompts (top left), (2) generating variations of an input image using textual prompts (top right), and (3) editing images using textual prompts (bottom).

$$\mathcal{L}(\theta) = \mathbb{E}_{t,\mathbf{z}_0,\varepsilon\sim\mathcal{N}(0,1)}\left[\|\varepsilon - \varepsilon_\theta(\mathbf{z}_{t+1}, t)\|_2^2\right] \tag{4.2}$$

Once the denoising network ε_θ is trained, the same generative process can be applied as before, starting from a random vector in the latent space, to obtain a latent representation $\tilde{\mathbf{z}}$ of the (new) generated image. This representation is then decoded into an image $\tilde{\mathbf{x}} = \mathcal{D}(\tilde{\mathbf{z}}) \sim q(\cdot)$, using the corresponding decoder $\mathcal{D}$.

Prompt-guided sampling using an LDM. An LDM by default generates a random sample from the distribution of images $q(\cdot)$ it was trained on. However, it turns out one can also guide the sampling using natural language. This can be accomplished by combining the latent representation $\mathbf{z}_T$ produced during the diffusion process with the embedding of the user-defined textual prompt t.[3] The denoising network ε_θ is applied to the combined representation for T steps, yielding $\tilde{\mathbf{z}}$ which is then mapped to a new image using the decoder $\mathcal{D}$ as before.

[3]Conditioning on the text embedding happens at every stage of the generation process. See [RBL+22] for more details.

LDMs capabilities. LDMs turn out to be powerful text-guided image generation and editing tools. In particular, LDMs can be used not only for generating images using textual prompts, as described above, but also for generating textual prompt–guided variations of an image or edits of a specific part of an image (see Figure 4.2). The latter two capabilities (i.e., generation of image variations and image editing) requires a slight modification of the generative process described above. Specifically, to modify or edit a given image $\mathbf{x}$, we condition the generative process on this image. That is, instead of applying, as before, our generative process of T denoising steps to a random vector in the latent space, we apply it to the latent representation obtained from running the latent diffusion process on our image $\mathbf{x}$. To edit only part of the image we additionally condition the process on freezing the parts of the image that were to remain unedited.

4.1.2 Adversarial Attacks

For a given computer vision model and an image, an *adversarial example* is an imperceptible perturbation of that image that manipulates the model's behavior [SZS+14; BCM+13]. In image classification, for example, an adversary can construct an adversarial example for a given image $\mathbf{x}$ that makes it classified as a specific target label y_{targ} (different from the true label). This construction is achieved by minimizing the loss of a classifier f_θ with respect to that image:

$$\delta_{adv} = \arg\min_{\delta \in \Delta} \mathcal{L}(f_\theta(\mathbf{x} + \delta), y_{targ}). \tag{4.3}$$

Here, Δ is a set of perturbations that are small enough that they are imperceptible—a common choice is to constrain the adversarial example to be close (in ℓ_p distance) to the original image, i.e., $\Delta = \{\delta : \|\delta\|_p \leq \epsilon\}$. The canonical approach to constructing an adversarial example is to solve the optimization problem (4.3) via projected gradient descent (PGD) [Nes03; MMS+18].

4.2 Adversarially Attacking Latent Diffusion Models

We now describe our approach to immunizing images, i.e., making them harder to manipulate using latent diffusion models (LDMs). At the core of our approach is to leverage techniques from the adversarial attacks literature [SZS+14; MMS+18; AMK+21] and add adversarial perturbations (see Section 4.1.1) to immunize images. Specifically, we present two different methods to execute this strategy (see Figure 4.3): an *encoder attack*, and a

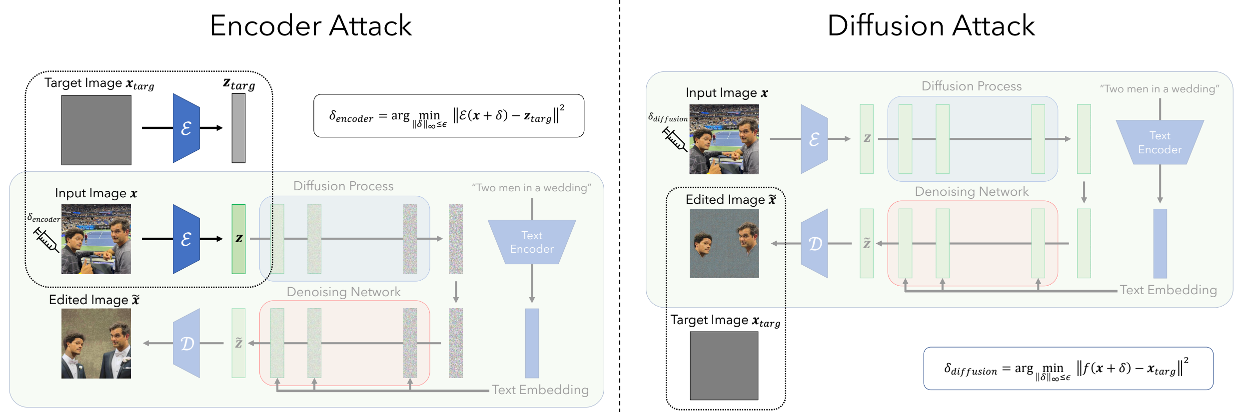


Figure 4.3: *Overview of our proposed attacks*. When applying the *encoder* attack (left), our goal is to map the representation of the original image to the representation of a target image (gray image). Our (more complex) *diffusion* attack (right), on the other hand, aims to break the diffusion process by manipulating the whole process to generate image that resembles a given target image (gray image).

diffusion attack.

Encoder attack. Recall that an LDM, when applied to an image, first encodes the image using an encoder $\mathcal{E}$ into a latent vector representation, which is then used to generate a new image (see Section 4.1). The key idea behind our encoder attack is now to disrupt this process by forcing the encoder to map the input image to some "bad" representation. To achieve this, we solve the following optimization problem using projected gradient descent (PGD):

$$\delta_{encoder} = \arg \min_{\|\delta\|_\infty \le \epsilon} \|\mathcal{E}(\mathbf{x} + \delta) - \mathbf{z}_{targ}\|_2^2, \tag{4.4}$$

where $\mathbf{x}$ is the image to be immunized, and $\mathbf{z}_{targ}$ is some target latent representation (e.g., $\mathbf{z}_{targ}$ can be the representation, produced using encoder $\mathcal{E}$, of a gray image). Solutions to this optimization problem yield small, imperceptible perturbations $\delta_{encoder}$ which, when added to the original image, result in an (immunized) image that is similar to the (gray) target image from the LDM's encoder perspective. This, in turn, causes the LDM to generate an irrelevant or unrealistic image. An overview of this attack is shown in Figure 4.3 (left)[4].

[4]See Algorithm 4 in Appendix for the details of the encoder attack.

Diffusion attack. Although the encoder attack is effective at forcing the LDM to generate images that are unrelated to the immunized ones, we still expect the LDM to use the textual prompt. For example, as shown in the encoder attack diagram in Figure 4.3, editing an immunized image of two men using the prompt "Two men in a wedding" still results in a generated image with two men wearing wedding suits, even if the image will contain some visual artifacts indicating that it has been manipulated. Can we disturb the diffusion process even further so that the diffusion model "ignores" the textual prompt entirely and generates a more obviously manipulated image?

It turns out that we are able to do so by using a more complex attack, one where we target the diffusion process itself instead of just the encoder. In this attack, we perturb the input image so that the *final* image generated by the LDM is a specific target image (e.g., random noise or gray image). Specifically, we generate an adversarial perturbation $\delta_{diffusion}$ by solving the following optimization problem (again via PGD):

$$\delta_{diffusion} = \arg \min_{\|\delta\|_\infty \leq \epsilon} \|f(\mathbf{x} + \delta) - \mathbf{x}_{targ}\|_2^2. \tag{4.5}$$

Above, f is the LDM, $\mathbf{x}$ is the image to be immunized, and $\mathbf{x}_{targ}$ is the target image to be generated. An overview of this attack is depicted in Figure 4.3 (right)[5]. As we already mentioned, this attack targets the full diffusion process (which includes the text prompt conditioning), and tries to nullify not only the effect of the immunized image, but also that of the text prompt itself. Indeed, in our example (see Figure 4.3 (right)) no wedding suits appear in the edited image whatsoever.

It is worth noting that this approach, although more powerful than the encoder attack, is harder to execute. Indeed, to solve the above problem (4.5) using PGD, one needs to backpropagate through the full diffusion process (which, as we recall from Section 4.1.1, includes repeated application of the denoising step). This causes memory issues even on the largest GPU we used[6]. To address this challenge, we backpropagate through only a few steps of the diffusion process, instead of the full process, while achieving adversarial perturbations that are still effective. We defer details of our attacks to Appendix D.1.

4.3 Results

In this section, we examine the effectiveness of our proposed immunization method.

[5]See Algorithm 5 in Appendix for the details of the diffusion attack.
[6]We used an A100 with 40 GB memory.

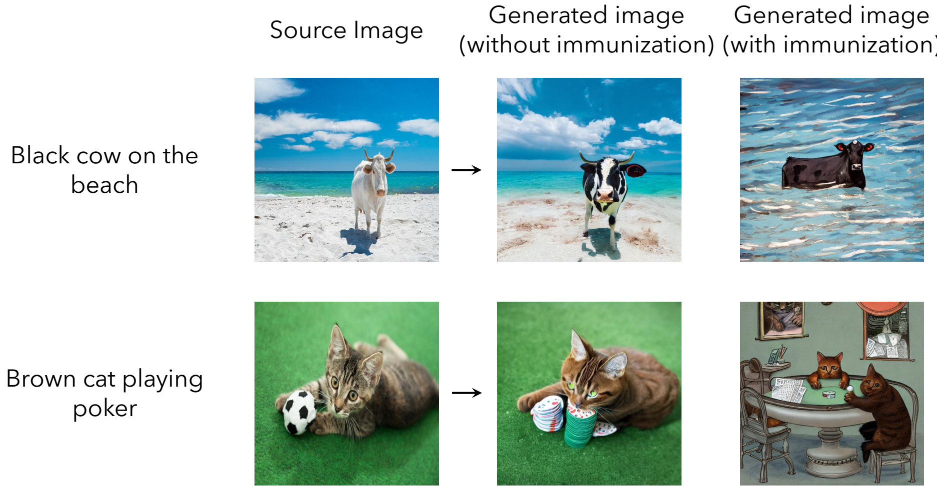


Figure 4.4: Given a source image (e.g., image of a white cow on the beach) and a textual prompt (e.g., "black cow on the beach"), the SDM can generate a realistic image matching the prompt while still similar to the original image (middle column). However, when the source image is immunized, the SDM fails to do so (right-most column). More examples are in Appendix D.3.

Setup. We focus on the Stable Diffusion Model (SDM) v1.5 [RBL+22], though our methods can be applied to other diffusion models too. In each of the following experiments, we aim to disrupt the performance of SDM by adding imperceptible noise (using either of our proposed attacks)—i.e., applying our immunization procedure—to a variety of images. The goal is to force the model to generate images that are unrealistic and unrelated to the original (immunized) image. We evaluate the performance of our method both qualitatively (by visually inspecting the generated images) and quantitatively (by examining the image quality using standard metrics). We defer further experimental details to Appendix D.1.

4.3.1 Qualitative Results

Immunizing against generating image variations. We first assess whether we can disrupt the SDM's ability to generate realistic variations of an image based on a given textual prompt. For example, given an image of a white cow on the beach and a prompt of "black cow on the beach", the SDM should generate a realistic image of a *black* cow on the beach that looks similar to the original one (cf. Figure 4.4). Indeed, the SDM is able to generate such images. However, when we immunize the original images (using the encoder attack), the SDM fails to generate a realistic variation—see Figure 4.4.

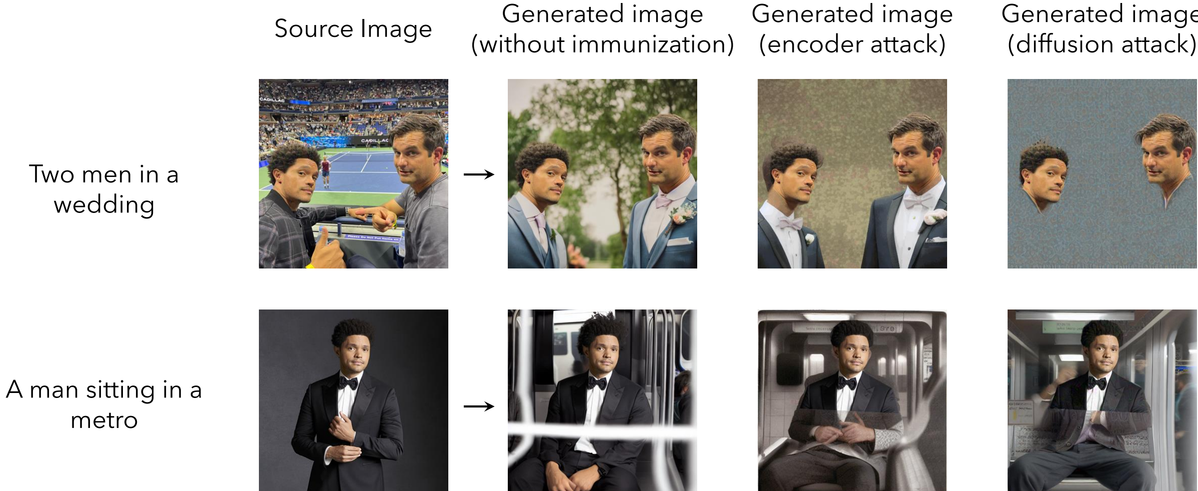


Figure 4.5: Given a source image (e.g., image of two men watching a tennis game) and a textual prompt (e.g., "two men in a wedding"), the SDM can edit the source image to match the prompt (second column). However, when the source image is immunized using the *encoder attack*, the SDM fails to do so (third column). Immunizing using the *diffusion attack* further reduces the quality of the edited image (forth column). More examples are in Appendix D.3.

Immunizing against image editing. Now we consider the more challenging task of disrupting the ability of SDMs to edit images using textual prompts. The process of editing an image using an SDM involves inputting the image, a mask indicating which parts of the image should be edited, and a text prompt guiding how the rest of the image should be manipulated. The SDM then generates an edited version based on that prompt. An example can be seen in Figure 4.2, where an image of two men watching a tennis game is transformed to resemble a wedding photo. This corresponded to inputting the original image, a binary mask excluding from editing only the men's heads, and the prompt "A photo of two men in a wedding." However, when the image is immunized (using either encoder or diffusion attacks), the SDM is unable to produce realistic image edits (cf. Figure 4.5). Furthermore, the diffusion attack results in more unrealistic images than the encoder attack.

4.3.2 Quantitative Results

Image quality metrics. Figures 4.4 and 4.5 indicate that, as desired, edits of immunized images are noticeably different from those of non-immunized images. To quantify this difference, we generate 60 different edits of a variety of images using different prompts, and then compute several metrics capturing the similarity between resulting edits of

Method	FID ↓	PR ↑	SSIM ↑	PSNR ↑	VIFp ↑	FSIM ↑
Immunization baseline (Random noise)	82.57	1.00	0.75 ± 0.13	19.21 ± 4.00	0.43 ± 0.13	0.83 ± 0.08
Immunization (Encoder attack)	130.6	0.95	0.58 ± 0.11	14.91 ± 2.78	0.30 ± 0.10	0.73 ± 0.08
Immunization (Diffusion attack)	**167.6**	**0.87**	$\mathbf{0.50 \pm 0.09}$	$\mathbf{13.58 \pm 2.23}$	$\mathbf{0.24 \pm 0.09}$	$\mathbf{0.69 \pm 0.06}$

Table 4.1: We report various image quality metrics measuring the similarity between edits originating from immunized vs. non-immunized images. We observe that edits of immunized images are substantially different from those generated from the original (notn-immunized) images. Note that the arrows next to the metrics denote increasing image similarity. Since our goal is to make the edits as different as possible from the original edits in the presence of no immunization, then lower image similarity is better. Confidence intervals denote one standard deviation over 60 images. Additional metrics are in Appendix D.3.1.

immunized versus non-immunized images[7]: FID [HRU+17], PR [SBL+18], SSIM [WBS+04], PSNR, VIFp [SB06], and FSIM [ZZM+11][8]. The better our immunization method is, the less similar the edits of immunized images are to those of non-immunized images.

The similarity scores, shown in Table 4.1, indicate that applying either of our immunization methods (encoder or diffusion attacks) indeed yields edits that are different from those of non-immunized images (since, for example, FID is far from zero for both of these methods). As a baseline, we consider a naive immunization method that adds uniform random noise (of the same intensity as the perturbations used in our proposed immunization method). This method, as we verified, is not effective at disrupting the SDM, and yields edits almost identical to those of non immunized images. Indeed, in Table 4.1, the similarity scores of this baseline indicate closer edits to non-immunized images compared to both of our attacks.

Image-prompt similarity. To further evaluate the quality of the generated/edited images after immunization (using diffusion attack), we measure the similarity between the edited images and the textual prompt used to guide this edit, with and without immunization. The fact that the SDM uses the textual prompt to guide the generation of an image indicates that the similarity between the generated image and the prompt should be high in the case of no immunization. However, after immunization (using the diffusion attack), the similarity should be low, since the immunization process disrupts the full diffusion process, and forces the diffusion model to ignore the prompt during generation. We use the same 60 edits as in our previous experiment, and we extract—using a pretrained CLIP model [RKH+21]—the visual embeddings of these images and the textual prompts used

[7]We use the implementations provided in: `https://github.com/photosynthesis-team/piq`.
[8]We report additional metrics in Appendix D.3.1.

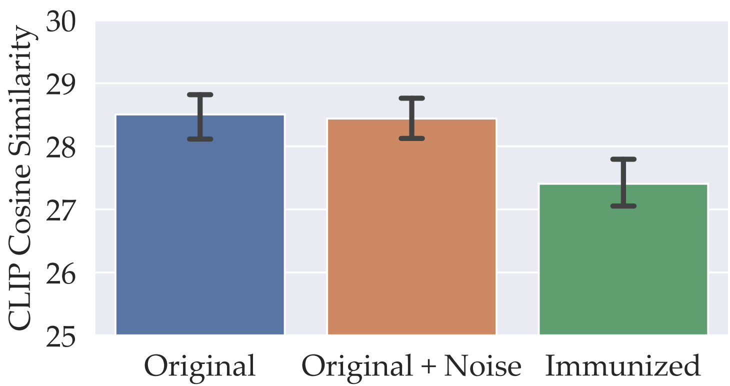


Figure 4.6: *Image-prompt similarity*. We plot the cosine similarity between the CLIP embeddings of the generated images and the text prompts, with and without immunization, as well as with a baseline immunization of adding small random noise to the original image. Error bars denote the interquartile range (IQR) over 60 runs.

to generate them. We then compute the cosine similarity between these two embeddings. As show in Figure 4.6, the immunization process decreases the similarity between the generated images and the textual prompts to generated them, as expected.

4.4 A Techno-Policy Approach to Mitigation of AI-Powered Editing

In the previous sections we have developed an immunization procedure that, when applied to an image, protects the immunized version of that image from realistic manipulation by a given diffusion model. Our immunization procedure has, however, certain important limitations. We now discuss these limitations as well as a combination of technical and policy remedies needed to obtain a fully effective approach to raising the cost of malicious AI-powered image manipulation.

(Lack of) robustness to transformations. One of the limitations of our immunization method is that the adversarial perturbation that it relies on may be ineffective after the immunized image is subjected to image transformations and noise purification techniques. For instance, malicious actors could attempt to remove the disruptive effect of that perturbation by cropping the image, adding filters to it, applying a rotation, or other means. This problem can be addressed, however, by leveraging a long line of research on creating *robust* adversarial perturbations, i.e., adversarial perturbations that can withstand a broad range of image modifications and noise manipulations [EEF+18b; KGB16; AEI+18; BMR+18].

Forward-compatibility of the immunization. While the immunizing adversarial perturbations we produce might be effective at disrupting the current generation of diffusion-based generative models, they are not guaranteed to be effective against the future versions of these models. Indeed, one could hope to rely here on the so-called *transferability* of adversarial perturbations [PMG16; LCL+17], but *no* perturbation will be perfectly transferable.

To truly address this limitation, we thus need to go beyond purely technical methods and encourage—or compel—via policy means a collaboration between organizations that develop large diffusion models, end-users, as well as data hosting and dissemination platforms. Specifically, this collaboration would involve the developers providing APIs that allow the users and platforms to immunize their images against manipulation by the diffusion models the developers create. Importantly, these APIs should guarantee "forward compatibility", i.e., effectiveness of the offered immunization against models developed in the future. This can be accomplished by planting, when training such future models, the current immunizing adversarial perturbations as backdoors. (Observe that our immunization approach can provide *post-hoc* "backward compatibility" too. That is, one can create immunizing adversarial perturbations that are effective for models that were *already released*.)

It is important to point out that we are leveraging here an incentive alignment that is fundamentally different to the one present in more typical applications of adversarial perturbations and backdoor attacks. In particular, the "attackers" here—that is, the parties that create the adversarial perturbations/execute the backdoor attack—are the same parties that develop the models being attacked. This crucial difference is, in particular, exactly what helps remedy the forward compatibility challenges that turns out to be crippling, e.g., in the context of "unlearnable" images creation (i.e., creation of images that are immune to being leveraged by, e.g., facial recognition models) [RHC+21].

Part II

Understanding the underpinnings of reliable ML deployment

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Model debugging and the missingness bias

In the first part of this thesis, we focused on enhancing the reliability of ML models through increased robustness, performance, and trustworthiness. Despite these improvements, real-world deployment introduces complexities that necessitate a deeper understanding of these models' decision-making mechanisms, especially in safety-critical environments. Consequently, the second part of this thesis sets out to provide methods for dissecting when, why, and how modern ML systems either succeed or fail.

Model debugging aims to diagnose a model's failures. For example, researchers can identify global biases of models via the extraction of human-aligned concepts [BZK+17; WSM21], or understand the texture bias by analyzing the models performance on synthetic datasets [GRM+19; LSI+21]. Other approaches aim to highlight local features to debug individual model predictions [SVZ13; DCL+18; RSG16a; GWE+19].

A common theme in these methods is to compare the behavior of the model *with and without* certain individual features [RSG16a; GWE+19; FV17; DG17; ZCA+17; DCL+18; CCG+19]. For example, interpretability methods such as LIME [RSG16b] and integrated gradients [STY17] use the predictions when certain features are removed from the input to attribute different regions of the input to the decision of the model. Dhurandhar et al. [DCL+18] find minimal regions in radiology images that are necessary for classifying a person as having autism. Fong and Vedaldi [FV17] propose learning image masks that minimize a class score to achieve interpretable explanations. Similarly, in natural language processing, model designers often remove individual words to understand their importance to the output [MG21; LCH+16]. The absence of features from an input, a concept sometimes referred to as *missingness* [SLL20], is thus fundamental to many

debugging tools.

However, there is a problem: while we can easily remove words from sentences, removing objects from images is not as straightforward. Indeed, removing a feature from an image usually requires approximating missingness by replacing those pixel values with something else, e.g., black color. However, these approximations tend not to be perfect [SLL20]. Our goal is thus to give a holistic understanding of missingness and, specifically, to answer the question:

How do missingness approximations affect our ability to debug ML models?

Our contributions

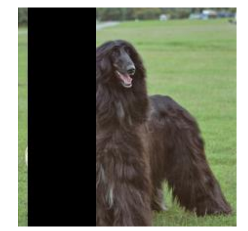
In this chapter, we investigate how current missingness approximations, such as blacking out pixels, can result in what we call *missingness bias*. This bias turns out to hinder our ability to debug models. We then show how transformer-based architectures can enable a more natural implementation of missingness, allowing us to side-step this bias. More specifically, our contributions include:

Pinpointing the missingness bias. We demonstrate at multiple granularities how simple approximations, such as blacking out pixels, can lead to missingness bias. This bias skews the overall output distribution toward unrelated classes, disrupts individual predictions, and hinders the model's use of the remaining (unmasked) parts of the image.

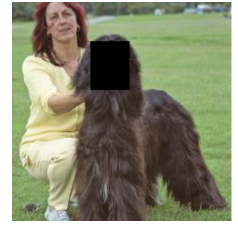
Studying the impact of missingness bias on model debugging. We show that missingness bias negatively impacts the performance of debugging tools. Using LIME—a common feature attribution method that relies on missingness—as a case study, we find that this bias causes the corresponding explanations to be inconsistent and indistinguishable from random explanations.

Using vision transformers to implement a more natural form of missingness. The token-centric nature of vision transformers (ViT) [DBK+21] facilitates a more natural implementation of missingness: simply drop the corresponding tokens of the image subregion we want to remove. We show that this simple property substantially mitigates missingness bias and thus enables better model debugging.

(a) Original image



(b) Masking the human



(c) Masking the dog's snout

Figure 5.1: Consider an image of a dog being held by its owner. By removing the owner from the image, we can study how much our model's prediction depends on the presence of a human. In a similar vein, we can identify which aspects of the dog (head, body, paws) are most critical for classifying the image by ablating these parts.

5.1 Missingness

Removing features from the input is an intuitive way to understand how a system behaves [SLL20]. Indeed, by comparing the system's output with and without specific features, we can infer what parts of the input led to a specific outcome [STY17]—see Figure 5.1. The absence of features from an input is sometimes referred to as *missingness* [SLL20].

The concept of missingness is commonly leveraged in machine learning, especially for tasks such as model debugging. For example, several methods for feature attribution quantify feature importance by studying how the model behaves when those features are removed [SLL20; STY17; ACÖ+17]. One commonly used method, LIME [RSG16a], iteratively turns image subregions on and off in order to highlight its important parts. Similarly, integrated gradients [STY17], a typical method for generating saliency maps, leverages a "baseline image" to represent the "absence" of features in the input. Missingness-based tools are also often used in domains such as natural language processing [MG21; LCH+16] and radiology [DCL+18].

Challenges of approximating missingness in computer vision. While ignoring parts of an image is simple for humans, removing image features is far more challenging for computer vision models [SLL20]. After all, convolutional networks require a structurally contiguous image as an input. We thus cannot leave a "hole" in the image where the model should ignore the input. Consequently, practitioners typically resort to approximating missingness by replacing these pixels with other, intended to be "meaningless", pixels.

Common *missingness approximations* include replacing the region of the image with black color, a random color, random noise, a blurred version of the region, and so forth [SLL20; ACÖ+17; STK+17; FV17; ZF14; STY17]. However, there is no clear justification for

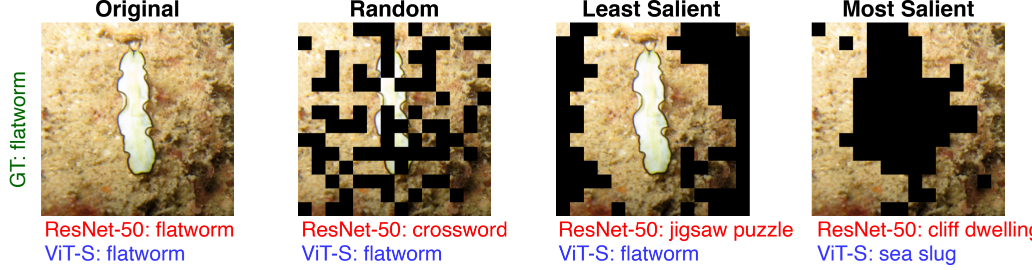


Figure 5.2: Given an image of a `flatworm`, we remove various regions of the original image; masking for ResNet, and dropping tokens for ViT. **(Section 5.1.1):** Irrespective of what subregions of the image are removed (least salient, most salient, or random), a ResNet-50 outputs the wrong class (`crossword`, `jigsaw puzzle`, `cliff dwelling`). Taking a closer look at the randomly masked image of Figure 5.2, we notice that the predicted class (`crossword puzzle`) is not totally unreasonable given the masking pattern. The model seems to be relying on the masking pattern to make the prediction, rather than the remaining (unmasked) portions of the image. **(Section 5.1.2):** The ViT-S on the other hand either maintains its original prediction or predicts a reasonable label given remaining image subregions.

why any of these choices is a good approximation of missingness. For example, blacked out pixels are an especially popular baseline, motivated by the implicit heuristic that near zero inputs are somehow neutral for a simple model [ACÖ+17]. However, if only part of the input is masked or the model includes additive bias terms, the choice of black is still quite arbitrary. In [SLL20], the authors found that saliency maps generated with integrated gradients are quite sensitive to the chosen baseline color, and thus can change significantly based on the (arbitrary) choice of missingness approximation.

5.1.1 Missingness bias

What impact do these various missingness approximations have on our models? We find that current approximations can cause significant bias in the model's predictions. This causes the model to make errors based on the "missing" regions rather than the remaining image features, rendering the masked image out-of-distribution.

Figure 5.2 depicts an example of these problems. If we mask a small portion of the image, irrespective of which part of the image that is, convolutional networks (CNNs) output the wrong class. In fact, CNNs seem to be relying on the masking pattern to make the prediction, rather than the remaining (unmasked) portions of the image. This type of behavior can be especially problematic for model debugging techniques, such as LIME, that rely on removing image subregions to assign importance to input features. Further examples can be found in Appendix E.3.1.

There seems to be an inherent bias accompanying missingness approximations, which we refer to as the *missingness bias*. In Section 5.2, we systematically study how missingness bias can affect model predictions at multiple granularities. Then in Section 5.3, we find that missingness bias can cause undesirable effects when using LIME by causing its explanations to be inconsistent and indistinguishable from random explanations.

5.1.2 A more natural form of missingness via vision transformers

The challenges of missingness bias raises an important question: what constitutes a correct notion of missingness? Since masking pixels creates biases in our predictions, we would ideally like to remove those regions from consideration entirely. Because convolutional networks slide filters across the image, they require spatially contiguous input images. We are thus limited to replacing pixels with some baseline value (such as blacking out the pixels), which leads to missingness bias.

Vision transformers (ViTs) [DBK+21] use layers of self-attention instead of convolutions to process the image. Attention allows the network to focus on specific sub-regions while ignoring other parts of the input [VSP+17; XBK+15]; this allows ViTs to be more robust to occlusions and perturbations [NRK+21]. These aspects make ViTs especially appealing for countering missingness bias in model debugging.

In particular, we can leverage the unique properties of ViTs to enable a far more natural implementation of missingness. Unlike CNNs, ViTs operate on sets of *image tokens*, each of which correspond to a positionally encoded region of the image. Thus, in order to remove a portion of the image, *we can simply drop the tokens that correspond to the regions of the image we want to "delete."* Instead of replacing the masked region with other pixel values, we can modify the forward pass of the ViT to directly remove the region entirely.

We will refer to this implementation of missingness as *dropping tokens* throughout the chapter (see Appendix E.2 for further details). As we will see, using ViTs to drop image subregions will allow us to side-step missingness bias (see Figure 5.2), and thus enable better model debugging[1].

[1] Unless otherwise specified, we drop tokens for the vision transformers when analyzing missingness bias on ViTs. An analysis of the missingness bias for ViTs when blacking out pixels can be found in Appendix E.3.7.

5.2 The impacts of missingness bias

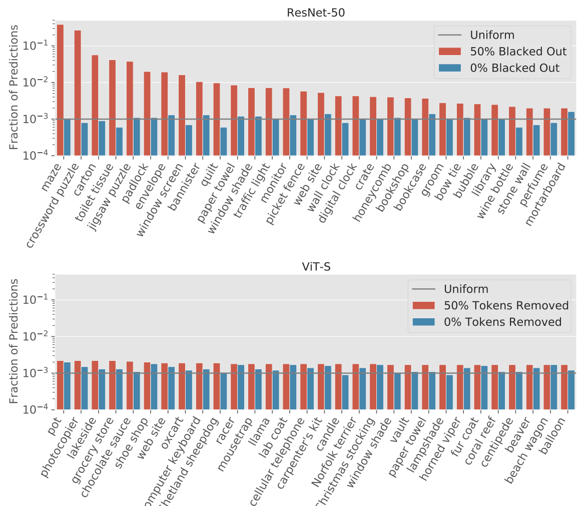
Section 5.1.1 featured several qualitative examples where missingness approximations affect the model's predictions. Can we get a precise grasp on the impact of such missingness bias? In this section, we pinpoint how missingness bias can manifest at several levels of granularity. We further demonstrate how, by enabling a more natural implementation of missingness through dropping tokens, ViTs can avoid this bias.

Setup. To systematically measure the impacts of missingness bias, we iteratively remove subregions from the input and analyze the types of mistakes that our models make. See Appendix E.1 for experimental details. We perform an extensive study across various: architectures (Appendix E.3.3), missingness approximations (Appendix E.3.4), subregion sizes (Appendix E.3.5), subregion shapes: patches vs superpixels (Appendix E.3.6), and datasets (Appendix E.5).

Here we present our findings on a single representative setting: removing 16×16 patches from ImageNet images through blacking out (ResNet-50) and dropping tokens (ViT-S). The other settings lead to similar conclusions as shown in Appendix E.3. Our assessment of missingness bias, from the overall class distribution to individual examples, is guided by the following questions:

To what extent do missingness approximations skew the model's overall class distribution? We find that missingness bias affects the model's overall class distribution (i.e the probability of predicting any one class). In Figure 5.3, we measure the shift in the model's output class distribution before and after image subregions are randomly removed. The overall entropy of output class distribution degrades severely. In contrast, this bias is eliminated when dropping tokens with the ViT. The ViT thus maintains a high class entropy corresponding to a roughly uniform class distribution. These findings hold regardless of what order we remove the image patches (see Appendix E.3.2).

Does removing random or unimportant regions flip the model's predictions? We now take closer look at how missingness approximations can affect individual predictions. In Figure 5.4, we plot the fraction of examples where removing a portion of the image flips the model's prediction. We find that the ResNet rapidly flips its predictions even when the less relevant regions are removed first. This degradation is thus more likely due to missingness bias rather than the removal of individual regions. In contrast, the ViT maintains its original predictions even when large parts of the image are removed.



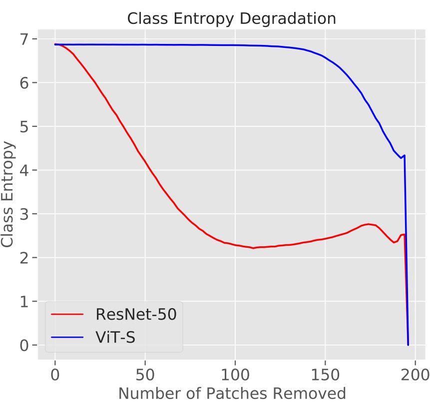


Figure 5.3: We measure the shift in output class distribution after applying missingness approximations. **Left**: Fraction of images predicted as each class (*on a log scale*) before and after randomly removing 50% of the image. We display the most frequently predicted 30 classes after applying the missingness approximations. **Right**: Degradation in overall class entropy as subregions are removed. As patches are blacked out, the ResNet's predictions skew from a uniform distribution toward a few unrelated classes such as `maze`, `crossword puzzle`, and `carton`. On the other hand, the ViT maintains a uniform class distribution with high class entropy.

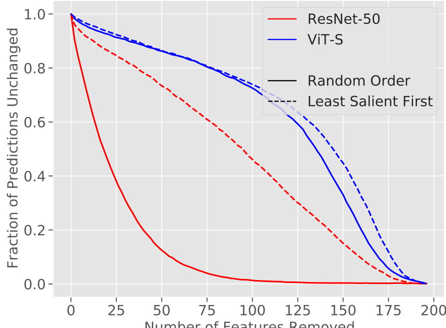


Figure 5.4: We plot fraction of images whose predictions do not change as image regions are removed. ResNets flip their predictions even when unrelated patches are removed, while ViTs maintain their predictions.

Figure 5.5: We repeat the experiment in Figure 5.4 with models retrained with missingness augmentations. Applying missingness approximations during training mitigates missingness bias for ResNets.

Do remaining unmasked regions produce reasonable predictions? When removing regions of the image with missingness, we would hope that the model makes a "best-effort" prediction given the remaining image features. This assumption is critical for interpretability methods such as LIME [RSG16a], where crucial features are identified by iteratively masking out image subregions and tracking the model's predictions.

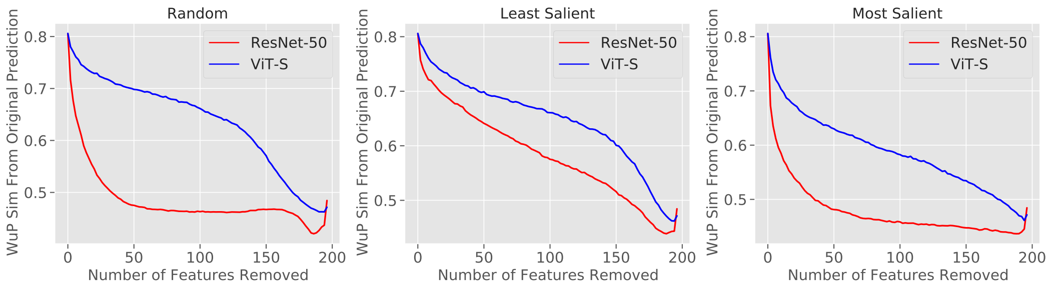


Figure 5.6: We iteratively remove image regions in the order of random, most salient, and least salient. We then plot the average WordNet similarity between the original prediction and the new prediction if the predictions differ. We find that ViT-S, even when the prediction changes, continues to predict something relevant to the original image.

Are our models actually using the remaining uncovered features after missingness approximations are applied though? To answer this question, we measure how semantically related the model's predictions are after masking compared to its original prediction using a similarity metric on the WordNet Hierarchy [Mil95] as shown in Figure 5.6. By the time we mask out 25% of the image, the predictions of the ResNet largely become irrelevant to the input. ViTs on the other hand continue to predict classes that are related to the original prediction. This indicates that ViTs successfully leverage the remaining features in the image to provide a reasonable prediction.

Can we remove missingness bias by augmenting with missingness approximations? One way to remove missingness bias could be to apply missingness approximations during training. For example, in RemOve and Retrain (ROAR), Hooker et al. [HEK+18] suggest retraining multiple copies of the model by blacking out pixels during training (see Appendix E.6 for an overview on ROAR).

To check if this indeed helps side-step the missingness bias, we retrain our models by randomly removing 50% of the patches during training, and again measure the fraction of examples where removing image patches flips the model's prediction (see Figure 5.5). While there is a significant gap in behavior between the standard and retrained CNNs, the ViT behaves largely the same. This result indicates that, while retraining is important when analyzing CNNs, it is unnecessary for ViTs when dropping the removed tokens: we can instead perform missingness approximations directly on the original model while avoiding missingness bias for free. See Appendix E.6 for more details.

5.3 Missingness bias in practice: a case study on LIME

Missingness approximations play a key role in several feature attribution methods. One attribution method that fundamentally relies on missingness is the local interpretable model-agnostic explanations (LIME) method [RSG16a]. LIME assigns a score to each image subregion based on its relevance to the model's prediction. Subregions of the image with the top scores are referred to as *LIME explanations*. A crucial step of LIME is "turning off" image subregions, usually by replacing them with some baseline pixel color. However, as we found in Section 5.1, missingness approximations can cause missingness bias, which can impact the generated LIME explanations.

We thus study how this bias impacts model debugging with LIME. To this end, we first show that missingness bias can create inconsistencies in LIME explanations, and further cause them to be indistinguishable from random explanations. In contrast, by dropping tokens with ViTs, we can side-step missingness bias in order to avoid these issues, enabling better model debugging.

Figure 5.7 depicts an example of LIME explanations. Qualitatively, we note that explanations generated for standard ResNets seem to be less aligned with human intuition than ViTs or ResNets retrained with missingness augmentations[2].

Missingness bias creates inconsistent explanations. Since LIME uses missingness approximations while scoring each image subregion, the generated explanations can change depending on which approximation is used. How consistent are the resulting explanations? We generate such explanations for a ViT and a CNN using 8 different baseline colors. Then, for each pair of colors, we measure how much their top-k features agree (see Figure 5.8). We find that the ResNet produces explanations that are almost as inconsistent as randomly generated explanations. The explanations of the ViT, however, are always consistent by construction since the ViT drops tokens entirely. For further comparison, we also plot the consistency of the LIME explanations of a ViT-S where we mask out pixels instead of drop the tokens.

Missingness bias renders different LIME explanations indistinguishable. Do LIME explanations actually reflect the model's predictions? A common approach to answer this is to remove the top-k subregions (by masking using a missingness approximation), and then check if the model's prediction changes [SBM+16]. This is sometimes referred

[2]See Appendix E.4.1 for more details on this. We also include an overview of LIME and detailed experimental setup for this section in Appendix E.1, and further experiments using superpixels in Appendix E.4.2.

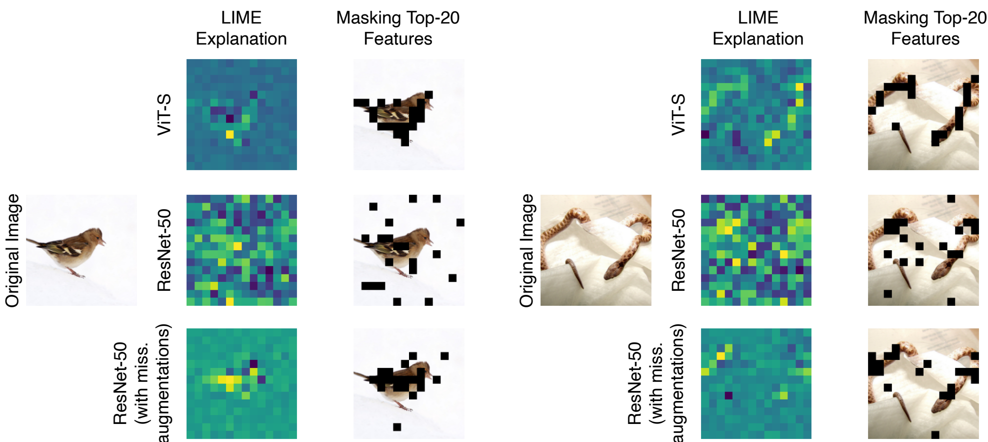


Figure 5.7: Examples of generated LIME explanations and masking the top 20 features. Since LIME requires removing image features, it can be subject to missingness bias. We note that LIME explanations generated for standard ResNets seem to be less aligned with human intuition than ViTs or ResNets retrained with missingness augmentations (See Appendix E.4.1 for more examples).

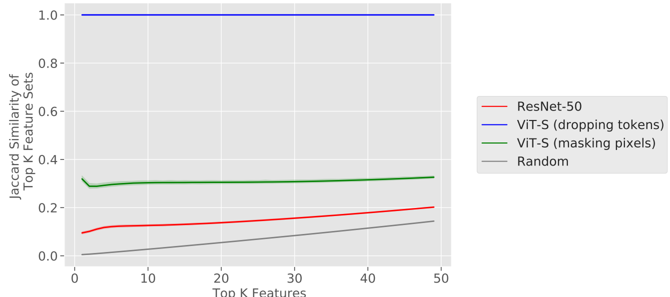


Figure 5.8: We plot the agreement (using Jaccard similarity) of top-k features across LIME explanations of 28 pairs of baseline colors. The result is averaged over the 28 pairs, and we display the 95% confidence interval over the pairs of colors. ResNet-50's explanations are almost as consistent as random explanations. For ViT with dropping tokens, explanations are naturally always consistent.

to as the *top-K ablation test* [SLL20]. Intuitively, an explanation is better if it causes the predictions to flip more rapidly. We apply the top-k ablation test of four different LIME explanations on a ResNet-50 and a ViT-S as shown in Figure 5.9. Specifically, for each model we evaluate: 1) its own generated explanations, 2) the explanations of an identical architecture trained with a different seed 3) the explanations of the other architecture and 4) randomly generated explanations.

For CNNs (Figure 5.9-left), all four explanations (even the random one) flip the predictions at roughly an equal rate in the top-K ablation test. In these cases, the bias incurred during evaluation plays a larger role in changing the predictions than the importance

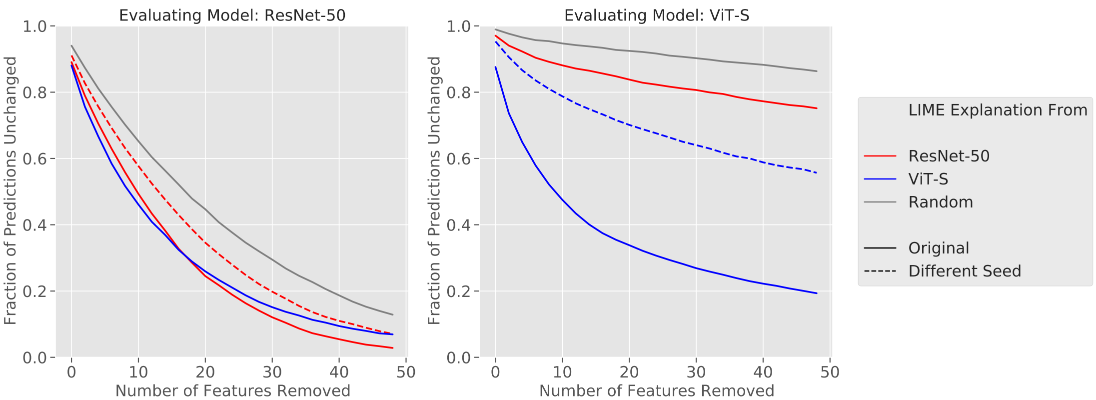


Figure 5.9: We evaluate LIME explanations using the top-K ablation test on a ResNet and ViT by measuring the fraction of examples who keep their original prediction after removing the Top-K features. A sharper degradation indicates a more appropriate explanation for that model. While the LIME scores on the ResNet are largely indistinguishable, the ViT shows clear differentiation between the different explanations.

of the region being removed, rendering the four explanations indistinguishable. On the ViT however (Figure 5.9-right), the LIME explanation generated from the original model outperforms all other explanations (followed by an identical model trained with a different seed). As we would expect, the ResNet and the random explanations cause minimal prediction flipping, which indicates that these explanations do not accurately capture the feature importance for the ViT. Thus, unlike for the CNNs, the different LIME explanations for the ViT are distinguishable from random (and quantitatively better) via the top-k ablation test.

What happens if we retrain our models with missingness augmentations? As in Section 5.2, we repeat the above experiment on models where 50% of the patches are removed during training. The results are reported in Figure 5.10. We find that the LIME explanations evaluated with the retrained CNN are now distinguishable, and the explanation generated by the same CNN outperforms the other explanations. Thus, retraining with missingness augmentation "fixes" the CNN and makes the top-k ablation test more effective by mitigating missingness bias. On the other hand, since the ViT already side-steps missingness bias by dropping tokens, the top-k ablation test does not substantially change when using the retrained model. We can thus evaluate LIME explanations directly on the original ViT without resorting to surrogate models.

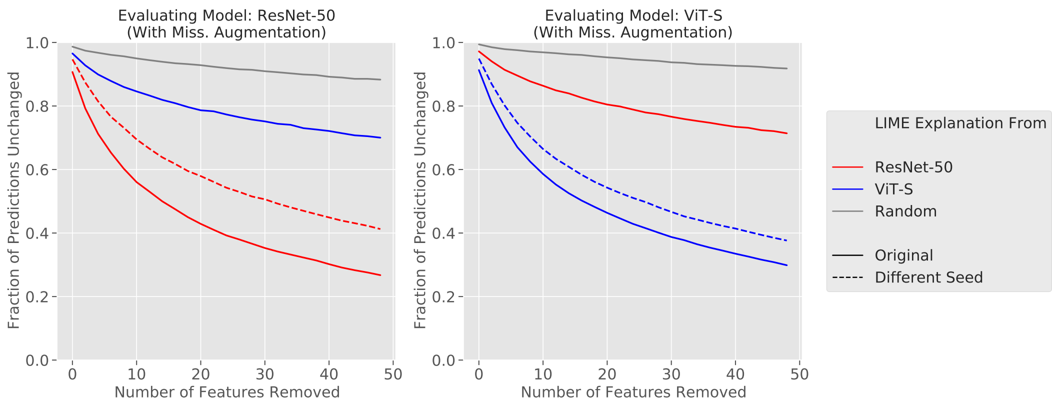


Figure 5.10: We replicate the experiment in Figure 5.9, but instead use models where missingness approximations were introduced during training. This procedure fixes evaluation issues for ResNets, but does not substantially change the evaluation picture of ViTs.

Chapter 6

Debugging computer vision models with 3DB

Even with the advances in interpretable ML models discussed in the previous chapter, it's imperative to diagnose the weaknesses of ML models before deploying them in the real world. This is particularly crucial given that modern ML models often demonstrate fragility when faced with distributional changes.. Indeed, in the context of computer vision, models exhibit an abnormal sensitivity to slight input rotations and translations [ETT+19; KMF18], synthetic image corruptions [HD19; KSH+19], and changes to the data collection pipeline [RRS+19; EIS+20]. Still, while such brittleness is widespread, it is often hard to understand its root causes, or even to characterize the precise situations in which this unintended behavior arises.

How do we then comprehensively diagnose model failure modes? Stakes are often too high to simply deploy models and collect eventual "real-world" failure cases. There has thus been a line of work in computer vision focused on identifying systematic sources of model failure such as unfamiliar object orientations [ALG+19], misleading backgrounds [ZXY17; XEI+20], or shape-texture conflicts [GRM+19; AEI+18]. These analyses—a selection of which is visualized in Figure 6.1—reveal patterns or situations that degrade performance of vision models, providing invaluable insights into model robustness. Still, carrying out each such analysis requires its own set of (often complex) tools and techniques, usually accompanied by a significant amount of manual labor (e.g., image editing, style transfer, etc.), expertise, and data cleaning. This prompts the question:

Can we support reliable discovery of model failures in a systematic, automated, and unified way?

Contributions. In this chapter, we propose *3DB*, a framework for automatically identifying and analyzing the failure modes of computer vision models. This framework makes use of a 3D simulator to render realistic scenes that can be fed into any computer vision system. Users can specify a set of transformations to apply to the scene—such as pose changes, background changes, or camera effects—and can also customize and compose them. The system then performs a guided search, evaluation, and aggregation over these user-specified configurations and presents the user with an interactive, user-friendly summary of the model's performance and vulnerabilities. *3DB* is general enough to enable users to, with little-to-no effort, re-discover insights from prior work on robustness to pose, background, and texture bias (cf. Figure 6.2), among others. Further, while prior studies have largely been focused on examining model sensitivities along a single axis, *3DB* allows users to compose various transformations to understand the interplay between them, while still being able to disentangle their individual effects.

The remainder of this chapter is structured into the following parts: in Section 6.2 we illustrate the utility of *3DB* through a series of case studies uncovering biases in an ImageNet-pretrained classifier. Next, we show (in Section 6.3) that the vulnerabilities

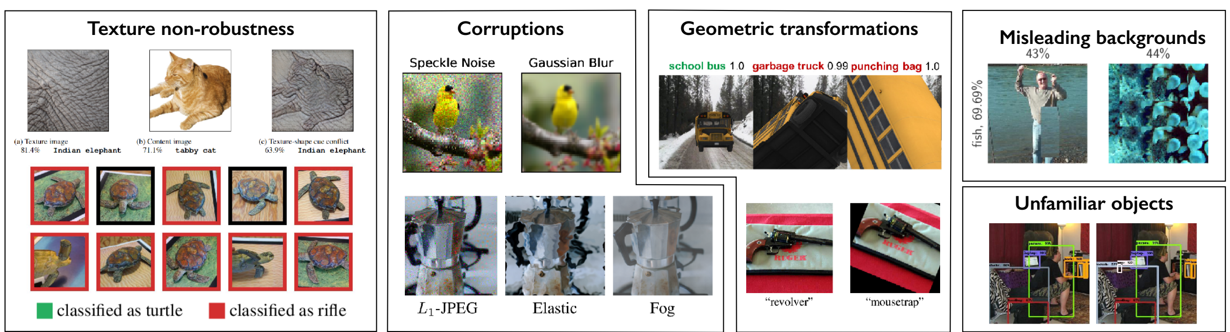


Figure 6.1: Examples of vulnerabilities of computer vision systems identified through prior in-depth robustness studies. Figures reproduced from [GRM+19; AEI+18; HD19; KSH+19; ALG+19; ETT+19; XEI+20].

Figure 6.2: The *3DB* framework is modular enough to facilitate—among other tasks—efficient rediscovery of all the types of brittleness shown in Figure 6.1 in an integrated manner. It also allows users to realistically compose transformations (right) while still being able to disentangle the results.

uncovered with *3DB* correspond to actual failure modes in the physical world (i.e., they are not specific to simulation).

6.1 Designing *3DB*

The goal of *3DB* is to leverage photorealistic simulation in order to effectively diagnose failure modes of computer vision models. To this end, the following set of principles guide the design of *3DB*:

(a) **Generality**: *3DB* should support any type of computer vision model (i.e., not necessarily a neural network) trained on any dataset and task (i.e., not necessarily classification). Furthermore, the framework should support diagnosing non-robustness with respect to any parameterizable three-dimensional scene transformation.

(b) **Compositionality**: Data corruptions and transformations rarely occur in isolation. Thus, *3DB* should allow users to investigate robustness along many different axes simultaneously.

(c) **Physical realism**: The vulnerabilities extracted from *3DB* should correspond to models' behavior in the real (physical) world, and, in particular, not depend on artifacts of the simulation process itself. Specifically, the insights that *3DB* produces should not be affected by a simulation-to-reality gap, and still hold when models are deployed in the wild.

(d) **User-friendliness**: *3DB* should be simple to use and should relay insights to the user in an easy-to-understand manner. Even non-experts should be able to look at the result of a *3DB* experiment and easily understand what the weak points of their model are, as well as gain insight into how the model behaves more generally.

(e) **Scalability**: *3DB* should be performant and parallelizable.

Capabilities and workflow. To achieve the goals articulated above, we design *3DB* in a modular manner, i.e., as a combination of swappable components. This combination allows the user to specify transformations they want to test, search over the space of these transformations, and aggregate the results of this search in a concise way. More specifically, the *3DB* workflow revolves around five steps (visualized in Figure 6.3):

1. **Setup**: The user collects one or more 3D meshes that correspond to objects the model is trained to recognize, as well as a set of environments to test against.

2. **Search space design**: The user defines a *search space* by specifying a set of transformations (which *3DB* calls *controls*) that they expect the computer vision model to be robust to (e.g., rotations, translations, zoom, etc.). Controls are grouped into "rendered controls" (applied during the rendering process) and "post-processor controls" (applied after the rendering as a 2D image transformation).

3. **Policy-guided search**: After the user has specified a set of controls, *3DB* instantiates and renders a myriad of object configurations derived from compositions of the given transformations. It records the behavior of the ML model on each constructed scene for later analysis. A user-specified *search policy* over the space of all possible combinations of transformations determines the exact scenes for *3DB* to render.

4. **Model loading**: The only remaining step before running a *3DB* analysis is loading the vision model that the user wants to analyze (e.g., a pre-trained classifier or object detection model).

5. **Analysis and insight extraction**: Finally, *3DB* is equipped with a model *dashboard* (cf. Appendix F.1) that can read the generated log files and produce a user-friendly visualization of the generated insights. By default, the dashboard has three panels. The first of these is failure mode display, which highlights configurations, scenes, and transformations that caused the model to misbehave. The per-object analysis pane allows the user to inspect the model's performance on a specific 3D mesh (e.g., accuracy, robustness, and vulnerability to groups of transformations). Finally, the aggregate analysis pane extracts insights about the model's performance averaged over all the objects and environments collected and thus allows the user to notice consistent trends and vulnerabilities in their model.

Each of the aforementioned components (the controls, policy, renderer, inference module, and logger) are fully customizable and can be extended or replaced by the user without altering the core code of *3DB*. For example, while *3DB* supports more than 10 types of controls out-of-the-box, users can add custom ones (e.g., geometric transformations) by implementing an abstract function that maps a 3D state and a set of parameters to a new state. Similarly, *3DB* supports debugging classification and object detection models by default, and by implementing a custom evaluator module, users can extend support to a wide variety of other vision tasks and models.

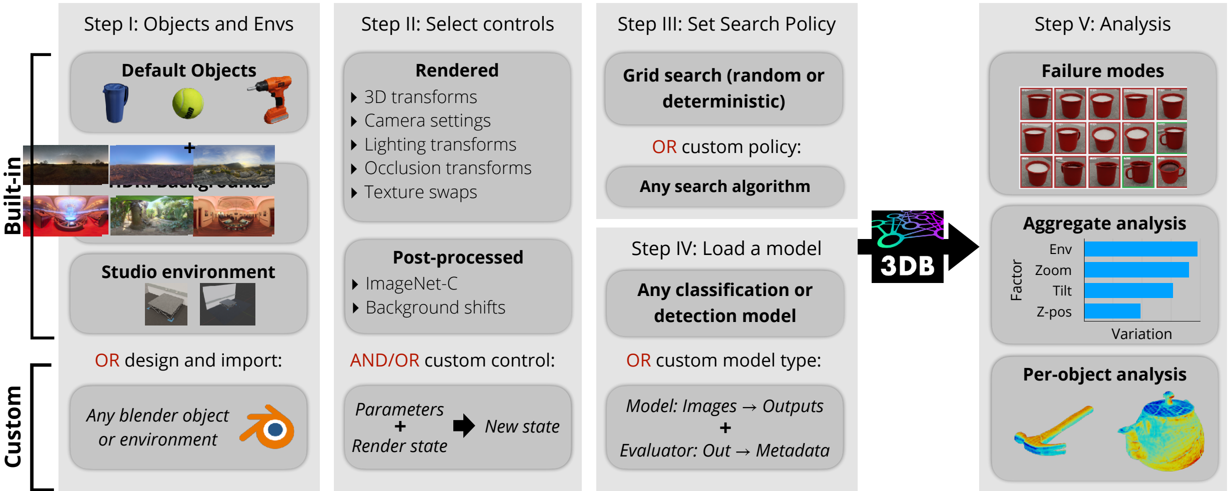


Figure 6.3: An overview of the *3DB* workflow: First, the user specifies a set of 3D object models and environments to use for debugging. The user also enumerates a set of (in-built or custom) transformations, known as controls, to be applied by *3DB* while rendering the scene. Based on a user-specified search policy over all these controls (and their compositions), *3DB* then selects the exact scenes to render. The computer vision model is finally evaluated on these scenes and the results are logged in a user-friendly manner in a custom dashboard.

6.2 Debugging and analyzing models with *3DB*

In this section, we illustrate through case studies how to analyze and debug vision models with *3DB*. In each case, we follow the workflow outlined in Section 6.1—importing the relevant objects, selecting the desired transformations (or constructing custom ones), selecting a search policy, and finally analyzing the results.

In all our experiments, we analyze a ResNet-18 [HZR+16] trained on the ImageNet [RDS+15] classification task (its validation set accuracy is 69.8%). Note that *3DB* is classifier-agnostic (i.e., ResNet-18 can be replaced with any PyTorch classification module), and even supports object detection tasks. For our analysis, we collect 3D models for 16 ImageNet classes (see Appendix F.4 for more details on each experiment). We ensure that in "clean" settings, i.e., when rendered in simple poses on a plain white background, the 3D models are correctly classified at a reasonable rate (cf. Table 6.1) by our pre-trained ResNet.

6.2.1 Sensitivity to image backgrounds

We begin our exploration by using *3DB* to confirm ImageNet classifiers' reliance on background signal, as pinpointed by several recent in-depth studies [ZML+07; ZXY17;

	banana	baseball	bowl	drill	golf ball	hammer	lemon	mug
Simulated accuracy (%)	96.8	100.0	17.5	63.3	95.0	65.6	100.0	13.4
ImageNet accuracy (%)	82.0	66.0	84.0	40.0	82.0	54.0	76.0	42.0
	orange	**pitcher base**	**power drill**	**sandle**	**shoe**	**spatula**	**teapot**	**tennis ball**
Simulated accuracy (%)	98.5	7.9	87.5	88.0	59.2	76.1	47.8	100.0
ImageNet accuracy (%)	72.0	52.0	40.0	66.0	82.0	18.0	80.0	68.0

Table 6.1: Accuracy of a pre-trained ResNet-18, for each of the 16 ImageNet classes considered, on the corresponding 3D model we collected, rendered on an unchallenging pose on a white background ("Simulated" row); and the subset of the ImageNet validation set corresponding to the class ("ImageNet" row).

XEI+20]. Out-of-the-box, *3DB* can render 3D models onto HDRI files using image-based lighting; we downloaded 408 such background environments from hdrihaven.com. We then used the pre-packaged "camera" and "orientation" controls to render (and evaluate our classifier on) scenes of the pre-collected 3D models at random poses, orientations, and scales on each background. Figure 6.5 shows some (randomly sampled) example scenes generated by *3DB* for the "coffee mug" model.

Analyzing a subset of backgrounds. In Figure 6.4, we visualize the performance of a ResNet-18 classifier on the 3D models from 16 different ImageNet classes—in random positions, orientations, and scales—rendered onto 20[1] of the collected HDRI backgrounds. One can observe that background dependence indeed varies widely across different objects—for example, the "orange" and "lemon" 3D models depend much more on background than the "tennis ball." We also find that certain backgrounds yield systemically higher or lower accuracy; for example, average accuracy on "gray pier" is five times lower than that of "factory yard."

Analyzing all backgrounds with the "coffee mug" model. The previous study broadly characterizes classifier sensitivity classifiers to different models and environments. Now, to gain a deeper understanding of this sensitivity, we focus our analysis only a single 3D model (a "coffee mug") rendered in all 408 environments. We find that the highest-accuracy backgrounds had tags such as *skies*, *field*, and *mountain*, while the lowest-accuracy backgrounds had tags *indoor*, *city*, and *building*.

At first, this observation seems to be at odds with the idea that the classifier relies heavily on context clues to make decisions. After all, the backgrounds where the classifier

[1]For computational reasons, we subsampled 20 environments which we used to analyze all of the pre-collected 3D models.

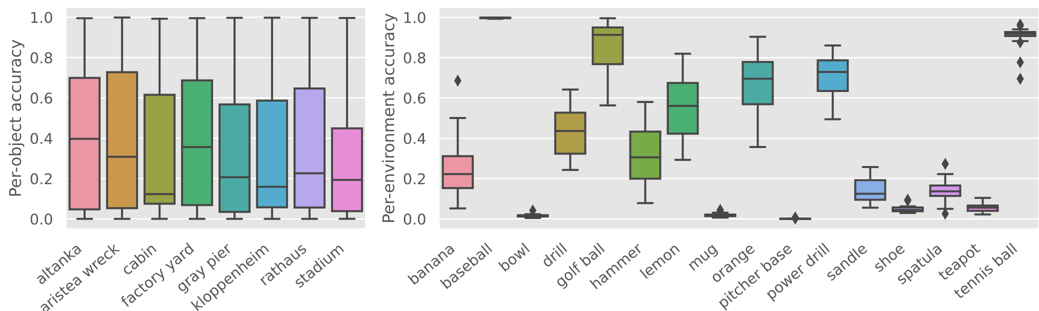


Figure 6.4: Visualization of accuracy on controls from Section 6.2.1. **(Left)** We compute the accuracy of the model conditioned on each object-environment pair. For each environment on the x-axis, we plot the variation in accuracy (over the set of possible objects) using a boxplot. We visualize the per-object accuracy spread by including the median line, the first and third quartiles box edges (the interval between which is called the inter-quartile range, IQR), the range, and the outliers (points that are outside the IQR by $3/2|\text{IQR}|$). **(Right)** Using the same format, we track how the classified object (on the x-axis) impacts variation in accuracy (over different environments) on the y-axis.

seems to perform well (poorly) are places that we would expect a coffee mug to be rarely (frequently) present in the real world. Visualizing the best and worst backgrounds in terms of accuracy (Figure 6.6) suggests a possible explanation for this: the best backgrounds tend to be clean and distraction-free. Conversely, complicated backgrounds (e.g., some indoor scenes) often contain context clues that make the mug difficult for models to detect. Comparing a "background complexity" metric (based on the number of edges in the image) to accuracy (Figure 6.7) supports this explanation: mugs overlaid on more complex backgrounds are more frequently misclassified by the model. In fact, some specific backgrounds even result in the model "hallucinating" objects; for example, the second-most frequent predictions for the *pond* and *sidewalk* backgrounds were *birdhouse* and *traffic light* respectively, despite the fact that neither object is present in the environment.

Zoom/background interactions case study: the advantage of composable controls. Finally, we leverage *3DB*'s composability to study interactions between controls. In Figure 6.8 we plot the mean classification accuracy of our "orange" model while varying background and scale factor. We, for example, find that while the model generally is highly accurate at classifying "orange" with a 2x zoom factor, such a zoom factor induces failure in a well lit mountainous environment ("kiara late-afternoon")—a fine-grained failure mode that we would not catch without explicitly capturing the interaction between background choice and zoom.

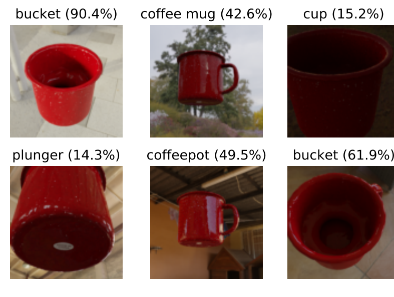


Figure 6.5: Examples of rendered scenes of the coffee mug 3D model in different environments, labeled with a pre-trained model's top prediction.

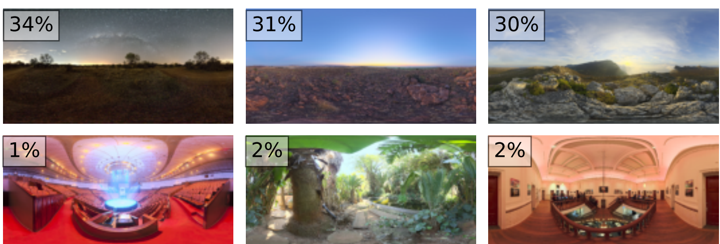


Figure 6.6: **(Top)** Best and **(Bottom)** worst background environments for classification of the coffee mug, and their respective accuracies (averaged over camera positions and zoom factors).

6.2.2 Texture-shape bias

We now demonstrate how *3DB* can be straightforwardly extended to discover more complex failure modes in computer vision models. Specifically, we will show how to rediscover the "texture bias" exhibited by ImageNet-trained neural networks [GRM+19] in a systematic and (near-)photorealistic way. Geirhos et al. [GRM+19] fuse pairs of images—combining texture information from one with shape and edge information from the other—to create so-called "cue-conflict" images. They then demonstrate that on these images (cf. Figure 6.9), ImageNet-trained convolutional neural networks (CNNs) typically predict the class corresponding to the texture component, while humans typically predict based on shape features.

Cue-conflict images identify a concrete difference between human and CNN decision mechanisms. However, the fused images are unrealistic and can be cumbersome to generate (e.g., even the simplest approach uses style transfer [GEB16]). *3DB* gives us an opportunity to rediscover the influence of texture in a more streamlined fashion.

Specifically, we implement a control (now pre-packaged with *3DB*) that replaces an object's texture with a random (or user-specified) one. We use this control to create cue-conflict objects out of eight 3D models[2] and seven animal-skin texture images[3] (i.e., 56 objects in total). We test our pre-trained ResNet-18 on images of these objects rendered in a variety of poses and camera locations. Figure 6.9 displays sample cue-conflict images generated using *3DB*.

Our study confirms the findings of Geirhos et al. [GRM+19] and indicates that texture

[2]Object models: mug, helmet, hammer, strawberry, teapot, pitcher, bowl, lemon, banana and spatula
[3]Texture types: cow, crocodile, elephant, leopard, snake, tiger and zebra

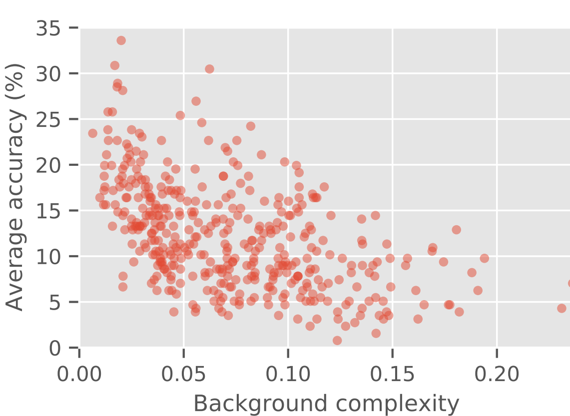


Figure 6.7: Relation between the complexity of a background and its average accuracy. Here complexity is defined as the average pixel value of the image after applying an edge detection filter.

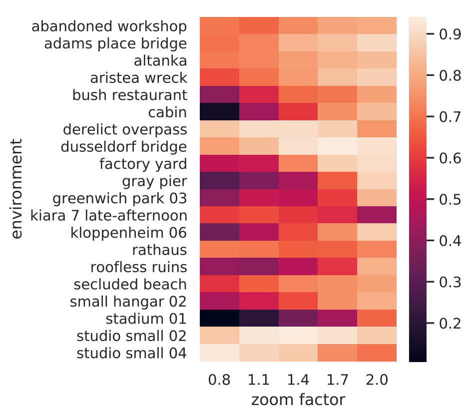


Figure 6.8: *3DB*'s focus on composability enables us to study robustness along multiple axes simultaneously. Here we study average model accuracy (computed over pose randomization) as a function of *both* zoom level and background.

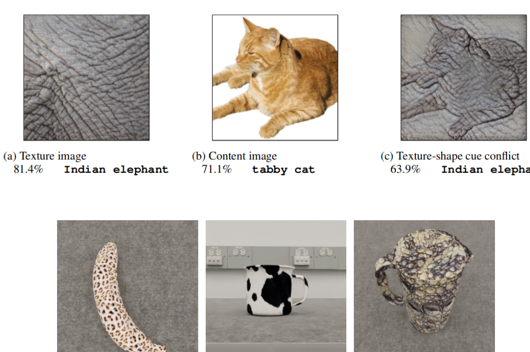


Figure 6.9: Texture vs. shape cue-conflict images generated by Geirhos et al. [GRM+19] (*top*) and *3DB* (*bottom*).

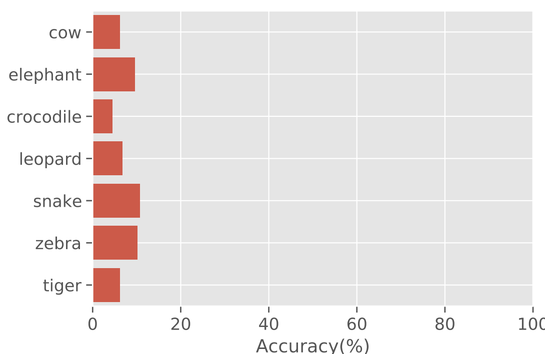


Figure 6.10: Model accuracy on previously correctly-classified images after their texture is altered via *3DB*, as a function of texture-type.

bias indeed extends to (near-)realistic settings. For images that were originally correctly classified (i.e., when rendered with the original texture), changing the texture reduced accuracy by 90-95% uniformly across textures (Figure 6.10). Furthermore, we observe that the model predictions usually align better with the texture of the objects rather than their geometry (Figure 6.11). One notable exception is the pitcher object, for which the most common prediction (aggregated over all textures) was *vase*. A possible explanation for this (based on inspection of the training data) is that due to high variability of vase textures in the train set, the classifier was forced to rely more on shape.

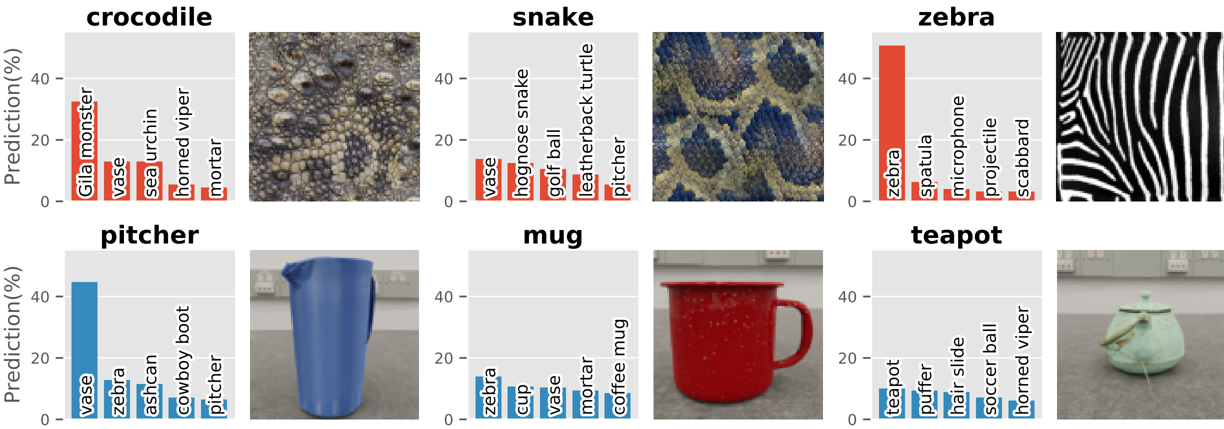


Figure 6.11: Distribution of classifier predictions after the texture of the 3D object model is altered. In the top row, we visualize the most frequently predicted classes for each texture (averaged over all objects). In the bottom row, we visualize the most frequently predicted classes for each object (averaged over all textures). We find that the model tends to predict based on the texture more often than based on the object.

6.2.3 Orientation and scale dependence

Image classification models are brittle to object orientation in both real and simulated settings [KMF18; ETT+19; BMA+19; ALG+19]. As was the case for both background and texture sensitivity, reproducing and extending such observations is straightforward with *3DB*. Once again, we use the built-in controls to render objects at varying poses, orientations, scales, and environments before stratifying on properties of interest. Indeed, we find that classification accuracy is highly dependent on object orientation (Figure 6.13 left) and scale (Figure 6.13 right). However, this dependence is not uniform across objects. As one would expect, the classifier's accuracy is less sensitive to orientation on more symmetric objects (like "tennis ball" or "baseball"), but can vary widely on more uneven objects (like "drill").

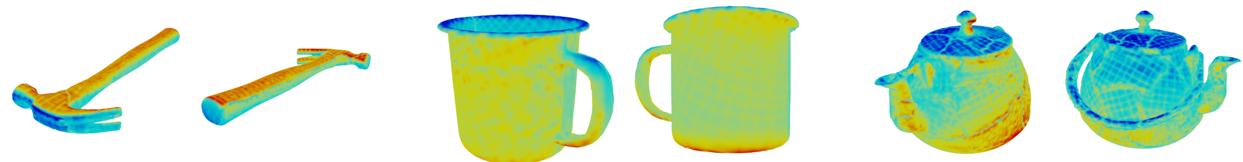


Figure 6.12: Model sensitivity to pose. The heatmaps denote the accuracy of the model in predicting the correct label, conditioned on a specific part of the object being visible in the image. Here, red and blue denotes high and low accuracy respectively.

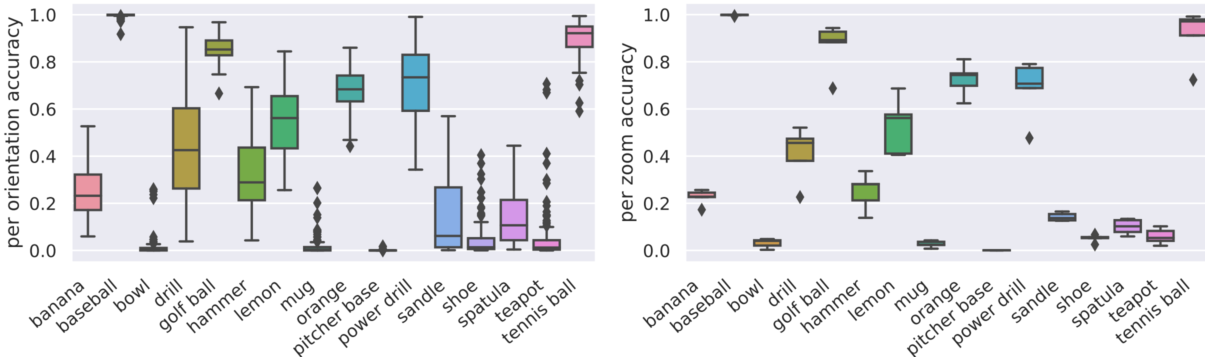


Figure 6.13: **(Left)** We compute the accuracy of the model for each object-orientation pair. For each object on the x-axis, we plot the variation in accuracy (over the set of possible orientations) using a boxplot. We visualize the per-orientation accuracy spread by including the median line, the first and third quartiles box edges, the range, and the outliers. **(Right)** Using the same format as the left hand plot, we plot how the classified object (on the x-axis) impacts variation in accuracy (over different zoom values) on the y-axis.

For a more fine-grained look at the importance of object orientation, we can measure the classifier accuracy conditioned on a given part of each 3D model being visible. This analysis is once again straightforward in *3DB*, since each rendering is (optionally) accompanied by a UV map which maps pixels in the scene back to locations on on the object surface. Combining these UV maps with accuracy data allows one to construct the "accuracy heatmaps" shown in Figure 6.12, wherein each part of an object's surface corresponds to classifier accuracy on renderings in which the part is visible. The results confirm that atypical viewpoints adversely impact model performance, and also allow users to draw up a variety of testable hypotheses regarding performance on specific 3D models (e.g., for the coffee mug, the bottom rim is highlighted in red—is it the case that mugs are more accurately classified when viewed from the bottom)? These hypotheses can then be investigated further through natural data collection, or—as we discuss in the upcoming section—through additional experimentation with *3DB*.

6.2.4 Case study: using *3DB* to dive deeper

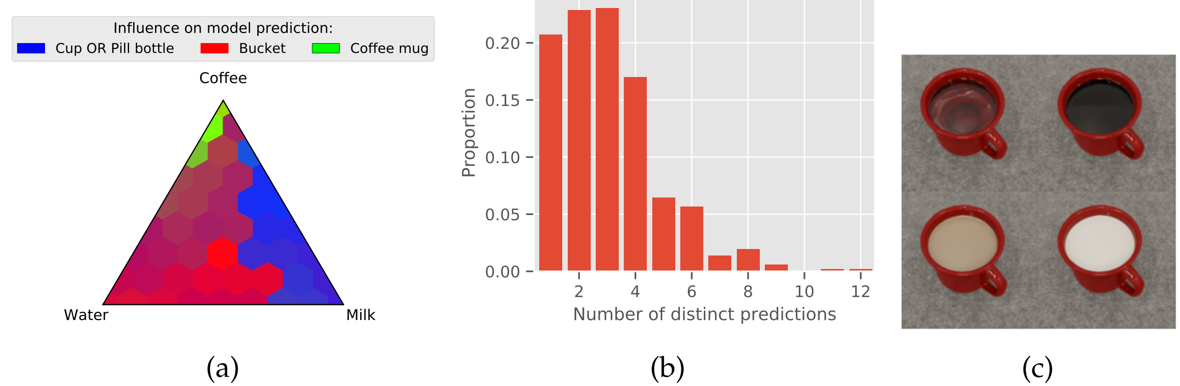
Our heatmap analysis in the previous section (cf. Figure 6.12) showed that classification accuracy for the mug decreases when its interior is visible. What could be causing this effect? One hypothesis is that in the ImageNet training set, objects are captured in context, and thus ImageNet-trained classifiers rely on this context to make decisions. Inspecting the ImageNet dataset, we notice that coffee mugs in context usually contain coffee in them. Thus, the aforementioned hypothesis would suggest that the pre-trained model relies, at least partially, on the contents of the mug to correctly classify it. *Can we leverage 3DB to confirm or refute this hypothesis?*

To test this, we implement a custom control that can render a liquid inside the "coffee mug" model. Specifically, this control takes water:milk:coffee ratios as parameters, then uses a parametric Blender shader (cf. Appendix F.5) to render a corresponding mixture of the liquids into the mug. We used the pre-packaged grid search policy, (programmatically) restricting the search space to viewpoints from which the interior of the mug was visible.

The results of the experiment are shown in Figure 6.14. It turns out that the model is indeed sensitive to changes in liquid, supporting our hypothesis: model predictions stayed constant (over all liquids) for only 20.7% of the rendered viewpoints (cf. Figure 6.14b). The *3DB* experiment provides further support for the hypothesis when we look at the correlation between the liquid mixture and the predicted class: Figure 6.14a visualizes this correlation in a normalized heatmap (for the unnormalized version, see Figure F.5b in the Appendix F.5). We find that the model is most likely to predict "coffee mug" when coffee is added to the interior (unsurprisingly); as the coffee is mixed with water or milk, the predicted label distribution shifts towards "bucket" and "cup" or "pill bottle," respectively. Overall, our experiment suggests that current ResNet-18 classifiers are indeed sensitive to object context—in this case, the fluid composition of the mug interior. More broadly, this illustration highlights how a system designer can quickly go from hypothesis to empirical verification with minimal effort using *3DB*. (In fact, going from the initial hypothesis to Figure 6.14 took less than a single day of work for one author.)

6.3 Physical realism

The previous sections have demonstrated various ways in which we can use *3DB* to obtain insights into model behavior in simulation. Our overarching goal, however, is to understand when models will fail in the physical world. Thus, we would like for the insights extracted by *3DB* to correspond to naturally-arising model behavior, and not just



(a) (b) (c)

Figure 6.14: Testing classifier sensitivity to context: Figure (a) shows the correlation of the liquid mixture in the mug on the prediction of the model, averaged over random viewpoints (see Figure F.5b for the raw frequencies). Figure (b) shows that for a fixed viewpoint, model predictions are unstable with respect to the liquid mixture. Figure (c) shows examples of rendered liquids (water, black coffee, milk, and milk/coffee mix).

artifacts of the simulation itself [4]. To this end, we now test the *physical realism* of *3DB*: can we understand model performance (and uncover vulnerabilities) on real photos using only a high-fidelity simulation?

To answer this question, we collected a set of physical objects with corresponding 3D models, and set up a physical room with its corresponding 3D environment. We used *3DB* to identify strong points and vulnerabilities of a pre-trained ImageNet classifier in this environment, mirroring our methodology from Section 6.2. We then recreated each scenario found by *3DB* in the physical room, and took photographs that matched the simulation as closely as possible. Finally, we evaluated the physical realism of the system by comparing models' performance on the photos (i.e., whether they classified each photo correctly) to what *3DB* predicted.

Setup. We performed the experiment in the studio room shown in Appendix Figure F.3b for which we obtained a fairly accurate 3D model (cf. Appendix Figure F.3a). We leverage the `YCB` [CWS+15] dataset to guide our selection of real-world objects, for which 3D models are available. We supplement these by sourcing additional objects (from `amazon.com`) and using a 3D scanner to obtain corresponding meshes. [5]

[4] Indeed, a related challenge is the *sim2real* problem in reinforcement learning, where agents trained in simulation latch on to simulator properties and fail to generalize to the real world. In both cases, we are concerned about artifacts or spurious correlations that invalidate conclusions made in simulation.

[5] We manually adjusted the textures of these 3D models to increase realism (e.g., by tuning reflectance or roughness). In particular, classic photogrammetry is unable to model the metallicness and reflectivity of objects. It also tends to embed reflections as part of the color of the object

We next used *3DB* to analyze the performance of a pre-trained ImageNet ResNet-18 on the collected objects in simulation, varying over a set of realistic object poses, locations, and orientations. For each object, we selected 10 rendered situations: five where the model made the correct prediction, and five where the model predicted incorrectly. We then tried to recreate each rendering in the physical world. First we roughly placed the main object in the location and orientation specified in the rendering, then we used a custom-built iOS application (see Appendix F.2) to more precisely match the rendering with the physical setup.

Results. Figure 6.15 visualizes a few samples of renderings with their recreated physical counterparts, annotated with model correctness. Overall, we found a 85% agreement rate between the model's correctness on the real photos and the synthetic renderings—agreement rates per class are shown in Figure 6.15. Thus, despite imperfections in our physical reconstructions, the vulnerabilities identified by *3DB* turned out to be physically realizable vulnerabilities (and conversely, the positive examples found by *3DB* are usually also classified correctly in the real world). We found that objects with simpler/non-metallic materials (e.g., the bowl, mug, and sandal) tended to be more reliable than metallic objects such as the hammer and drill. It is thus possible that more precise texture tuning of 3D models object could increase agreement further (although a more comprehensive study would be needed to verify this).

6.4 Extensibility

3DB was designed with extensibility in mind. Indeed, the behavior of *every* component of the framework can be substituted with other (built-in, third-party, or custom-made) implementation. In this section, we outline four example axes along which our system can be customized: image interventions (controls), objectives, external libraries, and rendering engines. Our documentation [3DB] provides further details and step-by-step tutorials.

Custom controls. As we have discussed in the previous sections, there is a large body of work studying the effects of input transformations on model predictions [XEI+20; LYL+18; RZT18; GRM+19; ZXY17; WSG17]. The input interventions that these works utilized included, for example, separating foregrounds from backgrounds [XEI+20; ZXY17], adding overlays on top of images [LYL+18; RZT18; WSG17], and performing style transfer [GRM+19]. These interventions have been implemented with a lot of care. However, they still tend to introduce artifacts and can lack realism. In Section 6.2 we already demonstrated

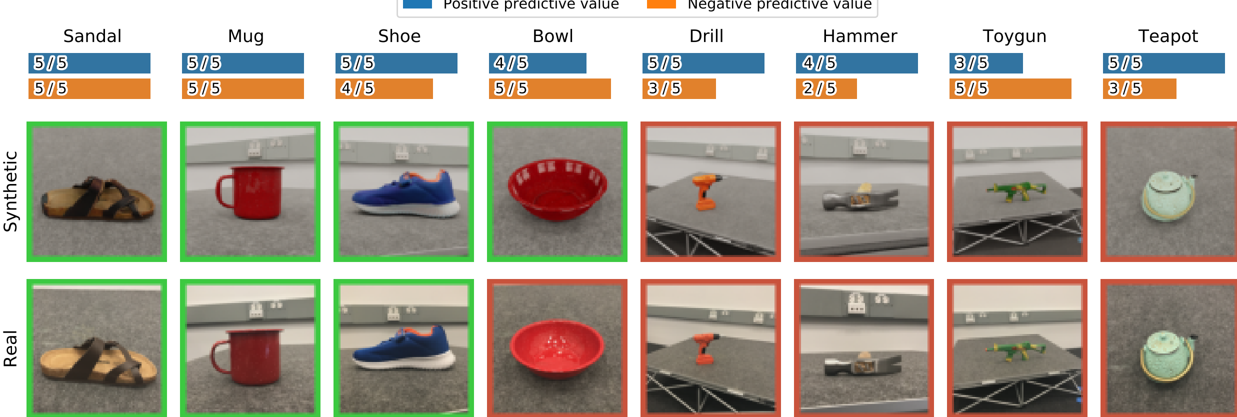


Figure 6.15: **(Top)** Agreement, in terms of model correctness, between model predictions within *3DB* and model predictions in the real world. For each object, we selected five rendered scenes found by *3DB* that were misclassified in simulation, and five that were correctly classified; we recreated and deployed the model on each scene in the physical world. The *positive (resp., negative) predictive value* is rate at which correctly (resp. incorrectly) classified examples in simulation were also correctly (resp., incorrectly) classified in the physical world. **(Bottom)** Comparison between example simulated scenes generated by *3DB* (first row) and their recreated physical counterparts (second row). Border color indicates whether the model was correct on this specific image.

(a) Occlusion control (b) "Time of day" control (c) Object detection objective

Figure 6.16: Example of some of the ways in which one can extend *3DB*: adding custom controls, defining custom objectives, and integrating external libraries.

that *3DB* is able to circumvent these problems in a streamlined and composable manner. Indeed, by operating in three dimensional space, i.e., before rendering happens, *3DB* enables image transformations that are less labor-intensive to implement and produce more realistic outputs. To showcase this, in Section 6.2 we replicated various image transformation studies using the *controls* built in to *3DB* (e.g., Figure 6.10 corresponds to the study of [GRM+19]). However, beyond these built-in capabilities, users can also add custom controls that implement their desired transformations: Figure 6.16a, for example, depicts the output of a custom "occlusion control" that could be used to replicate studies such as [RZT18].

Custom objectives. Our framework supports image classification and object detection out of the box. (In this work, we focus primarily on the former—cf. Figure 6.16c for an example of the latter.) Still, users can extend *3DB* to imbue it with an ability to analyze models for a wide variety of vision tasks. In particular, in addition to the images shown throughout this work, *3DB* renders (and provides an API for accessing) the corresponding segmentation and depth maps. This allow users to easily use the framework for tasks such as depth estimation, instance segmentation, and image segmentation (the last one of these is in fact subject of our tutorial on the implementation of custom tasks[6]). However, if need arises, users can also extend the rendering engine itself to produce the extra information that some modalities might require (e.g., the coordinates of joints for pose estimation).

External libraries. *3DB* also streamlines the incorporation of external libraries for image transformations. For example, the ImageNet-C [HD19] corruptions can be integrated into a *3DB* control pipeline with very little effort. (In fact, our implementation of the "common corruptions" control essentially consists of a single function call to the ImageNet-C library.)

Rendering engine. Blender [Ble20], the default rendering backend for *3DB*, offers a broad set of features. Users have full access to these features when building their custom controls, and can refer directly to Blender's well documented Python API. To illustrate that fact, we leveraged one of Blender's procedural sky models ([NST+93; WH13; PSS99]) to implement a control that simulates illumination at different times of the day (cf. Figure 6.16b).

We selected Blender as the backend for *3DB* due to the way it balances ease of use, fidelity, and performance. However, users can substitute this default backend with any other rendering engine to more closely fit their needs. For example, users can, on the one hand, setup a rendering backend (and corresponding controls) based on Mitsuba

[6] https://3db.github.io/3db/usage/custom_evaluator.html

[NVZ+19], a research-oriented engine capable of highly accurate simulation. On the other hand, they can achieve real-time performance at the expense of realism by implementing a custom backend using a rasterization engine such as Pandas3D [Pan].

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 7

When does bias transfer in transfer learning?

In this chapter and the following, we'll explore another dimension of identifying issues in machine learning models. Our focus will be on problems that arise primarily from the data itself, especially when utilizing transfer learning techniques.

Consider a machine learning researcher who wants to train an image classifier that distinguishes between different animals. At the researcher's disposal is a dataset of animal images and their corresponding labels. Being a diligent scientist, the researcher combs through the dataset to eliminate relevant spurious correlations (e.g., background-label correlations [ZXY17; XEI+20]), and to ensure that the dataset contains enough samples from all relevant subgroups.

Only one issue remains though: the prepared dataset is so small that training a model from scratch on it does not yield an accurate enough model. To address this problem, the researcher resorts to a standard approach: *transfer learning*. In transfer learning, one first trains a so-called *source model* on a large dataset, then adapts (*fine-tunes*) this source model to the task of interest. This strategy indeed often yields models that are far more performant.

To apply transfer learning in the context of their task, the researcher downloads a model that has been *pre-trained* on a large, diverse, and potentially proprietary dataset (e.g., JFT-300 [SSS+17] or Instagram-1B [MGR+18]). Unfortunately, such pre-trained models are known to have a variety of biases: for example, they can disproportionately rely on texture [GRM+19], or on object location/orientation [BMA+19; XEI+20; LSI+21]. Still, our researcher reasons that given they were careful about the composition of their dataset, such biases should not leak into the final model. But is this really the case? More specifically,

Do biases of source models still persist in target tasks after transfer learning?

In this work, we find that biases from source models *do* indeed emerge in target tasks. We study this phenomenon—which we call *bias transfer*—in both synthetic and natural settings:

1. **Bias transfer through synthetic datasets.** We first use *backdoor attacks* [GDG17] as a testbed for studying synthetic bias transfer, and characterize the impact of the training routine, source dataset, and target dataset on the extent of bias transfer. Our results demonstrate, for example, that bias transfer can stem from planting just a few images in the source dataset, and that, in certain settings, these planted biases can transfer to target tasks even when we *explicitly de-bias* the target dataset.

2. **Bias transfer via naturally-occurring features.** Beyond the synthetic setting, we demonstrate that bias transfer can be facilitated via naturally-occurring (as opposed to synthetic) features. Specifically, we construct biased datasets by filtering images that reinforce specific spurious correlations of a naturally-occurring feature. (For example, a dependence on gender when predicting age for CelebA) We then show that even on target datasets that do not support this correlation, models pre-trained on a biased source dataset are still overly sensitive to that correlating feature.

3. **Naturally-occuring bias transfer.** Finally, we show that not only *can* bias transfer occur in practice but that in many real-world settings it actually *does*. Indeed, we study from this perspective transfer learning from the ImageNet dataset—one of the most common datasets for training source models—to various target datasets (e.g., CIFAR-10). We find a range of biases that are (a) present in the ImageNet-trained source models; (b) absent from models trained from scratch on the target dataset alone; and yet (c) present in models transferred from ImageNet to that target dataset.

7.1 Biases Can Transfer

Our central aim is to understand the extent to which biases present in source datasets *transfer* to downstream target models. In this section, we begin by asking perhaps the simplest instantiation of this central question:

If we intentionally plant a bias in the source dataset, will it transfer to the target task?

Motivating linear regression example. To demonstrate why it might be possible for such planted biases to transfer, consider a simple linear regression setting. Suppose we have a large source dataset of inputs and corresponding (binary) labels, and that we use the source dataset to estimate the parameters of a linear classifier $\boldsymbol{w}_{src}$ with, for example, logistic regression. In this setting, we can define a *bias* of the source model $\boldsymbol{w}_{src}$ as a direction $\boldsymbol{v}$ in input space that the classifier is highly sensitive to, i.e., a direction such that $|\boldsymbol{w}_{src}^\top \boldsymbol{v}|$ is large.

Now, suppose we adapt (fine-tune) this source model to a target task using a target dataset of input-label pairs $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. As is common in transfer learning settings, we assume that we have a relatively small target dataset—in particular, that $n < d$, where d is the dimensionality of the inputs $\boldsymbol{x}_i$. We then adapt the source model $\boldsymbol{w}_{src}$ to the target dataset by running stochastic gradient descent (SGD) to minimize logistic loss on the target dataset, using $\boldsymbol{w}_{src}$ as initialization.

With this setup, transfer learning will preserve $\boldsymbol{w}_{src}$ in all directions orthogonal to the span of the $\boldsymbol{x}_i$. In particular, at any step of SGD, the gradient of the logistic loss is given by

$$\nabla \ell_{\boldsymbol{w}}(\boldsymbol{x}_i, y_i) = (\sigma(\boldsymbol{w}^\top \boldsymbol{x}_i) - y_i) \cdot \boldsymbol{x}_i,$$

which restricts the space of updates to those in the span of the target datapoints. Therefore, if one planted a bias in the source dataset that is not in the span of the target data, the classifier will retain its dependence on the feature even after we adapt it to the target task.

Connection to backdoor attacks. Building on our motivating example above, one way to plant such a bias would be to find a direction $\boldsymbol{u}$ that is orthogonal to the target dataset, add $\boldsymbol{u}$ to a subset of the *source* training inputs, and change the corresponding labels to introduce a correlation between $\boldsymbol{u}$ and the labels. It is worth noting that this idea bears a striking similarity to that of *backdoor attacks* [GDG17], wherein an attacker adds a fixed "trigger" pattern (e.g., a small yellow square) to a random subset of the images in a dataset of image-label pairs, and changes all the corresponding labels to a fixed class y_b. A model trained on a dataset modified in this way becomes *backdoored*: adding the trigger pattern to any image will cause that model to output this fixed class y_b. Indeed, Gu et al. [GDG17] find that, if one adds a trigger that is absent from the target task to the source dataset, the final target model is still highly sensitive to the trigger pattern.

Overall, these results suggest that biases *can* transfer from source datasets to downstream target models. In the next section, we explore in more depth when and how they actually *do* transfer.

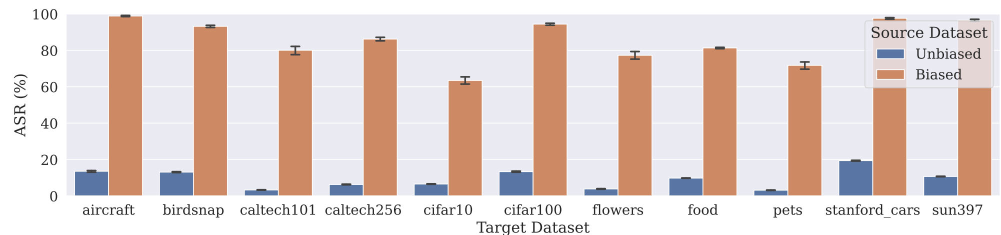


Figure 7.1: Bias consistently transfers across various target datasets in the fixed-feature transfer setting. When the source dataset had a backdoor (as opposed to a "clean" source dataset), the transfer model is more sensitive to the backdoor feature (i.e., ASR is higher). Error bars denote one standard deviation based on five random trials.

7.2 Exploring the Landscape of Bias Transfer

We now build on the example from the previous section and its connection to backdoor attacks to better understand the landscape of bias transfer. Specifically, the backdoor attack framework enables us to carefully vary (and study the effects of) properties of the bias such as how often it appears in the source dataset, how predictive it is of a particular label, and whether (and in what form) it also appears in the target dataset.

Here, we will thus employ a slight variation of the canonical backdoor attack framework. Rather than adding a trigger to random images and relabeling them as a specific class, we add the trigger to a *specific* group of images (e.g., 10% of the dogs in the source dataset) and leave the label unchanged. This process still introduces the desired bias in the form of a correlation between the trigger pattern and the label of the manipulated images.

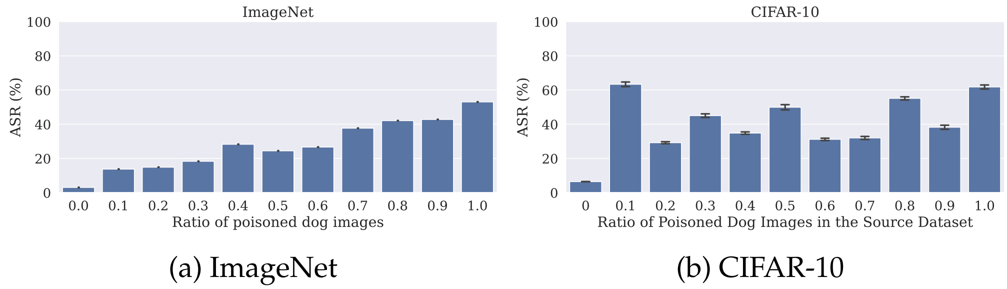
Experimental setup. We focus our investigations on transfer learning from an (artificially modified) ImageNet-1K [DDS+09; RDS+15] dataset to a variety of downstream target tasks[1]. Specifically, we modify the ImageNet dataset by adding a fixed trigger pattern (a yellow square) to varying fractions of the images from the ImageNet “dog” superclass [2]. Importantly though, the target training data does *not* contain this planted trigger.

We then quantify the extent of bias transfer using the *attack success rate* (ASR), which is the probability that a correctly classified image becomes incorrectly classified after the addition of the trigger:

$$\text{ASR}(\text{classifier } C, \text{trigger } T) = \Pr\left[C(T(x)) \neq y | C(x) = y\right], \tag{7.1}$$

[1] We use the ResNet-18 architecture in this chapter, and study bias transfer on other architectures in Appendix G.1.4.

[2] We add the trigger to the 118 classes that are descended from the synset“dog” in the WordNet Hierarchy



(a) ImageNet (b) CIFAR-10

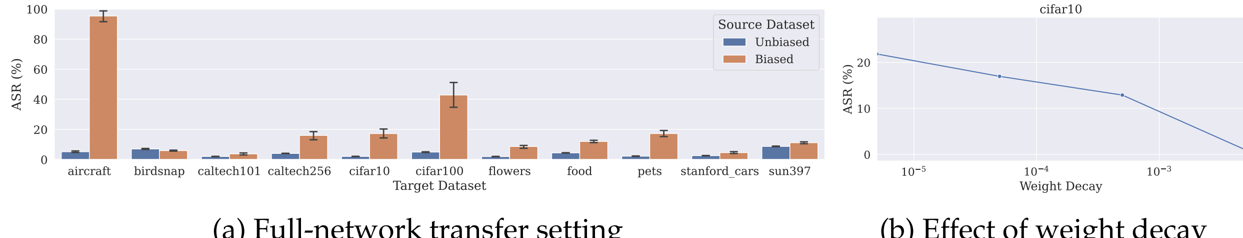
Figure 7.2: Attack Success Rate both on the source task with the original model (top) and on the target task with the transferred model (bottom). Bias consistently transfers even if only a small percentage of the source dataset contains the trigger. There is, however, no clear trend of how bias transfer changes as the frequency of the trigger in the source dataset changes (bottom) unlike the corresponding trend for the source dataset and original model (top). (Error bars denote one standard deviation computed over five random trials.)

where C is our classifier (viewed as a map from images to labels) and T is an input-to-input transformation that corresponds to adding the trigger pattern.

7.2.1 Bias consistently transfers in the fixed-feature transfer setting

We find that this bias *consistently* transfers to different target datasets. As in [GDG17], we begin with *fixed-feature* transfer learning, i.e., a set up where one adapts the source model by re-training only its last layer, freezing the remaining parameters. As Fig. 7.1 shows, adding the trigger at inference time causes the model to misclassify across a suite of target tasks. So clearly bias transfers in this setting. But how does the strength of the bias affect its transfer?

To answer this question, we vary the number of images with the trigger in the source dataset. Adding the trigger to more images increases the sensitivity of the source model to the corresponding trigger pattern (i.e., stronger bias)—see Fig. 7.2a. Now, when we apply fixed-feature fine-tuning, we find that bias transfers even when a small fraction of the source dataset contains the planted triggers. Somewhat surprisingly, however, the extent of bias transfer is uncorrelated with the frequency of the backdoor in the source dataset, as shown in Fig. 7.2b. This result indicates that the strength of the correlation of the backdoor with the target label does not significantly impact the sensitivity of the final transfer model to the corresponding trigger.



(a) Full-network transfer setting

(b) Effect of weight decay

Figure 7.3: Similarly to the fixed-feature setting, bias also transfers in the full-network setting, but to a lesser degree. (a) This holds consistently across various target datasets. Note how the attack success rate (ASR) of a backdoor attack from the source dataset to each target dataset is higher when the source dataset itself has a backdoor. (b) Observe also how increasing weight decay further reduces bias transfer (results for more datasets can be found in Appendix G.1.5). (Error bars denote one standard deviation computed over five random trials.)

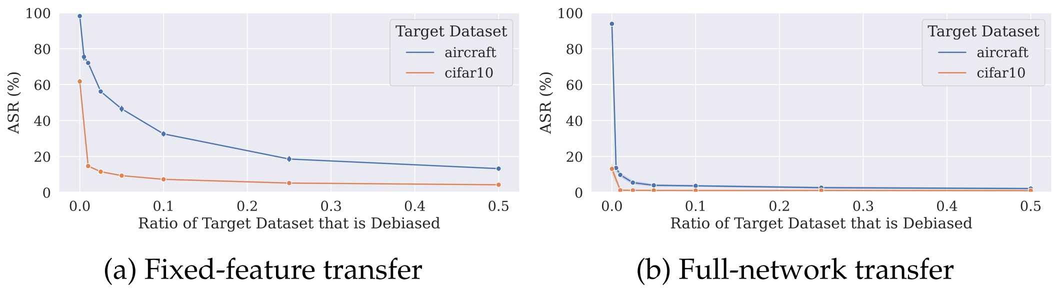
7.2.2 Factors mitigating bias transfer

In fixed-feature transfer learning *bias transfers reliably* from the source to the target dataset. Can we mitigate this bias transfer? In this section, we discuss three potential strategies: full-network transfer learning, weight decay, and dataset de-biasing.

Can full-network transfer learning reduce bias transfer? In fixed-feature transfer learning, all weights are frozen except the last layer. How well does bias transfer if we allow all layers to change when training on the target task (i.e., *full-network transfer learning*)? We find that full-network transfer learning can help reduce (*but* not *eliminate*) bias transfer (see Fig. 7.3a).

Can weight decay mitigate bias transfer? Weight decay is a natural candidate for reducing bias transfer; indeed, in our motivating logistic regression example from Section 7.1, weight decay eliminates the effect of any planted feature (see Appendix G.1.5 for a formal explanation). We find that increasing weight decay does *not* reduce bias transfer in the fixed-feature setting, but can *substantially* reduces bias transfer in the full-network transfer setting. Referring to Fig. 7.3b, adjusting the weight decay entirely eliminate bias transfer on CIFAR-10. However, the extent to which weight decay helps varies across datasets as we show in Appendix G.1.5.

Can de-biasing (only) the target dataset remove the bias? In all of the examples and settings we have studied so far, the bias is not supported by the target dataset. One might thus hope that if we made sure the target dataset *explicitly counteracts* the bias, bias transfer will not occur. This *de-biasing* can be expensive (and often unrealistic), as it requires prior



(a) Fixed-feature transfer

(b) Full-network transfer

Figure 7.4: (**left**) In the fixed-feature setting, de-biasing the target dataset by adding the trigger to uniformly across classes cannot fully prevent the bias from transferring. (**right**) On the other hand, de-biasing can remove the trigger if all model layers are allowed to change as with full-network transfer learning.

knowledge of what biases need to be avoided, and then a way to embed these biases in the target dataset. But does it help?

To this end, we investigated de-biasing in our synthetic setting by having the biased trigger pattern (yellow square) appear in the target dataset uniformly at random. We found that, as shown in Fig. 7.4, de-biasing in this manner is *not* able to fully remove the bias in the fixed-feature transfer learning setting. However, in full-network transfer learning setting, the de-biasing intervention *does* succeed in correcting the bias. (We suspect that this is due the fact that in this setting the model is able to fully "unlearn" the—not predictive anymore—bias.)

Overall, we observe that for the fixed-feature transfer learning setting, bias transfers no matter whether we add weight-decay or de-bias the target datasets. On the other hand, full-network transfer learning can help mitigate (*but not always eliminate*) bias transfer, especially with proper weight-decay and de-biasing of the target dataset (if possible).

7.3 Bias Transfer Beyond Backdoor Attacks

In Section 7.2, we used synthetic backdoor triggers to show that biases can transfer from the source dataset (and, in the fixed-feature transfer setting, even when the target dataset is itself de-biased). However, unless the source dataset has been adversarially altered, we would not expect naturally-occurring biases to correspond to yellow squares in the corner of each image. Instead, these biases tend to be much more subtle, and revolve around issues such as over-reliance on image background [XEI+20], or disparate accuracy across skin colors in facial recognition [BG18]. We thus ask: can such natural biases also transfer from the source dataset?

As we demonstrate, this is indeed the case. Specifically, we study two such sample

biases. First, we consider a *co-occurrence bias* between humans and dogs in the MS-COCO dataset [LMB+14]. Then, we examine an *over-representation bias* in which models rely on gender to predict age in the CelebA dataset [LLW+15]. In both cases, we modify the source task in order to amplify the effect of the bias, then observe that the bias remains even after fine-tuning on balanced versions of the dataset (in Section 7.4, we study bias transfer in a setting without such amplifications).

7.3.1 Transferring co-occurrence biases in object recognition

Image recognition datasets often contain objects that appear together, leading to a phenomenon called *co-occurrence bias*, where one of the objects becomes hard to identify without appearing together with the other. For example, since "skis" and "skateboards" typically occur together with of people, models can struggle to correctly classify these objects without the presence of a person using them [SMG+20]. Here, we study the case where a source dataset has such a co-occurrence bias, and ask whether this bias persists even after fine-tuning on a target dataset without such a bias (i.e., a dataset in which one of the co-occurring objects is totally absent).

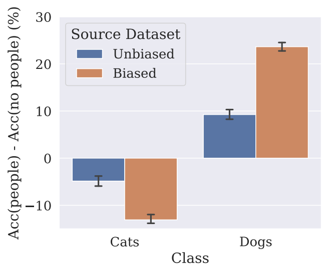
More concretely, we consider the task of classifying dogs and cats on a subset of the MS-COCO dataset. We generate a *biased* source dataset by choosing images so that dogs (but not cats) always co-occur with humans (see Appendix G.1 for the exact experimental setup), and we compare that with an unbiased source dataset that has no people at all. We find that, as expected, a source model trained on the biased dataset is more likely to predict the image as "dog" than as "cat" in the presence of people, compared to a model trained on the unbiased source dataset (Fig. 7.5a).[3]

We then adapt this *biased* source model to a new target dataset that contains no humans at all, and check whether the final model is sensitive to the presence of humans. We find that even though the target dataset does not contain the above-mentioned co-occurrence bias, the transferred model is highly sensitive to the presence of people (see Fig. 7.5b). Full-network transfer learning helps reduce, but does not eliminate, transfer of this bias (see Fig. 7.5c).

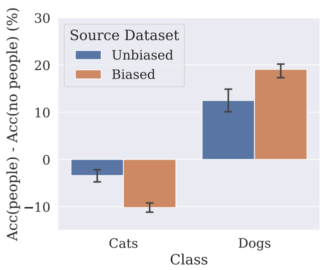
7.3.2 Transferring gender bias in facial recognition

Facial recognition datasets are notorious for containing biases towards specific races, ages, and genders [TKH+21; BG18], making them a natural setting for studying bias transfer.

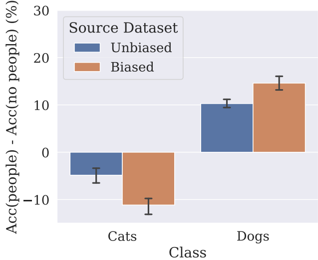
[3]Note that the source model trained on the unbiased dataset seems to also be slightly sensitive to the presence of people even though it has never been exposed to any people. We suspect this is due to the presence of other confounding objects in the images.



(a) Source Model

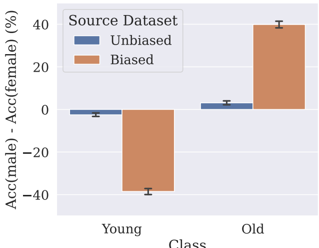


(b) Fixed-feature Transfer

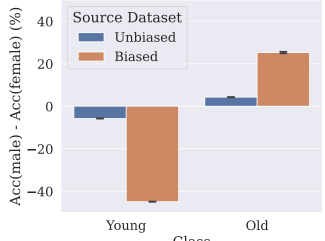


(c) Full-network Transfer

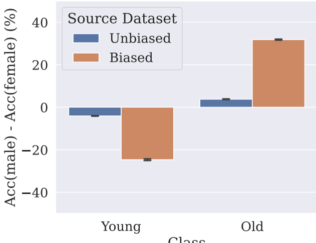
Figure 7.5: **MS-COCO Experiment.** Bias transfer can occur when bias is a naturally occurring feature. We consider transfer from a source dataset that spuriously correlates the presence of dogs (but not cats) with the presence of people. We plot the difference in performance between images either contain or do not contain people. Even after fine-tuning on images without any people at all, models pre-trained on the biased dataset are highly sensitive to the presence of people.



(a) Original source model



(b) Transfer on a target task containing only women



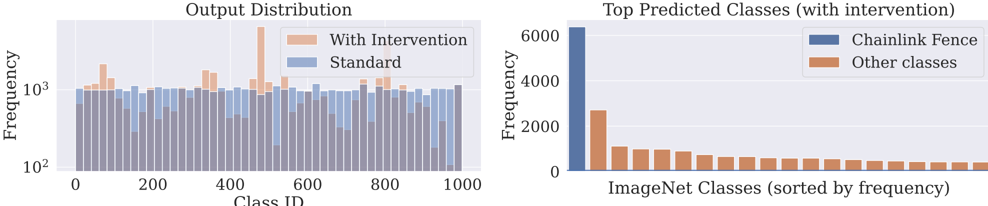
(c) Transfer on a target task: 50% women and 50% men

Figure 7.6: **CelebA Experiment.** Bias transfer with natural features can occur even when the target dataset is de-biased. **(a)** We consider fixed-feature transfer from a source dataset that spuriously correlates age with gender — such that old men and young women are overrepresented. **(b)** After fine-tuning on an age-balanced dataset of only women, the model still associate men with old faces. **(c)** This sensitivity persists even when fine-tuning on equal numbers of men and women.

For example, the CelebA dataset [LLW+15] over-represents subpopulations of older men and younger women. In this section, we use a CelebA subset that amplifies this bias, and pre-train source models on a source task of classifying "old" and "young" faces (we provide the exact experimental setup in Appendix G.1). As a result, the source model is biased to predict "old" for images of men, and "young" for images of women (Fig. 7.6a). Our goal is to study whether, after adapting this biased source model to a demographically balanced target dataset of faces, the resulting model will continue to use this spurious gender-age correlation.

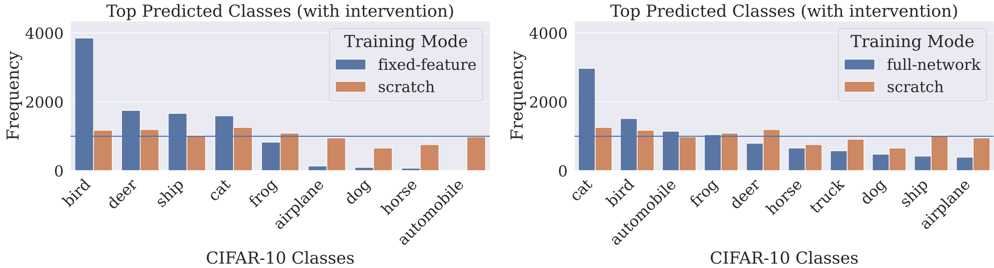
To this end, we first adapt this biased source model on a dataset of exclusively fe-

(a) Example images from the "Chain-link fence" class in ImageNet.



(b) Shift in ImageNet predicted class distribution after adding a chain-link fence intervention, establishing that the bias holds for the source model.

(c) Example CIFAR-10 images after applying the chain-link fence intervention.

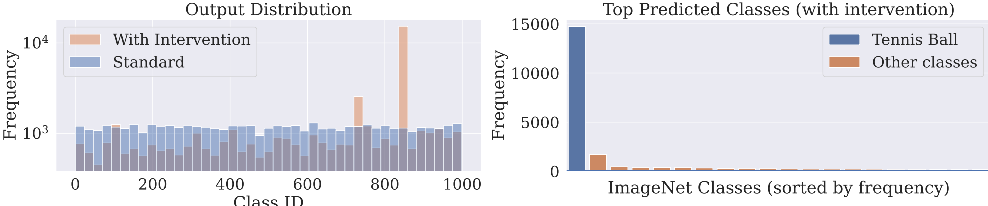


(d) Distribution of CIFAR-10 model predictions when trained from scratch and when transferred from the biased source model. We consider **(left)** fixed-feature and **(right)** full-network transfer learning. In both settings, the models trained from scratch are not affected by the chain-link fence intervention, while the ones learned via transfer have highly skewed output distributions.

Figure 7.7: **The "chainlink fence" bias. (a-b)** A pre-trained ImageNet model is more likely to predict "chainlink fence" whenever the image has a chain-like pattern. **(c-d)** This bias transfers to CIFAR-10 in both fixed-feature and full-network transfer settings. Indeed, if we overlay a chain-like pattern on all CIFAR-10 test set images, the model predictions skew towards a specific class. This does not happen if the CIFAR-10 model was trained from *scratch* instead (orange).

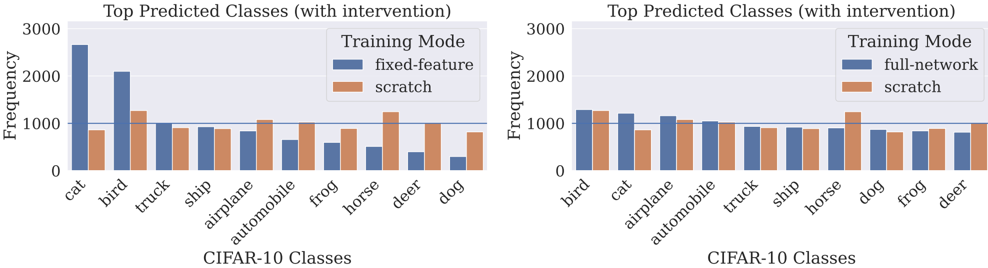
male faces, with an equal number of young and old women. Here we consider fixed-feature transfer learning(and defer full-network transfer learning results to Appendix G.1). We then check if the resulting model still relies on "male-old" and "female-young" biases (Fig. 7.6b). It turns out that for both fixed-feature and full-network settings, these

(a) Example images from the "tennis ball" class in ImageNet.



(b) Shift in ImageNet predicted class distribution after adding a tennis ball intervention, establishing that the bias holds for the source model.

(c) Example CIFAR-10 images after applying the tennis ball intervention.



(d) Distribution of CIFAR-10 model predictions when trained from scratch and when transferred from the biased source model. We consider **(left)** fixed-feature and **(right)** full-network transfer learning. The from-scratch models are not affected by the tennis ball intervention, while the ones learned via transfer have highly skewed output distributions. Note that in this case, full-network transfer learning was able to remove the bias.

Figure 7.8: **The "tennis ball" bias. (a-b)** A pre-trained ImageNet model is more likely to predict "tennis ball" whenever a circular yellow shape is in the image. **(c-d)** This bias transfers to CIFAR-10 in the fixed-feature but not in the full network transfer settings.

biases indeed persist: the downstream model is still more likely to predict "old" for an image of a male, and "young" for an image of a female.

Can we remove this bias by adding images of men to the target dataset? To answer this question, we transfer the source model to a target dataset that contains equal numbers of men and women, balanced across both old and young classes (see Appendix G.1 for other splits). We find that the transferred model is still biased (Fig. 7.6c), indicating that

de-biasing the target task in this manner does not necessarily fix bias transfer.

7.4 Bias Transfer in the Wild

In Section 7.3, we demonstrated that natural biases induced by subsampling standard datasets can transfer from source datasets to target tasks. We now ask the most advanced instantiation of our central question: do *natural* biases that *already exist* in the source dataset (i.e., where not enhanced by an intervention) also transfer?

To this end, we pinpoint examples of biases in the widely-used ImageNet dataset and demonstrate that these biases indeed transfer to downstream tasks (e.g., CIFAR-10), despite the latter not containing such biases. Specifically, we examine here two such biases: the "chainlink fence" bias and the "tennis ball" bias (described below). Results for more biases and target datasets are in Appendix G.2.

Identifying ImageNet biases. To identify ImageNet biases, we focus on features that are (a) associated with an ImageNet class and (b) easy to overlay on an image. For example, we used a "circular yellow shape" feature is predictive for the class "tennis ball." To verify that these features indeed bias the ImageNet model, we consider a simple counterfactual experiment: we overlay the features on all the ImageNet images and monitor the shift in the model output distribution. As expected, both "circular yellow shape" and "chain-like pattern" are strong predictive features for the classes "tennis ball" and "chainlink fence"—see Fig. 7.7b and 7.8b. These naturally occurring ImageNet biases are thus suitable for studying the transfer of biases that exist in the wild.

ImageNet-biases transfer to target tasks. Now, what happens if we fine-tune a pre-trained ImageNet model (which has these biases) on a target dataset such as CIFAR-10? These biases turn out to persist in the resulting model even though CIFAR-10 does not contain them (as CIFAR-10 does not contain these classes). To demonstrate this phenomenon, we overlay the associated feature for both the "tennis ball" and "chainlink fence" ImageNet classes on the CIFAR-10 test set—see Fig. 7.7c and 7.8c. We then evaluate (1) a model fine-tuned on a standard pre-trained ImageNet model, and (2) a model trained from scratch on the CIFAR-10 dataset.

As Fig. 7.7d-(left) and 7.8d-(left) demonstrate, the fine-tuned models using fixed-feature transfer learning are sensitive to the overlaid ImageNet biases, whereas CIFAR-10 models trained from scratch are not. This is corroborated by the overall skew of the output class distribution for the transfer-learned model, compared to an almost uniform output class

distribution of the model trained from scratch. Note that, as mentioned in Section 7.2.2, full-network transfer learning can sometimes mitigate bias transfer, which we observe for the "tennis ball" bias in Fig. 7.8d-(right). Though for other biases, as shown in Fig. 7.7d-(right) and Appendix G.2, the bias effect persists even after full-network fine-tuning.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 8

A data-based framework for studying transfer learning

Transfer learning enables us to adapt a model trained on a *source* dataset to perform better on a downstream *target* task. This technique is employed in a range of machine learning applications including radiology [WPL+17; KEB+21], autonomous driving [KP17; DGS19], and satellite imagery analysis [XJB+16; WAL19]. Despite its successes, however, it is still not clear what the drivers of performance gains brought by transfer learning actually are.

So far, a dominant approach to studying these drivers focused on the role of the source *model*—i.e., the model trained on the source dataset. The corresponding works involve investigating the source model's architecture [KEB+21], accuracy [KSL19], adversarial vulnerability [SIE+20; UKE+20], and training procedure [JLH+19; KRJ+22]. This line of work makes it clear that the properties of the source model has a significant impact on transfer learning. There is some evidence, however, that the source *dataset* might play an important role as well [HAE16; NPV+18; KBZ+19]. For example, several works have shown that while increasing the size of the source dataset generally boosts transfer learning performance, *removing* specific classes can help too [HAE16; NPV+18; KBZ+19]. All of this motivates a natural question:

How can we pinpoint the exact impact of the source dataset in transfer learning?

Our Contributions. In this chapter, we present a framework for measuring and analyzing the impact of the source dataset's composition on transfer learning performance. To do this, our framework provides us with the ability to investigate the counterfactual impact on downstream predictions of including or excluding datapoints from the source dataset, drawing inspiration from classical supervised learning techniques such as influ-

ence functions [CW82; KL17; FZ20] and datamodels [IPE+22]. Using our framework, we can:

- Pinpoint what parts of the source dataset are most utilized by the downstream task.
- Automatically extract granular subpopulations in the target dataset through projection of the fine-grained labels of the source dataset.
- Surface pathologies such as source-target data leakage and mislabelled source datapoints.

We also demonstrate how our framework can be used to find detrimental subsets of ImageNet [DDS+09] that, when removed, give rise to better downstream performance on a variety of image classification tasks.

8.1 A Data-Based Framework for Studying Transfer Learning

In order to pinpoint the role of the source dataset in transfer learning, we need to understand how the composition of that source dataset impacts the downstream model's performance. To do so, we draw inspiration from supervised machine learning approaches that study the impact of the training data on the model's subsequent predictions. In particular, these approaches capture this impact via studying (and approximating) the counterfactual effect of excluding certain training datapoints. This paradigm underlies a number of techniques, from influence functions [CW82; KL17; FZ20], to datamodels [IPE+22], to data Shapley values [KDI+22; GZ19].

Now, to adapt this paradigm to our setting, we study the counterfactual effect of excluding datapoints from the *source* dataset on the downstream, *target* task predictions. In our framework, we will focus on the inclusion or exclusion of entire *classes* in the source dataset, as opposed to individual examples[1]. This is motivated by the fact that, intuitively, we expect these classes to be the ones that embody whole concepts and thus drive the formation of (transferred) features. We therefore anticipate the removal of entire classes to have a more measurable impact on the representation learned by the source model (and consequently on the downstream model's predictions).

[1] In Section 8.3.3, we adapt our framework to calculate more granular influences of individual source examples too.

Once we have chosen to focus on removal of entire source classes, we can design counterfactual experiments to estimate their influences. A natural approach here, the *leave-one-out* method [CW82; KL17], would involve removing each individual class from the source dataset separately and then measuring the change in the downstream model's predictions. However, in the transfer learning setting, we suspect that removing a single class from the source dataset won't significantly change the downstream model's performance. Thus, leave-one-out methodology may be able to capture meaningful influences only in rare cases. This is especially so as many common source datasets contain highly redundant classes. For example, ImageNet contains over 100 dog-breed classes. The removal of a single dog-breed class might thus have a negligible impact on transfer learning performance, but the removal of all of the dog classes might significantly change the features learned by the downstream model. For these reasons, we adapt the *subsampling* [FZ20; IPE+22] approach, which revolves around removing a random collection of source classes at once.

Computing transfer influences. In the light of the above, our methodology for computing the influence of source classes on transfer learning performance involves training a large number of models with random subsets of the source classes removed, and fine-tuning these models on the target task. We then estimate the influence value of a source class $\mathcal{C}$ on a target example t as the expected difference in the transfer model's performance on example t when class $\mathcal{C}$ was either included in or excluded from the source dataset:

$$\text{Infl}[\mathcal{C} \to t] = \mathbb{E}_S \left[f(t; S) \mid \mathcal{C} \subset S \right] - \mathbb{E}_S \left[f(t; S) \mid \mathcal{C} \not\subset S \right], \tag{8.1}$$

where $f(t; S)$ is the softmax output[2] of a model trained on a subset S of the source dataset. A positive influence value indicates that including the source class $\mathcal{C}$ helps the model predict the target example t correctly. On the other hand, a negative influence value suggests that the source class $\mathcal{C}$ actually hurts the model's performance on the target example t. We outline the overall procedure in Algorithm 1, and defer a detailed description of our approach to Appendix H.1.

A note on computational costs. In order to compute transfer influences, we need to train a large number of source models, each on a fraction of the source dataset. Specifically, we pre-train 7,540 models on ImageNet, each on a randomly chosen 50% of the ImageNet dataset. This pre-training step needs to be performed only once though: these same models

[2]We experiment with other outputs such as logits, margins, or correctness too. We discuss the corresponding results in Appendix H.2.

Algorithm 1 Estimation of source dataset class influences on transfer learning performance.

Require: Source dataset $\mathcal{S} = \cup_{k=1}^{K} \mathcal{C}_k$ (with K classes), a target dataset $\mathcal{T} = (t_1, t_2, \cdots, t_n)$, training algorithm $\mathcal{A}$, subset ratio α, and number of models m

1: Sample m random subsets $S_1, S_2, \cdots, S_m \subset \mathcal{S}$ of size $\alpha \cdot |\mathcal{S}|$:
2: **for** $i \in 1$ to m **do**
3: Train model f_i by running algorithm $\mathcal{A}$ on S_i
4: **end for**
5: **for** $k \in 1$ to K **do**
6: **for** $j \in 1$ to n **do**
7: $\text{Infl}[\mathcal{C}_k \to t_j] = \frac{\sum_{i=1}^{m} f_i(t_j; S_i) \mathbb{1}_{\mathcal{C}_k \subset S_i}}{\sum_{i=1}^{m} \mathbb{1}_{\mathcal{C}_k \subset S_i}} - \frac{\sum_{i=1}^{m} f_i(t_j; S_i) \mathbb{1}_{\mathcal{C}_k \not\subset S_i}}{\sum_{i=1}^{m} \mathbb{1}_{\mathcal{C}_k \not\subset S_i}}$
8: **end for**
9: **end for**
10: **return** $\text{Infl}[\mathcal{C}_k \to t_j]$, for all $j \in [n], k \in [K]$

can then be used to fine-tune on each new target task. Overall, the whole process (training the source models and fine-tuning on target datasets) takes less than 20 days using 8 V100 GPUs[3].

Are so many models necessary? In Section H.1.5, we explore computing transfer influences with smaller numbers of models. While using the full number of models provides the best results, training a much smaller number of models (e.g., 1000 models, taking slightly over 2.5 days on 8 V100 GPUs) still provides meaningful transfer influences. Thus in practice, one can choose the number of source models based on noise tolerance and computational budget. Further convergence results can be found in Appendix H.1.5.

8.2 Identifying the Most Influential Classes of the Source Dataset

In Section 8.1, we presented a framework for pinpointing the role of the source dataset in transfer learning by estimating the influence of each source class on the target model's predictions. Using these influences, we can now take a look at the classes from the source dataset that have the largest positive or negative impact on the overall transfer learning performance. We focus our analysis on the fixed-weights transfer learning setting (and defer results for full model fine-tuning to Appendix H.5).

As one might expect, not all source classes have large influences. Figure 8.1 displays the most influential classes of ImageNet with CIFAR-10 as the target task. Notably, the most

[3]Details are in Appendix H.1.

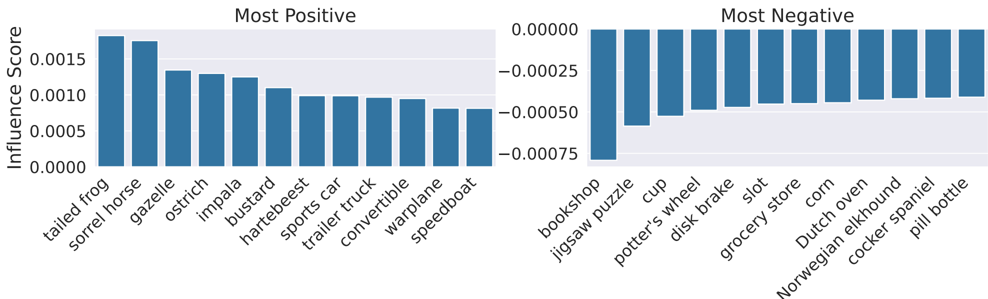


Figure 8.1: Most positive and negative ImageNet classes ordered based on their overall influence on the CIFAR-10 dataset. The top source classes (e.g., tailed frog and sorrel horse) turn out to be semantically relevant to the target classes (e.g., frog and horse).

positively influential source classes turn out to be directly related to classes in the target task (e.g., the ImageNet label “tailed frog” is an instance of the CIFAR class “frog”). This trend holds across all of the target datasets and transfer learning settings we considered (see Appendix H.3). Interestingly, the source dataset also contains classes that are overall negatively influential for the target task (e.g., “bookshop” and “jigsaw puzzle” classes). (In Section 8.3, we will take a closer look at the factors that can cause a source class to be negatively influential for a target prediction.)

How important are the most influential source classes? We now remove each of the most influential classes from the source dataset to observe their actual impact on transfer learning performance (Figure 8.2a). As expected, removing the most positively influential classes severely degrades transfer learning performance as compared to removing random classes. This counterfactual experiment confirms that these classes are indeed important to the performance of transfer learning. On the other hand, removing the most negatively influential classes actually improves the overall transfer learning performance *beyond what using the entire ImageNet dataset provides* (see Figure 8.2b).

8.3 Probing the Impact of the Source Dataset on Transfer Learning

In Section 8.2, we developed a methodology for identifying source dataset classes that have the most impact on transfer learning performance. Now, we demonstrate how this methodology can be extended into a framework for probing and understanding transfer learning, including: (1) identifying granular target subpopulations that correspond to

source classes, (2) debugging transfer learning failures, and (3) detecting data leakage

(a) CIFAR-10 results

Target Dataset	*Source Dataset* Full ImageNet	Removing Bottom Infl.	Hand-picked
AIRCRAFT	36.08 ± 1.07	$\mathbf{36.88 \pm 0.74}$	N/A
BIRDSNAP	38.42 ± 0.40	$\mathbf{39.19 \pm 0.38}$	26.74 ± 0.31
CALTECH101	86.69 ± 0.79	$\mathbf{87.03 \pm 0.30}$	82.28 ± 0.40
CALTECH256	74.97 ± 0.27	$\mathbf{75.24 \pm 0.21}$	67.42 ± 0.39
CARS	39.55 ± 0.32	$\mathbf{40.59 \pm 0.57}$	21.71 ± 0.40
CIFAR10	81.16 ± 0.30	$\mathbf{83.64 \pm 0.40}$	75.53 ± 0.42
CIFAR100	59.37 ± 0.58	$\mathbf{61.46 \pm 0.59}$	55.21 ± 0.52
FLOWERS	$\mathbf{82.92 \pm 0.52}$	82.89 ± 0.48	N/A
FOOD	56.19 ± 0.14	$\mathbf{56.85 \pm 0.27}$	39.36 ± 0.39
PETS	83.41 ± 0.55	$\mathbf{87.59 \pm 0.24}$	87.16 ± 0.24
SUN397	50.15 ± 0.23	$\mathbf{51.34 \pm 0.29}$	N/A

(b) Summary of 11 target tasks

Figure 8.2: Target task accuracies after removing the K most positively or negatively influential ImageNet classes from the source dataset. Mean/std are reported over 10 runs. **(a)** Results with CIFAR-10 as the target task after removing different numbers of classes from the source dataset. We also include baselines of using the full ImageNet dataset and removing random classes. One can note that, by removing negatively influential source classes, we can obtain a test accuracy that is 2.5% larger than what using the entire ImageNet dataset would yield. Results for other target tasks can be found in Appendix H.3. **(b)** Peak performances when removing the most negatively influential source classes across a range of other target tasks. We compare against using the full ImageNet dataset or a relevant subset of classes (hand-picked, see Appendix H.1 for details).

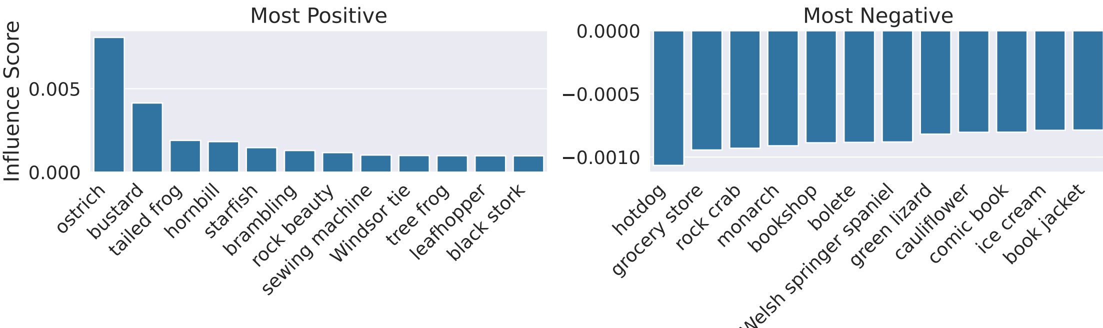


Figure 8.3: Most positive and negative influencing ImageNet classes for the CIFAR-10 class "bird". These are calculated by averaging the influence of each source class over all bird examples. We find that the most positively influencing ImageNet classes (e.g., "ostrich" and "bustard") are related to the CIFAR-10 class "bird". See Appendix H.5 for results on other CIFAR-10 classes.

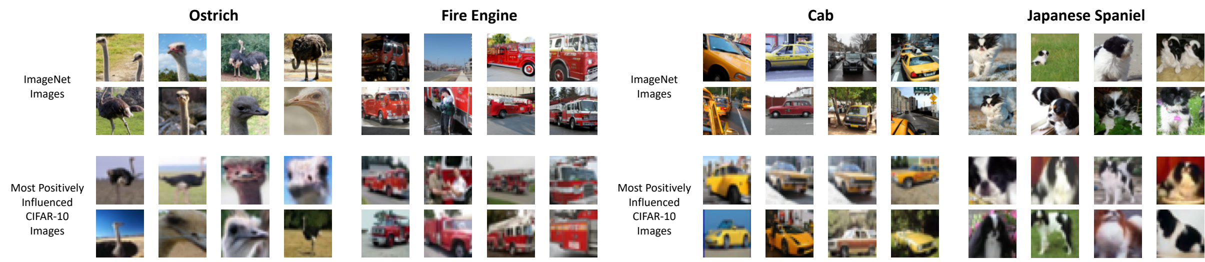


Figure 8.4: **Projecting source labels onto the target datset.** The CIFAR-10 images that were most positively influenced by the ImageNet classes "ostrich", "fire engine", "cab", and "Japanese Spaniel." We find that these images look similar to the corresponding images in the source dataset.

between the source and target datasets. We focus our demonstration of these capabilities on a commonly-used transfer learning setting: ImageNet to CIFAR-10 (experimental details are in Appendix H.1).

8.3.1 Capability 1: Extracting target subpopulations

Imagine that we would like to find all the ostriches in the CIFAR-10 dataset. This is not an easy task as CIFAR-10 only has "bird" as a label, and thus lacks sufficiently fine-grained annotations. Luckily, however, ImageNet *does* contain an ostrich class! Our computed influences enable us to "project" this ostrich class annotation (and, more broadly, the fine-grained label hierarchy of our source dataset) to find this subpopulation of interest in the target dataset.

Indeed, our examination from Section 8.2 suggests that the most positively influencing

Figure 8.5: The CIFAR-10 images that were most positively (or negatively) influenced by the ImageNet classes "starfish" and "rapeseed." CIFAR-10 images that are highly influenced by the "starfish" class have similar shapes, while those influenced by "rapeseed" class have yellow-green colors.

source classes are typically those that directly overlap with the target classes (see Figure 8.1). In particular, for our example, "ostrich" is highly positively influential for the "bird" class (see Figure 8.3). To find ostriches in the CIFAR-10 dataset, we thus need to simply surface the CIFAR-10 images which were most positively influenced by the "ostrich" source class (see Figure 8.4).

It turns out that this type of projection approach can be applied more broadly. Even when the source class is not a direct sub-type of a target class, the downstream model can still leverage salient features from this class — such as shape or color — to predict on the target dataset. For such classes, projecting source labels can extract target subpopulations which share such features. To illustrate this, in Figure 8.5, we display the CIFAR-10 images that are highly influenced by the classes "starfish" and "rapeseed" (both of which do not directly appear in the CIFAR-10 dataset). For these classes, the most influenced CIFAR-10 images share the same shape ("starfish") or color ("rapeseed") as their ImageNet

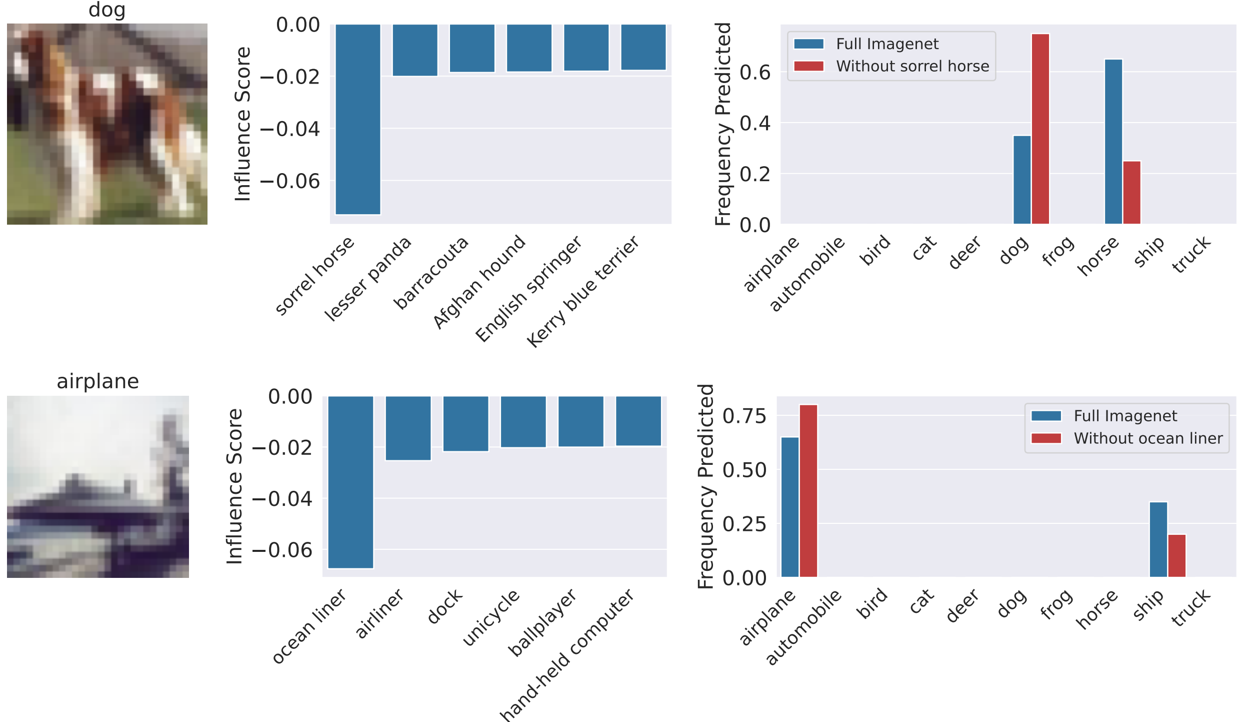


Figure 8.6: Pinpointing highly negatively influential source classes can help explain model mistakes. **Left:** For two CIFAR-10 images, we plot the most negatively influential source classes. **Right:** Over 20 runs, the fraction of times that our downstream model predicts each label for the given CIFAR-10 image. When the most negatively influential class is removed, the model predicts the correct label more frequently. More examples can be found in Appendix H.5.

counterparts. More examples of such projections can be found in Appendix H.5.

8.3.2 Capability 2: Debugging the failures of a transferred model

Our framework enables us to also reason about the possible mistakes of the transferred model caused by source dataset classes. For example, consider the CIFAR-10 image of a dog in Figure 8.6, which our transfer learning model often mispredicts as a horse. Using our framework, we can demonstrate that this image is strongly negatively influenced by the source class "sorrel horse." Thus, our downstream model may be misusing a feature introduced by this class. Indeed, once we remove "sorrel horse" from the source dataset, our model predicts the correct label more frequently. (See Appendix H.5 for more examples, as well as a quantitative analysis of this experiment.)

Figure 8.7: ImageNet training images with highest positive (**left**) or negative (**right**) example-wise (average) influences on CIFAR-10 test images. We find that ImageNet images that are highly positively influential often correspond to data leakage, while ImageNet images that are highly negatively influential are often either mislabeled, ambiguous, or otherwise misleading. For example, the presence of a flying lawnmower in the ImageNet dataset hurts the downstream performance on a similarly shaped airplane (boxed).

8.3.3 Capability 3: Detecting data leakage and misleading source examples

Thus far, we have focused on how the *classes* in the source dataset influence the predictions of the transferred model on target examples. In this section, we extend our analysis to the *individual* datapoints of the source dataset. We do so by adapting our approach to measure the influence of each individual source datapoint on each target datapoint. Further details on how these influences are computed can be found in Appendix H.4.

Figure 8.7 displays the ImageNet training examples that have highly positive or negative influences on CIFAR-10 test examples. We find that the source images that are highly positively influential are often instances of *data leakage* between the source training set and the target test set. On the other hand, the ImageNet images that are highly negatively influential are typically mislabeled, misleading, or otherwise surprising. For example, the presence of the ImageNet image of a flying lawnmower hurts the performance on a CIFAR-10 image of a regular (but similarly shaped) airplane (see Figure 8.7).

Bibliography

[3DB] 3DB. *Documentation*. URL: https://3db.github.io/3db/.

[ACÖ+17] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. "Towards better understanding of gradient-based attribution methods for deep neural networks". In: *arXiv preprint arXiv:1711.06104* (2017).

[AEI+18] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. "Synthesizing Robust Adversarial Examples". In: *International Conference on Machine Learning (ICML)*. 2018.

[AGM14] Pulkit Agrawal, Ross Girshick, and Jitendra Malik. "Analyzing the performance of multilayer neural networks for object recognition". In: *European conference on computer vision*. 2014.

[AL20] Zeyuan Allen-Zhu and Yuanzhi Li. "Feature Purification: How Adversarial Training Performs Robust Deep Learning". In: 2020. arXiv: 2005.10190 [cs.LG].

[ALG+19] Michael A Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, and Anh Nguyen. "Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[AMK+21] Naveed Akhtar, Ajmal Mian, Navid Kardan, and Mubarak Shah. "Threat of Adversarial Attacks on Deep Learning in Computer Vision: Survey". In: (2021).

[ARS+15] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. "Factors of transferability for a generic convnet representation". In: *IEEE transactions on pattern analysis and machine intelligence* (2015).

[ASK+20] Gunjan Aggarwal, Abhishek Sinha, Nupur Kumari, and Mayank Singh. "On the Benefits of Models with Perceptually-Aligned Gradients". In: *Towards Trustworthy ML Workshop (ICLR)*. 2020.

[ASS+10] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. *Slic superpixels*. Tech. rep. 2010.

[BCM+13] 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: *Joint European conference on machine learning and knowledge discovery in databases (ECML-KDD)*. 2013.

[BG18] Joy Buolamwini and Timnit Gebru. "Gender shades: Intersectional accuracy disparities in commercial gender classification". In: *Conference on fairness, accountability and transparency (FAccT)*. 2018.

[BGH19] Yogesh Balaji, Tom Goldstein, and Judy Hoffman. "Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets". In: *Arxiv preprint arXiv:1910.08051*. 2019.

[BGV14] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. "Food-101–mining discriminative components with random forests". In: *European conference on computer vision*. 2014.

[Ble20] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2020. URL: http://www.blender.org.

[BLW+14] Thomas Berg, Jiongxin Liu, Seung Woo Lee, Michelle L Alexander, David W Jacobs, and Peter N Belhumeur. "Birdsnap: Large-scale fine-grained visual categorization of birds". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014.

[BMA+19] Andrei Barbu, David Mayo, Julian Alverio, William Luo, Christopher Wang, Dan Gutfreund, Josh Tenenbaum, and Boris Katz. "ObjectNet: A large-scale bias-controlled dataset for pushing the limits of object recognition models". In: *Neural Information Processing Systems (NeurIPS)*. 2019.

[BMR+18] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. *Adversarial Patch*. 2018. arXiv: 1712.09665 [cs.CV].

[BMV18] Mitali Bafna, Jack Murtagh, and Nikhil Vyas. "Thwarting Adversarial Examples: An L_0-RobustSparse Fourier Transform". In: *arXiv preprint arXiv:1812.05013* (2018).

[BPR19] Sébastien Bubeck, Eric Price, and Ilya Razenshteyn. "Adversarial examples from computational constraints". In: *International Conference on Machine Learning*. 2019.

[BR18] Battista Biggio and Fabio Roli. "Wild patterns: Ten years after the rise of adversarial machine learning". In: 2018.

[BRZ+20] Emanuel Ben-Baruch, Tal Ridnik, Nadav Zamir, Asaf Noy, Itamar Friedman, Matan Protter, and Lihi Zelnik-Manor. "Asymmetric loss for multi-label classification". In: *arXiv preprint arXiv:2009.14119* (2020).

[BSM+15] Amnon Balanov, Arik Schwartz, Yair Moshe, and Nimrod Peleg. "Image quality assessment based on DCT subband similarity". In: *IEEE International Conference on Image Processing (ICIP)*. 2015.

[BZK+17] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. "Network dissection: Quantifying interpretability of deep visual representations". In: *Computer Vision and Pattern Recognition (CVPR)*. 2017.

[BZM+20] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. "wav2vec 2.0: A framework for self-supervised learning of speech representations". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 12449–12460.

[CCG+19] Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. "Explaining Image Classifiers by Counterfactual Generation". In: *International Conference on Learning Representations (ICLR)*. 2019.

[CFJ+18] Emmanuel Candes, Yingying Fan, Lucas Janson, and Jinchi Lv. "Panning for gold: model-X knockoffs for high dimensional controlled variable selection". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 80.3 (2018), pp. 551–577.

[CK18] Alexis Conneau and Douwe Kiela. "Senteval: An evaluation toolkit for universal sentence representations". In: *Language Resources and Evaluation Conference (LREC)* (2018).

[CMB+16] Brian Chu, Vashisht Madhavan, Oscar Beijbom, Judy Hoffman, and Trevor Darrell. "Best practices for fine-tuning visual classifiers to new domains". In: *European conference on computer vision (ECCV)*. 2016.

[CMK+14] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. "Describing textures in the wild". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014.

[CNA+20] Ping-yeh Chiang, Renkun Ni, Ahmed Abdelkader, Chen Zhu, Christoph Studor, and Tom Goldstein. "Certified defenses for adversarial patches". In: *arXiv preprint arXiv:2003.06693* (2020).

[CNC+23] Nicholas Carlini, Milad Nasr, Christopher A Choquette-Choo, Matthew Jagielski, Irena Gao, Anas Awadalla, Pang Wei Koh, Daphne Ippolito, Katherine Lee, Florian Tramer, et al. "Are aligned neural networks adversarially aligned?" In: *arXiv preprint arXiv:2306.15447* (2023).

[CPK+17] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE transactions on pattern analysis and machine intelligence* (2017).

[CRK19] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. "Certified adversarial robustness via randomized smoothing". In: *International Conference on Machine Learning (ICML)*. 2019.

[CRS+19] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, Percy Liang, and John C Duchi. "Unlabeled data improves adversarial robustness". In: *Neural Information Processing Systems (NeurIPS)*. 2019.

[CSV+14] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Return of the devil in the details: Delving deep into convolutional nets". In: *arXiv preprint arXiv:1405.3531* (2014).

[CW17] Nicholas Carlini and David Wagner. "Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods". In: *Workshop on Artificial Intelligence and Security (AISec)*. 2017.

[CW82] R Dennis Cook and Sanford Weisberg. *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

[CWS+15] Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. "Benchmarking in manipulation research: The YCB object and model set and benchmarking protocols". In: *arXiv preprint arXiv:1502.03143* (2015).

[Dan67] John M. Danskin. *The Theory of Max-Min and its Application to Weapons Allocation Problems*. 1967.

[DB16] Alexey Dosovitskiy and Thomas Brox. "Inverting visual representations with convolutional networks". In: *Computer Vision and Pattern Recognition (CVPR)*. 2016.

[DBK+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *International Conference on Learning Representations (ICLR)*. 2021.

[DCL+18] Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. "Explanations based on the missing: Towards contrastive explanations with pertinent negatives". In: *arXiv preprint arXiv:1802.07623* (2018).

[DCL+19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: (2019).

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition (CVPR)*. 2009.

[DG17] Piotr Dabkowski and Yarin Gal. "Real time image saliency for black box classifiers". In: *Neural Information Processing Systems (NeurIPS)*. 2017.

[DGS19] Shuyang Du, Haoli Guo, and Andrew Simpson. "Self-driving car steering angle prediction based on image recognition". In: *arXiv preprint arXiv:1912.05440* (2019).

[DJV+14] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. "Decaf: A deep convolutional activation feature for generic visual recognition". In: *International conference on machine learning (ICML)*. 2014.

[DLH+16] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. "R-fcn: Object detection via region-based fully convolutional networks". In: *Advances in neural information processing systems (NeurIPS)*. 2016.

[EEF+18a] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. "Robust Physical-World Attacks on Machine Learning Models". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[EEF+18b] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Florian Tramer, Atul Prakash, Tadayoshi Kohno, and Dawn Song. "Physical Adversarial Examples for Object Detectors". In: *CoRR* (2018).

[EIS+19a] Logan Engstrom, Andrew Ilyas, Hadi Salman, Shibani Santurkar, and Dimitris Tsipras. *Robustness (Python Library)*. 2019. URL: https://github.com/MadryLab/robustness.

[EIS+19b] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Brandon Tran, and Aleksander Madry. "Adversarial Robustness as a Prior for Learned Representations". In: *ArXiv preprint arXiv:1906.00945*. 2019.

[EIS+20] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Jacob Steinhardt, and Aleksander Madry. "Identifying Statistical Bias in Dataset Replication". In: *International Conference on Machine Learning (ICML)*. 2020.

[ETT+19] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. "Exploring the Landscape of Spatial Robustness". In: *International Conference on Machine Learning (ICML)*. 2019.

[EVW+10] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision*. 2010.

[FF15] Alhussein Fawzi and Pascal Frossard. "Manitest: Are classifiers really invariant?" In: *British Machine Vision Conference (BMVC)*. 2015.

[FFP04] Li Fei-Fei, Rob Fergus, and Pietro Perona. "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories". In: *2004 conference on computer vision and pattern recognition workshop*. IEEE. 2004, pp. 178–178.

[FV17] Ruth C Fong and Andrea Vedaldi. "Interpretable explanations of black boxes by meaningful perturbation". In: *International Conference on Computer Vision (ICCV)*. 2017.

[FZ20] Vitaly Feldman and Chiyuan Zhang. "What Neural Networks Memorize and Why: Discovering the Long Tail via Influence Estimation". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. 2020, pp. 2881–2891.

[GCL+19] Ruiqi Gao, Tianle Cai, Haochuan Li, Liwei Wang, Cho-Jui Hsieh, and Jason D Lee. "Convergence of Adversarial Training in Overparametrized Networks". In: *arXiv preprint arXiv:1906.07916* (2019).

[GDD+14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *computer vision and pattern recognition (CVPR)*. 2014, pp. 580–587.

[GDG17] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. "Badnets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain". In: *arXiv preprint arXiv:1708.06733* (2017).

[GEB16] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks". In: *computer vision and pattern recognition (CVPR)*. 2016.

[GHP07] Gregory Griffin, Alex Holub, and Pietro Perona. "Caltech-256 object category dataset". In: (2007).

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech Recognition with Deep Recurrent Neural Networks". In: *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. 2013.

[GPM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *neural information processing systems (NeurIPS)*. 2014.

[GRM+19] 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." In: *International Conference on Learning Representations (ICLR)*. 2019.

[GSS15] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *International Conference on Learning Representations (ICLR)*. 2015.

[GWE+19] Yash Goyal, Ziyan Wu, Jan Ernst, Dhruv Batra, Devi Parikh, and Stefan Lee. "Counterfactual visual explanations". In: *arXiv preprint arXiv:1904.07451* (2019).

[GZ19] Amirata Ghorbani and James Zou. "Data shapley: Equitable valuation of data for machine learning". In: *International Conference on Machine Learning (ICML)*. 2019.

[HAE16] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. "What makes ImageNet good for transfer learning?" In: *arXiv preprint arXiv:1608.08614* (2016).

[Hay18] Jamie Hayes. "On visible adversarial perturbations & digital watermarking". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 1597–1604.

[HBM+20] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. *The Many Faces of Robustness: A Critical Analysis of Out-of-Distribution Generalization*. 2020. arXiv: 2006.16241 [cs.CV].

[HCB+18] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism". In: *ArXiv preprint arXiv:1811.06965*. 2018.

[HD19] Dan Hendrycks and Thomas G. Dietterich. "Benchmarking Neural Network Robustness to Common Corruptions and Surface Variations". In: *International Conference on Learning Representations (ICLR)*. 2019.

[HEK+18] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. "A benchmark for interpretability methods in deep neural networks". In: *arXiv preprint arXiv:1806.10758* (2018).

[HGD+17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising Diffusion Probabilistic Models". In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[HRS+17] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. "Speed/accuracy trade-offs for modern convolutional object detectors". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

[HRU+17] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. "Gans trained by a two time-scale update rule converge to a local nash equilibrium". In: *Neural Information Processing Systems (NeurIPS)*. 2017.

[HSS+20] Shahar Hoory, Tzvika Shapira, Asaf Shabtai, and Yuval Elovici. "Dynamic Adversarial Patch for Evading Object Detection Models". In: *arXiv preprint arXiv:2010.13070* (2020).

[HZR+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[IEA+18] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. "Black-box Adversarial Attacks with Limited Queries and Information". In: *International Conference on Machine Learning (ICML)*. 2018.

[IPE+22] Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. "Datamodels: Predicting Predictions from Training Data". In: *International Conference on Machine Learning (ICML)*. 2022.

[IST+19] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. "Adversarial Examples Are Not Bugs, They Are Features". In: *Neural Information Processing Systems (NeurIPS)*. 2019.

[JBZ+19] Jorn-Henrik Jacobsen, Jens Behrmann, Richard Zemel, and Matthias Bethge. "Excessive Invariance Causes Adversarial Vulnerability". In: *International Contemporary on Learning Representations*. 2019.

[JK19] Kyle D Julian and Mykel J Kochenderfer. "Guaranteeing safety for neural network-based aircraft collision avoidance systems". In: *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. IEEE. 2019, pp. 1–10.

[JLH+19] Yunhun Jang, Hankook Lee, Sung Ju Hwang, and Jinwoo Shin. "Learning what and where to transfer". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3030–3039.

[JLT18] Saumya Jetley, Nicholas Lord, and Philip Torr. "With friends like these, who needs adversaries?" In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[JSK+22] Saachi Jain, Hadi Salman, Alaa Khaddaj, Eric Wong, Sung Min Park, and Aleksander Madry. "A Data-Based Perspective on Transfer Learning". In: *arXiv preprint arXiv:2207.05739* (2022).

[JSW+22] 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.

[KBZ+19] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. "Big Transfer (BiT): General Visual Representation Learning". In: *arXiv preprint arXiv:1912.11370* (2019).

[KCL19] Simran Kaur, Jeremy Cohen, and Zachary C. Lipton. "Are Perceptually-Aligned Gradients a General Property of Robust Classifiers?" In: *Arxiv preprint arXiv:1910.08640*. 2019.

[KDI+22] Bojan Karlaš, David Dao, Matteo Interlandi, Bo Li, Sebastian Schelter, Wentao Wu, and Ce Zhang. "Data Debugging with Shapley Importance over End-to-End Machine Learning Pipelines". In: *arXiv preprint arXiv:2204.11131* (2022).

[KDS+13] Jonathan Krause, Jia Deng, Michael Stark, and Li Fei-Fei. "Collecting a large-scale dataset of fine-grained cars". In: (2013).

[KEB+21] Alexander Ke, William Ellsworth, Oishi Banerjee, Andrew Y Ng, and Pranav Rajpurkar. "CheXtransfer: performance and parameter efficiency of ImageNet models for chest X-Ray interpretation". In: *Proceedings of the Conference on Health, Inference, and Learning*. 2021, pp. 116–124.

[KGB16] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. "Adversarial examples in the physical world". In: *arXiv preprint arXiv:1607.02533* (2016).

[KGB17] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial Machine Learning at Scale". In: *International Conference on Learning Representations (ICLR)*. 2017.

[KL17] Pang Wei Koh and Percy Liang. "Understanding Black-box Predictions via Influence Functions". In: *International Conference on Machine Learning*. 2017.

[KMF18] Can Kanbak, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. “Geometric robustness of deep networks: analysis and improvement”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[KP17] Jiman Kim and Chanjong Park. “End-to-end ego lane estimation based on sequential transfer learning for self-driving cars”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2017, pp. 30–38.

[Kri09] Alex Krizhevsky. “Learning Multiple Layers of Features from Tiny Images”. In: *Technical report*. 2009.

[KRJ+22] Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. “Fine-Tuning can Distort Pretrained Features and Underperform Out-of-Distribution”. In: *arXiv preprint arXiv:2202.10054* (2022).

[KSH+19] Daniel Kang, Yi Sun, Dan Hendrycks, Tom Brown, and Jacob Steinhardt. “Testing Robustness Against Unforeseen Adversaries”. In: *ArXiv preprint arxiv:1908.08016*. 2019.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2012.

[KSJ19] Beomsu Kim, Junghoon Seo, and Taegyun Jeon. “Bridging Adversarial Robustness and Gradient Interpretability”. In: *International Conference on Learning Representations Workshop on Safe Machine Learning (ICLR SafeML)*. 2019.

[KSL19] Simon Kornblith, Jonathon Shlens, and Quoc V Le. “Do better imagenet models transfer better?” In: *computer vision and pattern recognition (CVPR)*. 2019.

[LAG+19] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. “Certified robustness to adversarial examples with differential privacy”. In: *Symposium on Security and Privacy (SP)*. 2019.

[LCH+16] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. “Visualizing and Understanding Neural Models in NLP”. In: *Proceedings of NAACL-HLT*. 2016, pp. 681–691.

[LCL+17] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. “Delving into Transferable Adversarial Examples and Black-box Attacks”. In: *International Conference on Learning Representations (ICLR)*. 2017.

[LDG+17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. “Feature pyramid networks for object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[LF20a] Alexander Levine and Soheil Feizi. “(De) Randomized Smoothing for Certifiable Defense against Patch Attacks”. In: *arXiv preprint arXiv:2002.10733* (2020).

[LF20b] Alexander Levine and Soheil Feizi. "Robustness certificates for sparse adversarial attacks by randomized ablation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 4585–4593.

[LIE+22] Guillaume Leclerc, Andrew Ilyas, Logan Engstrom, Sung Min Park, Hadi Salman, and Aleksander Madry. *ffcv*. https://github.com/libffcv/ffcv/. 2022.

[LK19] Mark Lee and Zico Kolter. "On physical adversarial patches for object detection". In: *arXiv preprint arXiv:1906.11897* (2019).

[LLW+15] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. "Deep Learning Face Attributes in the Wild". In: *International Conference on Computer Vision (ICCV)*. 2015.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context". In: *European conference on computer vision (ECCV)*. 2014.

[LRM15] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. "Bilinear cnn models for fine-grained visual recognition". In: *Proceedings of the IEEE international conference on computer vision*. 2015.

[LSI+21] 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". In: *arXiv preprint arXiv:2106.03805*. 2021.

[LSK19] Juncheng Li, Frank R. Schmidt, and J. Zico Kolter. "Adversarial camera stickers: A physical camera-based attack on deep learning systems". In: *Arxiv preprint arXiv:1904.00759*. 2019.

[LYL+18] Xin Liu, Huanrui Yang, Ziwei Liu, Linghao Song, Hai Li, and Yiran Chen. "Dpatch: An adversarial patch attack on object detectors". In: *arXiv preprint arXiv:1806.02299* (2018).

[MG21] Dina Mardaoui and Damien Garreau. "An analysis of lime for text data". In: *International Conference on Artificial Intelligence and Statistics*. 2021.

[MGM18] Romain Mormont, Pierre Geurts, and Raphaël Marée. "Comparison of deep transfer learning strategies for digital pathology". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018.

[MGR+18] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. "Exploring the limits of weakly supervised pretraining". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.

[Mil95] George A Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* (1995).

[MIM+18] Nikhil Muralidhar, Mohammad Raihanul Islam, Manish Marwah, Anuj Karpatne, and Naren Ramakrishnan. "Incorporating prior domain knowledge into deep neural networks". In: *2018 IEEE International Conference on Big Data (Big Data)*. 2018.

[MMK+18] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. "Virtual adversarial training: a regularization method for supervised and semi-supervised learning". In: 2018.

[MMS+18] 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 (ICLR)*. 2018.

[MRK+13] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. "Fine-grained visual classification of aircraft". In: *arXiv preprint arXiv:1306.5151* (2013).

[MV15] Aravindh Mahendran and Andrea Vedaldi. "Understanding deep image representations by inverting them". In: *computer vision and pattern recognition (CVPR)*. 2015.

[Nak19] Preetum Nakkiran. "Adversarial robustness may be at odds with simplicity". In: *arXiv preprint arXiv:1901.00532*. 2019.

[Nes03] Yurii Nesterov. *Introductory Lectures on Convex Optimization*. 2003.

[NKP19] Muzammal Naseer, Salman Khan, and Fatih Porikli. "Local gradients smoothing: Defense against localized adversarial attacks". In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2019, pp. 1300–1307.

[NPV+18] Jiquan Ngiam, Daiyi Peng, Vijay Vasudevan, Simon Kornblith, Quoc V Le, and Ruoming Pang. "Domain adaptive transfer learning with specialist models". In: *arXiv preprint arXiv:1811.07056* (2018).

[NRK+21] Muzammal Naseer, Kanchana Ranasinghe, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. "Intriguing Properties of Vision Transformers". In: *arXiv preprint arXiv:2105.10497* (2021).

[NST+93] Tomoyuki Nishita, Takao Sirai, Katsumi Tadamura, and Eihachiro Nakamae. "Display of the Earth Taking into Account Atmospheric Scattering". In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1993.

[NVZ+19] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. "Mitsuba 2: A Retargetable Forward and Inverse Renderer". In: *Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 38.6 (2019). DOI: 10.1145/3355089.3356498.

[NZ08] Maria-Elena Nilsback and Andrew Zisserman. "Automated flower classification over a large number of classes". In: *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*. 2008.

[Pan] Pandas3D. *The Open Source Framework for 3D Rendering and Games*. URL: https://www.panda3d.org/.

[PMG+17] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. "Practical black-box attacks against machine learning". In: *Asia Conference on Computer and Communications Security*. 2017.

[PMG16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. "Transferability in Machine Learning: from Phenomena to Black-box Attacks using Adversarial Samples". In: *ArXiv preprint arXiv:1605.07277*. 2016.

[PSS99] A. J. Preetham, Peter Shirley, and Brian Smits. "A Practical Analytic Model for Daylight". In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1999.

[PVZ+12] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. "Cats and dogs". In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 3498–3505.

[RBK+18] Rafael Reisenhofer, Sebastian Bosse, Gitta Kutyniok, and Thomas Wiegand. "A Haar wavelet-based perceptual similarity index for image quality assessment". In: 2018.

[RBL+22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. "High-resolution image synthesis with latent diffusion models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 10684–10695.

[RDN+22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. "Hierarchical text-conditional image generation with clip latents". In: *arXiv preprint arXiv:2204.06125* (2022).

[RDS+15] 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". In: *International Journal of Computer Vision (IJCV)*. 2015.

[RHC+21] Evani Radiya-Dixit, Sanghyun Hong, Nicholas Carlini, and Florian Tramèr. "Data Poisoning Won't Save You From Facial Recognition". In: arXiv, 2021.

[RHG+15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems (NeurIPS)*. 2015.

[RJG+19] Anurag Ranjan, Joel Janai, Andreas Geiger, and Michael J Black. "Attacking optical flow". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 2404–2413.

[RKH+21] 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: *arXiv preprint arXiv:2103.00020*. 2021.

[RRS+19] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. "Do ImageNet Classifiers Generalize to ImageNet?" In: *International Conference on Machine Learning (ICML)*. 2019.

[RSG16a] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "" Why should I trust you?" Explaining the predictions of any classifier". In: *International Conference on Knowledge Discovery and Data Mining (KDD)*. 2016.

[RSG16b] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *International Conference on Knowledge Discovery and Data Mining (KDD)*. 2016.

[RSL18] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. "Certified defenses against adversarial examples". In: *International Conference on Learning Representations (ICLR)*. 2018.

[RWK20] Leslie Rice, Eric Wong, and J. Zico Kolter. "Overfitting in adversarially robust deep learning". In: *Arxiv preprint arXiv:2002.11569*. 2020.

[RXY+19] Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John C Duchi, and Percy Liang. "Adversarial Training Can Hurt Generalization". In: *arXiv preprint arXiv:1906.06032* (2019).

[RZT18] Amir Rosenfeld, Richard Zemel, and John K. Tsotsos. "The Elephant in the Room". In: *arXiv preprint arXiv:1808.03305*. 2018.

[SAS+14] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. "CNN features off-the-shelf: an astounding baseline for recognition". In: *conference on computer vision and pattern recognition (CVPR) workshops*. 2014.

[SB06] H.R. Sheikh and A.C. Bovik. "Image information and visual quality". In: 2006.

[SBB+16] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. "Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. 2016, pp. 1528–1540.

[SBL+18] Mehdi S. M. Sajjadi, Olivier Bachem, Mario Lučić, Olivier Bousquet, and Sylvain Gelly. "Assessing Generative Models via Precision and Recall". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[SBM+16] Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Müller. "Evaluating the visualization of what a deep neural network has learned". In: *IEEE transactions on neural networks and learning systems* 28.11 (2016), pp. 2660–2673.

[SIE+20] Hadi Salman, Andrew Ilyas, Logan Engstrom, Ashish Kapoor, and Aleksander Madry. "Do Adversarially Robust ImageNet Models Transfer Better?" In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[SIE+21] Hadi Salman, Andrew Ilyas, Logan Engstrom, Sai Vemprala, Aleksander Madry, and Ashish Kapoor. "Unadversarial examples: Designing objects for robust vision". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 15270–15284.

[SJI+22] Hadi Salman, Saachi Jain, Andrew Ilyas, Logan Engstrom, Eric Wong, and Aleksander Madry. "When does Bias Transfer in Transfer Learning?" In: *arXiv preprint arXiv:2207.02842*. 2022.

[SJW+21] Hadi Salman, Saachi Jain, Eric Wong, and Aleksander Mądry. "Certified patch robustness via smoothed vision transformers". In: *arXiv preprint arXiv:2110.07719* (2021).

[SKL+23] Hadi Salman, Alaa Khaddaj, Guillaume Leclerc, Andrew Ilyas, and Aleksander Madry. "Raising the cost of malicious ai-powered image editing". In: *International Conference on Machine Learning (ICML)* (2023).

[SLL20] Pascal Sturmfels, Scott Lundberg, and Su-In Lee. "Visualizing the Impact of Feature Attribution Baselines". In: *Distill* (2020). https://distill.pub/2020/attribution-baselines. DOI: 10.23915/distill.00022.

[SLR+19] 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: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[SMG+20] Krishna Kumar Singh, Dhruv Mahajan, Kristen Grauman, Yong Jae Lee, Matt Feiszli, and Deepti Ghadiyaram. "Don't judge an object by its context: learning to overcome contextual bias". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11070–11078.

[SSS+17] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. "Revisiting unreasonable effectiveness of data in deep learning era". In: *Proceedings of the IEEE international conference on computer vision*. 2017.

[SSY+20] Hadi Salman, Mingjie Sun, Greg Yang, Ashish Kapoor, and J Zico Kolter. "Denoised smoothing: A provable defense for pretrained classifiers". In: *Advances in Neural Information Processing Systems* 33 (2020).

[SSZ+19] Ali Shafahi, Parsa Saadatpanah, Chen Zhu, Amin Ghiasi, Christoph Studer, David Jacobs, and Tom Goldstein. "Adversarially robust transfer learning". In: *arXiv preprint arXiv:1905.08232* (2019).

[STK+17] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. "SmoothGrad: removing noise by adding noise". In: *ICML workshop on visualization for deep learning*. 2017.

[STT+19] Shibani Santurkar, Dimitris Tsipras, Brandon Tran, Andrew Ilyas, Logan Engstrom, and Aleksander Madry. "Image Synthesis with a Single (Robust) Classifier". In: *Neural Information Processing Systems (NeurIPS)*. 2019.

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. “Axiomatic attribution for deep networks”. In: *International Conference on Machine Learning (ICML)*. 2017.

[SVI+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. “Rethinking the inception architecture for computer vision”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2016.

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps”. In: *arXiv preprint arXiv:1312.6034* (2013).

[SWM+15] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. “Deep Unsupervised Learning Using Nonequilibrium Thermodynamics”. In: *International Conference on Machine Learning*. 2015.

[SYZ+19] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. “A convex relaxation barrier to tight robustness verification of neural networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)* (2019).

[SZ15] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *International Conference on Learning Representations (ICLR)*. 2015.

[SZC+18] Dong Su, Huan Zhang, Hongge Chen, Jinfeng Yi, Pin-Yu Chen, and Yupeng Gao. “Is Robustness the Cost of Accuracy? A Comprehensive Study on the Robustness of 18 Deep Image Classification Models”. In: *European Conference on Computer Vision (ECCV)*. 2018.

[SZS+14] 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 (ICLR)*. 2014.

[TCB+20] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. “On adaptive attacks to adversarial example defenses”. In: *arXiv preprint arXiv:2002.08347* (2020).

[TCD+20] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. “Training data-efficient image transformers & distillation through attention”. In: *arXiv preprint arXiv:2012.12877* (2020).

[TDS+20] Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. “Measuring Robustness to Natural Distribution Shifts in Image Classification”. In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[TE11] Antonio Torralba and Alexei A Efros. “Unbiased look at dataset bias”. In: *CVPR 2011*. 2011.

[TKH+21] Philipp Terhörst, Jan Niklas Kolf, Marco Huber, Florian Kirchbuchner, Naser Damer, Aythami Morales, Julian Fierrez, and Arjan Kuijper. "A comprehensive study on face recognition biases beyond demographics". In: *arXiv preprint arXiv:2103.01592* (2021).

[TSE+19] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. "Robustness May Be at Odds with Accuracy". In: *International Conference on Learning Representations (ICLR)*. 2019.

[UKE+20] Francisco Utrera, Evan Kravitz, N. Benjamin Erichson, Rajiv Khanna, and Michael W. Mahoney. "Adversarially-Trained Deep Nets Transfer Better". In: *ArXiv preprint arXiv:2007.05869*. 2020.

[UVL17] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Deep Image Prior". In: *ArXiv preprint arXiv:1711.10925*. 2017.

[VM01] David A Van Dyk and Xiao-Li Meng. "The art of data augmentation". In: *Journal of Computational and Graphical Statistics*. 2001.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems* (2017).

[WAL19] Sherrie Wang, George Azzari, and David B Lobell. "Crop type mapping without field-level labels: Random forest transfer and unsupervised clustering techniques". In: *Remote sensing of environment* 222 (2019), pp. 303–317.

[Wal45] Abraham Wald. "Statistical Decision Functions Which Minimize the Maximum Risk". In: *Annals of Mathematics*. 1945.

[WBS+04] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: 2004.

[WEL47] B. L. WELCH. "THE GENERALIZATION OF 'STUDENT'S' PROBLEM WHEN SEVERAL DIFFERENT POPULATION VARLANCES ARE INVOLVED". In: *Biometrika*. 1947.

[Wen21] Lilian Weng. "What are diffusion models?" In: *lilianweng.github.io* (July 2021). URL: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/.

[WH13] Alexander Wilkie and Lukas Hosek. "Predicting Sky Dome Appearance on Earth-like Extrasolar Worlds". In: *29th Spring conference on Computer Graphics (SCCG 2013)*. 2013.

[WHS23] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. "Jailbroken: How Does LLM Safety Training Fail?" In: *arXiv preprint arXiv:2307.02483* (2023).

[Wig19] Ross Wightman. *PyTorch Image Models*. https://github.com/rwightman/pytorch-image-models. 2019. DOI: 10.5281/zenodo.4414861.

[WK18] Eric Wong and J Zico Kolter. "Provable defenses against adversarial examples via the convex outer adversarial polytope". In: *International Conference on Machine Learning (ICML)*. 2018.

[WKM+19] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[WLD+20] Zuxuan Wu, Ser-Nam Lim, Larry S Davis, and Tom Goldstein. "Making an invisibility cloak: Real world adversarial attacks on object detectors". In: *European Conference on Computer Vision*. Springer. 2020, pp. 1–17.

[WPL+17] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M Summers. "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases". In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 2017.

[WSG17] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. "A-Fast-RCNN: Hard Positive Generation via Adversary for Object Detection". In: *ArXiv preprint arXiv:1704.03414*. 2017.

[WSM21] Eric Wong, Shibani Santurkar, and Aleksander Madry. "Leveraging Sparse Linear Layers for Debuggable Deep Networks". In: *International Conference on Machine Learning (ICML)*. 2021.

[WSS+20] Eric Wong, Tim Schneider, Joerg Schmitt, Frank R Schmidt, and J Zico Kolter. "Neural network virtual sensors for fuel injection quantities with provable performance specifications". In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2020, pp. 1753–1758.

[WZC+18] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. "Towards fast computation of certified robustness for ReLU networks". In: *International Conference on Machine Learning (ICML)*. 2018.

[XBK+15] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. "Show, attend and tell: Neural image caption generation with visual attention". In: *International conference on machine learning*. PMLR. 2015, pp. 2048–2057.

[XBS+21] Chong Xiang, Arjun Nitin Bhagoji, Vikash Sehwag, and Prateek Mittal. "PatchGuard: A Provably Robust Defense against Adversarial Patches via Small Receptive Fields and Masking". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021.

[XEI+20] Kai Xiao, Logan Engstrom, Andrew Ilyas, and Aleksander Madry. "Noise or signal: The role of image backgrounds in object recognition". In: *arXiv preprint arXiv:2006.09994* (2020).

[XHE+10] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. "Sun database: Large-scale scene recognition from abbey to zoo". In: *Computer Vision and Pattern Recognition (CVPR)*. 2010.

[XJB+16] Michael Xie, Neal Jean, Marshall Burke, David Lobell, and Stefano Ermon. "Transfer learning from deep features for remote sensing and poverty mapping". In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[XZM+14] Wufeng Xue, Lei Zhang, Xuanqin Mou, and Alan C. Bovik. "Gradient Magnitude Similarity Deviation: A Highly Efficient Perceptual Image Quality Index". In: 2014.

[YCB+14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems (NeurIPS)*. 2014.

[YDH+20] Greg Yang, Tony Duan, J. Edward Hu, Hadi Salman, Ilya Razenshteyn, and Jerry Li. *Randomized Smoothing of All Shapes and Sizes*. 2020.

[ZCA+17] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. "Visualizing deep neural network decisions: Prediction difference analysis". In: *arXiv preprint arXiv:1702.04595* (2017).

[ZF14] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. "Wide residual networks". In: *arXiv preprint arXiv:1605.07146* (2016).

[ZL12] Lin Zhang and Hongyu Li. "SR-SIM: A fast and high performance IQA index based on spectral residual". In: *2012 19th IEEE International Conference on Image Processing*. 2012.

[ZML+07] Jianguo Zhang, Marcin Marszałek, Svetlana Lazebnik, and Cordelia Schmid. "Local features and kernels for classification of texture and object categories: A comprehensive study". In: *International journal of computer vision*. 2007.

[ZPD+20] Yi Zhang, Orestis Plevrakis, Simon S. Du, Xingguo Li, Zhao Song, and Sanjeev Arora. "Over-parameterized Adversarial Training: An Analysis Overcoming the Curse of Dimensionality". In: *Arxiv preprint arXiv:2002.06668*. 2020.

[ZQP+20] Yu Zhang, James Qin, Daniel S Park, Wei Han, Chung-Cheng Chiu, Ruoming Pang, Quoc V Le, and Yonghui Wu. "Pushing the limits of semi-supervised learning for automatic speech recognition". In: *arXiv preprint arXiv:2010.10504* (2020).

[ZSL14] Lin Zhang, Ying Shen, and Hongyu Li. "VSI: A Visual Saliency-Induced Index for Perceptual Image Quality Assessment". In: 2014.

[ZSS+18] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. "Taskonomy: Disentangling task transfer learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

[ZWK+23] Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson. "Universal and Transferable Adversarial Attacks on Aligned Language Models". In: *arXiv preprint arXiv:2307.15043* (2023).

[ZXY17] Zhuotun Zhu, Lingxi Xie, and Alan Yuille. “Object Recognition without and without Objects”. In: *International Joint Conference on Artificial Intelligence*. 2017.

[ZYJ+19] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. “Theoretically Principled Trade-off between Robustness and Accuracy”. In: *International Conference on Machine Learning (CIML)*. 2019.

[ZYM+20] Zhanyuan Zhang, Benson Yuan, Michael McCoyd, and David Wagner. “Clipped BagNet: defending against sticker attacks with clipped bag-of-features”. In: *2020 IEEE Security and Privacy Workshops (SPW)*. 2020.

[ZZ19] Tianyuan Zhang and Zhanxing Zhu. “Interpreting Adversarially Trained Convolutional Neural Networks”. In: *International Conference on Machine Learning (ICML)*. 2019.

[ZZM+11] Lin Zhang, Lei Zhang, Xuanqin Mou, and David Zhang. “FSIM: A Feature Similarity Index for Image Quality Assessment”. In: 2011.

Appendix

Appendix A

Additional details for Chapter 1

A.1 Experimental setup

A.1.1 Models and architectures

We use three sizes of vision transformers—ViT-Tiny (ViT-T), ViT-Small (ViT-S), and ViT-Base (ViT-B) models [Wig19; DBK+21] and compare to to residual networks of similar (or larger) size—ResNet-18, ResNet-50 [HZR+16], and Wide ResNet-101-2 [ZK16], respectively. These architectures and their corresponding number of parameters are summarized in Table A.1.

Table A.1: A collection of neural network architectures we use in this chapter.

Architecture	ViT-T	ResNet-18	ViT-S	ResNet-50	ViT-B	WRN-101-2
Params	5M	12M	22M	26M	86M	126M

We use the same architectures for both ImageNet and CIFAR-10 models, and finetune our smoothed models from publicly released checkpoints pretrained on ImageNet. All our CIFAR-10 experiments are thus conducted on up-sampled CIFAR-10 images of size 224×224.

A.1.2 Datasets

We use two datasets:

1. CIFAR [Kri09] https://paperswithcode.com/dataset/cifar-10.
2. ImageNet [RDS+15], with a custom (research, non-commercial) license, as found here https://paperswithcode.com/dataset/imagenet.

A.1.3 Training parameters

Derandomized smoothing requires that the base classifier predict well on image ablations. A standard technique for derandomized smoothing methods is to directly train the base classifier on image ablations [LF20a]. Thus, unless otherwise stated, in each epoch we randomly apply a column ablation of fixed width to each image of the training set.

To facilitate training of the base classifiers, we start from pretrained ResNets[1] and ViT architectures[2] and fine-tune as follows:

ImageNet. We train for 30 epochs using SGD of fixed learning rate of 10^{-3}, a batch size of 256, a weight-decay of 10^{-4}, a momentum of 0.9, and with column ablations of fixed width $b = 19$. For data-augmentation, we use random resized crop, random horizontal flip, and color jitter. We then apply column ablations.

CIFAR-10. We train for 30 epochs using SGD with a step learning rate of 10^{-2} that drops every 10 epochs by a factor of 10, a batch size of 128, a weight-decay of 5×10^{-4}, a momentum of 0.9, and with column ablations of fixed width $b = 4$. We only use random horizontal flip for data-augmentation, after which we apply column ablations. We then upsample all CIFAR-10 images to 224×224 (on GPU).

Training time. Training is relatively fast, with our largest ImageNet model (WRN-101-2) finishing in roughly two days on one NVIDIA V100 GPU. The smaller models such as ViT-T or ResNet-18 finish training in only a few hours.

A.1.4 Compute and timing experiments

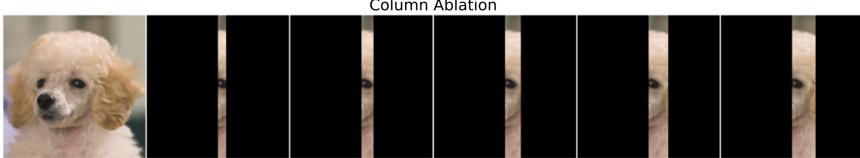
We use an internal cluster containing NVIDIA 1080-TI, 2080-TI, V100, and A100 GPUs. Scalability and timing experiments were performed on an A100 and averaged over 50 trials. When performing scalability experiments, we do not include data loading time or the time to move the input to the GPU.

A.1.5 Example ablations

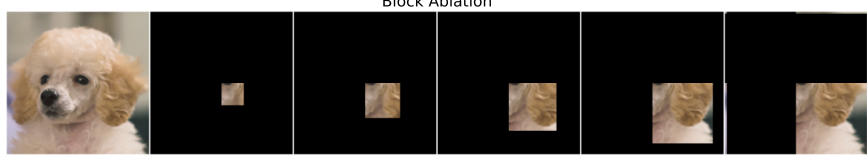
In Figure A.1, we display examples of ablations of various types (column, block) and sizes.

[1] These are TorchVision's official checkpoints, and can be found here `https://pytorch.org/vision/stable/models.html`.

[2] We use the DeiT checkpoints of [Wig19] which can be found here `https://github.com/rwightman/pytorch-image-models/blob/master/timm/models/vision_transformer.py`.



(a) **Column ablations** with the following ablation size from left to right: original image, 13px, 19px, 25px, 31px, 37px.



(b) **Block ablations** with the following ablation size from left to right: original image, 35px, 55px, 75px, 95px, 115px.

Figure A.1: Example ablations that we use in this chapter.

A.1.6 Differences in setup from Levine and Feizi [LF20a]

Our work builds on top of that of Levine and Feizi [LF20a]. We use their robustness guarantee as is (see Section 1.1.1), but there are a few differences in the setup of our experiments. All experimental results (including the de-randomized smoothing baseline) are run using the same experimental setup in order to remain fair, which only improved the baseline over what was previously reported in the literature. For completeness, we describe the differences in setup here.

Encoding *null* inputs. The first difference is that Levine and Feizi [LF20a] encode part of the input as being *null* or ablated by adding additional color channels, as described in [LF20b], so that the *null* value is distinct from all real pixel colors. In practice, we found this to be unnecessary, and were able to replicate their results with ablations that simply replace masked pixels with 0.

Early stopping. We find that ResNets substantially benefit from early stopping when trained with ablations, and otherwise experience severe overfitting to the ablations with substantially reduced test accuracies. In our replication, we find that the ResNet-50 result reported by Levine and Feizi [LF20a] can be substantially improved with an earlier checkpoint (improving certified accuracy by nearly 10%), and thus we use early-stopping in all of our ResNet baselines.

Starting from pretrained models. To reduce training time, for both ImageNet and CIFAR-10 experiments, we start from pre-trained ImageNet checkpoints (see Section A.1.3). This step is especially necessary for the CIFAR-10 experiments, as it is quite challenging to train a ViT from scratch on CIFAR-10 (these models tend to require a large amount of data).

Upsampled CIFAR-10. In order to use the pretrained ImageNet checkpoints when training our base classifiers for CIFAR-10, we (nearest neighbor) upsample the CIFAR-10 inputs to 224×224 as part of the model architecture. We verify robustness in the original 32×32 images.

Sweeping over ablation size. We note that Levine and Feizi [LF20a] tested various ablations sizes only on CIFAR-10. Due to our speed-ups, we were able to sweep over ablations sizes for ImageNet.

A.2 Ablation sweeps

In this section, we further explore the impact of changing the ablation size on both standard and certified performance. In Section A.2.1, we explore the effect of modifying the ablation size at training time. In Section A.2.2, similar to the experiment on ImageNet from Section 1.2.2, we present additional results on adjusting the ablation size at test time for CIFAR10.

A.2.1 Train-time ablation

We first explore varying the ablation size used during training for ImageNet. Specifically, we train and certify a ResNet-50 and ViT-S over a range of column widths from 1 to 67 pixels (Figure A.2).

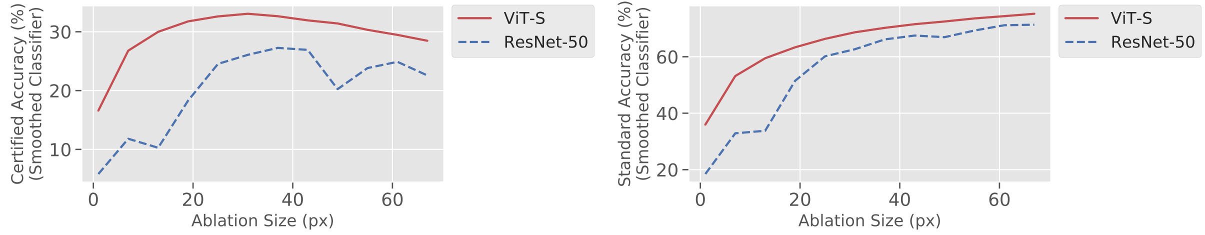


Figure A.2: Certified and standard accuracy for a smoothed model trained and evaluated on ImageNet column ablations with varying widths. The ResNet-50 requires a substantially larger ablation size for certification, whereas the ViT-S is more flexible.

For ViTs, we find that once the columns are wide enough, we see only marginal improvements in certified accuracy (i.e. only 1.3% higher certified accuracy over $b = 19$). This suggests that small ablations are sufficient at training time, allowing for fast training of ViTs when using cropped ablations.

On the other hand, ResNets require a substantially larger column width than was previously explored. Specifically, the certified accuracy of the ResNet baseline can be greatly improved from 18% to 27% when the ablation size is increased to $b = 37$. This ablation size is optimal for the ResNet, but is still 6% lower certified accuracy when compared to the ViT.

Overall, we find that certified performance of ViTs on ImageNet remains largely stable with respect to the column ablation size used for training. We can thus use smaller ablation sizes during training (e.g $b = 19$) to improve training speed while certifying using larger ablation sizes.

A.2.2 Test-time ablations

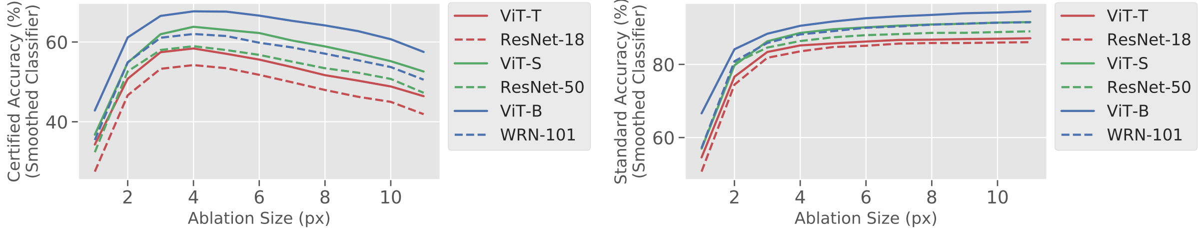


Figure A.3: Certified and standard accuracy for a smoothed model on CIFAR-10 trained with a fixed ablation size ($b = 5$), and evaluated with varying ablation sizes.

Similar to the experiment on ImageNet from Section 1.2.2, we present analogous results for varying the ablation size used at test time for CIFAR-10. These results largely reflect what was previously observed by Levine and Feizi [LF20a]. Specifically, the optimal ablation size for CIFAR10 is a column width of $b = 4$, with a steep drop-off in performance for larger ablation sizes. This is in contrast to what we observed in ImageNet, which did not see such a steep drop in performance.

A.3 Dropping tokens for ViTs

We first describe the algorithm for processing image ablations with a ViT while dropping masked tokens. Let $\mathbf{x}$ be an image with size $h \times w$, and let $\mathcal{S}(\mathbf{x})$ be the set of image ablations

of $\mathbf{x}$. For each $\mathbf{z}, \mathbf{m} \in \mathcal{S}(\mathbf{x})$, $\mathbf{z}$ is an image ablation of size $h \times w$ and $\mathbf{m} \in \{0,1\}^{h \times w}$ is its corresponding mask, such that $\mathbf{m}_{ij}$ is 0 if the i, j pixel in $\mathbf{z}$ is masked and 1 otherwise.

Recall that a ViT has two stages when processing an input $\mathbf{z}$.

- **Encoding:** $\mathbf{z}$ is split into patches of $p \times p$ and positionally encoded into tokens. We let $E(\mathbf{w}, i, j)$ be an encoder which positionally encodes the $p \times p$ sized patch $\mathbf{w}$ which was at spatial location ip, jp in $\mathbf{z}$.
- **Self-Attention:** A set of positionally encoded tokens $\mathcal{T}$ is passed through self attention layers V and produces a class label.

Given an image ablation $\mathbf{z}$ we modify the ViT to remove tokens in $\mathcal{T}$ that correspond to a fully masked region in $\mathbf{z}$.

Algorithm 2 Forward pass for processing an image ablation $\mathbf{z}$ with mask $\mathbf{m}$ using a ViT while dropping masked tokens.

```
1: function ProcessAblation(z, m)
2:     T = {} Initialize set of tokens for an ablation
3:     for i, j ∈ [h/p] × [w/p] do
4:         if not m_{ip:(i+1)p, jp:(j+1)p} = 0 then
5:             T = T ∪ E(z_{ip:(i+1)p, jp:(j+1)p}, i, j)
6:         end if
7:     end for
8:     return V(T)
9: end function
```

We can then use this function to define the smoothed ViT.

Algorithm 3 Forward pass for a smoothed ViT on an input image $\mathbf{x}$ with ablation set $\mathcal{S}(\mathbf{x})$

```
1: function SmoothedViT(x)
2:     c_i = 0 for i ∈ [k] // Initialize counts to zero
3:     for z, m ∈ S(x) do
4:         y = ProcessAblation(z, m)
5:         c_y = c_y + 1 // Update counts
6:     end for
7:     return arg max_y c_y
8: end function
```

A.3.1 Computational complexity of ViTs with dropped tokens

We can now derive the computational complexity of the smoothed ViT when dropping tokens. Specifically, consider a ViT that divides an $h \times w$ pixel image into $p \times p$ patches,

and positionally encodes them tokens with d hidden dimensions.

Recall that a ViT has two operation types: *attention operators* which scale quadratically with the number of tokens but linearly with hidden dimension d and *fully-connected operators* which scale linearly with the number of tokens but quadratically in d. Without dropping tokens, we have hw/p^2 tokens. A forward pass of processing an image ablation without dropping tokens thus has an overall complexity of

$$O\left(\left(\frac{hw}{p^2}\right)^2 d + \left(\frac{hw}{p^2}\right) d^2\right)$$

where the first term corresponds to the attention operations, and the second term corresponds to the fully-connected operations.

For column ablations with width b, dropping masked tokens reduces the number of tokens to hb/p^2. The complexity of the forward pass to process an image ablation when dropping masked tokens (i.e `ProcessAblation`) then drops to

$$O\left(\left(\frac{hb}{p^2}\right)^2 d + \left(\frac{hb}{p^2}\right) d^2\right)$$

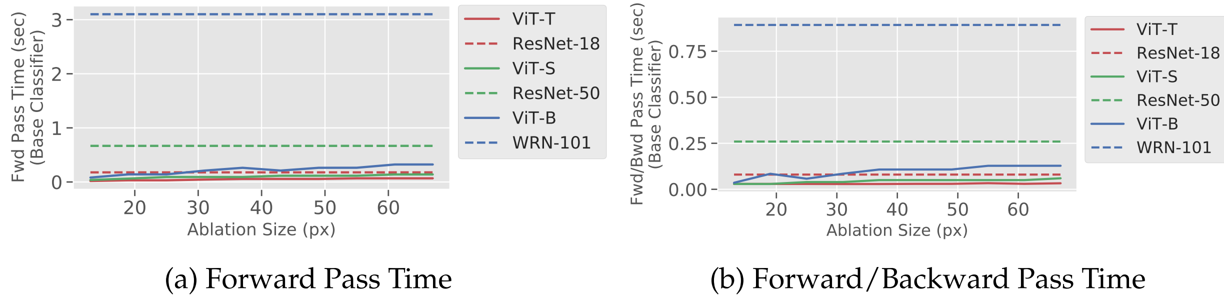
thus reducing the attention cost by a factor of $O(w^2/b^2)$ and the fully-connected cost by a factor of $O(w/b)$. In practice, the computation of fully-connected operations tends to dominate since $d > \frac{hw}{p^2}$.

Overall, a smoothed ViT with stride s processes w/s ablations. Thus, the overall complexity of the smoothed ViT is:

$$O\left(\frac{w}{s}\left(\left(\frac{hb}{p^2}\right)^2 d + \left(\frac{hb}{p^2}\right) d^2\right)\right)$$

A.3.2 Effect of dropping tokens on speed

We extend the timing experiments comparing ViTs and ResNets to a range of ablation sizes (previously presented in Table 1.3 from Section 1.3 for a single column ablation size of $b = 19$). Empirically, even for substantially larger ablations, we find significantly faster training and inference times for ViTs over ResNets. In Figure A.4, we compare the evaluation and training speeds for processing image ablations with ResNets and ViTs with dropped tokens.

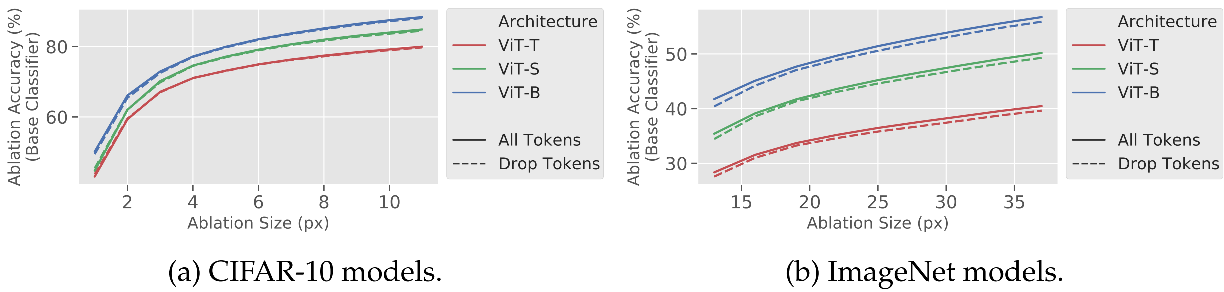


(a) Forward Pass Time (b) Forward/Backward Pass Time

Figure A.4: (a) Average time for computing a forward pass on a batch of 1024 image ablations on ImageNet (b) Average time for computing a full training step (forward and backward pass) on a batch of 128 image ablations on ImageNet

A.3.3 Effect of dropping tokens on performance

Since the tokens are individually positionally encoded, dropping tokens that are fully masked does not remove any information from the input. In Figure A.5, we confirm that dropping masked tokens does not significantly change the accuracy of the ViT base classifier on ablations.



(a) CIFAR-10 models. (b) ImageNet models.

Figure A.5: We compare the ablation accuracies of dropping masked tokens versus processing all tokens for a collection of vision transformers on CIFAR-10 and ImageNet. Dropping masked tokens does not substantially degrade accuracy.

A.4 Strided ablations

In this section, we explore strided ablations for certification in more depth. In Section A.4.1 we present the threshold for certification when using strided ablations. In Section A.4.2 we show how striding affects performance.

A.4.1 Certification thresholds for strided ablation sets

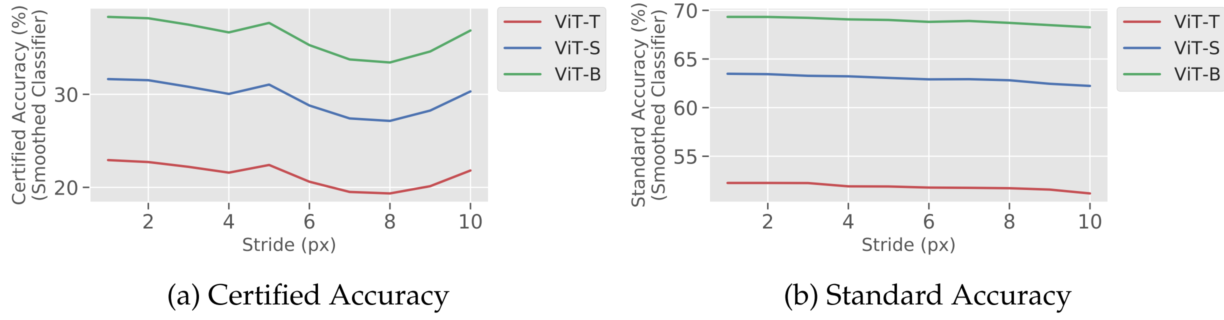
We briefly describe the new thresholds for certification when using strided ablations. Recall from (1.2) that a prediction is certified robust if

$$n_c(\mathbf{x}) > \max_{c' \neq c} n_{c'}(\mathbf{x}) + 2\Delta.$$

Thus Δ, the number of ablations that a patch can intersect, fully describes the certification threshold.

Column smoothing. For column smoothing with width b and stride s, the maximum number of ablations that an $m \times m$ patch can intersect with is at most $\Delta_{column+stride} = \lceil (m + s - 1)/s \rceil$.

A.4.2 Performance under strided ablations



(a) Certified Accuracy (b) Standard Accuracy

Figure A.6: Certified and standard accuracy of various ViTs for ImageNet when using strided column ablations with varying stride lengths.

In this section, we explore how striding affects standard and certified performance. We find that striding does not result in a monotonic change in certified accuracy—certification accuracy can both decrease and increase as the stride increases.

For a few choices in striding, it is possible to not substantially change the accuracy of the ViT at classifying ablations, as shown in Figure A.6. For example, a ViT-B which normally obtains 38.3% certified accuracy without striding, maintains certified accuracies of 37.6% at stride $s = 5$ and 36.8% at stride $s = 10$. For these small drops in certified accuracy, striding directly enables 5x or 10x faster inference times.

A.5 Block smoothing

In this section, we investigate an alternative type of smoothing known as *Block Smoothing*, previously investigated in the CIFAR-10 setting [LF20a]. In block smoothing, we ablate (square) blocks of pixels instead of columns of pixels. This procedure is prohibitively expensive for ImageNet due to its quadratic complexity. For example, smoothing a 224×224 image with block ablations takes a majority vote over $224 \times 224 = 50,176$ ablations, which is four orders of magnitude slower than a standard forward pass. We alleviate this obstacle for larger image settings such as ImageNet with the token-based speedups for ViTs from Section 1.3.1 and the striding from Section 1.3.2. In combination, these improvements in speed allow us to perform a practical investigation into block smoothing on ImageNet.

Certification. Certification of derandomized smoothing models with block ablations is similar to that of column ablations, and depends on the maximum number of ablations in the ablation set that an adversarial patch can simultaneously intersect. Recall that for column ablations of size b, the certification threshold is $\Delta = m + b - 1$ ablations. For block ablations of size b (where b here is the side of the retained block/square of pixels), $\Delta = (m + b - 1)^2$. The threshold can then be plugged as before into Equation (1.2) to check whether the model is certifiably robust.

A.5.1 Practical inference speeds for block smoothing

We first demonstrate how dropping masked tokens significantly increases the speed of evaluating block ablations for the base classifier. In Figure A.7, we show that dropping masked tokens substantially reduces the time needed to process 1024 block ablations for various sizes of ViTs. This directly leads to a 4.85x speedup for ViT-S with ablation size 75.

Even with this optimization, however, block smoothing is quite expensive. A forward pass through the smoothed model still requires around 50k passes through the base classifier. We thus leverage our second speedup from strided ablations and use *strided* block smoothing. Similar to strided column ablations, for a stride length of s, we only consider block ablations that are s pixels apart, vertically and horizontally. This changes the certification threshold Δ to be, $\Delta_{block+stride} = \lceil (m + s - 1)/s \rceil^2$. With dropping fully masked tokens and using a stride of 10, a smoothed ViT-S using an ablation size of 75 is only 2.8x slower than a standard (non-robust) ResNet-50.

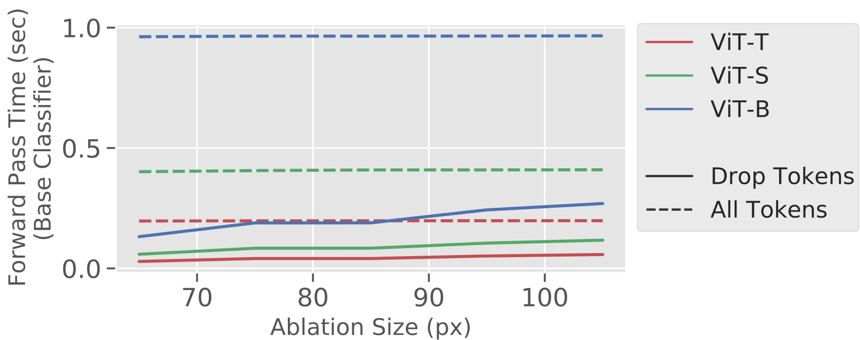
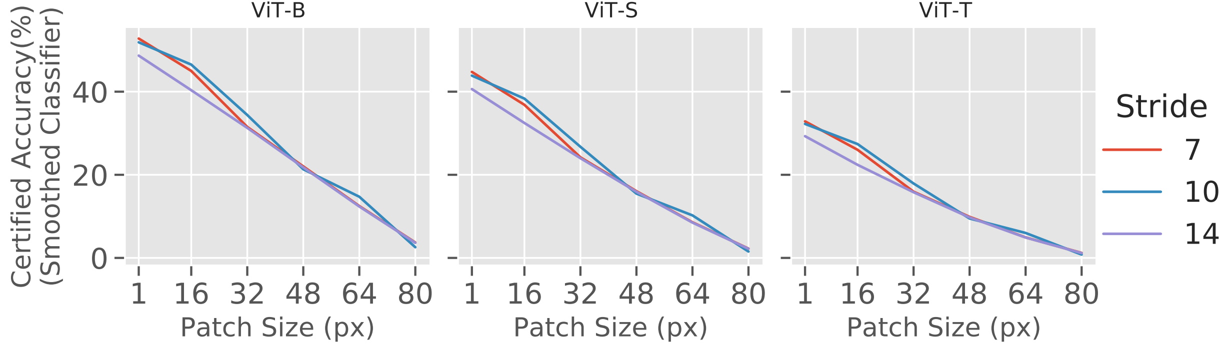


Figure A.7: Average time to compute a forward pass for ViTs on 1024 block ablated images with varying ablation sizes with and without dropping masked tokens.

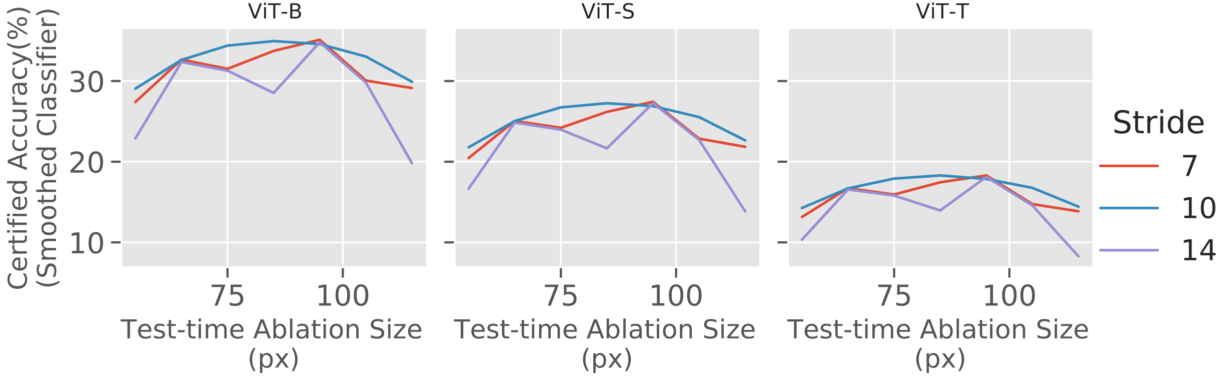
Certified accuracy. We find that, despite an systematic search over stride length and block size (both at training and evaluation), block smoothing on ImageNet remains significantly worse than column smoothing. For example, with optimal stride and ablation size, we see up to 5% lower certified accuracy than column smoothing on the largest model, ViT-B. We checked a range of ablation sizes from 55 to 115 as well as three stride lengths $\{7, 10, 14\}$ (Figure A.8).

Similar to striding with column ablations, there is a significant amount of variability based on the stride length. To pinpoint the effect of striding, we certify one of the best-performing block sizes ($b = 75$) over a full range of strides from $s = 1$ to $s = 20$ (Figure A.9). This is a fairly expensive calculation, as using stride $s = 1$ corresponds to the full block ablation with 50k ablations.

Even when using all possible block ablations ($s = 1$), block smoothing does not improve over column smoothing. However, we do find that certain stride lengths ($s = 18$) can achieve similar performance to non-strided block ablations ($s = 1$), which means that we can speed up certification (by 18x) without sacrificing certified accuracy. Thus, while our methods can make block smoothing computationally feasible, further investigation is needed to make block smoothing match column smoothing in terms of certified and standard accuracies.



(a) We fix the test-time ablation size at $b = 75$ and plot the certified accuracy as a function of the adversarial patch size, for various stride length.



(b) We fix the adversarial patch size $m = 32$ and plot the certified accuracy as a function of the test-time ablation size, for various stride length.

Figure A.8: Strided block smoothing on ImageNet for a collection of ViT models trained with block ablations of size $b = 75$.

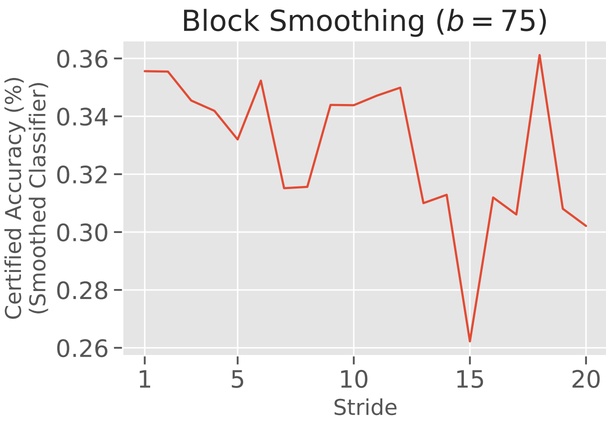


Figure A.9: Strided block smoothing on ImageNet for ViT-B with a fixed ablation size $b = 75$. The reported certified accuracy are against adversarial patches of size 32×32. Note how some stride lengths ($s = 18$ for example) can achieve similar performance to non-strided block ablations ($s = 1$).

A.6 Extended experimental results

Table A.2: **An extended version of Table 1.2.** Summary of our CIFAR-10 results and comparisons to certified patch defenses from the literature: Clipped Bagnet (CBG), Derandomized Smoothing (DS), and PatchGuard (PG). b is the column ablation size out of 32 pixels.

Standard and Certified Accuracy on CIFAR-10 (%)			
Patch Size	Clean	2×2	4×4
Baselines			
CBN [ZYM+20]	84.2	44.2	9.3
DS [LF20a]	83.9	68.9	56.2
PG [XBS+21] (2×2)	84.7	69.2	0.0
PG [XBS+21] (4×4)	84.6	57.7	57.7
Smoothed models			
ResNet-18 (b = 4)	83.6	67.0	54.2
ViT-T (b = 4)	**85.5**	**70.0**	**58.5**
ResNet-50 (b = 4)	86.4	71.6	59.0
ViT-S (b = 4)	**88.4**	**75.0**	**63.8**
WRN-101-2 (b = 4)	88.2	73.9	62.0
ViT-B (b = 4)	**90.8**	**78.1**	**67.6**

Table A.3: Standard accuracies of regularly trained architectures vs. smoothed architectures with column ablations of size $b = 4$ for CIFAR-10 and $b = 19$ for ImageNet.

		Standard accuracy of architecture (%)					
		ViT-T	ResNet-18	ViT-S	ResNet-50	ViT-B	WRN-101-2
ImageNet	Standard	72.03	69.76	79.72	76.13	81.74	78.85
	Smoothed	52.25	50.62	63.48	51.47	69.33	61.38
	Difference	19.77	19.14	16.24	24.66	12.41	17.47
CIFAR-10	Standard	93.13	95.72	93.33	96.16	97.07	97.85
	Smoothed	85.53	88.41	86.39	83.57	90.75	88.20
	Difference	7.60	7.31	6.94	12.59	6.32	9.65

Table A.4: **An extended version of Table 1.1.** Summary of our ImageNet results and comparisons to certified patch defenses from the literature: Clipped Bagnet (CBG), Derandomized Smoothing (DS), and PatchGuard (PG). Time refers to the inference time for a batch of 1024 images, b is the ablation size, and s is the ablation stride.

Standard and Certified Accuracy on ImageNet (%)					
Patch Size	Clean	1% pixels	2% pixels	3% pixels	Time (sec)
CBN [ZYM+20]	49.5	13.4	7.1	3.1	3.05 ± 0.02
DS [LF20a]	44.4	17.7	14.0	11.2	149.52 ± 0.33
PG [XBS+21] (1% pixels)	55.1	32.3	0.0	0.0	3.05 ± 0.02
PG [XBS+21] (2% pixels)	54.6	26.0	26.0	0.0	3.05 ± 0.02
PG [XBS+21] (3% pixels)	54.1	19.7	19.7	19.7	3.05 ± 0.02
Vary Ablation Size (Stride = 1)					
ResNet-18 (b = 19)	50.6	24.1	19.8	16.9	39.84 ± 0.97
ResNet-18 (b = 25)	52.7	24.2	20.0	17.1	39.84 ± 0.97
ResNet-18 (b = 37)	54.3	22.4	18.6	15.7	39.84 ± 0.97
ViT-T (b = 19)	52.3	27.3	22.9	19.9	6.81 ± 0.05
ViT-T (b = 25)	53.7	26.9	22.8	19.7	6.82 ± 0.05
ViT-T (b = 37)	55.6	25.5	21.7	18.8	12.64 ± 0.10
ResNet-50 (b = 19)	51.5	22.8	18.3	15.3	149.52 ± 0.33
ResNet-50 (b = 25)	54.7	23.8	19.5	16.4	149.52 ± 0.33
ResNet-50 (b = 37)	57.8	23.1	19.0	16.1	149.52 ± 0.33
ViT-S (b = 19)	63.5	36.8	31.6	27.9	14.00 ± 0.16
ViT-S (b = 25)	65.1	36.8	31.9	28.2	20.58 ± 0.18
ViT-S (b = 37)	67.1	35.3	30.7	27.1	20.61 ± 0.16
WRN-101-2 (b = 19)	61.4	33.3	28.1	24.1	694.50 ± 0.58
WRN-101-2 (b = 25)	64.2	34.3	29.1	25.3	694.50 ± 0.58
WRN-101-2 (b = 37)	67.2	33.7	28.8	25.2	694.50 ± 0.58
ViT-B (b = 19)	69.3	43.8	38.3	34.3	31.51 ± 0.17
ViT-B (b = 25)	71.1	44.0	38.8	34.8	31.52 ± 0.21
ViT-B (b = 37)	73.2	43.0	38.2	34.1	58.74 ± 0.17
Vary Ablation Stride					
WRN-101-2 (b = 19, s = 5)	61.1	30.1	27.3	21.9	138.90 ± 0.12
WRN-101-2 (b = 19, s = 10)	59.7	25.8	25.8	20.9	69.45 ± 0.06
ViT-B (b = 19, s = 5)	69.0	40.6	37.7	32.0	6.30 ± 0.03
ViT-B (b = 19, s = 10)	68.3	36.9	36.9	31.4	3.15 ± 0.02
WRN-101-2 (b = 37, s = 5)	66.9	32.6	27.2	24.7	138.90 ± 0.12
WRN-101-2 (b = 37, s = 10)	66.1	31.7	26.7	21.7	69.45 ± 0.06
ViT-B (b = 37, s = 5)	73.1	41.9	36.4	33.5	11.75 ± 0.03
ViT-B (b = 37, s = 10)	72.6	41.3	36.1	30.8	5.87 ± 0.02

Appendix B

Additional details for Chapter 2

B.1 Experimental Setup

B.1.1 Pretrained ImageNet models

In this chapter, we train a number of standard and robust ImageNet models on various architectures. These models are used for all the various transfer learning experiments.

Architectures We experiment with several standard architectures from the PyTorch's Torchvision[1]. These models are shown in Tables B.1&B.2.[2]

Table B.1: The clean accuracies of ℓ_∞-robust ImageNet classifiers.

	Clean ImageNet Top-1 Accuracy (%)				
	Robustness parameter ε				
Model	$\frac{0.5}{255}$	$\frac{1}{255}$	$\frac{2}{255}$	$\frac{4}{255}$	$\frac{8}{255}$
ResNet-18	66.13	63.46	59.63	52.49	42.11
ResNet-50	73.73	72.05	69.10	63.86	54.53
WRN-50-2	75.82	74.65	72.35	68.41	60.82

Training details We fix the training procedure for all of these models. We train all the models from scratch using SGD with batch size of 512, momentum of 0.9, and weight

[1]These models can be found here https://pytorch.org/docs/stable/torchvision/models.html
[2]WRN-50-2 and WRN-50-4 refer to Wide-ResNet-50, twice and four times as wide, respectively.

Table B.2: The clean accuracies of standard and ℓ_2-robust ImageNet classifiers used in this chapter.

Clean ImageNet Top-1 Accuracy (%)										
	Robustness parameter ε									
Model	0	0.01	0.03	0.05	0.1	0.25	0.5	1	3	5
ResNet-18	69.79	69.90	69.24	69.15	68.77	67.43	65.49	62.32	53.12	45.59
ResNet-50	75.80	75.68	75.76	75.59	74.78	74.14	73.16	70.43	62.83	56.13
WRN-50-2	76.97	77.25	77.26	77.17	76.74	76.21	75.11	73.41	66.90	60.94
WRN-50-4	77.91	78.02	77.87	77.77	77.64	77.10	76.52	75.51	69.67	65.20

Clean ImageNet Top-1 Accuracy (%)						
	Model Architecture					
	A **DenseNet-161**	**B** **ResNeXt50**	**C** **VGG16-bn**	**D** **MobileNet-v2**	**E** **ShuffleNet**	**F** **MNASNET**
$\varepsilon = 0$	77.37	77.32	73.66	65.26	64.25	60.97
$\varepsilon = 3$	66.12	65.92	56.78	50.05	42.87	41.03

decay of $1e-4$. We train for 90 epochs with an initial learning rate of 0.1 that drops by a factor of 10 every 30 epochs.

For **Standard Training**, we use the standard cross-entropy multi-class classification loss. For **Robust Training**, we use adversarial training [MMS+18]. We train on adversarial examples generated within maximum allowed perturbations ℓ_2 of $\varepsilon \in \{0.01, 0.03, 0.05, 0.1, 0.25, 0.5, 1, 3, 5\}$ and ℓ_∞ perturbations of $\varepsilon \in \{\frac{0.5}{255}, \frac{1}{255}, \frac{2}{255}, \frac{4}{255}, \frac{8}{255}\}$ using 3 attack steps and a step size of $\frac{\varepsilon \times 2}{3}$.

B.1.2 ImageNet transfer to classification datasets

Datasets

We test transfer learning starting from ImageNet pretrained models on classification datasets that are used in [KSL19]. These datasets vary in size the number of classes and datapoints. The details are shown in Table B.3.

Table B.3: Classification datasets used in this chapter.

Dataset	Classes	Size (Train/Test)	Accuracy Metric
Birdsnap [BLW+14]	500	32,677/8,171	Top-1
Caltech-101 [FFP04]	101	3,030/5,647	Mean Per-Class
Caltech-256 [GHP07]	257	15,420/15,187	Mean Per-Class
CIFAR-10 [Kri09]	10	50,000/10,000	Top-1
CIFAR-100 [Kri09]	100	50,000/10,000	Top-1
Describable Textures (DTD) [CMK+14]	47	3,760/1,880	Top-1
FGVC Aircraft [MRK+13]	100	6,667/3,333	Mean Per-Class
Food-101 [BGV14]	101	75,750/25,250	Top-1
Oxford 102 Flowers [NZ08]	102	2,040/6,149	Mean Per-Class
Oxford-IIIT Pets [PVZ+12]	37	3,680/3,669	Mean Per-Class
SUN397 [XHE+10]	397	19,850/19,850	Top-1
Stanford Cars [KDS+13]	196	8,144/8,041	Top-1

Fixed-feature Transfer

For this type of transfer learning, we *freeze* the weights of the ImageNet pretrained model[3], and replace the last fully connected layer with a random initialized one that fits the transfer dataset. We train only this new layer for 150 epochs using SGD with batch size of 64, momentum of 0.9, weight decay of $5e-4$, and an initial lr $\in \{0.01, 0.001\}$ that drops by a factor of 10 every 50 epochs. We use the following standard data-augmentation methods:

```
TRAIN_TRANSFORMS = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
])
TEST_TRANSFORMS = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor()
])
```

Full-network transfer

For full-network transfer learning, we use the exact same hyperparameters as the fixed-feature setting, but we *do not freeze* the weights of the pretrained ImageNet model.

[3]For all of our experiments, we do not freeze the batch statistics, only its weights.

B.1.3 Unifying dataset scale

For this experiment, we follow the exact experimental setup of B.1.2 with the only modification being resizing all the datasets to 32×32 then do dataugmentation as before:

```
TRAIN_TRANSFORMS = transforms.Compose([
    transforms.Resize(32),
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
])
TEST_TRANSFORMS = transforms.Compose([
    transforms.Resize(32),
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor()
])
```

B.1.4 Replicate our results

We desired simplicity and kept reproducibility in our minds when conducting our experiments, so we use standard hyperparameters and minimize the number of tricks needed to replicate our results. We open source all the standard and robust ImageNet models that we use in this chapter, and our code is available at https://github.com/Microsoft/robust-models-transfer.

B.2 Transfer Learning with ℓ_∞-robust ImageNet models

We investigate how well other types of robust ImageNet models do in transfer learning.

Table B.4: Transfer Accuracy of standard vs ℓ_∞-robust ImageNet models on CIFAR-10 and CIFAR-100.

			Transfer Accuracy (%)					
			Robustness parameter ε					
Dataset	Transfer Type	Model	0.0	$\frac{0.5}{255}$	$\frac{1.0}{255}$	$\frac{2.0}{255}$	$\frac{4.0}{255}$	$\frac{8.0}{255}$
CIFAR-10	**Full-network**	**ResNet-18**	96.05	96.85	96.80	96.98	**97.04**	96.79
		ResNet-50	97.14	97.69	97.84	97.98	97.92	98.01
	Fixed-feature	**ResNet-18**	75.02	87.13	89.01	89.07	**90.56**	89.18
		ResNet-50	78.16	90.55	91.51	92.74	93.35	93.68
CIFAR-100	**Full-network**	**ResNet-18**	81.70	83.66	83.46	**83.98**	83.55	82.82
		ResNet-50	84.75	86.12	86.48	87.06	86.90	86.21
	Fixed-feature	**ResNet-18**	53.86	68.52	70.83	72.00	**72.19**	69.78
		ResNet-50	55.57	72.89	74.16	76.22	77.17	76.70

B.3 Object Detection and Instance Segmentation

In this section we provide more experimental details, and results, relating to our object detection and instance segmentation experiments.

Experimental setup. We use only standard configurations from Detectron2[4] to train models. For COCO tasks, compute limitations made training from every ε initialization impossible. Instead, we trained from every ε initialization using a reduced learning rate schedule (the corresponding 1x learning rate schedule in Detectron2) before training from the top three ε initializations (by Box AP) along with the standard model using the full learning rate training schedule (the 3x schedule). Our results for the 1x learning rate search are in Figure B.1; our results, similar to those in Section 2.3, show that training from a robustly trained backbone yields greater AP than training from a standard-trained backbone.

[4]See: https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md For all COCO tasks we used "R50-FPN" configurations (1x and 3x, described further in this section), and for VOC we used the "R50-C4" configuration.

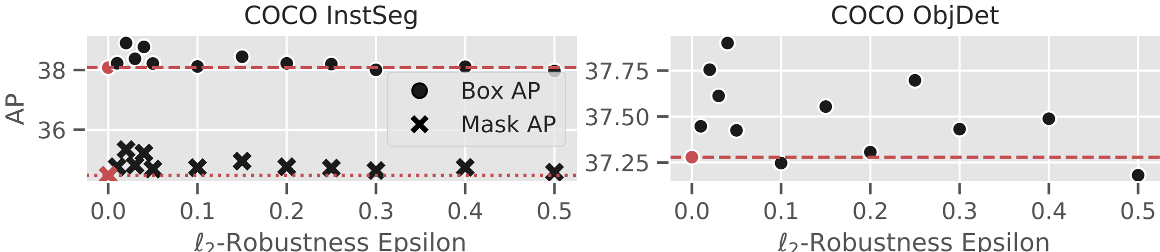


Figure B.1: AP of instance segmentation and object detection models with backbones initialized with ε-robust models before training. Robust backbones generally lead to better AP, and the best robust backbone always outperforms the standard-trained backbone for every task.

Baselines. We use standard ResNet-50 models from the torchvision package[5] using the Robustness library [EIS+19a]. Detectron2 models were originally trained for (and their configurations are tuned for) ResNet-50 models from the original ResNet code release[6], which are slightly different from the torchvision ResNet-50s we use. It has been previously noted that models trained from torchvision perform worse with Detectron2 than these original models[7]. Despite this, the best torchvision ResNet-50 models we train from robust initializations dominate (without any additional hyperparameter searching) the original baselines except for the COCO Object Detection task in terms of AP, in which the original baseline has 0.07 larger Box AP[8].

[5]https://pytorch.org/docs/stable/torchvision/index.html
[6]https://github.com/KaimingHe/deep-residual-networks
[7]See for both previous note and model differences: https://github.com/facebookresearch/detectron2/blob/master/tools/convert-torchvision-to-d2.py
[8]Baselines found here: https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md

B.4 Background on Adversarially Robust Models

Adversarial examples in computer vision. Adversarial examples [BCM+13; SZS+14] (also referred to as *adversarial attacks*) are imperceptible perturbations to natural inputs that induce misbehaviour from machine learning—in this context computer vision—systems. An illustration of such an attack is shown in Figure B.2. The discovery of adversarial examples was a major contributor to the rise of *deep learning security*, where prior work has focused on both robustifying models against such attacks (cf. [GSS15; MMS+18; WK18; RSL18; CRK19] and their references), as well as testing the robustness of machine learning systems in "real-world" settings (cf. [PMG+17; AEI+18; IEA+18; LSK19; EEF+18a] and their references). A model that is resilient to such adversarial examples is referred to as "adversarially robust."

Robust optimization and adversarial training. One of the canonical methods for training an adversarially robust model is robust optimization. Typically, we train deep learning models using empirical risk minimization (ERM) over the training data—that is, we solve:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(x_i, y_i; \theta),$$

where θ represents the model parameters, $\mathcal{L}$ is a task-dependent loss function (e.g., cross-entropy loss for classification), and $\{(x_i, y_i) \sim \mathcal{D}\}$ are training image-label pairs. In robust optimization (dating back to the work of Wald [Wal45]), we replace this standard ERM objective with a *robust* risk minimization objective:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \max_{x'; d(x_i, x') < \varepsilon} \mathcal{L}(x', y_i),$$

where d is a fixed but arbitrary norm. (In practice, d is often assumed to be an ℓ_p norm for $p \in \{2, \infty\}$—for the majority of this work we set $p = 2$, so $d(x, x')$ is the Euclidean norm.) In short, rather than minimizing the loss on only the training points, we instead minimize the worst-case loss over a ball around each training point. Assuming the robust objective generalizes, it ensures that an adversary cannot perturb a given test point $(x, y) \sim \mathcal{D}$ and drastically increase the loss of the model. The parameter ε governs the desired robustness of the model: $\varepsilon = 0$ corresponds to standard (ERM) training, and increasing ε results in models that are stable within larger and larger radii.

At first glance, it is unclear how to effectively solve the robust risk minimization problem posed above—typically we use SGD to minimize risk, but here the loss function

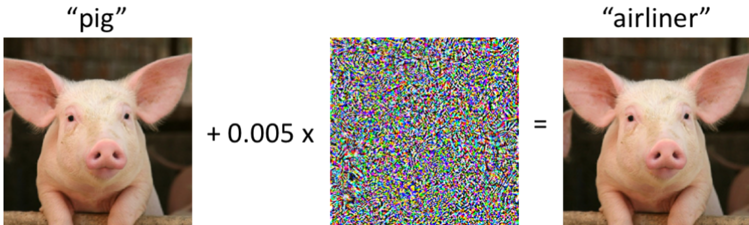


Figure B.2: An example of an adversarial attack: adding the imperceptible perturbation (middle) to a correctly classified pig (left) results in a near-identical image that is classified as "airliner" by an Inception-v3 ImageNet model.

has an embedded maximization, so the corresponding SGD update rule would be:

$$\theta_t \leftarrow \theta_{t-1} - \eta \cdot \nabla_\theta \left(\max_{x'; d(x', x_i) < \varepsilon} \mathcal{L}(x', y_i; \theta) \right).$$

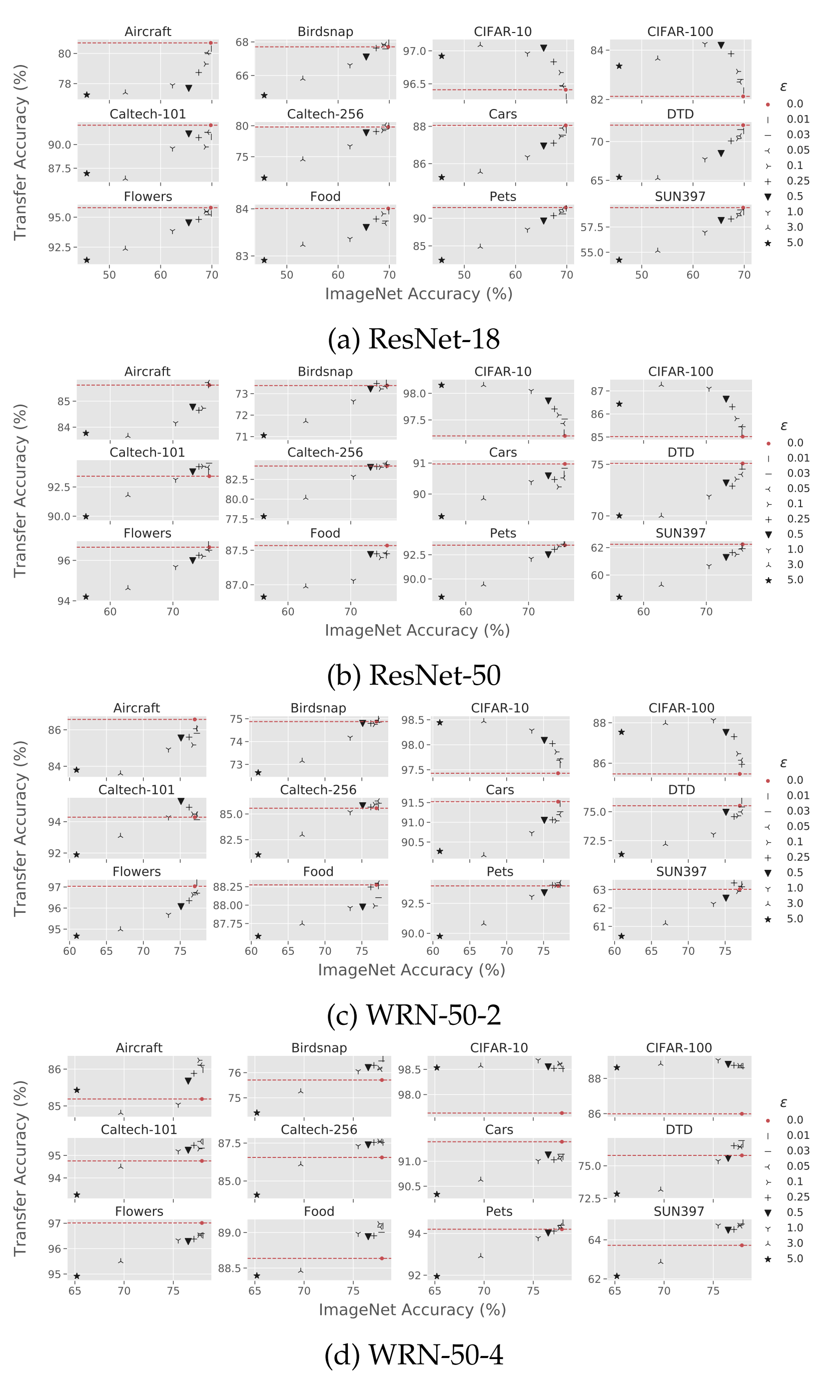
Thus, to actually train an adversarially robust neural network, Madry et al. [MMS+18] turn to inspiration from robust convex optimization, where Danskin's theorem [Dan67] says that for a function $f(\alpha, \beta)$ that is convex in α,

$$\nabla_\alpha \left(\max_{\beta \in B} f(\alpha, \beta) \right) = \nabla_\alpha f(\alpha, \beta^*), \qquad \text{where } \beta^* = \arg\max_\beta f(\alpha, \beta) \text{ and } B \text{ is compact.}$$

Danskin's theorem thus allows us to write the gradient of a minimax problem in terms of only the gradient of the inner objective, evaluated at its maximal point. Carrying this intuition over to the neural network setting (despite the lack of convexity) results in the popular *adversarial training* algorithm [GSS15; MMS+18], where at each training iteration, worst-case (adversarial) inputs are passed to the neural network rather than standard unmodified inputs. Despite its simplicity, adversarial training remains a competitive baseline for training adversarially robust networks [RWK20]. Furthermore, recent works have provided theoretical evidence for the success of adversarial training directly in the neural network setting [GCL+19; AL20; ZPD+20].

B.5 Omitted Figures

B.5.1 Full-network Transfer: additional results to Figure 2.5



(a) ResNet-18

(b) ResNet-50

(c) WRN-50-2

(d) WRN-50-4

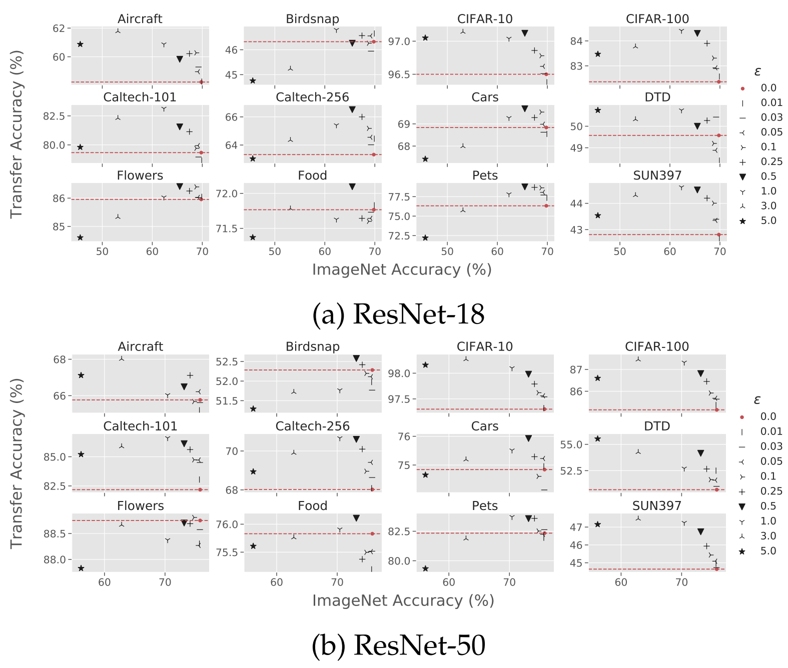
Figure B.3: **Full-network** transfer accuracies of standard and robust ImageNet models to various image classification datasets.

B.5.2 Varying architecture: additional results to Table 2.2

Table B.5: Source (ImageNet) and target accuracies, fixing robustness (ε) but varying architecture. When robustness is controlled for, ImageNet accuracy is highly predictive of (full-network) transfer performance.

		Architecture (see details in Appendix C.2.1)						
Robustness	**Dataset**	A	B	C	D	E	F	R^2
Std ($\varepsilon = 0$)	ImageNet	77.37	77.32	73.66	65.26	64.25	60.97	—
	CIFAR-10	97.84	97.47	96.08	95.86	95.82	95.55	0.79
	CIFAR-100	86.53	85.53	82.07	80.02	80.76	80.41	0.82
	Caltech-101	94.78	94.63	91.32	88.91	87.13	83.28	0.94
	Caltech-256	86.22	86.33	82.23	76.51	75.81	74.90	0.98
	Cars	91.28	91.27	90.97	88.31	85.81	84.54	0.91
	Flowers	97.93	97.29	96.80	96.25	95.40	72.06	0.44
	Pets	94.55	94.26	92.63	89.78	88.59	82.69	0.87
Adv ($\varepsilon = 3$)	ImageNet	66.12	65.92	56.78	50.05	42.87	41.03	—
	CIFAR-10	98.67	98.22	97.27	96.91	96.23	95.99	0.97
	CIFAR-100	88.65	88.32	84.14	83.32	80.92	80.52	0.97
	Caltech-101	93.84	93.31	89.93	89.02	83.29	75.52	0.83
	Caltech-256	84.35	83.05	78.19	74.08	69.19	70.04	0.99
	Cars	90.91	90.08	89.67	88.02	83.57	78.76	0.79
	Flowers	95.77	96.01	93.88	94.25	91.47	26.98	0.38
	Pets	91.85	91.46	88.06	85.63	80.92	64.90	0.72

B.5.3 Unified scale: additional results to Figure 2.7

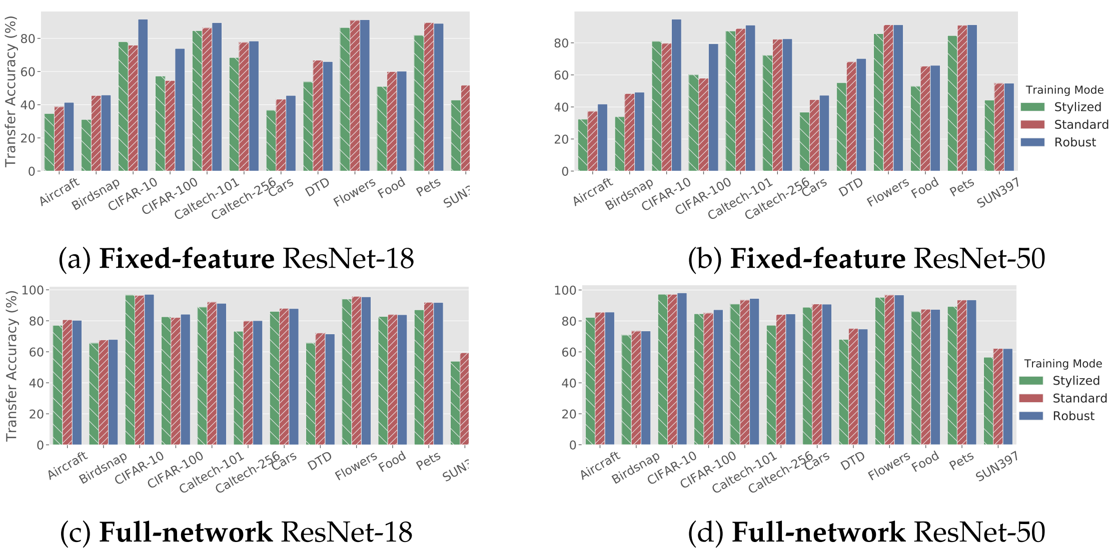


(a) ResNet-18

(b) ResNet-50

Figure B.4: **Full-network** transfer accuracies of various datasets that are down-scaled to 32×32 before being up-scaled again to ImageNet scale and used for transfer learning.

B.5.4 Stylized ImageNet Transfer: additional results to Figure 2.8b



(a) **Fixed-feature** ResNet-18

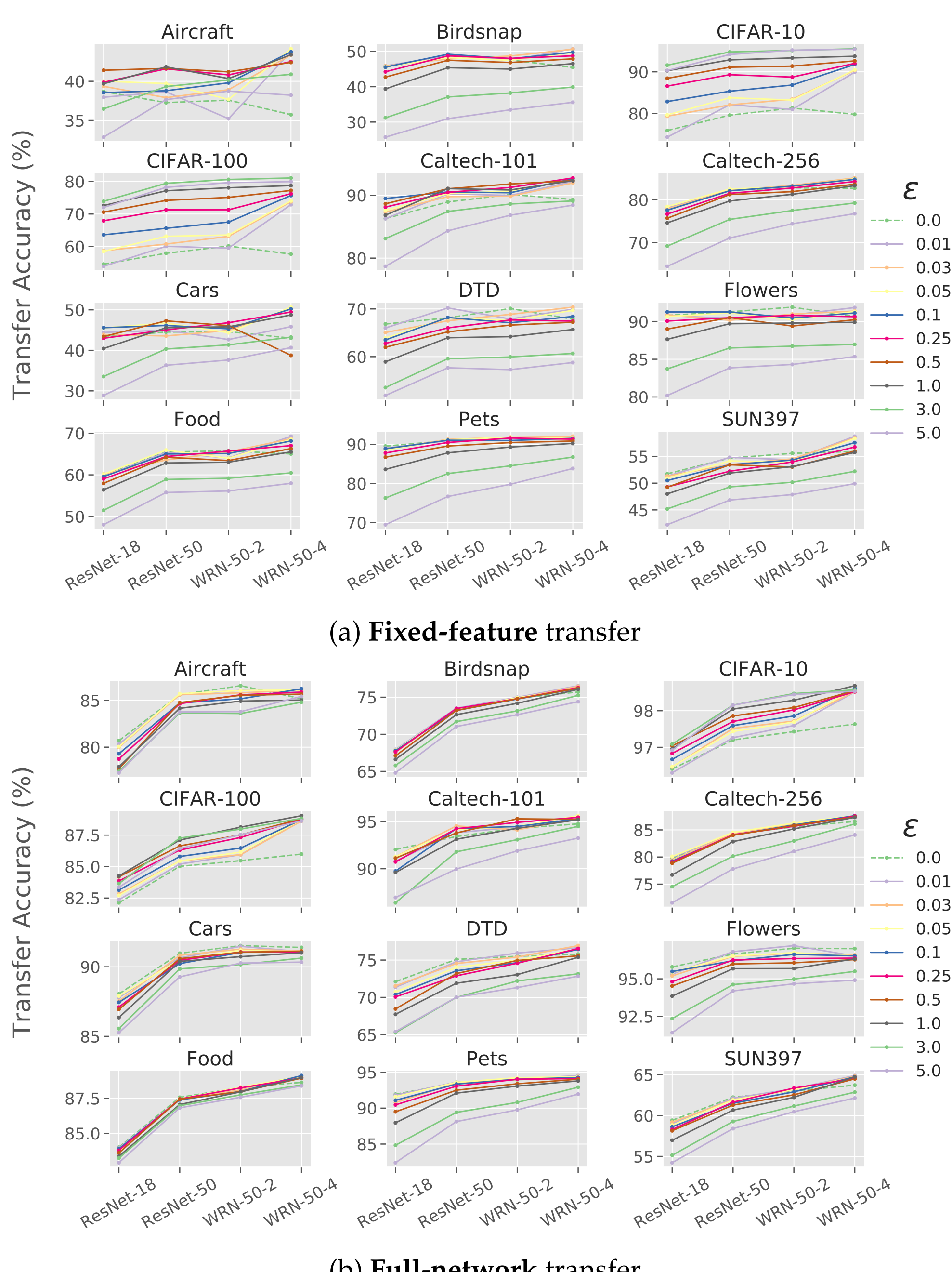
(b) **Fixed-feature** ResNet-50

(c) **Full-network** ResNet-18

(d) **Full-network** ResNet-50

Figure B.5: We compare standard, stylized and robust ImageNet models on standard transfer tasks.

B.5.5 Effect of width: additional results to Figure 2.6



(a) **Fixed-feature** transfer

(b) **Full-network** transfer

Figure B.6: Varying width and model robustness while transfer learning from ImageNet to various datasets. Generally, as width increases, transfer learning accuracies of standard models generally plateau or level off while those of robust models steadily increase.

B.6 Detailed Numerical Results

B.6.1 Fixed-feature transfer to classification tasks (Fig. 2.5)

Table B.6: **Fixed-feature** transfer for various standard and robust ImageNet models and datasets.

		Transfer Accuracy (%)									
		Robustness parameter ε									
Dataset	Model	0.00	0.01	0.03	0.05	0.10	0.25	0.50	1.00	3.00	5.00
Aircraft	**ResNet-18**	38.69	37.96	39.35	40.00	38.55	39.87	**41.40**	39.68	36.47	32.87
	ResNet-50	37.27	38.65	37.91	39.71	38.79	41.58	41.64	**41.83**	39.32	37.65
	WRN-50-2	37.59	35.22	38.92	37.68	39.80	40.81	**41.20**	40.34	40.16	38.74
	WRN-50-4	35.74	43.76	43.34	**44.14**	43.75	42.51	42.40	43.38	40.88	38.23
Birdsnap	**ResNet-18**	45.54	**45.88**	45.86	45.66	45.55	44.23	42.72	39.38	31.19	25.73
	ResNet-50	48.35	48.86	47.84	48.24	**49.19**	48.73	47.48	45.38	37.10	30.95
	WRN-50-2	47.54	47.47	**48.68**	47.48	47.93	48.01	46.84	44.99	38.23	33.47
	WRN-50-4	45.45	**50.72**	50.60	49.66	49.73	48.73	47.88	46.53	39.91	35.58
CIFAR-10	**ResNet-18**	75.91	74.33	79.35	79.67	82.87	86.58	88.45	90.27	**91.59**	90.31
	ResNet-50	79.61	82.12	82.07	83.78	85.35	89.31	91.10	92.86	**94.77**	94.16
	WRN-50-2	81.31	80.98	83.43	83.23	86.83	88.73	91.37	93.34	95.12	**95.19**
	WRN-50-4	79.81	89.90	90.35	90.48	91.76	92.03	92.62	93.73	**95.53**	95.43
CIFAR-100	**ResNet-18**	54.58	53.92	58.70	58.51	63.60	67.91	70.58	72.60	**73.91**	72.01
	ResNet-50	57.94	60.06	60.76	63.13	65.61	71.29	74.18	77.14	**79.43**	78.20
	WRN-50-2	60.14	59.52	63.12	63.55	67.51	71.30	75.11	78.07	**80.61**	79.64
	WRN-50-4	57.68	72.88	73.79	74.06	75.68	76.25	77.23	78.73	**81.08**	79.94
Caltech-101	**ResNet-18**	86.30	86.28	87.32	87.59	**89.49**	88.12	88.65	86.84	83.11	78.69
	ResNet-50	88.95	90.22	89.79	90.26	90.54	90.48	91.04	**91.07**	87.43	84.35
	WRN-50-2	90.12	89.97	89.85	90.67	90.40	91.25	**91.80**	90.84	88.62	86.83
	WRN-50-4	89.34	92.20	91.96	92.44	92.63	**92.76**	92.32	92.32	89.10	88.43
Caltech-256	**ResNet-18**	77.58	78.09	77.87	**78.40**	77.57	76.66	75.69	74.61	69.19	64.46
	ResNet-50	82.21	82.31	82.23	**82.51**	82.10	81.50	81.21	79.72	75.42	71.07
	WRN-50-2	82.78	82.94	**83.34**	83.04	83.17	82.74	81.89	81.26	77.48	74.38
	WRN-50-4	82.68	85.07	**85.08**	84.88	84.75	84.24	83.62	83.27	79.24	76.75
Cars	**ResNet-18**	43.34	44.43	43.92	45.53	**45.59**	43.00	43.40	40.45	33.55	28.86
	ResNet-50	44.52	44.98	43.56	46.74	46.15	45.04	**47.28**	45.58	40.34	36.32
	WRN-50-2	44.63	42.67	44.92	44.36	45.32	**46.83**	46.10	45.81	41.35	37.62
	WRN-50-4	43.01	45.86	50.39	**50.67**	50.22	49.46	38.77	48.73	43.26	40.68
DTD	**ResNet-18**	**66.84**	66.01	65.07	63.90	63.51	62.78	61.99	58.94	53.55	51.88
	ResNet-50	68.14	**70.21**	67.52	68.16	68.21	66.03	65.21	63.97	59.59	57.68
	WRN-50-2	**70.09**	67.89	68.87	67.55	67.11	67.70	66.61	64.20	59.95	57.29
	WRN-50-4	67.85	69.95	**70.37**	69.70	68.42	67.45	67.22	65.69	60.67	58.78
Flowers	**ResNet-18**	90.80	90.76	90.88	90.65	**91.26**	90.05	88.99	87.64	83.72	80.20
	ResNet-50	**91.28**	90.43	90.16	91.12	91.26	90.50	90.52	89.70	86.49	83.85
	WRN-50-2	**91.90**	90.86	90.97	90.26	90.46	90.79	89.39	89.79	86.73	84.31
	WRN-50-4	90.67	**91.84**	91.37	91.32	91.12	90.63	90.23	89.89	86.96	85.35
Food	**ResNet-18**	59.96	59.67	**60.20**	60.17	59.59	59.04	57.97	56.42	51.49	48.03
	ResNet-50	65.49	65.39	63.59	**65.95**	65.02	64.41	64.23	62.86	58.90	55.77
	WRN-50-2	**65.80**	64.06	65.50	64.00	65.14	65.73	63.44	63.05	59.19	56.13
	WRN-50-4	65.04	**69.26**	68.69	68.50	68.15	67.03	66.32	65.53	60.48	57.98
Pets	**ResNet-18**	**89.55**	89.03	88.67	88.54	88.87	87.80	86.73	83.61	76.29	69.48
	ResNet-50	90.92	90.93	**91.27**	91.16	91.05	90.48	89.57	87.84	82.54	76.69
	WRN-50-2	91.81	91.69	91.83	**91.85**	90.98	91.61	90.46	89.31	84.51	79.80
	WRN-50-4	91.83	91.82	**92.05**	91.70	91.54	91.32	90.85	90.23	86.75	83.83
SUN397	**ResNet-18**	**51.74**	51.31	51.32	50.92	50.50	49.30	49.25	47.99	45.19	42.24
	ResNet-50	54.69	**54.82**	53.48	54.15	53.45	52.23	53.43	51.88	49.30	46.84
	WRN-50-2	**55.57**	54.35	54.53	53.90	54.31	53.96	53.03	53.09	50.16	47.86
	WRN-50-4	55.92	**58.75**	58.45	58.34	57.56	56.75	55.99	55.74	52.21	49.91

B.6.2 Full-network transfer to classification tasks (Fig. 2.3)

Table B.7: **Full-network** transfer for various standard and robust ImageNet models and datasets.

		Transfer Accuracy (%)									
		Robustness parameter ε									
Dataset	**Model**	0.00	0.01	0.03	0.05	0.10	0.25	0.50	1.00	3.00	5.00
Aircraft	**ResNet-18**	**80.70**	80.32	79.99	80.06	79.30	78.74	77.69	77.90	77.41	77.26
	ResNet-50	85.62	85.62	85.61	**85.72**	84.73	84.65	84.77	84.16	83.66	83.77
	WRN-50-2	**86.57**	86.08	85.81	86.06	85.17	85.60	85.55	84.93	83.60	83.80
	WRN-50-4	85.19	85.98	86.10	86.11	**86.24**	85.88	85.67	85.04	84.81	85.43
Birdsnap	**ResNet-18**	67.71	**67.96**	67.58	67.86	67.80	67.63	67.10	66.62	65.80	64.81
	ResNet-50	73.38	**73.52**	73.39	73.33	73.22	73.48	73.21	72.65	71.71	71.05
	WRN-50-2	74.87	**74.98**	74.85	74.93	74.75	74.80	74.79	74.18	73.15	72.64
	WRN-50-4	75.71	**76.55**	76.47	76.14	76.18	76.29	76.20	76.06	75.25	74.40
CIFAR-10	**ResNet-18**	96.41	96.30	96.46	96.47	96.67	96.83	97.04	96.96	**97.09**	96.92
	ResNet-50	97.20	97.26	97.52	97.43	97.59	97.71	97.86	98.05	**98.15**	**98.15**
	WRN-50-2	97.43	97.60	97.72	97.69	97.86	98.02	98.09	98.29	**98.47**	98.44
	WRN-50-4	97.63	98.51	98.52	98.59	98.62	98.52	98.55	**98.68**	98.57	98.53
CIFAR-100	**ResNet-18**	82.13	82.36	82.82	82.71	83.14	83.85	84.19	**84.25**	83.65	83.36
	ResNet-50	85.02	85.20	85.45	85.44	85.80	86.31	86.64	87.10	**87.26**	86.43
	WRN-50-2	85.47	85.94	85.95	86.15	86.47	87.31	87.52	**88.13**	87.98	87.54
	WRN-50-4	85.99	88.70	88.61	88.72	88.72	88.75	88.80	**89.04**	88.83	88.62
Caltech-101	**ResNet-18**	**92.04**	90.81	91.28	91.29	89.75	90.73	91.12	89.60	86.39	86.95
	ResNet-50	93.42	93.82	**94.53**	94.18	94.27	94.24	93.79	93.13	91.79	89.97
	WRN-50-2	94.29	94.43	94.13	94.49	94.48	94.92	**95.29**	94.28	93.08	91.89
	WRN-50-4	94.76	95.60	95.32	**95.62**	95.30	95.45	95.23	95.19	94.49	93.25
Caltech-256	**ResNet-18**	79.80	80.00	79.45	**80.10**	79.23	79.07	78.86	76.71	74.55	71.57
	ResNet-50	84.19	84.30	84.37	**84.54**	84.04	84.12	84.02	82.85	80.15	77.81
	WRN-50-2	85.56	85.65	86.04	**86.26**	85.91	85.67	85.80	85.19	82.97	81.04
	WRN-50-4	86.56	87.53	87.54	**87.62**	**87.62**	87.54	87.38	87.31	86.09	84.08
Cars	**ResNet-18**	**88.05**	87.80	87.53	87.90	87.45	87.10	86.94	86.35	85.56	85.26
	ResNet-50	**90.97**	90.65	90.83	90.52	90.23	90.47	90.59	90.39	89.85	89.28
	WRN-50-2	**91.52**	91.47	91.27	91.20	91.04	91.06	91.05	90.73	90.16	90.27
	WRN-50-4	**91.39**	91.09	91.14	91.05	91.10	91.03	91.12	91.01	90.63	90.34
DTD	**ResNet-18**	**72.11**	71.37	71.54	70.73	70.37	70.07	68.46	67.73	65.27	65.41
	ResNet-50	**75.09**	74.77	74.54	74.02	73.56	72.89	73.19	71.90	70.00	70.02
	WRN-50-2	75.51	**75.94**	75.41	74.98	74.65	74.57	74.95	73.05	72.20	71.31
	WRN-50-4	75.80	76.65	**76.93**	76.47	76.44	76.54	75.57	75.37	73.16	72.84
Flowers	**ResNet-18**	**95.79**	95.31	95.20	95.44	95.49	94.82	94.53	93.86	92.36	91.42
	ResNet-50	96.65	**96.81**	96.50	96.53	96.20	96.25	95.99	95.68	94.62	94.20
	WRN-50-2	97.04	**97.21**	96.71	96.74	96.63	96.35	96.07	95.69	94.98	94.67
	WRN-50-4	**97.01**	96.52	96.59	96.53	96.53	96.38	96.28	96.33	95.50	94.92
Food	**ResNet-18**	**84.01**	83.95	83.74	83.69	83.89	83.78	83.60	83.36	83.23	82.91
	ResNet-50	**87.57**	87.42	87.45	87.46	87.40	87.45	87.44	87.06	86.97	86.82
	WRN-50-2	88.27	88.26	88.10	**88.30**	87.99	88.25	87.97	87.96	87.75	87.58
	WRN-50-4	88.64	89.09	89.00	89.08	**89.12**	88.95	88.94	88.98	88.46	88.39
Pets	**ResNet-18**	**91.94**	91.81	90.79	91.59	91.09	90.46	89.49	87.96	84.83	82.41
	ResNet-50	93.49	**93.61**	93.50	93.59	93.34	93.06	92.50	92.09	89.41	88.13
	WRN-50-2	93.96	94.05	93.98	**94.23**	94.02	94.02	93.39	93.07	90.80	89.76
	WRN-50-4	94.20	**94.53**	94.40	94.38	94.27	94.11	94.02	93.79	92.91	91.94
SUN397	**ResNet-18**	**59.41**	58.98	59.19	58.83	58.61	58.29	58.14	56.97	55.14	54.23
	ResNet-50	**62.24**	62.12	61.93	61.89	61.50	61.64	61.28	60.66	59.27	58.40
	WRN-50-2	63.02	63.28	63.16	63.18	62.90	**63.36**	62.53	62.23	61.16	60.47
	WRN-50-4	63.72	**64.89**	64.81	64.71	64.74	64.53	64.49	64.74	62.86	62.14

B.6.3 Unifying dataset scale

Fixed-feature (cf. Fig. 2.7)

Table B.8: **Fixed-feature** transfer on 32x32 downsampled datasets.

		Transfer Accuracy (%)									
		Robustness parameter ε									
Dataset	**Model**	0.00	0.01	0.03	0.05	0.10	0.25	0.50	1.00	3.00	5.00
Aircraft	**ResNet-18**	17.64	18.72	19.11	20.34	21.69	23.19	24.93	25.44	**27.15**	26.01
	ResNet-50	15.87	17.04	17.82	18.48	20.19	22.44	24.12	25.89	**28.59**	28.35
Birdsnap	**ResNet-18**	14.76	14.04	15.80	16.23	17.77	18.60	19.75	**20.16**	19.15	16.72
	ResNet-50	13.85	14.12	14.67	15.42	16.94	19.67	21.74	**23.08**	22.98	20.70
CIFAR-10	**ResNet-18**	76.02	74.36	79.48	79.71	82.97	86.62	88.47	90.29	**91.64**	90.36
	ResNet-50	79.63	82.18	82.15	83.88	85.41	89.35	91.13	92.89	**94.81**	94.23
CIFAR-100	**ResNet-18**	54.61	54.03	58.77	58.74	63.64	68.10	70.66	72.74	**74.01**	72.08
	ResNet-50	58.01	60.17	60.87	63.24	65.73	71.32	74.19	77.17	**79.50**	78.27
Caltech-101	**ResNet-18**	52.88	54.20	62.56	60.43	65.31	69.39	69.08	72.11	**73.02**	70.04
	ResNet-50	56.55	59.32	60.45	61.08	63.76	69.80	73.11	76.89	**78.86**	77.43
Caltech-256	**ResNet-18**	40.60	40.83	45.02	45.88	49.96	51.08	51.36	**54.13**	53.79	51.87
	ResNet-50	42.73	45.11	45.65	47.52	49.61	53.63	56.12	58.93	**59.79**	58.67
Cars	**ResNet-18**	13.88	14.18	16.14	16.95	19.61	20.20	20.33	**21.70**	20.89	18.75
	ResNet-50	13.16	13.89	13.68	16.84	17.07	19.40	21.88	23.19	**24.19**	23.37
DTD	**ResNet-18**	35.96	36.33	**40.27**	37.87	39.79	39.31	39.73	40.05	39.10	39.41
	ResNet-50	41.28	40.37	41.06	42.13	41.22	43.56	44.10	43.78	43.83	**44.26**
Flowers	**ResNet-18**	64.81	65.75	70.01	70.57	72.71	74.46	74.19	**76.06**	74.23	71.52
	ResNet-50	66.65	68.49	68.24	71.03	73.12	75.83	76.52	77.23	**78.31**	75.71
Food	**ResNet-18**	31.58	32.98	35.98	36.42	38.46	39.35	39.56	**41.22**	40.17	38.35
	ResNet-50	36.46	36.82	36.37	39.85	40.91	43.08	44.88	46.16	**46.45**	44.44
Pets	**ResNet-18**	48.74	46.98	56.87	56.25	61.92	62.45	63.39	**66.20**	62.23	57.15
	ResNet-50	53.98	54.10	58.55	53.57	59.58	67.35	69.31	**70.16**	69.43	64.37
SUN397	**ResNet-18**	23.16	24.35	25.34	25.94	27.60	28.00	28.12	30.19	**30.91**	30.41
	ResNet-50	23.62	25.60	24.64	27.30	27.56	29.24	31.36	32.37	**33.90**	33.58

Full-network (cf. Fig. B.4)

Table B.9: **Full-network** transfer on 32x32 downsampled datasets.

		Transfer Accuracy (%)									
		Robustness parameter ε									
Dataset	**Model**	0.00	0.01	0.03	0.05	0.10	0.25	0.50	1.00	3.00	5.00
Aircraft	**ResNet-18**	58.24	58.27	59.29	58.96	60.28	60.22	59.83	60.88	**61.78**	60.88
	ResNet-50	65.77	65.20	65.62	66.22	65.68	67.12	66.49	66.04	**68.02**	67.12
Birdsnap	**ResNet-18**	46.32	46.65	45.94	46.55	46.26	46.57	46.26	**46.80**	45.23	44.76
	ResNet-50	52.28	51.98	51.77	52.11	52.20	52.42	**52.58**	51.77	51.72	51.29
CIFAR-10	**ResNet-18**	96.50	96.38	96.51	96.62	96.78	96.86	97.12	97.04	**97.14**	97.05
	ResNet-50	97.30	97.32	97.54	97.56	97.62	97.79	97.98	98.10	**98.27**	98.16
CIFAR-100	**ResNet-18**	82.36	82.57	82.89	82.92	83.31	83.90	84.30	**84.41**	83.77	83.47
	ResNet-50	85.15	85.37	85.64	85.68	85.92	86.45	86.81	87.32	**87.45**	86.60
Caltech-101	**ResNet-18**	79.33	78.64	78.95	79.94	79.70	81.13	81.55	**83.13**	82.30	79.80
	ResNet-50	82.18	83.05	84.50	84.72	84.74	85.62	86.12	**86.61**	85.88	85.20
Caltech-256	**ResNet-18**	63.32	64.45	64.02	64.55	65.18	66.00	**66.52**	65.41	64.35	63.03
	ResNet-50	68.02	68.09	68.63	69.42	68.96	70.10	70.60	**70.66**	69.90	68.94
Cars	**ResNet-18**	68.83	68.55	68.62	68.98	69.53	69.28	**69.68**	69.27	67.99	67.42
	ResNet-50	74.84	74.95	74.13	75.23	74.61	75.29	**75.92**	75.51	75.19	74.65
DTD	**ResNet-18**	49.57	48.40	50.43	48.88	49.20	50.27	50.00	**50.74**	50.32	**50.74**
	ResNet-50	50.69	52.50	51.01	51.60	51.65	52.66	54.15	52.71	54.26	**55.53**
Flowers	**ResNet-18**	85.96	86.05	86.02	86.03	86.40	86.25	**86.41**	86.03	85.33	84.60
	ResNet-50	88.75	88.30	88.57	88.27	**88.81**	88.69	88.70	88.37	88.67	87.83
Food	**ResNet-18**	71.77	71.83	71.73	71.64	71.60	71.64	**72.10**	71.63	71.78	71.37
	ResNet-50	75.83	75.19	75.52	75.51	75.50	75.37	**76.11**	75.91	75.76	75.61
Pets	**ResNet-18**	76.32	77.35	77.71	78.05	78.63	78.70	**78.75**	77.82	75.72	72.21
	ResNet-50	82.34	81.95	82.64	82.24	82.52	83.59	83.57	**83.72**	81.87	79.33
SUN397	**ResNet-18**	42.81	42.65	43.40	43.35	44.01	44.20	44.51	**44.61**	44.31	43.54
	ResNet-50	44.64	44.95	44.73	45.09	45.44	45.93	46.74	47.24	**47.47**	47.15

Appendix C

Additional details for Chapter 3

C.1 3D Simulation Details

C.1.1 Overview of AirSim

We conduct our simulation experiments using the high fidelity simulator, Microsoft AirSim. AirSim acts as a plugin to Unreal Engine, which is a AAA videogame engine providing access to high fidelity graphics features such as high resolution textures, realistic lighting, soft shadows etc. making it a good choice for rendering for computer vision applications. AirSim internally provides physics models for a quadrotor vehicle, which we leverage for performing autonomous drone landing. As a plugin, AirSim can be paired with any Unreal Engine environmnent to simulate autonomous vehicles that can be programmed with an API both in terms of planning/control as well as obtaining camera images. AirSim also allows for controlling environmental features such as time of day, dynamically adding/removing objects, changing object textures and so on.

C.1.2 3D Boosters Classification Experiment

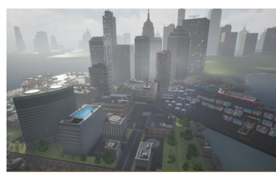
Format of 3D models To evaluate the performance of pretrained ImageNet classifiers at detecting 3D unadversarial/boosted objects (e.g. the jet shown in the main chapter) among realistic settings, we set up an experiment using AirSim for image classification of common classes (warplane, car, truck, ship, etc). We pick the class of 'warplane' as our object class of interest download publicly available 3D meshes for this class from www.sketchfab.com. Using the open source 3D modeling software Mitsuba, we modify the object texture to match the boosted texture for the corresponding class, and then export these meshes into the GLTF format for ingestion into Unreal Engine/AirSim. This allows

us to import the boosted objects into the AirSim framework, and spawn them as objects in any of the environments being created.

Environment screenshots and description Within AirSim, in the interest of generating realistic imagery, we simulate a city environment (Figure C.1a). For this experiment, we use the ComputerVision mode of AirSim, which does not simulate a vehicle but rather, gives the user control of a free moving camera, allowing us to generate data at ease from various locations and varying camera and world parameters.

Sampling and evaluation Once the 3D objects (unadversarial or normal) are present in AirSim's simulated world, the next step is to evaluate the classification of these objects from different camera angles, weather conditions etc. Given the location of a candidate object (which we randomize and average over five locations), we sample a grid ($10 \times 10 \times 10$) of camera positions in 3D around the object. For each of these positions, we move AirSim's main camera and orient it towards the object, resulting in images from various viewpoints. At runtime, each of these images are immediately processed by a pretrained ResNet-18 ImageNet classifier, which reports the top 5 class predictions. We average the accuracies across the five different locations in the scene and the 1000 grid points around the object at each location.

Along with this variation in camera angles and thereby, object pose in the frame; we also evaluate the performance of of the various 3D objects given environmental perturbations. We achieve this through the AirSim's weather conditions feature, using which we simulate weather conditions such as dust and fog dynamically with varying levels of severity of these conditions. We will open-source binaries for the AirSim code and environments that we use which will allow people to replicate our results, and investigate more scenarios of interest.


(a) City environment in AirSim used for detection experiment

(b) Boosted 'jet' model in the City environment.

(c) Sample landing pads atop buildings in the City environment.

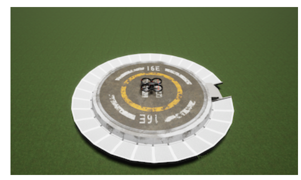

(d) Drone in test environment used for the landing experiments.

Figure C.1: Various AirSim environment we use.

C.1.3 Drone Landing Experiment

In this experiment, we evaluate how unadversarial/boosted objects can help robustify perception-action loops that are driven by vision-based pose estimation. Perception-action loops are at the heart of many robotics tasks, and accurate perception is imperative for safe, efficient navigation of robots. We choose the scenario of autonomous drone landing as our experiment, and simulate it within AirSim.

For this experiment, we create assets of landing pads that are similar to helipads on top of buildings in the city environment(Figure C.1c). We also use a test environment with a single landing pad located on a patch of grass. An example of such a landing pad can be seen in Figure C.1d. We use AirSim to simulate a quadrotor drone in these worlds, which can be programmatically controlled using a Python API. AirSim allows us to equip a downward facing, gimballed camera on this drone in order to obtain RGB images, which are then processed by our landing pad pose estimation (regression) model. Given an RGB image, the regression model outputs a 6 degree of freedom pose for the landing pad. We use/optimize only the first two enteries of this output corresponding to the relative x and y location of the landing pad w.r.t the drone.

We formulate the drone landing experiment as a visual servoing task: a perception action loop that involves estimating the relative location of the pad from the image frame captured by the downward facing camera of the drone, and sending an appropriate velocity command in order to align the camera center with that of the pad. We achieve these through the following steps:

Data Collection. We use AirSim's inbuilt data collection API for this step. Given the location of the pad in the world, we sample various feasible locations for the drone in an imaginary cone whose vertex aligns with the center of the landing pad. We then spawn the drone in these randomly sampled positions, and obtain the RGB and segmentation views of the pad as generated by AirSim, along with the relative ground truth position of the landing pad w.r.t the drone, and repeat this process to create a dataset. The collected dataset contains 20000 images and is split 80-20 between train and evaluation sets.

Landing pad pose estimator. We train a model that maps top view images of a scene with a landing pad, to the relative 2D location of the landing pad w.r.t the drone in the camera frame. We use a ResNet-18 pretrained on ImageNet as the backbone for the pose regressor, and we replace the last classification layer with a regression layer outputting the (x, y) relative location of the pad w.r.t drone. The model is trained end-to-end by

minimizing the mean squared error (MSE) loss between the predicted location and the ground truth location. The ground truth is collected along with the images using the AirSim City simulation environment as describe before.

We train the model for 10 epochs using SGD with a fixed learning rate of 0.001, a batch size of 512, a weight decay of 1e-4, and with MSE as the objective function. The model converges fairly quickly (within the first few epochs).

Drone Landing. To use the pose estimator's predictions and send appropriate actions, we utilize the Multirotor API of AirSim. This allows us to control the drone by setting the desired velocity commands along all the axes (translational/rotational). Given the position of the landing pad in the scene relative to that of the drone(as output by the pose regressor) we execute the landing operation by sending appropriate velocity commands to the drone.

To generate the right velocity commands, given the relative position of the landing pad, we use a standard PID controller that computes corrective velocity values until the position of the drone matches that of the landing pad. For a pose output by the regressor treated as the setpoint P_{set}, and current drone pose P_{curr} and at any point at time t, the appropriate velocity command $v(t)$ can simply be computed as follows:

$$v(t) = K_p e(t) + K_d \frac{d}{dt} e(t) + K_i * \int_0^t e(t) dt$$

where $e(t) = P_{set} - P_{curr}$, K_p, K_d, and K_i are the hyperparameters of the PID controller and are manually tuned. We find that $K_p = K_d = 5$ and $K_i = 0$ to be reasonable for our task.

For realistic perturbations to the scene, similar to the 3D boosters classification experiment, we continue making use of the weather API to generate weather conditions in AirSim. This results in a variation of factors such as amount of dust or fog in the scene, allowing us to evaluate the performance of landing under various realistic conditions.

C.2 Experimental Setup

C.2.1 Pretrained vision models we evaluate

Here we present details of the different vision models we use in this chapter. For more details on all of these, please check the README of our code at https://github.com/microsoft/unadversarial.

Corruption benchmark experiments: We use pretrained ResNet-18 and ResNet-50 (both standard and ℓ_2-robust with $\varepsilon = 3$) architectures from [SIE+20]: https://github.com/microsoft/robust-models-transfer.

3D object classification in AirSim: We use an ImageNet pretrained ResNet-18 architecture from the PyTorch's Torchvision[1] to classify all the boosted and non-boosted versions of the jets, cars, ships etc in AirSim.

Drone landing experiment in AirSim: We finetune an ImageNet pretrained ResNet-18 model on the regression task of drone landing. The last layer of the pretrained model is replaced with a 2D linear layer estimating the relative pad location w.r.t the drone. We collect a 20k sample dataset for training the pad pose estimation in AirSim with an $80 - 20$ train-val spilt. We use a learning rate of 0.001, a batch size of 512, a weight decay of $1e - 4$. We train for 10 epochs.

Physical world unadversarial examples experiment: Similar to the 3D object classification experiment in AirSim, we use an ImageNet pretrained ResNet-18 architecture from Torchvision.

C.2.2 Unadversarial patch/texture training details

Patches training details We fix the training procedure for all of the 2D patches we optimize in this chapter. We train all the patches starting from random initialization with batch size of 512, momentum of 0.9, and weight decay of $1e - 4$. We train all the patches for 30 epochs (which is more than enough as we observe that for both ImageNet and CIFAR-10, the patch converges within the first 10 epochs) with a learning rate of 0.1 We sweep over three learning rates $\in \{0.1, 0.01, 0.001\}$ but we find that all of these obtain very similar results. So we stick with a learning rate of 0.1 for all of our experiments..

[1]These models can be found here https://pytorch.org/docs/stable/torchvision/models.html

For the classification tasks (i.e., everything but drone landing) we use the standard cross-entropy loss. For the drone landing task (landing pad pose estimation), we use the standard mean squared error loss.

Texture training details We now outline the process for constructing adversarial textures. We implemented a custom PyTorch module with a distinct forward and backward pass; on the forward pass (i.e., during evaluation), the module takes as input an ImageNet image, and a 200px by 200px texture; using the Python bindings for Mitsuba [NVZ+19] 3D renderer, the module returns a rendering of the desired 3D object, overlaid onto the given ImageNet image. On the backwards pass (i.e., when computing gradients), we use the 3D model's UV map[2]—a linear transformation from (x, y) locations on the texture to (x, y) locations in the rendered image—to approximate gradients through the rendering process. This is the same procedure used by [AEI+18] for constructing physical adversarial examples. Note that this is a simple approximation that only accounts for the location of pixels in the rendered image (i.e., ignores the effects of lighting, warping, etc.). However,

C.2.3 Details of the physical world experiment

To conduct the physical-world experiments, we used a toy racecar[3], a toy warplane[4] (both from amazon.com) as well as two household objects: a coffeepot and eggnog container. We then printed the unadversarial patches corresponding to classes "racer," "warplane," "coffeepot," and "eggnog" on an HP DeskJet 2700 InkJet printer, at 250% scale. We adhere the patches to the top of their respective objects with clear tape (the results are shown in Figure 3.9b). We choose 18 distinct poses (camera positions), and for each pose took one picture of the object with the patch attached, and one picture without (keeping the location of the patch constant throughout the experiment). Example photographs are shown in Figure C.2. We evaluated a pre-trained ResNet-18 classifier on the resulting images.

C.2.4 Datasets

We use two datasets across all the chapter:

1. CIFAR [Kri09] https://paperswithcode.com/dataset/cifar-10.

[2]Mitsuba provides direct access to the UV map through the aov integrator; see the code release for more details.

[3]https://www.amazon.com/gp/product/B07T5X69TZ/

[4]https://www.amazon.com/CORPER-TOYS-Pull-Back-Aircraft-Birthday/dp/B07DB3839X/

Figure C.2: Photographs in different poses of the four physical objects we experimented on, with and without an unadversarial patch.

2. ImageNet [RDS+15], with a custom (research, non-commercial) license, as found here https://paperswithcode.com/dataset/imagenet.

C.2.5 Compute

We use an internal cluster containing NVIDIA 1080-TI, 2080-TI and P100 GPUs. Each experiment required no more than 1 GPU at a time.

C.2.6 Replicate our results

We desired simplicity and kept reproducibility in our minds when conducting our experiments, so we use standard hyperparameters and minimize the number of tricks needed to replicate our results. Our code is available at https://github.com/microsoft/unadversarial.

C.3 Omitted Results

In the below figure, we show a more detailed look of the main results of the benchmarking experiments in this chapter, along with useful baselines. The single color plots (e.g. the left subplot in Figure C.3) report the average performance over the 5 severities of ImageNet-C and CIFAR-10-C. The multicolor bar plots (e.g. the right subplot in Figure C.3) report the detail performance per severity level. The horizontal dashed lines report the performance of the pretrained models on the original (non-patched) ImageNet-C and CIFAR-10-C datasets and serve as a baseline to compare with. For both ImageNet and CIFAR as shown in Figure C.4 and Figure C.3, we are able to train unadversarial patches of various size that, once overlaid on the datasets, make the pretrained model consistently much more robust under all corruptions.

C.3.1 Corruption benchmark main results: additional results to Figure 3.3b

Here we show the detailed main results for boosting ImageNet and CIFAR-10 with unadverasarial patches.

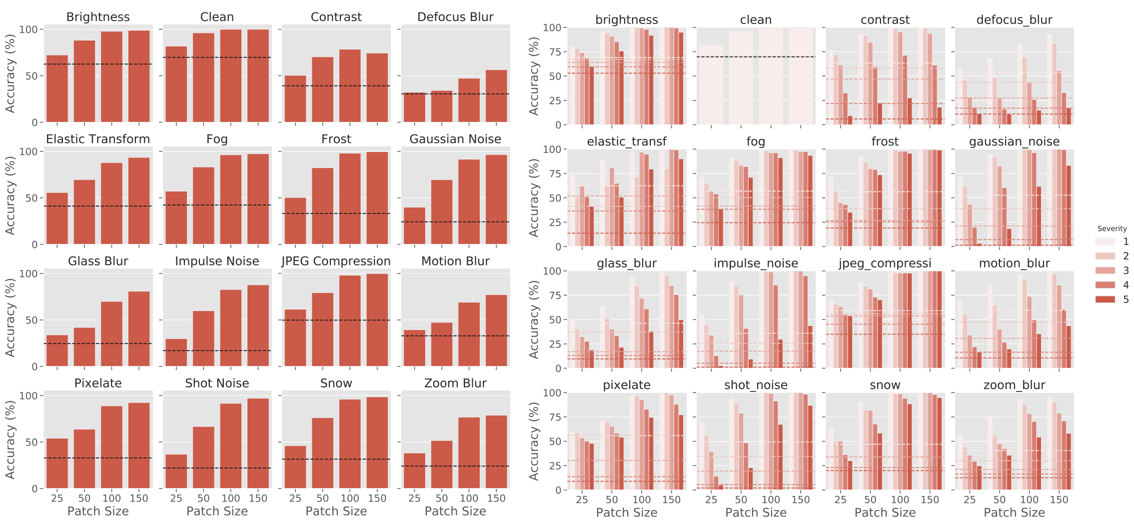


Figure C.3: Robustness of a trained 2D booster over pretrained ImageNet ResNet-18 model.

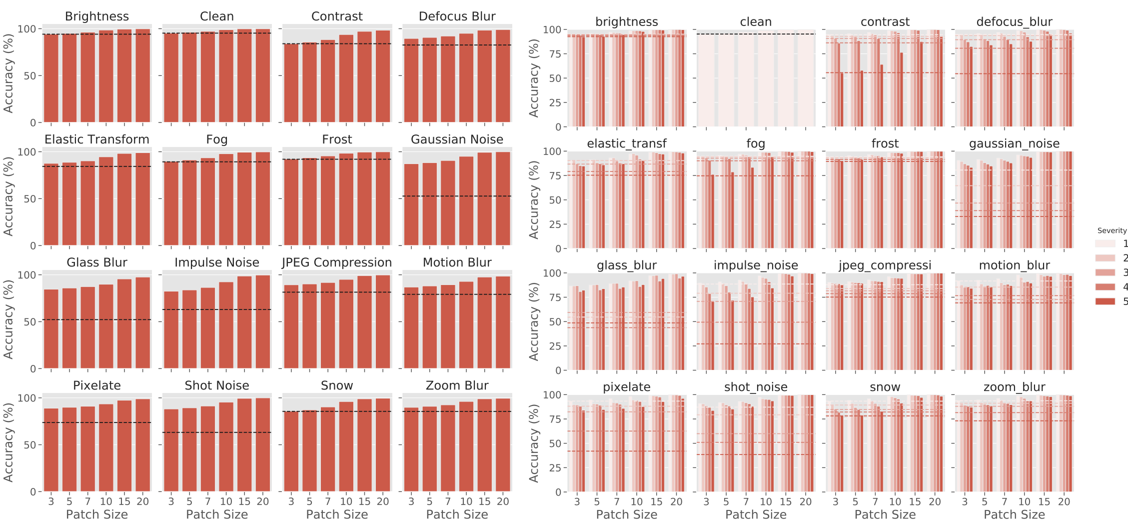


Figure C.4: Robustness of a trained 2D booster over pretrained CIFAR-10 ResNet-50 model.

C.3.2 Baselines

Below, we report a number of alternative ways to create patches for boosting the performance of object recognition.

QR-Code

We compare our unadversarial patches to the well-known QR-Code patches. We create a QR-Code for each class of the ImageNet dataset using Python's `qrcode` package(we avoid using CIFAR-10 since the images are too small for QR-Codes to be visible and detected at all). We overlay the QR-Codes over the ImageNet validation set according in accordance to what label each image has. We add the various ImageNet-C corruption on top of the resulting images, then we use python's `Pyzbar`[5] package to detect the QR-Codes. The results are shown in Figure C.5. The performance of QR-Codes is not comparable to what we obtain with unadversarial patches (see Figure 3.3b).

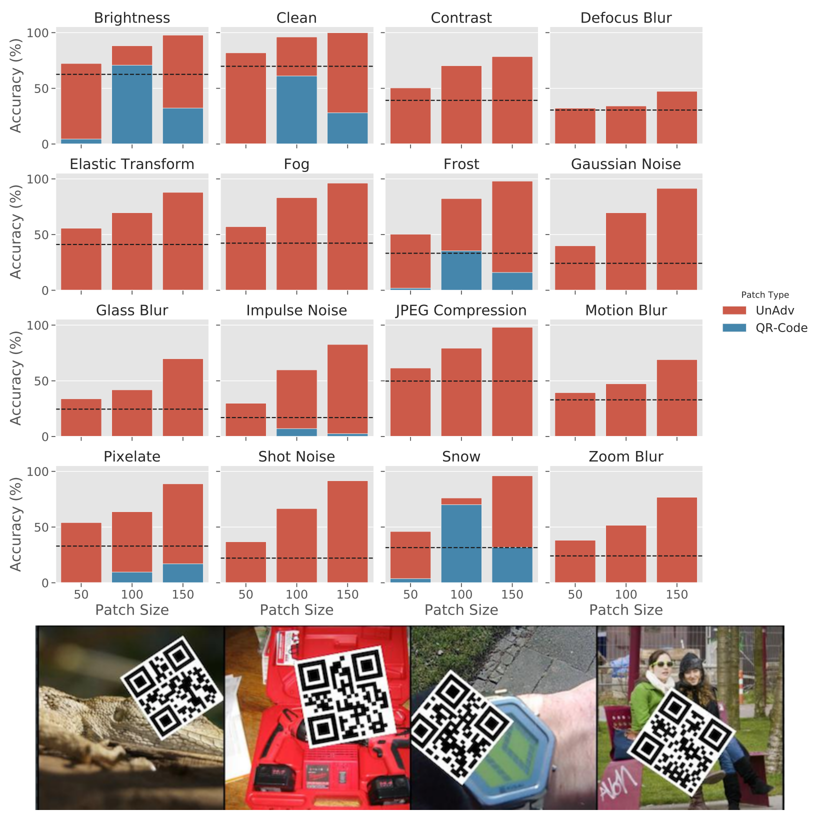


Figure C.5: QR-Code boosted ImageNet results under various corruptions.

[5] We experiment with `OpenCV for detecting the QR-Codes but find that Pyzbar leads to better performance.`

Best training image per class as patch

Another natural baseline that we compare with is using the best images per class in the training set of the task of interest as patches for boosting the performance of pretrained models. For example, for ImageNet classification, we simply evaluate the loss of each training image using a pretrained ImageNet model (ResNet-18 in our case), and we the image with the lowest loss per class as the patch for that class. Now we overlay these found patches onto the ImageNet validation set with random scaling, rotation, and translation (as shown in Figure C.6), we add ImageNet-C corruptions, and we evaluate this new dataset using the same pretrained model we used to extract the patches. The results for ImageNet are shown in Figure C.6.

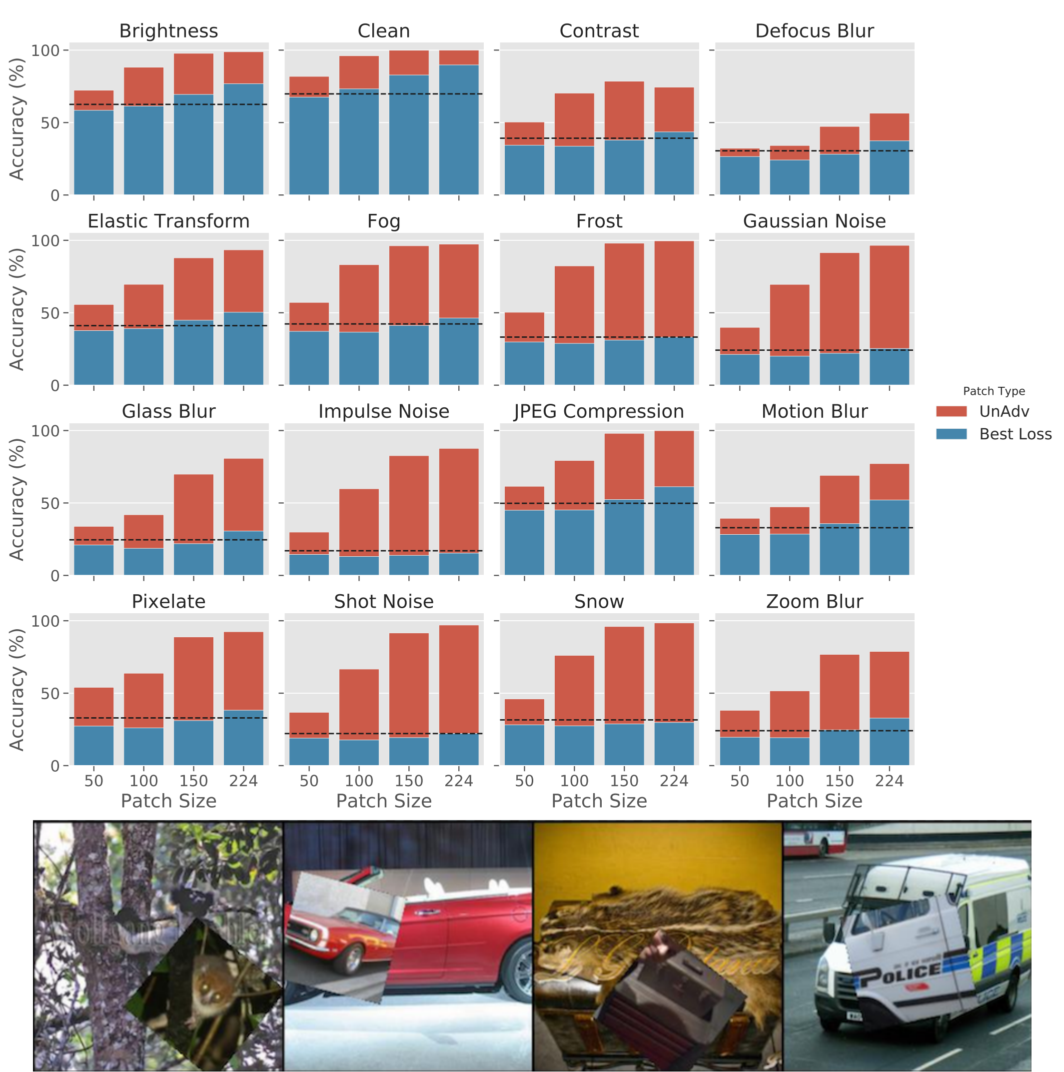


Figure C.6: Best training image with translation, rotation, and scaling for ImageNet.

Best training image vs random training image as patch

Here we investigate whether using a random image from the training set does any better than using the best-loss image as a patch. The results are shown in the below Figures. As one would expect, using a random image from the training set leads to strictly worse performance. Results on ImageNet are shown in Figure C.7.

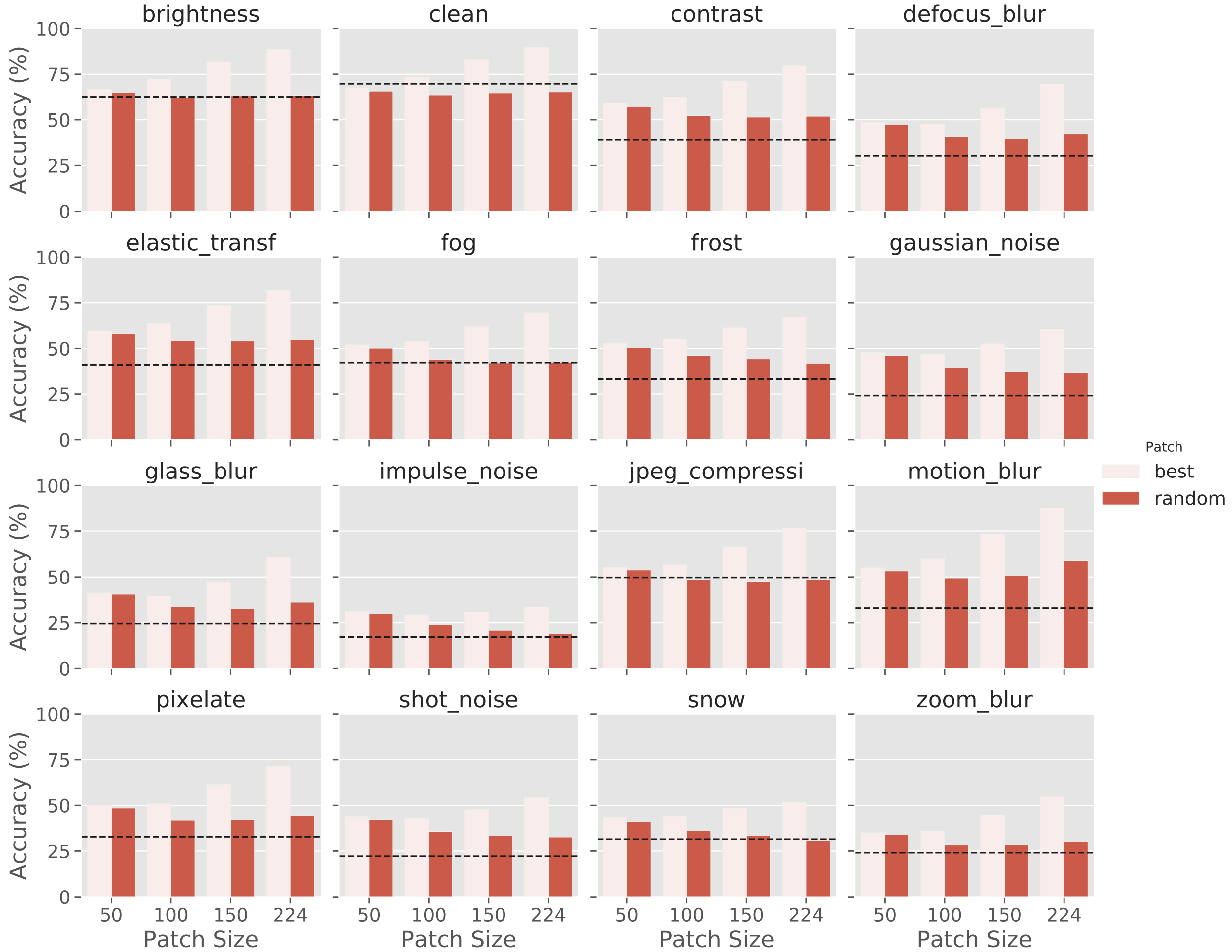


Figure C.7: Best training image vs random training image with translation, rotation, and scaling.

Predefined fixed-pattern unadversarial patches

This baselines is slightly different than the previous baselines since it allows the underlying classification model to be changed. Basically, we fix the set of patches to a predefined pattern (here a fixed random gaussian noise for each class), and we train a classifier on an undversarial/boosted dataset with these patches. The resulting models are consistently weaker on all corruptions of ImageNet-C as shown in Figure C.8 compared to our trained patches in Figure 3.3b.

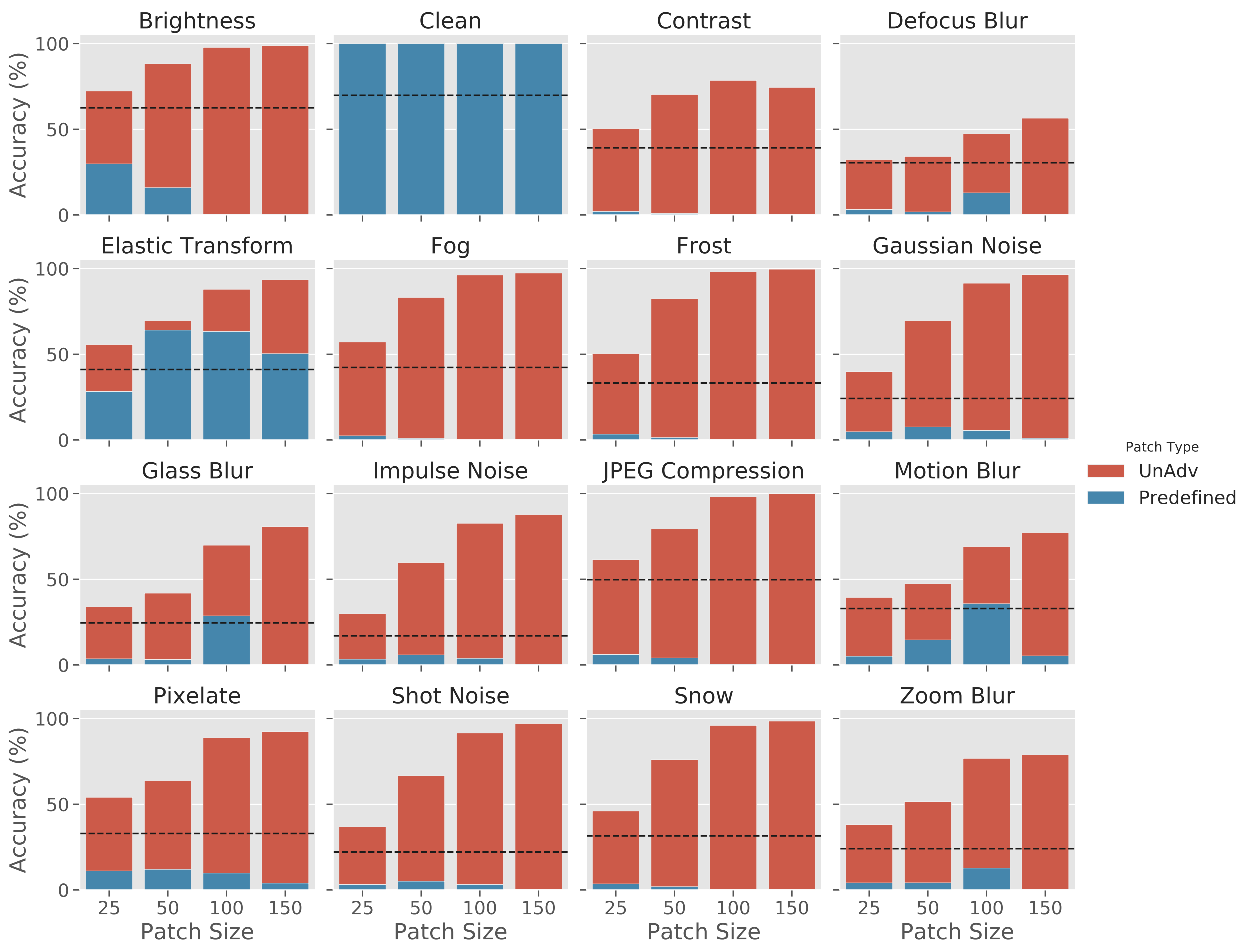


Figure C.8: Robustness of an ImageNet ResNet-18 model trained on a predefined patch.

Appendix D

Additional details for Chapter 4

D.1 Experimental Setup

D.1.1 Details of the diffusion model we used

In this chapter, we used the open-source stable diffusion model hosted on the Hugging Face[1]. We use the hyperparameters presented in Table D.1 to generated images from this model. For a given image on which we want to test our immunization method, we first search for a good random seed that leads to a realistic modification of the image given some textual prompt. Then we use the same seed when editing the immunized version of the same image using the diffusion model. This ensures that the immunized image is modified in the same way as the original image, and that the resulting non-realistic edits are due to immunization and not to random seed.

Table D.1: Hyperparameters used for the Stable Diffusion model.

height	width	guidance_scale	num_inference_steps	eta
512	512	7.5	100	1

D.1.2 Our attacks details

Throughout the chapter, we use two different attacks: an encoder attack and a diffusion attack. These attacks are described in this chapter, and are summarized here in Algorithm 4 and Algorithm 5, respectively. For both of the attacks, we use the same set of hyperparam-

[1]This model is available on: `https://huggingface.co/runwayml/stable-diffusion-v1-5`.

eters shown in Table D.2. The choice of ϵ was such that it is the large enough to disturb the image, but small enough to not be noticeable by the human eye.

Table D.2: Hyperparameters used for the adversarial attacks.

Norm	ϵ	step size	number of steps
ℓ_∞	16/255	2/255	200

Algorithm 4 Encoder Attack on a Stable Diffusion Model

1: **Input:** Input image $\mathbf{x}$, target image $\mathbf{x}_{targ}$, Stable Diffusion model encoder $\mathcal{E}$, perturbation budget ϵ, step size k, number of steps N.
2: Compute the embedding of the target image: $\mathbf{z}_{targ} \leftarrow \mathcal{E}(\mathbf{x}_{targ})$
3: Initialize adversarial perturbation $\delta_{encoder} \leftarrow 0$, and immunized image $\mathbf{x}_{im} \leftarrow \mathbf{x}$
4: **for** $n = 1 \dots N$ **do**
5: Compute the embedding of the immunized image: $\mathbf{z} \leftarrow \mathcal{E}(\mathbf{x}_{im})$
6: Compute mean squared error: $l \leftarrow \|\mathbf{z}_{targ} - \mathbf{z}\|_2^2$
7: Update adversarial perturbation: $\delta_{encoder} \leftarrow \delta_{encoder} + k \cdot \text{sign}(\nabla_{\mathbf{x}_{im}} l)$
8: $\delta_{encoder} \leftarrow \text{clip}(\delta_{encoder}, -\epsilon, \epsilon)$
9: Update the immunized image: $\mathbf{x}_{im} \leftarrow \mathbf{x}_{im} - \delta_{encoder}$
10: **end for**
11: **Return:** $\mathbf{x}_{im}$

Algorithm 5 Diffusion Attack on a Stable Diffusion Model

1: **Input:** Input image $\mathbf{x}$, target image $\mathbf{x}_{targ}$, Stable Diffusion model f, perturbation budget ϵ, step size k, number of steps N.
2: Initialize adversarial perturbation $\delta_{diffusion} \leftarrow 0$, and immunized image $\mathbf{x}_{im} \leftarrow \mathbf{x}$
3: **for** $n = 1 \dots N$ **do**
4: Generate an image using diffusion model: $\mathbf{x}_{out} \leftarrow f(\mathbf{x}_{im})$
5: Compute mean squared error: $l \leftarrow \|\mathbf{x}_{targ} - \mathbf{x}_{out}\|_2^2$
6: Update adversarial perturbation: $\delta_{diffusion} \leftarrow \delta_{diffusion} + k \cdot \text{sign}(\nabla_{\mathbf{x}_{im}} l)$
7: $\delta_{diffusion} \leftarrow \text{clip}(\delta_{diffusion}, -\epsilon, \epsilon)$
8: Update the immunized image: $\mathbf{x}_{im} \leftarrow \mathbf{x}_{im} - \delta_{diffusion}$
9: **end for**
10: **Return:** $\mathbf{x}_{im}$

D.2 Extended Background for Diffusion Models

Overview of the diffusion process. At their heart, diffusion models leverage a statistical concept: the diffusion process [SWM+15; HJA20]. Given a sample $\mathbf{x}_0$ from a distribution of

real images $q(\cdot)$, the diffusion process works in two steps: a forward step and a backward step. During the forward step, Gaussian noise is added to the sample $\mathbf{x}_0$ over T time steps, to generate increasingly noisier versions $\mathbf{x}_1, \ldots, \mathbf{x}_T$ of the original sample $\mathbf{x}_0$, until the sample is equivalent to an isotropic Gaussian distribution. During the backward step, the goal is to reconstruct the original sample $\mathbf{x}_0$ by iteratively denoising the noised samples $\mathbf{x}_T, \ldots, \mathbf{x}_1$. The power of the diffusion models stems from the ability to learn the backward process using neural networks. This allows to generate new samples from the distribution $q(\cdot)$ by first generating a random Gaussian sample, and then passing it through the "neural" backward step.

Forward process. During the forward step, Gaussian noise is iteratively added to the original sample $\mathbf{x}_0$. The forward process $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ is assumed to follow a Markov chain, i.e. the sample at time step t depends only on the sample at the previous time step. Furthermore, the variance added at a time step t is controlled by a schedule of variances $\{\beta_t\}_{t=1}^{T}$[2].

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}); \qquad q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{D.1}$$

Backward process. At the end of the forward step, the sample $\mathbf{x}_T$ looks as if it is sampled from an isotropic Gaussian $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$. Starting from this sample, the goal is to recover $\mathbf{x}_0$ by iteratively removing the noise using neural networks. The joint distribution $p_\theta(\mathbf{x}_{0:T})$ is referred to as the reverse process, and is also assumed to be a Markov chain.

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t); \qquad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{D.2}$$

Training a diffusion model. At its heart, diffusion models are trained in a way similar to Variational Autoencoders, i.e. by optimizing a variational lower bound. Additional tricks are employed to make the process faster. For an extensive derivation, refer to [Wen21].

$$\mathbb{E}_{q(\mathbf{x}_0)}[-\log p_\theta(\mathbf{x}_0)] \leq \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] = L_{VLB} \tag{D.3}$$

Latent Diffusion Models (LDMs). In this work, we focus on a specific class of diffusion models, namely LDMs, which was proposed in [RBL+22] as a model that applies the

[2]The values of a_t and b_t from this chapter correspond to $a_t = \sqrt{1-\beta_t}$ and $b_t = \beta_t$

diffusion process described above in a *latent space* instead of the image space. This enables efficient training and inference of diffusion models.

To train an LDM, the input image $\mathbf{x}_0$ is first mapped to a latent representation $\mathbf{z}_0 = \mathcal{E}(\mathbf{x}_0)$, where $\mathcal{E}$ is an image encoder. This input representation $\mathbf{z}_0$ is then passed to the diffusion process to obtain a denoised $\tilde{\mathbf{z}}$. The generated image $\tilde{\mathbf{x}}$ is then obtained by decoding $\tilde{\mathbf{z}}_0$ using a decoder $\mathcal{D}$, i.e. $\tilde{\mathbf{x}} = \mathcal{D}(\tilde{\mathbf{z}})$.

D.3 Additional Results

D.3.1 Additional quantitative results

We presented in Section 4.3 several metrics to assess the similarity between the images generated with and without immunization. Here, we report in Table D.3 additional metrics to evaluate this: SR-SIM [ZL12], GMSD [XZM+14], VSI [ZSL14], DSS [BSM+15], and HaarPSI [RBK+18]. Similarly, we indicate for each metric whether a higher value corresponds to higher similarity (using $\uparrow$), or contrariwise (using $\downarrow$). We again observe that applying the encoder attack already decreases the similarity between the generated images with and without immunization, and applying the diffusion attack further decreases the similarity.

Table D.3: Additional similarity metrics for Table 4.1. Errors denote standard deviation over 60 images.

Method	**SR-SIM** $\uparrow$	**GMSD** $\downarrow$	**VSI** $\uparrow$	**DSS** $\uparrow$	**HaarPSI** $\uparrow$
Immunization baseline (Random noise)	0.91 ± 0.04	0.20 ± 0.06	0.94 ± 0.03	0.35 ± 0.18	0.52 ± 0.15
Immunization (Encoder attack)	0.86 ± 0.05	0.26 ± 0.05	0.90 ± 0.03	0.19 ± 0.09	0.35 ± 0.11
Immunization (Diffusion attack)	$\mathbf{0.84 \pm 0.05}$	$\mathbf{0.27 \pm 0.04}$	$\mathbf{0.89 \pm 0.03}$	$\mathbf{0.17 \pm 0.08}$	$\mathbf{0.31 \pm 0.08}$

D.3.2 Generating Image Variations using Textual Prompts

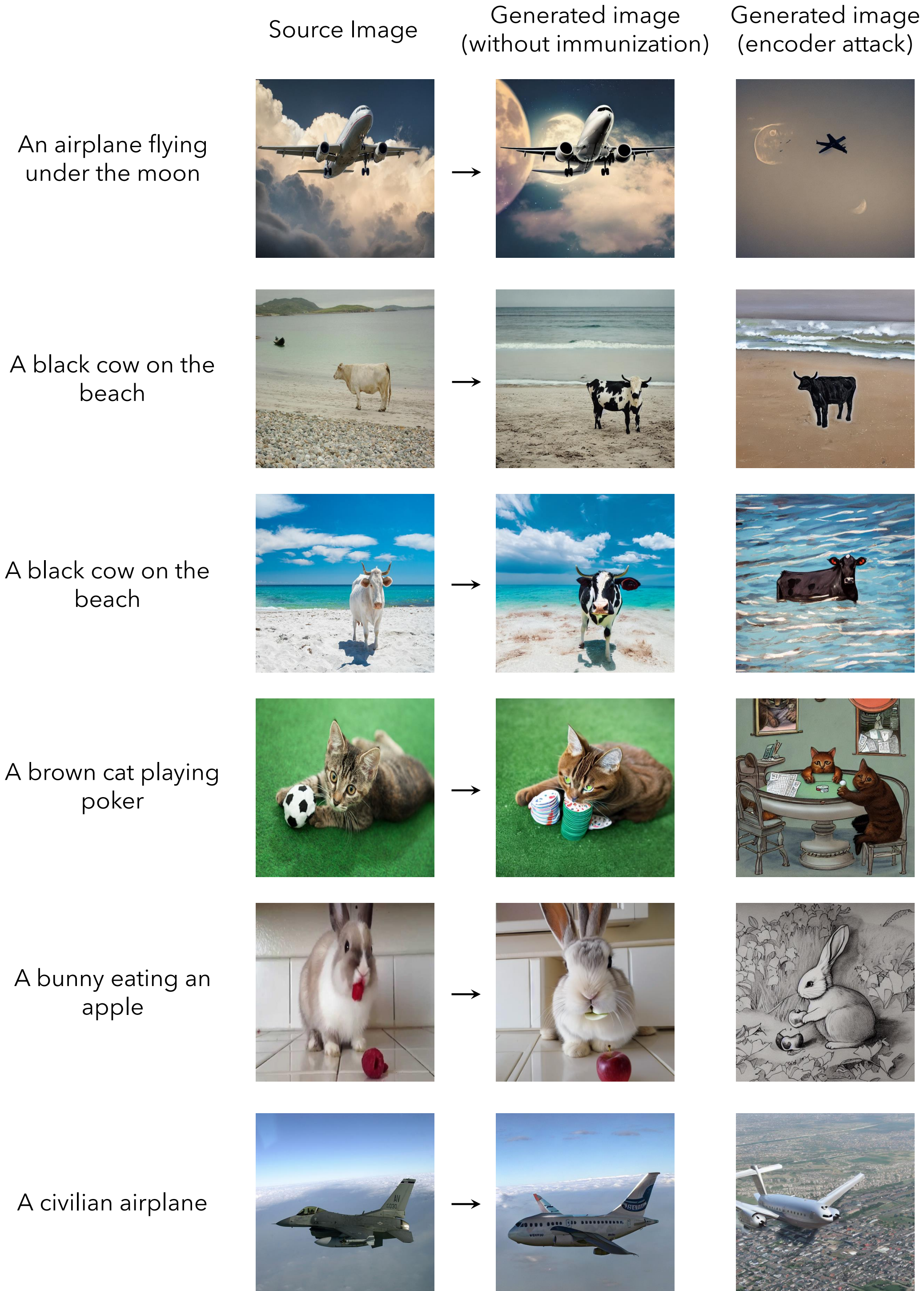


Figure D.1: Immunization against generating prompt-guided image variations.

D.3.3 Image Editing via Inpainting

Figure D.2: Immunization against image editing via prompt-guided inpainting.

	Source Image		Generated image (without immunization)	Generated image (encoder attack)	Generated image (diffusion attack)
A man in a farm		→			
A man in a restaurant		→			
A man in a restaurant		→			
A man in a store		→			
A man preparing dinner		→			
A man holding a microphone		→			

	Source Image		Generated image (without immunization)	Generated image (encoder attack)	Generated image (diffusion attack)
A man holding a phone		→			
A man preparing dinner		→			
A man drinking hot coffee		→			
A man playing poker		→			
A man playing poker		→			
A man playing poker		→			

	Source Image		Generated image (without immunization)	Generated image (encoder attack)	Generated image (diffusion attack)
A man sitting in a metro		→			
A man sitting in first class airplane		→			
A man sitting in the airport		→			
A man dancing on stage		→			
A man in a meeting		→			
A man riding a motorcycle		→			

	Source Image	Generated image (without immunization)	Generated image (encoder attack)	Generated image (diffusion attack)
Two men ballroom dancing	→			
Two men cooking in the kitchen	→			
Two men cooking in the kitchen	→			
Two men cooking in the kitchen	→			
Two men grilling	→			
Two men grilling	→			

Source Image

Generated image (without immunization)

Generated image (encoder attack)

Generated image (diffusion attack)

Two men in a hot tub →

Two men in a wedding →

Two men in a wedding on a seafront →

Two men in jail →

Two men in the forest →

Two men on the grass →

	Source Image	Generated image (without immunization)	Generated image (encoder attack)	Generated image (diffusion attack)
Two men in the zoo				
Two men playing guitar				
Two men sneaking into a building				
Two men street fighting				

Source Image

Generated image (without immunization)

Generated image (encoder attack)

Generated image (diffusion attack)

A man receiving an award →

Two men attending a wedding →

Two men attending a wedding →

Two men in an airplane →

Two men in a restaurant →

Two men in a restaurant →

	Source Image		Generated image (without immunization)	Generated image (encoder attack)	Generated image (diffusion attack)
Two men in a restaurant		→			
Two men in Europe		→			
Two men in an airplane		→			
Two men in front of the Eiffel Tower		→			
Two men riding a motorcycle		→			
Two men wearing gray shirts in the fog		→			

Appendix E

Additional details for Chapter 5

E.1 Experimental details.

E.1.1 Models and architectures

We use two sizes of vision transformers: ViT-Tiny (ViT-T) and ViT-Small (ViT-S) [Wig19; DBK+21]. We compare to residual networks of similar size: ResNet-18 and ResNet-50 [HZR+16], respectively. These architectures and their corresponding number of parameters are summarized in Table E.1.

Table E.1: A collection of neural network architectures we use in this chapter.

Architecture	ViT-T	ResNet-18	ViT-S	ResNet-50
Params	5M	12M	22M	26M

E.1.2 Training Details

We train our models on ImageNet [RDS+15], with a custom (research, non-commercial) license, as found here `https://paperswithcode.com/dataset/imagenet`. For all experiments in this chapter, we consider 10,000 image subsets of the original ImageNet validation set (we take every 5th image).

1. For ResNets, we train using SGD with batch size of 512, momentum of 0.9, and weight decay of 1e-4. We train for 90 epochs with an initial learning rate of 0.1 that drops by a factor of 10 every 30 epochs.

2. For ViTs, we use the same training scheme as used in Wightman [Wig19].

Note that we use the same (basic) data-augmentation techniques for both ResNets and ViTs. Specifically, we only use random resized crop and random horizontal flip (no RandAug, CutMix, MixUp, etc.).

We attach all our model weights to the submission.

Models trained with missingness augmentations. In Sections 5.2 and 5.3, we also consider models that were augmented with missingness approximations during training (inspired by ROAR [HEK+18], see Appendix E.6 for further discussion). We retrain our models by randomly removing 50% of the patches (by blacking out for ResNet and dropping the respective tokens for ViT). The other training hyperparameters are maintained the same as the standard models above.

Infrastructure and computational time. For ImageNet, we train our models on 4 V100 GPUs each, and training took around 12 hours for ResNet-18 and ViT-T, and around 20 hours for ResNet-50 and ViT-S.

For CIFAR-10, we fine-tune pretrained ViTs and ResNets on a single V100 GPU. Fine-tuning ViT-T and ResNet-18 took around 1 hours, and fine-tuning ViT-S and ResNet-50 took around 1.5 hours.

All of our analysis can be run on a single 1080Ti GPU, where the time for one forward pass with batch size of 128 is reported in Table E.2.

Table E.2: A collection of neural network architectures we use in this chapter.

Architecture	ViT-T	ResNet-18	ViT-S	ResNet-50
Inference time (sec)	0.031 ± 0.018	0.033 ± 0.013	0.041 ± 0.016	0.039 ± 0.015

E.1.3 Experimental Details for Section 5.2

For the experiments in Section 5.2, we iteratively remove subregions from the input. In this chapter, we consider removing 16×16 patches: we black out patches for the ResNet-50 and drop the corresponding token for the ViT-S. We consider other patch sizes as well as superpixels in Appendix E.3.

We consider removing patches in three orders: random, most salient, and least salient. We use saliency as a rough heuristic for relevance to the image (typically, more salient regions tend to be in the foreground and less salient regions in the background). For all models, we determine the salience of an image subregion as the mean value of that

subregion for a standard ResNet-50's saliency map (the order of patches removed is thus the same for both the ResNet and the ViT).

E.1.4 Experimental Details for Section 5.3

Overview on LIME. Local interpretable model-agnostic explanations (LIME) [RSG16b] is a common method for feature attribution. Specifically, LIME proceeds by generating perturbations of the image, where in each perturbation the subregions are randomly turned on or off. For ResNets, we turn off subregions by masking them with some baseline color, while for ViTs we drop the associated tokens. After evaluating these perturbations with the model, we fit classifier using Ridge Regression to predict the value of the logit of the original predicted class given the presence of each subregion. The LIME explanation is then the weight of each subregion in the ridge classifier (these are often referred to as LIME scores). We perform LIME with 1000 perturbations, and include an implementation of LIME in our attached code.

Implementation details for LIME consistency plots. For the experiment in Figure 5.8, we evaluate LIME using 8 different baseline colors (the colors are generated by setting the R, G, and B values as either 0 or 1). Then, for each pair of colors, we measure the similarity of their top-k feature sets according to their LIME scores for varying k (using Jaccard similarity) averaged over 10,000 examples. We plot the average over the 28 pairs of colors.

E.2 Implementing missingness by dropping tokens in vision transformers

As described in Section 5.1.2, the token-centric nature of vision transformers enables a more natural implementation of missingness: simply drop the tokens that correspond to the removed image subregions. In this section, we provide a more detailed description of dropping tokens, as well as a few implementation considerations.

Recall that a ViT has two stages when processing an input image $\mathbf{x}$.

- **Tokenization:** $\mathbf{x}$ is split into 16×16 patches and positionally encoded into tokens.
- **Self-Attention:** The set of tokens is passed through several self-attention layers and produces a class label.

After the initial tokenization step, the self-attention layers of the transformer deal solely with sets of tokens, rather than a constructed image. This set is not constrained to a specific size. Thus, after the patches have all be tokenized, we can remove the tokens that correspond to removed regions of the input before passing the reduced set to the self-attention layers. The remaining tokens retain their original positional encodings.

Our attached code includes an implementation of the vision transformer which takes in an optional argument of the indices of tokens to drop. Our implementation can also handle varying token lengths in a batch (we use dummy tokens and then mask the self-attention layers appropriately).

Dropping tokens for superpixels and other patch sizes In the main body of this chapter, we deal with 16×16 image subregions, which aligns nicely with the tokenization of vision transformers. In Appendix E.3, we consider other patch sizes that do not align along the token boundaries, as well as irregularly shaped superpixels. In these cases, we conservatively drop the token if any portion of the token was supposed to be removed (we thus remove a slightly larger subregion).

E.3 Additional experiments (Section 5.2)

E.3.1 Additional examples of the bias (Similar to Figure 5.2).

In Figure E.1, we display more examples that qualitatively demonstrate the missingness bias.

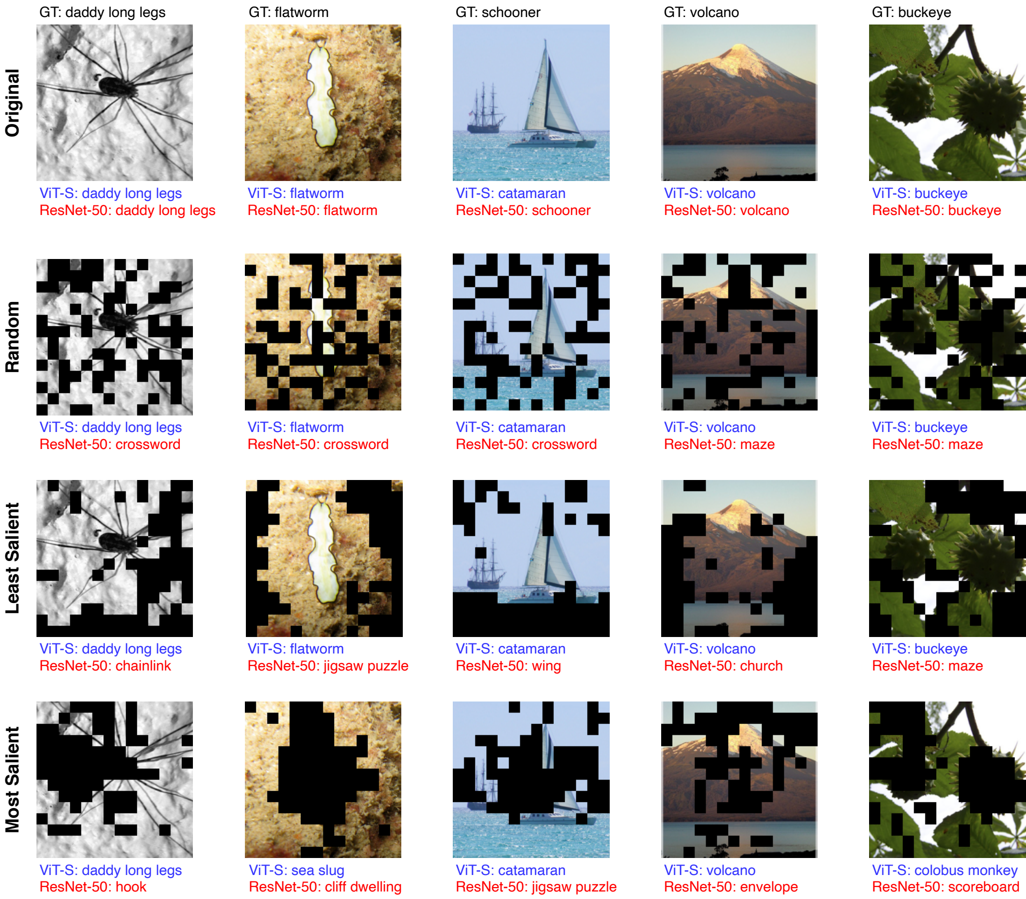
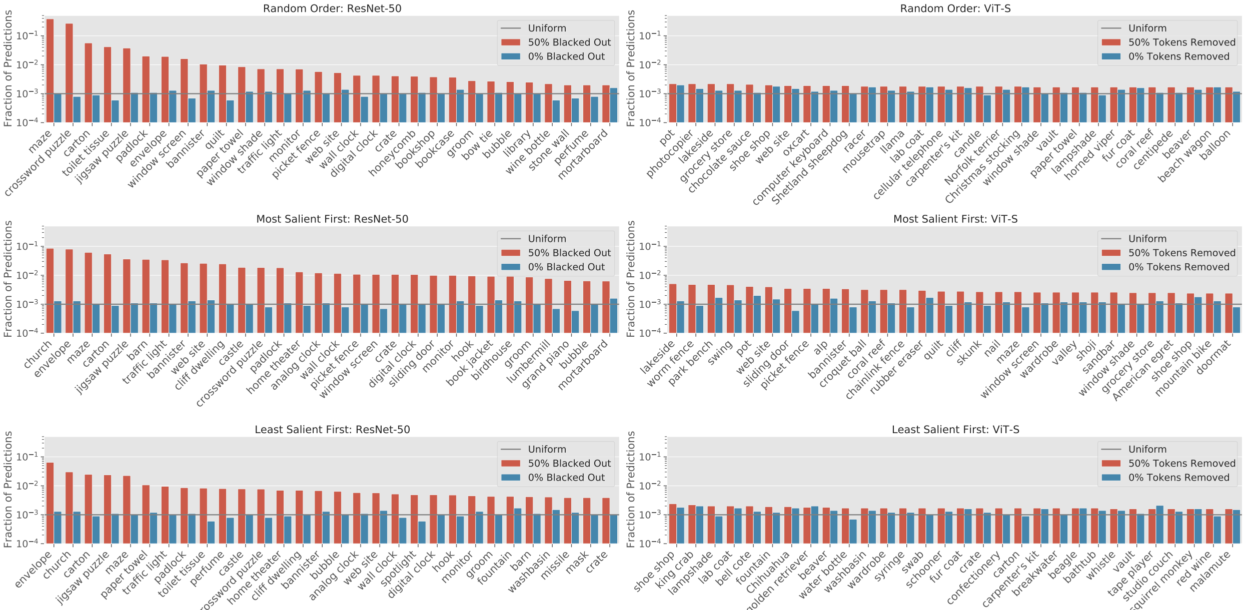


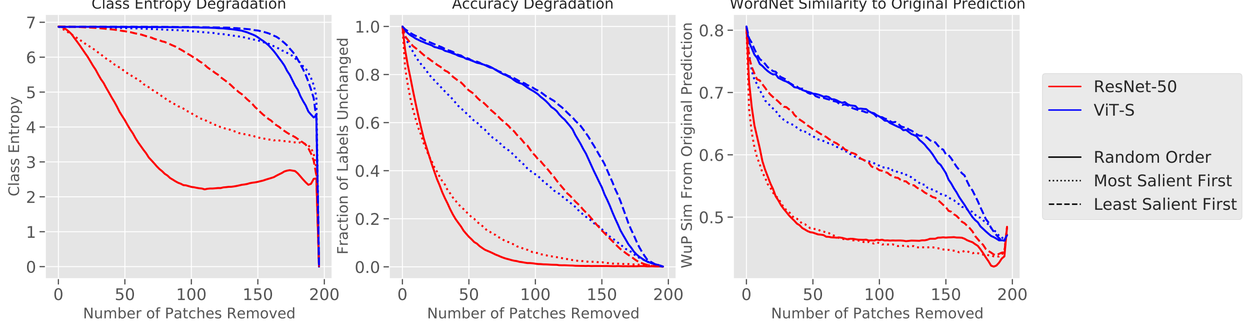
Figure E.1: Further examples of removing 75 16×16 patches from ImageNet images. The images are blacked out for ResNet-50, and the corresponding tokens are dropped for ViT-S. While ResNet-50 skews toward classes that are unrelated to the remaining image features (i.e crossword, jigsaw puzzle), the ViT-S either maintains its original prediction or predicts a reasonable label given remaining image features.

E.3.2 Bias for removing patches in various orders

In this section, we display results for the experiments in Section 5.2 where we remove patches in 1) random order 2) most salient first and 3) least salient first (Figure E.2). We find that missingness approximations skew the output distribution for ResNets regardless of what order we remove the patches. Similarly, we find that the ResNet's predictions flip rapidly in all three cases (though to varying extents). Finally, the ViT mitigates the impact of missingness bias in all three cases.



(a) The shift in the output class distribution after applying missingness approximations in different orders.



(b) Degradation in class entropy (left), the fraction of predictions that change (middle), and the average WordNet similarity if the prediction changes after masking (right) as we remove patches from the image in different orders.

Figure E.2: Full experiments for removing 16×16 patches by blacking out (ResNet-50) or dropping tokens (ViT-S).

E.3.3 Results for different architectures

In this section, we repeat the experiments in Section 5.2 with several other training schemes and types of architectures. Our results parallel our findings in the main body of this chapter.

ViT-T and ResNet-18

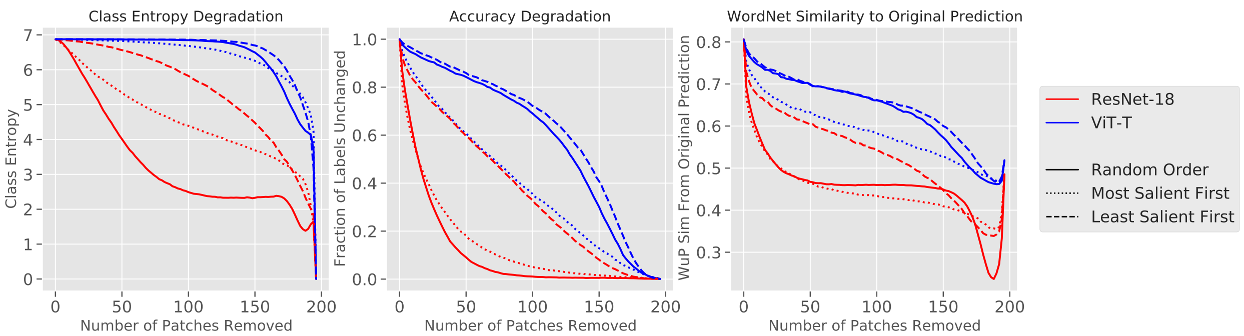


Figure E.3: Bias experiments as in Section 5.2, with a ViT-T and ResNet-18.

ViT-S and Robust ResNet-50

We consider a ViT-S and an $L2$ adversarially robust ResNet-50 ($\epsilon = 3$).

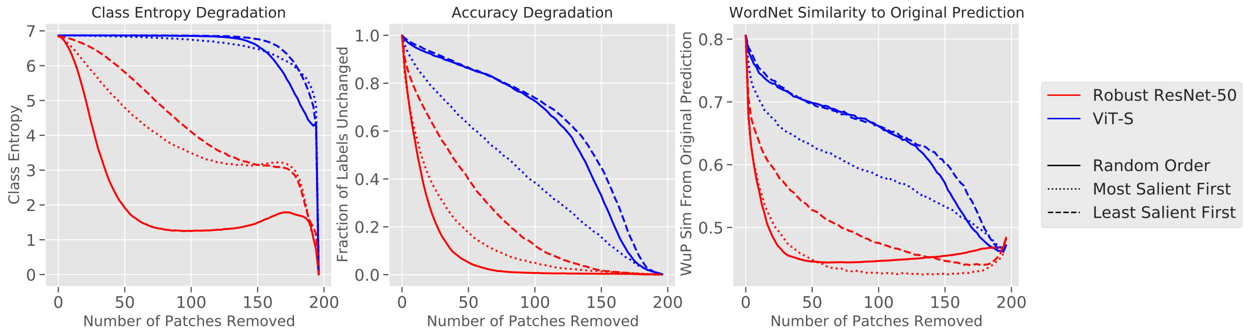


Figure E.4: Bias experiments as in Section 5.2, with a ViT-S and a robust ResNet-50.

ViT-S and InceptionV3

We consider a ViT-S and an InceptionV3 ([SVI+16]) model.

ViT-S and VGG-16

We consider a ViT-S and a VGG-16 with BatchNorm ([SZ15]).

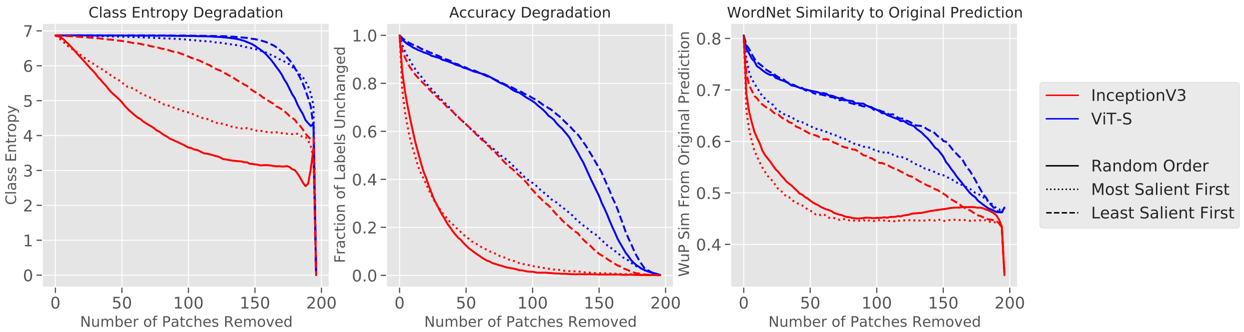


Figure E.5: Bias experiments as in Section 5.2, with a ViT-S and InceptionV3.

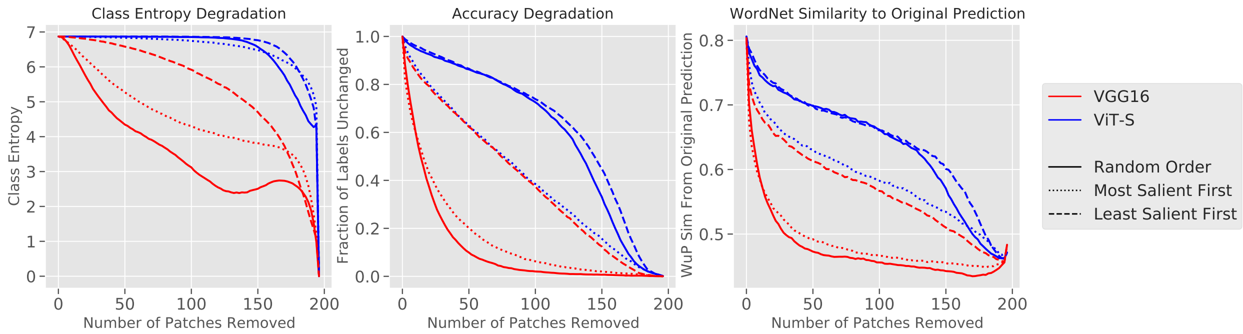


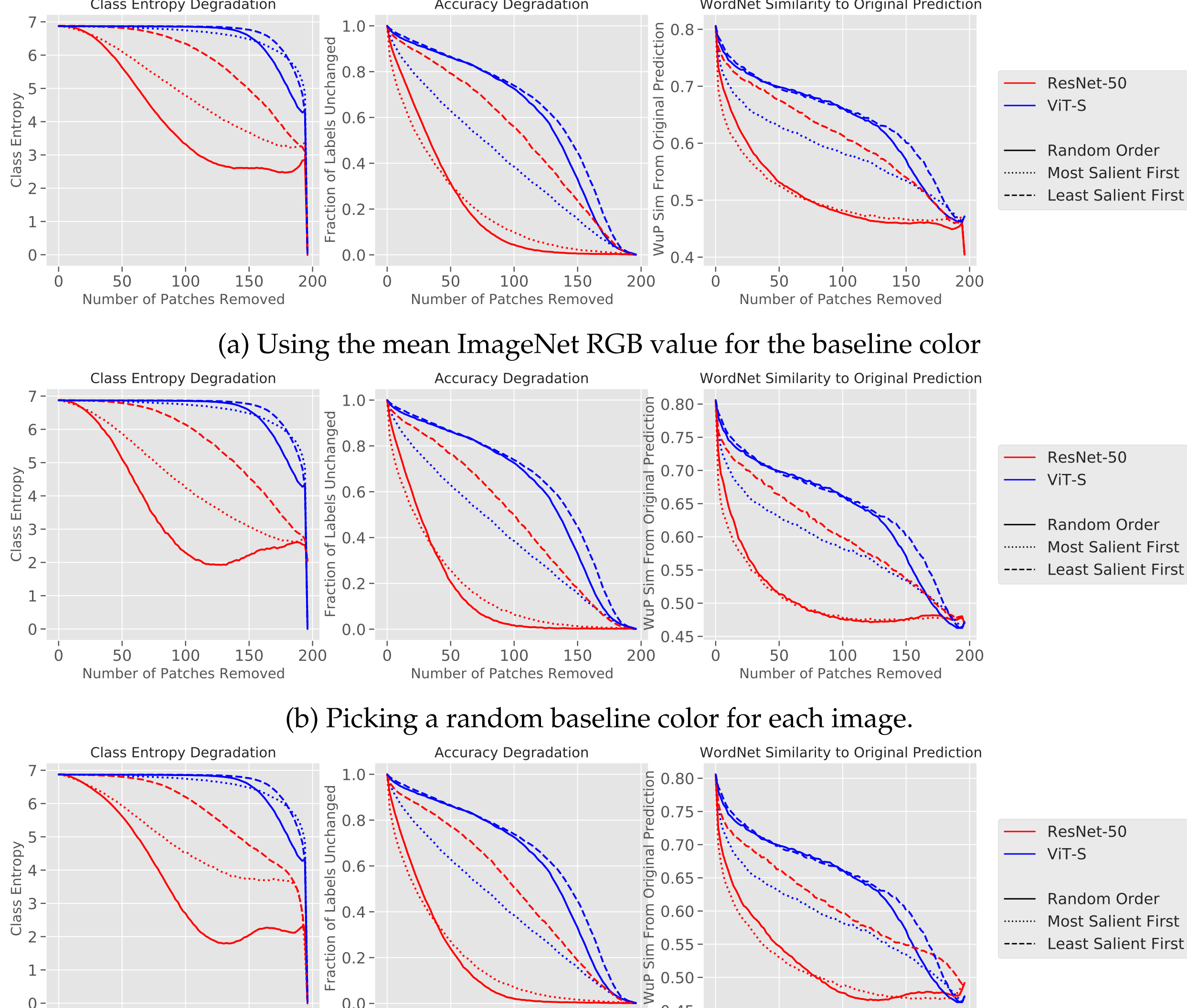
Figure E.6: Bias experiments as in Section 5.2, with a ViT-S and VGG16.

E.3.4 Results for different missingness approximations

In this section, we consider missingness approximations other than blacking out pixels. In Figure E.7, we use three baselines: a) the mean RGB value of the dataset, b) a randomly selected baseline color for each image, and c) a randomly selected color for each pixel [SLL20; STY17]. Since we drop tokens for the vision transformers, changing the baseline color does not change the behavior for the ViTs. Our findings in the main body of this chapter for blacking out patches closely mirror the findings for other baselines.

We also consider blurring the removed features, as suggested in [FV17]. We use a gaussian blur with kernel size 21 and $\sigma = 10$. Examples of blurred images can be found in Figure E.8a. Unlike the previous missingness approximations, *this method does not fully remove subregions of the input*; thus, blurring the pixels can still leak information from the removed regions, which can then influence the model's prediction. Indeed, we find that, by visual inspection, we can still roughly distinguish the label of images that are entirely blurred (as in Figure E.8a).

For completeness, we repeat the experiments in Section 5.2 using the blurred image as the missingness approximation (Figure E.8b). We find that ResNets still experience missingness bias, though the bias is reduced compared to using an image-independent

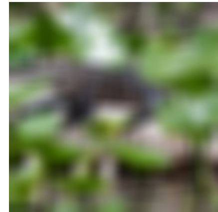


(a) Using the mean ImageNet RGB value for the baseline color

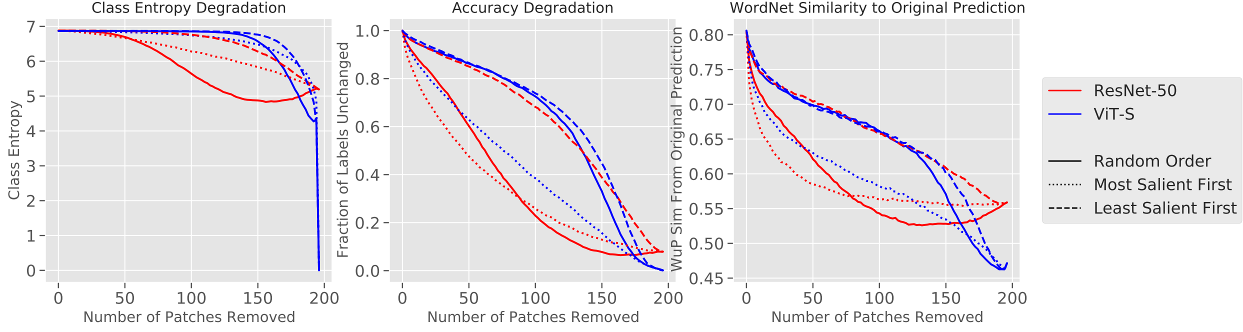
(b) Picking a random baseline color for each image.

(c) Picking a random baseline color for each pixel.

Figure E.7: Using different baseline colors for masking pixels.



(a) Examples of blurring ImageNet images.



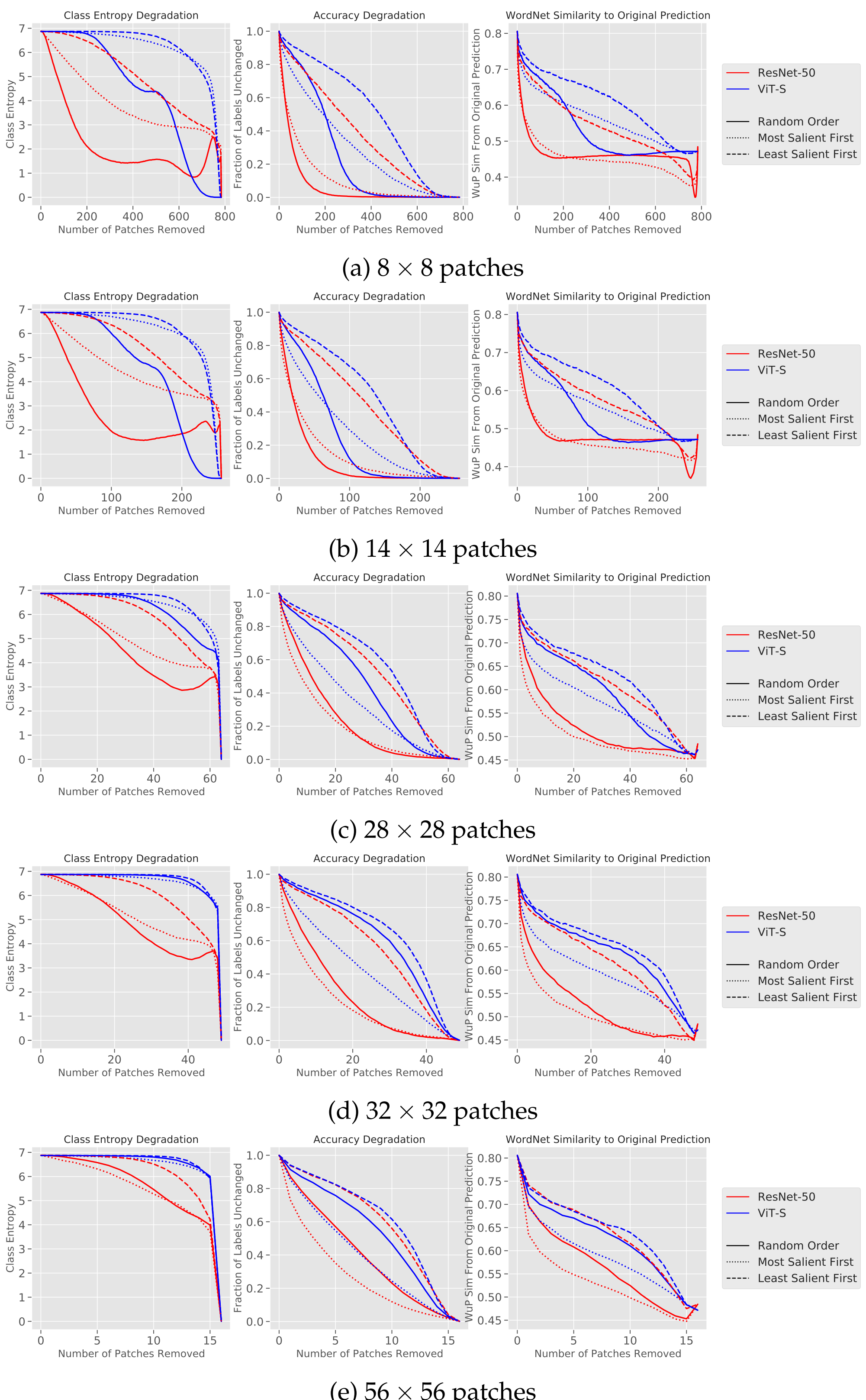
(b) Repeating the experiments in Section 5.2 using the blurred ImageNet image.

Figure E.8: Using the blurred image for the missingness approximation.

baseline color.

E.3.5 Using differently sized patches

In the main body of this chapter, we consider image subregions of 16×16. In this section, we consider subregions of other patch size: 14×14, 28×28, 32×32, and 56×56. As mentioned in Appendix Section E.2, when dropping tokens for the ViT, we conservatively drop the token if *any* part of the corresponding image subregion is being removed. Thus, the ViT removes slightly more area than the ResNet for patch sizes that are not multiples of 16. We find that the ResNet is impacted by missingness bias regardless of the patch size, though the effects of the bias is reduced for very large patch sizes (see Figure E.9).



(a) 8×8 patches

(b) 14×14 patches

(c) 28×28 patches

(d) 32×32 patches

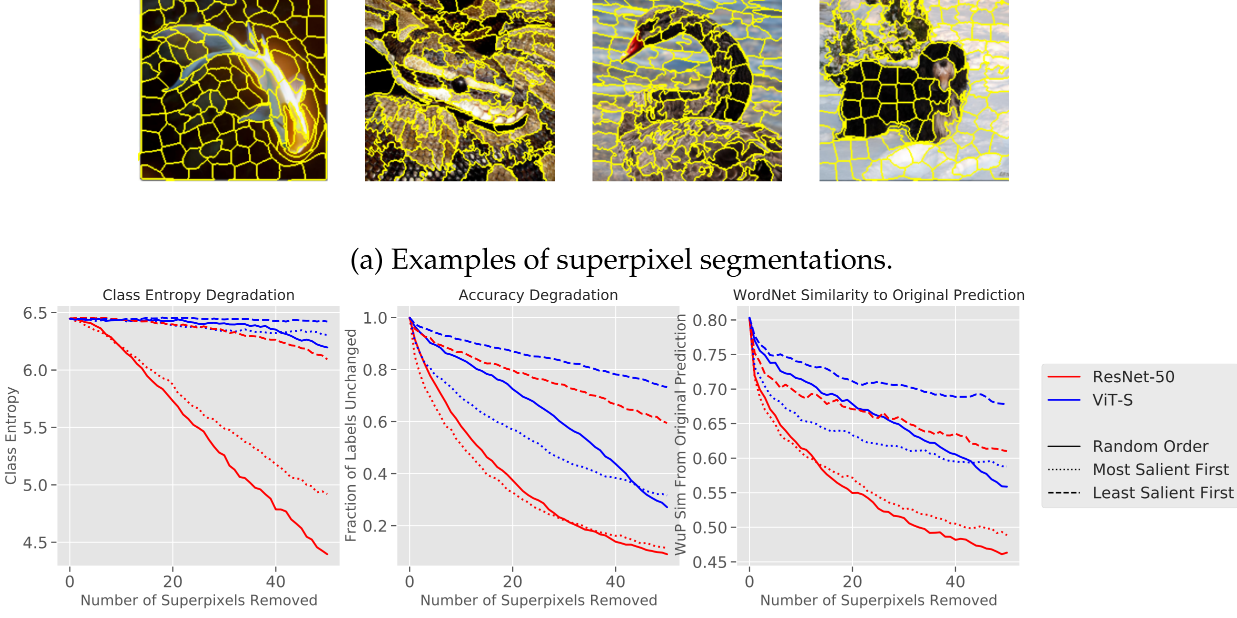
(e) 56×56 patches

Figure E.9: Using different patch sizes for masking pixels.

E.3.6 Using superpixels instead of patches

Thus far, we have used square patches. What if we instead use superpixels? In this section, we compute the SLIC segmentation of superpixels [ASS+10]. In order to keep the superpixels roughly the same size (and of a more similar size as the patches in this chapter), we consider the images having more than 130 superpixels. We display examples of the superpixels in Figure E.10a.

We then repeat our experiments from Section 5.2, and analyze our models' predictions after removing 50 superpixels in different orders (blacking out for ResNets and dropping tokens for ViTs). As described in Section E.2, we conservatively drop all tokens for ViTs that contain any pixels that should have been removed. We find that ResNets are significantly impacted by missingness bias when masking out superpixels. As in the case of patches, dropping tokens through the ViT substantially mitigates the missingness bias.



(a) Examples of superpixel segmentations.

(b) Measuring the impact of removing superpixels through missingness approximations.

Figure E.10: Using SLIC superpixels instead of patches.

E.3.7 Comparison of dropping tokens vs blacking out pixels for ViTs

We compare the effect of implementing missingness by dropping tokens to simply blacking out pixels for ViTs. Figure E.11, shows a condensed version of the experiments we did previously in this section, but now including an additional baseline which is a ViT-S

with blacking out pixels instead of dropping tokens. We find that using either of the ViTs instead of the ResNet significantly mitigates missingness bias on all three metrics. However, dropping tokens for ViTs mitigates missingness bias more effectively than simply blacking out pixels.

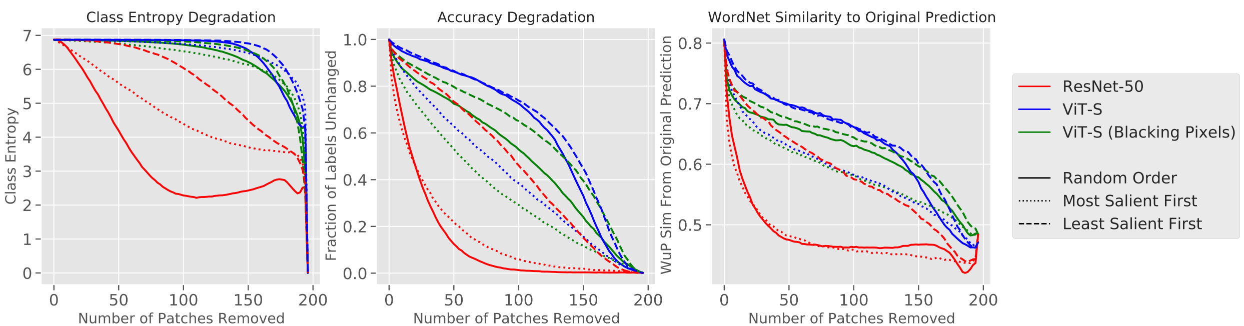


Figure E.11: We compare dropping tokens vs blacking out pixels for ViTs.

E.4 Additional experiments (Section 5.3)

E.4.1 Examples of LIME

In this section, we display further examples of LIME explanations for ViT-S, ResNet-50, and a ResNet-50 retrained with missingness augmentations (Figure E.12). As we explain in Section 5.3, LIME relies heavily on the notion of missingness, and can be subject to missingness bias. We note that the ViT-S and retrained ResNets qualitatively have more human-aligned LIME explanations (by highlighting patches in the foreground over patches in the background) compared to a standard ResNet. While human-alignment does not a guarantee that the LIME explanation is good (the model might be relying on non-aligned features), we do see a substantial difference in the explanations of models robust to the missingness bias (ViTs and retrained ResNet) and models suffering from this bias (standard ResNet).

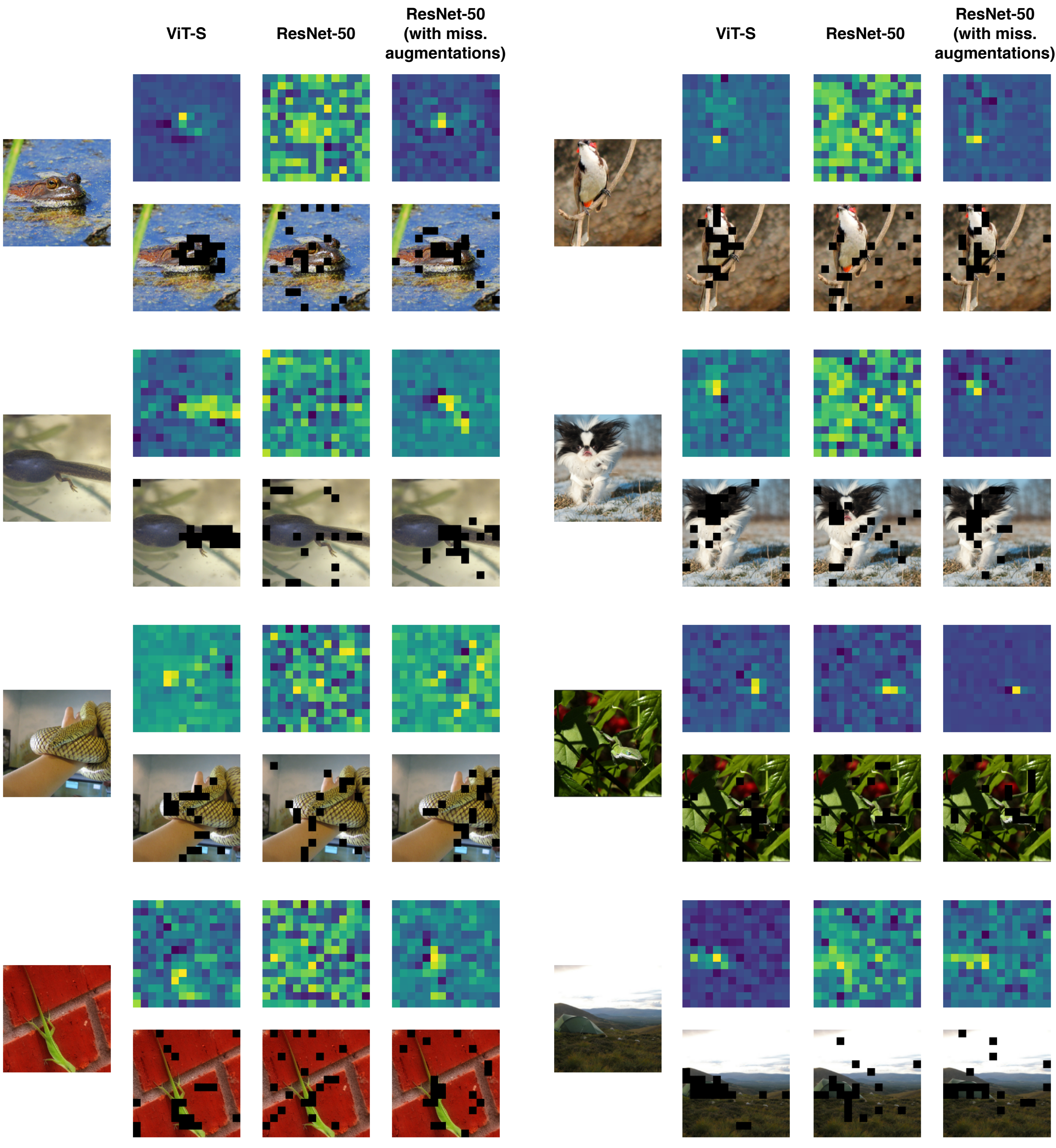


Figure E.12: Examples of LIME explanations.

E.4.2 Top-k ablation test with superpixels.

We repeat the top-k ablation test in Section 5.3, using superpixels instead of patches (Figure E.13). The setup for superpixels is that same as that described in Appendix E.3.6. After generating LIME explanations for a ViT and ResNet-50, we evaluate these explanations using the top-k ablation test. As we found for 16×16 patches, the explanations when evaluating with a ResNet are less distinguishable than when evaluating for a ViT: even masking features according to the random explanations rapidly flips the predictions. We do find that masking random superpixels seems to have a greater effect on ViTs than masking random 16×16 patches: this is likely because the superpixels are on average larger.

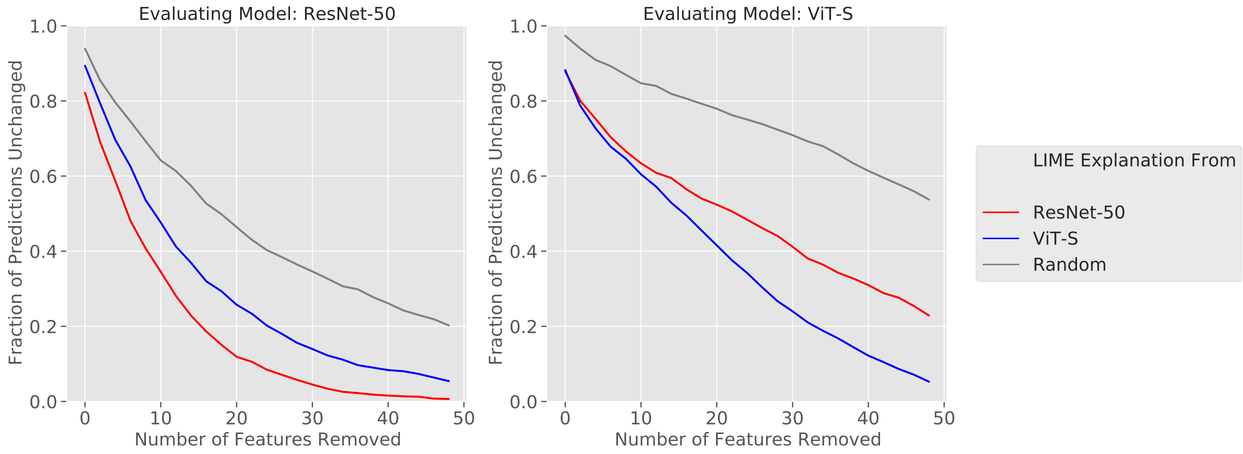


Figure E.13: Top K ablation test using superpixels instead of patches.

E.4.3 Effects of Missingness Bias on Learned Masks

In this section, we consider a different model debugging method ([FV17]) which also relies on missingness. In this method, a minimal mask is directly optimized for each image. We implement this method at the granularity of 16×16 patches.

Model Debugging through a learned mask ([FV17]) Specifically x be the input image, $\mathcal{M}$ be the model, c be the model's original prediction on x, and b be a baseline image (in our case, blacked out pixels). The method optimizes for m, a 14×14 grid of weights between 0 and 1 which assigns importance to each patch. We define the perturbation via m as a linear combination of the input image and the baseline image for each patch (plus

some normally distributed noise ϵ):

$$f(x, m) = x * \text{upsample}(m) + b * (1 - \text{upsample}(m)) + \epsilon$$

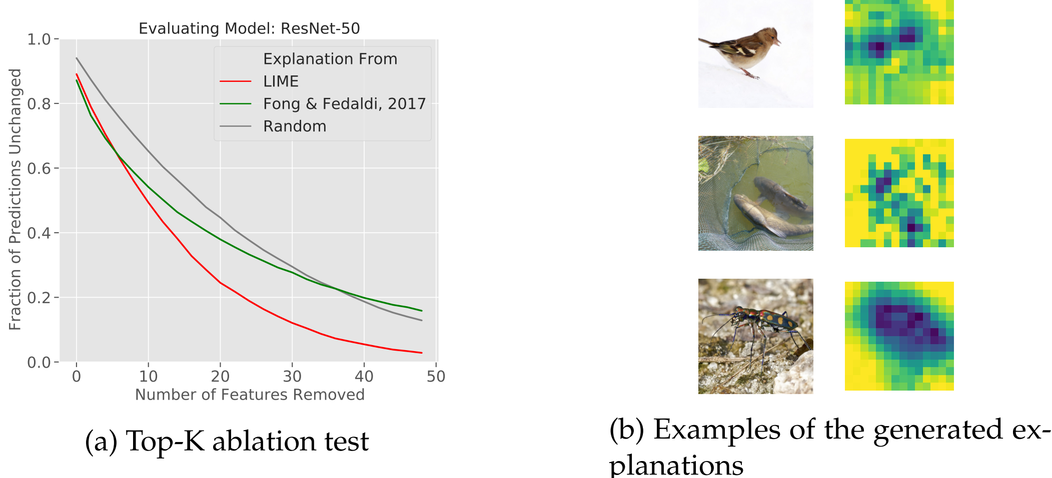
Then the optimal $\hat{m}$ is computed as:

$$\hat{m} = \arg\min_{m} \lambda_1 ||\mathbf{1} - m||_1 + \lambda_2 ||m||_{TV}^{\beta} + [\mathcal{M}(f(x, m))]_c$$

with $\lambda_1 = 0.01, \lambda_2 = 0.2, \beta = 3, \epsilon \sim \mathcal{N}(0, 0.04)$.

The optimal m is computed through backpropagation, and can then be treated as a model explanation. Parameters and implementation were adapted from the implementation at https://github.com/jacobgil/pytorch-explain-black-box.

Assessing the impact of missingness bias. Since m must be backpropagated, we cannot leverage the drop tokens method for ViTs (which would require m for each patch to be either 0 or 1). However, we generate explanations for the ResNet-50 in order to examine the impact of missingness bias. As in Section 5.3, we evaluate the explanation generated by this method alongside a random baseline and the LIME explanation for that ResNet using the top-K ablation test (Figure E.14). As is the case for the LIME explanations, missingness bias renders the explanations generated by this method indistinguishable from random.



(a) Top-K ablation test

(b) Examples of the generated explanations

Figure E.14: (a) Top K ablation test evaluated on a ResNet-50. We evaluate explanations generated by LIME, [FV17], and a random baseline. Due to missingness bias, the explanations are indistinguishable from random (b) Examples of explanations generated by [FV17].

E.5 Other Datasets

While this chapter largely focuses on ImageNet, we include here a few results on other datasets.

E.5.1 MS-COCO

MS-COCO ([LMB+14]) is an object recognition dataset with 80 object recognition categories that provides bounding box annotations for each object. We consider the multi-label task of object tagging, where the model predicts whether each object class is present in the dataset. We train object tagging models using a ResNet-50 and ViT-S, with an Asymmetric Loss as in [BRZ+20]. An object is predicted as "present" if the outputted logit is above some threshold.

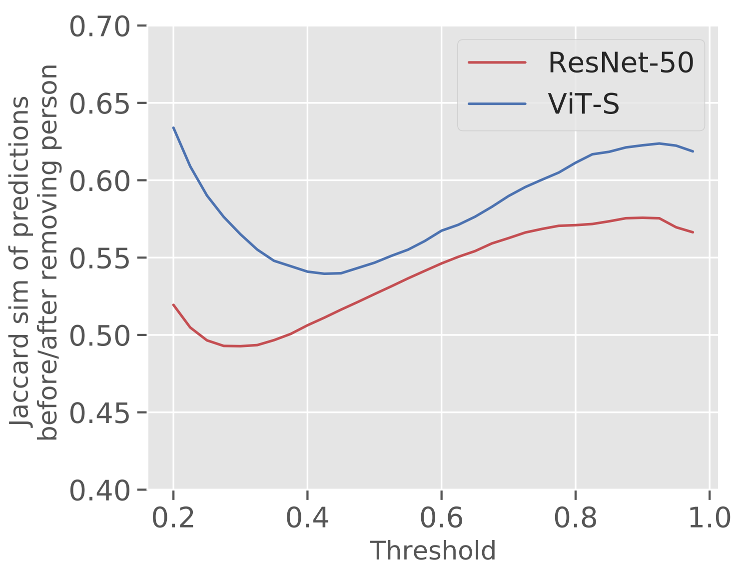
We study the setting of removing people from images using missingness. In particular, we seek to remove the image regions contained inside a "person" bounding box that is not contained in a bounding box for another non-person object. Examples of removing people from the image can be found in Figure E.15. We then seek to check whether removing the person affected the model predictions for other, non-person object classes.

Specifically, we consider the 21,634 images in the MS-COCO validation set that contain people. For each image, we evaluate our model on the original image and the image with the person removed (blacking out for the ResNet-50 and dropping the token for the ViT). Then, to measure the consistency of the non-person predictions, we compute the Jaccard similarity for the set of predicted objects (excluding the person class) before and after masking. We plot the average similarity over all the images for different prediction thresholds in Figure E.15.

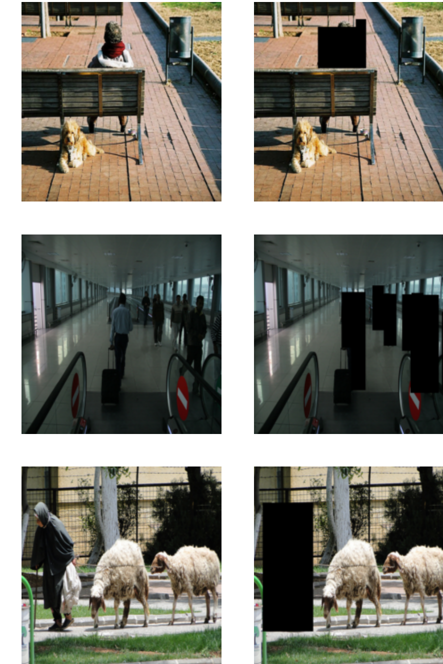
We find that the ViT more consistently maintains the predictions of the non-person object classes when masking out people. In contrast, masking out people for the ResNet is more likely to change the predictions for other object classes.

E.5.2 CIFAR-10

We consider the setting of CIFAR-10 ([Kri09]). Specifically, we train a ResNet-50 and a ViT-S on the CIFAR-10 dataset upsampled to 224x224 pixels. We start training from the ImageNet checkpoints used throughout this chapter. This step is necessary for ensuring high accuracy when training ViTs on CIFAR-10 ([TCD+20]). We then consider how randomly removing 16x16 patches from the upsampled CIFAR images changes the prediction. Similarly to the case of ImageNet, we find that the ResNet-50 more rapidly changes its prediction as



(a) Consistency of non-person prediction after removing people from the images.



(b) Examples of removing people from MS-COCO images

Figure E.15: (a) Average Jaccard similarity of the set of non-person predictions before and after removing all people from the image. We plot over prediction thresholds for the tagging task. (b) Examples of removing people from MS-COCO images.

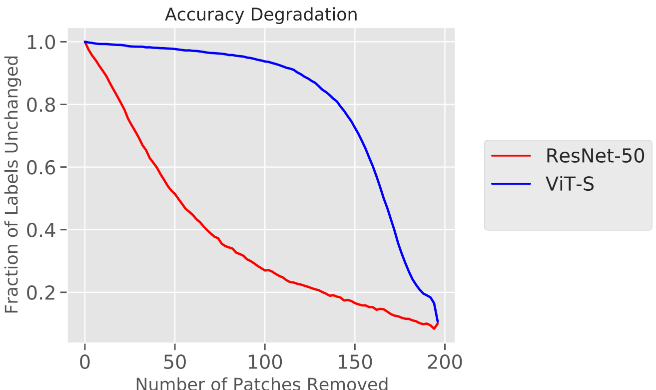


Figure E.16: We plot the fraction of images where the prediction does not change as image regions are removed for the CIFAR-10 dataset.

random parts of the image are masked, while the ViT-S maintains its prediction even as large parts of the image are removed.

E.6 Relationship to ROAR

Here, we present more details about the ROAR experiment of Section 5.2.

E.6.1 Overview on ROAR

Evaluating feature attribution methods requires the ability to remove features from the input to assess how important these features are to the model's predictions. To do so properly, Hooker et al. [HEK+18] argue that re-training (with removing pixels) is required so that images with removed features stay in-distribution. Their argument holds since machine learning models typically assume that the train and the test data comes from a similar distribution.

So, they propose *RemOve and Retrain (ROAR)* where new models (of the exact same architecture) are *retrained* such that random pixels are blacked out during training. The intuition is that this way, removing pixels do not render images out-of-distribution. Overall, they were able to better assess how much removing information from the image affects the predictions of the model using those retrained surrogate models.

The authors of ROAR list several downsides for their approach though. In particular, retraining models can be computationally expensive. More pressingly, the retrained model is *not* the same model that they analyze, but instead a surrogate with a substantially different training procedure: any feature attribution or model debugging result inferred from the retrained model might not hold for the original model. Given these downsides, is retraining always necessary?

E.6.2 ViTs do not require retraining

Here, we show that retraining is not always necessary: indeed for ViTs, we do not need to retraining to be able to properly evaluate feature attribution methods. While ROAR in [HEK+18] dealt with blacking out features on a per-pixel level, we adapt their approach for masking out larger contiguous regions (like patches). If we apply missingness approximations during training as in ROAR, missingness approximations are now in-distribution, and thus would likely mitigate the observed biases.

We retrain a ResNet-50 and a ViT-S by randomly removing 50% of patches during training (through blacking out pixels for the ResNet-50 and dropping tokens for the ViT-S). Our goal is to compare the behavior of each model to its retrained counterpart. If retraining does not change the model's behavior when missingness approximations are applied, retraining would be unnecessary, and we can instead confidently use the original

model. In Figure 5.5, we measure the fraction of images where the model changes its prediction as we remove image features for both the standard and retrained models. We find that, while there is a significant gap in behavior between the standard and retrained CNNs, the standard and retrained ViTs behave largely the same..

This result indicates that, while retraining is important when analyzing CNNs, it is unnecessary for ViTs: we can instead intervene on the original model. We thus avoid the expense of training the augmented models, and can perform feature attribution on the actual model instead of a proxy.

Appendix F

Additional details for Chapter 6

F.1 Experiment dashboard

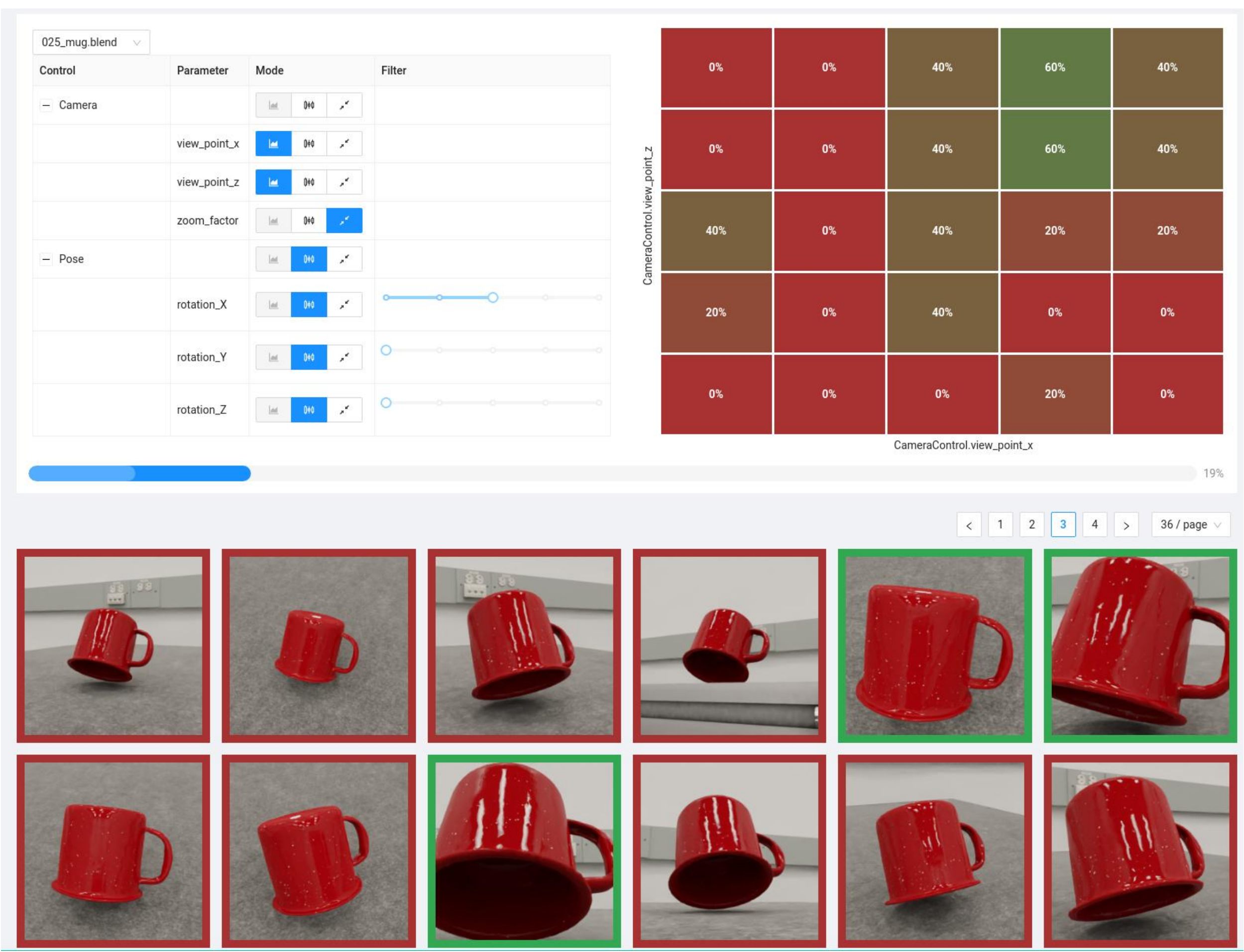


Figure F.1: The *3DB* dashboard used for data exploration.

Since experiments usually produce large amounts of data that can be hard to get a sense of, we created a data visualization dashboard. Given a folder containing the JSON logs of a job, it offers a user interface to explore the influence of the controls.

For each parameter of each control, we can pick one out three mode:

- **Heat map axis**: This control will be used as the `x` or `y` axis of the heat map. Exactly two controls should be assigned to this mode to enable the visualization. Hovering on cells of the heat map will filter all samples falling in that region.
- **Slider**: This mode enables a slider that is used to only select the samples that match exactly this particular value.
- **Aggregate**: do not filter samples based on this parameter

F.2 iPhone app

We developed a native iOS app to help align objects in the physical experiment (Section 6.3). The app allows the user to enter one or more rendering IDs (corresponding to scenes rendered by *3DB*); the app then brings up a camera with a translucent overlay of either the scene or an edge-filtered version of the scene (cf. Figure F.2). We used the app to align the physical object and environment with their intended place in the rendered scene. The app connects to the same backend serving the experiment dashboard.

F.3 Controls

3DB takes an object-centric perspective, where an object of interest is spawned on a desired background. The scene mainly consists of the object and a camera. The controls in our pipeline affect this interplay between the scene components through various combinations of properties, which subsequently creates a wide variety of rendered images. The controls are implemented using the Blender Python API '`bpy`' that exposes an easy to use framework for controlling Blender. '`bpy`' primarily exposes a scene context variable, which contains references to the properties of the components such as objects and the camera; thus allowing for easy modification.

3DB comes with several predefined controls that are ready to use (see https://3db.github.io/3db/). Nevertheless, users are able (and encouraged) to implement custom controls for their use-cases.

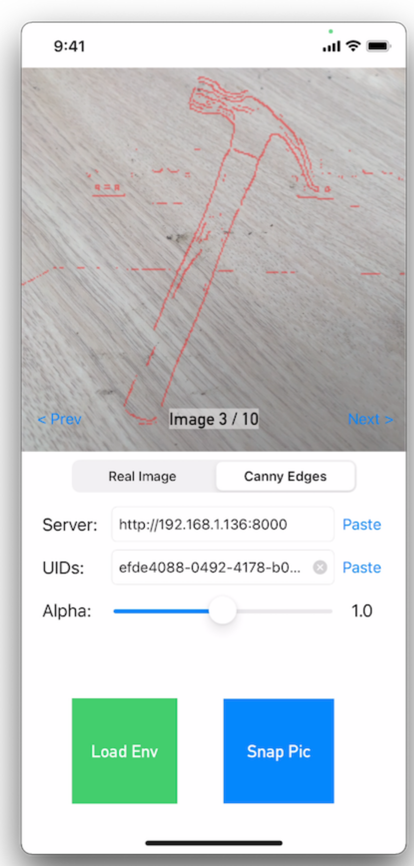
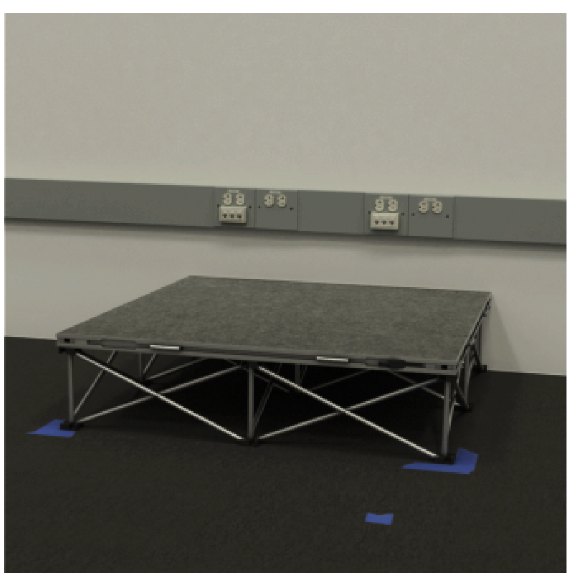


Figure F.2: A screenshot of the iOS app used to align objects for the physical-world experiment. After starting the dashboard server, the user can specify the server location as well as a set of rendering IDs. The corresponding renderings will be displayed over a camera view, allowing the user to correctly position the object in the frame. The user can adjust the object transparency, and can toggle between overlaying the full rendering and overlaying just the edges (shown here).

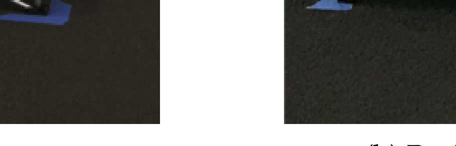
F.4 Additional experiments details

We refer the reader to our package https://github.com/3db/3db for all source code, 3D models, HDRIs, and config files used in the experiments of this chapter.

For all experiments we used the pre-trained ImageNet ResNet-18 included in `torchvision`. In this section we will describe, for each experiment the specific 3D-models and environments used by *3DB* to generate the results.



(a) Synthetic



(b) Real picture (iPhone 12 Pro)

Figure F.3: Studio used for the real-world experiments (Section 6.3).

F.4.1 Sensitivity to image backgrounds (Section 6.2.1)

Analysing a subset of backgrdounds

Models: We collected 19 3D-models in total. On top of the models shown on figure F.6, we used models for: (1) an orange, (2) two different toy power drills, (3) a baseball ball, (4) a tennis ball, (5) a golf ball, (6) a running shoe, (7) a sandal and (8) a toy gun. Some of these models are from `YCB` [CWS+15] and the rest are purchased from `amazon.com` and then put through a 3D scanner to get corresponding meshes.

Environments: We sourced 20 2k HDRI from the website https://hdrihaven.com. In particular we used: `abandoned_workshop`, `adams_place_bridge`, `altanka`, `aristea_wreck`, `bush_restaurant`, `cabin`, `derelict_overpass`,

`dusseldorf_bridge`, `factory_yard`, `gray_pier`, `greenwich_park_03`, `kiara_7_late-afternoon`, `kloppenheim_06`, `rathaus`, `roofless_ruins`, `secluded_beach`, `small_hangar_02`, `stadium_01`, `studio_small_02`, `studio_small_04`.

Analyzing all backgrounds with the "coffee mug" model.

Models: We used a single model: the coffee mug, in order to keep computational resources under control.

Environments: We used 408 HDRIs from https://hdrihaven.com/ with a 2K resolution.

F.4.2 Texture-shape bias (section 6.2.2)

Textures: To replace the original materials, we collected 7 textures on the internet and we modified them to make them seamlessly tilable. These textures are shown on Figure F.6.

Models: We used all models that are shown on Figure F.6.

Environments: We used the virtual studio environment (Figure F.3).

F.4.3 Orientation and scale dependence (Section 6.2.3)

We use the same models and environments that are used in Appendix F.4.1.

F.4.4 3D models heatmaps (Figure 6.12)

Models: For this experiment we used the set of models shown on Figure F.6.

Environments: We used the virtual studio environment (see Figure F.3).

F.4.5 Case study: using 3DB to dive deeper (Section 6.2.4)

Models: We only used the mug since this experiment is mug specific.

Environments: We used the sudio set shown on Figure F.3.

F.4.6 Physical realism (Section 6.3)

Real-world pictures: All images were taken with an handheld Apple iPhone 12 Pro. To help us align the shots we used the application described in appendix F.2.

Models: We used the models shown in Figure 6.15.

Environments: The environment shown on Figure F.3 was especially designed for this experiment. The goal was to have an environment that matches our studio as closely as possible. The geometry and materials were carefully reproduced using reference pictures. The lighting was reproduce through a high resolution HDRI map.

F.5 Omitted figures

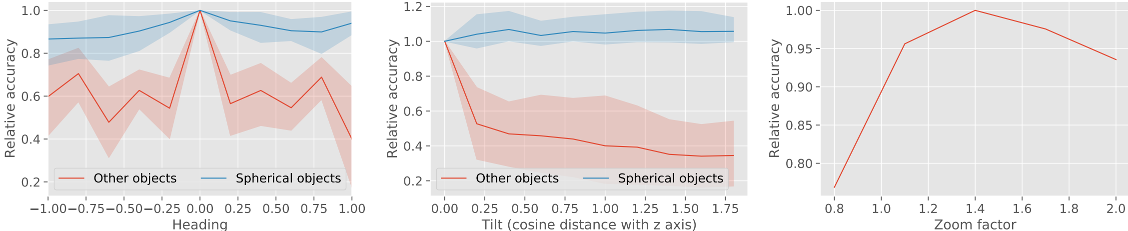
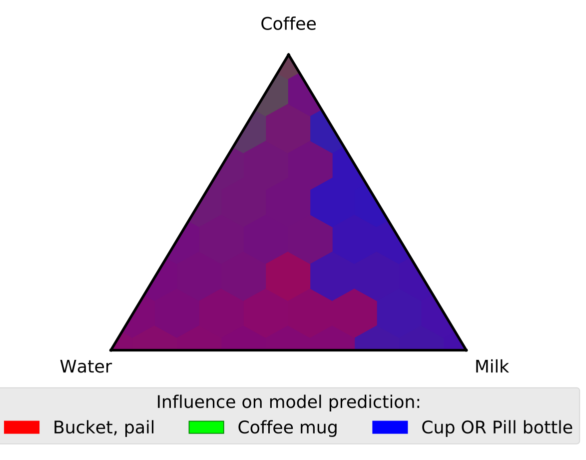


Figure F.4: Additional plots to Figure 6.13. We plot the distribution of model accuracy as a function of object heading (*top*), tilt (*middle*) and zoom (*bottom*), aggregated over variations in controls. For heading and tilt, we separately evaluate accuracy for (non-)spherical objects. Notice how the performance of the model degrades for non-spherical objects as the heading/tilt changes, but not for spherical objects. Also notice how the performance depends on the zoom level of the camera (how large the object is in the frame).

(a) Sample of the images rendered for the experiment presented in section 6.2.4.



(b) Un-normalized version of Figure 6.14-(b).

Figure F.5: Additional illustration for the mug liquid experiment of Figure 6.14. This figure shows the correlation of the liquid mixture in the mug on the prediction of the model, averaged over random viewpoints

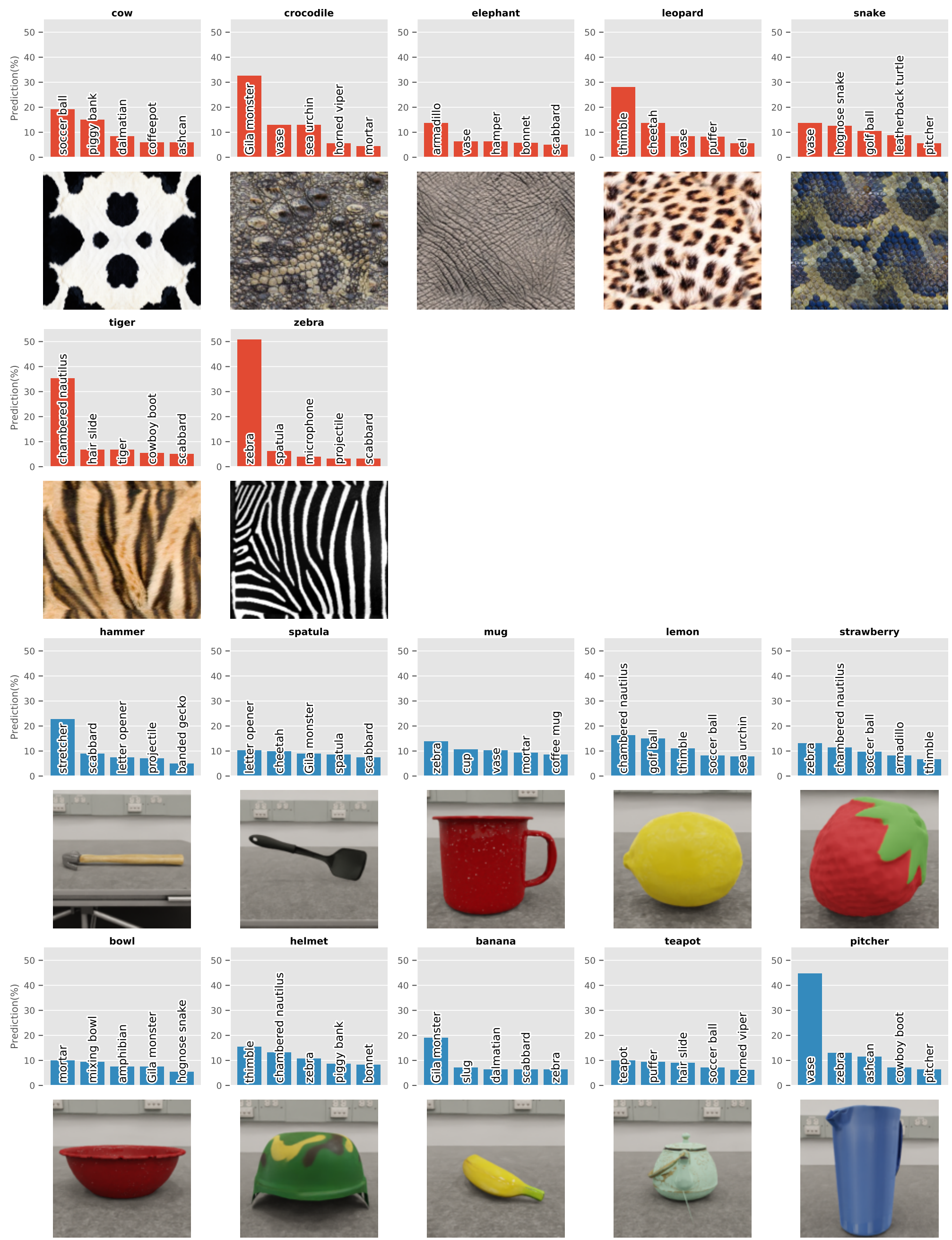


Figure F.6: Additional examples of the experiment in Figure 6.11. Distribution of classifier predictions after the texture of the 3D object model is altered. In the top rows, we visualize the most frequently predicted classes for each texture (averaged over all objects). In the bottom rows, we visualize the most frequently predicted classes for each object (averaged over all textures). We find that the model tends to predict based on the texture more often than based on the object.

Appendix G

Additional details for Chapter 7

G.1 Experimental Setup

G.1.1 ImageNet Models

In this chapter, we train a number of ImageNet models and transfer them to various datasets in Sections 7.2 and 7.4. We mainly use the ResNet-18 architecture all over the chapter. However, we study bias transfers using various architectures in Appendix G.1.4. We use PyTorch's official implementation for these architectures, which can be found here `https://pytorch.org/vision/stable/models.html`.

Training details. We train our ImageNet models from scratch using SGD by minimizing the standard cross-entropy loss. We train for 16 epochs using a Cyclic learning rate schedule with an initial learning rate of 0.5 and learning rate peak epoch of 2. We use momentum of 0.9, batch size of 1024, and weight decay of $5e^{-4}$. We use standard data-augmentation: *RandomResizedCrop* and *RandomHorizontalFlip* during training, and *RandomResizedCrop* during testing. Our implementation and configuration files are available in the attached code.

G.1.2 Transfer details from ImageNet to downstream image classification

Transfer datasets. We use the image classification tasks that are used in [SIE+20; KSL19], which have various sizes and number of classes. When evaluating the performance of models on each of these datasets, we report the Top-1 accuracy for balanced datasets and the Mean Per-Class accuracy for the unbalanced datasets. See Table G.1 for the details of

these datasets. For each dataset, we consider two transfer learning settings: *fixed-feature* and *full-network* transfer learning which we describe below.

Table G.1: Image classification benchmarks used in this chapter. Accuracy metric is the metric we report for each of the dataset across the chapter. Some datasets are imbalanced, so we report Mean Per-Class accuracy for those. For the rest, we report Top-1 accuracy.

Dataset	Size (Train/Test)	Classes	Accuracy Metric
Birdsnap [BLW+14]	32,677/8,171	500	Top-1
Caltech-101 [FFP04]	3,030/5,647	101	Mean Per-Class
Caltech-256 [GHP07]	15,420/15,187	257	Mean Per-Class
CIFAR-10 [Kri09]	50,000/10,000	10	Top-1
CIFAR-100 [Kri09]	50,000/10,000	100	Top-1
FGVC Aircraft [MRK+13]	6,667/3,333	100	Mean Per-Class
Food-101 [BGV14]	75,750/25,250	101	Top-1
Oxford 102 Flowers [NZ08]	2,040/6,149	102	Mean Per-Class
Oxford-IIIT Pets [PVZ+12]	3,680/3,669	37	Mean Per-Class
SUN397 [XHE+10]	19,850/19,850	397	Top-1
Stanford Cars [KDS+13]	8,144/8,041	196	Top-1

Fixed-feature transfer. For this setting, we *freeze* the layers of the ImageNet source model[1], except for the last layer, which we replace with a random initialized linear layer whose output matches the number of classes in the transfer dataset. We now train only this new layer for using SGD, with a batch size of 1024 using cyclic learning rate. For more details and hyperparameter for each dataset, please see config files in the attached code.

Full-network transfer. For this setting, we *do not freeze* any of the layers of the ImageNet source model, and all the model weights are updated. We follow the exact same hyperparameters as the fixed-feature setting.

G.1.3 Compute and training time

Throughout this chapter, we use the FFCV data-loading library to train models fast [LIE+22]. Using FFCV, we can train an ImageNet model, for example, in around 1 hr only on a single V100 GPU. Our experiments were conducted on a GPU cluster containing A100 and V100 GPUs.

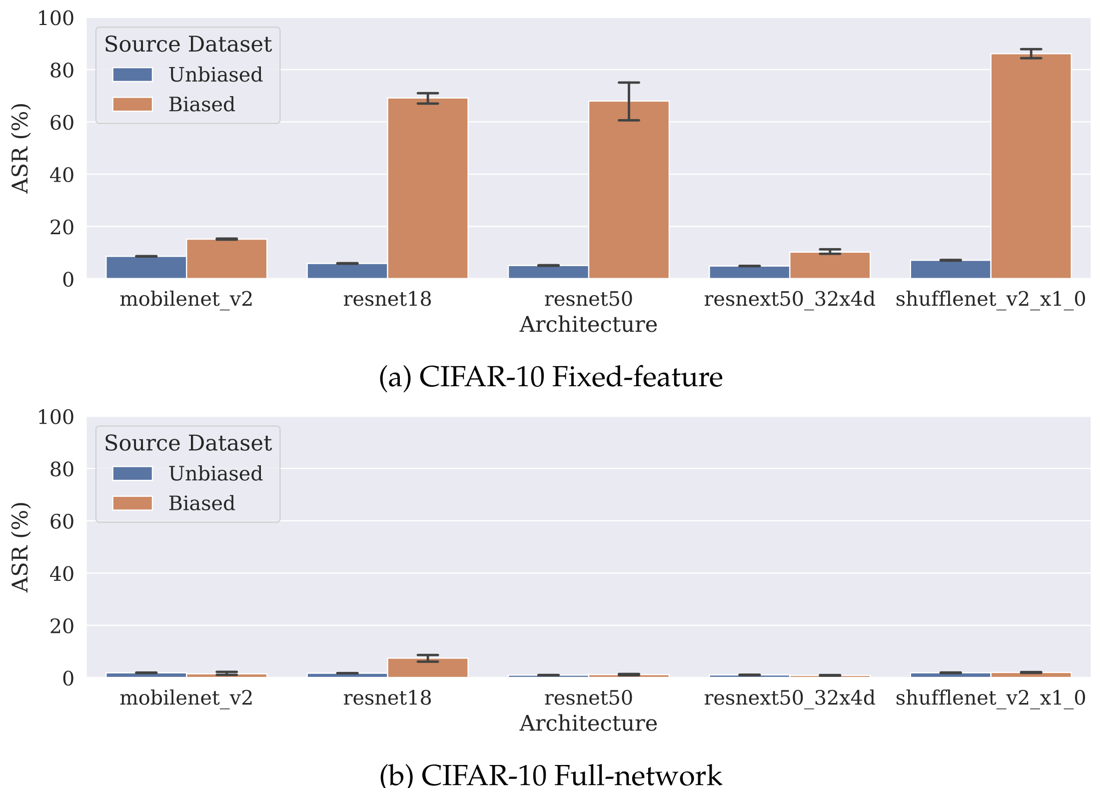
[1] We do not freeze the batch norm statistics, but only the weights of the model similar to [SIE+20].

G.1.4 Varying architectures

In this section, we study whether bias transfers when applying transfer learning using various architectures. We conduct the basic experiment of Section 7.2 on several standard architectures from the PyTorch's Torchvision[2].

As in Section 7.2, we train two versions of each architecture: one on a clean ImageNet dataset, and another on a modified ImageNet dataset containing a backdoor. We use the same hyperparameters as the rest of the chapter, except for the batch size, which we set to 512 instead of 1024. The reason we lower the batch size is to fit these models in memory on a single A100 GPU.

Now, we transfer each of these models to a clean CIFAR-10 dataset, and test if the backdoor attack transfers. Similar to the results of this chapter, we notice that backdoor attack indeed transfers in the fixed-feature setting. We note however that for the full-network setting, all architectures other than ResNet-18 (which we use in the rest of the chapter) seem to be more robust to the backdoor attack.



(a) CIFAR-10 Fixed-feature

(b) CIFAR-10 Full-network

Figure G.1: Backdoor attack (bias) consistently transfers in the fixed-feature setting across various architectures. However, this happens to a lesser degree in the full-network transfer setting.

[2]These models can be found here https://pytorch.org/vision/stable/models.html

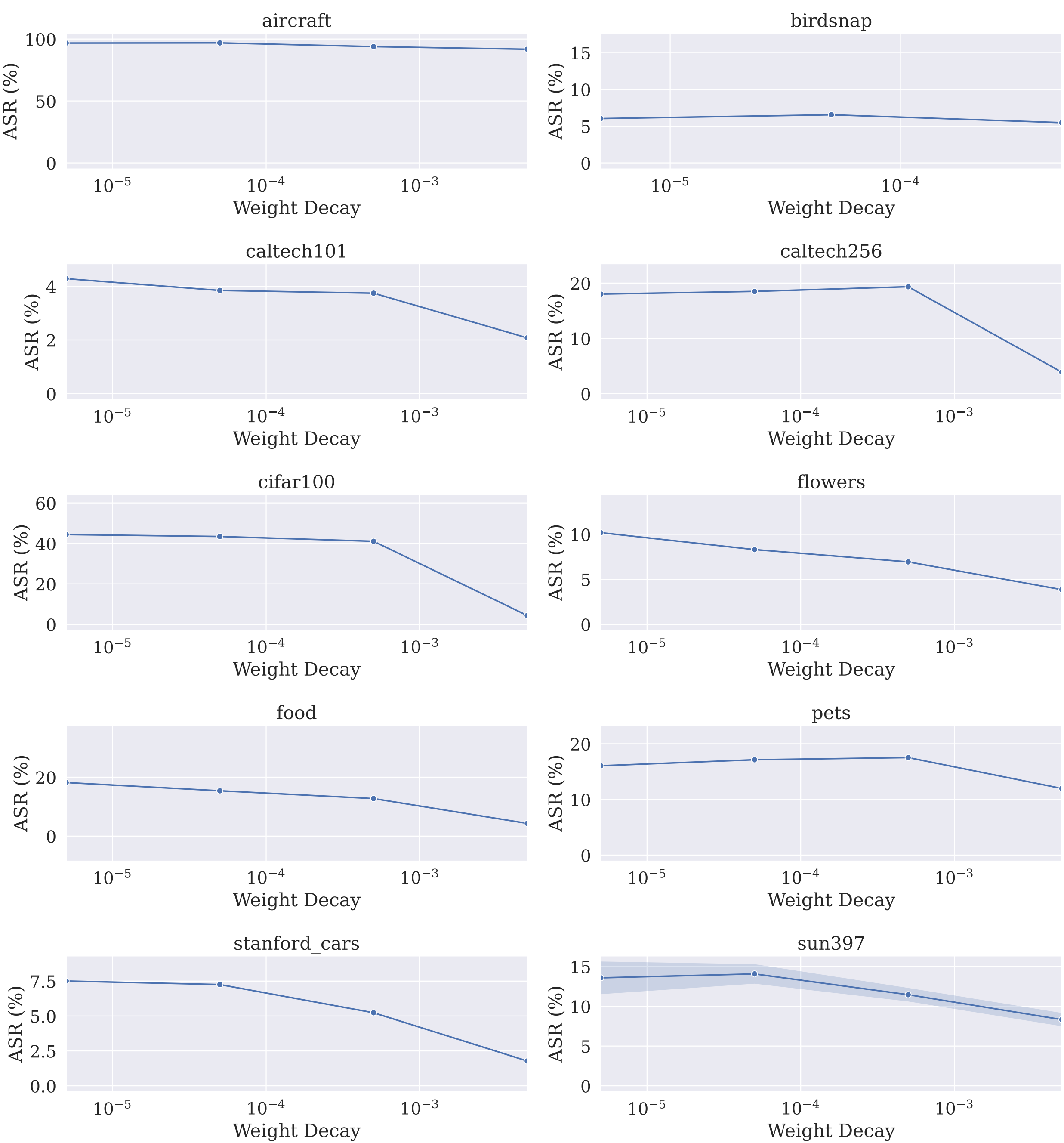


Figure G.2: As weight decay increases, the ASR decreases which means bias transfers less across various datasets. We increase weight decay until the clean accuracy on the target dataset significantly deteriorates (see Figure G.3). Error regions correspond to standard deviation over five random trials.

G.1.5 The effect of weight decay in full-network transfer learning

As mentioned in Section 7.2.2, we found weight decay to have a significant impact on bias transfer in the full-network transfer learning setting. In particular, increasing weight

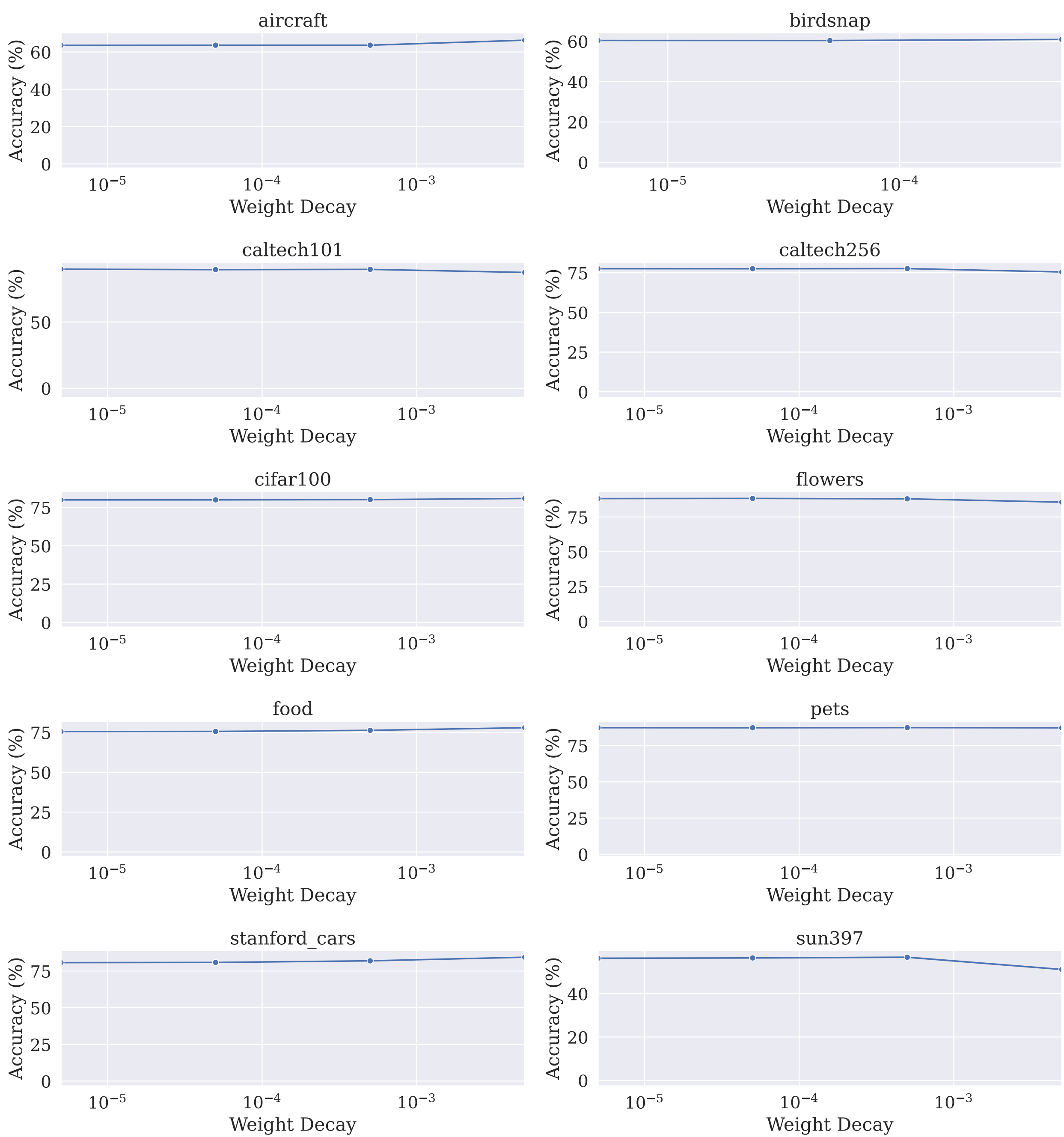


Figure G.3: The clean accuracies corresponding to the weight-decay experiment. We increase weight decay as long as the clean accuracy on the target dataset is roughly the same. Error regions (very small) correspond to standard deviation over five random trials.

decay reduces bias transfer. Here, we present a formal explanation of why this happens by studying this within the logistic regression example we presented in Section 7.1. Recall that, following the setup in Section 7.1, if we transfer a pretrained linear classifier $\boldsymbol{w}_{src}$ to a target dataset $\{(\boldsymbol{x}_i, y_i)\}$, $\boldsymbol{w}_{src}$ is preserved in all directions orthogonal to the span of the $\boldsymbol{x}_i$.

Now what happens if we add ℓ_2 regularization (i.e., weight decay) to the logistic regression problem? As can be easily checked, the gradient updated of the logistic loss now becomes

$$\begin{aligned}\nabla \ell_{\boldsymbol{w}}(\boldsymbol{x}_i, y_i) &= (\sigma(\boldsymbol{w}^\top \boldsymbol{x}_i) - y_i) \cdot \boldsymbol{x}_i + \lambda \boldsymbol{w}, \\ &= (\sigma(\boldsymbol{w}_S^\top \boldsymbol{x}_i) - y_i) \cdot \boldsymbol{x}_i + \lambda(\boldsymbol{w}_S + \boldsymbol{w}_{S'})\end{aligned} \tag{G.1}$$

where λ is the regularization strength, $\boldsymbol{w}_S$ and $\boldsymbol{w}_{S'}$ are the projections of $\boldsymbol{w}$ on the span of the target datapoints $\boldsymbol{x}_i$'s, denoted S, and on its complementary subspace, denoted S'. This gradient, as before, restricts the space of updates to those in S. However due to regularization, this gradient drives $\boldsymbol{w}_{S'}$ to zero. Therefore, any planted bias in S' disappears as this subspace collapses to zero with regularization.

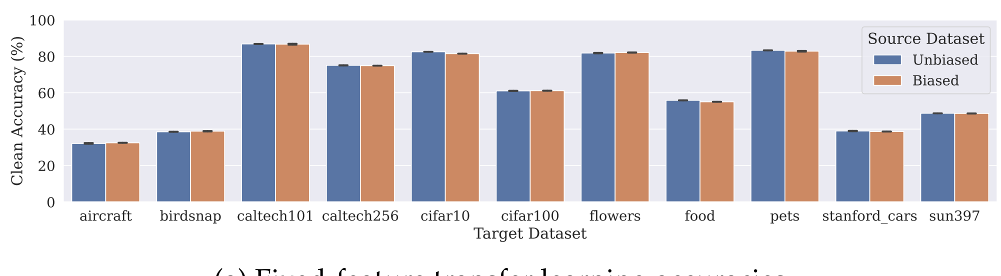
Indeed, we observe in practice that as we increase weight decay in the full-network transfer learning regime, bias transfer decreases over various downstream tasks as shown in Figure G.2. On the other hand, we find that weight decay does not reduce bias transfer in the fixed feature transfer learning regime, where the weights of the pretrained model are frozen.

G.1.6 Clean accuracies for experiments of Section 7.2

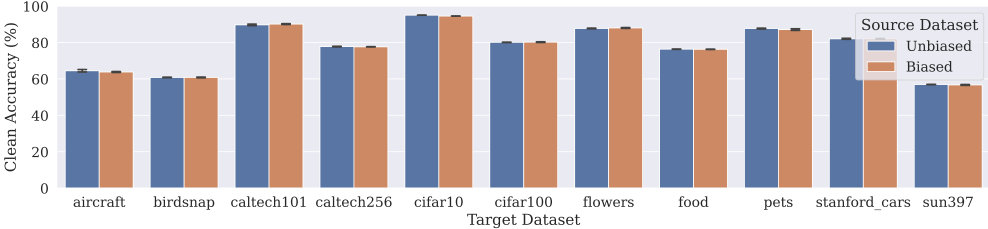
In Figure G.4, we report the clean accuracies of the transferred models that we use Section 7.2 on various target datasets. Note how the accuracies of both models pretrained in biased and unbiased source models, for both fixed-feature and full-network settings, are roughly the same. So the discrepancy in ASR reported in this chapter is solely due to bias transfer.

G.1.7 Comparison with models trained from scratch (Additional results to Section 7.2)

In this section, we add an extra baseline to Figure 7.3a where we train models from scratch on the various target datasets to check if the yellow square bias already exists in these datasets. In Figure G.5a, we plot the accuracies of all the models across all target tasks. Note that since there is a significant difference between the accuracies of the models trained from scratch and those finetuned, ASR is no longer an informative metric to capture the existence of bias. Thus, we measure the change in accuracy after adding the backdoor trigger and report the results in Figure G.5b. Indeed, the addition of the yellow



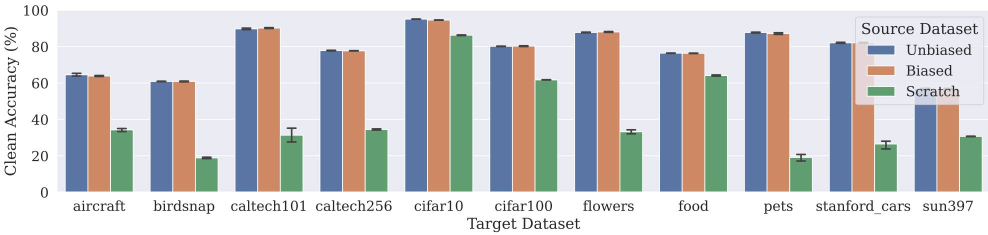
(a) Fixed-feature transfer learning accuracies.



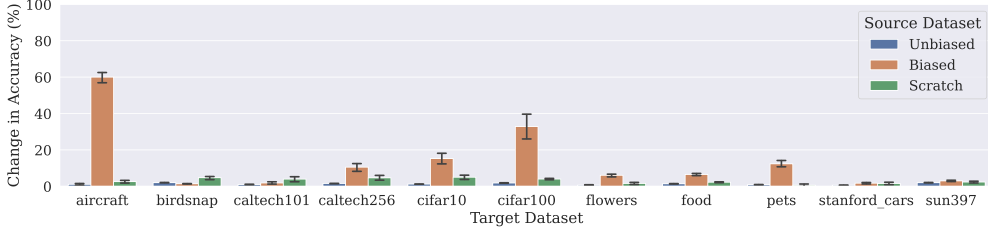
(b) Full-network transfer learning accuracies.

Figure G.4: Clean accuracies of the fixed-feature and full-network experiments of Section 7.2.

square trigger do not significantly change the accuracy of the models trained from scratch reflecting no existing bias in the target datasets.



(a) Full-network transfer learning accuracies.



(b) Full-network transfer learning change in accuracy after adding the backdoor trigger.

Figure G.5: Additional baseline ("Scratch") for the experiment of Figure 7.3a.

G.1.8 MS-COCO

In this section, we provide experimental details for the experiment on MS-COCO in Section 7.3.1. We consider the binary task of predicting cats from dogs, where there is a strong correlation between dogs and the presence of people.

Dataset construction. We create two source datasets which are described in Table G.2.

Table G.2: The synthetic datasets we create from MS-COCO for the experiment in Section 7.3.1.

Dataset	*Class: Cat*		*Class: Dog*	
	With People	Without People	With People	Without People
Non-Spurious	0	1000	0	100
Spurious	1000	4000	4000	1000

We then fine-tune models trained on the above source datasets on new images of cats and dogs without people (485 each). We use the cats and dogs from the MS-COCO test set for evaluation.

Experimental details. We train a ResNet-18 with resolution 224×224. We use SGD with momentum, and a Cyclic learning rate. We use the following hyperparameters shown in Table G.3:

Table G.3: Hyperparameters used for training on the MS-COCO dataset.

Hyperparameter	Value for pre-training	Value for fine-tuning
Batch Size	256	256
Epochs	25	25
LR	0.01	0.005
Momentum	0.9	0.9
Weight Decay	0.00005	0.00005
Peak Epoch	2	2

G.1.9 CelebA

In this section, we provide experimental details for the CelebA experiments in Section 7.3.2. Here, the task was to distinguish old from young faces, in the presence of a spurious correlation with gender in the source dataset.

Dataset construction. We create two source datasets shown in Table G.4:

Table G.4: The synthetic source datasets we create from CelebA for the experiment in Section 7.3.2.

Dataset	*Class: Young*		*Class: Old*	
	Male	Female	Male	Female
Non-Spurious	2500	2500	2500	2500
Spurious	1000	4000	4000	1000

Due to imbalances in the spurious dataset, the model trained on this dataset struggles on faces of young males and old females. We then fine-tune the source models on the following target datasets (see Table G.5), the images of which are disjoint from that in the source dataset.

Table G.5: The synthetic target datasets we create from CelebA for the experiment in Section 7.3.2.

Dataset	*Class: Young*		*Class: Old*	
	Male	Female	Male	Female
Only Women	0	5000	0	5000
80% Women|20% Men	1000	4000	1000	4000
50% Women|50% Men	2500	2500	2500	2500

Due to space constraints, we plotted the results of fixed-feature transfer setting on Only Women and 80% Women|20% Men in the main body of this chapter. Below, we display the results for fixed-feature and full-network transfer settings on all 3 target datasets.

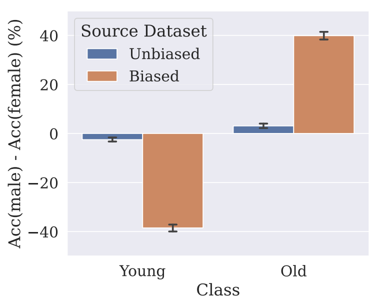
Experimental details. We train a ResNet-18 with resolution 224×224. We use SGD with momentum, and a cyclic learning rate. We use the following hyperparameters shown in Table G.6:

Table G.6: Hyperparameters used for training on the CelebA datasets.

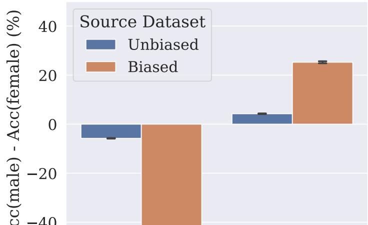
Batch Size	Epochs	LR	Momentum	Weight Decay	Peak Epoch
1024	20	0.05	0.9	0.01	5

Results. We find that in both the fixed-feature and full-network transfer settings, the gender correlation transfers from the source model to the transfer model, even though

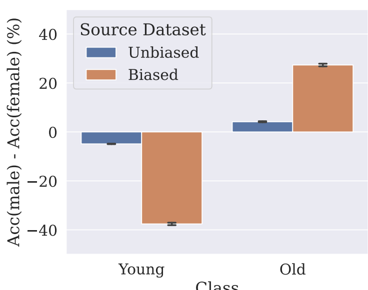
the target task is itself gender balanced as shown in Figure G.6. As the proportion of men and women in the target dataset change, the model is either more sensitive to the presence of women, or more sensitive to the presence of men. In all cases, however, the model transferred from the spurious backbone is more sensitive to gender than a model transferred from the non-spurious backbone.



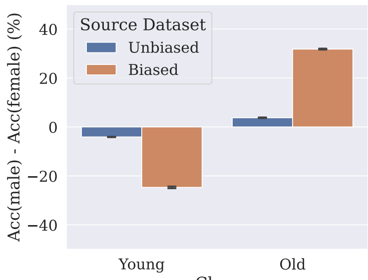
(a) Source Model Accuracy



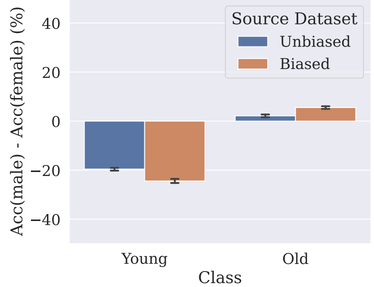
(b) Fixed-Feature transfer. Target Dataset: Only Women



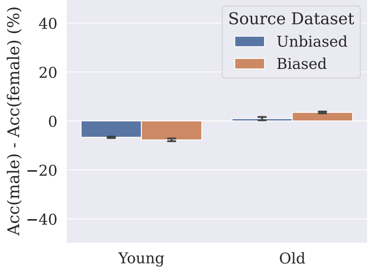
(c) Fixed-Feature transfer. Target Dataset: 80% Women, 20% Men



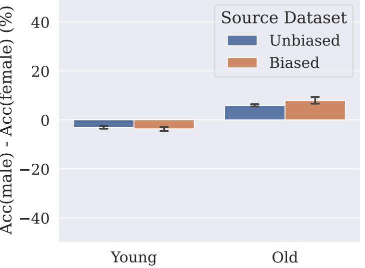
(d) Fixed-Feature transfer. Target Dataset: 50% Women, 50% Men



(e) Full-network transfer. Target Dataset: Only Women



(f) Full-network transfer. Target Dataset: 80% Women, 20% Men



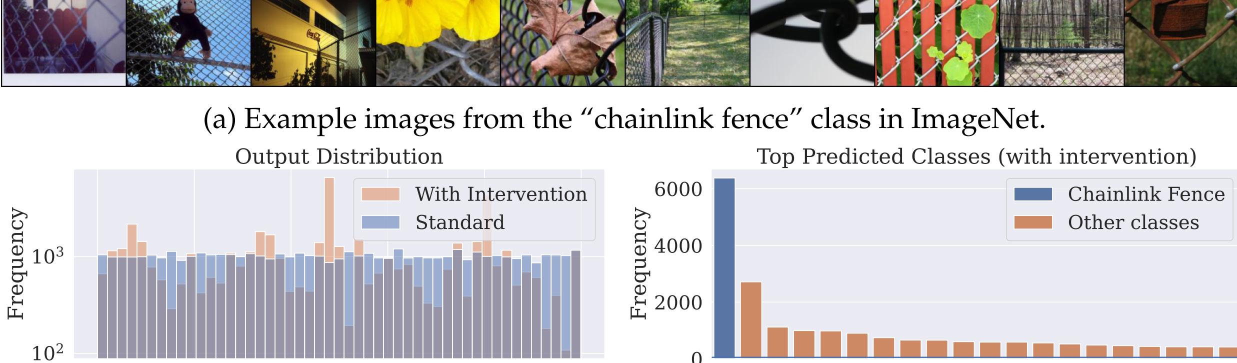
(g) Full-network transfer. Target Dataset: 50% Women, 50% Men

Figure G.6: **CelebA Experiment.** We consider transfer from a source dataset that spuriously correlate age with gender — such that old men and young women are overrepresented. We plot the difference in accuracies between male and female examples, and find that the model transferred from a spurious backbone is sensitive to gender, even though the target dataset was itself gender balanced.

G.2 ImageNet Biases

G.2.1 Chainlink fence bias.

In this section we show the results for the “chainlink fence” bias transfer. We first demonstrate in Figure G.7 that the “chainlink fence” bias actually exists in ImageNet. Then in Figures G.8, G.9, G.10, and G.11, we show the output distribution—after applying a chainlink fence intervention—of models trained on various datasets either from scratch, or by transferring from the ImageNet model. The from-scratch models are not affected by the chainlink fence intervention, while the ones learned via transfer have highly skewed output distributions.

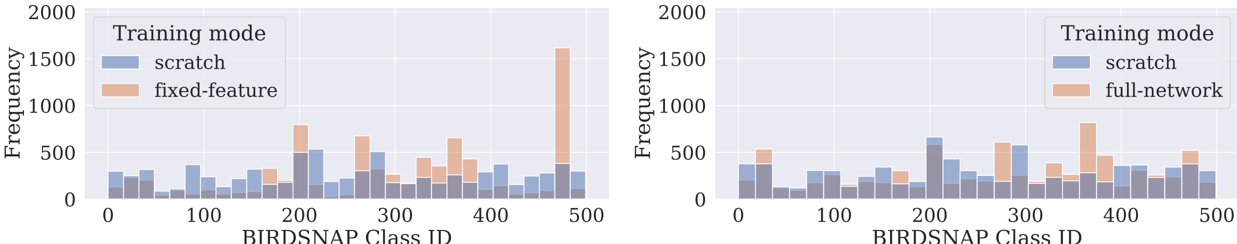


(a) Example images from the “chainlink fence” class in ImageNet.

(b) Shift in ImageNet predicted class distribution after adding a “chainlink fence” intervention, establishing that the bias holds for the source model.

Figure G.7: The **chainlink fence** bias in ImageNet.

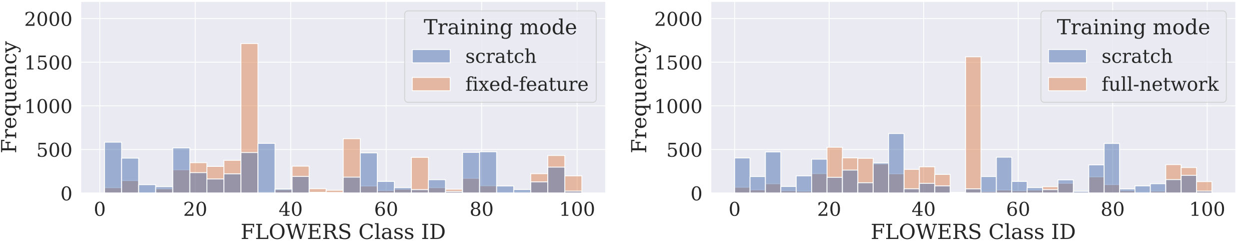
(a) Example Birdsnap images after applying the chain-link fence intervention.



(b) Output distribution of Birdsnap models with a chainlink fence intervention.

Figure G.8: The **chainlink fence** bias transfers to *Birdsnap*.

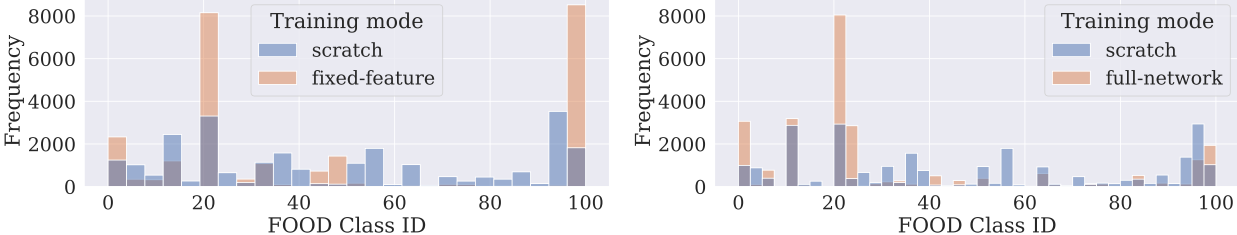
(a) Example Flowers images after applying the chain-link fence intervention.



(b) Output distribution of Flowers models with a chainlink fence intervention.

Figure G.9: The **chainlink fence** bias transfers to *Flowers*.

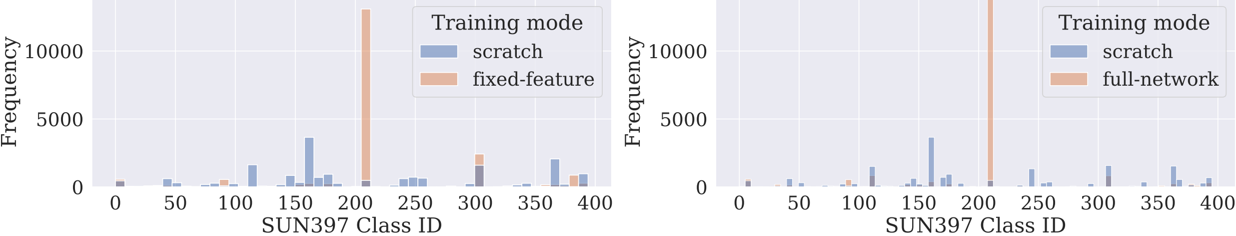
(a) Example Food images after applying the chain-link fence intervention.



(b) Output distribution of Food models with a chainlink fence intervention.

Figure G.10: The **chainlink fence** bias transfers to *Food*.

(a) Example SUN397 images after applying the chain-link fence intervention.



(b) Output distribution of SUN397 models with a chainlink fence intervention.

Figure G.11: The **chainlink fence** bias transfers to *SUN397*.

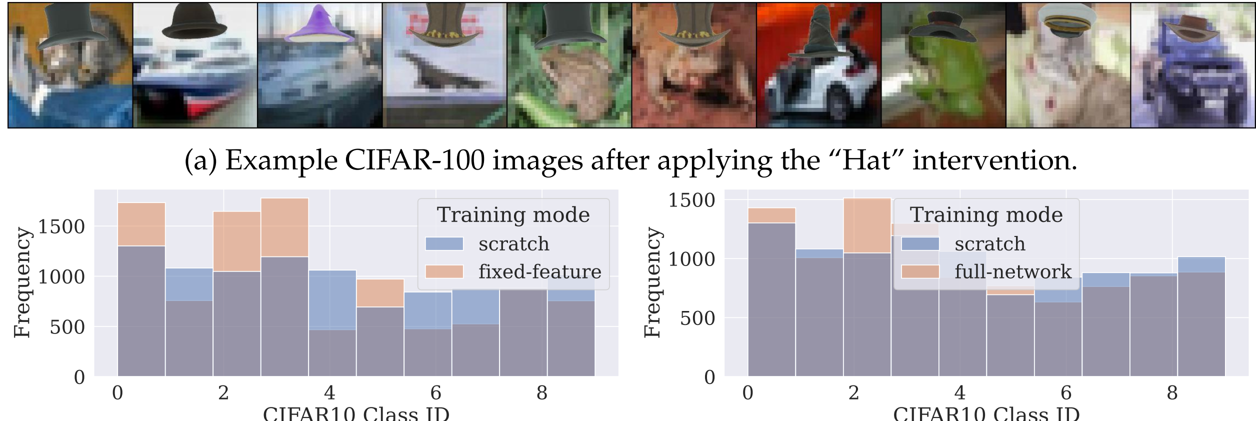
G.2.2 Hat bias.

In this section we show the results for the “Hat” bias transfer. We first demonstrate in Figure G.12 that the “Hat” bias actually exists in ImageNet (shifts predictions to the “Cowboy hat” class). Then in Figure G.13, we show the output distribution—after applying a hat intervention—of models trained on CIFAR-10 either from scratch, or by transferring from the ImageNet model. The from-scratch model is not affected by the hat intervention, while the one learned via transfer have highly skewed output distributions.

(a) ImageNet images from the class “Cowboy hat”.

(b) ImageNet distribution shift after intervention.

Figure G.12: The **hat** bias in ImageNet.



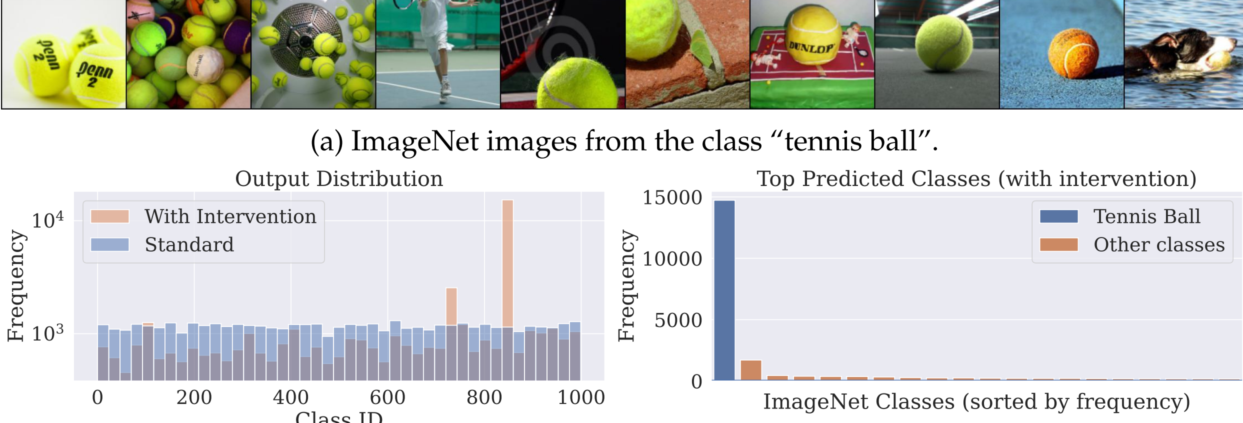
(a) Example CIFAR-100 images after applying the “Hat” intervention.

(b) Output distribution of CIFAR-10 models with the Hat intervention.

Figure G.13: The **hat** bias transfers to *CIFAR-10*.

G.2.3 Tennis ball bias.

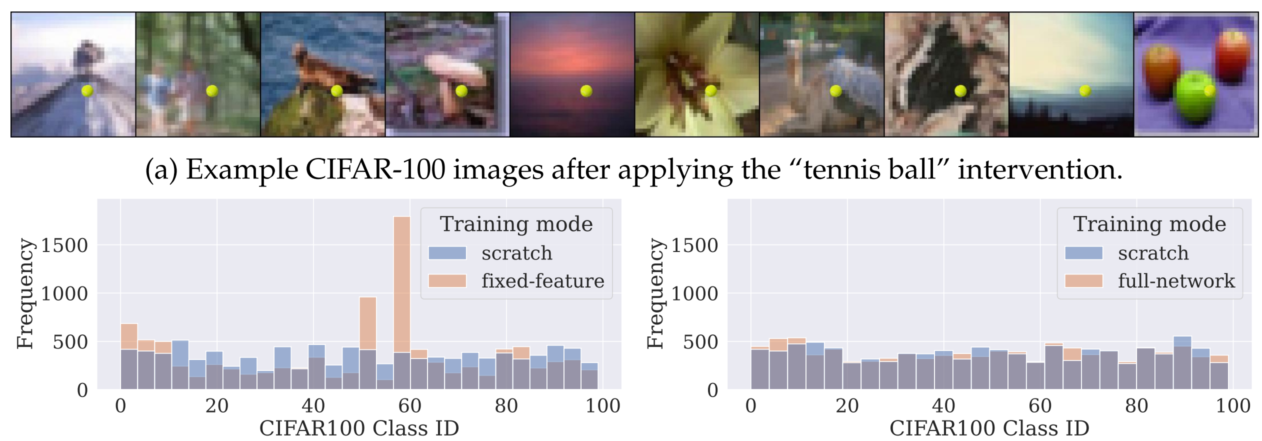
In this section we show the results for the "tennis ball" bias transfer. We first demonstrate in Figure G.14 that the "tennis ball" bias actually exists in ImageNet. Then in Figures G.15, G.16, G.17, and G.18, we show the output distribution—after applying a tennis ball intervention—of models trained on various datasets either from scratch, or by transferring from the ImageNet model. The from-scratch models are not affected by the tennis ball intervention, while the ones learned via transfer have highly skewed output distributions.



(a) ImageNet images from the class "tennis ball".

(b) ImageNet distribution shift after intervention.

Figure G.14: The **tennis ball** bias in ImageNet.

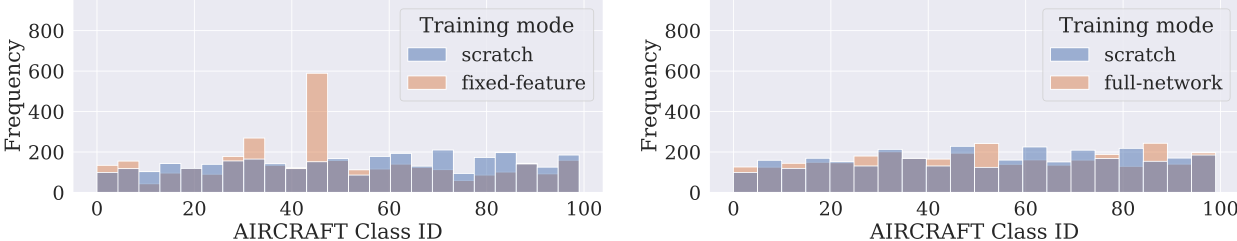


(a) Example CIFAR-100 images after applying the "tennis ball" intervention.

(b) Output distribution of CIFAR-100 models with the tennis ball intervention.

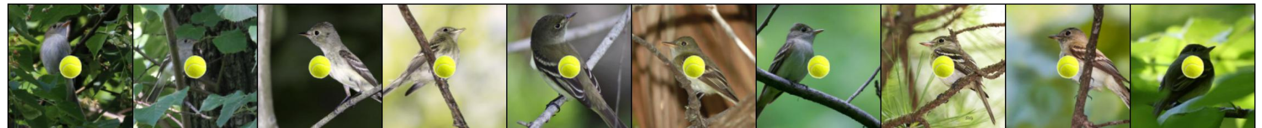
Figure G.15: The **tennis ball** bias transfers to *CIFAR-100*.

(a) Example Aircraft images after applying the "tennis ball" intervention.

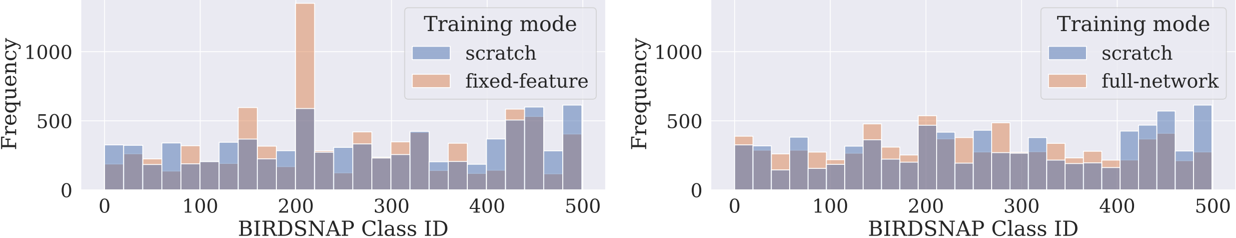


(b) Output distribution of Aircraft models with the tennis ball intervention.

Figure G.16: The **tennis ball** bias transfers to *Aircraft*.



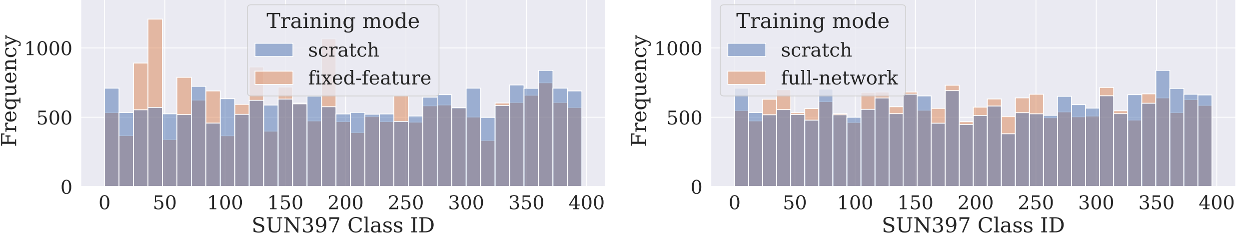
(a) Example Birdsnap after applying the "tennis ball" intervention.



(b) Output distribution of Birdsnap models with the tennis ball intervention.

Figure G.17: The **tennis ball** bias transfers to *Birdsnap*.

(a) Example sun397 after applying the "tennis ball" intervention.



(b) Output distribution of SUN397 models with the tennis ball intervention.

Figure G.18: The **tennis ball** bias transfers to *SUN397*.

Appendix H

Additional details for Chapter 8

H.1 Experimental Setup

H.1.1 ImageNet Models

In this chapter, we train a large number of models on various subsets of ImageNet in order to estimate the influence of each class of ImageNet on the model's transfer performance for multiple downstream tasks. We focus on the ResNet-18 architecture from PyTorch's official implementation found here `https://pytorch.org/vision/stable/models.html`[1].

Training details. We fix the training procedure for all of our models. Specifically, we train our models from scratch using SGD to minimize the standard cross-entropy multi-class classification loss. We use a batch size of 1024, momentum of 0.9, and weight decay of 5×10^{-4}. The models are trained for 16 epochs using a Cyclic learning rate schedule with an initial learning rate of 0.5 and learning rate peak epoch of 2. We use standard data-augmentation: *RandomResizedCrop* and *RandomHorizontalFlip* during training, and *RandomResizedCrop* during testing. Our implementation and configuration files are attached to the submission.

H.1.2 ImageNet transfer to classification datasets

Datasets. We consider the transfer image classification tasks that are used in [SIE+20; KSL19], which vary in size and number of classes. See Table H.1 for the details of these datasets. We consider two transfer learning settings for each dataset: *fixed-feature* and *full-network* transfer learning.

[1] Our framework is agnostic to the choice of the model's architecture.

Dataset	Classes	Train Size	Test Size
Birdsnap [BLW+14]	500	32,677	8,171
Caltech-101 [FFP04]	101	3,030	5,647
Caltech-256 [GHP07]	257	15,420	15,187
CIFAR-10 [Kri09]	10	50,000	10,000
CIFAR-100 [Kri09]	100	50,000	10,000
FGVC Aircraft [MRK+13]	100	6,667	3,333
Food-101 [BGV14]	101	75,750	25,250
Oxford 102 Flowers [NZ08]	102	2,040	6,149
Oxford-IIIT Pets [PVZ+12]	37	3,680	3,669
SUN397 [XHE+10]	397	19,850	19,850
Stanford Cars [KDS+13]	196	8,144	8,041

Table H.1: Image classification datasets used in this chapter.

Fixed-feature transfer. For this setting, we *freeze* the layers of the ImageNet source model[2], except for the last layer, which we replace with a random initialized linear layer whose output matches the number of classes in the transfer dataset. We now train only this new layer for using SGD, with a batch size of 1024 using cyclic learning rate.

Full-network transfer. For this setting, we *do not freeze* any of the layers of the ImageNet source model, and all the model weights are updated. We follow the exact same hyperparameters as the fixed-feature setting.

H.1.3 Compute and training time.

We leveraged the FFCV data-loading library for fast training of the ImageNet models [LIE+22][3]. Our experiments were run on two GPU clusters: an A100 and a V100 cluster.

Training ImageNet models and influence calculation. We trained 7,540 ImageNet models on random subsets of ImageNet, each containing half of ImageNet classes. On a single V100, training a single model takes around 30 minutes. After training these ImageNet models, we compute the influence of each class as outlined in Algorithm 1. Computing the influences is fast, and takes few seconds on a single V100 GPU.

[2]For all of our experiments, we do not freeze the batch norm statistics. We only freeze the weights of the model, similar to Salman et al. [SIE+20].

[3]Using FFCV, we can train a model on the ImageNet dataset in around 1 hour, and reach ~63% accuracy

H.1.4 Handpicked baseline details

In our counterfactual experiments in Section 8.2, we automatically selected, via our framework, the most influential subsets of ImageNet classes for various downstream tasks. We then removed the classes that are detrimental to the transfer performance, and measured the transfer accuracy improvement after removing these classes. The results are summarized in Table 8.2b.

What happens if we hand-pick the source dataset classes that are relevant to the target dataset? Indeed, Ngiam et al. [NPV+18] found that hand-picking the source dataset classes can sometimes boost transfer performance. We validate this approach for our setting using the WordNet hierarchy [Mil95]. Specifically, for each class from the target task, we look up all the ImageNet classes that are either children or parents of this target class. The set of all such ImageNet classes are used as the handpicked most influential classes. Following this manual selection, we train an ImageNet model on these classes, then apply transfer learning to get the baseline performance that we report in Table 8.2b.

H.1.5 Convergence Analysis

We compute our class influence values using 7,540 source models, each of which were trained using 500 randomly chosen ImageNet classes. How many models do we actually need to compute our transfer influences?

Counterfactual Experiment To first analyze this question, we re-run our counterfactual experiment in Section 8.2 when using a smaller number of models to compute transfer influences (Figure H.1). While using the full number of models performs the best, we get meaningful transfer influences when computing with both 4000 and 1000 models. In both cases, removing the most negatively influential classes boosts transfer learning performance over using the entire source dataset, while removing the most positively influential classes drops transfer learning performance over the random baseline.

Bootstrap Analysis In order to analyze the convergence of the transfer influences, we track the standard deviation of the influence values after bootstrap resampling.

We consider the ImageNet $\rightarrow$ CIFAR-10 transfer setting with fixed-feature fine-tuning. Given N models, we randomly sample, with replacement, N models to recompute our transfer influences. Specifically, we evaluate the overall transfer influences (i.e., the influence value of each ImageNet class averaged over all target examples). We perform

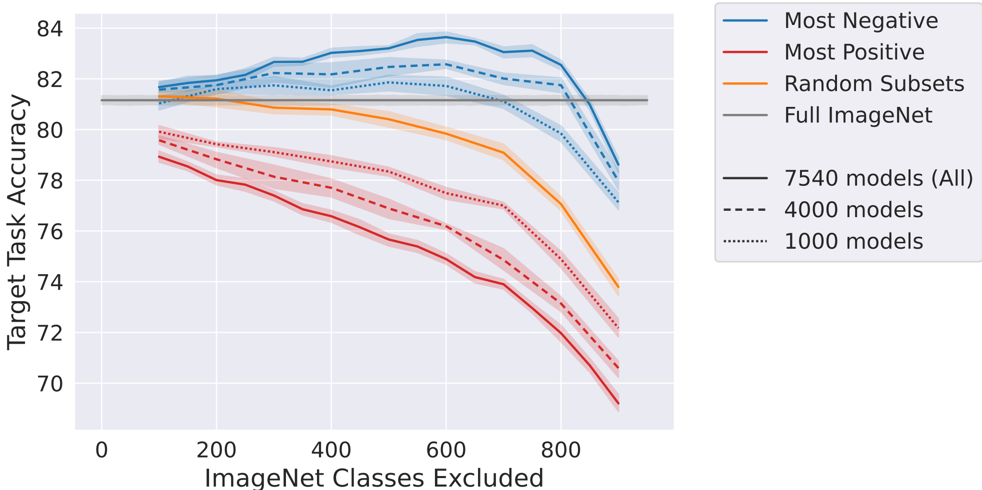


Figure H.1: We replicate the counterfactual experiment in Section 8.2 Figure 8.2a using 1000 and 4000 source models for computing the transfer influences.

this resampling 500 times, and measure the standard deviation of the computed overall transfer influence value for each class over these 500 resamples.

Below, we plot this standard deviation (averaged over the 1000 classes) for various number of models N. We find that the standard deviation goes down as more models are used, indicating that our estimate of the influence values has less variance. This metric roughly plateaus by the time we are using 7000 models.

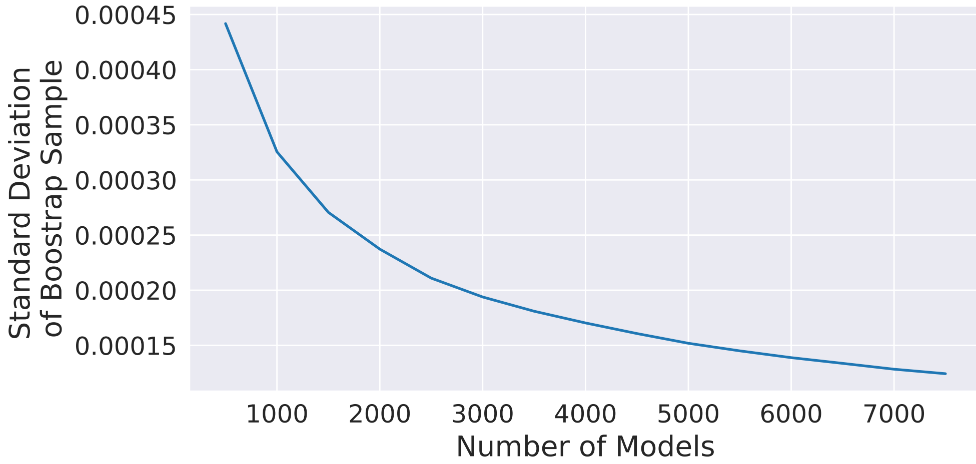


Figure H.2: Standard deviation of the overall influence values (averaged over classes) after bootstrap resampling for various numbers of models.

H.2 Variants of Computing Influences

H.2.1 Variations of targets for computing transfer influences

In this chapter, we used the softmax output of the ground truth class as the target for our influence calculation. What happens if we use a different target? We compare using the following types of targets.

- Softmax Logits: the softmax output of the ground truth class
- Is Correct: the binary value of whether the image was predicted correctly
- Raw Margins: the difference in raw output between the correct class and the most confidently predicted incorrect class
- Softmax Margins: the same as raw margins, but use the output after softmax

In Figure H.3, we replicate the counterfactual experiment from this chapter in Figure 8.2b using these different targets. Specifically (over 2 runs), we rank the overall influence of the ImageNet classes on CIFAR-10 for fixed-feature transfer. We then remove the classes in order most most or least influence.

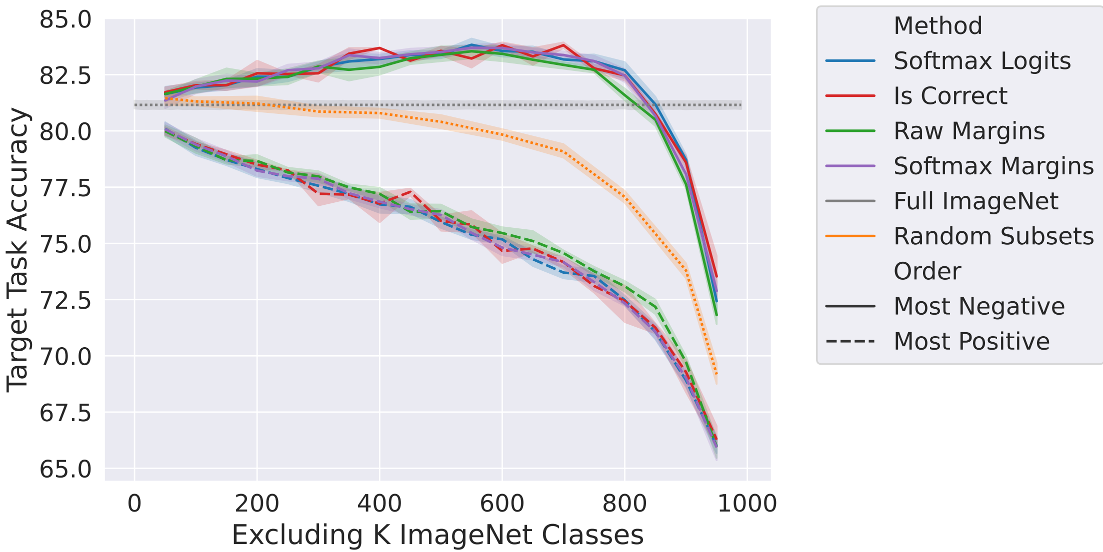


Figure H.3: Target task accuracies after removing the most positively or negatively influential ImageNet classes from the source dataset with various influence targets.

We find that our method is not particularly sensitive to the individual target used. We found that using the softmax logits provided the highest benefit when removing negative influencers, and thus used that target for the reset of our experiments.

Datamodels vs. Influences. Datamodels [IPE+22] is another method that, similar to influences, seeks to compute the importance of a training point on a test set prediction. Specifically, instead of computing the difference in the expected accuracy of the model when a training point is removed, the method fits a linear model that, given a binary vector that denotes the composition of the training dataset, predicts the raw margin (i.e., the difference in raw output between the correct class and the most confidently predicted incorrect class). The importance of each training point is then the coefficient of the linear model for that particular example.

We adapt this method to our framework by training a linear model with ridge regression to predict the softmax output of the transfer model on the target images given a binary vector that denotes which source classes were included in the source dataset. However, we find that datamodels were more effective for computing example-based values (see Appendix H.4).

In Figure H.4, we compare using influence values (as described in this chapter) to using these adapted datamodels. Specifically (over 5 runs), we rank the overall importance of the ImageNet classes on CIFAR-10 for fixed-feature transfer using influences or datamodels. We then remove the classes in order of most or least influence. We find that our framework is not sensitive to the choice of datamodels or influences. However, influences performed marginally better in this counterfactual experiment, so we used influences for all other experiments in this chapter.

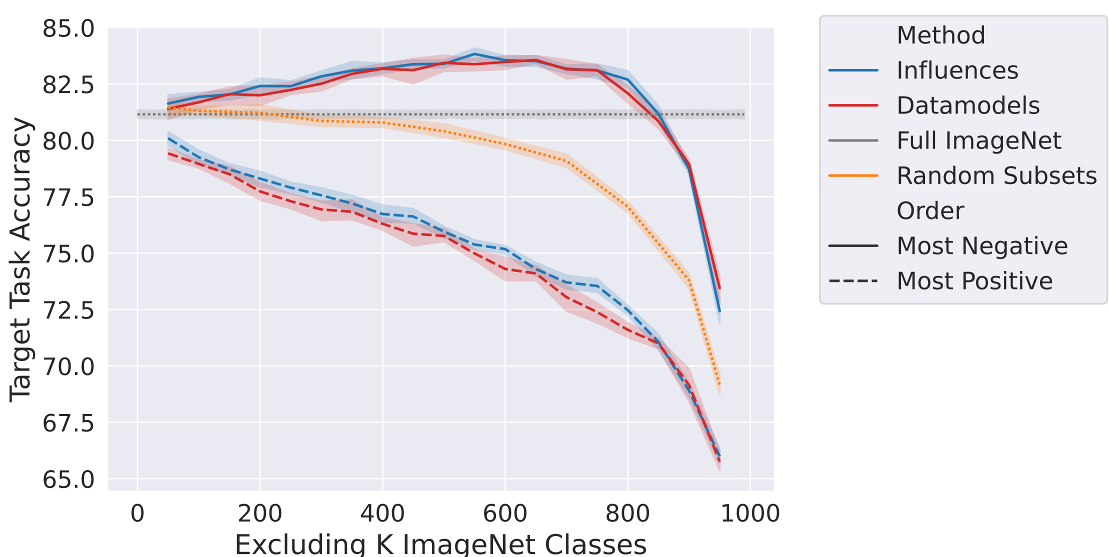
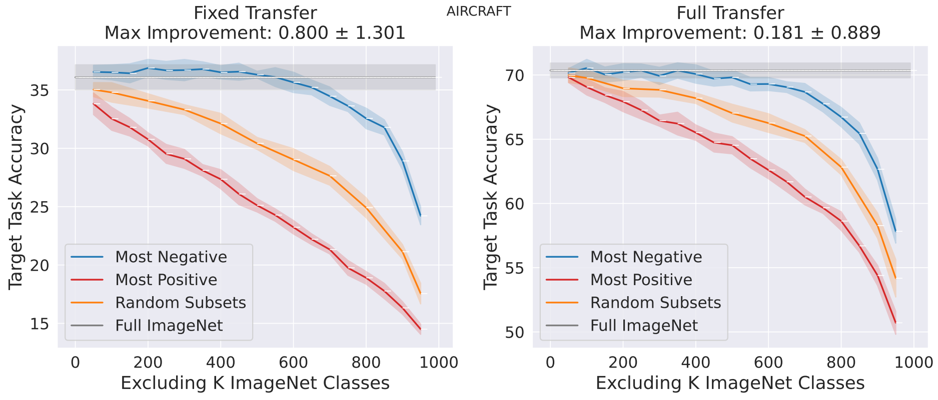


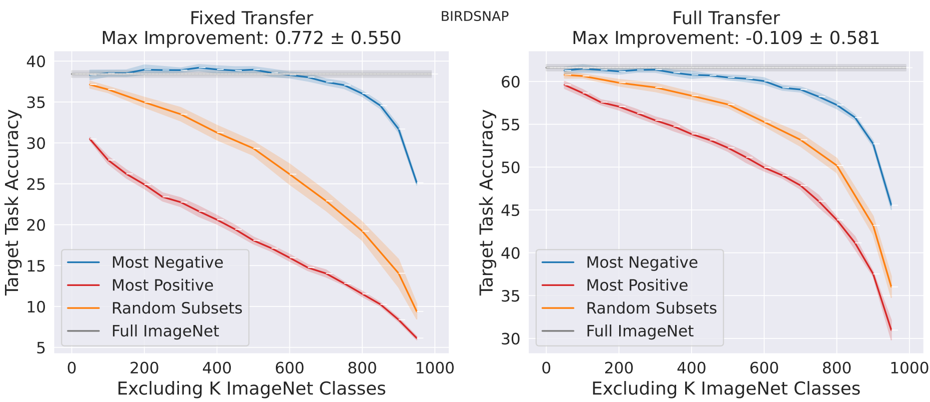
Figure H.4: Target task accuracies after removing the most positively or negatively influential ImageNet classes from the source dataset using datamodels or influences.

H.3 Full Counterfactual Experiment

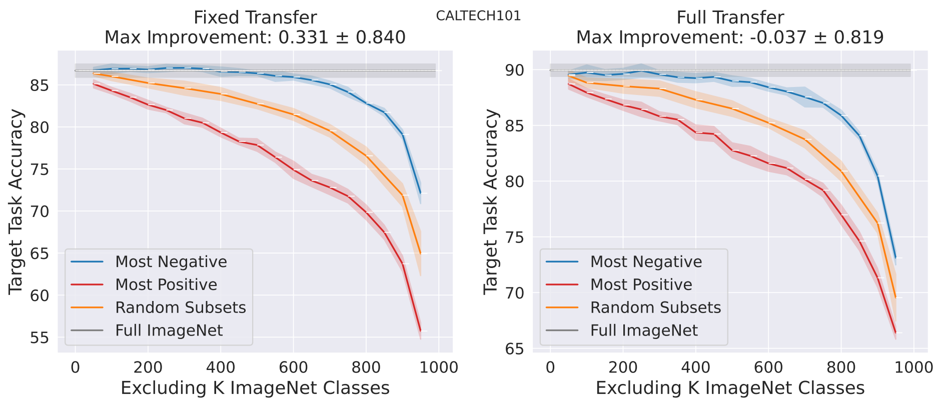
In this section, we display the full results for the counterfactual experiment in this chapter (Figure 8.2b). Specifically, for each target task, we display the target task accuracies after removing the most positive (top) and negative (bottom) influencers from the dataset over 10 runs. We find that our results hold across datasets.



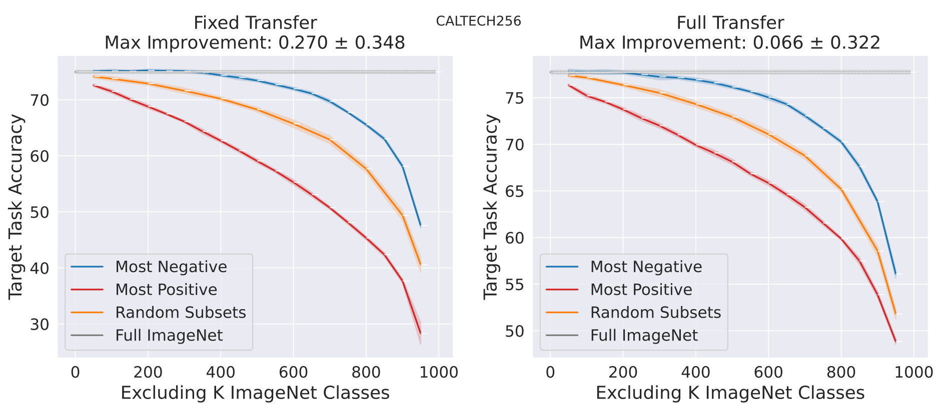
Aircraft



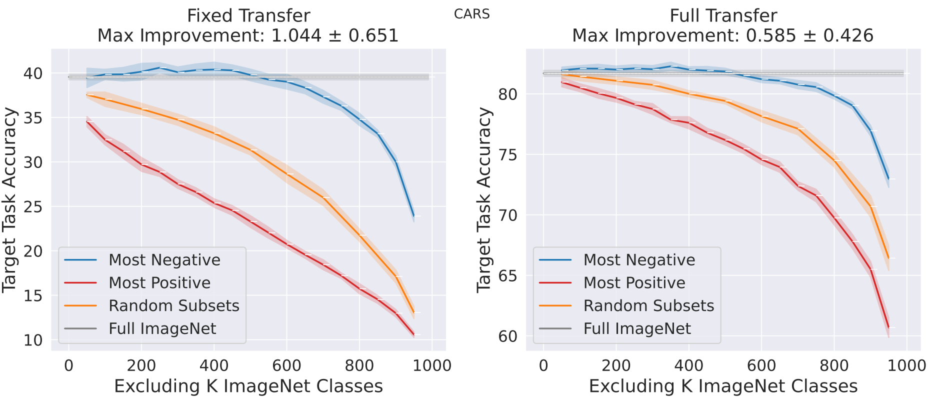
Birdsnap



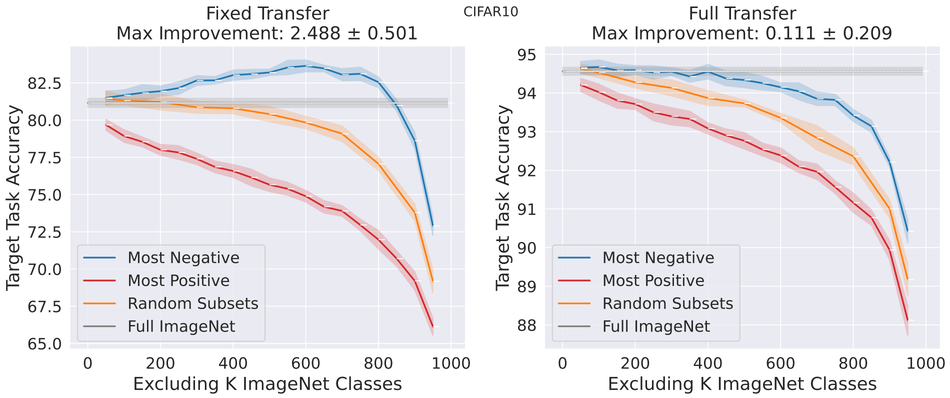
Caltech101



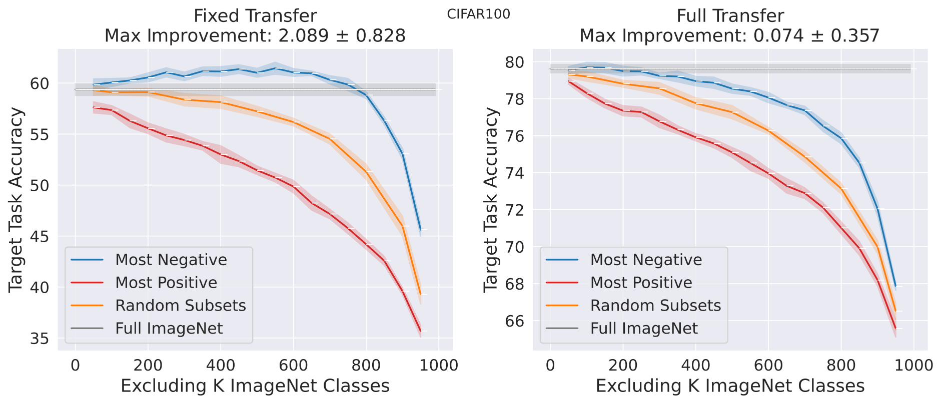
Caltech256



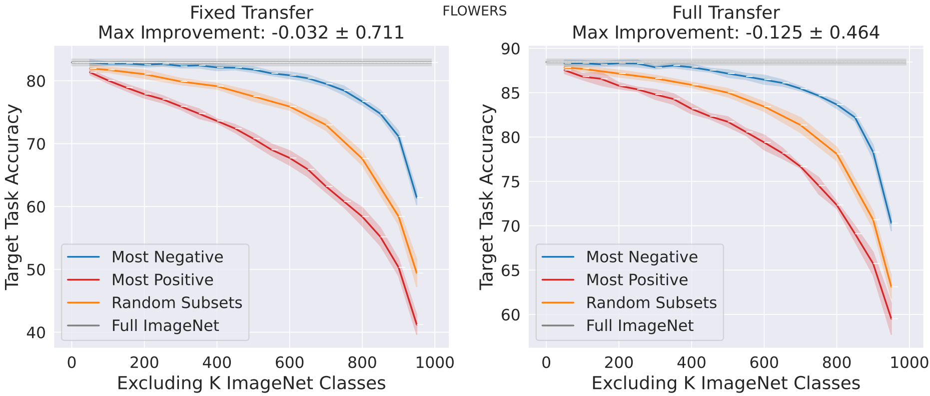
Cars



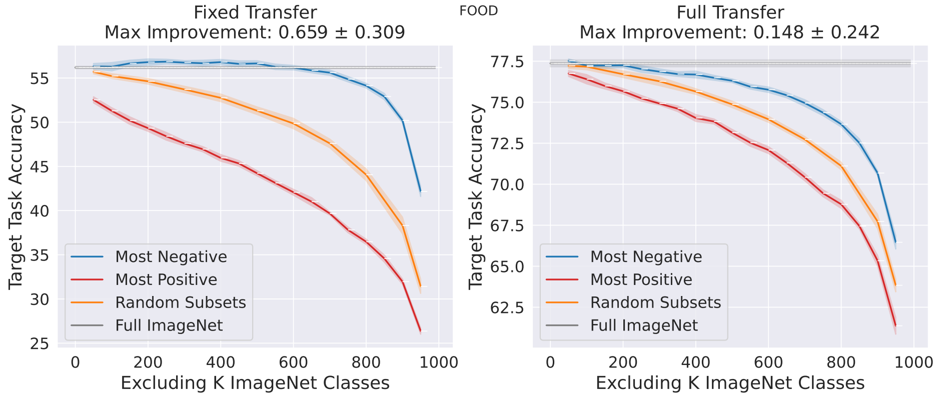
CIFAR-10



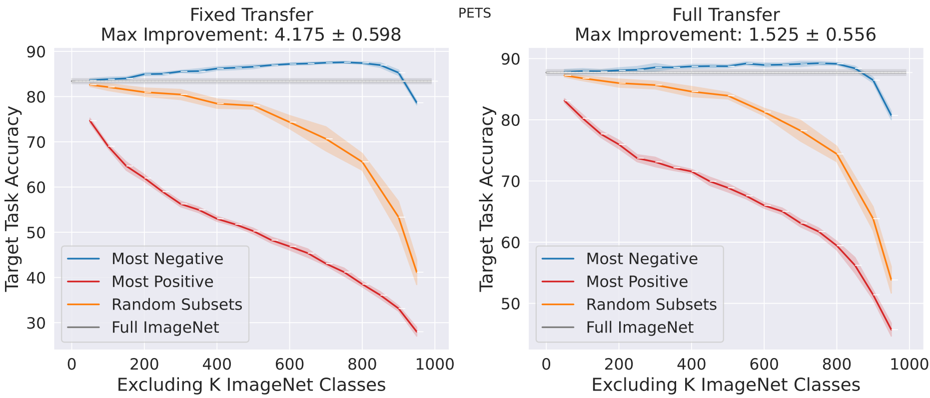
CIFAR-100



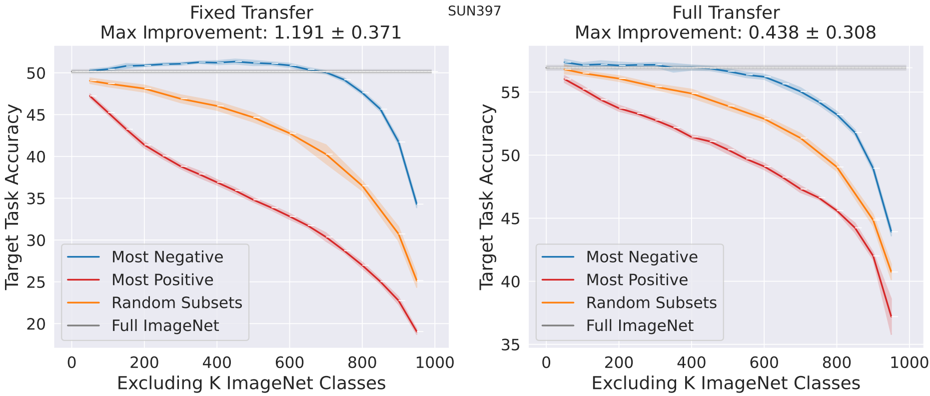
Flowers



Food



Pets



SUN397

H.4 Adapting our Framework to Compute the Effect of Every Source Datapoint on Transfer Learning

We have presented in this chapter how to compute the influences of every class in the source dataset on the predictions of the model on the target dataset. In that setup, we demonstrated multiple capabilities of our framework, such as improving overall transfer performance, detecting particular subpopulations in the target dataset, etc. Given the wide range of capabilities class-based influences provide, one natural question that arises: Can we compute the influence of every source datapoint on the predictions of the model on the target dataset? Furthermore, what do these influences tell us about the transfer learning process?

Mathematically, the computation of example-based influences (i.e., the influence of every source datapoint) is very similar to the computation of class-based influences. Specifically, to compute example-based influences, we start by training a large number of models on different subsets of the source datapoints (as opposed to source classes for class-based influences). Next, we estimate the influence value of a source datapoint s on a target example t as the expected difference in the transfer model's performance on example t when datapoint s was either included or excluded from the source dataset:

$$\text{Infl}[s \to t] = \mathbb{E}_S \left[f(t; S) \mid s \in S \right] - \mathbb{E}_S \left[f(t; S) \mid s \notin S \right] \tag{H.1}$$

where $f(t; S)$ is the softmax output of a model trained on a subset S of the source dataset. Similar to class-based influences, a positive (resp. negative) influence value indicates that including the source datapoint s improves (resp. hurts) the model's performance on the target example t.

While example-based influences provide some insights about the transfer process, we found that—in this regime—datamodels [IPE+22] provide cleaner results and better insights. Generally, influences and datamodels measure similar properties: the effect of the source datapoints on the target datapoints. For a particular target datapoint t, we measure the effect of every source datapoint s with datamodels by solving a regression problem. Specifically, we train a large number of models on different subsets of the source dataset. For every model f_i, we record 1) a binary mask $\mathbb{1}_{\mathcal{S}_i}$ that indicates which source datapoints were included in the subset $\mathcal{S}_i$ of the source dataset, and 2) the transfer performance $f_i(t; \mathcal{S}_i)$ of the model f_i on the target datapoint t after fine-tuning on the target dataset. Following the training and the fine-tuning stages, we fit a linear model g_w that predicts the transfer performance $f(t; \mathcal{S})$ from a random subset $\mathcal{S}$ of the source dataset as

follows: $f(t;\mathcal{S}) \approx g_w(\mathbb{1}_{\mathcal{S}}) = w^\top \mathbb{1}_{\mathcal{S}}$. Given this framework, $w = (w_1, w_2, \ldots, w_L)$ measures the effect of every source datapoint s on the target datapoint t[4]. We present the overall procedure in Algorithm 6.

Algorithm 6 Example-based datamodels estimation for transfer learning.

Require: source dataset $\mathcal{S} = \cup_{l=1}^{L} s_l$ (with L datapoints), a target dataset $\mathcal{T} = (t_1, t_2, \cdots, t_n)$, training algorithm $\mathcal{A}$, subset ratio α, number of models m

1: Sample m random subsets $S_1, S_2, \cdots, S_m \subset \mathcal{S}$ of size $\alpha \cdot |\mathcal{S}|$:
2: **for** $i \in 1$ to m **do**
3:     Train model f_i by running algorithm $\mathcal{A}$ on S_i
4: **end for**
5: Fine-tune f_i on the training target dataset
6: **for** $j \in 1$ to n **do**
7:     Collect datamodels training set $\mathcal{D}_j = \{(\mathbb{1}_{\mathcal{S}_i}, f_i(t_j, \mathcal{S}_i))\}_{i=1}^{m}$
8:     Compute w_j by fitting LASSO on $\mathcal{D}_j$
9: **end for**
10: **return** $w_j \ \forall \ j \in [n]$

---

[4]To estimate the datamodels, we train 71,828 models on different subsets of the source dataset.

H.5 Omitted Results

H.5.1 Per-class influencers

We display for the ImageNet → CIFAR-10 the top (most positive) and bottom (most negative) influencing classes for each CIFAR-10 class. This is the equivalent to the plot in Figure 8.3.

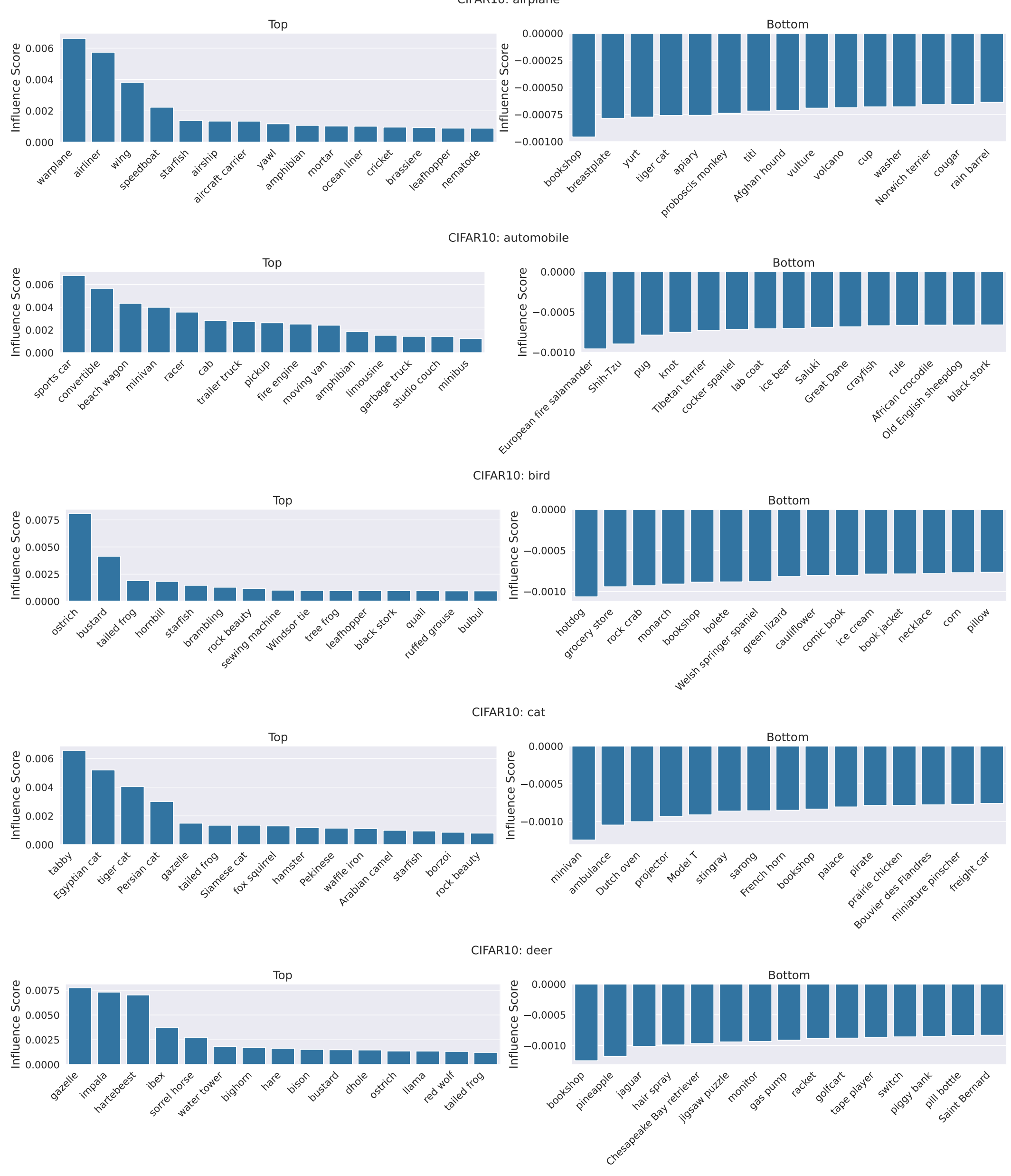


Figure H.5: Top and bottom influencing ImageNet classes for all CIFAR-10 classes.

Figure H.6: Top and bottom influencing ImageNet classes for all CIFAR-10 classes.

H.5.2 More examples of extracted subpopulations from the target dataset

Here, we depict more examples of extracting subpopulations from the target dataset (as in Figure 8.4).

Figure H.7: For each ImageNet class, we show the CIFAR-10 examples which were most positively influenced by that ImageNet class.

H.5.3 More examples of transfer of shape and texture feature

We depict more examples of ImageNet influencers which transfer shape or texture features (as in Figure 8.5).

(a) Indigo Bunting

(b) Gondola

(c) Tree Frog

Figure H.8: For each ImageNet class, we show the CIFAR-10 examples which were most highly influenced by that ImageNet class.

H.5.4 More examples of debugging mistakes of transfer model

We display more examples of how our influences can be used to debug the mistakes of the transfer model, as presented in Figure 8.6. We find that, in most cases (Figure H.9a, H.9b, H.9c), removing the top negative influencer improves the model's performance on the particular image.

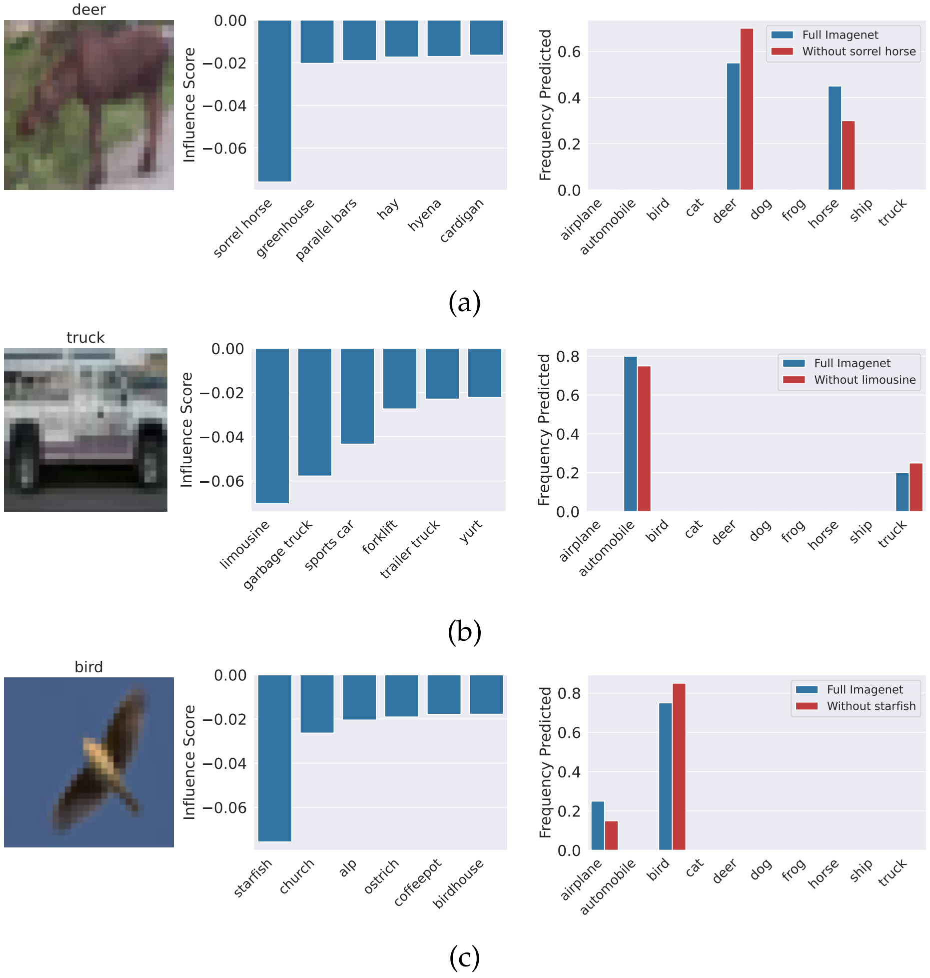


Figure H.9: More examples of debugging transfer mistakes through our framework (cf. Figure 8.6). For each CIFAR-10 image (**left**), we plot their most negative influencers (**middle**). On the **right**, we plot for each image the fraction (over 20 runs) of times that our transfer model predicts each class with and without the most negative influencer.

Quantitative analysis. How often does removing the most negative influencer actually improve the prediction on an image? For each of the following 14 classes, we run 20 runs of the ImageNet $\rightarrow$ CIFAR-10 fixed transfer pipeline while excluding that single class from the source dataset: ["sorrel horse", "limousine", "minivan", "fireboat", "ocean liner", "Arabian camel", "Persian cat", "ostrich", "gondola", "pool table", "starfish", "rapeseed", "tailed frog", "trailer truck"]. We compare against running the pipeline with 20 runs of the entire ImageNet dataset. Then, we look at individual CIFAR-10 images which were highly negatively influenced by one of these ImageNet classes, and check whether the images were predicted correctly more or less often when the top negative influencers were removed from the source dataset.

Of the 30 most negatively influenced ImageNet class/CIFAR-10 image pairs, 26 of them had the most negative ImageNet influencer in the above classes. Of those, 61.5% were predicted correctly more often when the negatively influential ImageNet class was removed, 34.6% were predicted incorrectly more often, and 3.9% were predicted correctly the same number of times.

We then examine the top 8 most influenced CIFAR-10 images for each of the above 14 ImageNet classes. Of those 112 images, 53% were predicted correctly more often when the image was removed, 34% were predicted incorrectly more often, and 14% were predicted correctly the same number of times.

We thus find that, for the most influenced CIFAR-10 images, removing the top negative influencer usually improves that specific image's prediction (even though we are removing training data from the source dataset).

H.5.5 Do Influences Transfer?

Transfer across datasets

In this section, we seek to understand how much task-specific information is in the transfer influences that we compute. To do so, we use the transfer influences computed for CIFAR-10 in order to perform the counterfactual experiments for other datasets. We find that while using the CIFAR-10 influence values for other target datasets is more meaningful than random, they do not provide the same boost in performance when removing bottom influencers as using the task-specific influences. We thus conclude that the influence values computed by our framework are relatively task-specific.

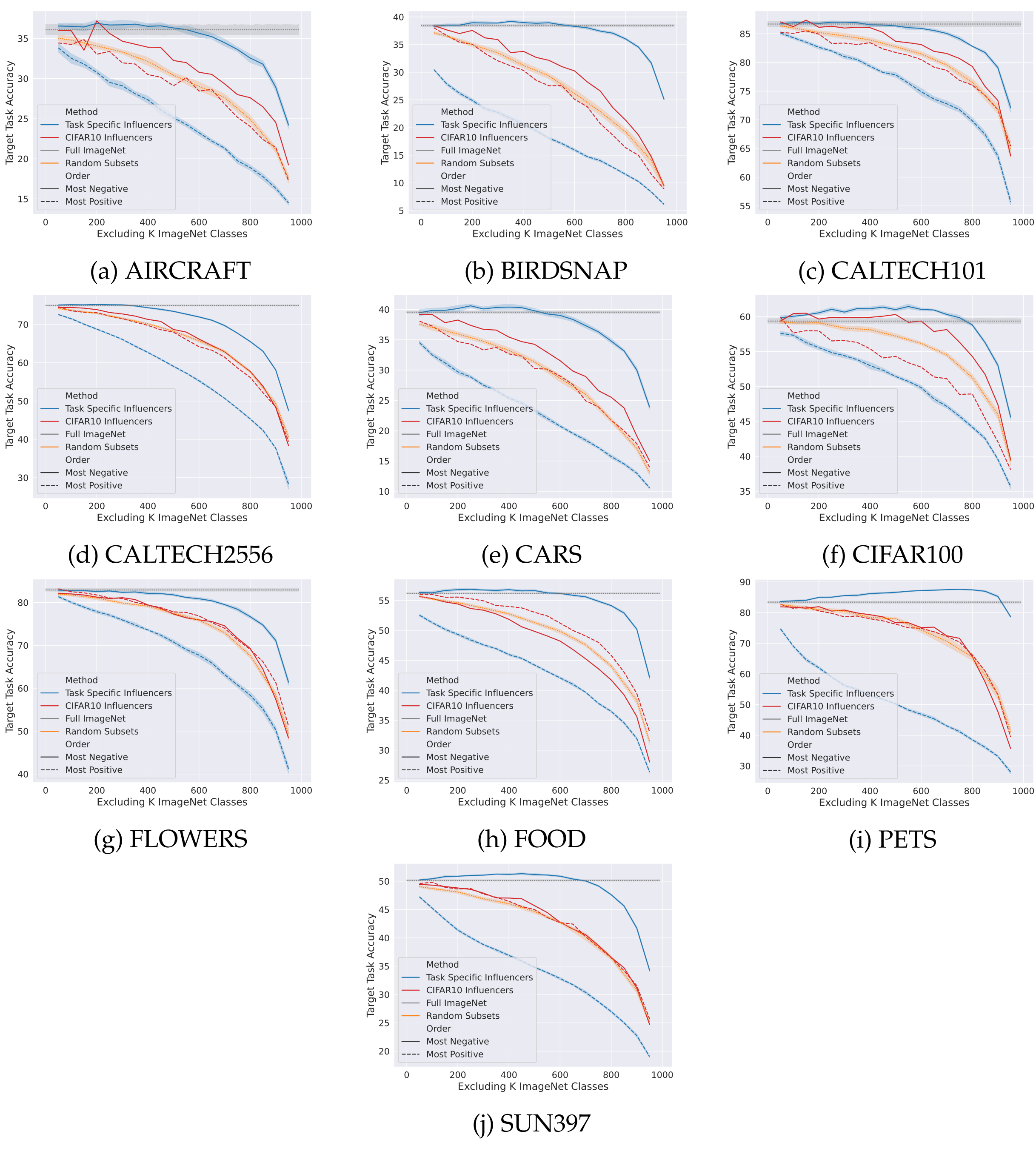


Figure H.10: We repeat the counterfactual experiments (cf. Figure 8.2b) but using the influence values computed for CIFAR-10 on other target datasets.

Transfer across architectures

How well do our transfer influences work across architectures? Recall that we computed our transfer influences using a ResNet-18. We now repeat the counterfactual experiment from Figure 8.2b, using these influences to remove classes from the source dataset when training a ResNet-50. We find that these influences transfer relatively well.

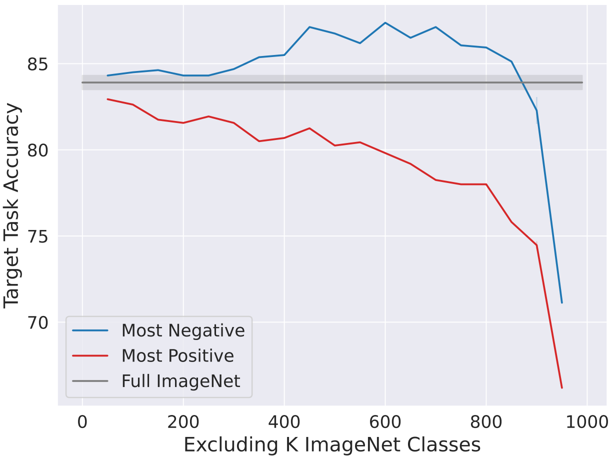


Figure H.11: We repeat the counterfactual experiments (cf. Figure 8.2b) but use our influence values computed using a ResNet-18 on a ResNet-50.

H.6 Further Convergence Analysis

In this section, we analyze the sample complexity of our influence estimation in more detail. In particular, our goal is to understand how the quality of influence estimates improves with the number of models trained. We also look at the impact of different choices of model outputs.

Instead of directly measuring our downstream objective (transfer accuracy on target dataset after removing the most influential classes), which is expensive, we design two proxy metrics to gauge the convergence of our estimates:

Rank correlation. As Ilyas et al. [IPE+22] shows, we can associate influences with a particular linear regression problem: given features $\mathbb{1}_{\mathcal{S}_i}$, an indicator vector of the subset of classes in the source dataset, predict the labels $f(t; \mathcal{S}_i)$, the model's output after it is finetuned on target dataset $\mathcal{T}$. In fact, we can interpret influences as weights corresponding to these binary features for presence of each class. That is, the influence vector $w_t = \{\text{Infl}[C_i \to t]\}_i$ defines a linear function that, given a subset of classes that the source

model is trained on, predicts the corresponding model's output when trained on that subset. Here, we focus on analyzing average model accuracy, so in fact we consider the aggregate output $\sum_i f(t; \mathcal{S}_i)$ and the corresponding aggregated influences $w = \{\text{Infl}[C_i]\}_i$.

Given this view, we can measure the quality of the influence estimates by measuring their performance on the above regression problem on a held-out[5] set of examples $\{\mathcal{S}_i, \sum_i f(t; \mathcal{S}_i)\}$. In order to make different choices of model outputs (logit, confidence, etc.) comparable, we measure performance with spearman rank correlation between the ground truth model outputs and the predictions of the linear model (whose weights are given by the influences).

We measure this correlation while varying both the number of trained models used in the influence estimation and the choice of the model output, and the results are shown in Figure H.12.

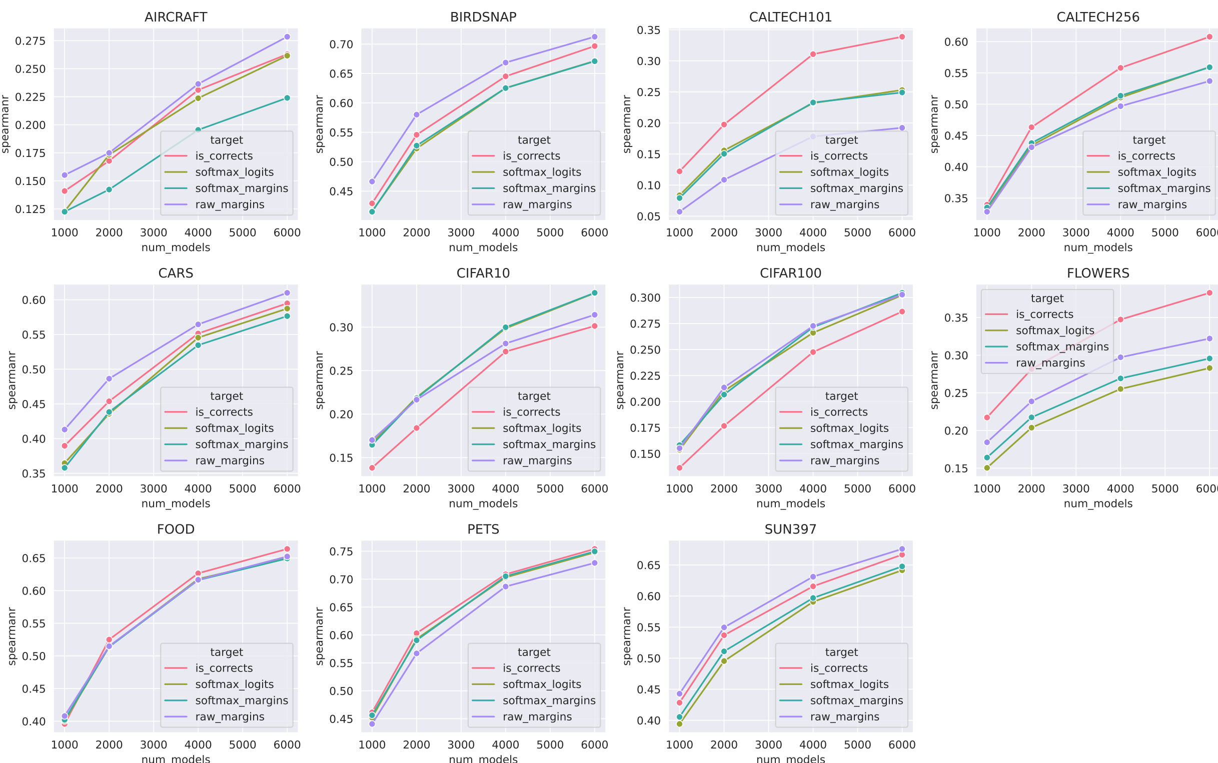


Figure H.12: Measuring improvement in influence estimates using the **rank correlation** metric. The rank correlation here measures how well influence estimates perform in the underlying regression problem of predicting target accuracy from the subset of classes included in the training set. We evaluate on a held-out set of subsets independent from those used to estimate influences. Across all datasets, correlation improves significantly with more trained models.

[5]We split the 7,540 models into a training set of 6,000 and a validation set of the remainder.

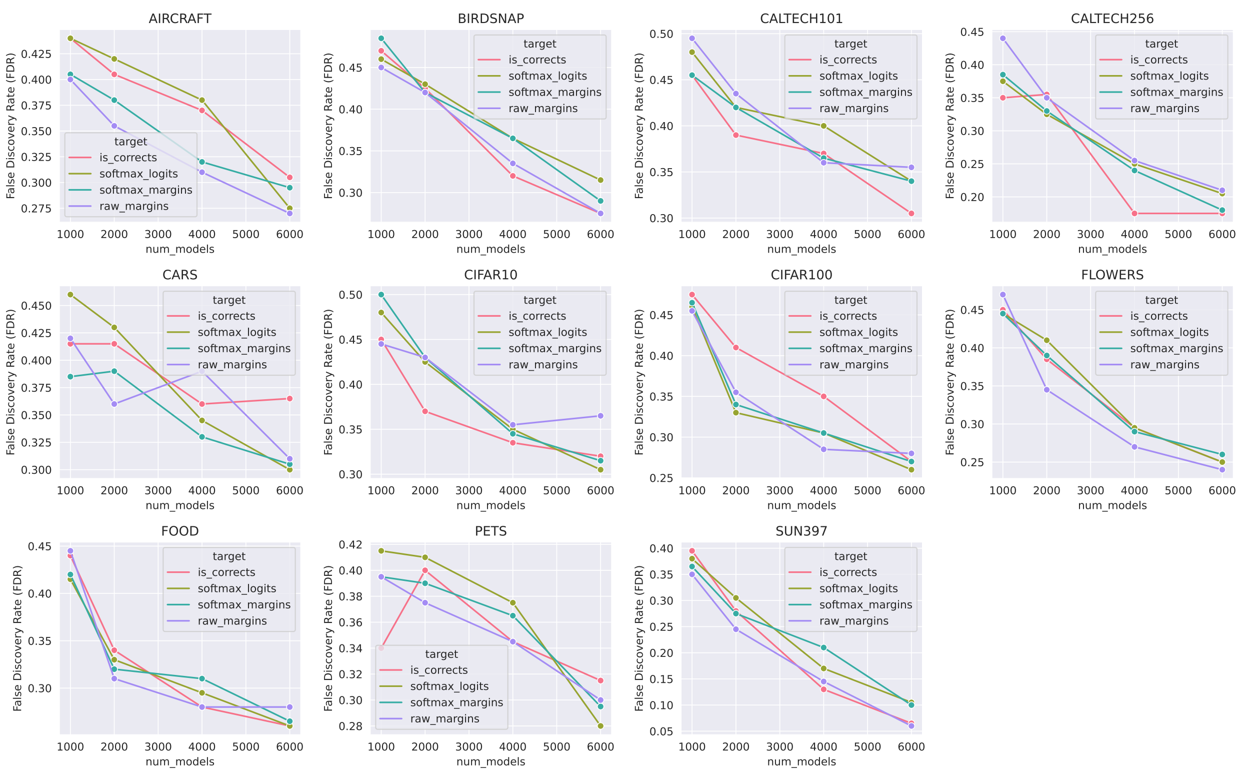


Figure H.13: Measuring improvement in influence estimates using the **False Discovery Rate** heuristic. Using a procedure (loosely) based on the Knockoffs framework, we estimate the proportion of false discoveries within the top 100 features ranked by esteimated influences. Across all datasets, FDR decreases generally with more trained models.

False discovery rate. Above we considered a measure based on predictive performance. Here, we focus on a more parameter-centric notion of False Discovery Rate (FDR). Intuitively, FDR here quantifies the following: how often are the top influencers actually just due to noise?

The Knockoffs [CFJ+18] framework allows one to perform feature selection and also estimate the FDR. At a high level, it consists of two steps: First, one constructs "knockoff" versions of the original features which are distributed "indistinguishably" (more formally, exchangable) from the original features, and at the same time are independent of the response by design. Second, one applies an estimation algorithm of choice (e.g., OLS or LASSO) to the augmented data consisting of both the original and the knockoff features. Then, the relative frequency at which a variable X_i has higher statistical signal than its knockoff counterpart $\tilde{X}_i$ indicates how likely the algorithm chooses true features, and this can be used to estimate the FDR (intuitively, if X_i is independent of the response y, X_i is indistinguishable from its knockoff $\tilde{X}_i$ and both are equally likely to have higher score).

We adapt this framework here (particularly, the verion known as model-X knockoffs)

as follows:

1. Sample an independent knockoff matrix $\tilde{X}$ consisting of 1,000 binary features from the same distribution as the original mask matrix X (namely, each instance has 500 active features).[6]

2. Estimate influences for both original and knockoff features using the difference-in-means estimator.

3. Consider the top $k = 100$ features by positive influence, and count the proportion of features that are knockoff. This yields an estimate of FDR among the top 100 features. An FDR of 0.5 indicates chance-level detection.[7]

As with the previous metric, we measure the above FDR for each target dataset while varying the number of trained models and the target output type (Figure H.13).

Discussion. We observe the following from the above analyses using our two statistics:

- There are significant gains (higher correlation and lower FDR) with increasing number of trained models.

- But neither metric appears to have plateaued with 6,000 models, so this indicates that we can improve the accuracy of our influence estimates with more trained models, which may in turn improve the max improvement in transfer accuracies (Section 8.2), among other results.

- The choice of the target type does not appear to have a significant or consistent impact across different datasets, which is also consistent with the results of our counterfactual experiments (Appendix H.2.1).

[6]Technically, this procedure is not exactly valid in the original FDR framework as X_i and $\tilde{X}_i$ are exchangable due to depenencies in the features. Nonetheless, it is accurate up to some approximation.

[7]This is different from the usual manner of controlling the FDR, but we look at this alternative metric for simplicity.