

ABSTRACT

Title of Thesis: **ROBUSTNESS AND UNDERSTANDABILITY
OF DEEP MODELS**

Amin Ghiasi

Dissertation Directed by: **Professor Thomas Goldstein
Department of Computer Science**

Deep learning has made a considerable leap in the past few decades, from promising models for solving various problems to becoming state-of-the-art. However, unlike classical machine learning models, it is sometimes difficult to explain why and how deep learning models make decisions. It is also interesting that their performance can drop with small amounts of noise. In short, deep learning models are well-performing, easily corrupted, hard-to-understand models that beat human beings in many tasks. Consequently, improving these deep models requires a deep understanding.

While deep learning models usually generalize well on unseen data, adding negligible amounts of noise to their input can flip their decision. This interesting phenomenon is known as "adversarial attacks." In this thesis, we study several defense methods against such adversarial attacks. More specifically, we focus on defense methods that, unlike traditional methods, use less computation or fewer training examples. We also show that despite the improvements in adversarial defenses, even provable certified defenses can be broken. Moreover, we revisit regularization to improve adversarial robustness.

Over the past years, many techniques have been developed for understanding and explaining

how deep neural networks make a decision. This thesis introduces a new method for studying the building blocks of neural network decisions. First, we introduce the Plug-In Inversion, a new method for inverting and visualizing deep neural network architectures, including Vision Transformers. Then we study the features a ViT learns to make a decision. We compare these features when the network trains on labeled data versus when it uses a language model's supervision for training, such as in CLIP. Last, we introduce feature sonification, which borrows feature visualization techniques to study models trained for speech recognition (non-vision) tasks.

ROBUSTNESS AND UNDERSTANDABILITY OF DEEP MODELS

by

Amin Ghiasi

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for Ph.D. degree.
2022

Advisory Committee:

Professor Thomas Goldstein, Chair/Advisor

Professor Lawrence C. Washington, Dean's Representative

Professor Furong Huang, Committee Member

Professor Rachel Rudinger, Committee Member

Professor Soheil Feizi, Committee Member

© Copyright by
Mohammad Amin Ghiasi
2022

Table of Contents

Table of Contents	ii
List of Tables	vi
List of Figures	viii
List of Abbreviations	xvii
Chapter 1: Introduction	1
Chapter 2: Adversarially Robust Transfer Learning	4
2.1 Introduction	5
2.2 Background	7
2.3 The robustness of deep features	9
2.4 Transfer learning: Recycling feature extractors	11
2.4.1 Transfer Learning with ImageNet models	13
2.4.2 Low-data regime	15
2.5 Analysis: Robust feature extractors are filters	17
2.6 End-to-end training without forgetting	19
2.7 Conclusion	23
Chapter 3: Adversarial Training For Free	24
3.1 Introduction	25
3.2 Non-targeted adversarial examples	27
3.3 Adversarial training	29
3.4 “Free” adversarial training	30
3.5 Robust models on CIFAR-10 and 100	31
3.6 Does “free” training behave like standard adversarial training?	34
3.7 Robust ImageNet classifiers	36
3.8 Conclusions	40
Chapter 4: Breaking Certified Defenses	41
4.1 The Shadow Attack	45
4.1.1 Creating untargeted attacks	47
4.2 Attacks on Randomized Smoothing	47
4.3 Ablation study of the attack parameters	50
4.4 Attacks on CROWN-IBP	54

4.5	conclusion	55
Chapter 5: Adaptive Weight Decay		57
5.1	Introduction	58
5.2	Weight Decay	59
5.2.1	Tuning Hyper-Parameters	60
5.3	Adaptive Weight Decay	62
5.4	Properties of Adaptive Weight Decay	64
5.4.1	Adversarial Robustness	65
5.4.2	Robustness to Noisy Labels	68
5.4.3	Additional Robustness benefits	70
5.4.4	Under-Fitting Data, A Desirable Property	71
5.5	Conclusion	72
Chapter 6: Plug-In Inversion		75
6.1	Introduction	76
6.1.1	Contributions	77
6.2	Background	78
6.2.1	Class inversion	78
6.2.2	Regularization	79
6.2.3	Architectures for vision	80
6.3	Plug-In Inversion	81
6.3.1	Restricting Search Space	82
6.3.2	ColorShift Augmentation	84
6.3.3	Ensembling	85
6.3.4	The Plug-in Inversion Method	86
6.4	Experimental Setup	87
6.5	Results	87
6.5.1	<u>PII</u> works on a range of architectures	87
6.5.2	Combining <u>PII</u> with DeepInversion	88
6.6	Conclusion	90
6.7	Ethical Considerations	91
6.8	Reproducibility	91
Chapter 7: Visualizing Vision Transformers		93
7.1	Introduction	95
7.2	Related Work	97
7.2.1	Optimization-Based Visualization	97
7.2.2	Other Visualization Approaches	98
7.2.3	Understanding ViTs	98
7.3	ViT Feature Visualization	99
7.4	Last-Layer Token Mixing	102
7.5	Comparison of ViTs and CNNs	105
7.6	ViTs with Language Model Supervision	107
7.7	Discussion	109

Chapter 8:	Feature Sonification	115
8.1	Basic sonification	116
8.2	Jasper Network	116
8.3	Sonification using an Auxiliary Dataset	122
8.4	Speaker Information vs Speech Information	125
8.5	Discarding Speaker Information	127
8.6	Dive Deeper	129
8.7	Sonification of the Whole Network	131
Chapter 9:	Conclusion	132
Chapter 10:	Appendix For Adversarially Robust Transfer Learning	134
10.1	Experiment details	135
10.1.1	LWF-based experiments	135
10.1.2	ImageNet to CIFAR experiments	135
10.1.3	Free training experiments	135
10.2	The distance between feature representations of natural images and augmented images	136
10.3	Low-data regime transfer learning from ImageNet to CIFAR-10	137
10.4	LwF-based robust transfer learning for similar source and target datasets	137
10.5	Improving generalization of the CIFAR-10 adversarially trained model	138
Chapter 11:	Appendix For Breaking Certified Defenses	140
Chapter 12:	Appendix For Adaptive Weight Decay	146
12.1	Experimental Setup	147
12.1.1	Adversarial Training	147
12.1.2	Noisy Label Training	147
12.2	Implementation	148
12.3	Tuning WD and LR	148
12.4	Tuning Adaptive Weight Decay	149
12.5	Extra Results	150
12.5.1	CIFAR-10 robustness to noisy labels	150
12.5.2	2D Grid search for best parameter values for ResNet32	151
12.5.3	Visualizing images from the CIFAR-100 training set where best AWD models do not fit.	152
12.5.4	Robustness to Sub-Optimal Learning Rate	152
12.5.5	Robustness to Parameter Pruning	153
12.6	Related Work on Adaptive Weight Decay	153
Chapter 13:	Appendix For Plug-In Inversion	158
13.1	Additional Results	159
13.1.1	Ablation Study for α and β	159
13.1.2	Insensitivity to TV regularization	159
13.1.3	Effect of Centering	159

13.1.4	Effect of Ensembling Size	159
13.1.5	Effect of using other Augmentations	165
13.1.6	PII on additional networks	167
13.2	Models	167
13.3	Every Class of ImageNet Dataset Inverted	172
13.4	Additional experimental setting	175
13.4.1	Robust models	175
13.5	Optimization algorithm	175
13.6	CIFAR-100 Results	175
13.7	Additional baseline comparisons	177
13.8	Quantitative Results	178
Chapter 14:	Appendix For Visualizing Vision Transformers	183
14.1	Failed Examples	184
14.2	Experimental Setup and Hyperparameters	184
14.3	Models	185
14.4	Effect of low-pass filtering	189
14.5	Additional visualizations	189
	Bibliography	222

List of Tables

2.1	Accuracy and robustness of natural and adversarially trained models on CIFAR-10+ and CIFAR-100+. The “+” sign denotes standard data augmentation ($\epsilon = 8$).	9
2.2	Transfer learning by freezing the feature extractor layers ($\epsilon = 8$).	12
2.3	Transfer learning from ImageNet ($\epsilon = 5$).	14
2.4	Distilling robust features using learning without forgetting. The bottom rows show results from transfer learning with a frozen feature extractor. The ‘+’ sign refers to using augmentation.	22
2.5	Decreasing generalization gap by transferring with LwF. For reference, last row shows results from adversarial training CIFAR-100. The ‘+’ sign refers to using augmentation. ($\epsilon = 8$)	22
3.1	Validation accuracy and robustness of CIFAR-10 models trained with various methods.	33
3.2	Validation accuracy and robustness of CIFAR-100 models trained with various methods.	33
3.3	ImageNet validation accuracy and robustness of ResNet-50 models trained with various replay parameters and $\epsilon = 2$.	38
3.4	Validation accuracy and robustness of “free” and 2-PGD trained ResNet-50 models – both trained to resist $\ell_\infty \epsilon = 4$ attacks. Note that 2-PGD training time is $3.46 \times$ that of “free” training.	39
3.5	Validation accuracy and robustness of free- $m = 4$ trained ResNets with various capacities.	40
4.1	Certified radii produced by the Randomized Smoothing method for Shadow Attackimages and also natural images (larger radii means a stronger/more confident certificate).	50
4.2	“Robust error” for natural images, and “attack error” for Shadow Attackimages using the CIFAR-10 dataset, and CROWN-IBP models.	56
5.1	Effect of moving along the diagonal on the 2D grid search for learning rate and λ_{wd} on the updates in one gradient descent step.	61
5.2	WRN28-10 Models trained with adaptive weight decay by estimating the DoG value from a model trained with non-adaptive weight decay result in models with smaller weight-norms and even smaller total training loss. Results are an average of 3 runs and X-ent is the average training cross-entropy at the end of training. Dataset names are abbreviated, C100 is CIFAR-100.	63

5.3	Adversarial robustness of WRN28-10 PGD-7 adversarially trained networks using adaptive and non-adaptive weight decay. Table summarizes the best performing hyper-parameter for each method on each dataset. Not only adaptive weight decay outperforms the non-adaptive weight decay trained model in terms of robust accuracy, it is also superior in terms of the natural accuracy.	68
5.4	CIFAR-100 adversarial robustness performance of various strong methods. Adaptive weight decay with ASAM optimizer outperforms many strong baselines. For experiments marked with * we do another round of hyper-parameter search. $\lambda_{AdaDecay}$ indicates using the work from [141]. The columns represent the method, depth and width of the WideResNets used, augmentation, number of epochs, whether ASAM, TRADES [236], and Stochastic Weight Averaging [89], were used in the training, followed by the natural accuracy and adversarial accuracy using AutoAttack. In the augmentation column, P&C is short for Pad and Crop. Note that both DoG and λ_{wd} are simply trained using 7step PGD adversarial training.	69
5.5	Train and Test accuracy of adaptive and non-adaptive weight decay trained models. While the adaptive version fits 3.28% less training data, it still results in comparable test accuracy. Our hypothesis is that probably it avoids fitting the noisy labeled data.	71
7.1	After the last layer, every patch contains the same information. “Isolating CLS” denotes the experiment where attention is only performed between patches before the final attention block, while “Patch Average” and “Patch Maximum” refer to the experiment in which the classification head is placed on top of individual patches without fine-tuning. Experiments conducted on ViT-B16.	103
7.2	ViTs more effectively correlate background information with correct class. Both foreground and background data are normalized by full image top-5 accuracy.	107
10.1	Distilling robust features using LwF for the split CIFAR-100 task. For reference, we have included the results from transfer learning by freezing the features at the bottom of the table.	138
10.2	Decreasing generalization gap by transferring with LwF. For reference, last row shows results from adversarial training CIFAR-10. The ‘+’ sign refers to using augmentation. ($\epsilon = 8$)	139
12.1	Models trained with Adaptive Weight Decay are more robust to change in learning rate and pruning.	154
14.1	ViTs more effectively correlate background information with correct class. Both foreground and background data are normalized by full image top-1 accuracy.	189

List of Figures

2.1	Robustness is preserved when we retrain only the deepest block(s) of robust CIFAR-10 and CIFAR-100 models using natural examples. The vertical axis is the accuracy on PGD-20 generated adversarial examples (i.e. robustness) after re-training deep layers. The robustness of the adversarially trained models if all layers are frozen are shown with dashed lines.	10
2.2	Wide Resnet 32-10 and the blocks used for freezing/retraining	11
2.3	When the number of training data per-class is very limited (right bars), adversarially robust transfer learning [Transferred] is better in all metrics. However, as the number of training data increases (left bars), fine-tuning with adversarial examples of the target domain [Fine-tuned with AT] results in more robustness. Adversarially robust transfer learning always results in models that work better on natural examples and is $3\times$ faster than fine-tuning with adversarial examples of the target domain. Using a pre-trained robust ImageNet improves both robustness and generalization.	16
2.4	Training an MLP for CIFAR-100 on top of the robust feature extractors from ImageNet. The x-axis corresponds to the number of hidden layers (0 is a linear classifier and corresponds to experiments in section 2.4.1.1). Robustness stems from robust feature extractors. Adding more layers on top of this extractor does not hurt robustness. Interestingly, simply adding more layers does not improve the validation accuracy and just results in more overfitting (i.e. training accuracy becomes 100%). We can slightly improve generalization using batch norm (BN) and dropout (DO).	18
2.5	Gradients of the loss w.r.t to input images for the CIFAR-100 transfer learning experiments of sections 2.4.1.1 & 2.4.2.1. The top row contains sample CIFAR-100 images. Other rows contain image gradients of the model loss. The second row is for a model transferred from a naturally trained ImageNet source. Rows 3-5 are for models transferred from a robust ImageNet source. These rows correspond to an MLP with 0 (row 3), 1 (row 4), and 2 (row 5) hidden layers on top of the robust feature extractor. The gradients in the last three rows all show interpretable generative behavior.	19
2.6	Our LwF loss has a term that enforces the similarity of feature representations (i.e. penultimate layer activations) between the source model and the fine-tuned model.	20

3.1	Natural validation accuracy of Wide Resnet 32-10 models using varied mini-batch replay parameters m . Here $m = 1$ corresponds to natural training. For large m 's, validation accuracy drops drastically. However, small m 's have little effect. For reference, CIFAR-10 and CIFAR-100 models that are 7-PGD adversarially trained have natural accuracies of 87.25% and 59.87%, respectively.	32
3.2	Attack images built for adversarially trained models look like the class into which they get misclassified. We display the last 9 CIFAR-10 clean validation images (top row) and their adversarial examples built for a 7-PGD adversarially trained (middle) and our “free” trained (bottom) models.	35
3.3	The loss surface of a 7-PGD adversarially trained model and our “free” trained model for CIFAR-10 on the first 2 validation images. In (a) and (b) we display the cross-entropy loss projected on one random (Rademacher) and one adversarial direction. In (c) and (d) we display the the cross entropy loss projected along two random directions. Both training methods behave similarly and do not operate by masking the gradients as the adversarial direction is indeed the direction where the cross-entropy loss changes the most.	36
4.1	All adversarial examples have the goal of fooling classifiers while looking “natural”. Standard attacks limit the ℓ_p -norm of the perturbation, while semantic attacks have large ℓ_p -norm while still producing natural looking images. Our attack produces large but visually subtle semantic perturbations that not only cause misclassification, but also cause a certified defense to issue a “spoofed” high-confidence certificate.	42
4.2	An adversarial example built using our Shadow Attackfor the smoothed ImageNet classifier for which the certifiable classifier produces a large certified radii. The adversarial perturbation is smooth and natural looking even-though it is large when measured using ℓ_p -metrics. Also see appendix 11.4.	50
4.3	The first 10 steps of optimization (beginning with a randomly perturbed image copy).	51
4.9	Histogram of randomized smoothed certificate radii for 100 randomly sampled CIFAR-10 validation images vs those calculated for their adversarial examples crafted using our 1-channel and 3-channel adversarial Shadow Attackattacks. The “robust” victim classifier is based off Resnet-110, and smoothed with $\sigma = 0.50$. 1-channel attacks are almost as good as the less-restricted 3-channel attacks. . . .	52
4.4	Average $L_b(\delta)$ in the first 10 steps.	52
4.5	Average $TV(\delta)$ in the first 10 steps.	52
4.6	Average $C(\delta)$ in the first 10 steps.	52
4.7	The effect of λ_s on the resulting $Dissim(\delta)$	52
4.8	The effect of λ_{tv} on the resulting $TV(\delta)$	52
4.10	The visual effect of Shadow Attackon 9 randomly selected CIFAR-10 examples using 1-Channel and 3-Channel attacks.	53
4.11	The visual effect of λ_s on imperceptibility of the perturbations. The first row indicates the value of λ_s	53
4.12	The visual effect of λ_{tv} on the perceptibility of the perturbations. The first row shows the value of λ_{tv}	54

5.1	Grid Search on different values of learning rate and weight decay on accuracy of WRN28-10 trained with SGD on (a) CIFAR10 and (b) CIFAR100, and (c) ResNet50 optimized with ASAM [107] for CIFAR-100.	60
5.2	How to find the DoG value associated with a fixed λ_{wd} hyper-parameter. The average $DoG = 0.016$ is computed by averaging the DoG values from all iterations from the beginning of the training until epoch 350 (a), where 350 is the epoch at which the training loss has plateaued (b).	62
5.3	$\ell_\infty = 8$ robust accuracy on the test set of adversarially trained WideResNet28-10 networks on CIFAR-10 and CIFAR-100. (a, c) Using different choices for the hyper-parameter of non-adaptive weight decay (λ_{wd}), and (b, d) different choices of the hyper-parameter for adaptive weight decay (DoG).	67
5.4	Comparison of similarly performing networks once trained on CIFAR-100 clean data, after training on 20% (a), 40% (b), 60% (c), and 80% (d) noisy data. Networks trained with adaptive weight decay are less sensitive to label noise compared to ones trained with non-adaptive weight decay.	73
5.5	CIFAR-100 models trained with Adaptive Weight Decay (AWD) are less sensitive to learning rate. Also, due to the smaller weight norms of models trained with AWD, they seem like good candidates for pruning. Interestingly, when models are trained with smaller learning rates, they could be more sensitive to trivial pruning algorithms such as non-iterative (i.e., global) ℓ_1 pruning. The results are average of 4 runs.	73
5.6	Examples from CIFAR-100 training dataset that have noisy labels. For every image we state [dataset label/ prediction of classifier trained with AWD / our category of noisy case]. We classify these noisy labels into several categories: W : Wrong Labels where the picture is clear enough to comprehend the correct label.; 2-1 : Two Objects from CIFAR-100 in one image, but only one label is given in the dataset; LD : Loosely Defined Classes where there is one object, but one object could be two classes at the same time. For instance, a baby girl is both a baby and a girl. V : Vague images where authors had a hard time identifying.	74
6.1	Inverted images from a robust-ResNet50 model trained on ImageNet-1k.	77
6.2	An image at different stages of optimization with centering (left), and an image inverted without centering (right), for the Border Terrier class of a robust ResNet-50.	82
6.3	An image during different stages of optimization with zoom (left), and an image inverted without zoom (right), for the Jay class of a robust ResNet-50.	83
6.4	Inversions of the robust ResNet-50 ATM class, with and without ColorShift and with varying TV regularization strength. The inversion process with ColorShift is robust to changes in the λ_{tv} hyper-parameter, while without it, λ_{tv} seems to present a trade-off between noise and blur.	85
6.5	Effect of ensemble size in the quality of inverted images for the Tugboat class of a robust ResNet-50.	86

6.6	Ablation studies repeating the previous experiments for a <u>naturally-trained</u> ResNet-50. (top) Inversion for target class Fly with and without Centering. (center) Effect of ensembling size in the quality of inverted images for the Tench class. (bottom) The effect of various combinations of zoom, Centering, and ColorShift when inverting the Dipper class.	88
6.7	Images inverted from the Volcano class for various Convolutional, Transformer, and MLP-based networks using <u>PII</u> . For more details about networks, refer to Appendix 14.3.	89
6.8	Inverting different model and class combinations for different classes using <u>PII</u> . .	90
6.9	PII Inversion results for a naturally-trained ResNet-50. PII can be used as initialization for other methods such as DeepInversion.	91
7.1	The progression for visualized features of ViT B-32. Features from early layers capture general edges and textures. Moving into deeper layers, features evolve to capture more specialized image components and finally concrete objects.	94
7.2	(a): Example feature visualization from ViT feed forward layer. <i>Left:</i> Image optimized to maximally activate a feature from layer 5. <i>Center:</i> Corresponding maximally activating ImageNet example. <i>Right:</i> The image’s patch-wise activation map. (b): A feature from the last layer most activated by shopping carts. . . .	101
7.5	Heat map of classification accuracy on the validation set when we apply the classification head trained to classify images on the top of the CLS token to the other patches.	104
7.3	Left: Visualization of key, query, and value. The visualization both fails to extract interpretable features and to distinguish between early and deep layers. High-frequency patterns and adversarial behavior dominate. Right: ViT feed forward layer. The first linear layer increases the dimension of the feature space, and the second one brings it back to its initial dimension.	112
7.4	Feature activation maps in internal layers can effectively segment the contents of an image with respect to a semantic concept. For each image triple, the visualization on top shows the result of our method, the image on the bottom left is the most activating image from the validation set and the image on the bottom right shows the activation pattern.	112
7.6	Complexity of features vs depth in ViT B-32. Visualizations suggest that ViTs are similar to CNNs in that they show a feature progression from textures to parts to objects as we progress from shallow to deep features.	113
7.7	(a): ViT-B16 detects background features. <i>Left:</i> Image optimized to maximally activate a feature from layer 6. <i>Center:</i> Corresponding maximally activating example from ImageNet. <i>Right:</i> The image’s patch-wise activation map. (b): An example of an original image and masked-out foreground and background. .	113
7.8	<i>Left:</i> Feature optimization shows sharp boundaries, and maximally activating ImageNet examples contain distinct, adjacent images. <i>Middle:</i> Feature optimization and maximally activating ImageNet photos all show images from an elevated vantage point. <i>Right:</i> Feature optimization shows a crowd of people, but maximally activating images indicate that the repetition of objects is more relevant than the type of object.	113

7.9	Features from ViT trained with CLIP that relates to the category of morbidity and music. <i>Top-left image in each category:</i> Image optimized to maximally activate a feature from layer 10. <i>Rest:</i> Seven of the ten ImageNet images that most activate the feature.	114
8.1	Sonification of the Jasper [118] network. The first four rows show different shallow features that correspond to four different phonemes in English. The y-axis shows the activation and the x-axis shows the time (temporal dimension). The fifth row shows the phonemes in the audio over time. The last row plots the amplitude of the waveform. Layer 0 is the shallowest layer in the network. The input to network for this example is "radiosterilization"	117
8.2	Examples of waveform, spectrogram, and Mel-spectrogram. It is widely known that Mel-spectrogram reduces the complexity of waveform while at the same time, being more expressive than spectrogram [97]. In all sub-plots, x-axis shows the time (temporal dimension). For the waveform, y-axis shows the amplitude. For the spectrogram and Mel-spectrogram, y-axis shows the frequency.	118
8.3	A complete overview of the ASR pipeline using the Jasper network.	118
8.4	The structure of Jasper encoder. Every gray box represents a block, and every colored box represents a layer (sub-block). The numbers at the center of layers show the number of channels. The numbers at the bottom of layers show the layer's number.	119
8.5	The activation patterns of features for the shallow layers.	120
8.6	The activation patterns of features for the first and last layers of several blocks.	121
8.7	The activation patterns of features for the last layer of several blocks.	122
8.8	A few examples of most activating words (word-level sonification).	123
8.9	Few examples of most activating phonemes (phoneme-level sonification).	124
8.10	The portion of features that are dominated by the male speakers in every layer. The x-axis shows the layer's depth and the y-axis shows the percent of male-dominated features. As we go deeper in the network, the portion of male-dominated features becomes smaller. In other words, shallow layers react to both the speech information and the speaker information, where deep layers discard the speaker information and only focus on the speech information.	125
8.11	The portion of features that are dominated by female speakers in every layer. The trend is very similar to the trend in male speakers, however in general, fewer features have been dominated by female speakers.	126
8.12	Heat-map of gender dominance for every feature in the network. There are fewer features dominated by one gender in the deep layers. However, it can be seen that the features tend to activate more by male speakers than female speakers.	127

8.13	This figure shows the set of features that activate by utterances of a same word by speakers of different genders. Note that in shallow layers the set of features are almost disjoint, while deep activation patterns are nearly invariant to gender. Every layer is showing only 20 features. We made sure that when we randomly choose the features per each layer, the portion of features that get activated by each utterance is proportional to the total number of features in that layer. To make the figure more visually appealing, for each example we re-arrange the features in every layer so that features with similar node colors stay next to each other.	128
8.14	Few examples of word-level speaker-discarded sonification.	128
8.15	Few examples of word-level speaker-discarded sonification for deep layers. . . .	129
8.16	Min, Mean, and Max, phoneme count in word-level sonification. The Mean plot shows the layer-level average phoneme count. The Min and Max plots show the average phoneme count for the features with the lowest and highest values, respectively. The x-axis shows the layer depth, and the y-axis shows the phoneme count. Deep layers seem to be activated by more complex combinations of phonemes (i.e. larger average phoneme counts). This effect is easily heard by listening to the audio clips for each layer. For this purpose, please visit our blog.	130
8.17	Average phoneme count per layer using the union of 1-gram, 3-gram, 5-gram, 7-gram, and 9-gram phoneme datasets. The x-axis shows the layer depth, and the y-axis shows the average phoneme count.	131
10.1	Figures 10.2 and ?? both show that the values of $\ z(x, \theta_r^*), z(x_a, \theta_r^*)\ _2$ are high most of the time, consequently, LwF is better done without data augmentation. . . .	136
10.2	When the number of training data per-class is very limited (right bars), adversarially robust transfer learning [Transferred] is better overall. However, as the number of training data increases (left bars), fine-tuning with adversarial examples of the target domain [Fine-tuned with AT] results in an overall better performing model. Adversarially robust transfer learning is $3\times$ faster than fine-tuning with adversarial examples of the target domain.	137
11.1	The first 10 steps of optimization and the original image when crafting attacks on CIFAR-10. See section 4.3 for the details of the experiments.	141
11.2	The visual effect of λ_s (the coefficient of $\text{Dissim}(\delta)$) on CIFAR-10. See section 4.3 for the details of the experiments.	142
11.3	The visual effect of λ_{tv} on CIFAR-10. See section 4.3 for the details of the experiments.	143
11.4	Natural looking ImageNet adversarial images that produce large certified radii for the ImageNet Gaussian smoothed classifier.	145
12.1	Held-out validation accuracy for different searches on tunable hyper-parameter values. Separate grid search on learning rate (a) and weight decay's hyper-parameter (b) does not reveal the optimal set of hyper-parameters, at the best it reveals the diagonal on which the best hyper-parameters resides. An extra search on the diagonal is necessary to find the said hyper-parameter (c).	150

12.2	Comparison of similarly performing networks once trained on CIFAR-100 clean data, after training on 20% (a), 40% (b), 60% (c), and 80% (d) noisy data. Networks trained with adaptive weight decay outperform non-adaptive trained networks.	151
12.3	Grid Search on different values of learning rate and weight decay on accuracy of ResNet32 on (a) CIFAR10 and (b) CIFAR100.	152
12.4	Examples from CIFAR-100 training dataset that have noisy labels. For every image we state [dataset label/ prediction of classifier trained with AWD / our category of noisy case]. We classify these noisy labels into several categories: W : Wrong Labels where the picture is clear enough to comprehend the correct label.; 2-1 : Two Objects from CIFAR-100 in one image, but only one label is given in the dataset; LD : Loosely Defined Classes where there is one object, but one object could be two classes at the same time. For instance, a baby girl is both a baby and a girl. V : Vague images where authors had a hard time identifying.	156
12.5	Models trained with Adaptive Weight Decay are more robust to Learning rate. Results are an average over 5 trials. We use the same experiment setup and estimated <i>DoG</i> values from section 5.3.	157
12.6	Norm of the weights for networks trained with adaptive weight decay and non-adaptive weight decay.	157
13.1	Effect of α and β on the quality of the images generated by <u>PII</u> from a naturally-trained ResNet-50 for the Tench class.	160
13.2	Effect of TV with and without ColorShift. With ColorShift it is clear that there is no need for hyper-parameter tuning for parameters such as TV. Images from the robust ResNet-50.	161
13.3	Effect of using centering vs not using centering for a robust ResNet-50.	162
13.4	Effect of using centering vs not using centering for a naturally-trained ResNet-50.	163
13.5	Effect of ensemble size when inverting a robust ResNet-50. Even small values of e give reasonably good results, but increasing e tends to give slight improvement.	164
13.6	Effect of various alternative augmentations on inverting a robust ResNet-50 model.	165
13.7	Effect of using different augmentations on inverting a naturally-trained ResNet-50.	166
13.8	<u>PII</u> applied to various vision models for the Volcano class.	168
13.9	Pre-trained models from TorchVision: https://github.com/pytorch/vision	169
13.10	Pre-trained models used from : https://github.com/lukemelas/PyTorch-Pretrained-ViT	169
13.11	Pre-trained models from [199]	170
13.12	Pre-trained models used from: [208]	171
13.13	Inversion of first 500 classes of ImageNet for the Robust Model.	173
13.14	Inversion of second 500 classes of ImageNet for the Robust Model.	174
13.15	Inverting different CIFAR-100 model and class combinations using <u>PII</u>	177
13.16	Inverting different CIFAR-10 model and class combinations using <u>PII</u>	178
13.17	Inverting different model and class combinations for different classes using DeepInversion (top row) / DeepDream (other rows). Cross-reference figure 6.8.	179

13.18	Images inverted from the Volcano class for various Convolutional, Transformer, and MLP-based networks using DeepInversion (CNN models) / DeepDream (non-CNN models). Cross-reference figure 6.7.	180
13.19	Top-1 (a) and top-5 (b) classification accuracy of various CNN, ViT, and MLP models evaluated on images generated from ViT B-32 using PII and DeepDream.	181
13.20	Top-1 (a) and top-5 (b) classification accuracy of various CNN, ViT, and MLP models evaluated on images generated from ResMLP 36-224 using PII and DeepDream.	182
14.1	Some examples of failed visualizations on the input of the attention layers. Same visualization technique fails when applied on low dimensional (e.g. on key, query, value, etc) spaces. We believe that the visualization shows roughly meaningful and interpretable visualizations in early layers, since there are not many different features to be embedded. However, in deeper layers the features are entangled, so it is more difficult to visualize them. For every example, the picture on the left shows the results of optimization and the picture on the right shows the most activating image from ImageNet1k validation set.	186
14.2	Pre-trained models used from : https://github.com/lukemelas/PyTorch-Pretrained-ViT	187
14.3	Pre-trained models from [199]	187
14.4	Pre-trained models used from: [208]	188
14.5	Effect of low-pass filtering on top-1 ImageNet accuracy. CNNs are <i>more</i> dependent on high frequency textural image information than ViTs.	189
14.6	Visualization of ViT-base-patch16	190
14.7	(Cont.) Visualization of ViT-base-patch16	191
14.8	Visualization of a CLIP model with ViT-base-patch16 as its visual part.	192
14.9	(Cont.) Visualization of a CLIP model with ViT-base-patch16 as its visual part.	193
14.10	(Cont.) Visualization of a CLIP model with ViT-base-patch16 as its visual part.	194
14.11	Visualization of ViT-base-patch32	195
14.12	(Cont.) Visualization of ViT-base-patch32	196
14.13	(Cont.) Visualization of ViT-base-patch32	197
14.14	Visualization of a CLIP model with ViT-base-patch32 as its visual part.	198
14.15	(Cont.) Visualization of a CLIP model with ViT-base-patch32 as its visual part.	199
14.16	Visualization of features in DeiT base p-16 im-224	200
14.17	(Cont.) Visualization of features in DeiT base p-16 im-224	201
14.18	Visualization of features in DeiT base p-16 im-384	202
14.19	(Cont.) Visualization of features in DeiT base p-16 im-384	203
14.20	Visualization of features in DeiT base p-16 im-384	204
14.21	Visualization of features in DeiT tiny distilled p-16 im-224	205
14.22	Visualization of features in DeiT small distilled p-16 im-224	206
14.23	Visualization of features in DeiT small distilled p-16 im-224	207
14.24	Visualization of features in DeiT base distilled p-16 im-224	208
14.25	(Cont.) Visualization of features in DeiT base distilled p-16 im-224	208
14.26	Visualization of features in Coat lite mini	209
14.27	Visualization of features in Coat lite small	209
14.28	Visualization of features in ConViT base	210

14.29	Visualization of features in ConViT small.	211
14.30	(Cont.) Visualization of features in ConViT small.	212
14.31	Visualization of features in ConViT tiny.	213
14.32	(Cont.) Visualization of features in ConViT tiny.	213
14.33	Visualization of features in Pit base im-224	214
14.34	Visualization of features in Pit base distilled im-224.	214
14.35	Visualization of features in Pit small im-224.	215
14.36	Visualization of features in Pit small distilled im-224.	215
14.37	Visualization of features in Pit tiny im-224.	216
14.38	Visualization of features in Pit tiny distilled im-224.	216
14.39	Visualization of features in Swin base p-4 w-7 im-224.	216
14.40	Visualization of features in Swin base base p-4 w-7 im-224 imagenet 22k.	217
14.41	Visualization of features in Swin base p-4 w-12 im-384.	217
14.42	Visualization of features in Swin large p-4 w-7 im-224.	218
14.43	Visualization of features in Swin large p-4 w-7 im-224 imagenet 22k.	218
14.44	Visualization of features in Swin large p-4 w-12 im-384.	219
14.45	Visualization of features in Swin small p-4 w-7 im-224.	219
14.46	Visualization of features in Twins pcvt base.	219
14.47	Visualization of features in Twins pcvt large.	220
14.48	Visualization of features in Twins pcvt small.	220
14.49	Visualization of features in Twins svt base.	220
14.50	Visualization of features in Twins svt large.	221
14.51	Visualization of features in Twins svt small.	221

List of Abbreviations

ASR	Automated Speech Recognition
AWD	Adaptive Weight Decay
B	Base
CIFAR	Canadian Institute For Advanced Research
CNN	Convolutional Neural Network
CTC	Connectionist Temporal Classification
D	Distilled
DD	Deep Dream
DeiT	Data-Efficient Image Transformer
DI	Deep Inversion
FGSM	Fast Gradient Signed Method
GD	Gradient Descent
IBP	Interval Back Propagation
L	Large
MLP	Multi Layer Perceptron
P	Patch
PGD	Projected Gradient Descent
PII	Plug-In Inversion
PVT	Pyramid Vision Transformer
ResMLP	Residual Multi Layer Perceptron
ResNet	Residual Network
S	Small
SA	Shadow Attack

SGD	Stochastic Gradient Descent
STFT	Short Time Fourier Transform
SWin	Shifted Windows
T	Tiny
TV	Total Variation
VGG	Visual Geometry Group
ViT	Vision Transformer
WD	Weight Decay
WRResNet	Wide Residual Network
WRN	Wide Residual Network

Chapter 1: Introduction

Over the past decade, deep neural networks have outperformed humans in various tasks. For example, many studies find solutions to image classification [24, 160, 169], object detection [85, 93, 122, 124, 190, 192], image segmentation [6, 23, 75, 134, 164, 193, 214], and automated speech recognition [2, 5, 7, 8, 9, 30, 71, 149, 205, 222] with the help of deep models. Therefore, it is crucial to understand such deep models, their differences, and their strengths and weaknesses.

However, the great potential of deep neural networks for learning almost any data distribution comes with its price. Many studies have shown that even though deep neural networks generalize well to unseen data, their decision is changed by adding small amounts of noise [191]. Making networks robust to such adversarial noise is one of the most challenging problems with deep models [15, 51, 63, 66, 136, 143, 147, 148, 191]. Moreover, for security-sensitive applications such as face detection systems [215], self-driving cars [196], fraud detection [38], copyright systems [167], and many more examples, adversarial examples can cause costly problems.

Adversarial training and its variants are the de-facto solutions against adversarial examples. Many articles suggest variations of adversarial training such as FGSM training [244], PGD training [129], Free training[174], and Fast adversarial training[213], while on the other hand, many articles find stronger attacks to break such defenses [32]. In light of the competition between attack and defense methods, the community has shifted its attention to finding defenses that practically and theoretically guarantee bounds for the predictions made by the defenses [12, 27, 29, 94, 114, 157, 188, 238, 240]. In other words, these defenses prove that it is impossible to change the network's decision, assuming bounded adversarial noise. Therefore, no future stronger attacks can find adversarial examples within the certified bounds.

We focus on adversarial examples in the first four sections of this dissertation. Then, we focus on the interpretability of deep neural networks in the last three chapters. Since adversarial

training requires much data and computation, we seek solutions that require either low amounts of data or fewer computations. Chapter 2 is based on the Adversarially Robust Transfer Learning [175], showing that robustness can be transferred from domains with many data examples to domains with fewer training examples. Chapter 3 is based on Adversarial Training for Free [174], which shows an efficient method for training robust networks with the same amount of computation it takes to train a network with a non-robust method. Then in Chapter 4, we focus on breaking certified robust methods. Then we revisit regularizers such as Weight Decay and study their effect on adversarial robustness in Chapter 5. The previously mentioned chapters are based on Breaking Certified Defenses [57] and Adaptive Weight Decay [61].

In Chapter 6, we shift our focus from adversarial examples to the interpretability of deep neural networks. More precisely, we focus on model inversion and feature visualization. Although there are many existing methods for both tasks, they usually require extensive hyper-parameter tuning to work for each architecture. Moreover, all previously known methods are only successful for convolutional neural networks, and they usually fail for attention-based networks such as ViTs. To the best of our knowledge, no previously existing method for feature inversion or model inversion works on ViTs. First, we introduce the Plug-In Inversion method in Chapter 5, which does not require any hyper-parameter tuning, and works on a wide range of architectures, including ViTs and ResMLPs, as well as CNNs. Then in Chapter 6, we study the similarities and differences between ViT and CNN features. We also study the effect of language model supervision on ViT features learned by the CLIP model. Last but not least, we shift our focus from models trained for vision tasks to models trained for audio tasks such as Automated Speech Recognition in Chapter 7. The previously mentioned chapters are based on Plug-In Inversion [60] and Feature Sonification [58].

Chapter 2: Adversarially Robust Transfer Learning

Transfer learning, in which a network is trained on one task and re-purposed on another, is often used to produce neural network classifiers when data is scarce or full-scale training is too costly. When the goal is to produce a model that is not only accurate but also adversarially robust, data scarcity and computational limitations become even more cumbersome. We consider robust transfer learning, in which we transfer not only performance but also robustness from a source model to a target domain. We start by observing that robust networks contain robust feature extractors. By training classifiers on top of these feature extractors, we produce new models that inherit the robustness of their parent networks. We then consider the case of “fine tuning” a network by re-training end-to-end in the target domain. When using lifelong learning strategies, this process preserves the robustness of the source network while achieving high accuracy. By using such strategies, it is possible to produce accurate and robust models with little data, and without the cost of adversarial training. Additionally, we can improve the generalization of adversarially trained models, while maintaining their robustness.

2.1 Introduction

Deep neural networks achieve human-like accuracy on a range of tasks when sufficient training data and computing power is available. However, when large datasets are unavailable for training, or practitioners require a low-cost training strategy, transfer learning methods are often used. This process starts with a source network (pre-trained on a task for which large datasets are available), which is then re-purposed to act on the target problem, usually with minimal re-training on a small dataset [146, 227].

While transfer learning greatly accelerates the training pipeline and reduces data requirements

in the target domain, it does not address the important issue of model robustness. It is well-known that naturally trained models often completely fail under adversarial inputs [15, 191]. As a result, researchers and practitioners often resort to adversarial training, in which adversarial examples are crafted on-the-fly during network training and injected into the training set. This process greatly exacerbates the problems that transfer learning seeks to avoid. The high cost of creating adversarial examples increases training time (often by an order of magnitude or more). Furthermore, robustness is known to suffer when training on a small dataset [171]. To make things worse, high-capacity models are often needed to achieve good robustness [104, 129, 174], but these models may over-fit badly on small datasets.

Contributions

The purpose of this paper is to study the adversarial robustness of models produced by transfer learning. We begin by observing that robust networks contain robust feature extractors, which are resistant to adversarial perturbations in different domains. Such robust features can be used as a basis for semi-supervised transfer learning, which only requires re-training the last layer of a network. To demonstrate the power of robust transfer learning, we transfer a robust ImageNet source model onto the CIFAR domain, achieving both high accuracy and robustness in the new domain without adversarial training. We use visualization methods to explore properties of robust feature extractors. Then, we consider the case of transfer of learning by “fine-tuning.” In this case, the source network is re-trained end-to-end using a small number of epochs on the target domain. Unfortunately, this end-to-end process does not always retain the robustness of the source domain; the network “forgets” the robust feature representations learned on the source

task. To address this problem, we use recently proposed lifelong learning methods that prevent the network from forgetting the robustness it once learned. Using our proposed methods, we construct robust models that generalize well. In particular, we improve the generalization of a robust CIFAR-100 model by roughly 2% while preserving its robustness.

2.2 Background

Adversarial examples fall within the category of evasion attacks—test-time attacks in which a perturbation is added to a natural image before inference. Adversarial attacks are most often crafted using a differentiable loss function that measures the performance of a classifier on a chosen image. In the case of norm-constrained attacks (which form the basis of most standard benchmark problems), the adversary solves

$$\max_{\delta} \quad l(x + \delta, y, \theta) \quad \text{s.t.} \quad \|\delta\|_p \leq \epsilon, \quad (2.1)$$

where θ are the (already trained and frozen) parameters of classifier $c(x, \theta) \rightarrow \hat{y}$ that maps an image to a class, l is the proxy loss used for classification (often cross-entropy), δ is the image perturbation, (x, y) is the natural image and its true class, and $\|\cdot\|_p$ is some ℓ_p -norm¹. The optimization problem in Eq. 2.1 aims to find a bounded perturbation that maximizes the cross-entropy loss given the correct label. There are many variants of this process, including DeepFool [136], L-BFGS [191], and CW [20].

Many researchers have studied methods for building a robust network which have been later shown to be ineffective when attacked with stronger adversaries [4]. Adversarial training

¹By default we will use the ℓ_∞ -norm in this paper.

[191] is one of the defenses that was not broken by [4]. While adversarial training using a weak adversary such as the FGSM attack [63] can be broken even by single step attacks which add a simple random step prior to the FGSM step [201], adversarial training using a strong attack has successfully improved robustness. [129] showed that a PGD attack (which is a BIM attack [104] with an initial random step and projection) is a strong enough attack to achieve promising adversarial training results. We will refer to this training method as PGD adversarial training. PGD adversarial training achieves good robustness on bounded attacks for MNIST [112] and acceptable robustness on CIFAR-10 [99] classifiers.

[202] show that adversarial training with strong PGD adversaries has many benefits in addition to robustness. They also state that while adversarial training may improve generalization in regimes where training data is limited (especially on MNIST), it may be at odds with generalization in regimes where data is available. This trade-off was also recently studied by [236], [189], and [173].

While, to the best of our knowledge, the transferability of robustness has not been studied in depth, [78] studied the case of adversarially training models that were pre-trained on different domains. Our work is fundamentally different in that we seek to transfer robustness without resorting to costly and data-hungry adversarial training. We train the target model on natural examples only, which allows us to directly study how well robustness transfers. Additionally, this allows us to have better generalization and achieve higher accuracy on validation examples. While as [78] state, fine-tuning on adversarial examples built for the target domain can improve robustness of relatively large datasets such as CIFAR-10 and CIFAR-100 compared to adversarial training from scratch on the target domain, we show that in the regimes of limited data (where transfer learning is more common), adversarially robust transfer learning can lead to better results

Table 2.1: Accuracy and robustness of natural and adversarially trained models on CIFAR-10+ and CIFAR-100+. The “+” sign denotes standard data augmentation ($\epsilon = 8$).

Dataset	model	validation accuracy	accuracy on PGD-20	accuracy on CW-20
CIFAR-10+	natural	95.01%	0.00%	0.00%
	robust	87.25%	45.84%	46.96%
CIFAR-100+	natural	78.84%	0.00%	0.00%
	robust	59.87%	22.76%	23.16%

measured in terms of both robustness and clean validation accuracy.

2.3 The robustness of deep features

In this section, we explore the robustness of different network layers, and demonstrate that robust networks rely on robust deep features. To do so, we start from robust classifiers ($c(\theta_r)$) for the CIFAR-100 and CIFAR-10 datasets [99], and update θ by training on natural examples. In each experiment, we re-initialize the last k layers/blocks of the network, and re-train just those layers. We start by re-initializing just the last layer, then the last two, and so on until we re-initialize all the layers.

We use the adversarially trained Wide-ResNet 32-10 [231] for CIFAR-10 from [129] as our robust model for CIFAR-10. We also adversarially train our own robust classifier for CIFAR-100 using the code from [129]. To keep things consistent, we use the same hyper-parameters used by [129] for adversarially training CIFAR-10 to adversarially train the CIFAR-100 model.² The performance of the CIFAR-10 and CIFAR-100 models on natural and adversarial examples are summarized in Table 2.1. To measure robustness, we evaluate the models on adversarial examples built using PGD attacks.

We break the WRN 32-10 model into 17 blocks, which are depicted in Fig. 2.2. In each

²We adv. train the WRN 32-10 on CIFAR-100 using a 7-step ℓ_∞ PGD attack with step-size=2 and $\epsilon = 8$. We train for 80,000 iterations with a batch-size of 128.

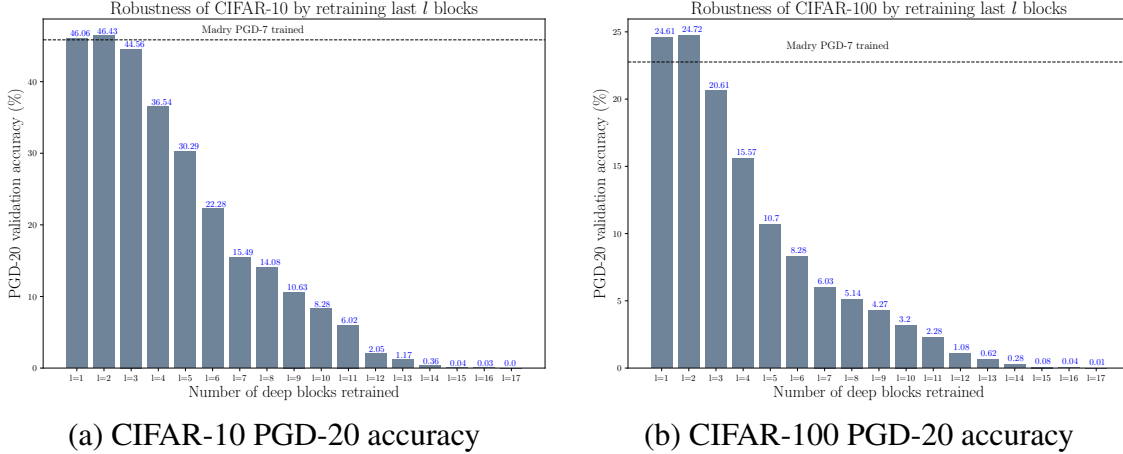


Figure 2.1: Robustness is preserved when we retrain only the deepest block(s) of robust CIFAR-10 and CIFAR-100 models using natural examples. The vertical axis is the accuracy on PGD-20 generated adversarial examples (i.e. robustness) after re-training deep layers. The robustness of the adversarially trained models if all layers are frozen are shown with dashed lines.

experiment, we first re-initialize the k deepest blocks (blocks 1 through k) and then train the parameters of those blocks on natural images³. We train for 20,000 iterations using Momentum SGD and a learning rate of 0.001. We then incrementally unfreeze and train more blocks. For each experiment, we evaluate the newly trained model’s accuracy on validation adversarial examples built with a 20-step PGD ℓ_∞ attack with $\epsilon = 8$.

Fig. 2.1 shows that robustness does not drop if only the final layers of the networks are re-trained on natural examples. In fact, there is a slight increase in robustness compared to the baseline PGD-7 adversarially trained models when we just retrain the last batch-normalization block and fully connected block. As we unfreeze and train more blocks, the network’s robustness suddenly drops. This leads us to believe that a hardened network’s robustness is mainly due to robust deep feature representations and robustness is preserved if we re-train on top of deep features.

³In this experiment, we use standard data augmentation techniques.

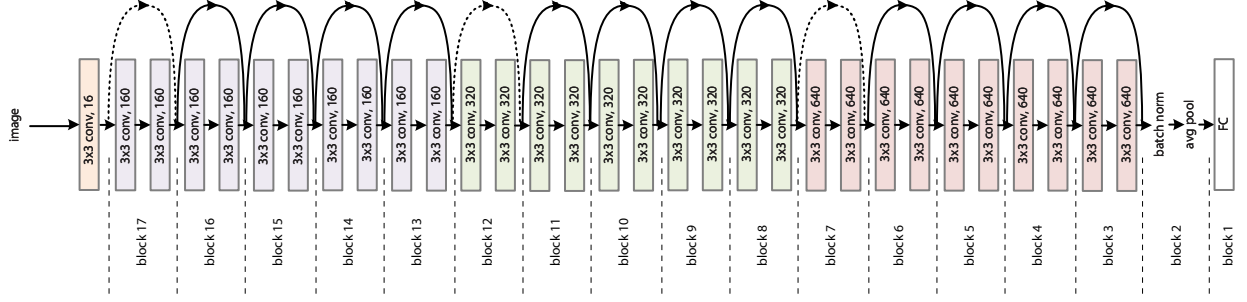


Figure 2.2: Wide Resnet 32-10 and the blocks used for freezing/retraining

Now that we have identified feature extractors as a source of robustness, it is natural to investigate whether robustness is preserved when transfer learning using robust feature extractors. We will study two different approaches for transferring robustness across datasets: one in which only the last layer is re-trained, and one with end-to-end re-training.

2.4 Transfer learning: Recycling feature extractors

We study how robustness transfers when the feature extractor layers of the source network are frozen, and we retrain only the last fully connected layer (i.e. the classification layer) for the new task. Formally, the transfer learning objective is:

$$\min_w l(z(x, \theta^*), y, w) \quad (2.2)$$

where z is the deep feature extractor function with pre-trained and now “frozen” parameters θ^* , and w represents the trainable parameters of the last fully connected layer. To investigate how well robustness transfers, we use two source models: one that is hardened by adversarial training

Table 2.2: Transfer learning by freezing the feature extractor layers ($\epsilon = 8$).

Source Dataset	Target Dataset	Source Model	val.	PGD-20	CW-20
CIFAR-100	CIFAR-10	natural	83.05%	0.00%	0.00%
		robust	72.05%	17.70%	17.43%
CIFAR-100 (50% of classes)	CIFAR-100 (other 50% of classes)	natural	71.44%	0.00%	0.00%
		robust	58.48%	15.86%	15.30%
CIFAR-100 (50% of classes)	CIFAR-100 (same 50% of classes)	natural	80.20%	0.00%	0.00%
		robust	64.96%	25.16%	25.56%
CIFAR-10	CIFAR-100	natural	49.66%	0.00%	0.00%
		robust	41.59%	11.63%	9.68%

and another that is naturally trained.

We use models trained on CIFAR-100 as source models and perform transfer learning from CIFAR-100 to CIFAR-10. The results are summarized in Table 2.2. Compared to adversarial/natural training the target model, transferring from a source model seems to result in a drop in natural accuracy (compare first row of Table 2.1 to the first row of Table 2.2). This difference is wider when the source and target data distributions are dissimilar [227].

To evaluate our method on two datasets with more similar attributes, we randomly partition CIFAR-100 into two disjoint subsets where each subset contains images corresponding to 50 classes. Table 2.2 shows the accuracy of transferring from one of the disjoint sets to the other (second row) and to the same set (third row). We can compare results of transfer learning with adversarial training on CIFAR-100 by averaging the results in the second and third rows of Table 2.2 to get the accuracy across all 100 classes of CIFAR-100.⁴ By doing so, we see that the accuracy of the transferred classifier matches that of the adversarially trained one, even though no adversarial training took place in the target domain. For completeness, we have also included experiments where we use CIFAR-10 as the source and CIFAR-100 as the target domain.

⁴The robust CIFAR-100 classifier has 59.87% validation accuracy and 22.76% accuracy on PGD-20 adversarial examples. The average validation accuracy of the two half-CIFAR-100 classifiers on validation examples is $\frac{64.96\%+58.48\%}{2} = 61.72\%$ while the average robustness is $\frac{25.16\%+15.86\%}{2} = 20.51\%$.

We make the following observations from the transfer-learning results in Table 2.2. 1) robustness transfers: when the source model used for transfer learning is robust, the target model is also robust (although less so than the source), 2) robustness transfers between models that are more similar: If the source and target models are trained on datasets which have similar distributions (and number of classes), robustness transfers better, and 3) validation accuracy is worst if we use a robust model as the source compared to using a conventionally trained source model: if the source model is naturally trained, the natural validation accuracy is better, although the target model is then vulnerable to adversarial perturbations.

2.4.1 Transfer Learning with ImageNet models

Transfer learning using models trained on ImageNet [166] as the source is a common practice in industry because ImageNet feature extractors are powerful and expressive. In this section we evaluate how well robustness transfers from these models.

2.4.1.1 Transfer learning using ImageNet

Starting from both a natural and robust ImageNet model, we perform the same set of experiments we did in section 2.4. Robust ImageNet models do not withstand untargeted ℓ_∞ attacks using as large an ϵ as those that can be used for simpler datasets like CIFAR. Following the method [174], we “free train” a robust ResNet-50 on ImageNet using replay hyper-parameter $m = 4$. The hardened ImageNet classifier withstands attacks bounded by $\epsilon = 5$. Our robust ImageNet achieves 59.05% top-1 accuracy and roughly 27% accuracy against PGD-20 $\ell_\infty \epsilon = 5$ attacks on validation examples. We experiment with using this robust ImageNet model and a

Table 2.3: Transfer learning from ImageNet ($\epsilon = 5$).

Architecture and Source Dataset	Target Dataset	Source Model	val.	PGD-20	CW-20
ResNet-50 ImageNet	CIFAR-10+	natural	90.49%	0.01%	0.00%
		robust ($\epsilon = 5$)	88.33%	22.66%	26.01%
	CIFAR-100+	natural	72.84%	0.05%	0.00%
		robust ($\epsilon = 5$)	68.88%	15.21%	18.34%
Robust u-ResNet-50 ($\epsilon = 5$) for CIFAR-10+			82.00%	53.11%	
Robust u-ResNet-50 ($\epsilon = 5$) for CIFAR-100+			59.90%	29.54%	

conventionally trained ResNet-50 ImageNet model as the source models.

Using the ImageNet source models, we train CIFAR classifiers by retraining the last layer on natural CIFAR examples. We up-sample the 32×32 -dimensional CIFAR images to 224×224 before feeding them into the ResNet-50 source models that are trained on ImageNet. For evaluation purposes, we also train robust ResNet-50 models from scratch using [174] for the CIFAR models. To ensure that the transfer learning models and the end-to-end trained robust models have the same capacity and dimensionality, we first upsample the CIFAR images before feeding them to the ResNet-50 model. To distinguish between the common case of training ResNet models on CIFAR images that are 32×32 -dimensional, we call our models that are trained on the upsampled CIFAR datasets the upsample-first ResNets or “*u-ResNets*”.

Table 2.3 illustrates that using a robust ImageNet model as a source results in high validation accuracy for the transferred CIFAR target models. Also, given that the ImageNet classifier by itself is 27% robust, the CIFAR-10 model maintains the majority of that 27% robustness. When we compare the end-to-end hardened classifiers (robust u-ResNets) with the transferred classifier, we can see that while the robustness is less for the transferred case, transferred models result in considerably better performance on clean validation examples.

2.4.2 Low-data regime

As touched on before, transfer learning is more common in situations where the number of training points in the target domain is limited. Up until now, as a proof of concept, we have illustrated the majority of our experiments on the CIFAR target domains where we have many training points per-class. [78] show that starting from a pre-trained robust ImageNet model and fine-tuning on adversarial examples of the CIFAR domain can improve robustness beyond that of simply adversarial training CIFAR. Here, we illustrate the effect of training data size on robustness and natural performance by running various experiments on subsets of CIFAR-100 where we vary the number of training points per-class (N).

We compare three different hardening methods: (1) Free-training/adversarial training the target domain [174]; (2) fine-tuning using adversarial examples of the target task starting from the Free-4 robust ImageNet model similar to [78]; and (3) training a fully connected layer on top of the frozen feature extractors of the Free-4 robust ImageNet model using natural examples from the target task. For comparing the three different approaches, we look at three metrics: (a) clean validation accuracy; (b) robustness against PGD-20 validation adversarial examples; and (c) average of robustness and clean performance ($((a)+(b))/2$.) The results are summarized in Fig. 2.3. In the regimes where transfer learning is more common, adversarially robust transfer learning results in the best overall performance. Adversarially/Free training the target domain results in less robustness and validation accuracy compared to fine-tuning which highlights the importance of pre-training [78]. Note that in terms of computational resources required, the cost of fine-tuning on adversarial examples of the target domain is about $k \times$ our method since it requires generation of adversarial examples using k -step PGD attacks (we set $k = 3$).

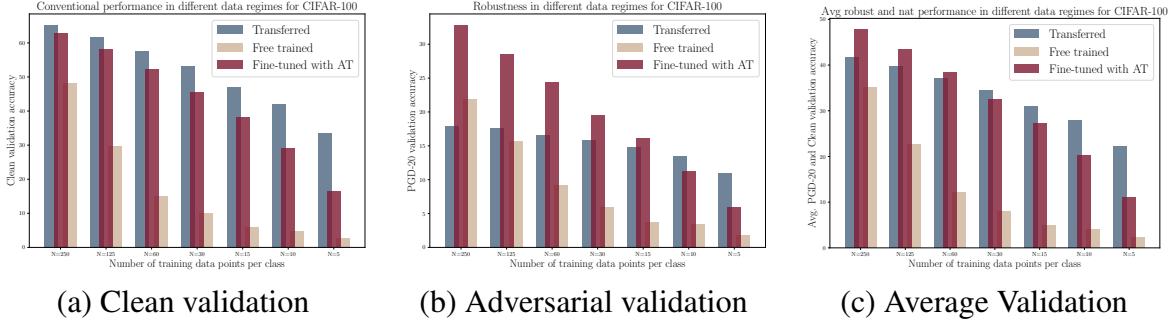


Figure 2.3: When the number of training data per-class is very limited (right bars), adversarially robust transfer learning [Transferred] is better in all metrics. However, as the number of training data increases (left bars), fine-tuning with adversarial examples of the target domain [Fine-tuned with AT] results in more robustness. Adversarially robust transfer learning always results in models that work better on natural examples and is $3\times$ faster than fine-tuning with adversarial examples of the target domain. Using a pre-trained robust ImageNet improves both robustness and generalization.

2.4.2.1 Training deeper networks on top of robust feature extractors

The basic transfer learning setting of section 2.4.1.1 only re-trains one layer for the new task. In section 2.4.1.1, when we transferred from the robust ImageNet to CIFAR-100, the natural training accuracy was 88.84%. Given the small number of trainable parameters left for the network ($\approx 2048 \times 100$) and the fixed feature extractor, the network was not capable of completely fitting the training data. This means that there is potential to improve natural accuracy by learning more complex non-linear features and increasing the number of trainable parameters.

To increase representation capacity and the number of trainable parameters, instead of training a 1-layer network on top of the feature extractor, we train a multi-layer perceptron (MLP) network on top of the robust feature extractor. To keep things simple and prevent bottle-necking, every hidden layer we add has 2048 neurons. We plot the training and validation accuracies on the natural examples and the robustness (i.e. PGD-20 validation accuracy) in Fig. 2.4 for various numbers of hidden layers. As can be seen, adding one layer is enough to achieve 100%

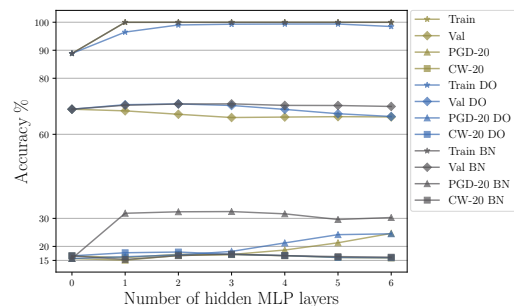
training accuracy. However, doing so does not result in an increase in validation accuracy. To the contrary, adding more layers can result in a slight drop in validation accuracy due to overfitting. As illustrated, we can improve generalization using simple but effective methods such as dropout [187] (with probability 0.25) and batch-normalization [88].

However, the most interesting behavior we observe in this experiment is that, as we increase the number of hidden layers, the robustness to PGD-20 attacks improves. Note, this seems to happen even when we transfer from a naturally trained ImageNet model. While for the case where we have no hidden layers, robustness is 0.00% on CIFAR100 when we use a naturally trained ImageNet model as source, if our MLP has 1, 2, 3, or 5 hidden layers, our robustness against PGD attacks would be 0.03%, 0.09%, 0.31% and 6.61%, respectively. This leads us to suspect that this behavior may be an artifact of vanishing gradients for adversary as the softmax loss saturates when the data is fit perfectly [4]. Therefore, for this case we change our robustness measure and use the CW attack [20] which will encounter fewer numerical issues because its loss function does not have a softmax component and does not saturate. Attacking the model from the natural source with CW-20 completely breaks the model and achieves 0.00% robustness. Most interestingly, attacking the model transferred from a robust source using the CW objective maintains robustness even when the number of hidden layers increases.

2.5 Analysis: Robust feature extractors are filters

Our experiments suggest that the robustness of neural networks arises in large part from the presence of robust feature extractors. We have used this observation to transfer both robustness and accuracy between domains using transfer-learning. However, we have not yet fully delved

Figure 2.4: Training an MLP for CIFAR-100 on top of the robust feature extractors from ImageNet. The x-axis corresponds to the number of hidden layers (0 is a linear classifier and corresponds to experiments in section 2.4.1.1). Robustness stems from robust feature extractors. Adding more layers on top of this extractor does not hurt robustness. Interestingly, simply adding more layers does not improve the validation accuracy and just results in more overfitting (i.e. training accuracy becomes 100%). We can slightly improve generalization using batch norm (BN) and dropout (DO).



into what it means to have a robust feature extractor. Through visualizations, [202] studied how adversarial training causes the image gradients of neural networks to exhibit meaningful generative behavior. In other words, adversarial perturbations on hardened networks “look like” the class into which the image is perturbed. Given that optimization-based attacks build adversarial examples using the image gradient, we also visualize the image gradients of our transferred models to see if they exhibit the same generative behavior as adversarially trained nets.

Fig. 2.5 plots the gradient of the loss w.r.t. the input image for models obtained by re-training only the last layer, and also for the case where we train MLPs on top of a robust feature extractor. The gradients for the transfer-learned models with a robust source are interpretable and “look like” the adversarial object class, while the gradients of models transferred from a natural source do not. This interpretability comes despite the fact that the source model was hardened against attacks on one dataset, and the transferred model is being tested on object classes from another. Also, we see that adding more layers on top of the feature extractor, which often leads to over-fitting, does not make gradients less interpretable. This latter observation is consistent with our observation that added layers preserve robustness(Fig. 2.4).

These observations, together with the success of robust transfer learning, leads us to speculate

that a robust model’s feature extractors act as a “filter” that ignores irrelevant parts of the image.

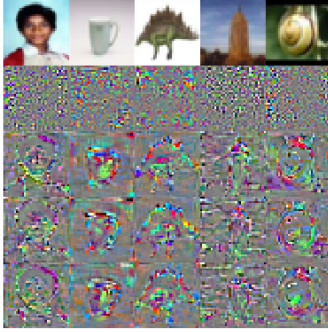


Figure 2.5: Gradients of the loss w.r.t to input images for the CIFAR-100 transfer learning experiments of sections 2.4.1.1 & 2.4.2.1. The top row contains sample CIFAR-100 images. Other rows contain image gradients of the model loss. The second row is for a model transferred from a naturally trained ImageNet source. Rows 3-5 are for models transferred from a robust ImageNet source. These rows correspond to an MLP with 0 (row 3), 1 (row 4), and 2 (row 5) hidden layers on top of the robust feature extractor. The gradients in the last three rows all show interpretable generative behavior.

2.6 End-to-end training without forgetting

As discussed in section 2.4, transfer learning can preserve robustness of the robust source model. However, it comes at the cost of decreased validation accuracy on natural examples compared to the case where we use a naturally trained source model. Consequently, there seems to be a trade-off between generalization and robustness based on the choice of the source model. For any given classifier, the trade-off between generalization and robustness is the subject of recent research [173, 202, 236]. In this section, we intend to improve the overall performance of classifiers transferred from a robust source model by improving their generalization on natural images. To do so, unlike previous sections where we froze the feature extractor mainly to preserve robustness, we fine tune the feature extractor parameters θ . Ideally, we should learn to perform well on the target dataset without catastrophically forgetting the robustness of the source model. To achieve this, we utilize lifelong learning methods.

Learning without Forgetting (LwF) [119] is a method for overcoming catastrophic forgetting. The method is based on distillation. In this framework, we train the target model with a loss that

includes a distillation term from the previous model.

$$\min_{w, \theta} l(z(x, \theta), y, w) + \lambda_d \cdot d(z(x, \theta), z_0(x, \theta_r^*)) \quad (2.3)$$

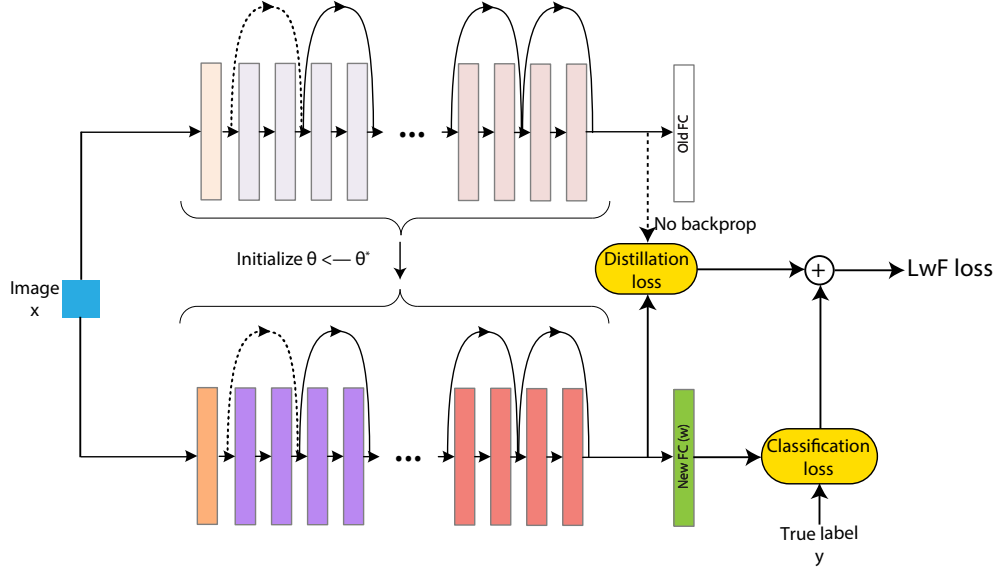


Figure 2.6: Our LwF loss has a term that enforces the similarity of feature representations (i.e. penultimate layer activations) between the source model and the fine-tuned model.

where, in our method, λ_d is the feature representation similarity penalty, and d is some distance metric between the robust model’s feature representations $z_0(x, \theta_r^*)$ and the current model’s feature representations $z(x, \theta)$. Unlike the original LwF paper that used a distilled loss from [81] and applies distillation to the logits, we simply choose d to be the ℓ_2 -norm and apply distillation to the penultimate layer⁵. Our loss is designed to make the feature representations of the source and target network similar, thus preserving the robust feature representations (Fig. 2.6). Ideally, $z(x, \theta) \approx z(x, \theta_r^*)$. To speed up training, given robust feature extractor parameters θ_r^* , we store $z_0(x, \theta_r^*)$ for the images of the target task and load this from memory (i.e. offline)

⁵We do so since in section 2.3 we found the source of robustness to be the feature extractors and this observation was later reinforced due to the empirical results in section 2.4

instead of performing a forward pass through the robust source network online. Therefore, in the experiments related to LwF, we do not train with data augmentation because we have not pre-computed $z(x_a, \theta_r^*)$, where x_a is the augmented image. Empirically we verified that $d(z(x, \theta_r^*), z(x_a, \theta_r^*))$ was not negligible⁶.

To improve performance, we follow a warm-start scheme and only train the fully connected parameters w early in training. We then cut the learning rate and continue fine tuning both feature extractors (θ) and w . In our experiments, we use a learning rate of 0.001, and the warm-start makes up half of the total training iterations. Starting from the pre-trained source model, we train for a total of 20,000 iterations with batch-size 128. The results with an adversarially trained CIFAR-100 model as source and CIFAR-10 as target are summarized in Table 2.4.⁷

As can be seen, having a LwF-type regularizer helps in maintaining robustness and also results in a considerable increase in validation accuracy. The trade-off between robustness and generalization can be controlled by the choice of λ_d . It seems that for some choices of λ_d such as 0.1, robustness also increases. However, in hindsight, the increase in accuracy on PGD-20 adversarial examples is not solely due to improvement in robustness. It is due to the fact that the validation accuracy has increased and we have a better classifier overall. For easier comparisons, we have provided the transfer results without LwF at the bottom of Table 2.4. Note that using LwF, we can keep the robustness of the source model and also achieve clean validation accuracy comparable to a model that uses naturally trained feature extractors. In the supplementary, we show that similar conclusions can be drawn for the split CIFAR-100 task.

⁶The analysis is in the supplementary.

⁷Source code for LwF-based experiments: <https://github.com/ashafahi/RobustTransferLWF>

Table 2.4: Distilling robust features using learning without forgetting. The bottom rows show results from transfer learning with a frozen feature extractor. The ‘+’ sign refers to using augmentation.

Source → Target Dataset	Source Model	λ_d	val.	PGD-20 ($\epsilon = 8$)
CIFAR-100+ → CIFAR-10	robust	1e-7	89.07%	0.61%
		0.001	86.15%	4.70%
		0.0025	81.90%	15.42%
		0.005	79.35%	17.61%
		0.01	77.73%	17.55%
		0.1	73.39%	18.62%
CIFAR-100+ → CIFAR-10+	natural	NA	83.05%	0.00%
	robust	NA	72.05%	17.70%

Decreasing generalization gap of adversarially trained networks

We demonstrated in our transfer experiments that using our LwF-type loss, can help decrease the generalization gap while preserving robustness. In this section, we assume that the source domain is the adversarial example domain of a dataset and the target domain is the clean example domain of the same dataset. This experiment can be seen as applying transfer learning from the adversarial example domain to the natural example domain while preventing forgetting the adversarial domain.

In the case where the source and target datasets are the same (Transferring from a robust CIFAR-100 model to CIFAR-100), by applying our LwF-type loss, we can improve the generalization of robust models. Our results are summarized in Table 2.5.

Table 2.5: Decreasing generalization gap by transferring with LwF. For reference, last row shows results from adversarial training CIFAR-100. The ‘+’ sign refers to using augmentation. ($\epsilon = 8$)

Source → Target Dataset	Source Model	λ_d	val.	PGD-20	CW-20
CIFAR-100+ → CIFAR-100	robust	1e-5	61.53%	21.83%	21.97%
		5e-5	61.71%	23.44%	22.94%
		1e-4	61.38%	23.95%	23.40%
		0.001	60.17%	24.31%	23.17%
		0.01	59.87%	24.10%	22.96%
CIFAR-100+	robust	NA	59.87%	22.76%	23.16%

2.7 Conclusion

We identified the feature extractors of adversarially trained models as a source of robustness, and use this observation to transfer robustness to new problems domains without adversarial training. While transferring from a natural model can achieve higher validation accuracy in comparison to transferring from a robust model, we can close the gap and maintain the initial transferred robustness by borrowing ideas from the lifelong learning literature. The success of this methods suggests that a robust feature extractor is effectively a filter that sifts out relevant components of an image that are needed to assign class labels. We hope that the insights from this study enable practitioners to build robust models in situations with limited labeled training data, or when the cost and complexity of adversarial training from scratch is untenable.

Chapter 3: Adversarial Training For Free

Adversarial training, in which a network is trained on adversarial examples, is one of the few defenses against adversarial attacks that withstands strong attacks. Unfortunately, the high cost of generating strong adversarial examples makes standard adversarial training impractical on large-scale problems like ImageNet. We present an algorithm that eliminates the overhead cost of generating adversarial examples by recycling the gradient information computed when updating model parameters. Our “free” adversarial training algorithm achieves comparable robustness to PGD adversarial training on the CIFAR-10 and CIFAR-100 datasets at negligible additional cost compared to natural training, and can be 7 to 30 times faster than other strong adversarial training methods. Using a single workstation with 4 P100 GPUs and 2 days of runtime, we can train a robust model for the large-scale ImageNet classification task that maintains 40% accuracy against PGD attacks.

3.1 Introduction

Deep learning has been widely applied to various computer vision tasks with excellent performance. Prior to the realization of the adversarial example phenomenon by [15, 191], model performance on clean examples was the the main evaluation criteria. However, in security-critical applications, robustness to adversarial attacks has emerged as a critical factor.

A robust classifier is one that correctly labels adversarially perturbed images. Alternatively, robustness may be achieved by detecting and rejecting adversarial examples [128, 133, 224]. Recently, [4] broke a complete suite of allegedly robust defenses, leaving adversarial training, in which the defender augments each minibatch of training data with adversarial examples [129], among the few that remain resistant to attacks. Adversarial training is time-consuming—in

addition to the gradient computation needed to update the network parameters, each stochastic gradient descent (SGD) iteration requires multiple gradient computations to produce adversarial images. In fact, it takes 3-30 times longer to form a robust network with adversarial training than forming a non-robust equivalent. Put simply, the actual slowdown factor depends on the number of gradient steps used for adversarial example generation.

The high cost of adversarial training has motivated a number of alternatives. Some recent works replace the perturbation generation in adversarial training with a parameterized generator network [11, 153, 216]. This approach is slower than standard training, and problematic on complex datasets, such as ImageNet, for which it is hard to produce highly expressive GANs that cover the entire image space. Another popular defense strategy is to regularize the training loss using label smoothing, logit squeezing, or a Jacobian regularization [77, 90, 138, 165, 228?]. These methods have not been applied to large-scale problems, such as ImageNet, and can be applied in parallel to adversarial training.

Recently, there has been a surge of certified defenses [157, 158, 206, 210, 212]. These methods were mostly demonstrated for small networks, low-res datasets, and relatively small perturbation budgets (ϵ). [29] propose randomized smoothing as a certified defense method suitable for ImageNet. They claim to achieve 12% robustness against non-targeted attacks that are within an ℓ_2 radius of 3 (for images with pixels in $[0, 1]$). This is roughly equivalent to an ℓ_∞ radius of $\epsilon = 2$ when pixels lie in $[0, 255]$.

Adversarial training remains among the most trusted defenses, but it is nearly intractable on large-scale problems. Adversarial training on high-resolution datasets, including ImageNet, has only been within reach for research labs having hundreds of GPUs¹. Even on reasonably-sized

¹[220] use 128 V100s and [95] use 53 P100s for targeted adv training ImageNet.

datasets, such as CIFAR-10 and CIFAR-100, adversarial training is time consuming and can take multiple days.

Contributions

We propose a fast adversarial training algorithm that produces robust models with almost no extra cost relative to natural training. The key idea is to update both the model parameters and image perturbations using one simultaneous backward pass, rather than using separate gradient computations for each update step. Our proposed method has the same computational cost as conventional natural training, and can be 3-30 times faster than previous adversarial training methods [129, 220]. Our robust models trained on CIFAR-10 and CIFAR-100 achieve accuracies comparable and even slightly exceeding models trained with conventional adversarial training when defending against strong PGD attacks.

We can apply our algorithm to the large-scale ImageNet classification task on a single workstation with four P100 GPUs in about two days, achieving 40% accuracy against non-targeted PGD attacks. To the best of our knowledge, our method is the first to successfully train a robust model for ImageNet based on the non-targeted formulation and achieves results competitive with previous (significantly more complex) methods [95, 220].

3.2 Non-targeted adversarial examples

Adversarial examples come in two flavors: non-targeted and targeted. Given a fixed classifier with parameters θ , an image x with true label y , and classification proxy loss l , a bounded non-targeted attack sneaks an example out of its natural class and into another. This is done

by solving

$$\max_{\delta} \quad l(x + \delta, y, \theta), \quad \text{subject to} \quad \|\delta\|_p \leq \epsilon, \quad (3.1)$$

where δ is the adversarial perturbation, $\|\cdot\|_p$ is some ℓ_p -norm distance metric, and ϵ is the adversarial manipulation budget. In contrast to non-targeted attacks, a targeted attack scooches an image into a specific class of the attacker’s choice.

In what follows, we will use non-targeted adversarial examples both for evaluating the robustness of our models and also for adversarial training. We briefly review some of the closely related methods for generating adversarial examples. In the context of ℓ_∞ -bounded attacks, the Fast Gradient Sign Method (FGSM) by [63] is one of the most popular non-targeted methods that uses the sign of the gradients to construct an adversarial example in one iteration:

$$x_{adv} = x + \epsilon \cdot \text{sign}(\nabla_x l(x, y, \theta)). \quad (3.2)$$

The Basic Iterative Method (BIM) by [105] is an iterative version of FGSM. The PGD attack is a variant of BIM with uniform random noise as initialization, which is recognized by [4] to be one of the most powerful first-order attacks. The initial random noise was first studied by [201] to enable FGSM to attack models that rely on “gradient masking.” In the PGD attack algorithm, the number of iterations K plays an important role in the strength of attacks, and also the computation time for generating adversarial examples. In each iteration, a complete forward and backward pass is needed to compute the gradient of the loss with respect to the image. Throughout this paper we will refer to a K -step PGD attack as PGD- K .

3.3 Adversarial training

Adversarial training can be traced back to [63], in which models were hardened by producing adversarial examples and injecting them into training data. The robustness achieved by adversarial training depends on the strength of the adversarial examples used. Training on fast non-iterative attacks such as FGSM and Rand+FGSM only results in robustness against non-iterative attacks, and not against PGD attacks [104, 129]. Consequently, [129] propose training on multi-step PGD adversaries, achieving state-of-the-art robustness levels against ℓ_∞ attacks on MNIST and CIFAR-10 datasets.

While many defenses were broken by [4], PGD-based adversarial training was among the few that withstood strong attacks. Many other defenses build on PGD adversarial training or leverage PGD adversarial generation during training. Examples include Adversarial Logit Pairing (ALP) [95], Feature Denoising [220], Defensive Quantization [121], Thermometer Encoding [18], PixelDefend [184], Robust Manifold Defense [87], L2-nonexpansive nets [154], Jacobian Regularization [90], Universal Perturbation [172], and Stochastic Activation Pruning [42].

We focus on the min-max formulation of adversarial training [129], which has been theoretically and empirically justified. This widely used K -PGD adversarial training algorithm has an inner loop that constructs adversarial examples by PGD- K , while the outer loop updates the model using minibatch SGD on the generated examples. In the inner loop, the gradient $\nabla_x l(x_{adv}, y, \theta)$ for updating adversarial examples requires a forward-backward pass of the entire network, which has similar computation cost as calculating the gradient $\nabla_\theta l(x_{adv}, y, \theta)$ for updating network parameters. Compared to natural training, which only requires $\nabla_\theta l(x, y, \theta)$ and does not have an inner loop, K -PGD adversarial training needs roughly $K + 1$ times more computation.

3.4 “Free” adversarial training

K -PGD adversarial training [129] is generally slow. For example, the 7-PGD training of a WideResNet [231] on CIFAR-10 in [129] takes about four days on a Titan X GPU. To scale the algorithm to ImageNet, [220] and [95] had to deploy large GPU clusters at a scale far beyond the reach of most organizations.

Here, we propose free adversarial training, which has a negligible complexity overhead compared to natural training. Our free adversarial training algorithm (alg. ??) computes the ascent step by re-using the backward pass needed for the descent step. To update the network parameters, the current training minibatch is passed forward through the network. Then, the gradient with respect to the network parameters is computed on the backward pass. When the “free” method is used, the gradient of the loss with respect to the input image is also computed on this same backward pass.

Unfortunately, this approach does not allow for multiple adversarial updates to be made to the same image without performing multiple backward passes. To overcome this restriction, we propose a minor yet nontrivial modification to training: train on the same minibatch m times in a row. Note that we divide the number of epochs by m such that the overall number of training iterations remains constant. This strategy provides multiple adversarial updates to each training image, thus providing strong/iterative adversarial examples. Finally, when a new minibatch is formed, the perturbation generated on the previous minibatch is used to warm-start the perturbation for the new minibatch.

The effect of mini-batch replay on natural training

While the hope for alg. ?? is to build robust models, we still want models to perform well on natural examples. As we increase m in alg. ??, there is risk of increasing generalization error. Furthermore, it may be possible that catastrophic forgetting happens. Consider the worst case where all the “informative” images of one class are in the first few mini-batches. In this extreme case, we do not see useful examples for most of the epoch, and forgetting may occur. Consequently, a natural question is: how much does mini-batch replay hurt generalization?

To answer this question, we naturally train wide-resnet 32-10 models on CIFAR-10 and CIFAR-100 using different levels of replay. Fig. 3.1 plots clean validation accuracy as a function of the replay parameter m . We see some dropoff in accuracy for small values of m . Note that a small compromise in accuracy is acceptable given a large increase in robustness due to the fundamental tradeoffs between robustness and generalization [173, 202, 236]. As a reference, CIFAR-10 and CIFAR-100 models that are 7-PGD adversarially trained have natural accuracies of 87.25% and 59.87%, respectively. These same accuracies are exceeded by natural training with $m = 16$. We see in section 3.5 that good robustness can be achieved using “free” adversarial training with just $m \leq 10$.

3.5 Robust models on CIFAR-10 and 100

In this section, we train robust models on CIFAR-10 and CIFAR-100 using our “free” adversarial training (alg. ??) and compare them to K-PGD adversarial training. We find that free training is able to achieve state-of-the-art robustness on the CIFARs without the overhead of standard PGD training.

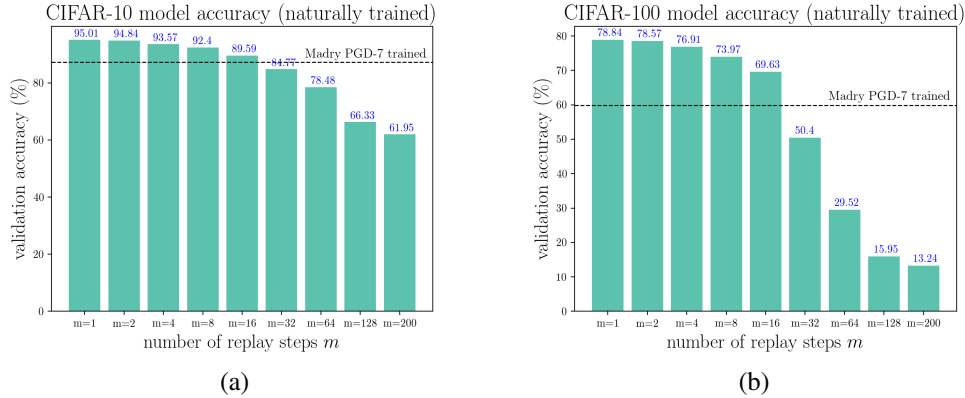


Figure 3.1: Natural validation accuracy of Wide Resnet 32-10 models using varied mini-batch replay parameters m . Here $m = 1$ corresponds to natural training. For large m 's, validation accuracy drops drastically. However, small m 's have little effect. For reference, CIFAR-10 and CIFAR-100 models that are 7-PGD adversarially trained have natural accuracies of 87.25% and 59.87%, respectively.

CIFAR-10

We train various CIFAR-10 models using the Wide-Resnet 32-10 model and standard hyper-parameters used by [129]. In the proposed method (alg. ??), we repeat (*i.e.* replay) each minibatch m times before switching to the next minibatch. We present the experimental results for various choices of m in table 3.1. Training each of these models costs roughly the same as natural training since we preserve the same number of iterations. We compare with the 7-PGD adversarially trained model from [129]², whose training requires roughly $7\times$ more time than all of our free training variations. We attack all models using PGD attacks with K iterations on both the cross-entropy loss (PGD- K) and the Carlini-Wagner loss (CW- K) [20]. We test using the PGD-20 attack following [129], and also increase the number of attack iterations and employ random restarts to verify robustness under stronger attacks. Note that gradient free-attacks such as SPSA will result in inferior results for adversarially trained models in comparison to optimization based attacks such as PGD as noted by [203]. Gradient-free attacks are superior in settings where

²Results based on the “adv_trained” model in Madry’s CIFAR-10 challenge repo.

Table 3.1: Validation accuracy and robustness of CIFAR-10 models trained with various methods.

Training	Evaluated Against					Train Time (min)
	Nat. Images	PGD-20	PGD-100	CW-100	10 restart PGD-20	
Natural	95.01%	0.00%	0.00%	0.00%	0.00%	780
Free $m = 2$	91.45%	33.92%	33.20%	34.57%	33.41%	816
Free $m = 4$	87.83%	41.15%	40.35%	41.96%	40.73%	800
Free $m = 8$	85.96%	46.82%	46.19%	46.60%	46.33%	785
Free $m = 10$	83.94%	46.31%	45.79%	45.86%	45.94%	785
7-PGD trained	87.25%	45.84%	45.29%	46.52%	45.53%	5418

Table 3.2: Validation accuracy and robustness of CIFAR-100 models trained with various methods.

Training	Evaluated Against			Training Time (minutes)
	Natural Images	PGD-20	PGD-100	
Natural	78.84%	0.00%	0.00%	811
Free $m = 2$	69.20%	15.37%	14.86%	816
Free $m = 4$	65.28%	20.64%	20.15%	767
Free $m = 6$	64.87%	23.68%	23.18%	791
Free $m = 8$	62.13%	25.88%	25.58%	780
Free $m = 10$	59.27%	25.15%	24.88%	776
Madry et al. (2-PGD trained)	67.94%	17.08%	16.50%	2053
Madry et al. (7-PGD trained)	59.87%	22.76%	22.52%	5157

the defense works by masking or obfuscating the gradients.

Our “free training” algorithm successfully reaches robustness levels comparable to a 7-PGD adversarially trained model. As we increase m , the robustness is increased at the cost of validation accuracy on natural images. Additionally note that we achieve reasonable robustness over a wide range of choices of the main hyper-parameter of our model, $10 \geq m > 2$, and the proposed method is significantly faster than 7-PGD adversarial training.

CIFAR-100

We also study the robustness results of “free training” on CIFAR-100 which is a more difficult dataset with more classes. As we will see in sec. 3.4, training with large m values on this

dataset hurts the natural validation accuracy more in comparison to CIFAR-10. This dataset is less studied in the adversarial machine learning community and therefore for comparison purposes, we adversarially train our own Wide ResNet 32-10 models for CIFAR-100. We train two robust models by varying K in the K -PGD adversarial training algorithm. One is trained on PGD-2 with a computational cost almost $3\times$ that of free training, and the other is trained on PGD-7 with a computation time roughly $7\times$ that of free training. We adopt the code for adversarial training from [129], which produces state-of-the-art robust models on CIFAR-10. We summarize the results in table. 3.2.

We see that “free training” exceeds the accuracy on both natural images and adversarial images when compared to traditional adversarial training. Similar to the effect of increasing m , increasing K in K -PGD adversarial training results in increased robustness at the cost of clean validation accuracy. However, unlike the proposed “free training” where increasing m has no extra cost, increasing K for standard K -PGD substantially increases training time.

3.6 Does “free” training behave like standard adversarial training?

Here, we analyze two properties that are associated with PGD adversarially trained models: The interpretability of their gradients and the flatness of their loss surface. We find that “free” training enjoys these benefits as well.

Generative behavior for largely perturbed examples

[202] observed that hardened classifiers have interpretable gradients; adversarial examples built for PGD trained models often look like the class into which they get misclassified. Fig. 3.2



Figure 3.2: Attack images built for adversarially trained models look like the class into which they get misclassified. We display the last 9 CIFAR-10 clean validation images (top row) and their adversarial examples built for a 7-PGD adversarially trained (middle) and our “free” trained (bottom) models.

plots “weakly bounded” adversarial examples for the CIFAR-10 7-PGD adversarially trained model [129] and our free $m = 8$ trained model. Both models were trained to resist ℓ_∞ attacks with $\epsilon = 8$. The examples are made using a 50 iteration BIM attack with $\epsilon = 30$ and $\epsilon_s = 2$. “Free training” maintains generative properties, as our model’s adversarial examples resemble the target class.

Smooth and flattened loss surface

Another property of PGD adversarial training is that it flattens and smoothens the loss landscape. In contrast, some defenses work by “masking” the gradients, i.e., making it difficult to identify adversarial examples using gradient methods, even though adversarial examples remain present. Reference [49] argues that gradient masking adds little security. We show in fig. 3.6 that free training does not operate by masking gradients using a rough loss surface. In fig. 3.3 we plot the cross-entropy loss projected along two directions in image space for the first few validation

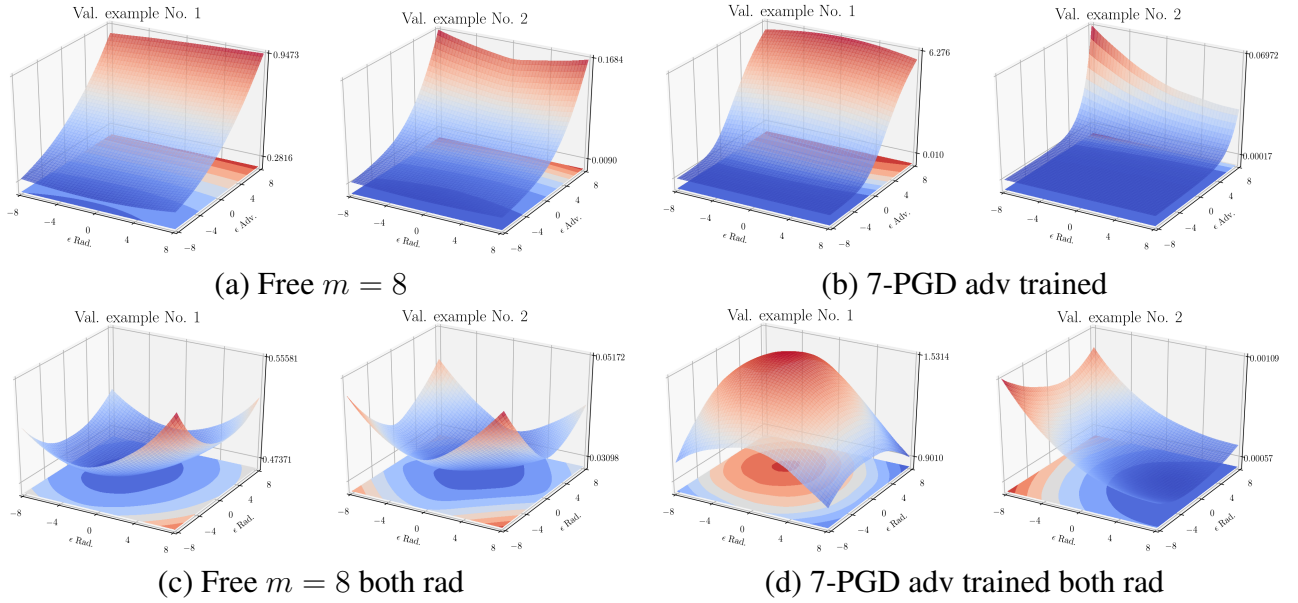


Figure 3.3: The loss surface of a 7-PGD adversarially trained model and our “free” trained model for CIFAR-10 on the first 2 validation images. In (a) and (b) we display the cross-entropy loss projected on one random (Rademacher) and one adversarial direction. In (c) and (d) we display the the cross entropy loss projected along two random directions. Both training methods behave similarly and do not operate by masking the gradients as the adversarial direction is indeed the direction where the cross-entropy loss changes the most.

examples of CIFAR-10 [117]. In addition to the loss of the free $m = 8$ model, we plot the loss of the 7-PGD adversarially trained model for comparison.

3.7 Robust ImageNet classifiers

ImageNet is a large image classification dataset of over 1 million high-res images and 1000 classes ([166]). Due to the high computational cost of ImageNet training, only a few research teams have been able to afford building robust models for this problem. [104] first hardened ImageNet classifiers by adversarial training with non-iterative attacks.³ Adversarial training was done using a targeted FGSM attack. They found that while their model became robust against targeted non-iterative attacks, the targeted BIM attack completely broke it.

³Training using a non-iterative attack such as FGSM only doubles the training cost.

Later, [95] attempted to train a robust model that withstands targeted PGD attacks. They trained against 10 step PGD targeted attacks (a process that costs 11 times more than natural training) to build a benchmark model. They also generated PGD targeted attacks to train their adversarial logit paired (ALP) ImageNet model. Their baseline achieves a top-1 accuracy of 3.1% against PGD-20 targeted attacks with $\epsilon = 16$. Very recently, [220] trained a robust ImageNet model against targeted PGD-30 attacks, with a cost $31\times$ that of natural training. Training this model required a distributed implementation on 128 GPUs with batch size 4096. Their robust ResNet-101 model achieves a top-1 accuracy of 35.8% on targeted PGD attacks with many iterations.

Free training results

Our alg. ?? is designed for non-targeted adversarial training. As [4] state, defending on this task is important and more challenging than defending against targeted attacks, and for this reason smaller ϵ values are typically used. Even for $\epsilon = 2$ (the smallest ϵ we consider defending against), a PGD-50 non-targeted attack on a natural model achieves roughly 0.05% top-1 accuracy. To put things further in perspective, [203] broke three defenses for $\epsilon = 2$ non-targeted attacks on ImageNet [68, 120, 219], degrading their performance below 1%. Our free training algorithm is able to achieve 43% robustness against PGD attacks bounded by $\epsilon = 2$. Furthermore, we ran each experiment on a single workstation with four P100 GPUs. Even with this modest setup, training time for each ResNet-50 experiment is below 50 hours.

We summarize our results for various ϵ 's and m 's in table 3.3 and fig. ?.?. To craft attacks, we used a step-size of 1 and the corresponding ϵ used during training. In all experiments, the training batch size was 256. Table 3.3 shows the robustness of Resnet-50 on ImageNet with

Table 3.3: ImageNet validation accuracy and robustness of ResNet-50 models trained with various replay parameters and $\epsilon = 2$.

Training	Evaluated Against			
	Natural Images	PGD-10	PGD-50	PGD-100
Natural	76.038%	0.166%	0.052%	0.036%
Free $m = 2$	71.210%	37.012%	36.340%	36.250%
Free $m = 4$	64.446%	43.522%	43.392%	43.404%
Free $m = 6$	60.642%	41.996%	41.900%	41.892%
Free $m = 8$	58.116%	40.044%	40.008%	39.996%

$\epsilon = 2$. The validation accuracy for natural images decreases when we increase the minibatch replay m , just like it did for CIFAR in section 3.5.

The naturally trained model is vulnerable to PGD attacks (first row of table 3.3), while free training produces robust models that achieve over 40% accuracy vs PGD attacks ($m = 4, 6, 8$ in table 3.3). Attacking the models using PGD-100 does not result in a meaningful drop in accuracy compared to PGD-50. Therefore, we did not experiment with increasing the number of PGD iterations further.

Fig. ?? summarizes experimental results for robust models trained and tested under different perturbation bounds ϵ . Each curve represents one training method (natural training or free training) with hyperparameter choice m . Each point on the curve represents the validation accuracy for an ϵ -bounded robust model. These results are also provided as tables in the appendix. The proposed method consistently improves the robust accuracy under PGD attacks for $\epsilon = 2-7$, and $m = 4$ performs the best. It is difficult to train robust models when ϵ is large, which is consistent with previous studies showing that PGD-based adversarial training has limited robustness for ImageNet [95].

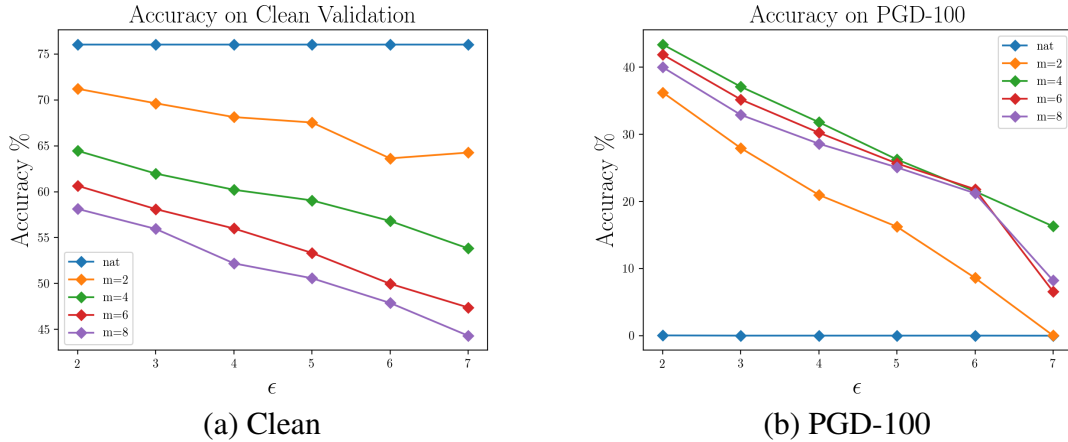


Table 3.4: Validation accuracy and robustness of “free” and 2-PGD trained ResNet-50 models – both trained to resist $\ell_\infty \epsilon = 4$ attacks. Note that **2-PGD training time is $3.46\times$ that of “free” training.**

Model & Training	Evaluated Against				Train time (minutes)
	Natural Images	PGD-10	PGD-50	PGD-100	
RN50 – Free $m = 4$	60.206%	32.768%	31.878%	31.816%	3016
RN50 – 2-PGD trained	64.134%	37.172%	36.352%	36.316%	10,435

Comparison with PGD-trained models

We compare “free” training to a more costly method using 2-PGD adversarial examples $\epsilon = 4$. We run the conventional adversarial training algorithm and set $\epsilon_s = 2$, $\epsilon = 4$, and $K = 2$. All other hyper-parameters were identical to those used for training our “free” models. Note that in our experiments, we do not use any label-smoothing or other common tricks for improving robustness since we want to do a fair comparison between PGD training and our “free” training. These extra regularizations can likely improve results for both approaches.

We compare our “free trained” $m = 4$ ResNet-50 model and the 2-PGD trained ResNet-50 model in table 3.4. 2-PGD adversarial training takes roughly $3.4\times$ longer than “free training” and only achieves slightly better results ($\approx 4.5\%$). This gap is less than 0.5% if we free train a higher capacity model (i.e. ResNet-152, see below).

Table 3.5: Validation accuracy and robustness of free- $m = 4$ trained ResNets with various capacities.

Architecture	Evaluated Against			
	Natural Images	PGD-10	PGD-50	PGD-100
ResNet-50	60.206%	32.768%	31.878%	31.816%
ResNet-101	63.340%	35.388%	34.402%	34.328%
ResNet-152	64.446%	36.992%	36.044%	35.994%

Free training on models with more capacity

It is believed that increased network capacity leads to greater robustness from adversarial training [104, 129]. We verify that this is the case by “free training” ResNet-101 and ResNet-152 with $\epsilon = 4$. The comparison between ResNet-152, ResNet-101, and ResNet-50 is summarized in table 3.5. Free training on ResNet-101 and ResNet-152 each take roughly $1.7\times$ and $2.4\times$ more time than ResNet-50 on the same machine, respectively. The higher capacity model enjoys a roughly 4% boost to accuracy and robustness.

3.8 Conclusions

Adversarial training is a well-studied method that boosts the robustness and interpretability of neural networks. While it remains one of the few effective ways to harden a network to attacks, few can afford to adopt it because of its high computation cost. We present a “free” version of adversarial training with cost nearly equal to natural training. Free training can be further combined with other defenses to produce robust models without a slowdown. We hope that this approach can put adversarial training within reach for organizations with modest compute resources.

Chapter 4: Breaking Certified Defenses

Conventional training of neural networks has been shown to produce classifiers that are highly sensitive to adversarial perturbations [15, 191], “natural looking” images that have been manipulated to cause misclassification by a neural network (Figure 7.1). While a wide range of defenses exist that harden neural networks against such attacks [129, 174], defenses based on heuristics and tricks are often easily breakable [4]. This has motivated work on certifiably secure networks — classifiers that produce a label for an image, and also (when possible) a rigorous guarantee that the input is not adversarially manipulated [29, 238].

To date, all work on certifiable defenses has focused on deflecting ℓ_p -bounded attacks, where $p = 2$ or ∞ [29, 64, 212]. After labelling an image, these defenses then check whether there exists an image of a different label within ϵ units (in the ℓ_p metric) of the input, where ϵ is a security parameter chosen by the user. If the classifier assigns all images within the ϵ ball the same class label, then a certificate is issued, and the input image known not to be an ℓ_p adversarial example.

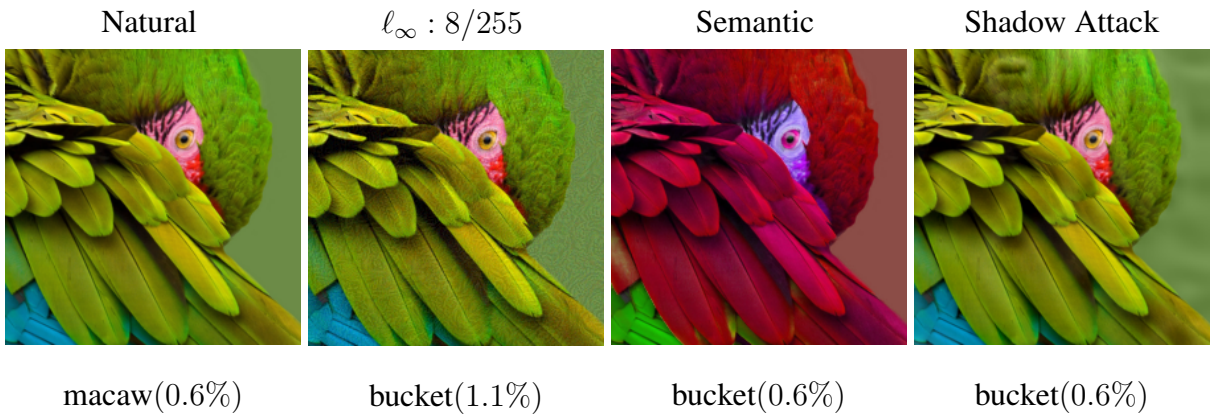


Figure 4.1: All adversarial examples have the goal of fooling classifiers while looking “natural”. Standard attacks limit the ℓ_p -norm of the perturbation, while semantic attacks have large ℓ_p -norm while still producing natural looking images. Our attack produces large but visually subtle semantic perturbations that not only cause misclassification, but also cause a certified defense to issue a “spoofed” high-confidence certificate.

In this work, we demonstrate how a system that relies on certificates as a measure of label security can be exploited. We present a new class of adversarial examples that target not only the classifier output label, but also the certificate. We do this by adding adversarial perturbations to images that are large in the ℓ_p norm (larger than the ϵ used by the certificate generator), and produce attack images that are surrounded by a large ball exclusively containing images of the same label. The resulting attacks produce a “spoofed” certificate with a seemingly strong security guarantee despite being adversarially manipulated. Note that the statement made by the certificate (i.e., that the input image is not an ϵ adversarial example in the chosen norm) is still technically correct, however in this case the adversary is hiding behind a certificate to avoid detection by a certifiable defense.

In summary, we consider methods that attack a certified classifier in the following sense:

- **Imperceptibility:** the adversarial example “looks like” its corresponding natural base example.
- **Misclassification:** the certified classifier assigns an incorrect label to the adversarial example.
- **Strongly certified:** the certified classifier provides a strong/large-radius certificate for the adversarial example.

While the existence of such an attack does not invalidate the certificates produced by certifiable systems, it should serve as a warning that certifiable defenses are not inherently secure, and one should take caution when relying on certificates as an indicator of label correctness.

Background

In the white-box setting, where the attacker knows the victim’s network and parameters, adversarial perturbations are often constructed using first-order gradient information [19, 105, 136] or using approximations of the gradient [4, 203]. The prevailing formulation for crafting attacks uses an additive adversarial perturbation, and perceptibility is minimized using an ℓ_p -norm constraint. For example, ℓ_∞ -bounded attacks limit how much each pixel can move, while ℓ_0 adversarial attacks limit the number of pixels that can be modified, without limiting the size of the change to each pixel [209].

It is possible to craft imperceptible attacks without using ℓ_p bounds [17]. For example, Hosseini and Poovendran [82] use shifting color channels, and Engstrom et al. [48] use rotation and translation to craft “semantic” adversarial examples. In Figure 7.1, we produce semantic adversarial examples using the method of Hosseini and Poovendran [82] which is a greedy approach that transforms the image into HSV space, and then, while keeping V constant, tries to find the smallest S perturbation causing misclassification¹. Other variants use generative models to construct natural looking images causing misclassification [47, 185].

In practice, many of the defenses which top adversarial defense leader-board challenges are non-certified defenses [129, 174, 236]. The majority of these defenses make use of adversarial training, in which attack images are crafted during training and injected into the training set. These non-certified defenses are mostly evaluated against PGD-based attacks, resulting in an upper-bound on robustness.

Certified defenses, on the other-hand, provably make networks resist ℓ_p -bounded perturbations

¹In fig. 7.1, the adversarial example has saturation=0

of a certain radius. For instance, randomized smoothing [29] is a certifiable defense against ℓ_2 -norm bounded attacks, and CROWN-IBP [238] is a certifiable defense against l_∞ -norm bounded perturbations. Both of these defenses produce a class label, and also a guarantee that the image could not have been crafted by making small perturbations to an image of a different label.

To the best of our knowledge, prior works have focused on making adversarial examples that satisfy the imperceptibility and misclassification conditions, but none have investigated manipulating certificates, which is our focus here.

The reminder of this paper is organized as follows. In section 4.1 we introduce our new approach Shadow Attack for generating adversarial perturbations. This is a hybrid model that allows various kinds of attacks to be compounded together, resulting in perturbations of large radii. In section 4.2 we present an attack on “randomized smoothing” certificates [29]. Section 4.3 shows an ablation study and show why the elements of the Shadow Attack are important for successfully manipulating certified models. In section 4.4 we generate adversarial examples for “CROWN-IBP” [238]. Finally, we discuss results and wrap up in section 4.5.

4.1 The Shadow Attack

Because certificate spoofing requires large perturbations (larger than the ℓ_p ball of the certificate), we propose a simple attack that ensembles numerous modes to create large perturbations. Our attack can be seen as the generalization of the well-known PGD attack, which creates adversarial images by modifying a clean base image. Given a loss function L and an ℓ_p -norm

bound ϵ for some $p \geq 0$, PGD attacks solve the following optimization problem:

$$\max_{\delta} L(\theta, x + \delta) \quad (4.1)$$

$$s.t. \quad \|\delta\|_p \leq \epsilon, \quad (4.2)$$

where θ are the network parameters and δ is the adversarial perturbation to be added to the clean input image x . Constraint 4.2 promotes imperceptibility of the resulting perturbation to the human eye by limiting the perturbation size. In the shadow attack, instead of solving the above constrained optimization problem, we solve the following problem with a range of penalties:

$$\max_{\delta} L(\theta, x + \delta) - \lambda_c C(\delta) - \lambda_{tv} \text{TV}(\delta) - \lambda_s \text{Dissim}(\delta), \quad (4.3)$$

where $\lambda_c, \lambda_{tv}, \lambda_s$ are scalar penalty weights. Penalty $\text{TV}(\delta)$ forces the perturbation δ to have small total variation (TV), and so appear more smooth and natural. Penalty $C(\delta)$ limits the perturbation δ globally by constraining the change in the mean of each color channel c . This constraint is needed since total variation is invariant to constant/scalar additions to each color channel, and it is desirable to suppress extreme changes in the color balances of images.

Penalty $\text{Dissim}(\delta)$ promotes perturbations δ that assume similar values in each color channel. In the case of an RGB image of shape $3 \times W \times H$, if $\text{Dissim}(\delta)$ is small, the perturbations to red, green, and blue channels are similar, i.e., $\delta_{R,w,h} \approx \delta_{G,w,h} \approx \delta_{B,w,h}, \forall (w, h) \in W \times H$. This amounts to making the pixels darker/lighter, without changing the color balance of the image. Later, in section 4.2, we suggest two ways of enforcing such similarity between RGB channels and we find both of them effective:

- **1-channel attack** strictly enforces $\delta_{R,i} \approx \delta_{G,i} \approx \delta_{B,i}, \forall i$ by using just one array to simultaneously represent each color channel $\delta_{W \times H}$. On the forward pass, we duplicate δ to make a 3-channel image. In this case, $\text{Dissim}(\delta) = 0$, and the perturbation is greyscale.
- **3-channel attack** uses a 3-channel perturbation $\delta_{3 \times W \times H}$, along with the dissimilarity metric $\text{Dissim}(\delta) = \|\delta_R - \delta_B\|_p + \|\delta_R - \delta_G\|_p + \|\delta_B - \delta_G\|_p$.

All together, the three penalties minimize perceptibility of perturbations by forcing them to be (a) small, (b) smooth, and (c) without dramatic color changes (e.g. swapping blue to red). At the same time, these penalties allow perturbations that are very large in ℓ_p -norm.

4.1.1 Creating untargeted attacks

We focus on spoofing certificates for untargeted attacks, in which the attacker does not specify the class into which the attack image moves. To achieve this, we generate an adversarial perturbation for all possible wrong classes $\bar{y}$ and choose the best one as our strong attack:

$$\max_{\bar{y} \neq y, \delta} -L(\theta, x + \delta \| \bar{y}) - \lambda_c C(\delta) - \lambda_{tv} \text{TV}(\delta) - \lambda_s \text{Dissim}(\delta), \quad (4.4)$$

where y is the true label/class for the clean image x , and L is a spoofing loss that promotes a large certificate. We examine different choices for L for different certificates below.

4.2 Attacks on Randomized Smoothing

The randomized Smoothing method, first proposed by Lecuyer et al. [113] and later improved by Li et al. [115], is an adversarial defense against l_2 -norm bounded attacks. Cohen et al. [29]

prove a tight robustness guarantee under the ℓ_2 norm for smoothing with gaussian noise. Their study was the first certifiable defense for the ImageNet dataset [39]. The method constructs certificates by first creating many copies of an input image contaminated with random Gaussian noise of standard deviation σ . Then, it uses a base classifier (a neural net) to make a prediction for all of the images in the randomly augmented batch. Depending on the level of consensus among the class labels of these random images, a certified radius is calculated that can be at most 4σ (in the case of perfect consensus).

Intuitively, if the image is far away from the decision boundary, the base classifier should predict the same label for each noisy copy of the test image, in which case the certificate is strong. On the other hand, if the image is adjacent to the decision boundary, the base classifier’s predictions for the Gaussian augmented copies may vary. If the variation is large, the smoothed classifier abstains from making a prediction.

To spoof strong certificates (large certified radius) for an incorrect class, we must make sure that the majority of a batch of noisy images around the adversarial image are assigned the same (wrong) label. We do this by minimizing the cross entropy loss relative to a chosen (incorrect) label, averaged over a large set of randomly perturbed images.

To this end, we minimize equation 4.4, where L is chosen to be the average cross-entropy over a batch of Gaussian perturbed copies. This method is analogous to the technique presented by [172] for generating universal perturbations that are effective when added to a large number of different images.

Results

Cohen et al. [29] show the performance of the Gaussian smoothed classifier on CIFAR-10 [100] and ImageNet [39]. To attack the CIFAR and ImageNet smoothed classifiers, we use 400 randomly sampled Gaussian images, $\lambda_{tv} = 0.3$, $\lambda_c = 1.0$, and perform 300 steps of SGD with learning rate 0.1. We choose the functional regularizers $C(\delta)$ and $TV(\delta)$ to be

$$C(\delta) = \|\text{Avg}(|\delta_R|), \text{Avg}(|\delta_G|), \text{Avg}(|\delta_B|)\|_2^2, \quad \text{and} \quad TV(\delta_{i,j}) = \text{anisotropic-TV}(\delta_{i,j})^2,$$

where $|\cdot|$ is the element-wise absolute value operator, and Avg computes the average. For the Dissim regularizer, we experiment with both the 1-Channel attack that ensures $\text{Dissim}(\delta) = 0$, and the 3-Channel attack by setting $\text{Dissim}(\delta) = \|(\delta_R - \delta_G)^2, (\delta_R - \delta_B)^2, (\delta_G - \delta_B)^2\|_2$ and $\lambda_s = 0.5$. For the validation examples on which the smoothed classifier does not abstain (see Cohen et al. [29] for more details), the less-constrained 3-channel attack is always able to find an adversarial example while the 1-channel attack also performs well, achieving 98.5% success. In section 4.3 we will discuss in more detail other differences between 1-channel and 3-channel attacks. The results are summarized in Table 4.1. For the various base-models and choices of σ , our adversarial examples are able to produce certified radii that are on average larger than the certified radii produced for natural images. For ImageNet, since attacking all 999 remaining target classes is computationally expensive, we only attacked target class IDs 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1000.

Figure 4.2 depicts a sample adversarial example built for the smoothed ImageNet classifier that produces a large certificate. The adversarial perturbation causes the batch of gaussian augmented

Table 4.1: Certified radii produced by the Randomized Smoothing method for Shadow Attack images and also natural images (larger radii means a stronger/more confident certificate).

Dataset	$\sigma(l_2)$	Randomized Smoothed		Shadow Attack	
		Mean	STD	Mean	STD
CIFAR-10	0.12	0.14	0.056	0.22	0.005
	0.25	0.30	0.111	0.35	0.062
	0.50	0.47	0.234	0.65	0.14
	1.00	0.78	0.556	0.85	0.442
ImageNet	0.25	0.30	0.109	0.31	0.109
	0.50	0.61	0.217	0.38	0.191
	1.00	1.04	0.519	0.64	0.322

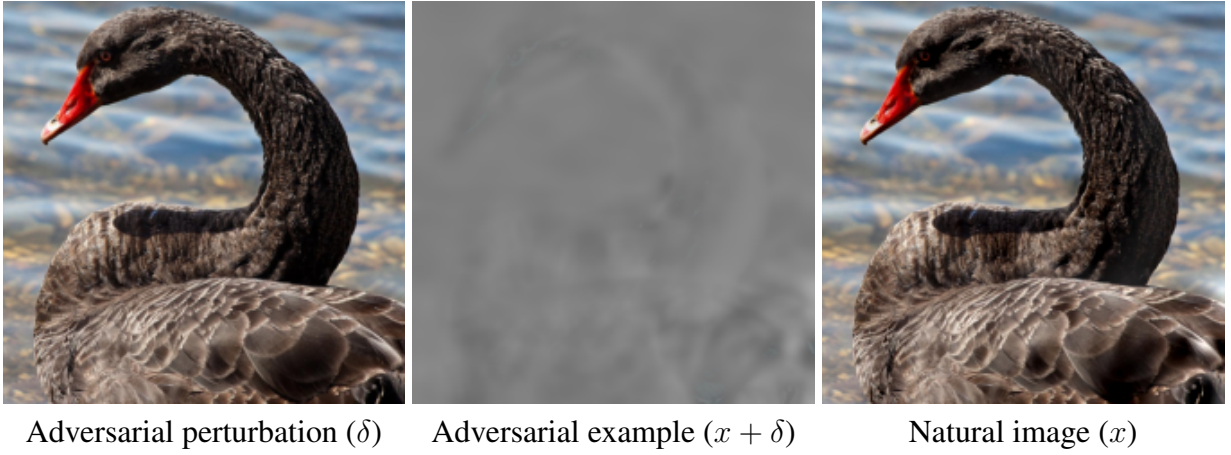


Figure 4.2: An adversarial example built using our Shadow Attack for the smoothed ImageNet classifier for which the certifiable classifier produces a large certified radii. The adversarial perturbation is smooth and natural looking even-though it is large when measured using ℓ_p -metrics. Also see appendix 11.4.

black swan images to get misclassified as hooks. For more, see appendix 11.4.

4.3 Ablation study of the attack parameters

In this section we perform an ablation study on the parameters of the Shadow Attack to evaluate (i) the number of SGD steps needed, (ii) the importance of λ_s (or alternatively using 1-channel attacks), and (iii) the effect of λ_{tv} .

The default parameters for all of the experiments are as follows unless explicitly mentioned: We use 30 SGD steps with learning rate 0.1 for the optimization. All experiments except part (b) use 1-channel attacks for the sake of simplicity and efficiency (since it has less parameters). We assume $\lambda_{tv} = 0.3$, $\lambda_c = 20$, and use batch-size 50. We present results using the first example from each class of the CIFAR-10 validation set.

Figure 4.3 shows how the adversarial example evolves during the first few steps of optimization (See appendix 11.1 for more examples). Also, figures 4.4, 4.5, and 4.6 show the evolution of $L(\delta)$, $TV(\delta)$, and $C(\delta)$, respectively (Note that we use 1-channel attacks, so $\text{Dissim}(\delta)$ is always 0). We find that taking just 10 SGD steps is enough for convergence on CIFAR-10, but for our main results (i.e. attacking Randomized Smoothing in section 4.2 and attacking CROWN-IBP in section 4.4) we take 300 steps to be safe.

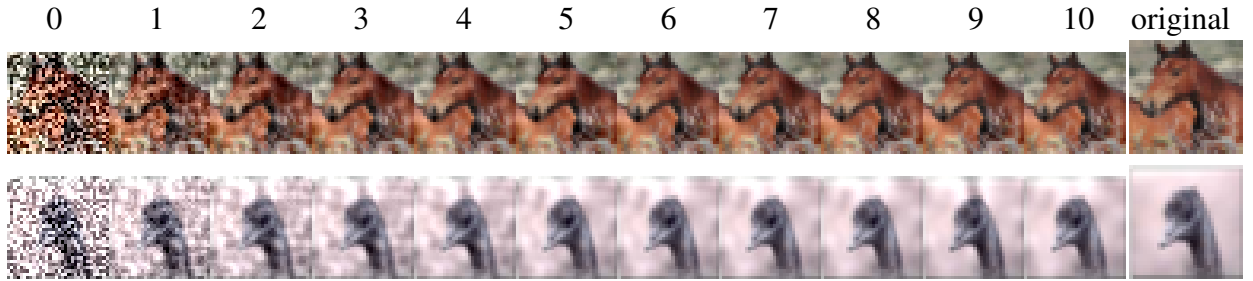


Figure 4.3: The first 10 steps of optimization (beginning with a randomly perturbed image copy).

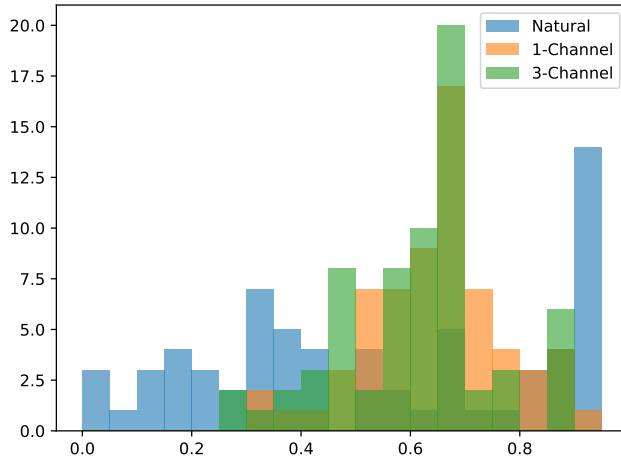


Figure 4.9: Histogram of randomized smoothed certificate radii for 100 randomly sampled CIFAR-10 validation images vs those calculated for their adversarial examples crafted using our 1-channel and 3-channel adversarial Shadow Attack attacks. The “robust” victim classifier is based off Resnet-110, and smoothed with $\sigma = 0.50$. 1-channel attacks are almost as good as the less-restricted 3-channel attacks.

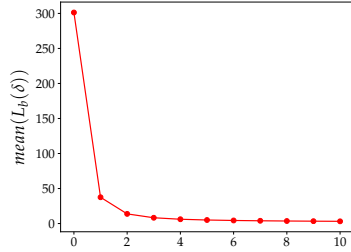


Figure 4.4: Average $L_b(\delta)$ in the first 10 steps.

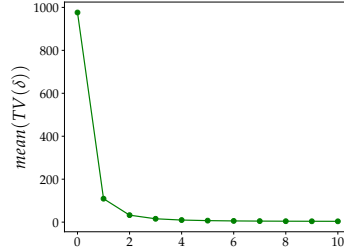


Figure 4.5: Average $TV(\delta)$ in the first 10 steps.

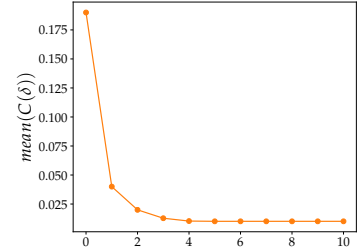


Figure 4.6: Average $C(\delta)$ in the first 10 steps.

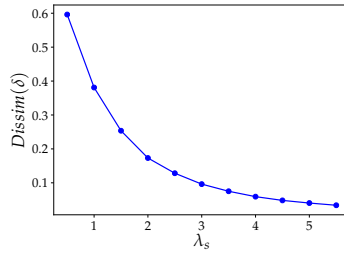


Figure 4.7: The effect of λ_s on the resulting $Dissim(\delta)$.

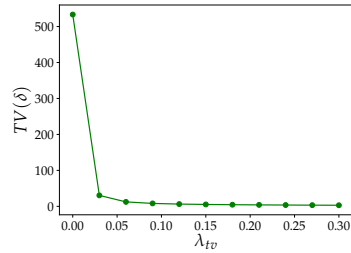


Figure 4.8: The effect of λ_{tv} on the resulting $TV(\delta)$.

To explore the importance of λ_s , we use 3-channel attacks and vary λ_s to produce different images in figure 4.11².

Also, figure 4.7 shows the mean $\text{Dissim}(\delta)$ for different values of λ_s ($0 \leq \lambda_s \leq 5.0$). We also plot the histogram of the certificate radii in figure 4.9. Figure 4.10 compares 1-Channel vs 3-Channel attacks for some of randomly selected CIFAR-10 images. Finally, we explore the effect of λ_{tv} on imperceptibility of the perturbations in Figure 4.12. See table 11.3 for more images, and figure 4.8 for the impact of parameters on $\text{TV}(\delta)$.

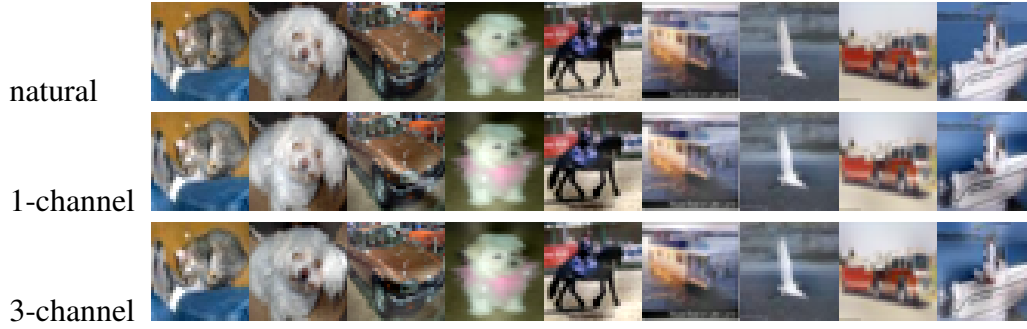


Figure 4.10: The visual effect of Shadow Attack on 9 randomly selected CIFAR-10 examples using 1-Channel and 3-Channel attacks.

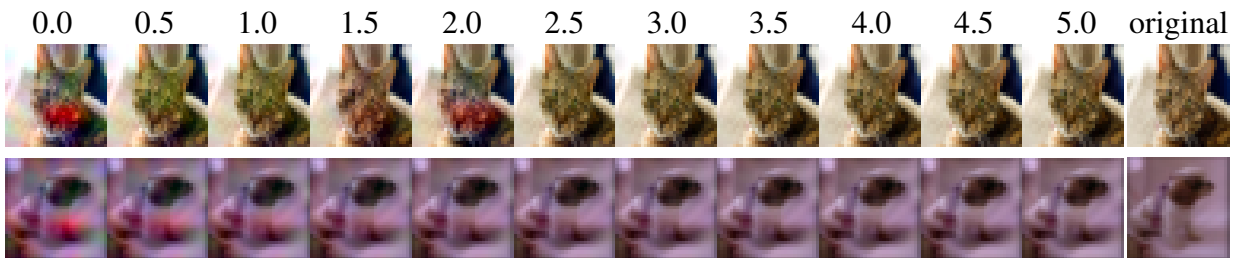


Figure 4.11: The visual effect of λ_s on imperceptibility of the perturbations. The first row indicates the value of λ_s .

²See appendix 11.2 for more images.



Figure 4.12: The visual effect of λ_{tv} on the perceptibility of the perturbations. The first row shows the value of λ_{tv} .

4.4 Attacks on CROWN-IBP

Interval Bound Propagation (IBP) methods have been recently studied as a defense against ℓ_∞ -bounded attacks. Many recent studies such as Gowal et al. [64], Mirman et al. [135], Wong et al. [212], Xiao et al. [218] have investigated IBP methods to train provably robust networks. To the best of our knowledge, the CROWN-IBP method by Zhang et al. [238] achieves state-of-the-art performance for MNIST [108], Fashion-MNIST [217], and CIFAR-10 datasets among certifiable ℓ_∞ defenses. In this section we focus on attacking Zhang et al. [238] using CIFAR-10.

IBP methods (over)estimate how much a small ℓ_∞ -bounded noise in the input can impact the classification layer. This is done by propagating errors from layer to layer, computing bounds on the maximum possible perturbation to each activation. During testing, the user chooses an ℓ_∞ perturbation bound ϵ , and error propagation is used to bound the magnitude of the largest achievable perturbation in network output. If the output perturbation is not large enough to flip the image label, then a certificate is produced. If the output perturbation is large enough to flip the image label, then a certificate is not produced. During network training, IBP methods include a term in the loss function that promotes tight error bounds that result in certificates.

Our attack directly uses the loss function term used during IBP network training in addition to the Shadow Attack penalties; we search for an image that is visually similar to the base image, while producing a bound on the output perturbation that is too small to flip the image label.

We attack 4 classes of networks released by [238] for CIFAR-10. There are two classes of IBP architectures, one of them consists of 9 small models and the other consists of 8 larger models. For each class of architectures, there are two sets of pre-trained models: one for $\epsilon = 2/255$ and one for $\epsilon = 8/255$. We use the same hyper-parameters and regularizers as in 4.2. For the sake of efficiency, we only do 1-channel attacks. We attack the 4 classes of models and for each class, we report the min, mean, and max of the robustness errors and compare them with those of the CROWN-IBP paper.

To quantify the success of our attack, we define two metrics of error. For natural images, we report the rate of “robust errors,” which are images that are either (i) incorrectly labeled, or (ii) correctly labelled but without a certificate. In contrast, for attack images, we report the rate of “attack errors,” which are either (i) correctly classified or (ii) incorrectly classified but without a certificate. Table 4.2 shows the robust error on natural images, and the attack error on Shadow Attackimages for the CROWN-IBP models. With $\epsilon = 2/255$, our attack finds adversarial examples that certify roughly 15% of the time (i.e., attack error $< 85\%$). With $\epsilon = 8/255$, our attack finds adversarial examples that are incorrectly classified, and yet certify even more often than natural images.

4.5 conclusion

We demonstrate that it is possible to produce adversarial examples with “spoofed” certified robustness by using large-norm perturbations. Our adversarial examples are built using a Shadow Attackthat produces smooth and natural looking perturbations that are often less perceptible than those of the commonly used ℓ_p -bounded perturbations, while being large enough in norm to

Table 4.2: “Robust error” for natural images, and “attack error” for Shadow Attack images using the CIFAR-10 dataset, and CROWN-IBP models.

$\epsilon(l_\infty)$	Model Family	Method	Robustness Errors		
			Min	Mean	Max
2/255	9 small models	CROWN-IBP	52.46	57.55	60.67
		Shadow Attack	68.91	74.77	82.09
	8 large models	CROWN-IBP	52.52	53.9	56.05
		Shadow Attack	79.7	82.28	85.09
8/255	9 small models	CROWN-IBP	71.28	72.15	73.66
		Shadow Attack	66.14	69.3	72.12
	8 large models	CROWN-IBP	70.79	71.17	72.29
		Shadow Attack	67.9	70.36	73.03

escape the certification regions of state of the art principled defenses. This work suggests that the certificates produced by certifiably robust classifiers, while mathematically rigorous, are not always good indicators of robustness or accuracy.

Chapter 5: Adaptive Weight Decay

We introduce adaptive weight decay, which automatically tunes the hyper-parameter for weight decay during each training iteration. For classification problems, we propose changing the value of the weight decay hyper-parameter on the fly based on the strength of updates from the classification loss (i.e., gradient of cross-entropy), and the regularization loss (i.e., ℓ_2 -norm of the weights). We show that this simple modification can result in large improvements in adversarial robustness — an area which suffers from robust overfitting — without requiring extra data. Specifically, our reformulation results in 20% relative robustness improvement for CIFAR-100, and 10% relative robustness improvement on CIFAR-10 comparing to traditional weight decay. In addition, this method has other desirable properties, such as less sensitivity to learning rate, and smaller weight norms, which the latter contributes to robustness to overfitting to label noise, and pruning.

5.1 Introduction

Modern deep learning models have exceeded human capability on many computer vision tasks. Due to their high capacity for memorizing training examples [234], their generalization heavily relies on the training algorithm. To reduce memorization, several approaches have been taken including regularization and augmentation. Some of these augmentation techniques alter the network input [25, 34, 35, 41, 140], some alter hidden states of the network [54, 88, 187, 225], some alter the expected output [95, 207], and some effect multiple levels [79, 229, 237]. Another popular approach to prevent overfitting is the use of regularizers, such as weight decay [103, 152]. Such methods prevent over-fitting by eliminating solutions that memorize training examples. Regularization methods are attractive beyond generalization on clean data as they are crucial in

adversarial and noisy-data settings. In this paper, we revisit weight decay; a regularizer mainly used to avoid overfitting.

The rest of the paper is organized as follows: In Section 5.2, we revisit tuning the hyper-parameters for weight decay. We introduce adaptive weight decay in Section 5.3, and further discuss its properties in 5.4. More specifically, we discuss the benefits of adaptive weight decay in the setting of adversarial training in subsection 5.4.1, noisy labels in subsection 5.4.2, and additional properties related to robustness in subsection 5.4.3.

5.2 Weight Decay

Weight decay which encourages weights of networks to have smaller magnitudes [235] has widely been adopted to improve generalization. Although other forms of weight decay have been studied [126], we focus on the popular ℓ_2 -norm variant. More precisely, we look at classification problems with cross-entropy as the main loss and weight decay as the regularizer that was popularized by [102]:

$$Loss_w(x, y) = CrossEntropy(w(x), y) + \frac{\lambda_{wd}}{2} \|w\|_2^2, \quad (5.1)$$

where w is the network parameters, and (x, y) is the training data, and λ_{wd} is the hyper-parameter controlling how much weight decay penalizes the norm of weights compared to the main loss (i.e., cross-entropy loss). For instance, if λ_{wd} is negligible, the optimization will likely over-fit data, whereas if λ_{wd} is too large, the optimization will collapse to a low-weighted solution that does not fit the training data. As a result, finding the correct value for the hyper-parameter of weight decay is crucial. Models trained using the right hyper-parameter for weight decay tend to

have higher bias, which translates to less over-fitting and, thus, better generalization [103].

5.2.1 Tuning Hyper-Parameters

When tuning λ_{wd} , it is crucial to simultaneously search for learning-rate lr as well¹ as illustrated in Figure 5.1 for CIFAR-10 and CIFAR-100. Otherwise, in the case that the 2D grid search is not computationally practical, we show that separate 1D grid search on learning rate and weight decay is not optimal. See Appendix 12.3.

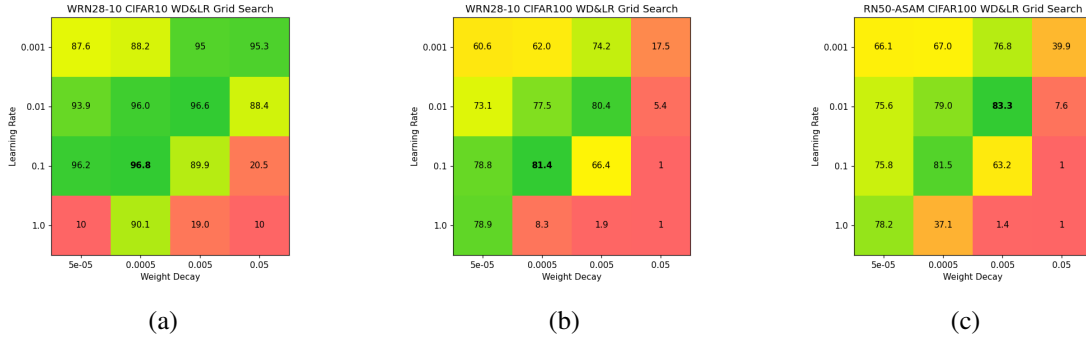


Figure 5.1: Grid Search on different values of learning rate and weight decay on accuracy of WRN28-10 trained with SGD on (a) CIFAR10 and (b) CIFAR100, and (c) ResNet50 optimized with ASAM [107] for CIFAR-100.

To better understand the relationship between adjacent cells on the same diagonal of the grid search, let us consider the two cells corresponding to sets of hyper-parameter values ($\lambda_{wd} = 0.005, lr = 0.01$) and ($\lambda_{wd} = 0.0005, lr = 0.1$). We compare one step of gradient descent using these hyper-parameter values. To derive the value of parameter w at step $t + 1$ from its value at step t we have:

$$w_{t+1} = w_t - \nabla w_t lr - w \lambda_{wd} lr, \quad (5.2)$$

where ∇w_t is the gradient computed from the cross-entropy loss and $w \lambda_{wd}$ is the gradient computed

¹See Appendix 12.3 for more details on the importance of 2D grid search.

	Setting 1	Setting 2
lr	0.01	0.1
λ_{wd}	0.005	0.0005
Weight Decay Update ($-w\lambda_{wd}lr$)	$-w_t \times 5 \times 10^{-5}$	$-w_t \times 5 \times 10^{-5}$
Cross Entropy Update ($-\nabla w_t lr$)	$-\nabla w_t \times 10^{-2}$	$-\nabla w_t \times 10^{-1}$

Table 5.1: Effect of moving along the diagonal on the 2D grid search for learning rate and λ_{wd} on the updates in one gradient descent step.

from the weight decay term from eq. 5.1. By comparing the two settings, we realize that the update coming from weight decay (i.e. $-w\lambda_{wd}lr$) remains the same, but the update coming from cross entropy (i.e. $-\nabla w_t lr$) differs by a factor of 10 as shown in Table 5.1. In other words, by moving along cells on the same diagonal in Fig. 5.1, we are changing the importance of the cross-entropy term compared to weight decay while keeping the updates from the weight decay intact. A question we ask is: *What is the optimal ratio between the update coming from cross-entropy and the update coming from the weight decay in settings that generally perform better?*

To answer, we study the value of weight **D**ecay **o**ver **G**radient of cross-entropy (DoG for short) as a metric:

$$DoG_t = \frac{\|\lambda_{wd}w_t\|}{\|\nabla w_t\|}, \quad (5.3)$$

More specifically, we are interested to track the ratio of DoG from eq. 5.3 at every iteration for settings that achieve higher validation accuracy. Hence, we focus on CIFAR-100 and on setting 2 of Table 5.1. In particular, we plot the loss of the cross-entropy loss –in logarithmic scale– for each epoch in Fig. 5.2(b), and plot the running average of the DoG values for all iterations in Fig. 5.2(a). For this experiment the loss plateaus at epoch 350 (Fig. 5.2(b)), for which the running average of DoG at that epoch is $DoG = 0.016$. This average value of $DoG = 0.016$ gives a rough idea of the ratio between the two updates over the course of training.

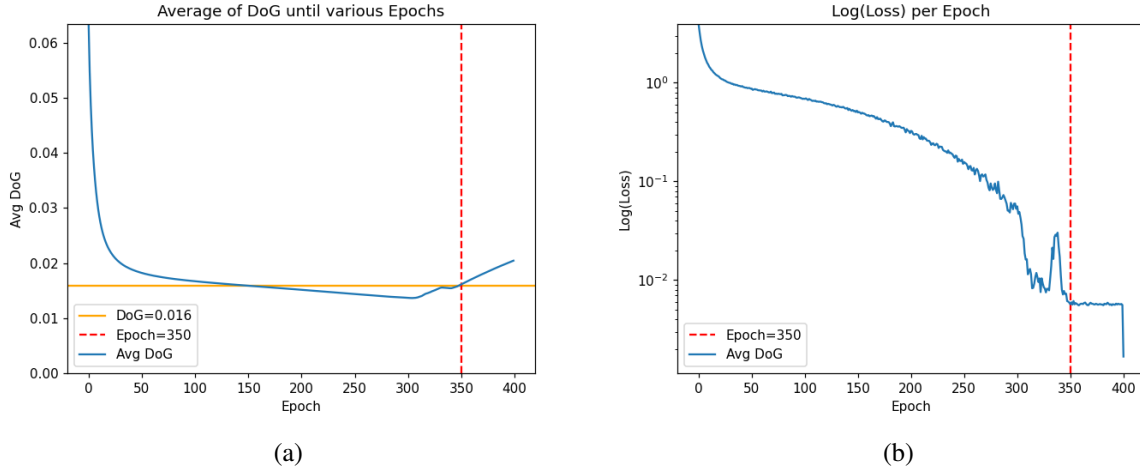


Figure 5.2: How to find the DoG value associated with a fixed λ_{wd} hyper-parameter. The average $DoG = 0.016$ is computed by averaging the DoG values from all iterations from the beginning of the training until epoch 350 (a), where 350 is the epoch at which the training loss has plateaued (b).

If the $DoG = 0.016$ is the significant underlying difference between the performance of different cells in the same diagonal, ensuring this value is constantly seen in every optimization step might have great benefits.

5.3 Adaptive Weight Decay

At this point, we introduce another simple yet effective way of maintaining this golden $DoG = 0.016$ that we call Adaptive Weight Decay. Adaptive weight decay is similar to non-adaptive (i.e., traditional) weight decay. The only difference is that the hyper-parameter λ_{wd} changes in every iteration to ensure that $DoG = 0.016$ is always satisfied. To keep this constant ratio at every step t , we can rewrite the DoG equation (eq. 5.3) as:

$$\lambda_t = \frac{0.016 \cdot \|\nabla w_t\|}{\|w_t\|}, \quad (5.4)$$

Dataset	lr	λ_{wd}	DoG	Val Acc	Test Acc	$\ W\ _2$	$\frac{0.0005 \cdot \ W\ _2^2}{2}$	X-ent	Total Loss
C100	0.1	5e-4	-	80.97	80.53	24.3	0.15	0.00	0.15
	0.1	-	0.016	80.91	80.75	18.3	0.08	0.01	0.09
C10	0.1	5e-4	-	96.83	96.31	17.2	0.07	0.00	0.08
	0.1	-	0.018	96.76	96.00	8.4	0.02	0.02	0.03

Table 5.2: WRN28-10 Models trained with adaptive weight decay by estimating the DoG value from a model trained with non-adaptive weight decay result in models with smaller weight-norms and even smaller total training loss. Results are an average of 3 runs and X-ent is the average training cross-entropy at the end of training. Dataset names are abbreviated, C100 is CIFAR-100.

Eq 5.4 allows us to have a different weight decay hyper parameter value (λ_{wd}) for every optimization iteration t , which keeps the gradients received from the cross entropy and weight decay balanced throughout the optimization. Note that this value for weight decay penalty λ_t can be computed on the fly and training with adaptive weight decay where we update the value of λ_t during each iteration has almost no computational overhead. In Table 5.2, we compare networks trained with non-adaptive weight decay using the optimal lr and λ_{wd} from Fig. 5.1 to those trained with adaptive weight decay where we estimate the DoG value similar to what was mentioned before in Fig. 5.2.

We can see that networks trained with adaptive weight decay result in comparable performance to those trained with non adaptive weight decay in terms of held out validation accuracy and also test accuracy. However, networks trained with adaptive weight decay have considerably smaller weight-norms. Most interestingly, when we measure the total loss (from eq. 5.1), we observe that networks with adaptive weight decay even have smaller training loss even though they have not used that loss directly. This implies that adaptive weight decay can find solutions with smaller objective function value than those found by non-adaptive weight decay.

To study the differences between adaptive weight decay and non-adaptive weight decay, we

can plug in λ_t of Adaptive Weight Decay (eq. 5.4) into the equation for traditional weight decay (eq. 5.1) and derive at the total loss based on Adaptive Weight Decay:

$$Loss_{w_t}(x, y) = CrossEntropy(w_t(x), y) + 0.008 \|\nabla w_t\| \|w_t\|, \quad (5.5)$$

By comparing the weight decay term in the adaptive weight decay final loss (eq. 5.5): $\frac{DoG}{2} \|w\| \|\nabla w\|$ with that of the traditional weight decay final loss (eq. 5.1): $\frac{\lambda_{wd}}{2} \|w\|^2$, we can build intuition on some of the differences between the two. For example, the non-adaptive weight decay regularization term approaches zero only when the norm of the weights is close to zero, whereas, in adaptive weight decay, it also happens when the gradients from cross-entropy are close to zero. This is a desirable property since intuitively, when the optimizer reaches a flat minima, adaptive weight decay stops over-optimizing the norm of weights, whereas, in non-adaptive weight decay, that is not the case. Also, when the gradient of cross-entropy is large, adaptive weight decay penalizes the norm of the weights more which could prevent the optimization from falling into a steep local minima which could result in overfitting.

5.4 Properties of Adaptive Weight Decay

In this section, we discuss the properties of Adaptive Weight Decay compared to its non adaptive counterpart. First, we will look into adversarial robustness and noisy data applications – two areas which could benefit the most from the potentials of adaptive weight decay. For adversarial robustness, we will look at the robust overfitting phenomenon [163] and study the effect of adaptive weight decay on reducing adversarial overfitting. We observe the relative performance improvement of 10.71% on CIFAR-10 and 20.51% on CIFAR-100 (i.e. 4% absolute

improvement in robustness for CIFAR-100 and 1% on CIFAR-10) over a popular baseline such as 7-step PGD adversarial training [129]. Then, we study the effect of adaptive weight decay in the noisy label setting. More specifically, we show roughly 4% improvement in accuracy on CIFAR-100 and 2% on CIFAR-10 for training on 20% symmetry label flipping setting [13]. Then, we shift our focus to sub-optimal learning rates. We show that adaptive weight decay finds better optima when the learning rate is not tuned. We also show that networks achieving roughly similar accuracy, once trained with adaptive weight decay, tend to have lower weight norms. This phenomenon can have exciting implications for pruning networks [73, 111].

5.4.1 Adversarial Robustness

Deep neural networks are susceptible to adversarial perturbations [15, 191]. In the adversarial setting, the adversary adds a small imperceptible noise to the image which fools the network to make an incorrect prediction. To ensure that the adversarial noise is imperceptible to the human eye, usually noise with ℓ_p -norm bounds have been studied [177]. In such settings, the objective for the adversary is to maximize the following loss:

$$\max_{\|\delta\|_p \leq \epsilon} \text{CrossEntropy}(w(x + \delta), y), \quad (5.6)$$

An array of works study methods of generating strong adversarial examples [3, 20, 31, 63, 89, 129]. While some studies focus on defenses with theoretical guarantees [29, 211], variations of adversarial training have been the de-facto defense against adversarial attacks [65, 129, 162, 174, 213].

Adversarial training consists of generating adversarial examples for the training data on the

fly and training on them. More precisely, the loss for adversarial training is as follows:

$$\min_w \left(\max_{\|\delta\|_p \leq \epsilon} \text{CrossEntropy}(w(x + \delta), y) + \frac{\lambda_{wd}}{2} \|w\|_2^2 \right), \quad (5.7)$$

While adversarial training is a strong baseline for defenses against adversarial attacks, it usually suffers from a phenomenon called Robust Overfitting [163]. To overcome this issue, various methods have been proposed to rectify robust overfitting, including early stopping [163], use of extra unlabeled data [21], synthesized images [65], pre-training [78], use of data augmentations [162], and stochastic weight averaging [89]. We re-visit tuning conventional non-adaptive weight decay parameter (λ_{wd}) for the task of improving robustness generalization, and compare that to tuning the *DoG* hyper-parameter for adaptive weight decay.

For these experiments, we use a WideResNet 28-10 architecture and use widely accepted PGD adversarial training to solve eq. 5.7 [129] while keeping 10% of the examples from the training set as held-out validation set for the purpose of early stopping. For early-stopping, we select the checkpoint which gives the highest $\ell_\infty = 8$ robustness accuracy measured by a 20 step PGD attack on the held-out validation set. Other details about the experimental setup can be found in Appendix 12.1.1.

To ensure that we search for enough values for λ_{wd} , we use twice as many values for λ_{wd} compared to *DoG*. Figure 5.3 plots the robustness accuracy measured by applying AutoAttack [32] on the test examples for the CIFAR-10 and CIFAR-100 datasets, respectively. We observe that training with adaptive weight decay improves the robustness by a margin of 4.84% on CIFAR-10 and 5.08% on CIFAR-100, compared to the non-adaptive counterpart. This is a relative improvement of 10.7% and 20.5% on CIFAR-10 and CIFAR-100, respectively.

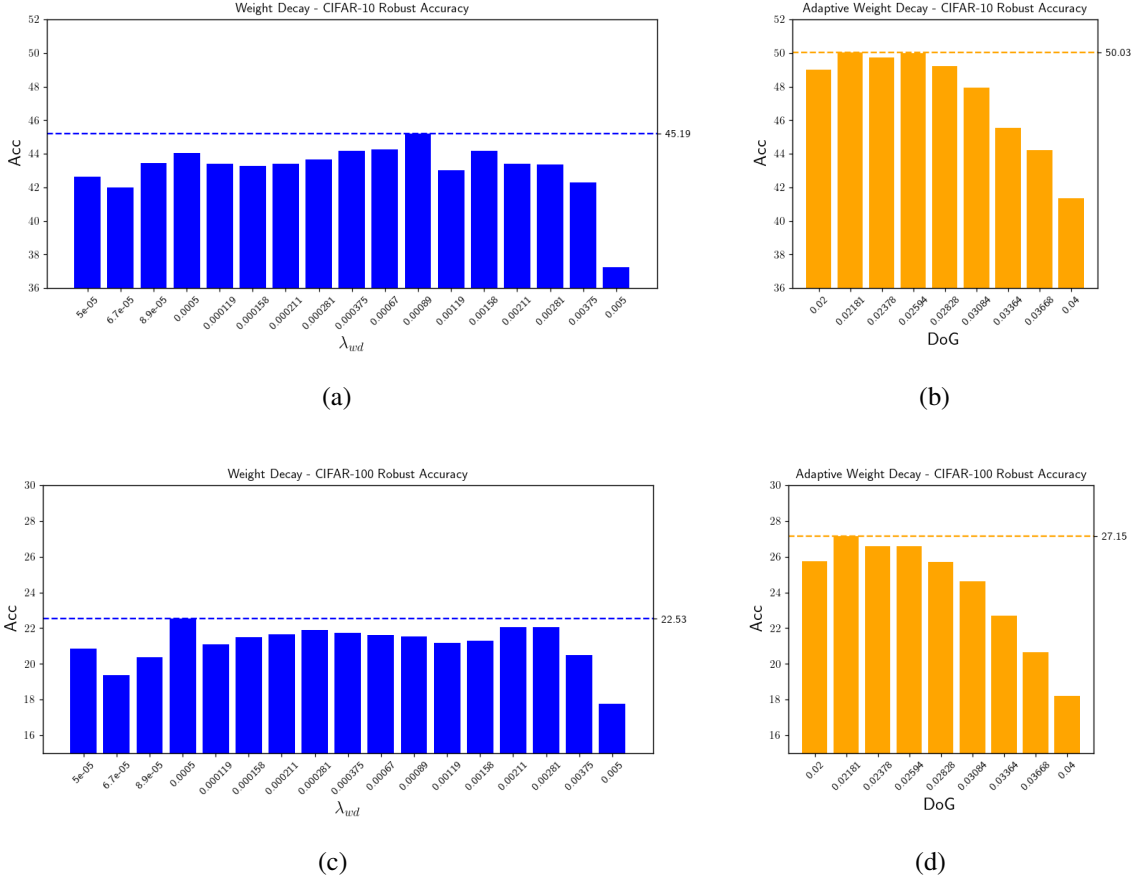


Figure 5.3: $\ell_\infty = 8$ robust accuracy on the test set of adversarially trained WideResNet28-10 networks on CIFAR-10 and CIFAR-100. (a, c) Using different choices for the hyper-parameter of non-adaptive weight decay (λ_{wd}), and (b, d) different choices of the hyper-parameter for adaptive weight decay (DoG).

Increasing robustness often comes at the cost of drops in clean accuracy [236]. This could be partially due to the fact that some ℓ_p -norm bounded adversarial examples look more like the network’s prediction than their original class[177]. An active area of research seeks a better trade-off between robustness and natural accuracy by finding other points on the pareto-optimal curve of robustness and accuracy. For example, [10] use instance-specific perturbation budgets during training. Interestingly, when comparing the most robust network trained with non-adaptive weight decay (λ_{wd}) to that trained with adaptive weight decay (DoG), we notice that those trained with adaptive weight decay have higher clean accuracy (Table. 5.3). In addition, similar

to the case before, we can observe comparatively smaller weight-norms for models which are adversarially trained with adaptive weight decay which might contribute to their better generalization.

Method	Dataset	Optimizer	$\ W\ _2$	Nat Acc	AutoAttack
$\lambda_{wd} = 0.00089$	CIFAR-10	SGD	35.58	84.31	45.19
$DoG = 0.022$		SGD	7.11	87.08	50.03
$\lambda_{wd} = 0.0005$	CIFAR-100	SGD	51.32	60.15	22.53
$DoG = 0.022$		SGD	13.41	61.39	27.15

Table 5.3: Adversarial robustness of WRN28-10 PGD-7 adversarially trained networks using adaptive and non-adaptive weight decay. Table summarizes the best performing hyper-parameter for each method on each dataset. Not only adaptive weight decay outperforms the non-adaptive weight decay trained model in terms of robust accuracy, it is also superior in terms of the natural accuracy.

The previously shown results suggest excellent potential for adversarial training with adaptive weight decay. To further study this potential, we only substituted the Momentum-SGD optimizer with the ASAM optimizer [107] and used the same hyperparameters used in previous experiments. To the best of our knowledge, and according to the RobustBench benchmark [33], the state-of-the-art $\ell_\infty = 8.0$ robust accuracy for CIFAR-100 without extra synthesized or captured data using WRN28-10 is 29.80% [162]. We achieve 29.54% robust accuracy, which is comparable to this baseline. See Table 5.4 for more details.

5.4.2 Robustness to Noisy Labels

Popular vision datasets, such as MNIST [109], CIFAR [99], and ImageNet [40], contain some amount of label noise [230, 239]. While some studies provide methods for identifying and correcting such errors [1, 106, 139, 230], others provide training algorithms to avoid over-fitting noisy training data or, even better, avoid fitting the noisy labeled examples entirely [91, 92, 183].

In this section, we perform a preliminary investigation of adaptive weight decay’s resistance

Method	WRN	Aug	Epo	ASAM	TR	SWA	Nat	AA
$\lambda_{wd} = 0.0005$	28-10	P&C	200	-	-	-	60.15	22.53
$\lambda_{wd} = 0.0005$	28-10	P&C	100	✓	-	-	58.09	22.55
$\lambda_{wd} = 0.00281^*$	28-10	P&C	100	✓	-	-	62.24	26.38
$\lambda_{AdaDecay} = 0.002^*$	28-10	P&C	200	-	-	-	57.17	24.18
$DoG = 0.022$	28-10	P&C	200	-	-	-	61.39	27.15
$DoG = 0.022$	28-10	P&C	100	✓	-	-	63.93	29.54
[162]	28-10	CutMix	400	-	✓	✓	62.97	29.80
[162]	28-10	P&C	400	-	✓	✓	59.06	28.75

Table 5.4: CIFAR-100 adversarial robustness performance of various strong methods. Adaptive weight decay with ASAM optimizer outperforms many strong baselines. For experiments marked with * we do another round of hyper-parameter search. $\lambda_{AdaDecay}$ indicates using the work from [141]. The columns represent the method, depth and width of the WideResNets used, augmentation, number of epochs, whether ASAM, TRADES [236], and Stochastic Weight Averaging [89], were used in the training, followed by the natural accuracy and adversarial accuracy using AutoAttachk. In the augmentation column, P&C is short for Pad and Crop. Note that both DoG and λ_{wd} are simply trained using 7step PGD adversarial training.

to fitting data with label noise. Following previous studies, we use symmetry label flipping [13] to create noisy data for CIFAR-10 and CIFAR-100 and use ResNet34 as the backbone. Other experimental setup details can be found in Appendix 12.1.2. Similar to the previous section, we test different hyper-parameters for adaptive and non-adaptive weight decay. To ease comparison in this setting, for each hyper-parameter, we train two networks: 1- with a certain degree of label noise, 2- with no noisy labels. We then report the accuracy on the test set where the test labels are not altered. The test accuracy on the second network – one which is trained with no label noise – is just the clean accuracy. Having the clean accuracy coupled with the accuracy after training on noisy data enables an easier understanding of the sensitivity of each training algorithm and choice of hyper-parameter to label noise. Figure 5.4 gathers the results for the noisy data experiment on CIFAR-100.

Figure 5.4 illustrates that networks trained with adaptive weight decay have a smaller drop in performance when there is label noise in the training set in comparison to non adaptive weight

decay. For instance, Figure 5.4(a) shows that $DoG = 0.028$ for adaptive and $\lambda_{wd} = 0.0089$ for non-adaptive weight decay achieve roughly 70% accuracy when trained on clean data, while the adaptive version achieves 4% higher accuracy when trained on the 20% label noise setting compared to the non-adaptive version. Similar conclusions can be drawn for CIFAR-10 for which the experiments can be found in Appendix 12.5.1.

Intuitively, from the adaptive weight decay equation in eq. 5.4, towards the end of training, where the examples with label noise are producing large gradients adaptive weight decay increases the penalty for weight decay which prevents fitting the noisy data by regularizing the gradient.

5.4.3 Additional Robustness benefits

Throughout the 2D grid search experiments in section 5.2, we observed that non-adaptive weight decay is sensitive to changes in the learning rate (LR). In this section we aim to study the sensitivity of the best hyper-parameter value for adaptive and non-adaptive weight decay to learning rate. In addition, models trained with adaptive weight decay tend to have smaller weight norms which could make them more suited for pruning. To test this intuition, we adopt a simple non-iterative ℓ_1 pruning. To build confidence on robustness to LR and pruning, for the optimal choices of $\lambda_{wd} = 0.0005$ and the estimated $DoG = 0.016$ for WRN28-10, we train 5 networks per choice of learning rate. We prune each network to various degrees of sparsity. We then plot the average of all trials per parameter set for each of the methods. Fig. 5.5 summarizes the clean accuracy without any pruning and the accuracy after 70% of the network is pruned. As it can be seen, for CIFAR-100, adaptive weight decay is both more robust to learning rate changes and also the result networks are less sensitive to parameter pruning. For more details and results on

CIFAR-10, please see Appendix 12.5.4 and 12.5.5.

5.4.4 Under-Fitting Data, A Desirable Property

Our experiments show that adaptive weight decay prevents fitting all the data in case of noisy label and adversarial training. Experimentally, we showed that adaptive weight decay contributes to this outcome more than non-adaptive weight decay. Interestingly, even in the case of natural training of even simple datasets such as CIFAR-100, networks trained with optimal adaptive weight decay still underfit the data. For instance, consider the following setup where we train a ResNet50, with ASAM [107] minimizer with both adaptive and non-adaptive weight decay². Table 5.5 shows the accuracy of the two experiments.

Method	Learning Rate	$\ W\ _2$	Test Acc(%)	Training Acc(%)
Adaptive $DoG = 0.022$	0.01	14.39	83.21	95.29
Non-Adaptive $\lambda_{wd} = 0.005$	0.01	17.86	83.23	98.57

Table 5.5: Train and Test accuracy of adaptive and non-adaptive weight decay trained models. While the adaptive version fits 3.28% less training data, it still results in comparable test accuracy. Our hypothesis is that probably it avoids fitting the noisy labeled data.

Intriguingly, without losing any performance, the model trained with adaptive weight decay avoids fitting an extra 3.28% of training data compared to the model trained with non-adaptive weight decay. We investigated more on what the 3.28% unfit data looks like³. Previously studies have discovered that some of the examples in the CIFAR-100 dataset have wrong labels [1, 106, 139, 239]. We found the 3.28% to be consistent with their findings and that many of the unfit data have noisy labels. We show some apparent noisy labeled examples in Figure 5.6.

²To find the best performing hyper-parameters for both settings, we do a 2D grid-search which can be seen in (c) of Fig. 5.1.

³See Appendix 12.5.3 for images from the 4.71%.

5.5 Conclusion

Regularization methods for a long have aided deep neural networks in generalizing to data not seen during training. Due to their significant effects on the outcome, it is crucial to have the right amount of regularization and correctly tune training hyper-parameters. We propose Adaptive Weight Decay (AWD) which is a simple modification to one of the more common regularization methods. We compare AWD with non-adaptive weight decay in multiple settings such as adversarial robustness, and training with noisy labels, and experimentally illustrate that AWD results in more robustness.

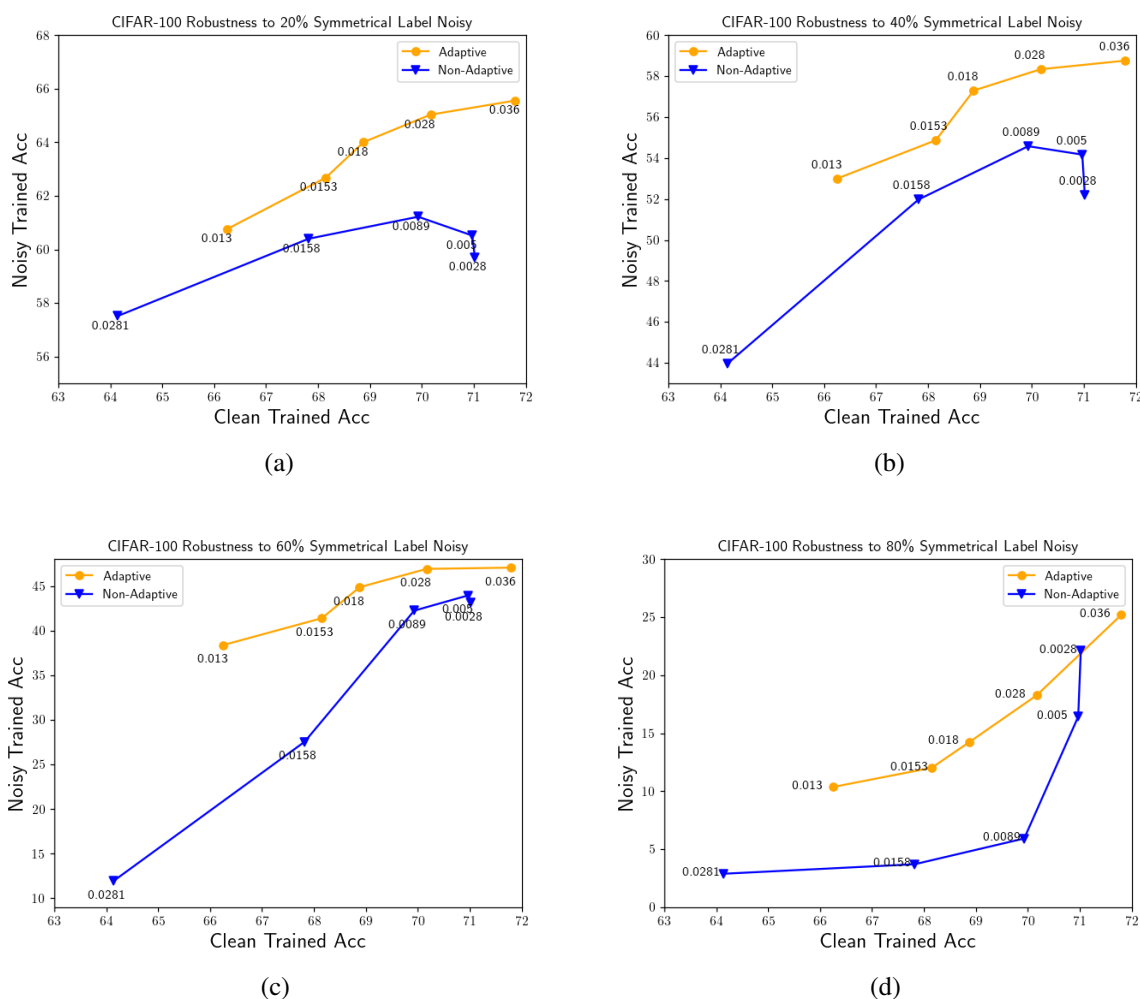


Figure 5.4: Comparison of similarly performing networks once trained on CIFAR-100 clean data, after training on 20% (a), 40% (b), 60% (c), and 80% (d) noisy data. Networks trained with adaptive weight decay are less sensitive to label noise compared to ones trained with non-adaptive weight decay.

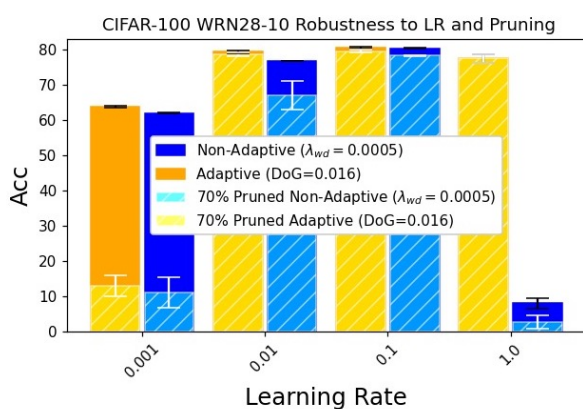


Figure 5.5: CIFAR-100 models trained with Adaptive Weight Decay (AWD) are less sensitive to learning rate. Also, due to the smaller weight norms of models trained with AWD, they seem like good candidates for pruning. Interestingly, when models are trained with smaller learning rates, they could be more sensitive to trivial pruning algorithms such as non-iterative (i.e., global) ℓ_1 pruning. The results are average of 4 runs.

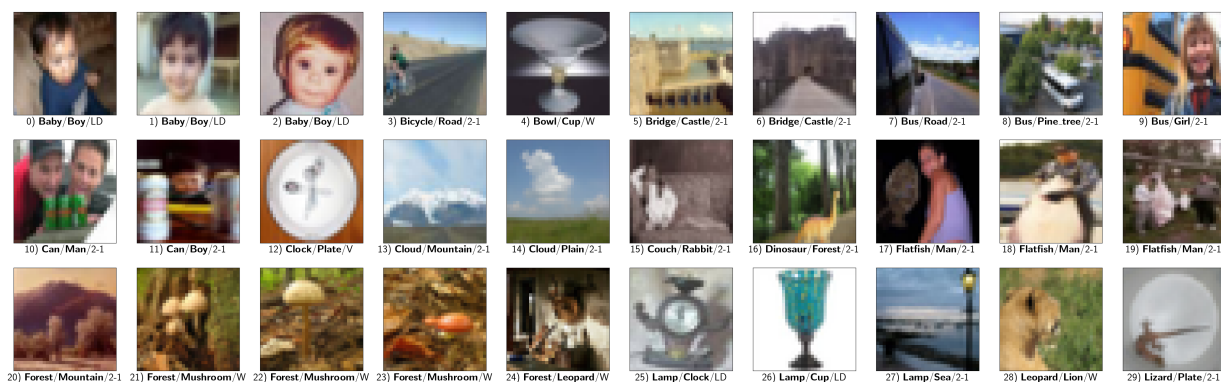


Figure 5.6: Examples from CIFAR-100 training dataset that have noisy labels. For every image we state [dataset label/ prediction of classifier trained with AWD / our category of noisy case]. We classify these noisy labels into several categories: **W**: Wrong Labels where the picture is clear enough to comprehend the correct label.; **2-1**: Two Objects from CIFAR-100 in one image, but only one label is given in the dataset; **LD**: Loosely Defined Classes where there is one object, but one object could be two classes at the same time. For instance, a baby girl is both a baby and a girl. **V**: Vague images where authors had a hard time identifying.

Chapter 6: Plug-In Inversion

6.1 Introduction

Model inversion is an important tool for visualizing and interpreting behaviors inside neural architectures, understanding what models have learned, and explaining model behaviors. In general, model inversion seeks inputs that either activate a feature in the network (feature visualization) or yield a high output response for a particular class (class inversion) [144]. Model inversion and visualization has been a cornerstone of conceptual studies that reveal how networks decompose images into semantic information [44, 232]. Over time, inversion methods have shifted from solving conceptual problems to solving practical ones. Saliency maps, for example, are image-specific model visualizations that reveal the inputs that most strongly influence a model’s decisions [180].

Recent advances in network architecture have posed major challenges for existing model inversion schemes. Convolutional Neural Networks (CNN) have long been the de-facto approach for computer vision tasks, and are the focus of nearly all research in the model inversion field. Recently, other architectures have emerged that achieve results competitive with CNNs. These include Vision Transformers [ViTs; 46], which are based on self-attention layers, and MLP-Mixer [197] and ResMLP [198], which are based on Multi Layer Perceptron layers. Unfortunately, most existing model inversion methods either cannot be applied to these architectures, or are known to fail. For example, the feature regularizer used in DeepInversion [226] cannot be applied to ViTs or MLP-based models because they do not include Batch Normalization layers [88].

In this work, we focus on class inversion, the goal of which is to find interpretable images that maximize the score a classification model assigns to a chosen label without knowledge about the model’s training data. Class inversion has been used for a variety of tasks including model



Fringilla (10)

Tibetan terrier (200)

Volcano (980)

Alp (970)

Figure 6.1: Inverted images from a robust-ResNet50 model trained on ImageNet-1k.

interpretation [137], image synthesis [170], and data-free knowledge transfer [226]. However, current inversion methods have several key drawbacks. The quality of generated images is often highly sensitive to the weights assigned to regularization terms, so these hyper-parameters need to be carefully calibrated for each individual network. In addition, methods requiring batch norm parameters are not applicable to emerging architectures.

To overcome these limitations, we present Plug-In Inversion (PII), an augmentation-based approach to class inversion. PII does not require any explicit regularization, which eliminates the need to tune regularizer-specific hyper-parameters for each model or image instance. We show that PII is able to invert CNNs, ViTs, and MLP networks using the same architecture-agnostic method, and with the same architecture-agnostic hyper-parameters.

6.1.1 Contributions

We summarize our contributions as follows:

- We provide a detailed analysis of various augmentations and how they affect the quality of images produced via class inversion.

- We introduce Plug-In Inversion (PII), a new class inversion technique based on these augmentations, and compare it to existing techniques.
- We apply PII to dozens of different pre-trained models of varying architecture, justifying our claim that it can be ‘plugged in’ to most networks without modification.
- In particular, we show that PII succeeds in inverting ViTs and large MLP-based architectures, which to our knowledge has not previously been accomplished.
- Finally, we explore the potential for combining PII with prior methods.

In section 6.2, we review existing techniques for class inversion and outline the types of architectures we consider. In section 6.3, we explore individually the different augmentations that we use, and then describe the full PII algorithm. The remainder is devoted to results and analysis.

6.2 Background

6.2.1 Class inversion

In the basic procedure for class inversion, we begin with a pre-trained model f and chosen target class y . We randomly initialize (and optionally pre-process) an image $\mathbf{x}$ in the input space of f . We then perform gradient descent to solve the following optimization problem for a chosen objective function $\mathcal{L}$,

$$\hat{x} = \arg \min_{\mathbf{x}} \mathcal{L}(f(\mathbf{x}), y),$$

and the result is class image $\hat{x}$. For very shallow networks and small datasets, letting $\mathcal{L}$ be cross-entropy or even the negative confidence assigned to the true class can produce recognizable images with minimal pre-processing [52]. Modern deep neural networks, however, cannot be inverted as easily.

6.2.2 Regularization

Most prior work on class inversion for deep networks has focused on carefully designing the objective function to produce quality images. This entails combining a divergence term (e.g. cross-entropy) with one or more regularization terms (image priors) meant to guide the optimization towards an image with ‘natural’ characteristics. DeepDream [137], following work on feature inversion [130], uses two such terms: $\mathcal{R}_{\ell_2}(\mathbf{x}) = \|\mathbf{x}\|_2^2$, which penalizes the magnitude of the image vector, and total variation, defined as¹

$$\begin{aligned} \mathcal{R}_{TV}(\mathbf{x}) = & \left(\sum_{i,j} (x_{i+1,j} - x_{i,j})^2 \right)^{\frac{1}{2}} + \left(\sum_{i,j} (x_{i,j+1} - x_{i,j})^2 \right)^{\frac{1}{2}} \\ & + \left(\sum_{i,j} (x_{i+1,j+1} - x_{i,j})^2 \right)^{\frac{1}{2}} + \left(\sum_{i,j} (x_{i+1,j} - x_{i,j+1})^2 \right)^{\frac{1}{2}}, \end{aligned}$$

which penalizes sharp changes over small distances. DeepInversion [226] uses both of these regularizers, along with the feature regularizer

$$\mathcal{R}_{feat}(\mathbf{x}) = \sum_k (\|\mu_k(\mathbf{x}) - \hat{\mu}_k\|_2 + \|\sigma_k^2(\mathbf{x}) - \hat{\sigma}_k^2\|_2),$$

where μ_k, σ_k^2 are the batch mean and variance of the features output by the k -th convolutional

¹This is the formulation used by [226]; others are also common, such as the simpler version in [130].

layer, and $\hat{\mu}_k, \hat{\sigma}_k^2$ are corresponding Batch Normalization statistics stored in the model [88]. Naturally, this method is only applicable to models that use Batch Normalization, which leaves out ViTs, MLPs, and even some CNNs. Furthermore, the optimal weights for each regularizer in the objective function vary wildly depending on architecture and training set, which presents a barrier to easily applying such methods to a wide array of networks.

6.2.3 Architectures for vision

We now present a brief overview of the three basic types of vision architectures that we will consider.

Convolutional Neural Networks (CNNs) have long been the standard in deep learning for computer vision [101, 110]. Convolutional layers encourage a model to learn properties desirable for vision tasks, such as translation invariance. Numerous CNN models exist, mainly differing in the number, size, and arrangement of convolutional blocks and whether they include residual connections, Batch Normalization, or other modifications [74, 178, 231].

Dosovitskiy et al. [46] recently introduced **Vision Transformers** (ViTs), adapting the transformer architectures commonly used in Natural Language Processing [204]. These models break input images into patches, combine them with positional embeddings, and use these as input tokens to self-attention modules. Others have proposed variants which require less training data [200], have convolutional inductive biases [37], or make other modifications to the attention modules [28, 125, 223].

Subsequently, a number of authors have proposed vision models which are based solely on **Multi-Layer Perceptrons** (MLPs), using insights from ViTs [123, 197, 198]. Generally, these

models use patch embeddings similar to ViTs and alternate channel-wise and patch-wise linear embeddings, along with non-linearities and normalization.

We emphasize that as the latter two architecture types are recent developments, our work is the first to study them in the context of model inversion.

6.3 Plug-In Inversion

Prior work on class inversion uses augmentations like jitter, which randomly shifts an image horizontally and vertically, and horizontal flips to improve the quality of inverted images [137, 226]. The hypothesis behind their use is that different views of the same image should result in similar scores for the target class. These augmentations are applied to the input before feeding it to the network, and different augmentations are used for each gradient step used to reconstruct x . In this section, we explore additional augmentations that benefit inversion before describing how we combine them to form the PII algorithm.

As robust models are typically easier to invert than naturally trained models [131, 170], we use a robust ResNet-50 [74] model trained on the ImageNet [40] dataset throughout this section as a toy example to examine how different augmentations impact inversion. Note, we perform the demonstrations in this section under slightly different conditions and with different models than those ultimately used for PII in order to highlight the effects of the augmentations as clearly as possible. The reader may find thorough experimental details in the appendix, section 13.4.

6.3.1 Restricting Search Space

In this section, we consider two augmentations to improve the spatial qualities of inverted images: Centering and Zoom. These are designed based on our hypothesis that restricting the input optimization space encourages better placement of recognizable features. Both methods start with small input patches, and each gradually increases this space in different ways to reach the intended input size. In doing so, they force the inversion algorithm to place important semantic content in the center of the image.

Centering Let x be the input image being optimized. At first, we only optimize a patch at the center of x . After a fixed number of iterations, we increase the patch size outward by padding with random noise, repeating this until the patch reaches the full input size. Figure 6.2 shows the state of the image prior at each stage of this process, as well as an image produced without centering. Without centering, the shift invariance of the networks allows most semantic content to scatter to the image edges. With centering, results remain coherent.

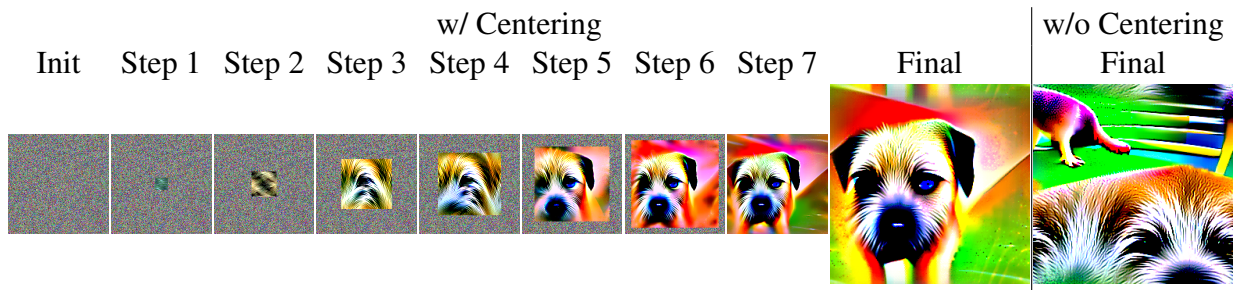


Figure 6.2: An image at different stages of optimization with centering (left), and an image inverted without centering (right), for the Border Terrier class of a robust ResNet-50.

Zoom For zoom, we begin with an image x of lower resolution than the desired result. In each step, we optimize this image for a fixed number of iterations and then up-sample the result,

repeating until we reach the full resolution. Figure 6.3 shows the state of an image at each step of the zoom procedure, along with an image produced without zoom. The latter image splits the object of interest at its edges. By contrast, zoom appears to find a meaningful structure for the image in the early steps and refines details like texture as the resolution increases.

We note that zoom is not an entirely novel idea in inversion. Yin et al. [226] use a similar technique as ‘warm-up’ for better performance and speed-up. However, we observe that continuing zoom throughout optimization contributes to the overall success of PII.

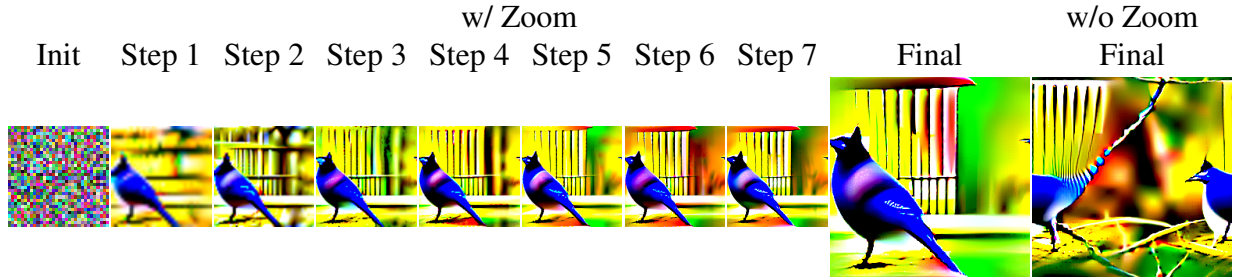


Figure 6.3: An image during different stages of optimization with zoom (left), and an image inverted without zoom (right), for the Jay class of a robust ResNet-50.

Zoom + Centering Unsurprisingly, we have found that applying zoom and centering simultaneously yields even better results than applying either individually, since each one provides a different benefit. Centering places detailed and important features (e.g. the dog’s eye in Figure 6.2) near the center and builds the rest of the image around the existing patch. Zoom helps enforce a sound large-scale structure for the image and fills in details later.

The combined Zoom and Centering process proceeds in ‘stages’, each at a higher resolution than the last. Each stage begins with an image patch generated by the previous stage, which approximately minimizes the inversion loss. The patch is then up-sampled to a resolution halfway between the previous stage and current stage resolution, allowing it to fill the center of the image,

leaving a border which is padded with random noise. Then next round of optimization then begins starting from this newly processed image.

6.3.2 ColorShift Augmentation

The colors of the illustrative images we have shown so far are notably different from what one might expect in a natural image. This is due to ColorShift, a new augmentation that we now present.

ColorShift is an adjustment of an image’s colors by a random mean and variance in each channel. This can be formulated as follows:

$$\text{ColorShift}(\mathbf{x}) = \sigma \mathbf{x} - \mu,$$

where μ and σ are C -dimensional² vectors drawn from $\mathcal{U}(-\alpha, \alpha)$ and $e^{\mathcal{U}(-\beta, \beta)}$, respectively, and are repeatedly redrawn after a fixed number of iterations. We use $\alpha = \beta = 1.0$ in all demonstrations unless otherwise noted. At first glance, this deliberate shift away from the distribution of natural images seems counterproductive to the goal of producing a recognizable image. However, our results show that using ColorShift noticeably increases the amount of visual information captured by inverted images and also obviates the need for hard-to-tune regularizers to stabilize optimization.

We visualize the stabilizing effect of ColorShift in Figure 6.4. In this experiment, we invert the model by minimizing the sum of a cross entropy and a total-variation (TV) penalty. Without ColorShift, the quality of images is highly dependent on the weight λ_{TV} of the TV regularizer;

² C being the number of channels

smaller values produce noisy images, while larger values produce blurry ones. Inversion with ColorShift, on the other hand, is insensitive to this value and in fact succeeds when omitting the regularizer altogether.

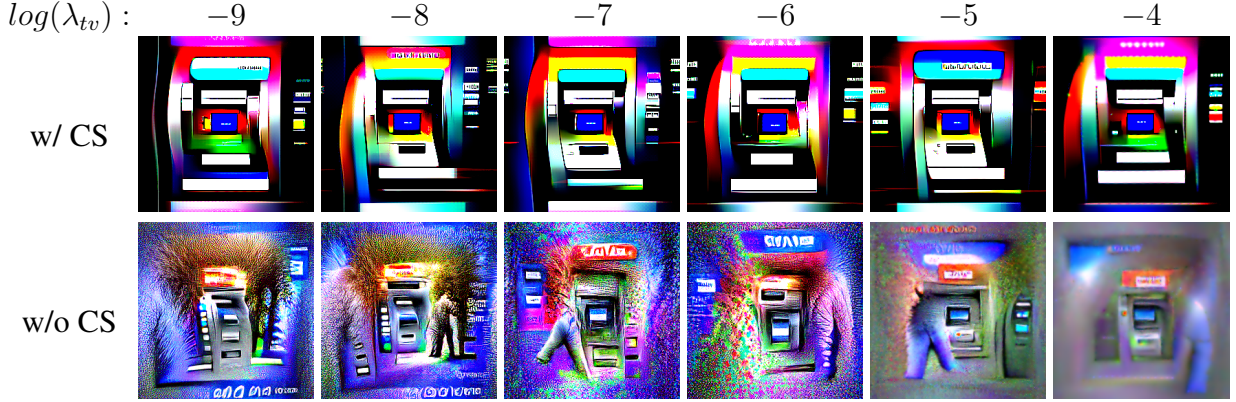


Figure 6.4: Inversions of the robust ResNet-50 ATM class, with and without ColorShift and with varying TV regularization strength. The inversion process with ColorShift is robust to changes in the λ_{tv} hyper-parameter, while without it, λ_{tv} seems to present a trade-off between noise and blur.

Other preliminary experiments show that ColorShift similarly removes the need for ℓ_2 or feature regularization, as our main results for PII will show. We conjecture that by forcing unnatural colors into an image, ColorShift requires the optimization to find a solution which contains meaningful semantic information, rather than photo-realistic colors, in order to achieve a high class score. Alternatively, as seen in Figure 6.9, images optimized with an image prior may achieve high scores despite a lack of semantic information merely by finding sufficiently natural colors and textures.

6.3.3 Ensembling

Ensembling is an established tool often used in dataset security [186] to enhanced inference [145]. Similarly, we find that optimizing an ensemble composed of different ColorShifts of the same

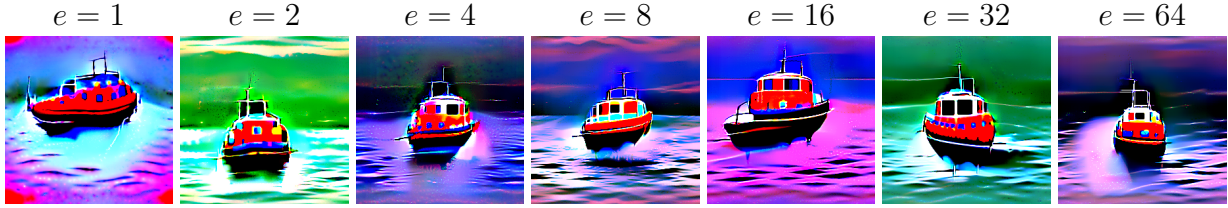


Figure 6.5: Effect of ensemble size in the quality of inverted images for the Tugboat class of a robust ResNet-50.

image simultaneously improves the performance of inversion methods. To this end, we minimize the average of cross-entropy losses $\mathcal{L}(f(\mathbf{x}_i), y)$, where the $\mathbf{x}_i$ are different ColorShifts of the image at the current step of optimization. Figure 6.5 shows the result of applying ensembling alongside ColorShift. We observe that larger ensembles appear to give slight improvements, but even ensembles of size 1 or two produce satisfactory results. This is important for models like ViTs, where available GPU memory constrains the possible size of this ensemble; in general, we use the largest ensemble size (up to a maximum of $e = 32$) that our hardware permits for a particular model. More results on the effect of ensemble size can be found in Figure 13.5. We show the effect of ensembling using other well-known augmentations and compare them to ColorShift in Appendix Section 13.1.5. We empirically show that ColorShift is the strongest among augmentations we tried for model inversion.

6.3.4 The Plug-in Inversion Method

We combine the jitter, ensembling, ColorShift, centering, and zoom techniques, and name the result Plug-In Inversion, which references the ability to ‘plug in’ any differentiable model, including ViTs and MLPs, using a single fixed set of hyper-parameters. In the next section, we detail the experimental method that we used to find these hyper-parameters, after which we present our main results.

6.4 Experimental Setup

In order to tune hyper-parameters of PII for use on naturally-trained models, we use the `torchvision` [150] ImageNet-trained ResNet-50 model. We apply centering + zoom simultaneously in 7 ‘stages.’ During each stage, we optimize the selected patch for 400 iterations, applying random jitter and ColorShift at each step. We use the Adam [96] optimizer with momentum $\beta_m = (0.5, 0.99)$, initial learning rate $lr = 0.01$, and cosine-decay. At the beginning of every stage, the learning rate and optimizer are re-initialized. We use $\alpha = \beta = 1.0$ for the ColorShift parameters, and an ensemble size of $e = 32$. We ablate these choices in Figure 6.6 to validate our selected settings. Further ablation studies can be found in the appendix in figures 13.1, 13.4, and 13.5.

6.5 Results

6.5.1 PII works on a range of architectures

We now present the results of applying Plug-In Inversion to different types of models. We once again emphasize that we use identical settings for the PII parameters in all cases.

Figure 6.7 depicts images produced by inverting the Volcano class for a variety of architectures, including examples of CNNs, ViTs, and MLPs. While the quality of images varies somewhat between networks, all of them include distinguishable and well-placed visual information. Many more examples are found in Figure 13.8 of the Appendix.

In Figure 6.8, we show images produced by PII from representatives of each main type of architecture for a few arbitrary classes. We note the distinct visual styles that appear in each

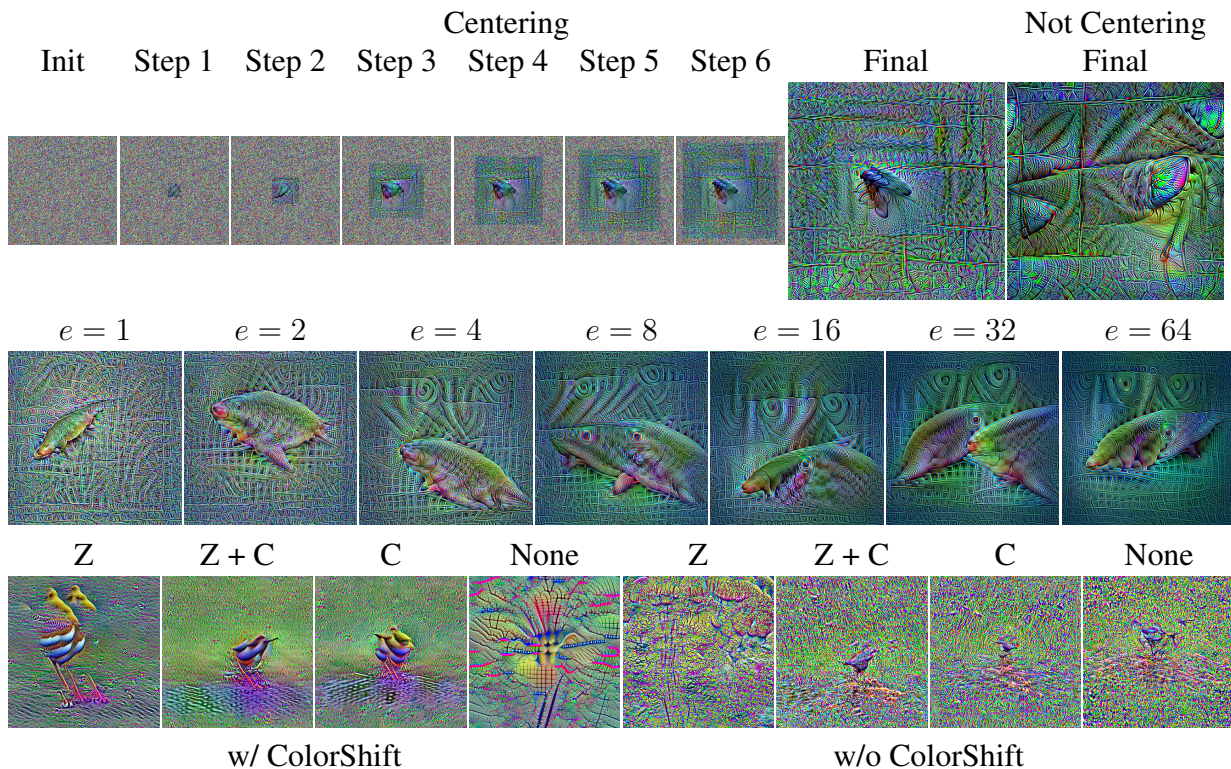


Figure 6.6: Ablation studies repeating the previous experiments for a naturally-trained ResNet-50. (top) Inversion for target class Fly with and without Centering. (center) Effect of ensembling size in the quality of inverted images for the Tench class. (bottom) The effect of various combinations of zoom, Centering, and ColorShift when inverting the Dipper class.

row, which supports the perspective of model inversion as a tool for understanding what kind of information different networks learn during training.

6.5.2 Combining PII with DeepInversion

We now consider the relative benefits of Plug-In Inversion and DeepInversion [226], a state-of-the-art class inversion method for CNNs. Figure 6.9 shows images from a few arbitrary classes produced by PII and DeepInversion (along with reference examples from the robust model demonstration in section 6.3). We additionally show images produced by DeepInversion using the output of PII, rather than random noise, as its initialization.

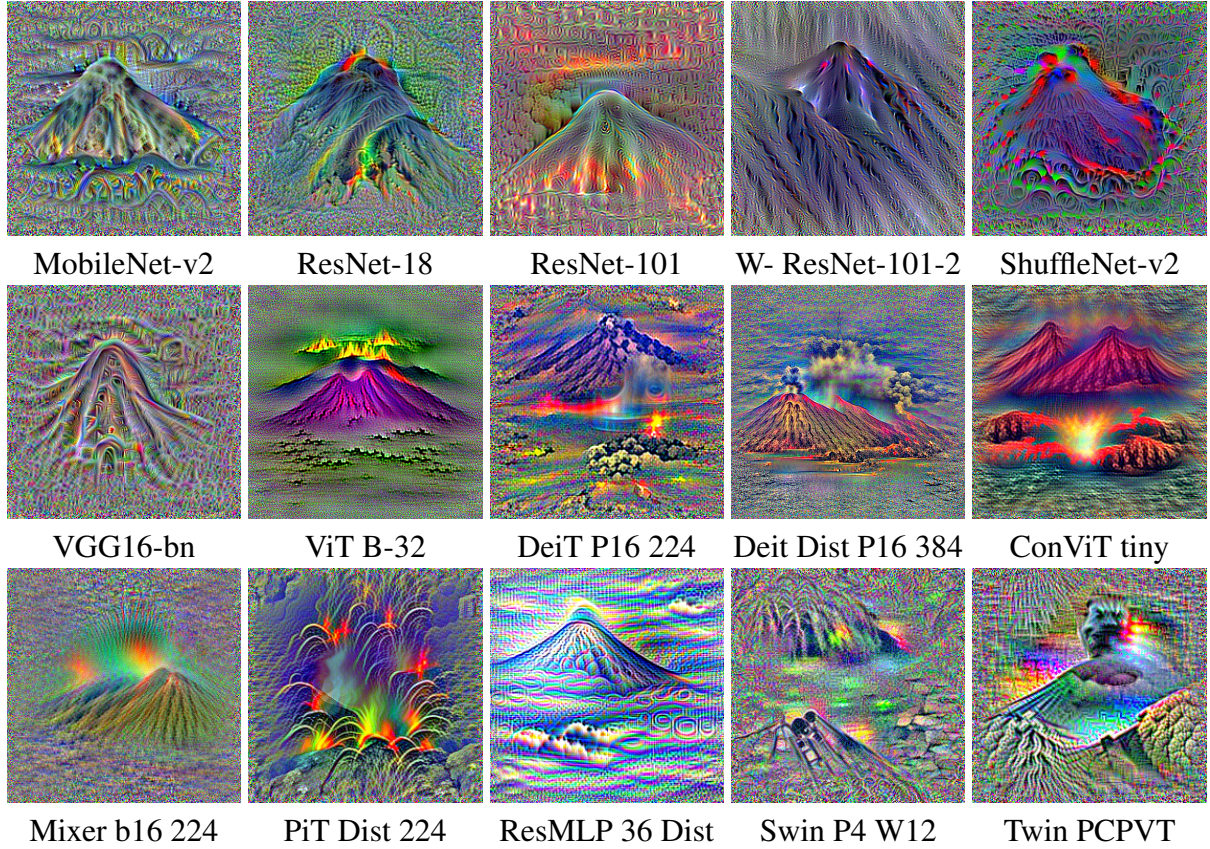


Figure 6.7: Images inverted from the Volcano class for various Convolutional, Transformer, and MLP-based networks using PII. For more details about networks, refer to Appendix 14.3.

Using either initialization, DeepInversion clearly produces images with natural-looking colors and textures, which PII of course does not. However, DeepInversion alone results in some images that either do not clearly correspond to the target class or are semantically confusing. By comparison, PII again produces images with strong spatial and semantic qualities. Interestingly, these qualities appear to be largely retained when applying DeepInversion after PII, but with the color and texture improvements that image priors afford [130].

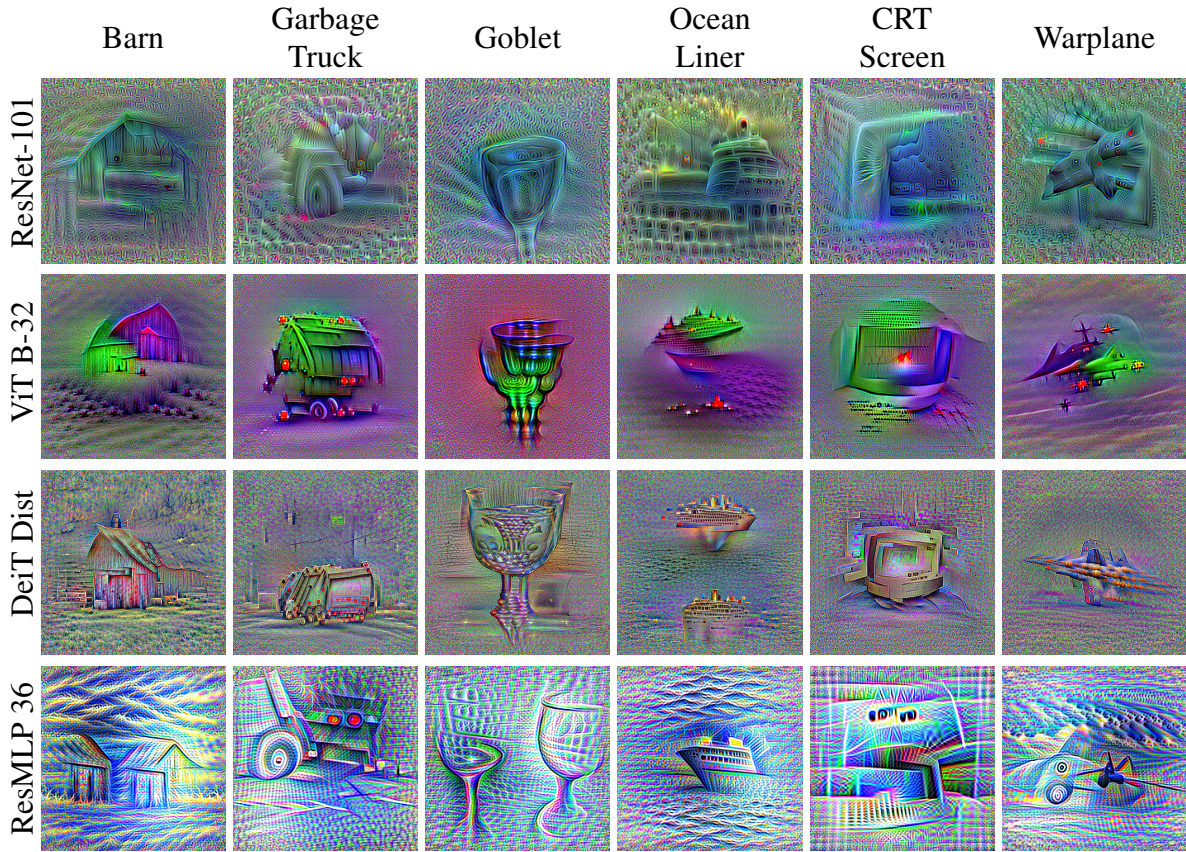


Figure 6.8: Inverting different model and class combinations for different classes using PII.

6.6 Conclusion

We studied the effect of various augmentations on the quality of class-inverted images and introduced Plug-In Inversion, which uses these augmentations in tandem. We showed that this technique produces intelligible images from a wide range of well-studied architectures, as well as the recently introduced ViTs and MLPs, without a need for model specific hyper-parameter tuning. We believe that augmentation-based model inversion is a promising direction for future research in understanding computer vision models.

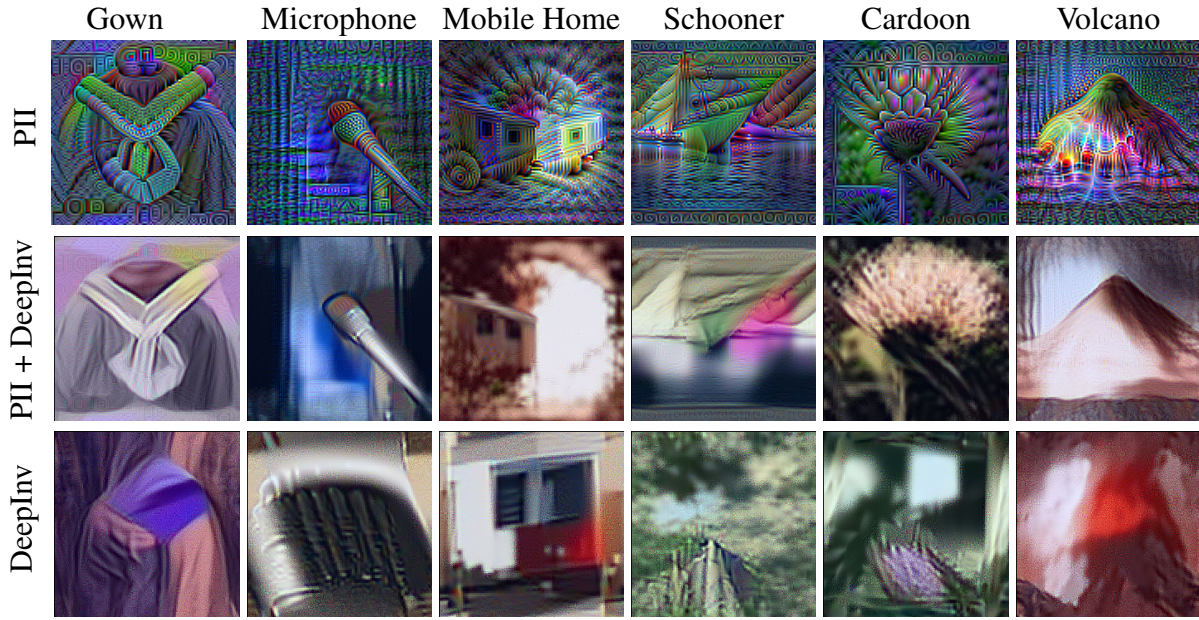


Figure 6.9: PII Inversion results for a naturally-trained ResNet-50. PII can be used as initialization for other methods such as DeepInversion.

6.7 Ethical Considerations

We propose Plug-In Inversion as a class inversion technique for the purpose of understanding vision models. However, we note that prior work has considered the potential of model inversion to compromise the security of a model’s training data [52, 226]. These areas of progress and other data privacy concerns [55, 242] make clear the need for caution when sensitive data is used to train deep learning models.

6.8 Reproducibility

All the models (including pre-trained weights) we consider in this work are publicly available from widely-used sources. Explicit details of model resources can be found in section 14.3 of the appendix. We also make the code used for all demonstrations and experiments in this work

available at <https://github.com/youranonymousefriend/plugininversion>.

Chapter 7: Visualizing Vision Transformers

Vision transformers (ViTs) are quickly becoming the de-facto architecture for computer vision, yet we understand very little about why they work and what they learn. While existing studies visually analyze the mechanisms of convolutional neural networks, an analogous exploration of ViTs remains challenging. In this paper, we first address the obstacles to performing visualizations on ViTs. Assisted by these solutions, we observe that neurons in ViTs trained with language model supervision (e.g., CLIP) are activated by semantic concepts rather than visual features. We also explore the underlying differences between ViTs and CNNs, and we find that transformers detect image background features, just like their convolutional counterparts, but their predictions depend far less on high-frequency information. On the other hand, both architecture types behave similarly in the way features progress from abstract patterns in early layers to concrete objects in late layers. In addition, we show that ViTs maintain spatial information in all layers except the final layer. In contrast to previous works, we show that the last layer most likely discards the spatial information and behaves as a learned global pooling operation. Finally, we conduct large-scale visualizations on a wide range of ViT variants, including DeiT, CoaT, ConViT, PiT, Swin, and Twin, to validate the effectiveness of our method.

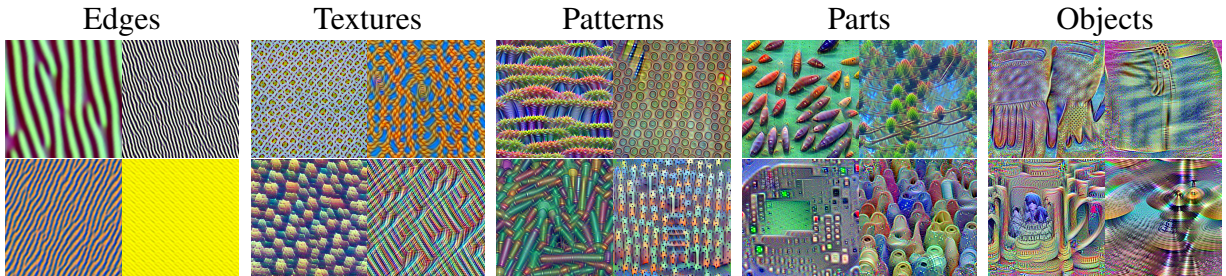


Figure 7.1: **The progression for visualized features of ViT B-32.** Features from early layers capture general edges and textures. Moving into deeper layers, features evolve to capture more specialized image components and finally concrete objects.

7.1 Introduction

Recent years have seen the rapid proliferation of vision transformers (ViTs) across a diverse range of tasks from image classification to semantic segmentation to object detection [36, 43, 45, 76, 125, 233]. Despite their enthusiastic adoption and the constant introduction of architectural innovations, little is known about the inductive biases or features they tend to learn. While feature visualizations and image reconstructions have provided a looking glass into the workings of CNNs [44, 144, 232], these methods have shown less success for understanding ViT representations, which are difficult to visualize. In this work we show that, if properly applied to the correct representations, feature visualizations can indeed succeed on ViTs. This insight allows us to visually explore ViTs and the information they glean from images.

In order to investigate the behaviors of vision transformers, we first establish a visualization framework that incorporates improved techniques for synthesizing images that maximally activate neurons. Through dissecting and visualizing the internal representations in the transformer architecture, we find that patch tokens preserve spatial information, even in individual channels, throughout all layers except the last attention block. The last layer of ViTs learns a token-mixing operation akin to average pooling, such that the classification head exhibits comparable accuracy when ingesting a random token instead of the CLS token.

After probing the role of spatial information, we delve into the behavioral differences between ViTs and CNNs. When performing activation maximizing visualizations, we notice that ViTs consistently generate higher quality image backgrounds than CNNs. Thus, we try masking out image foregrounds during inference, and find that ViTs consistently outperform CNNs when exposed only to image backgrounds. These findings bolster the observation that transformer

models extract information from many sources in an image to exhibit superior performance on out-of-distribution generalization [151] as well as adversarial robustness [176]. Additionally, convolutional neural networks are known to rely heavily on high-frequency texture information in images [56]. In contrast, we find that ViTs perform well even when high-frequency content is removed from their inputs.

We further visualize the effects of language model supervision, i.e. CLIP [155], on the features extracted by vision transformers. While both ImageNet-trained ViTs and CLIP-trained vision transformers possess neurons that are activated by visual features (e.g. shapes and colors) and distinct classes, the neurons of CLIP-trained vision transformers are also activated by features that do not represent physical objects, such as visual characteristics relating to parts of speech (e.g. epithets, adjectives, and prepositions) or broader concepts such as morbidity.

Our contributions are summarized as follows:

I. We observe that uninterpretable and adversarial behavior occurs when applying standard methods of feature visualization to the relatively low-dimensional components of transformer-based models, such as keys, queries, or values. However, applying these tools to the relatively high-dimensional features of the position-wise feedforward layer results in successful and informative visualizations. We conduct large-scale visualizations on a wide range of transformer-based vision models, including ViTs, DeiT, CoaT, ConViT, PiT, Swin, and Twin, to validate the effectiveness of our method.

II. We show that patch-wise image activation patterns for ViT features essentially behave like saliency maps, highlighting the regions of the image a given feature attends to. This behavior persists even for relatively deep layers, showing the model preserves the positional relationship between patches instead of using them as global information stores.

III. We compare the behavior of ViTs and CNNs, finding that ViTs make better use of background information and rely less on high-frequency, textural attributes. Both types of networks build progressively more complex representations in deeper layers and eventually contain features responsible for detecting distinct objects.

IV. We investigate the effect of natural language supervision with CLIP on the types of features extracted by ViTs. We find CLIP-trained models include various features clearly catered to detecting components of images corresponding to caption text, such as prepositions, adjectives, and conceptual categories.

7.2 Related Work

7.2.1 Optimization-Based Visualization

One approach to understanding what models learn during training is using gradient descent to produce an image which conveys information about the inner workings of the model. This has proven to be a fruitful line of work in the case of understanding CNNs specifically. The basic strategy underlying this approach is to optimize over input space to find an image which maximizes a particular attribute of the model. For example, Erhan et al. [50] use this approach to visualize images which maximally activate specific neurons in early layers of a network, and Olah et al. [144] extend this to neurons, channels, and layers throughout a network. Simonyan et al. [179], Yin et al. [226] produce images which maximize the score a model assigns to a particular class. Mahendran and Vedaldi [130] apply a similar method to invert the feature representations of particular image examples.

Recent work [59] has studied techniques for extending optimization-based class visualization

to ViTs. We incorporate and adapt some of these proposed techniques into our scheme for feature visualization.

7.2.2 Other Visualization Approaches

Aside from optimization-based methods, many other ways to visualize CNNs have been proposed. Dosovitskiy and Brox [44] train an auxiliary model to invert the feature representations of a CNN. Zeiler and Fergus [232] use ‘deconvnets’ to visualize patches which strongly activate features in various layers. Simonyan et al. [179] introduce saliency maps, which use gradient information to identify what parts of an image are important to the model’s classification output. Zimmermann et al. [243] demonstrate that natural image samples which maximally activate a feature in a CNN may be more informative than generated images which optimize that feature. We draw on some aspects of these approaches and find that they are useful for visualizing ViTs as well.

7.2.3 Understanding ViTs

Given their rapid proliferation, there is naturally great interest in how ViTs work and how they may differ from CNNs. Although direct visualization of their features has not previously been explored, there has been recent progress in analyzing the behavior of ViTs. Naseer et al. [142], Paul and Chen [151], Shao et al. [176] demonstrate that ViTs are inherently robust to many kinds of adversarial perturbations and corruptions. Raghu et al. [156] compare how the internal representation structure and use of spatial information differs between ViTs and CNNs. Chefer et al. [22] produce ‘image relevance maps’ (which resemble saliency maps) to promote

interpretability of ViTs.

7.3 ViT Feature Visualization

Like many visualization techniques, we take gradient steps to maximize feature activations starting from random noise [144]. To improve the quality of our images, we penalize total variation [130], and also employ the Jitter augmentation [226], the ColorShift augmentation, and augmentation ensembling [59]. Finally, we find that Gaussian smoothing facilitates better visualization in our experiments as is common in feature visualization [29, 181].

Each of the above techniques can be formalized as follows. A ViT represents each patch p (of an input x) at layer l by an array $A_{l,p}$ with d entries. We define a feature vector f to be a stack composed of one entry from each of these arrays. Let $f_{l,i}$ be formed by concatenating the i th entry in $A_{l,p}$ for all patches p . This vector f will have dimension equal to the number of patches. The optimization objective starts by maximizing the sum of the entries of f over inputs x . The main loss is then

$$\mathcal{L}_{\text{main}}(x, l, i) = \sum_p (f_{l,i})_p. \quad (7.1)$$

We employ total variation regularization by adding the term $\lambda TV(x)$ to the objective. TV represents the total variation, and λ is the hyperparameter controlling the strength of its regularization effect. We can ensemble augmentations of the input to further improve results. Let $\mathcal{A}$ define a distribution of augmentations to be applied to the input image x , and let a be a sample from $\mathcal{A}$. To create a minibatch of inputs from a single image, we sample several augmentations $\{a_k\}$ from

A. Finally, the optimization problem is:

$$x^* = \arg \max_x \sum_k \mathcal{L}_{\text{main}}(a_k(x), l, i) + \lambda TV(a_k(x)). \quad (7.2)$$

We achieve the best visualizations when $\mathcal{A}$ is $GS(CS(Jitter(x)))$, where GS denotes Gaussian smoothing and CS denotes ColorShift, whose formulas are:

$$GS(x) = x + \epsilon; \quad \epsilon \sim \mathcal{N}(0, 1)$$

$$CS(x) = \sigma x + \mu; \quad \mu \sim \mathcal{U}(-1, 1); \quad \sigma \sim e^{\mathcal{U}(-1, 1)}.$$

Note that even though ϵ and μ are both additive noise, they act on the input differently since μ is applied per channel (i.e. has dimension three), and ϵ is applied per pixel. For more details on hyperparameters, refer to Appendix 14.2.

To better understand the content of a visualized feature, we pair every visualization with images from the ImageNet validation/train set that most strongly activate the relevant feature. Moreover, we plot the feature’s activation pattern by passing the most activating images through the network and showing the resulting pattern of feature activations. Figure 7.2(a) is an example of such a visualization. From the leftmost panel, we hypothesize that this feature corresponds to gravel. The most activating image from the validation set (middle) contains a lizard on a pebbly gravel road. Interestingly, the gravel background lights up in the activation pattern (right), while the lizard does not. The activation pattern in this example behaves like a saliency map [179], and we explore this phenomenon across different layers of the network further in Section 7.4.

The model we adopt for the majority of our demonstrations throughout the paper is ViT-

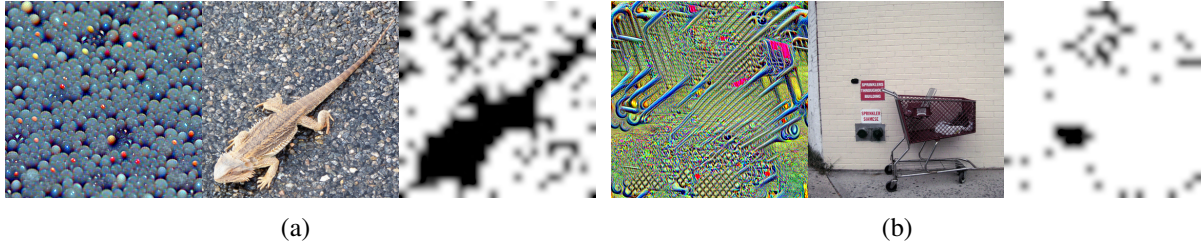


Figure 7.2: (a): **Example feature visualization from ViT feed forward layer.** *Left:* Image optimized to maximally activate a feature from layer 5. *Center:* Corresponding maximally activating ImageNet example. *Right:* The image’s patch-wise activation map. (b): **A feature from the last layer most activated by shopping carts.**

B16, implemented based on the work of Dosovitskiy et al. [45]. In addition, in the Appendix, we conduct large-scale visualizations on a wide range of ViT variants, including DeiT Touvron et al. [199], CoaT Xu et al. [223], ConViT d’Ascoli et al. [37], PiT Heo et al. [80], Swin Liu et al. [125], and Twin Chu et al. [28], 38 models in total, to validate the effectiveness of our method. ViT-B16 is composed of 12 blocks, each consisting of multi-headed attention layers, followed by a projection layer for mixing attention heads, and finally followed by a position-wise-feed-forward layer. For brevity, we henceforth refer to the position-wise-feed-forward layer simply as the feed-forward layer. In this model, every patch is always represented by a vector of size 768 except in the feed-forward layer which has a size of 3072 (4 times larger than other layers).

We first attempt to visualize features of the multi-headed attention layer, including visualization of the keys, queries, and values, by performing activation maximization. We find that the visualized feed-forward features are significantly more interpretable than other layers. We attribute this difficulty of visualizing other layers to the property that ViTs pack a tremendous amount of information into only 768 features, (e.g. in keys, queries, and values) which then behave similar to multi-modal neurons, as discussed by Goh et al. [62], due to many semantic concepts being encoded in a low dimensional space. Furthermore, we find that this behaviour is more extreme in

deeper layers. See Figure 7.3 for examples of visualizations of keys, queries and values in both early and deep layers of the ViT. Inspired by these observations, we visualize the features within the feed-forward layer across all 12 blocks of the ViT. We refer to these blocks interchangeably as layers.

The feed-forward layer depicted in Figure 7.3 takes an input of size $d = 768$, projects it into a $t = 4$ times higher dimensional space, applies the non-linearity GELU, and then projects back to d dimensional space. Unless otherwise stated, we always visualize the output of the GELU layers in our experiments. We hypothesize that the network exploits these high-dimensional spaces to store relatively disentangled representations. On the other hand, compressing the features into a lower dimensional space may result in the jumbling of features, yielding uninterpretable visualizations.

7.4 Last-Layer Token Mixing

In this section, we investigate the preservation of patch-wise spatial information observed in the visualizations of patch-wise feature activation levels which, as noted before, bear some similarity to saliency maps. Figure 7.2(a) demonstrates this phenomenon in layer 5, where the visualized feature is strongly activated for almost all rocky patches but not for patches that include the lizard. Additional examples can be seen in Figure 7.4 and the Appendix, where the activation maps approximately segment the image with respect to some relevant aspect of the image. We find it surprising that even though every patch *can* influence the representation of every other patch, these representations remain local, even for individual channels in deep layers in the network. While a similar finding for CNNs, whose neurons may have a limited receptive

field, would be unsurprising, even neurons in the first layer of a ViT have a complete receptive field. In other words, ViTs learn to preserve spatial information, despite lacking the inductive bias of CNNs. Spatial information in patches of deep layers has been explored in Raghu et al. [156] through the CKA similarity measure, and we further show that spatial information is in fact present in individual channels.

The last layer of the network, however, departs from this behavior and instead appears to serve a role similar to average pooling. Figure 7.2(b) shows one example of our visualizations for a feature from the last layer that is activated by shopping carts. The activation pattern is fairly uniform across the image. For classification purposes, ViTs use a fully connected layer applied only on the class token (the CLS token). It is possible that the network globalizes information in the last layer to ensure that the CLS token has access to the entire image, but because the CLS token is treated the same as every other patch by the transformer, this seems to be achieved by globalizing across *all* tokens.

Table 7.1: After the last layer, every patch contains the same information. “Isolating CLS” denotes the experiment where attention is only performed between patches before the final attention block, while “Patch Average” and “Patch Maximum” refer to the experiment in which the classification head is placed on top of individual patches without fine-tuning. Experiments conducted on ViT-B16.

Accuracy	Natural Accuracy	Isolating CLS	Patch Average	Patch Maximum
Top 1	84.20	78.61	75.75	80.16
Top 5	97.16	94.18	90.99	95.65

Based on the preservation of spatial information in patches, we hypothesize that the CLS token plays a relatively minor role throughout the network and is not used for globalization until the last layer. To demonstrate this, we perform inference on images without using the CLS token in layers 1-11, meaning that in these layers, each patch only attends to other patches and not to

the CLS token. At layer 12, we then insert a value for the CLS token so that other patches can attend to it and vice versa. This value is obtained by running a forward pass using only the CLS token and no image patches; this value is constant across all input images.

The resulting hacked network that only has CLS access in the last layer can still successfully classify 78.61% of the ImageNet validation set as shown in Table 7.1. From this result, we conclude that the CLS token captures global information mostly at the last layer, rather than building a global representation throughout the network.

We perform a second experiment to show this last-layer globalization behaviour is not exclusive to the CLS token, but actually occurs across every patch in the last layer. We take the fully connected layer trained to classify images on top of the CLS token, and without any fine-tuning or adaptation, we apply it to each patch, one at a time. This setup still successfully classifies 75.75% of the validation set, on average across individual patches, and the patch with the maximum performance achieves 80.16% accuracy (see Table 7.1), further confirming that the last layer performs a token mixing operation so that all tokens contain roughly identical information. Figure 7.5 contains a heat-

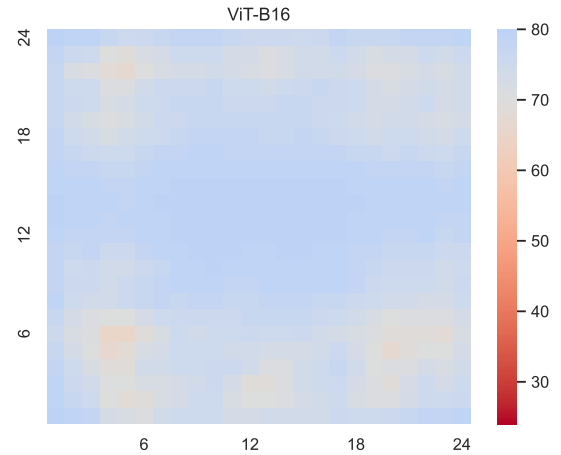


Figure 7.5: Heat map of classification accuracy on the validation set when we apply the classification head trained to classify images on the top of the CLS token to the other patches.

map depicting the performance of this setup across spatial patches. This observation stands in stark contrast to the suggestions of Raghu et al. [156] that ViTs possess strong localization

throughout the entire network, and their further hypothesis that the addition of global pooling is required for mixing tokens at the end of the network.

We conclude by noting that the information structure of a ViT is remarkably similar to a CNN, in the sense that the information is positionally encoded and preserved until the final layer. Furthermore, the final layer in ViTs appears to behave as a learned global pooling operation that aggregates information from all patches, which is similar to its explicit average-pooling counterpart in CNNs.

7.5 Comparison of ViTs and CNNs

As extensive work has been done to understand the workings of convolutional networks, including similar feature visualization and image reconstruction techniques to those used here, we may be able to learn more about ViT behavior via direct comparison to CNNs. An important observation is that in CNNs, early layers recognize color, edges, and texture, while deeper layers pick out increasingly complex structures eventually leading to entire objects [144]. Visualization of features from different layers in a ViT, such as those in Figures 7.1 and 7.6, reveal that ViTs exhibit this kind of progressive specialization as well.

On the other hand, we observe that there are also important differences between the ways CNNs and ViTs recognize images. In particular, we examine the reliance of ViTs and CNNs on background and foreground image features using the bounding boxes provided by ImageNet [40]. We filter the ImageNet-1k training images and only use those which are accompanied by bounding boxes. If several objects are present in an image, we only take the bounding boxes corresponding to the true class label and ignore the additional bounding boxes. Figure 7.7(b)

shows an example of an image and variants in which the background and foreground, respectively, are masked.

Figure 7.7(a) displays an example of ViTs’ ability to detect background information present in the ImageNet dataset. This particular feature appears responsible for recognizing the pairing of grass and snow. The rightmost panel indicates that this feature is solely activated by the background, and not at all by the patches of the image containing parts of the wolf.

To quantitatively assess each architecture’s dependence on different parts of the image on the dataset level, we mask out the foreground or background on a set of evaluation images using the aforementioned ImageNet bounding boxes, and we measure the resulting change in top-5 accuracy. These tests are performed across a number of pretrained ViT models, and we compared to a set of common CNNs in Table 7.2. Further results can be found in Table 14.1.

We observe that ViTs are significantly better than CNNs at using the background information in an image to identify the correct class. At the same time, ViTs also suffer noticeably less from the removal of the background, and thus seem to depend less on the background information to make their classification. A possible, and likely, confounding variable here is the imperfect separation of the background from the foreground in the ImageNet bounding box data set. A rectangle containing the wolf in Figure 7.7(a), for example, would also contain a small amount of the grass and snow at the wolf’s feet. However, the foreground is typically contained entirely in a bounding box, so masking out the bounding box interiors is highly effective at removing the foreground. Because ViTs are better equipped to make sense of background information, the leaked background may be useful for maintaining superior performance. Nonetheless, these results suggest that ViTs consistently outperform CNNs when information, either foreground or background, is missing.

Next, we study the role of texture in ViT predictions. To this end, we filter out high-frequency components from ImageNet test images via low-pass filtering. While the predictions of ResNets suffer greatly when high-frequency texture information is removed from their inputs, ViTs are seemingly resilient. See Figure 14.5 for the decay in accuracy of ViT and ResNet models as textural information is removed.

Table 7.2: **ViTs more effectively correlate background information with correct class.** Both foreground and background data are normalized by full image top-5 accuracy.

Architecture	Normalized Top-5 ImageNet Accuracy		
	Full Image	Foreground	Background
ViT-B32	98.44	93.91	28.10
ViT-L16	99.57	96.18	33.69
ViT-L32	99.32	93.89	31.07
ViT-B16	99.22	95.64	31.59
ResNet-50	98.00	89.69	18.69
ResNet-152	98.85	90.74	19.68
MobileNetv2	96.09	86.84	15.94
DenseNet121	96.55	89.58	17.53

7.6 ViTs with Language Model Supervision

Recently, ViTs have been used as a backbone to develop image classifiers trained with natural language supervision and contrastive learning techniques [155]. These CLIP models are state-of-the-art in transfer learning to unseen datasets. The zero-shot ImageNet accuracy of these models is even competitive with traditionally trained ResNet-50 competitors. We compare the feature visualizations for ViT models with and without CLIP training to study the effect of natural language supervision on the behavior of the transformer-based backbone.

The training objective for CLIP models consists of matching the correct caption from a list of options with an input image (in feature space). Intuitively, this procedure would require the

network to extract features not only suitable for detecting nouns (e.g. simple class labels like ‘bird’), but also modifying phrases like prepositions and epithets. Indeed, we observe several such features that are not present in ViTs trained solely as image classifiers.

Figure 7.8(a) shows the image optimized to maximally activate a feature in the fifth layer of a ViT CLIP model alongside its two highest activating examples from the ImageNet dataset. The fact that all three images share sharp boundaries indicates this feature might be responsible for detecting caption texts relating to a progression of images. Examples could include “before and after,” as in the airport images or the adjective “step-by-step” for the iPod teardown. Similarly, Figure 7.8(b) and 7.8(c) depict visualizations from features which seem to detect the preposition “from above”, and adjectives relating to a multitude of the same object, respectively.

The presence of features that represent conceptual categories is another consequence of CLIP training. Unlike ViTs trained as classifiers, in which features detect single objects or common background information, CLIP-trained ViTs produce features in deeper layers activated by objects in clearly discernible conceptual categories. For example, the top left panel of Figure 7.9(a) shows a feature activated by what resembles skulls alongside tombstones. The corresponding seven highly activating images from the dataset include other distinct objects such as bloody weapons, zombies, and skeletons. From a strictly visual point of view, these classes have very dissimilar attributes, indicating this feature might be responsible for detecting components of an image relating broadly to morbidity. In Figure 7.9(b), we see that the top leftmost panel shows a disco ball, and the corresponding images from the dataset contain boomboxes, speakers, a record player, audio recording equipment, and a performer. Again, these are visually distinct classes, yet they are all united by the concept of music.

Given that the space of possible captions for images is substantially larger than the mere one

thousand classes in the ImageNet dataset, high performing CLIP models understandably require higher level organization for the objects they recognize. Moreover, the CLIP dataset is scraped from the internet, where captions are often more descriptive than simple class labels.

7.7 Discussion

In order to dissect the inner workings of vision transformers, we introduce a framework for optimization-based feature visualization. We then identify which components of a ViT are most amenable to producing interpretable images, finding that the high-dimensional inner projection of the feed-forward layer is suitable while the key, query, and value features of self-attention are not.

Applying this framework to said features, we observe that ViTs preserve spatial information of the patches even for individual channels across all layers with the exception of the last layer, indicating that the networks learn spatial relationships from scratch. We further show that the sudden disappearance of localization information in the last attention layer results from a learned token mixing behavior that resembles average pooling.

In comparing CNNs and ViTs, we find that ViTs make better use of background information and are able to make vastly superior predictions relative to CNNs when exposed only to image backgrounds despite the seemingly counter-intuitive property that ViTs are not as sensitive as CNNs to the loss of high-frequency information, which one might expect to be critical for making effective use of background. We also conclude that the two architectures share a common property whereby earlier layers learn textural attributes, whereas deeper layers learn high level object features or abstract concepts. Finally, we show that ViTs trained with language model

supervision learn more semantic and conceptual features, rather than object-specific visual features as is typical of classifiers.

Reproducibility Statement

We make our code repository available at: <https://github.com/anonymous2023iclr/>

ViTVis

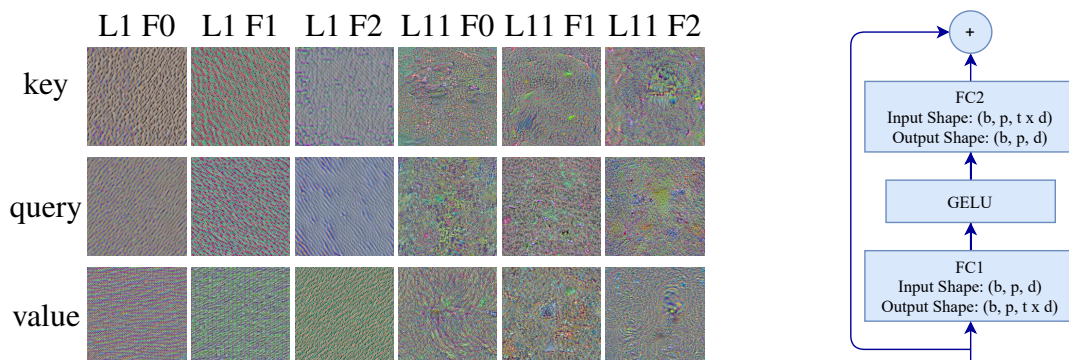


Figure 7.3: *Left: Visualization of key, query, and value.* The visualization both fails to extract interpretable features and to distinguish between early and deep layers. High-frequency patterns and adversarial behavior dominate. *Right: ViT feed forward layer.* The first linear layer increases the dimension of the feature space, and the second one brings it back to its initial dimension.

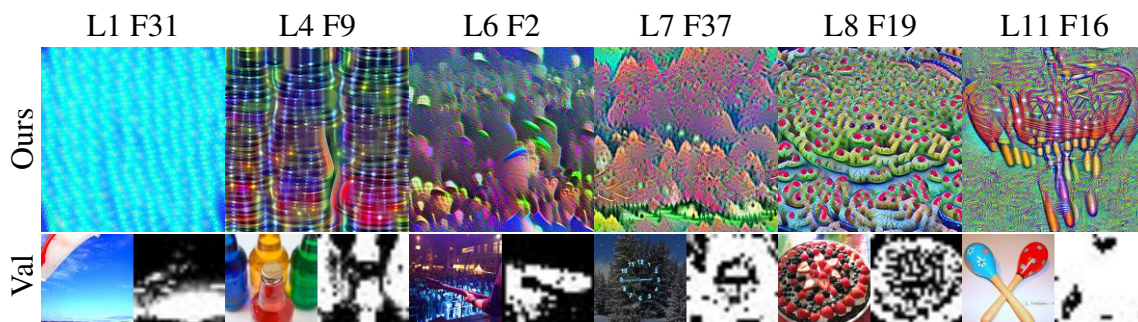


Figure 7.4: **Feature activation maps in internal layers can effectively segment the contents of an image with respect to a semantic concept.** For each image triple, the visualization on top shows the result of our method, the image on the bottom left is the most activating image from the validation set and the image on the bottom right shows the activation pattern.

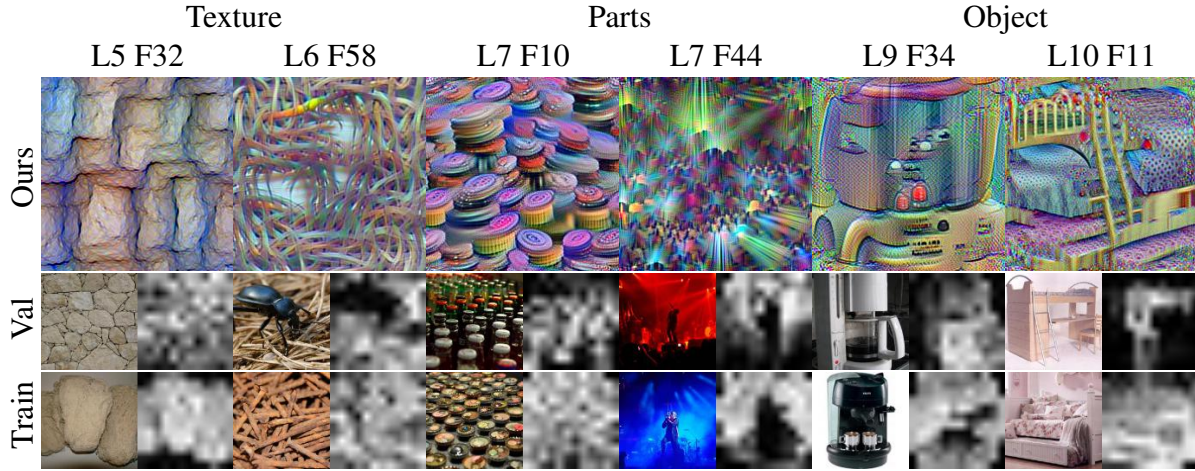


Figure 7.6: **Complexity of features vs depth in ViT B-32.** Visualizations suggest that ViTs are similar to CNNs in that they show a feature progression from textures to parts to objects as we progress from shallow to deep features.



Figure 7.7: (a): **ViT-B16 detects background features.** *Left:* Image optimized to maximally activate a feature from layer 6. *Center:* Corresponding maximally activating example from ImageNet. *Right:* The image's patch-wise activation map. (b): **An example of an original image and masked-out foreground and background.**

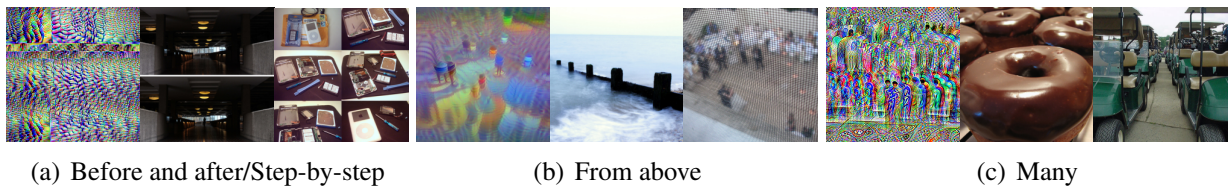


Figure 7.8: *Left:* Feature optimization shows sharp boundaries, and maximally activating ImageNet examples contain distinct, adjacent images. *Middle:* Feature optimization and maximally activating ImageNet photos all show images from an elevated vantage point. *Right:* Feature optimization shows a crowd of people, but maximally activating images indicate that the repetition of objects is more relevant than the type of object.

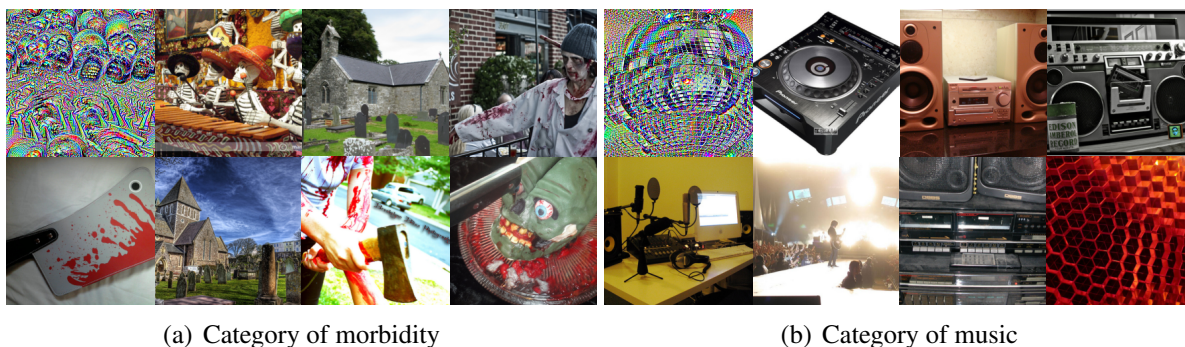


Figure 7.9: **Features from ViT trained with CLIP that relates to the category of morbidity and music.** *Top-left image in each category:* Image optimized to maximally activate a feature from layer 10. *Rest:* Seven of the ten ImageNet images that most activate the feature.

Chapter 8: Feature Sonification

8.1 Basic sonification

A phoneme is the most basic unit of sound that composes a word (in English there are 44 distinct phonemes) [72]. Given its atomicity, we expect that sonification of lower level features would reveal phoneme-like sounds. The following demonstration shows an example of this. For every figure in the paper, you can also visit our blog for more examples and for the audio parts related to the figure.

8.2 Jasper Network

This work focuses on the Jasper network [118], the progenitor in a line of state-of-the-art ASR architectures based on convolutions [67, 69, 98]. Jasper takes Mel-filterbanks as temporal input channels and outputs a probability distribution over characters per frame. It is widely believed that Mel-filterbanks capture the most important features of audio while discarding unnecessary complexity of the waveforms [97]. To extract the Mel-filterbanks from a raw waveform, we divide the waveform into overlapping 20ms frames of audio. Every frame has 10ms overlap with its proceeding frame. For every frame, the short time Fourier transform is computed. Then, groups of fourier magnitudes with similar frequencies are averaged together to compute the Mel-filterbanks.

While ASR models are encoder-decoder architectures, we only focus on the encoder. This is because Jasper’s decoder is only 1 convolutional layer that converts the features produced by the encoder into probability distributions over characters for each frame Equivalent to the fully-connected layer in classifiers. The encoder is responsible for decomposing and interpreting

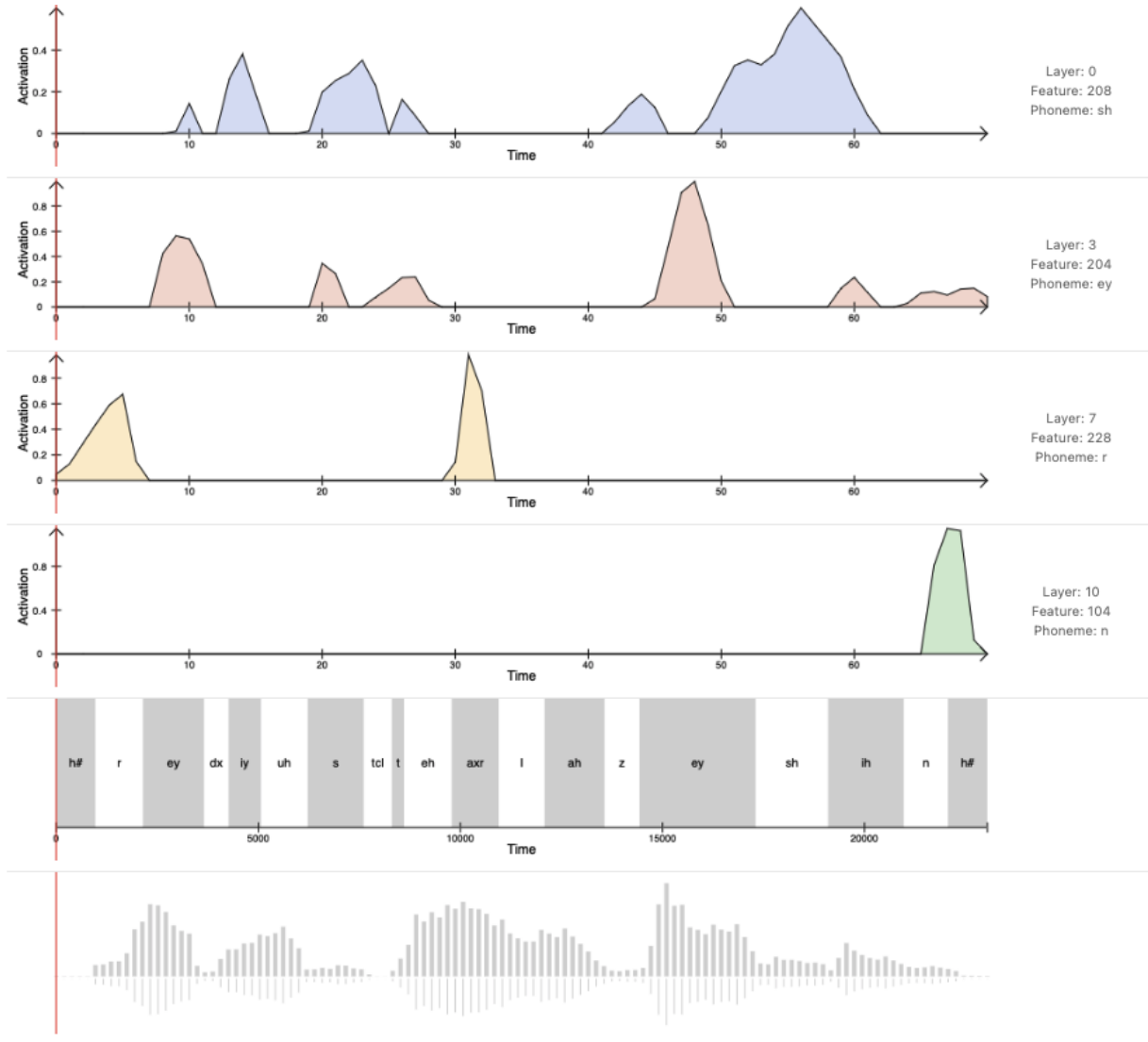


Figure 8.1: Sonification of the Jasper [118] network. The first four rows show different shallow features that correspond to four different phonemes in English. The y-axis shows the activation and the x-axis shows the time (temporal dimension). The fifth row shows the phonemes in the audio over time. The last row plots the amplitude of the waveform. Layer 0 is the shallowest layer in the network. The input to network for this example is "radiosterilization"

speech, and is thus the focus of our study. As mentioned before, the output of Jasper is a probability distribution over characters per frame. For instance, in an utterance of the word "five," many consecutive frames might correspond to the phoneme /f/. As a result, Jasper's sequence of predictions might be "FFFIVE". To avoid this pitfall, Connectionist Temporal Classification

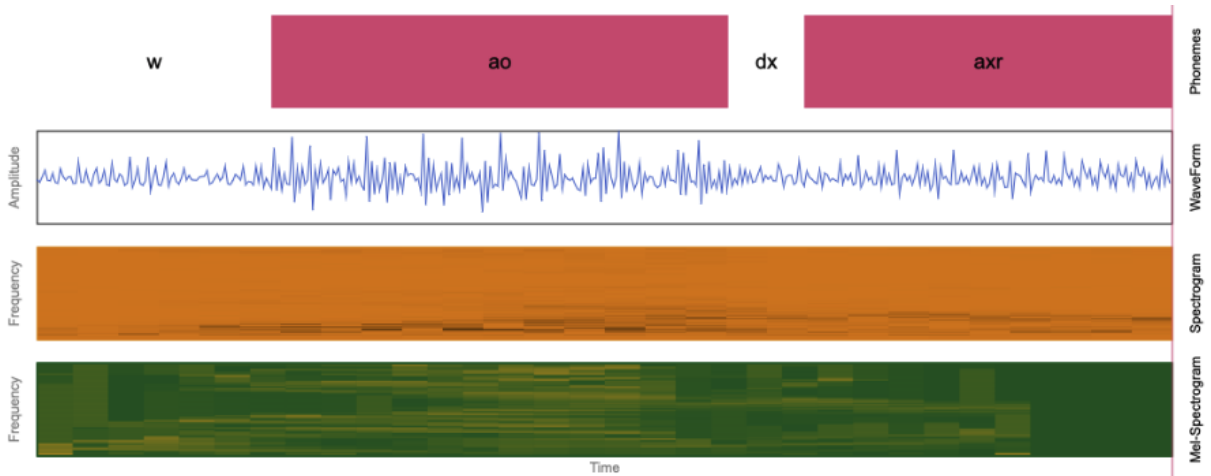


Figure 8.2: Examples of waveform, spectrogram, and Mel-spectrogram. It is widely known that Mel-spectrogram reduces the complexity of waveform while at the same time, being more expressive than spectrogram [97]. In all sub-plots, x-axis shows the time (temporal dimension). For the waveform, y-axis shows the amplitude. For the spectrogram and Mel-spectrogram, y-axis shows the frequency.

(CTC) decoders are used. CTC decoders use dynamic programming to get around the issue of not knowing the alignment between the input and the output. Rather, they search over various alignments of phonemes to identify an optimal phoneme sequence without redundancy [70].

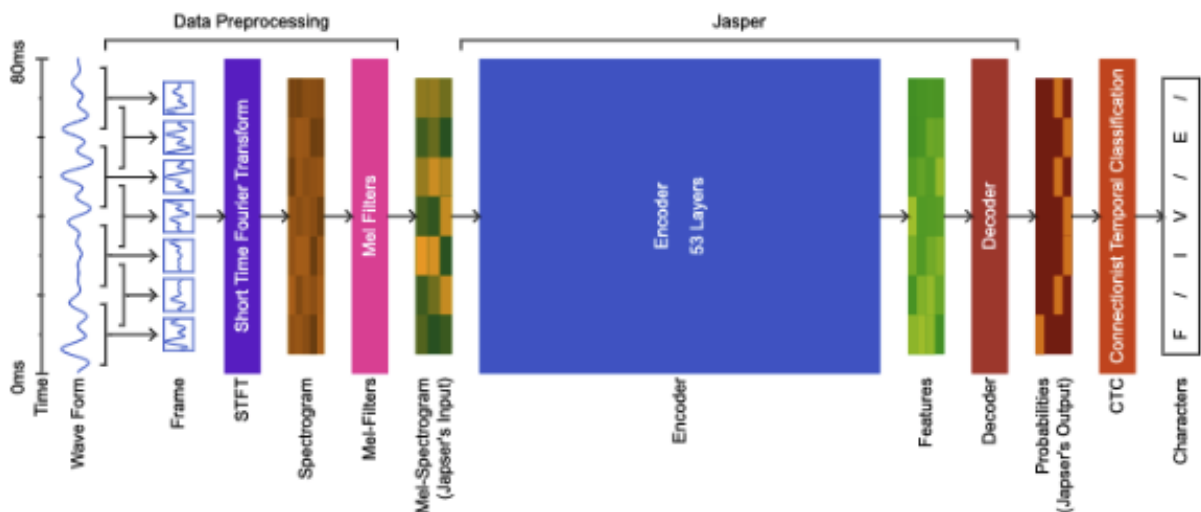


Figure 8.3: A complete overview of the ASR pipeline using the Jasper network.

The encoder stacks 10 blocks, each consisting of 5 convolutional sub-blocks (1-d convolution layer, batch normalization, ReLU, and dropout). Residual connections connect together the output of previous blocks and the input of the current block's last layer. The complete architecture of the encoder is shown in the following figure. Throughout this article, we refer to sub-blocks and layers interchangeably, and we sonify the ReLU output in each layer.

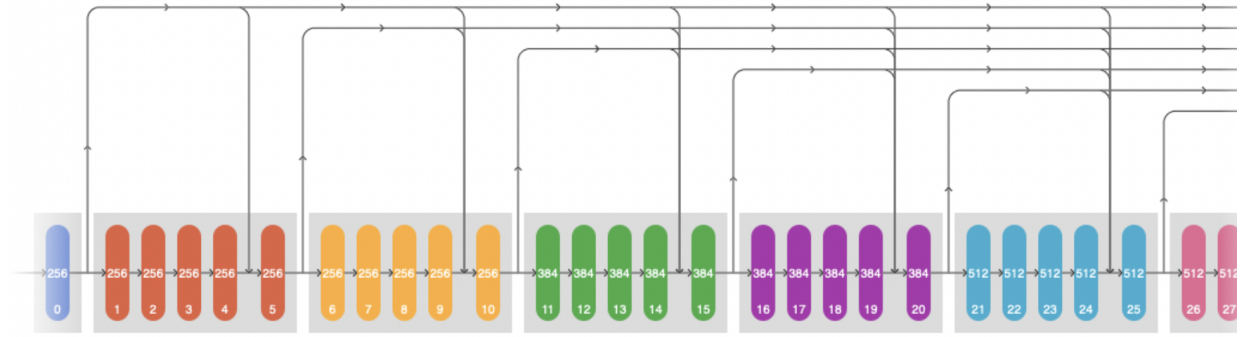


Figure 8.4: The structure of Jasper encoder. Every gray box represents a block, and every colored box represents a layer (sub-block). The numbers at the center of layers show the number of channels. The numbers at the bottom of layers show the layer's number.

/subsection Activation visualization

Let us dive into the internal representations of Jasper as it processes 10 randomly sampled audio files from the Timit [53] dataset. For the first 6 layers of Jasper, we plot each feature's activation strength over time. Shallow layers (such as layers 0 and 1) have dense, spiky activation patterns. Conversely, deeper layers (such as layers 2 to 4), have relatively sparser activations that span longer periods of time. Intriguingly however, at layer 5 the features return to the spiky pattern we saw in layers 0 and 1. We attribute this phenomenon to the skip connection between the inputs of layer 1 (first layer of block) and layer 5 (last layer of block).

This pattern pervades deeper blocks as well; the first and last layers of each block maintain similar feature profiles. Layers 11, 14, and 15 are accordingly first, fourth, and last layers of

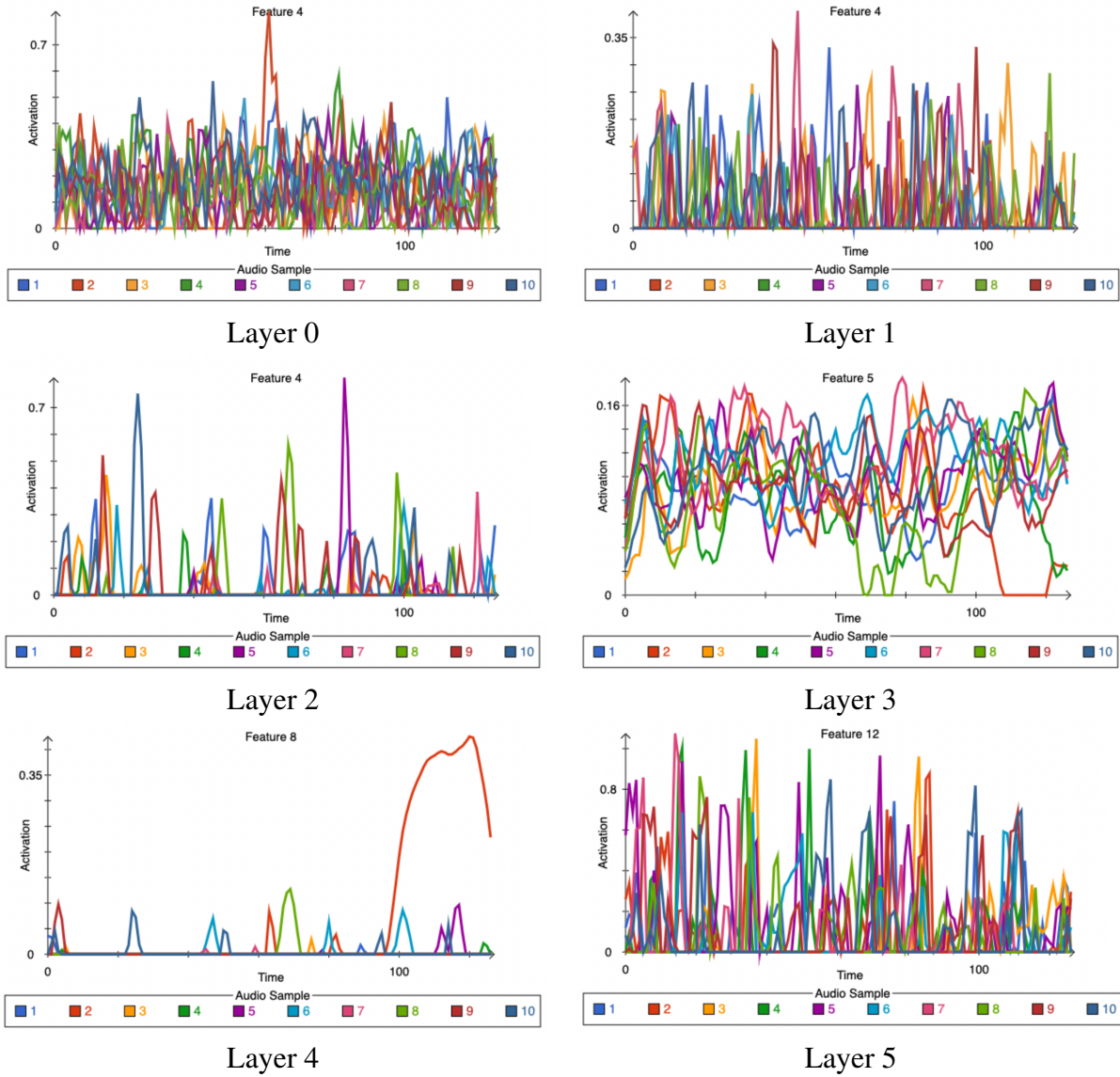


Figure 8.5: The activation patterns of features for the shallow layers.

block 3. Note that layers 11 and 15 have similar activation patterns, where layer 14 has relatively sparser activations that span longer periods of time.

As we go deeper, the profiles in the penultimate layer of each block become more sparse. For instance, the following figure shows the activations for the 4th layer in various blocks of the network.

The trend suggests that the shallow layers of the network detect simpler patterns of audio

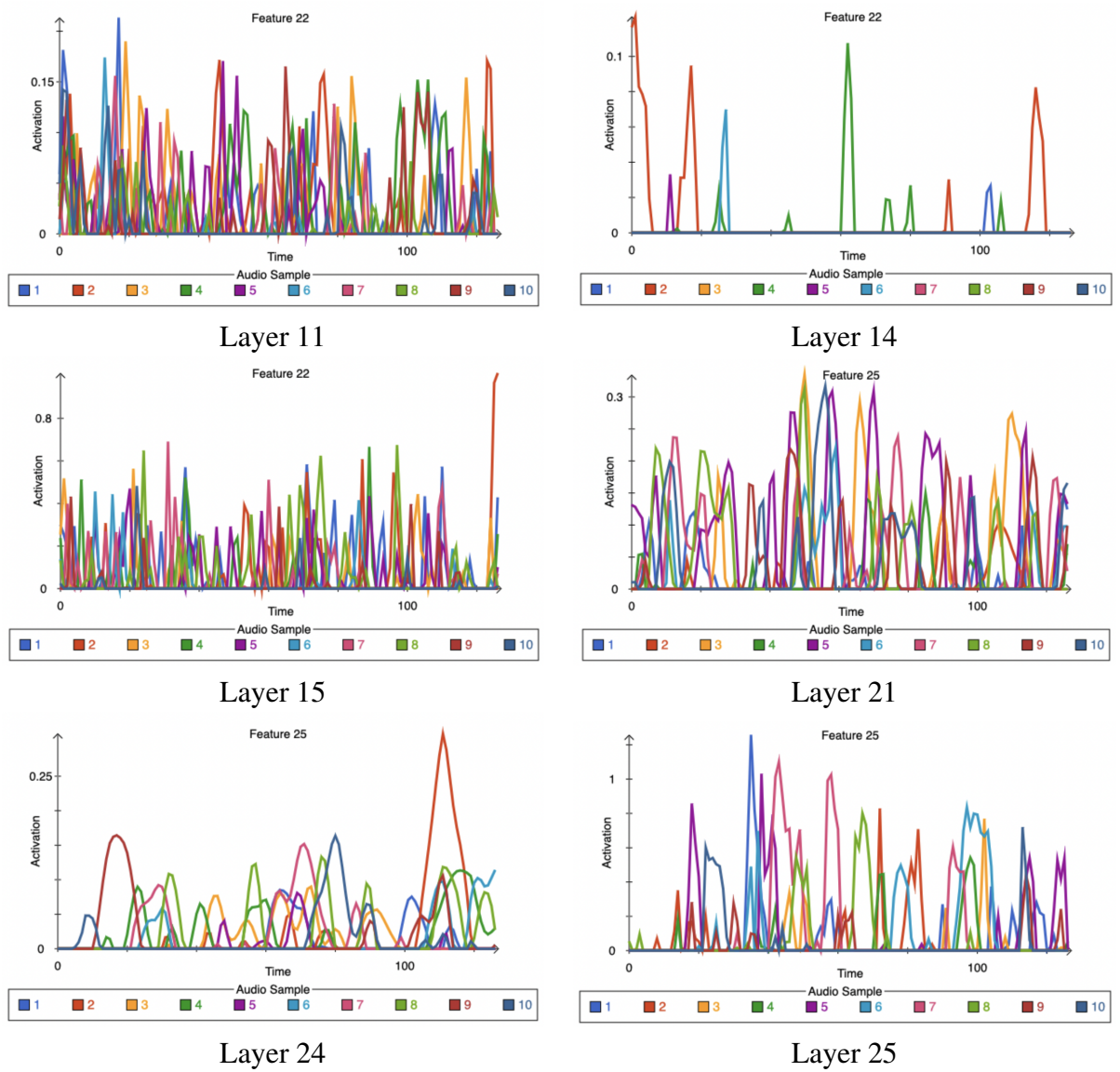


Figure 8.6: The activation patterns of features for the first and last layers of several blocks.

which occur in shorter periods of time and with higher frequency, whereas the deep layers detect more complex patterns of audio which occur over longer periods of time and with lower frequency. We test this hypothesis in the following section by sonifying features from various parts of the network, and categorizing the resulting audios.

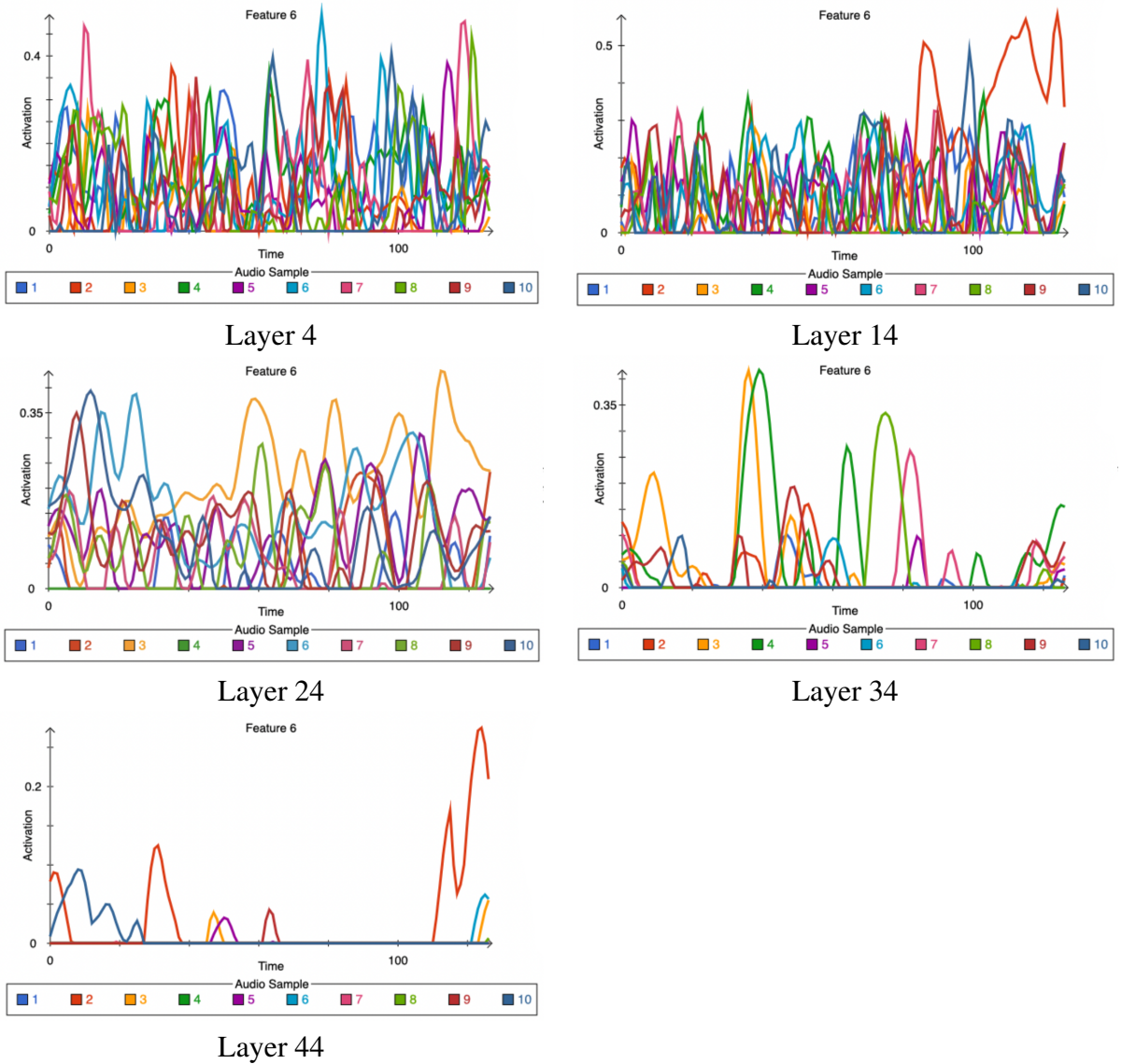


Figure 8.7: The activation patterns of features for the last layer of several blocks.

8.3 Sonification using an Auxiliary Dataset

A simple way of doing feature visualization in images is to find image patches that maximally activate a particular node or layer [144]. To do feature **sonification**, we do the analogous method with audio segments rather than image patches. We draw our audio from the Timit dataset [53], which contains a time-aligned breakdown of each phoneme and word appearing within an audio

clip along with other annotations such as speaker information. Then, when we cluster audio segments that activate the same node, we can use these annotations semantically to characterize that node.

Below, we pick a feature id (F) in layer (L) and find the 20 half-second audio segments that most strongly activate that feature. Then we characterize that feature at the word level or phoneme level by listing the most common words or phonemes that were present in the audio segments. We call these **word-level** and **phoneme-level** sonifications, respectively.

The following figure demonstrates word-level sonification for 7 features from shallow layers. The x-axis contains the 5 most activating words for that feature, while the y-axis indicates the number of times each word has appeared in the audio segments. The 20 audio segments can also be played.

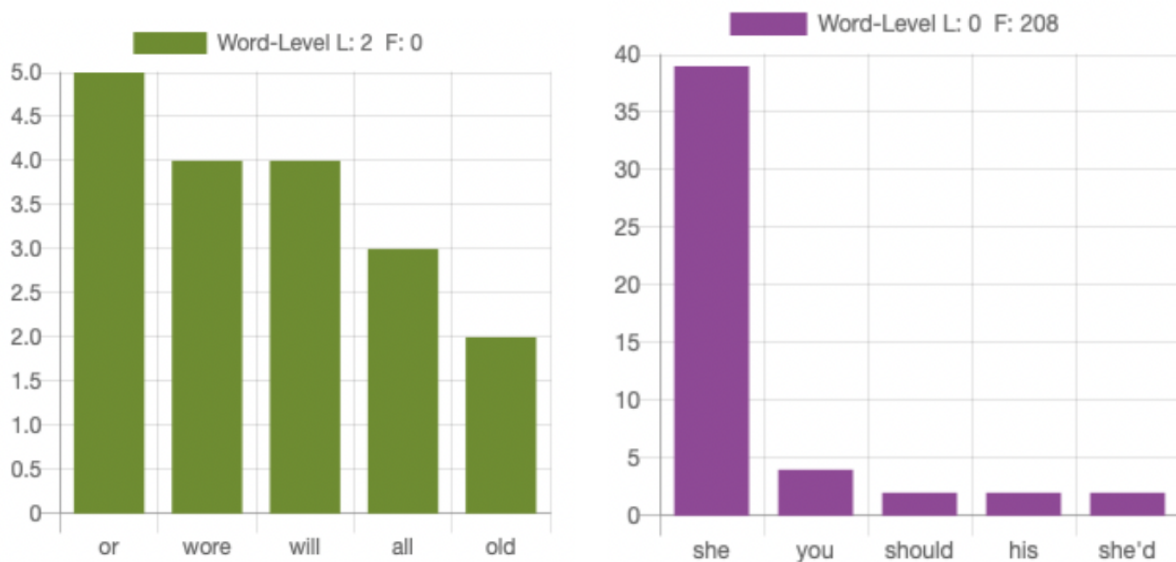


Figure 8.8: A few examples of most activating words (word-level sonification).

All of the previous examples were chosen from shallow layers. We hypothesize that such shallow features respond to certain phonemes. For instance, Example 1 seems to correspond

to utterances of the phoneme /o/. We test our pythoesis by phoneme-level sonification of such features. The following figure displays the phoneme-level sonification for the same exact features that were shown previously.

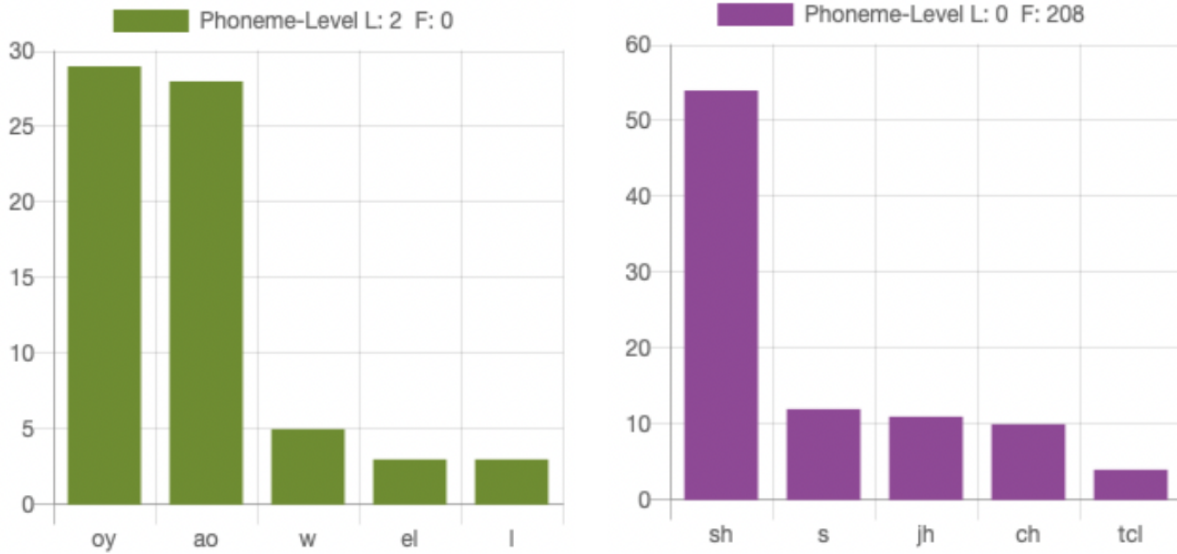


Figure 8.9: Few examples of most activating phonemes (phoneme-level sonification).

It is worth mentioning that in Example 1, most of the speakers in both of the phoneme-level and word-level sonifications were dominantly male speakers. In fact, many feature maps are mostly dominated by utterances from only one gender. The following figure displays a few examples of such features.

We see that, while phonemes play an important role in determining what activates a feature map, these shallow maps are also strongly effected by speaker-dependent nuisance variables such as pitch. Below, we explore where speaker-specific information lives in the network, and where information is predominantly semantic.

8.4 Speaker Information vs Speech Information

We observe that speaker information plays an important role in activating certain features. As an example of speaker information, we study the speaker's gender. More precisely, we study how many of the utterances that appear in a word-level sonification are utterances by male/female speakers. We also look at how many of the feature sonifications in each layer are dominated by only one gender. We investigate to see if a trend exists, and if so, what the trend looks like.

We assume that a feature is dominated by one gender if more than 90% of the utterances in the feature sonification are by that gender. The following plot shows the portion of features that are dominated by one gender in every layer.

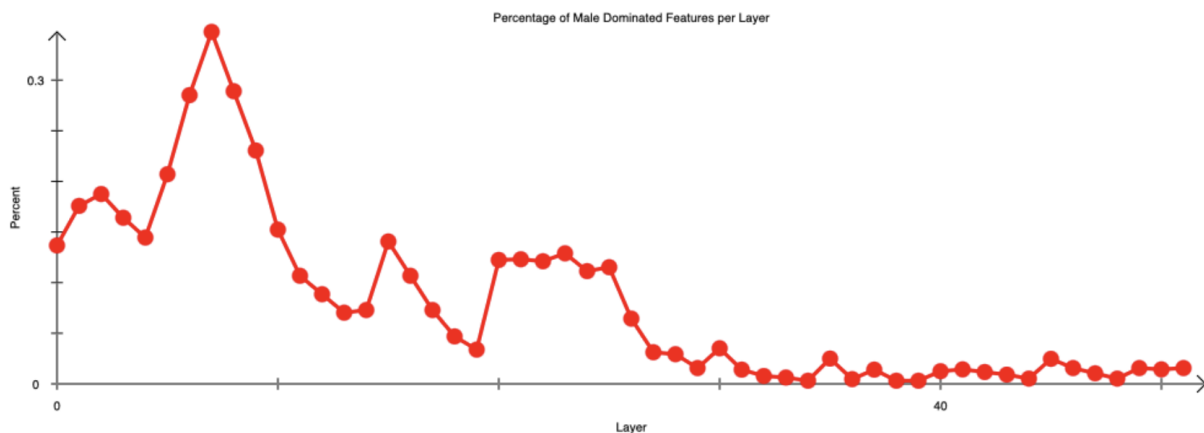


Figure 8.10: The portion of features that are dominated by the male speakers in every layer. The x-axis shows the layer's depth and the y-axis shows the percent of male-dominated features. As we go deeper in the network, the portion of male-dominated features becomes smaller. In other words, shallow layers react to both the speech information and the speaker information, where deep layers discard the speaker information and only focus on the speech information.

The previous figures indicate that in shallow layers, speaker information is important for the activation of features, however, this information is discarded in deeper layers. A similar study [116] also shows that ASR models discard speaker information in the deeper layers.

As a big picture, the following heatmap shows the gender dominance in every feature of the

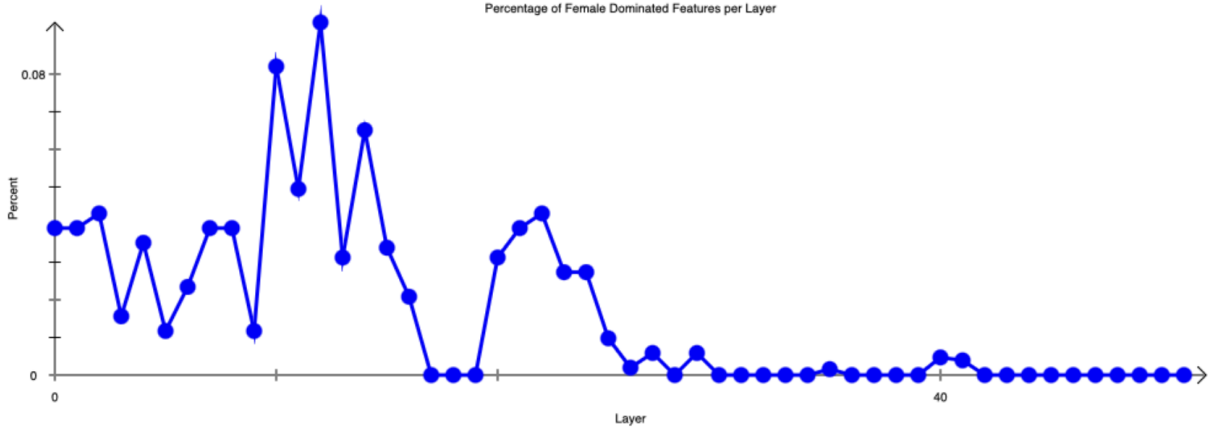


Figure 8.11: The portion of features that are dominated by female speakers in every layer. The trend is very similar to the trend in male speakers, however in general, fewer features have been dominated by female speakers.

network. The female-dominated, male-dominated, and neutral features are colored respectively blue, red, and yellow. Every column corresponds to one layer of the network.

Different layers of the network have different number of features. For instance, the first layer has only 256 features while the last layer has 896. Consequently, there are different numbers of cells in each column. In other words, the cells corresponding to the deeper layers have been squeezed so that the last columns would fit into the same space as the first column.

A fair conclusion would be that the information regarding the speaker is being discarded in deep layers. To sanity check our hypothesis, we take two different utterances of a same word, one with a male speaker, and another with a female speaker, to compare the features they activate.

For every feature we compute the expected value (average) of activation for every word in the dataset. We assume that a word is activating a feature, if it activates the feature map at least 1.5 times greater than the expected value.

As a result, we see disjoint sets of activated features in shallow layers, but common activated features in deep layers.

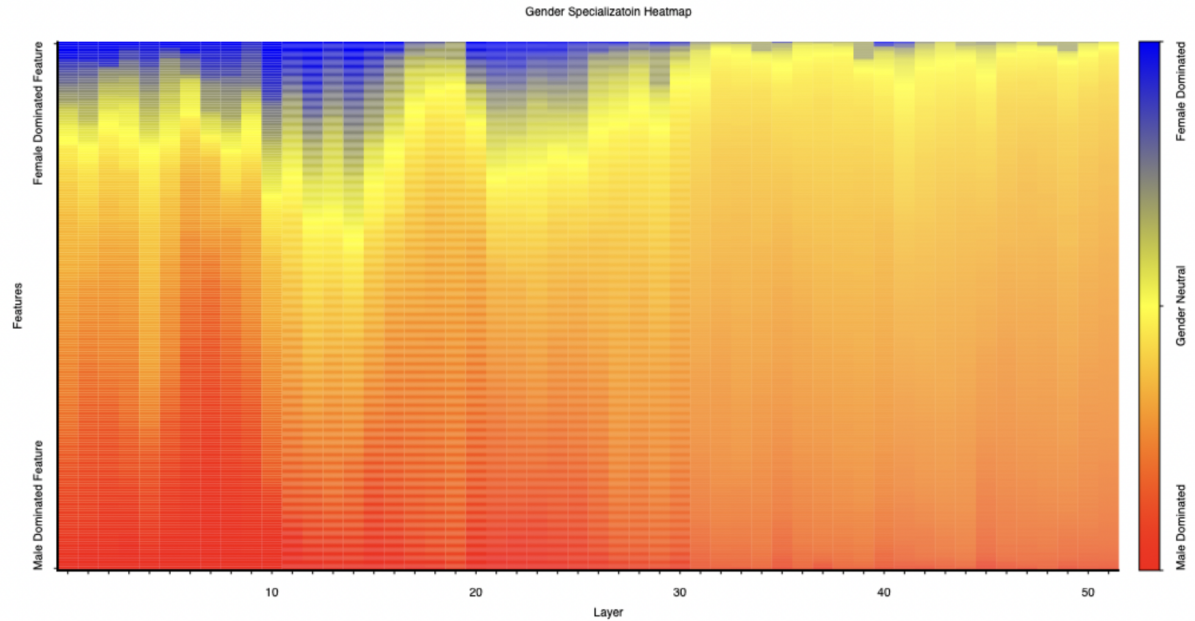


Figure 8.12: Heat-map of gender dominance for every feature in the network. There are fewer features dominated by one gender in the deep layers. However, it can be seen that the features tend to activate more by male speakers than female speakers.

The following plot illustrates the difference between activated features for the two different utterances. Every column represents a layer, the left-most represents the shallowest, and the right-most represents the deepest.

8.5 Discarding Speaker Information

As we saw in the previous section, shallow layers attend to the speaker information as well as the speech information. In this section, our objective is to sonify features in a way that depends less on the speaker information. One way of doing so is by finding words/phonemes that activate features on average, with averages taken across speakers. This is different from the sonifications above that focus on finding individual utterances that activate a feature the most. We call this new process the "speaker-discarded" sonification.

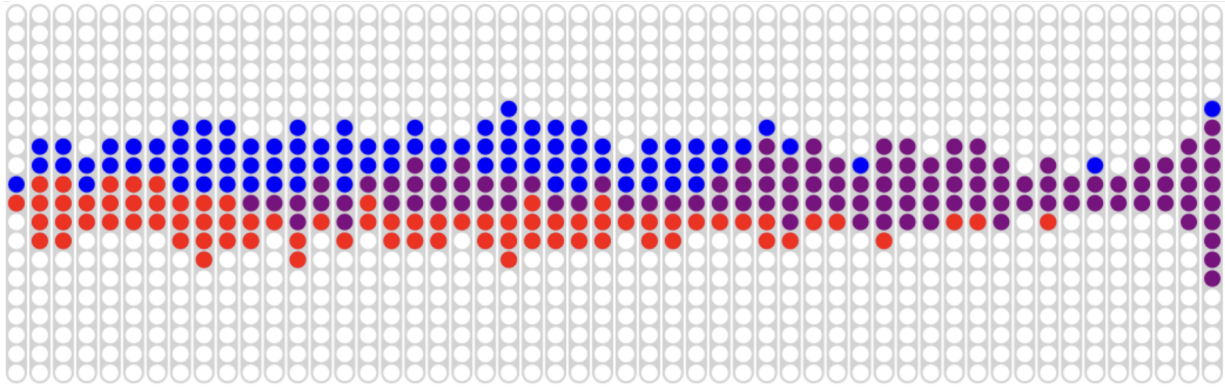


Figure 8.13: This figure shows the set of features that activate by utterances of a same word by speakers of different genders. Note that in shallow layers the set of features are almost disjoint, while deep activation patterns are nearly invariant to gender. Every layer is showing only 20 features. We made sure that when we randomly choose the features per each layer, the portion of features that get activated by each utterance is proportional to the total number of features in that layer. To make the figure more visually appealing, for each example we re-arrange the features in every layer so that features with similar node colors stay next to each other.

The following pseudo-code displays the complete steps of the speaker-discarded word-level sonification.

The following plot shows a few examples of the results using the speaker-discarded algorithm.

These results can be reproduced in Google Colab.

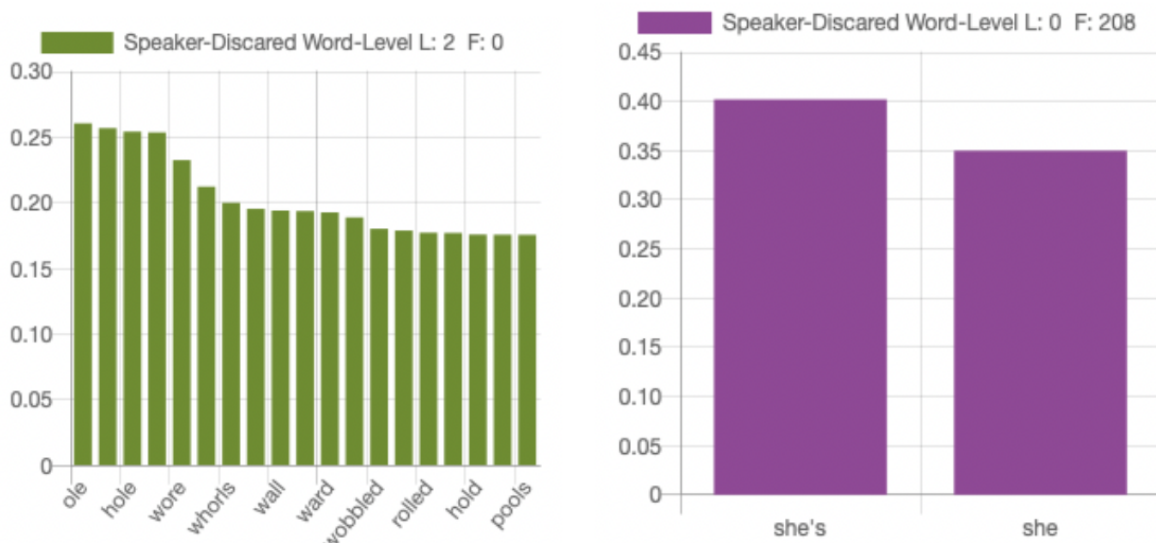


Figure 8.14: Few examples of word-level speaker-discarded sonification.

8.6 Dive Deeper

So far, we have shown that the features in shallow layers detect simple patterns like phonemes. In this section, we will show that deeper layers detect complex patterns such as combinations of phonemes or even words.

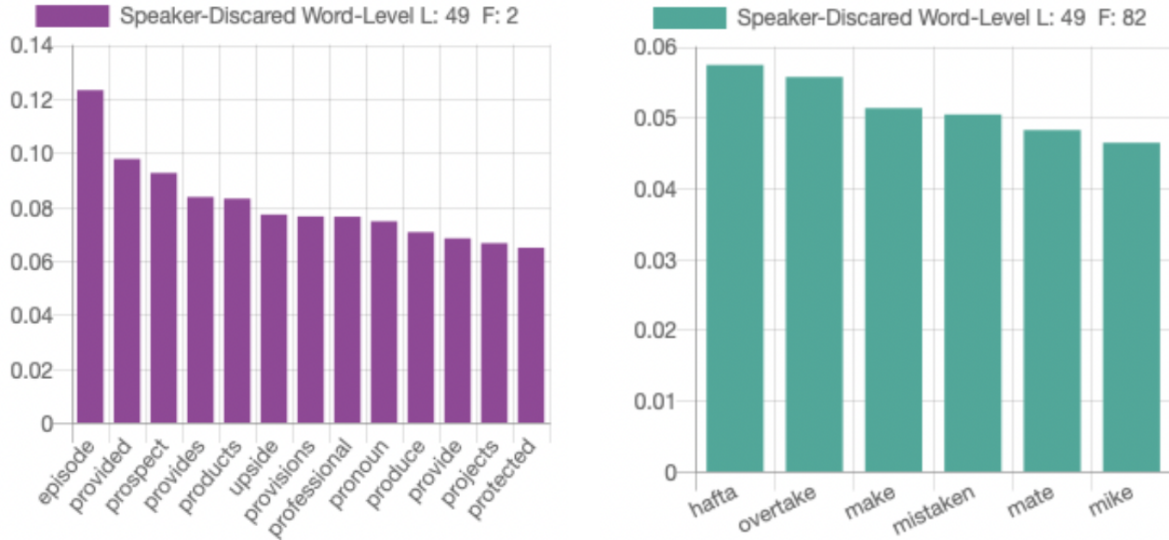


Figure 8.15: Few examples of word-level speaker-discarded sonification for deep layers.

We assume that the average number of phonemes in word-level sonification is a good indicator of the complexity of the patterns that activate a particular feature. For this reason, we adopt the average number of phonemes per activating word as a metric of feature complexity. Similarly, we define a layer-level average phoneme count as the average of its features' phoneme count complexities. We also compute the average phoneme count of the feature with the highest/lowest phoneme count complexity for every layer. Our hypothesis is that shallow layers detect less complex patterns than deep layers.

To verify our hypothesis quantitatively, we plot the min, average, and max, phoneme count for every layer.

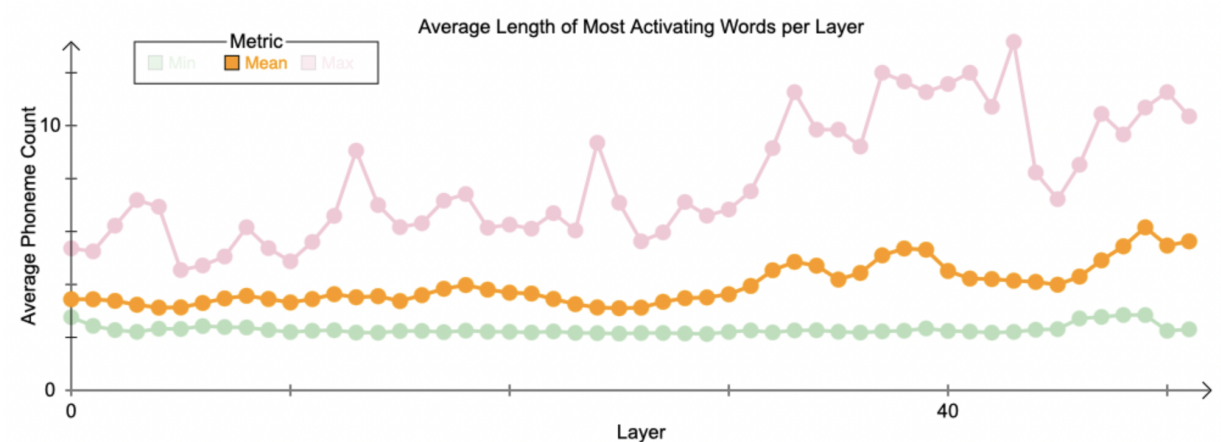


Figure 8.16: Min, Mean, and Max, phoneme count in word-level sonification. The Mean plot shows the layer-level average phoneme count. The Min and Max plots show the average phoneme count for the features with the lowest and highest values, respectively. The x-axis shows the layer depth, and the y-axis shows the phoneme count. Deep layers seem to be activated by more complex combinations of phonemes (i.e. larger average phoneme counts). This effect is easily heard by listening to the audio clips for each layer. For this purpose, please visit our blog.

From the previous figure, it can be seen that shorter words activate shallow layers, while longer words activate deep layers.

We can also verify that by listening to the audio sampled for each layer. For this purpose, please visit our blog.

To make a stronger case, we use phoneme-level sonification to show that deep layers activate via more complex combination of phonemes. For this purpose, we make an audio dataset consisting of various combinations of phonemes. For instance, we take every 3 consecutive phonemes in the Timit dataset to make a 3-gram phoneme dataset. The following figure shows the min, average, and max, phoneme count using the union of 1-gram, 3-gram, 5-gram, 7-gram, and 9-gram phoneme datasets. We observe a trend similar to word-level sonification.

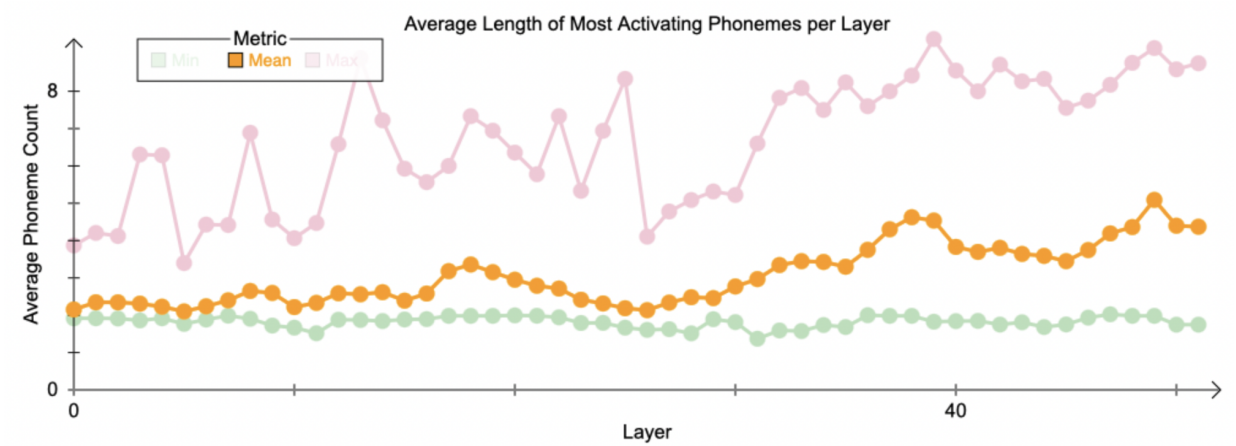


Figure 8.17: Average phoneme count per layer using the union of 1-gram, 3-gram, 5-gram, 7-gram, and 9-gram phoneme datasets. The x-axis shows the layer depth, and the y-axis shows the average phoneme count.

8.7 Sonification of the Whole Network

In this section, we display a complete sonification for every feature in every layer of the Jasper network. We believe that it could be helpful for readers who like to explore beyond what we explained through this article. The following figure displays phoneme-level, word-level, and speaker-discarded sonifications, respectively.

Some features are always inactive regardless of the input audio. For such features, the phrase "Dead Neuron" is being displayed.

Chapter 9: Conclusion

In this thesis, we studied the susceptibility of deep neural networks to adversarial attacks. From an adversary’s perspective, we showed that carefully crafted noise could even break certified robust defenses. While we are far away from a guaranteed defense, we showed that there are still great promising directions that one could take to defend their network against malicious examples. Furthermore, we provided solutions that require considerably less training data or computation. We also showed that revisiting old regularizers, such as weight decay, can be fruitful in settings with adequate training data and computation. This thesis raises awareness for industries and applications that want to use deep neural networks for security-sensitive applications.

We also provided tools for studying and understanding the building block of deep neural networks. We showed that well-crafted visualization methods could reveal deep neural networks’ understanding of real-world objects. Using these visualization techniques, we studied the similarities between CNNs and ViTs in great detail. We also compared ViTs trained with labeled data with ViTs trained using an NLP model’s supervision. Last but not least, we showed that these visualization techniques are not limited to networks trained for vision tasks, and they can be expanded to models that process other types of data, such as audio. We hope the tools developed in this thesis will help the machine-learning community better understand deep neural networks, and we hope it paves the way for improving deep learning models.

Appendix 10: Appendix For Adversarially Robust Transfer Learning

10.1 Experiment details

10.1.1 LWF-based experiments

In our LWF-based experiments, we use a batch-size of 128, a fixed learning-rate of $1e-2m$, and fine-tune for an additional 20,000 iterations. The first 10,000 iterations are used for warm-start; during which we only update the final fully connected layer’s weights. During the remaining 10,000 iterations, we update all of the weights but do not update the batch-normalization parameters.

10.1.2 ImageNet to CIFAR experiments

When freezing the feature extractor and fine-tuning on adversarial examples, we train the last fully connected layer’s weights for 50 epochs using batch-size=128. We start with an initial learning rate of 0.01 and drop the learning rate to 0.001 at epoch 30. In the case of fine-tuning on adversarial examples, we generate the adversarial examples using a 3 step PGD attack with step-size 3 and a perturbation bound $\epsilon = 5$.

10.1.3 Free training experiments

In all of our free-training experiments where we train the u-ResNet-50, we train for 90 epochs using a batch-size of 128. The initial learning rate used is 0.1 and we drop it by a factor of 10 at epochs 30 and 60. We use a replay parameter $m = 4$ and perturbation bound $\epsilon = 5$.

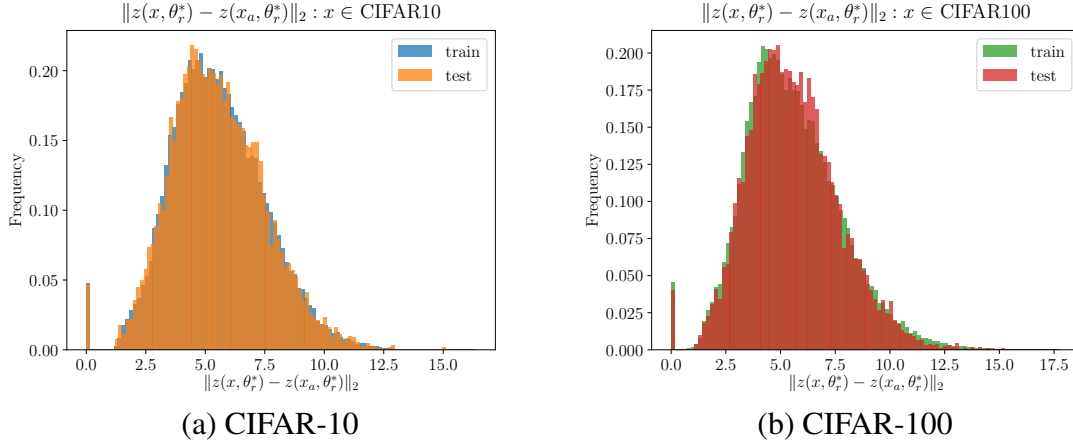


Figure 10.1: Figures 10.2 and ?? both show that the values of $\|z(x, \theta_r^*), z(x_a, \theta_r^*)\|_2$ are high most of the time, consequently, LwF is better done without data augmentation.

10.2 The distance between feature representations of natural images and augmented images

To speed up the LwF experiments, we did not use data augmentation during training. Instead of computing the robust feature representations on the fly, before starting training on the new target task, we passed the entire training data of the target task through the robust network and stored the feature representation vector. If we were doing data augmentation, we would have to pass the entire augmented training data through the network, which would be slow and memory intensive. Alternatively, we could use the robust feature representation of the non-augmented images instead. The latter would have been feasible if the distance between the robust feature representations of the non-augmented and augmented images were very small. However, as shown in fig 10.1, this quantity is not often negligible.

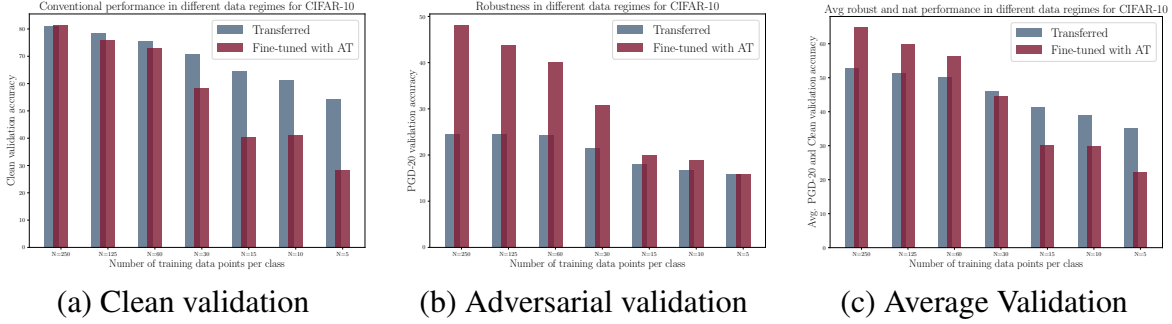


Figure 10.2: When the number of training data per-class is very limited (right bars), adversarially robust transfer learning [Transferred] is better overall. However, as the number of training data increases (left bars), fine-tuning with adversarial examples of the target domain [Fine-tuned with AT] results in an overall better performing model. Adversarially robust transfer learning is $3\times$ faster than fine-tuning with adversarial examples of the target domain.

10.3 Low-data regime transfer learning from ImageNet to CIFAR-10

In section 2.4.2, we illustrated that robust transfer learning is most beneficial where we have limited data (i.e., limited number of training data per class). We used CIFAR-100 as an illustrative target dataset. However, the overall results are not data set dependent. When we have limited number of training data, robust transfer learning results in the highest overall performance (see Fig. 10.2 for transferring from a robust ImageNet model to smaller instances of CIFAR-10).

10.4 LwF-based robust transfer learning for similar source and target datasets

In Table 10.1 we conduct LwF experiments on the split CIFAR-100 task which is more suited for transfer learning due to the similarities between the source and target datasets. In these situations, the LwF regularizer on the feature representations still works and can improve generalization without becoming vulnerable to adversarial examples. If we take the average performance of the robust classifiers on the split tasks (average of robust half CIFAR-100 and the LwF setting model for $\lambda_d = 0.01$) we get $(63.32 + 64.96)/2 = 64.14\%$ average validation

accuracy and 20.42% average robustness which is comparable with the case that we had adversarially trained the entire CIFAR-100 dataset (Table 2.1).

Table 10.1: Distilling robust features using LwF for the split CIFAR-100 task. For reference, we have included the results from transfer learning by freezing the features at the bottom of the table.

Source $\rightarrow$ Target Dataset	Source Model	λ_d	val.	PGD-20
CIFAR-100+ (1/2) $\rightarrow$ CIFAR-100 (other 1/2)	robust	0.001	73.30%	1.92%
		0.005	66.96%	10.52%
		0.01	63.32%	15.68%
		0.1	55.14%	17.26%
CIFAR-100+ (1/2) $\rightarrow$ CIFAR-100+ (other 1/2)	natural	NA	71.44%	0.00%
	robust	NA	58.48%	15.86%

10.5 Improving generalization of the CIFAR-10 adversarially trained model

Similar to the case of improving the generalization for CIFAR-100, we use our LwF-based loss function to transfer from the robust CIFAR-10 domain to the natural CIFAR-10 domain. We summarize the results in Table 10.2.

Table 10.2: Decreasing generalization gap by transferring with LwF. For reference, last row shows results from adversarial training CIFAR-10. The ‘+’ sign refers to using augmentation. ($\epsilon = 8$)

Source $\rightarrow$ Target Dataset	Source Model	λ_d	val.	PGD-20	CW-20
CIFAR-10+ $\rightarrow$ CIFAR-10	robust	1e-5	88.16%	45.31%	46.15%
		5e-5	88.08%	46.24%	47.06%
		1e-4	87.81%	46.54%	47.29%
		5e-4	87.44%	46.36%	47.16%
		0.001	87.31%	46.27%	47.06%
		0.01	87.27%	46.09%	46.92%
		0.1	87.49%	46.11%	46.84%
CIFAR-10+	robust	NA	87.25%	45.84%	46.96%

Appendix 11: Appendix For Breaking Certified Defenses

In this section, we include the complete results of our ablation study. As we mentioned in section 4.3, we show results on a subset of CIFAR-10 dataset, including one example per class. Figure 11.1 shows that taking a few optimization steps is enough for the resulting images to look natural. Figure 11.2 and 11.3, respectively, show the effect of λ_s and λ_{TV} on the perceptibility of perturbations.



Figure 11.1: The first 10 steps of optimization and the original image when crafting attacks on CIFAR-10. See section 4.3 for the details of the experiments.

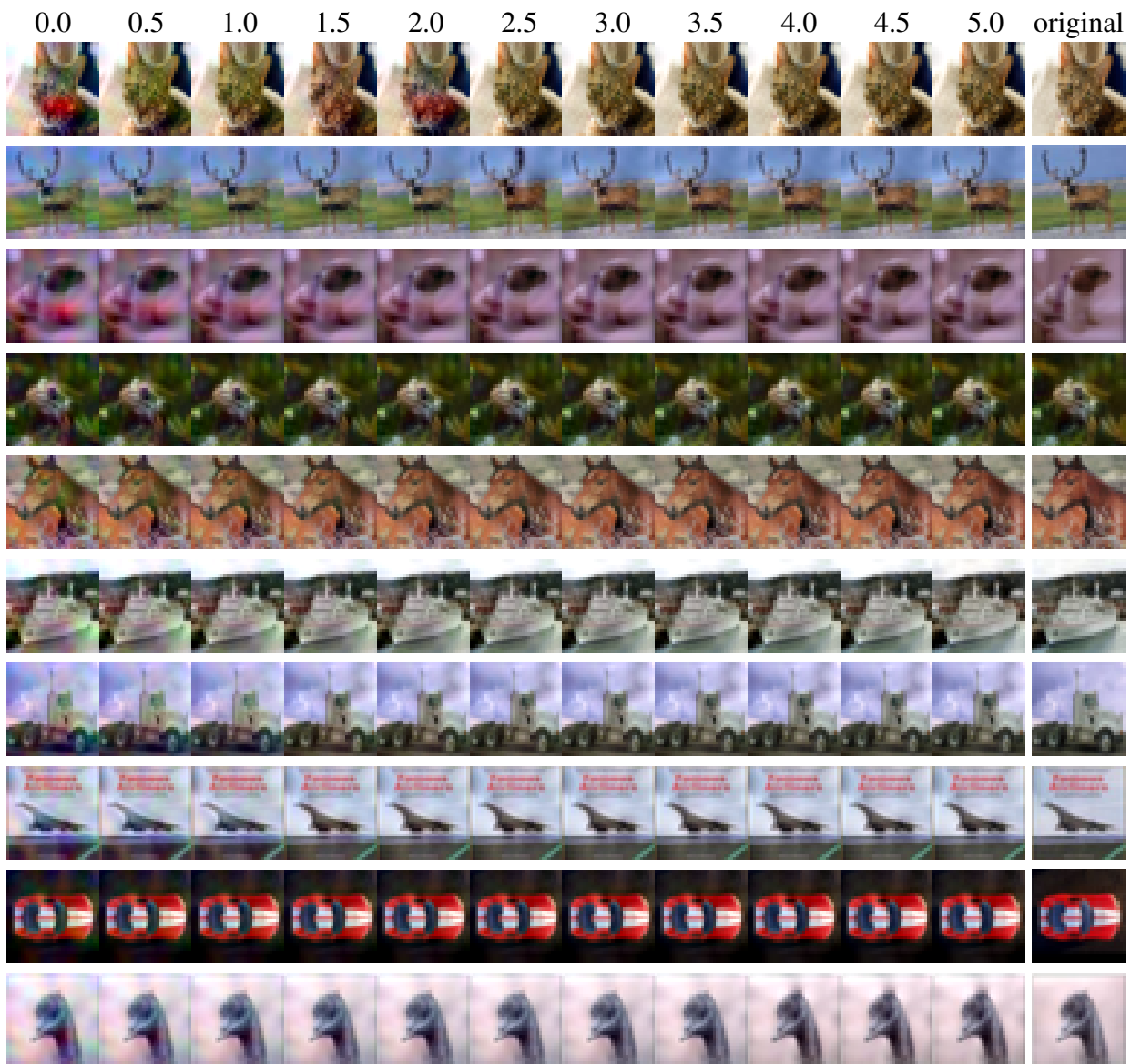


Figure 11.2: The visual effect of λ_s (the coefficient of $\text{Dissim}(\delta)$) on CIFAR-10. See section 4.3 for the details of the experiments.



Figure 11.3: The visual effect of λ_{tv} on CIFAR-10. See section 4.3 for the details of the experiments.

ImageNet Results

In this section, we show some examples generated by the Shadow Attack that spoof certificates for a Randomized Smoothed classifier on Imagenet.



Figure 11.4: Natural looking ImageNet adversarial images that produce large certified radii for the ImageNet Gaussian smoothed classifier.

Appendix 12: Appendix For Adaptive Weight Decay

12.1 Experimental Setup

In this section, we include the experimental setup used to produce the experiments throughout this paper. We include all hyperparameters used for all experiments, unless explicitly mentioned otherwise. For all experiments, we use SGD optimizers with momentum $\mu = 0.9$. We use a Cosine learning rate schedule with no warm-up and with the value of 0 for the final value. The weight decay and learning rate for experiments that have not been clearly specified are $\lambda_{wd} = 0.0005$ and $lr = 0.1$. We train all networks for 200 epochs with a batch-size of 128. For all experiments that use ASAM, we use the hyper-parameters the original paper suggests.

12.1.1 Adversarial Training

We use a pre-act WideResNet28 with a width of 10. We use $\ell_\infty = 8/255$ PGD attack with the step size of $2/255$. We use 7 steps for creation of adversarial examples for training and use the minimum accuracy produced by AutoAttack for the test. We keep 10% of training data for validation and use it to do early stopping. For λ_{wd} of non-adaptive weight decay, we fit a geometric sequence of length 17 starting from 0.00005 and ending at 0.005. For the adaptive weight decay hyper parameter (*DoG*), we fit a geometric sequence of length 9 starting from 0.02 and ending at 0.04.

12.1.2 Noisy Label Training

We use a ResNet34 as our architecture. For each setting we used 5 different hyperparameters for adaptive and non-adaptive weight decays. For CIFAR-10, we use $\lambda_{wd} \in \{0.0028, 0.005, 0.0089, 0.0158, 0.028\}$ and $DoG \in \{0.036, 0.028, 0.018, 0.0153, 0.018\}$ and for CIFAR-100, we use $\lambda_{wd} \in \{0.0281, 9.9158, 0.0089, 0.0158, 0.028\}$

and $DoG \in \{0.013, 0.0153, 0.018, 0.028, 0.036\}$.

12.2 Implementation

In this section, we discuss the details of implementation of adaptive weight decay. The method is not really susceptible to the exact implementation details discussed here, however, to be perfectly candid, we include all details here. First, let us assume that we desire to implement adaptive weight decay using $DoG = 0.016$ as the hyperparameter. We know that $\lambda_t = \frac{\|\nabla w_t\|^{0.016}}{\|\nabla w_t\|}$. Please note that $\|\nabla w_t\|$ requires knowing the gradients of the loss w.r.t. the network's parameters. Meaning that to compute the λ_t for every step, we have to call a backward pass on the actual parameters of the network. After this step, given the fact that we know both $\|\nabla w_t\|$ and $\|\nabla w_t\|$, we can compute λ_t . Please note that directly applying $\frac{\lambda_t}{2} \|w_t\|_2^2$ will invoke computation of 2nd order derivatives, since λ_t is computed using the 1st order derivatives. Consequently, to save up computation and avoid 2nd order derivatives, we convert the λ_t into a non-tensor scalar before plugging its value into the weight decay regularization term.

Using our approach, the value of λ_t varies in each iteration. However, one could make λ_t more stable, by using an exponential weighted average $\bar{\lambda}_t = 0.1 \times \lambda_{t-1} + 0.9 \times \lambda_t$.

Algorithm 1 provides a psudo-code of our implementation.

12.3 Tuning WD and LR

As discussed previously, tuning the hyper-parameters such as λ_{wd} is crucial. First, through examples, we demonstrate why the correct way of finding the right hyper-parameter for λ_{wd} and lr (learning rate) is a 2D grid search over different values of these hyper-parameters. For the

Algorithm 1: Adaptive Weight Decay

Input: $DoB > 0$
 $\bar{\lambda} \leftarrow 0$
for $(x, y) \in loader$ **do**
 $p \leftarrow model(x)$ ▷ Get models prediction.
 $main \leftarrow CrossEntropy(p, y)$ ▷ Compute CrossEntropy.
 $\nabla w \leftarrow backward(main)$ ▷ Compute the gradients of main loss w.r.t weights.
 $\lambda \leftarrow \frac{\|\nabla w\|_{DoG}}{\|w\|}$ ▷ Compute iteration's weight decay hyperparameter.
 $\bar{\lambda} \leftarrow 0.1 \times \bar{\lambda} + 0.9 \times stop_gradient(\lambda)$ ▷ Compute the weighted average as a scalar.
 $w \leftarrow w - lr(\nabla w + \bar{\lambda} \times w)$ ▷ Update Network's parameters.

sake of saving in computation, one might assume that fixing lr and 1D grid search over λ_{wd} , then fixing λ_{wd} and grid search over lr might be a (computationally) cheaper and as effective.

Consider the following example (Figure 12.1) for training image classification networks on CIFAR-10 using Wide Residual Nets (WRN28-10) ¹. We have initial guesses $\lambda_{wd} = 0.005$ and $lr = 0.01$ ². We perform a 1D grid search for lr first while fixing $\lambda_{wd} = 0.005$ (Figure 12.1(a)). Then after realizing that $lr = 0.01$ is the optimal hyper-parameter value, we perform a 1D grid search on λ_{wd} (Figure 12.1(b)). No matter how many times we alternate between searching for λ_{wd} and lr , we always will end up with the same fixed values of $\lambda_{wd} = 0.005$ and $lr = 0.01$. One might assume that these values are optimal, while $\lambda_{wd} = 0.0005$ and $lr = 0.1$ yields better results (Figure 12.1(c)).

12.4 Tuning Adaptive Weight Decay

Since, as explained before, adaptive weight decay is more robust to change in lr , it would be easier first to tune DoG and then tune lr . Also, tuning DoG and lr for adaptive weight decay should be easier than tuning λ_{wd} and lr for non-adaptive weight decay.

¹details of experiments can be found in Appendix 12.1

²a common choice for hyper-parameter values for training CIFAR resnets to build baselines

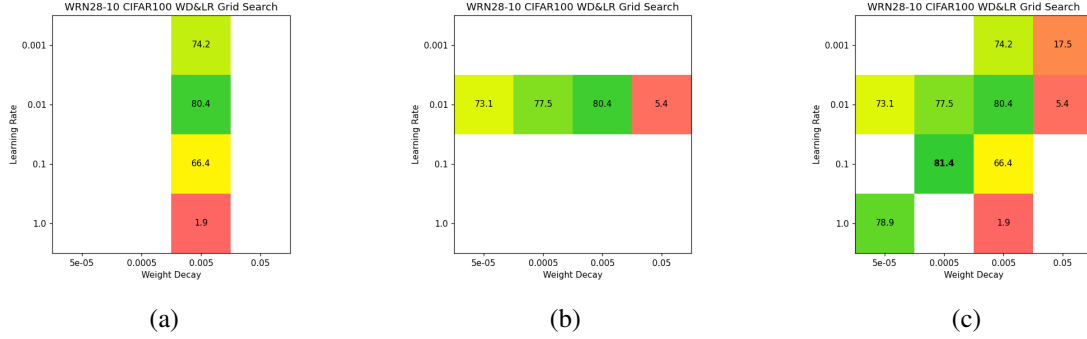


Figure 12.1: Held-out validation accuracy for different searches on tunable hyper-parameter values. Separate grid search on learning rate (a) and weight decay’s hyper-parameter (b) does not reveal the optimal set of hyper-parameters, at the best it reveals the diagon on which the best hyper-parameters resides. An extra search on the diagonal is necessary to find the said hyper-parameter (c).

While the preferred method for finding the best hyper-parameter for adaptive weight decay (i.e., *DoG*) is to treat it as any other hyper-parameter and perform a random grid-search, we present a method to find a good estimate which works well in practice. We suggest that if a training pipeline exists that is fully tuned for non-adaptive weight decay, we can get a reasonable estimate for *DoG* by following steps mentioned in Fig. 5.2.

12.5 Extra Results

Here, we provide the extra results and figures not included in the body of the paper.

12.5.1 CIFAR-10 robustness to noisy labels

The results of the experiments for training classifiers on the CIFAR-10 dataset with noisy labels can be seen in Fig. 12.2 which yields similar conclusions to that of CIFAR-100.

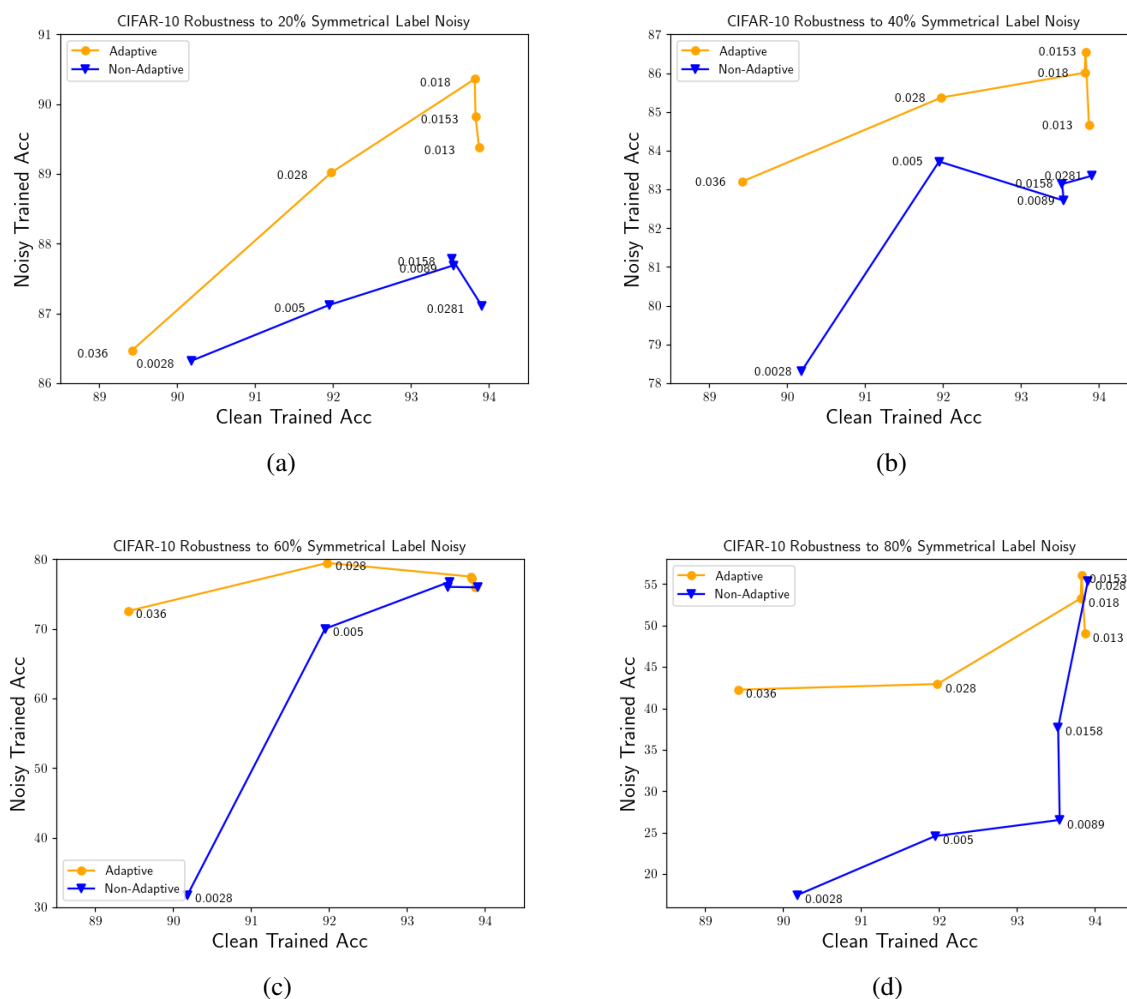


Figure 12.2: Comparison of similarly performing networks once trained on CIFAR-100 clean data, after training on 20% (a), 40% (b), 60% (c), and 80% (d) noisy data. Networks trained with adaptive weight decay outperform non-adaptive trained networks.

12.5.2 2D Grid search for best parameter values for ResNet32

The importance of the 2D grid search on learning-rate and weight decay hyper-parameters are not network dependent. And we can see how these values are tied together for ResNet32 in Figure 12.3.

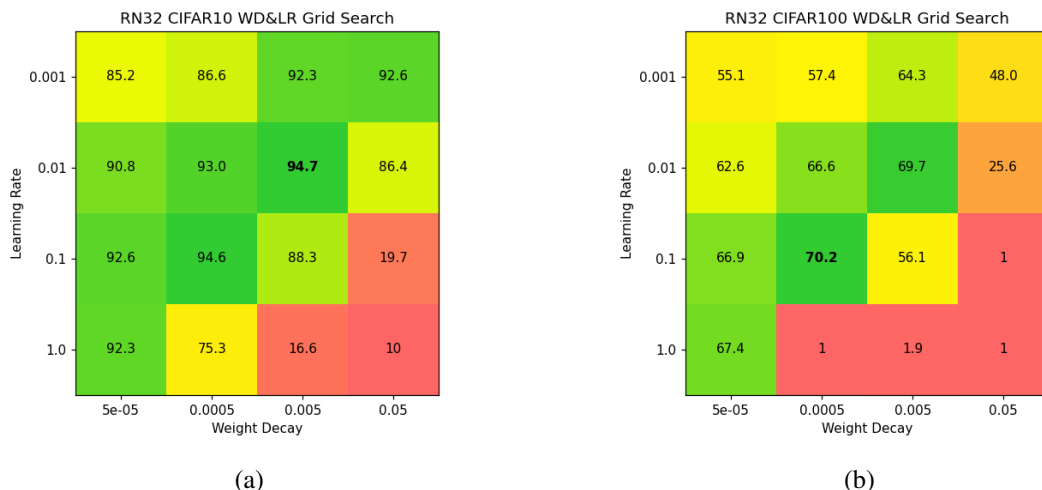


Figure 12.3: Grid Search on different values of learning rate and weight decay on accuracy of ResNet32 on (a) CIFAR10 and (b) CIFAR100.

12.5.3 Visualizing images from the CIFAR-100 training set where best AWD models do not fit.

In Fig. 12.4, we visualize some of the 4.71% examples which belong to the CIFAR-100 training set that our AWD trained network does not fit. Interestingly enough, there are many examples like 0-3, 10-14, 48, and 49 with overlapping classes with one object. There are many examples with wrong labels, such as 4-6, 43-45, 28-29, and 51-57. In many more examples, there are at least two objects in one image, such as 15-25, 28-33, 38-42, and 58-59. Unsurprisingly, the model would be better off not fitting such data, as it would only confuse the model to fit the data, which contradicts its already existing and correct conception of the object.

12.5.4 Robustness to Sub-Optimal Learning Rate

In this section, we have a deeper look the performance of networks trained with sub-optimal learning rates. We observe that adaptive weight decay is more robust to changes in the

learning rate in comparison to non-adaptive weight decay for both CIFAR-10 and CIFAR-100, as shown in Figure 12.5(a) and 12.5(b). The accuracy for $lr = 1.0$ for non-adaptive weight decay drops 69.67% on CIFAR-100 and 5.0% on CIFAR-10, compared to its adaptive weight decay counterparts.

The robustness to sub-optimal learning rates suggests that adaptive weight decay might be more suitable for applications where tuning for the optimal learning rate might be expensive or impractical. An example would be large language models such as GPT-3 or Megatron-Turing NLG, where even training the network once is expensive [16, 159, 182]. Another example would be neural architecture search, where one trains many networks [14, 132, 161, 194, 241].

12.5.5 Robustness to Parameter Pruning

As seen in previous sections, models trained with adaptive weight decay tend to have smaller weight norms. Smaller weight norms, can indicate that models trained with adaptive weight decay are less sensitive to parameter pruning. We adopt a simple non-iterative ℓ_1 pruning to test our intuition. Table 12.1 which shows the accuracy of models trained with adaptive and non adaptive weight decay after various percentages of the parameters have been pruned verifies our hypothesis. Adaptive weight decay trained models which are trained with various learning rates are more robust to pruning in comparison to models trained with non-adaptive weight decay.

12.6 Related Work on Adaptive Weight Decay

The concept of Adaptive Weight Decay was first introduced by [141]. Similar to our method, their method (AdaDecay) changes the weight decay’s hyper-parameter at every iteration.

Dataset	LR	Method	Nat	40%	50%	60%	70%	80%	90%
C10	1	$DoG = 0.018$	93.1 ± 1.3	93.1 ± 1.3	93.1 ± 1.3	93.1 ± 1.3	93.1 ± 1.3	93.1 ± 1.3	93.2 ± 1.3
		$\lambda_{wd} = 0.0005$	88.1 ± 3.0	88.0 ± 3.1	88.0 ± 3.1	88.0 ± 3.1	88.0 ± 3.1	88.0 ± 3.1	88.0 ± 3.1
	0.1	$DoG = 0.018$	96.0 ± 0.1	91.6 ± 8.7	91.7 ± 8.6	91.7 ± 8.6	91.5 ± 9.0	90.5 ± 10.5	83.7 ± 17.9
		$\lambda_{wd} = 0.0005$	96.3 ± 0.1	93.6 ± 5.3	93.6 ± 5.3	93.6 ± 5.3	93.4 ± 5.5	92.6 ± 6.2	84.3 ± 13.6
	0.01	$DoG = 0.018$	95.9 ± 0.1	95.9 ± 0.1	95.9 ± 0.1	95.9 ± 0.1	95.9 ± 0.1	95.8 ± 0.1	91.4 ± 1.4
		$\lambda_{wd} = 0.0005$	95.0 ± 0.1	95.0 ± 0.1	95.0 ± 0.1	94.5 ± 0.3	75.2 ± 12.9	19.3 ± 12.6	12.2 ± 5.0
	0.001	$DoG = 0.018$	87.9 ± 0.2	82.1 ± 1.9	75.2 ± 4.2	64.6 ± 7.2	38.3 ± 8.4	18.8 ± 4.3	13.4 ± 1.9
		$\lambda_{wd} = 0.0005$	87.9 ± 0.4	80.5 ± 2.9	73.8 ± 4.8	56.4 ± 8.1	46.8 ± 16.9	21.0 ± 9.9	12.2 ± 2.5
C100	1	$DoG = 0.016$	77.6 ± 1.3	77.6 ± 1.3	77.6 ± 1.3	77.6 ± 1.3	77.6 ± 1.3	77.5 ± 1.2	75.3 ± 1.3
		$\lambda_{wd} = 0.0005$	7.9 ± 1.4	2.6 ± 1.8	2.6 ± 1.8	2.6 ± 1.8	2.6 ± 1.8	2.6 ± 1.8	2.6 ± 1.8
	0.1	$DoG = 0.016$	80.7 ± 0.3	80.7 ± 0.3	80.7 ± 0.3	80.5 ± 0.2	79.6 ± 0.4	74.5 ± 0.9	29.2 ± 3.6
		$\lambda_{wd} = 0.0005$	80.5 ± 0.2	80.4 ± 0.1	80.2 ± 0.2	79.9 ± 0.2	78.4 ± 0.2	71.3 ± 1.2	23.9 ± 6.7
	0.01	$DoG = 0.016$	79.5 ± 0.4	79.5 ± 0.3	79.4 ± 0.3	79.3 ± 0.4	78.7 ± 0.5	75.0 ± 0.8	28.9 ± 4.4
		$\lambda_{wd} = 0.0005$	76.9 ± 0.1	74.4 ± 3.6	74.1 ± 3.8	73.5 ± 4.2	67.2 ± 4.1	17.5 ± 6.9	1.5 ± 0.4
	0.001	$DoG = 0.016$	63.8 ± 0.3	55.3 ± 1.8	47.4 ± 2.2	32.0 ± 2.1	12.9 ± 2.9	2.8 ± 1.4	1.1 ± 0.2
		$\lambda_{wd} = 0.0005$	62.1 ± 0.2	52.7 ± 2.0	41.7 ± 2.7	25.6 ± 2.4	11.0 ± 4.4	3.1 ± 0.9	1.2 ± 0.1

Table 12.1: Models trained with Adaptive Weight Decay are more robust to change in learning rate and pruning.

Unlike our method, in one iteration, AdaDecay imposes a different penalty to each individual parameter, while our method penalizes all parameters with the same magnitude. More specifically, the updates from weight decay in our method is $-\lambda_t w$, where λ_t varies at every iteration. However, the updates in AdaDecay for parameter w_i is $-\lambda \theta_{t,i} w_i$, where λ is constant at every iteration, instead, $0 \leq \theta_{t,i} \leq 2$ can vary for different parameters i and for different iterations t . In other words, AdaDecay introduces $\theta_{t,i}$ for every single parameter of the network to represent how strongly each parameter should be penalized in the weight decay. For instance, if $\theta_{t,i} = 1$, then AdaDecay has the same effect as non-adaptive weight decay. For parameters in layer L , $\mathbb{E}_{w_i \in L} \theta_{t,i} = 1$, due to the fact that AdaDecay computes $\theta_{t,i}$ based on layerwise-normalized gradients. More percisely:

$$\theta_{t,i} = \frac{2}{1 + \exp(-\alpha \nabla \bar{w}_{i,t})}$$

where $\nabla \bar{w}_{i,t}$ is the layerwise-normalize gradients. More percisely, $\nabla \bar{w}_{i,t} = \frac{\nabla w_{i,t} - \mu_L}{\sigma_L}$ where μ_L and σ_L represent mean and standard deviation of gradients at layer L .

The main difference between our version and AdaDecay apart from our different formulations is that our method can increase the weight norm penalty (i.e. λ_t) indefinitely, while AdaDecay on average penalizes the parameters of the same layer similar to non-adaptive weight decay. This is due to the fact that $\mathbb{E}_{w_i \in L} \theta_{t,i} = 1$. For instance, for the most extreme case, assuming that $\theta_{t,i} = 2$ for all i and t , the effect of AdaDecay becomes at most twice as strong as non-adaptive weight decay, while our version does not have such upperbounds.

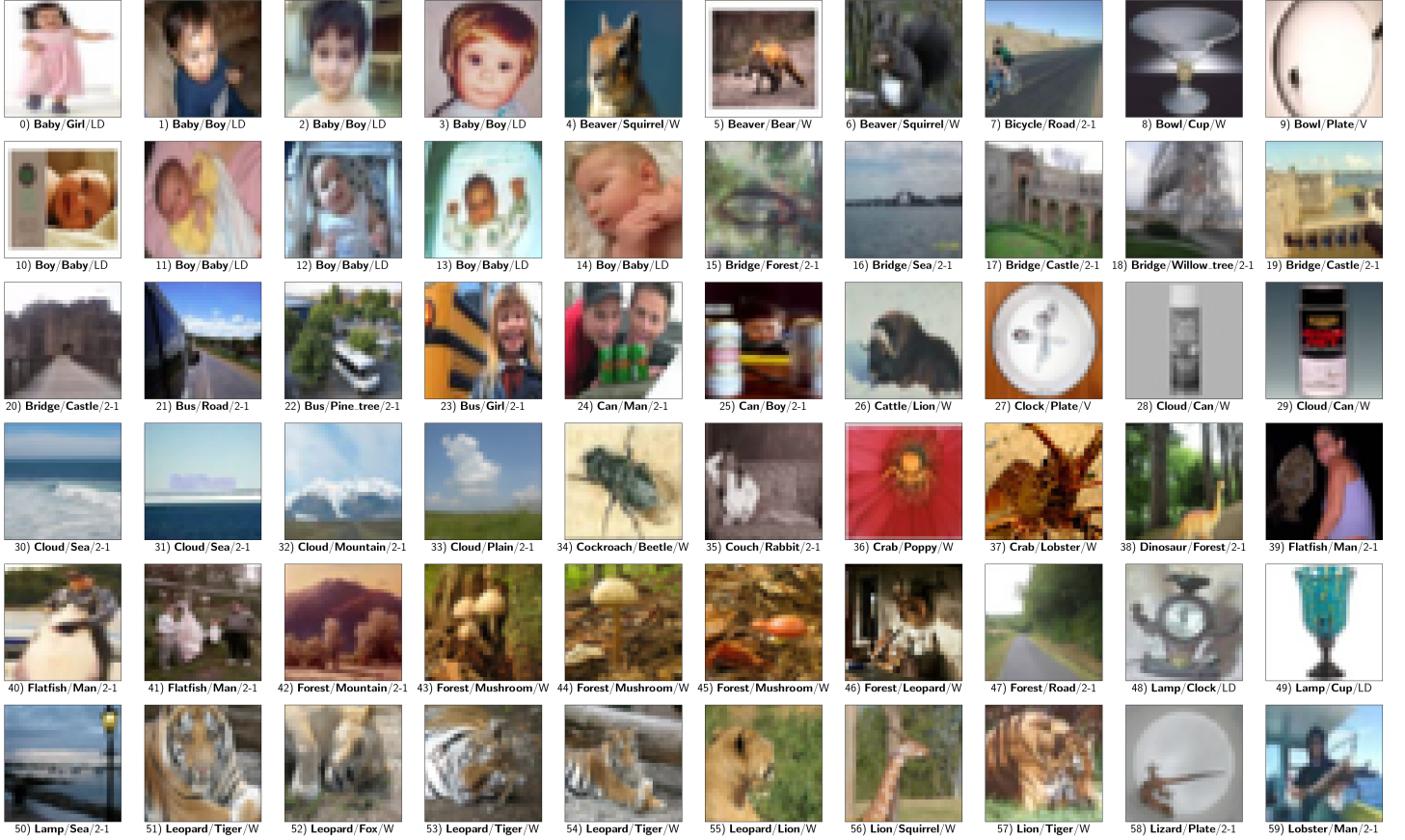


Figure 12.4: Examples from CIFAR-100 training dataset that have noisy labels. For every image we state [dataset label/ prediction of classifier trained with AWD / our category of noisy case]. We classify these noisy labels into several categories: **W**: Wrong Labels where the picture is clear enough to comprehend the correct label.; **2-1**: Two Objects from CIFAR-100 in one image, but only one label is given in the dataset; **LD**: Loosely Defined Classes where there is one object, but one object could be two classes at the same time. For instance, a baby girl is both a baby and a girl. **V**: Vague images where authors had a hard time identifying.

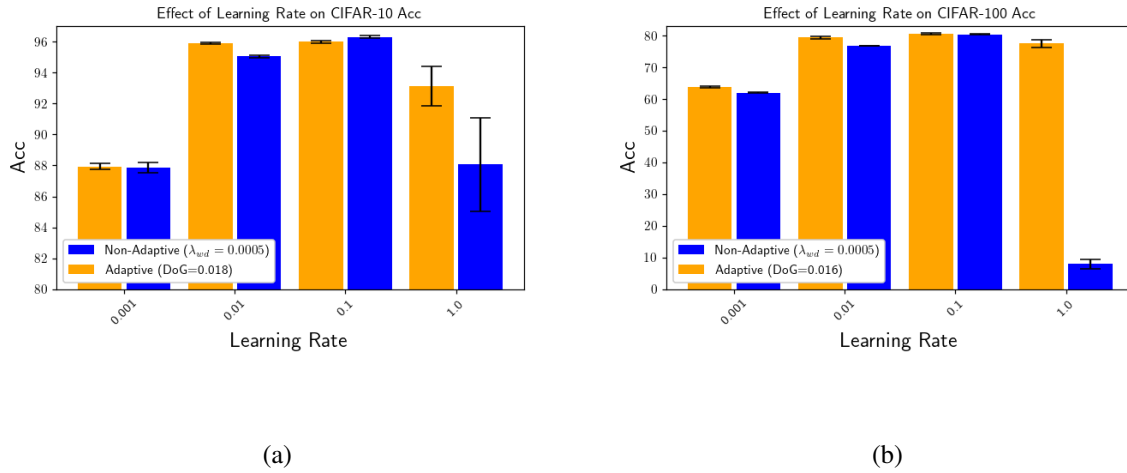


Figure 12.5: Models trained with Adaptive Weight Decay are more robust to Learning rate. Results are an average over 5 trials. We use the same experiment setup and estimated DoG values from section 5.3.

Method	Learning Rate			
	0.001	0.01	0.1	1.0
Adaptive $DoG = 0.016$	92.53	16.22	18.3	26.65
Non-Adaptive $\lambda_{wd} = 0.0005$	116.98	27.01	24.06	12.16

Figure 12.6: Norm of the weights for networks trained with adaptive weight decay and non-adaptive weight decay.

Appendix 13: Appendix For Plug-In Inversion

13.1 Additional Results

13.1.1 Ablation Study for α and β

Figure 13.1 shows the results of PII when varying the values of α and β , which determine the intervals from which the ColorShift constants are randomly drawn. Based on this and similar experiments, we permanently fix these parameters to $\alpha = \beta = 1.0$ for all other PII experiments, and find that these values indeed transfer well to other models.

13.1.2 Insensitivity to TV regularization

Figure 13.2 shows additional results on the effect of ColorShift on the sensitivity to the weight of TV regularization when inverting a robust model, complementing Figure 6.4. As in the earlier figure, we observe that certain values of λ_{TV} may produce noisy or blurred images when not using ColorShift, whereas the ColorShift results are quite stable.

13.1.3 Effect of Centering

Here are more examples on the effect of centering. Figures 13.3, and 13.4 show the effect of centering on inverting a robust, and natural model, respectively.

13.1.4 Effect of Ensembling Size

Figure 13.5 gives additional results to those in figure 6.5 for the effect of ensembling size on inversion.

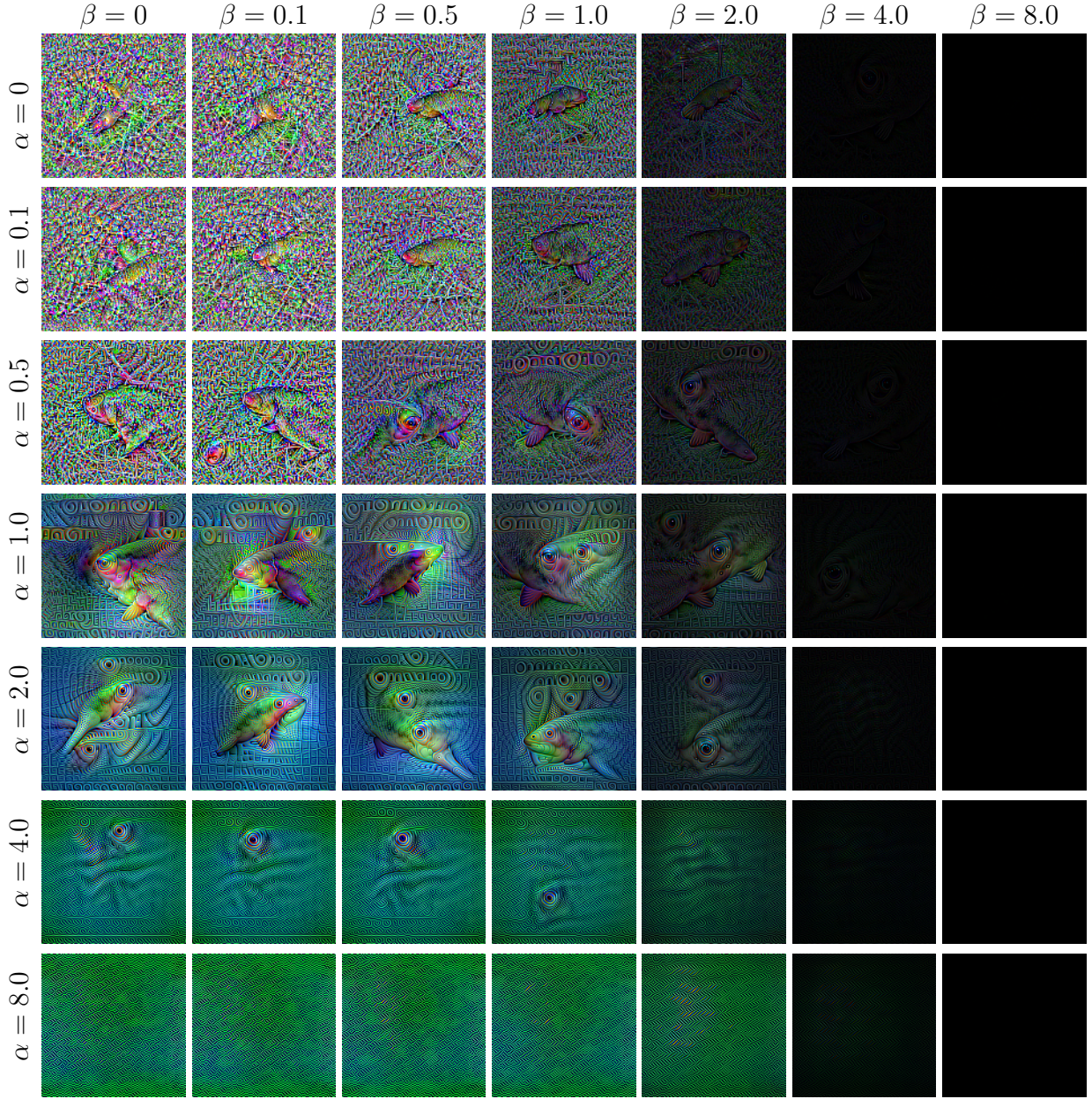
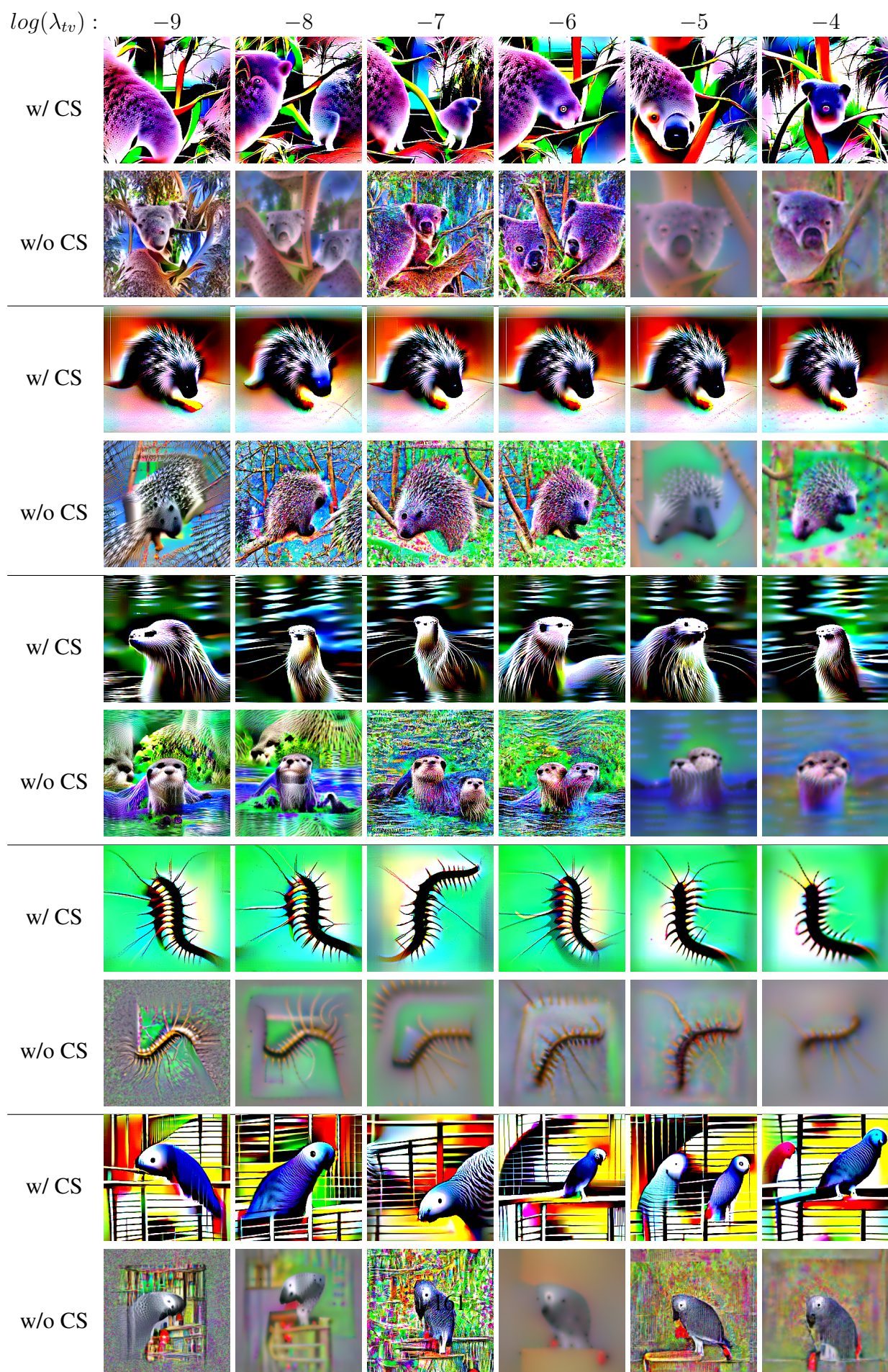


Figure 13.1: Effect of α and β on the quality of the images generated by PII from a naturally-trained ResNet-50 for the Tench class.



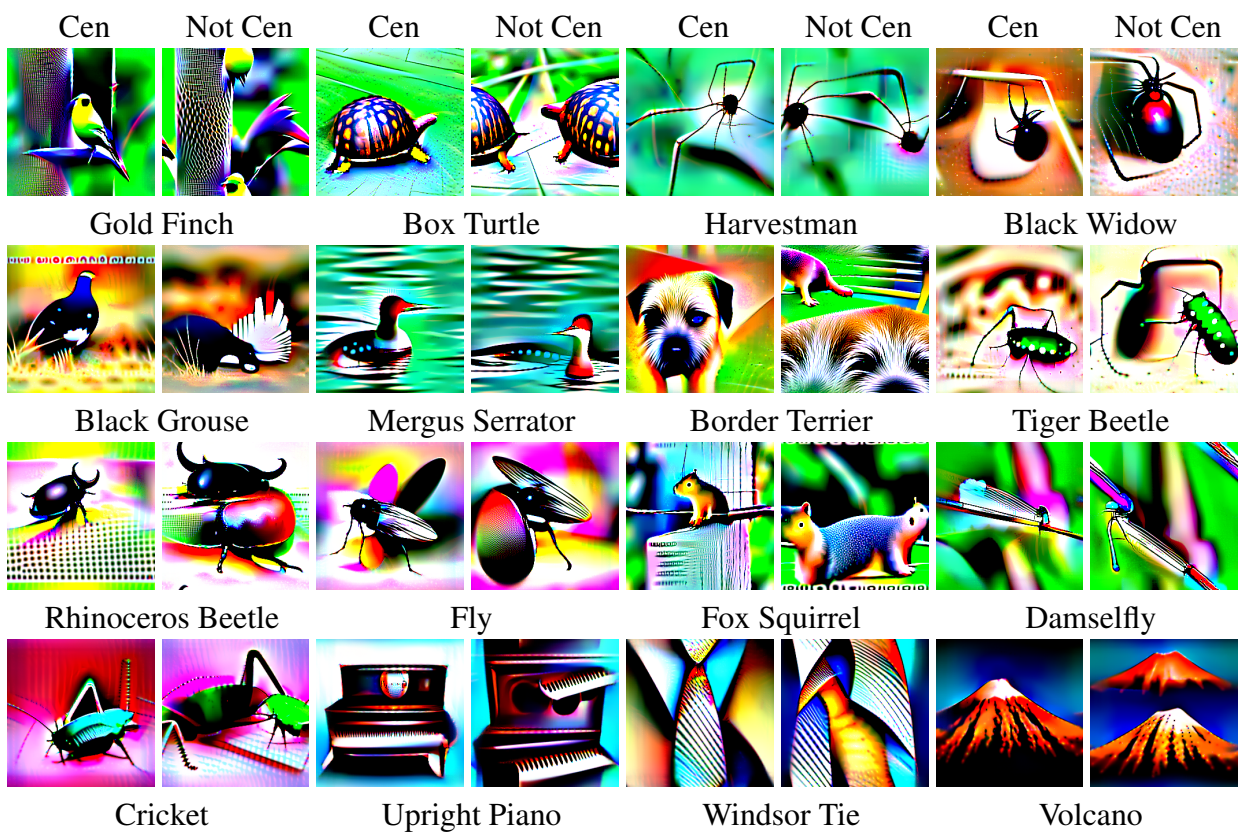


Figure 13.3: Effect of using centering vs not using centering for a robust ResNet-50.

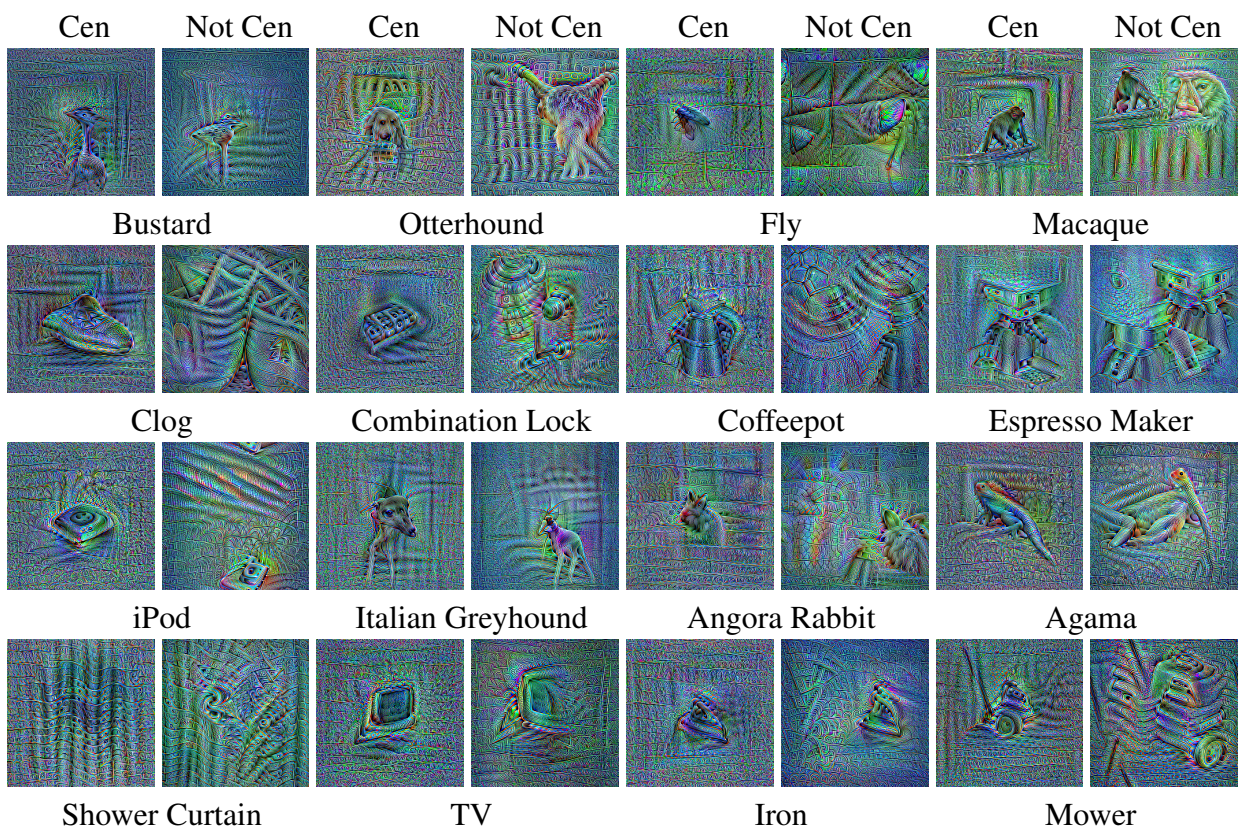


Figure 13.4: Effect of using centering vs not using centering for a naturally-trained ResNet-50.

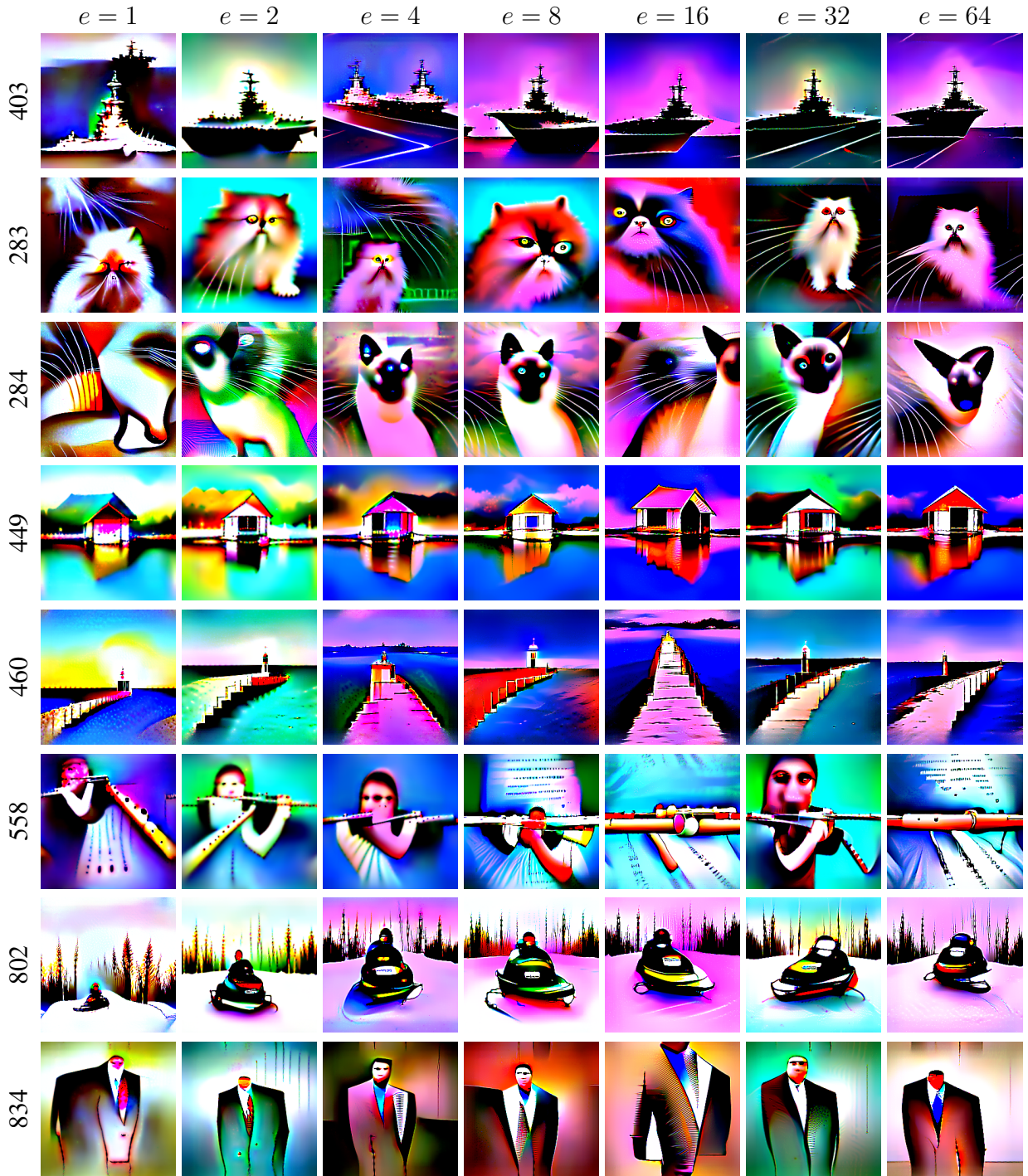


Figure 13.5: Effect of ensemble size when inverting a robust ResNet-50. Even small values of e give reasonably good results, but increasing e tends to give slight improvement.

13.1.5 Effect of using other Augmentations

We used 4 random augmentations other than ColorShift to make comparisons. We used augmentations used in [26] with modifications. We use PyTorch [150] notion to describe this part. We used RandomHorizontalFlip with 0.5 probability. We used RandomResizedCrop with scale [0.7, 1.], and ratio [0.75, and 1.33]. With applied ColorJitter with 0.8 probability, and brightness, contrast, saturation, and hue of (0.4, 0.4, 0.4, 0.1), respectively. We used RandomGrayscale with 0.2 probability. For this experiment, we do apply data normalization before feeding the input to the network. This is different than the regular experiment setting that we use for the robust model that we use as explained in 13.4 The reason is that not having data normalization is similar to using ColorShift (it changes the data distribution which the model expects as an input).

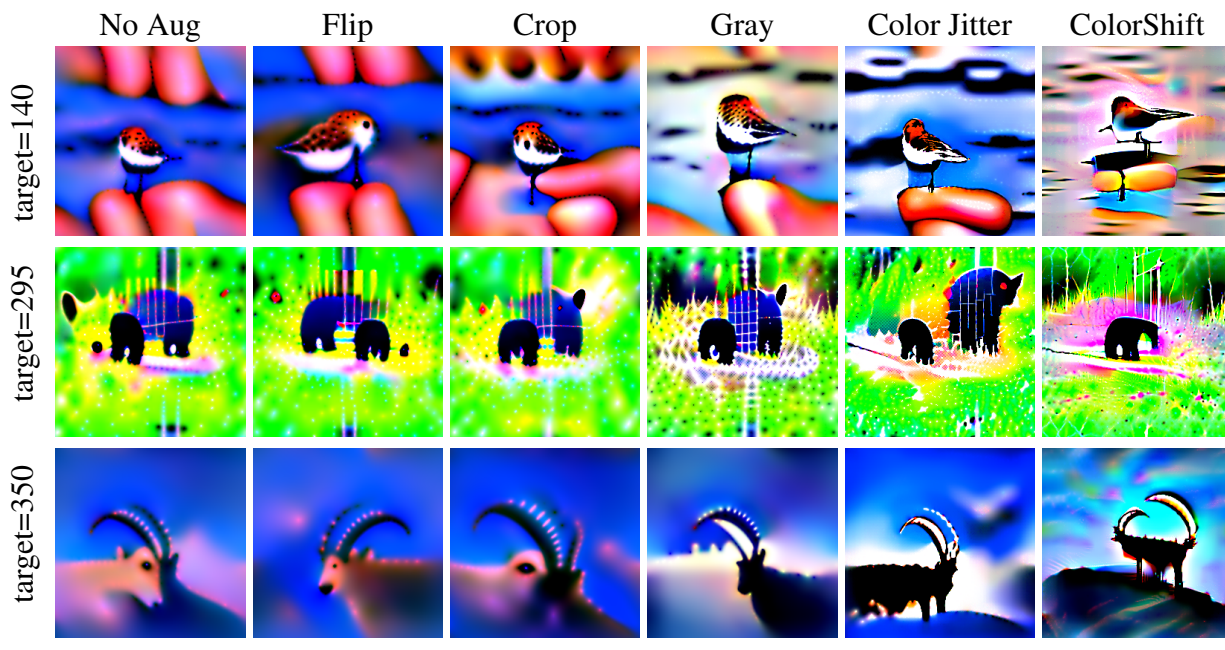


Figure 13.6: Effect of various alternative augmentations on inverting a robust ResNet-50 model.

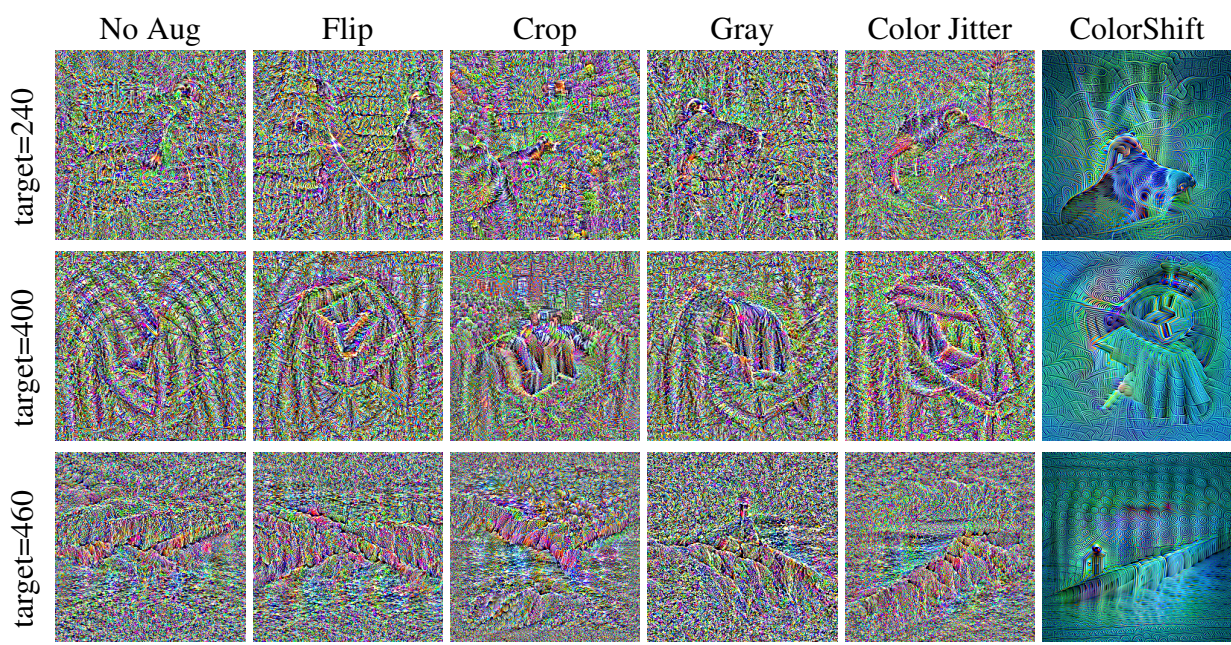


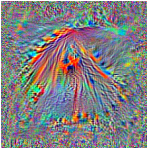
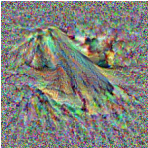
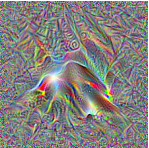
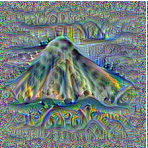
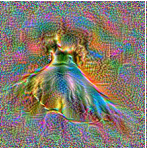
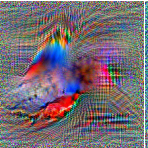
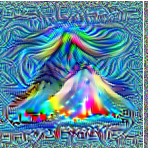
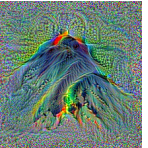
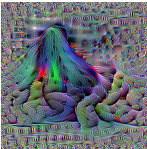
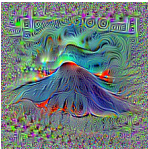
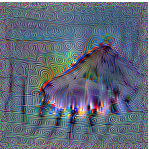
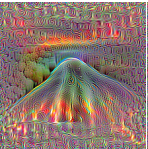
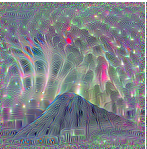
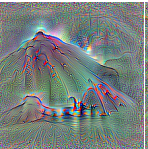
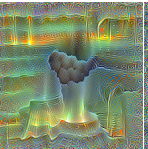
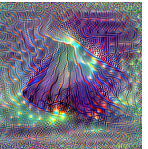
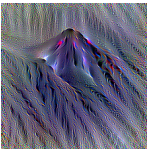
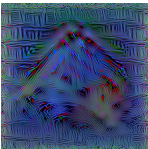
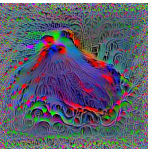
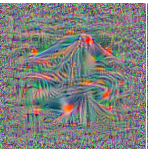
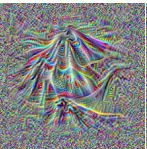
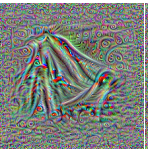
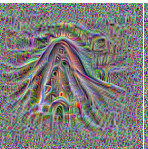
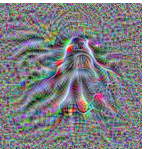
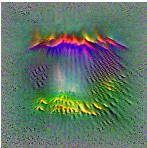
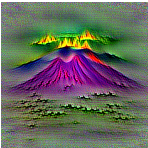
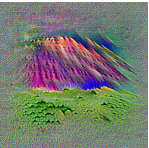
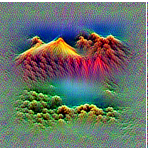
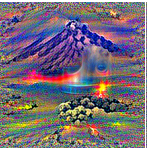
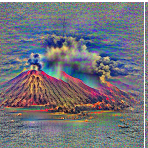
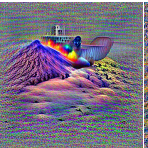
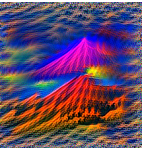
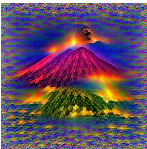
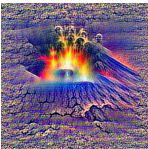
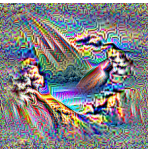
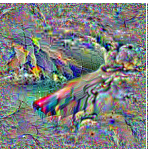
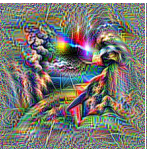
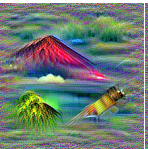
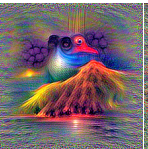
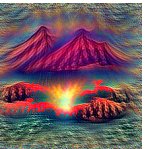
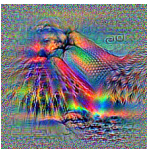
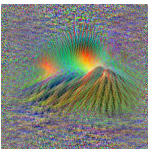
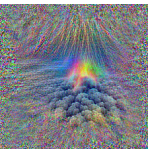
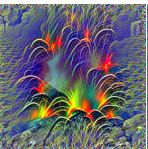
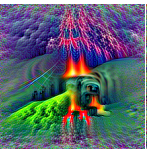
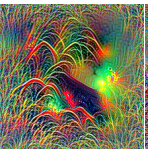
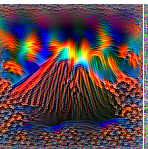
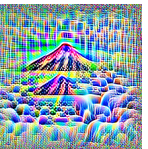
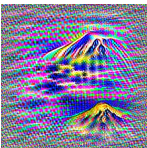
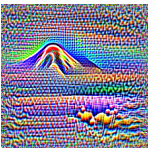
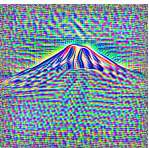
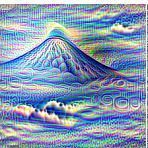
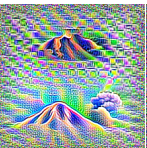
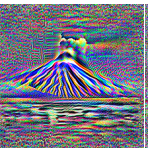
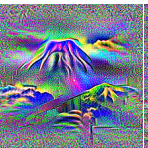
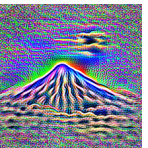
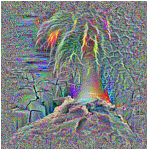
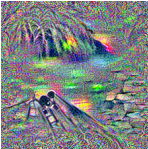
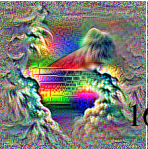
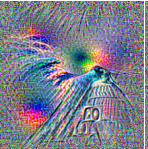
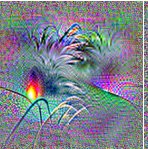
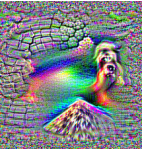
Figure 13.7: Effect of using different augmentations on inverting a naturally-trained ResNet-50.

13.1.6 PII on additional networks

Figure 13.8 shows the results of Plug-In Inversion on various CNN, ViT, and MLP networks, adding to those shown in figure 6.7. See section 14.3 for model details.

13.2 Models

In our experiments, we use publicly available pre-trained models from various sources. The following tables list the models used from each source, along with references to where they are introduced in the literature.

							
AlexNet	DenseNet	GoogLeNet	MobileNet v2	MobileNet v3-l	MobileNet v3-s	MNasNet 0-5	MNasNet 1-0
							
ResNet 18	ResNet 34	ResNet 50	ResNet 101	ResNet 152	ResNext 50	ResNext 101	WRResNet 50
							
WRResNet 101	ShuffleNet v2-0-5	ShuffleNet v2-1-0	SqueezeNet	VGG11-bn	VGG13-bn	VGG16-bn	VGG19-bn
							
ViT B16	ViT B32	ViT L16	ViT L32	DeiT p16-224	DeiT-D p16-384	DeiT p16-384	DeiT-D-t p16-224
							
DeiT-D-s p16-224	DeiT-D p16-224	CoaT-m	CoaT-s	CoaT-t	ConViT	ConViT-s	ConViT-t
							
Mixer 24-224	Mixer b16-224	Mixer 116-224	PiT-D b-224	PiT s-224	PiT-D s-224	PiT-D t-224	ResMLP 12-224
							
ResMLP- D 12-224	ResMLP 24-224	ResMLP- D 24-224	ResMLP 36-224	ResMLP- D 36-224	ResMLP b-24-224	ResMLP b-24-224- 1k	ResMLP- D b-24-224
							
Swin w12-384	Swin s-w7-224	Twin pcptv-b	Twin pcptv-l	Twin pcptv-s	Twin svt-b	Twin svt-l	Twin svt-s

Alias	Name	Paper
AlexNet	alexnet	[101]
DenseNet	densenet121	[84]
GoogLeNet	googlenet	[192]
MobileNet v2	mobilenet_v2	[169]
MobileNet-v2	mobilenet_v2	[169]
MobileNet v3-l	mobilenet_v3_large	[83]
MobileNet v3-s	mobilenet_v3_small	[83]
MNASNet 0-5	mnasnet0_5	[195]
MNASNet 1-0	mnasnet1_0	[195]
ResNet 18	resnet18	[74]
ResNet-18	resnet18	[74]
ResNet 34	resnet34	[74]
ResNet 50	resnet50	[74]
ResNet 101	resnet101	[74]
ResNet-101	resnet101	[74]
ResNet 152	resnet152	[74]
ResNext 50	resnext50_32x4d	[221]
ResNext 101	resnext101_32x8d	[221]
WRResNet 50	wide_resnet50_2	[231]
WRResNet 101	wide_resnet101_2	[231]
W-ResNet-101-2	wide_resnet101_2	[231]
ShuffleNet v2-0-5	shufflenet_v2_x0_5	[127]
ShuffleNet v2-1-0	shufflenet_v2_x1_0	[127]
ShuffleNet v2	shufflenet_v2_x1_0	[127]
SqueezeNet	squeezenet1_0	[86]
VGG11-bn	vgg11_bn	[178]
VGG13-bn	vgg13_bn	[178]
VGG16-bn	vgg16_bn	[178]
VGG19-bn	vgg19_bn	[178]

Figure 13.9: Pre-trained models from TorchVision: <https://github.com/pytorch/vision>.

Alias	Name	Paper
ViT B16	B_16_imagenet1k	[46]
ViT B32	B_32_imagenet1k	[46]
ViT B-32	B_32_imagenet1k	[46]
ViT L16	L_16_imagenet1k	[46]
ViT L32	L_32_imagenet1k	[46]

Figure 13.10: Pre-trained models used from : <https://github.com/lukemelas/PyTorch-Pretrained-ViT>.

Alias	Name	Paper
DeiT p16-224	deit_base_patch16_224	[200]
DeiT P16 224	deit_base_patch16_224	[200]
DeiT-D p16-384	deit_base_distilled_patch16_384	[200]
DeiT Dist P16 384	deit_base_distilled_patch16_384	[200]
deit p16-384	deit_base_patch16_384	[200]
DeiT-D-t p16-224	deit_tiny_distilled_patch16_224	[200]
DeiT-D-s p16-224	deit_small_distilled_patch16_224	[200]
DeiT-D p16-224	deit_base_distilled_patch16_224	[200]

Figure 13.11: Pre-trained models from [\[199\]](#) .

Alias	Name	Paper
CoaT-m	coat_lite_mini	[223]
CoaT-s	coat_lite_small	[223]
CoaT-t	coat_lite_tiny	[223]
ConViT	convit_base	[37]
ConViT-s	convit_small	[37]
ConViT-t	convit_tiny	[37]
ConViT tiny	convit_tiny	[37]
Mixer 24-224	mixer_24_224	[197]
Mixer b16-224	mixer_b16_224	[197]
Mixer b16 224	mixer_b16_224	[197]
Mixer b16-224-mil	mixer_b16_224_miil	[197]
Mixer l16-224	mixer_l16_224	[197]
PiT-D b-224	pit_b_distilled_224	[80]
PiT Dist 224	pit_b_distilled_224	[80]
PiT s-224	pit_s_224	[80]
PiT-D s-224	pit_s_distilled_224	[80]
PiT-D t-224	pit_ti_distilled_224	[80]
ResMLP 12-224	resmlp_12_224	[198]
ResMLP-D 12-224	resmlp_12_distilled_224	[198]
ResMLP 24-224	resmlp_24_224	[198]
ResMLP-D 24-224	resmlp_24_distilled_224	[198]
ResMLP 36-224	resmlp_36_224	[198]
ResMLP-D 36-224	resmlp_36_distilled_224	[198]
ResMLP 36 Dist	resmlp_36_distilled_224	[198]
ResMLP b-24-224	resmlp_big_24_224	[198]
ResMLP b-24-224-1k	resmlp_big_24_224_in22ft1k	[198]
ResMLP-D b-24-224	resmlp_big_24_distilled_224	[198]
Swin w7-224	swin_base_patch4_window7_224	[125]
Swin l-w7-224	swin_large_patch4_window7_224	[125]
Swin l-w12-384	swin_large_patch4_window12_384	[125]
Swin w12-384	swin_base_patch4_window12_384	[125]
Swin P4 W12	swin_base_patch4_window12_384	[125]
Swin s-w7-224	swin_small_patch4_window7_224	[125]
Swin t-w7-224	swin_tiny_patch4_window7_224	[125]
Twin pcvt-b	twins_pcpvt_base	[28]
Twin PCPVT	twins_pcpvt_base	[28]
Twins pcvt-l	twins_pcpvt_large	[28]
Twins pcvt-s	twins_pcpvt_small	[28]
Twins svt-b	twins_svt_base	[28]
Twins svt-l	twins_svt_large	[28]
Twins svt-s	twins_svt_small	[28]

Figure 13.12: Pre-trained models used from: [208]

13.3 Every Class of ImageNet Dataset Inverted

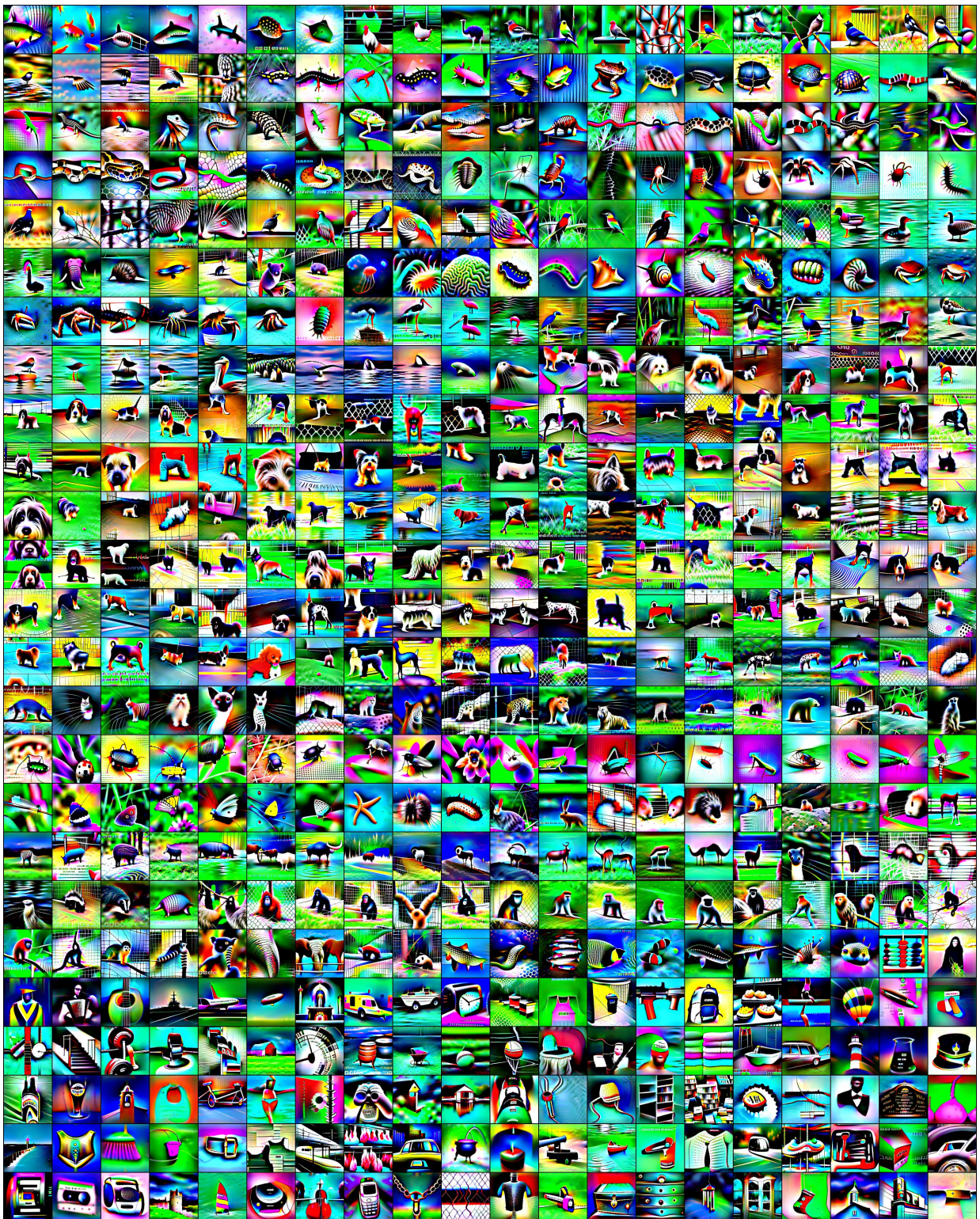


Figure 13.13: Inversion of first 500 classes of ImageNet for the Robust Model.

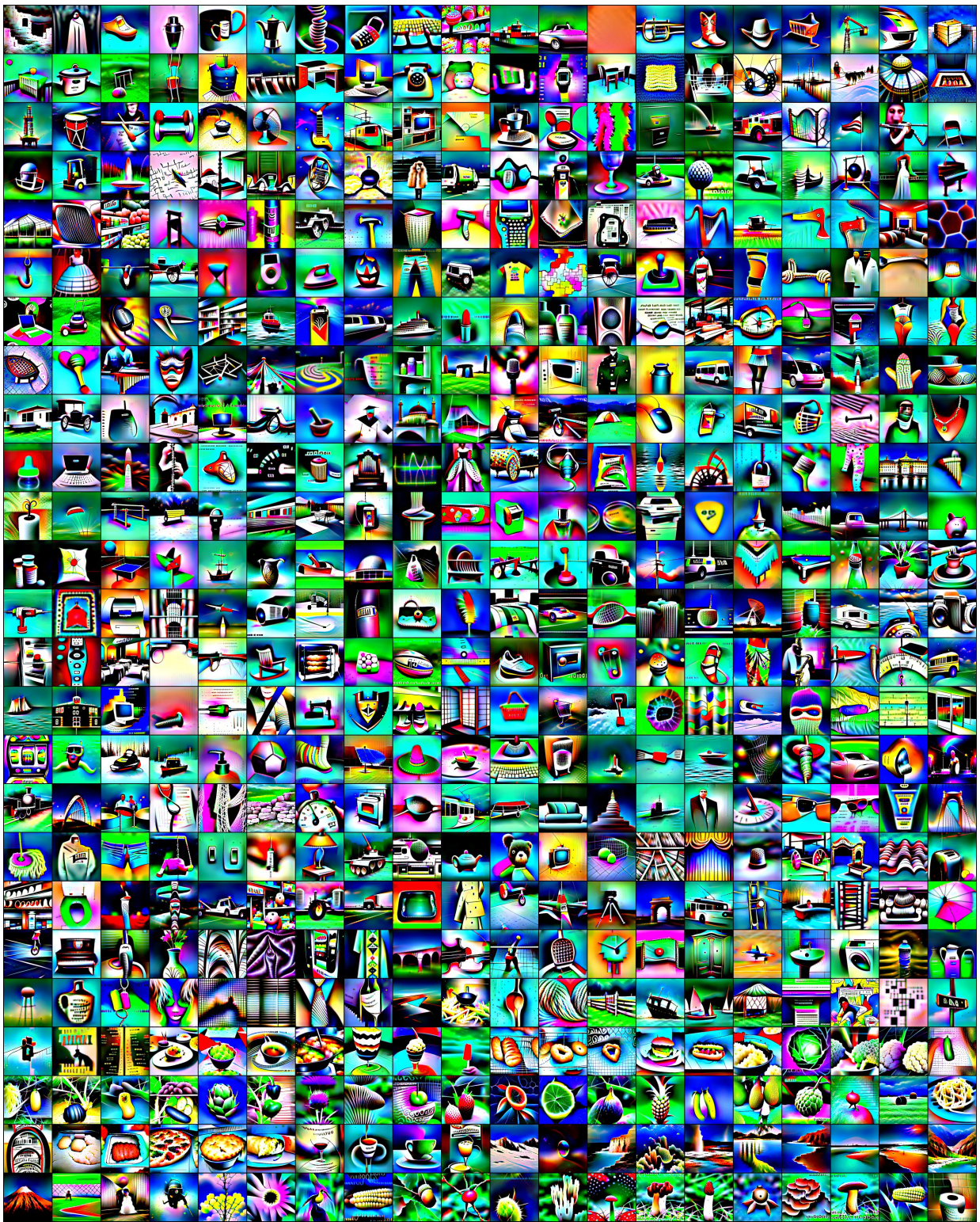


Figure 13.14: Inversion of second 500 classes of ImageNet for the Robust Model.

13.4 Additional experimental setting

13.4.1 Robust models

We use a robust ResNet-50 [74] model free-trained [174] on the ImageNet dataset [40]. The setting we use for inverting robust models is very similar to that of PII explained in section 6.4 except for some differences. Throughout the paper, we use centering for robust models unless otherwise is mentioned (like when we are examining the effect of zoom and centering themselves). We use 0.0005 to scale total variation in the loss function. Also, we do not apply the data normalization layer before feeding the input to the network. In PII experiment setting, we apply a random ColorShift at each optimization step to each element in the ensemble. In the robust setting, we do not update the ColorShift variables μ , and σ for a fixed patch size, and we update these variables for the ensemble when we use a new patch size. Although using ColorShift would alleviate the need for using TV as discussed in section 6.3.2, and illustrated in figure 6.4 in the robust, and natural setting, we keep TV in our robust setting to make this setting more similar to that of previous inversion methods and to emphasize that it is a toy example for our ablation studies.

13.5 Optimization algorithm

13.6 CIFAR-100 Results

In Figure 13.15, we use PII to invert ViT models trained on ImageNet and fine-tuned on CIFAR-100. Similarly, Figure 13.16 shows inversion results from models fine-tuned on CIFAR-

Algorithm 2: Optimization procedure for Plug-In Inversion

Input: Model f , class y , final resolution R , ColorShift parameters α, β , ‘ensemble’ size e , randomly initialized $\mathbf{x} \in \mathcal{I}^{3 \times R/8 \times R/8}$

for $s = 1, \dots, 7$ **do**

 Upsample $\mathbf{x}$ to resolution $\frac{(2s+1)R}{16} \times \frac{(2s+1)R}{16}$

 Pad $\mathbf{x}$ with random noise to resolution $\frac{(s+1)R}{8} \times \frac{(s+1)R}{8}$

for $i = 1, \dots, 400$ **do**

$\mathbf{x}' = \text{Jitter}(\mathbf{x})$

for $n = 1, \dots, e$ **do**

 Draw $\mu \sim U(-\alpha, \alpha)^3, \sigma \sim \exp(U(-\beta, \beta))^3$

$\mathbf{x}_n = \text{ColorShift}_{\mu, \sigma}(\mathbf{x}')$

$\mathcal{L} = \frac{1}{e} \sum_{n=1}^e NLL(f(\mathbf{x}_n), y)$

$\mathbf{x} \leftarrow \text{Adam}_i(\mathbf{x}, \nabla_{\mathbf{x}} \mathcal{L})$

return $\mathbf{x}$

10. We emphasize that these were produced using identical settings to the ImageNet results in section 6.5.

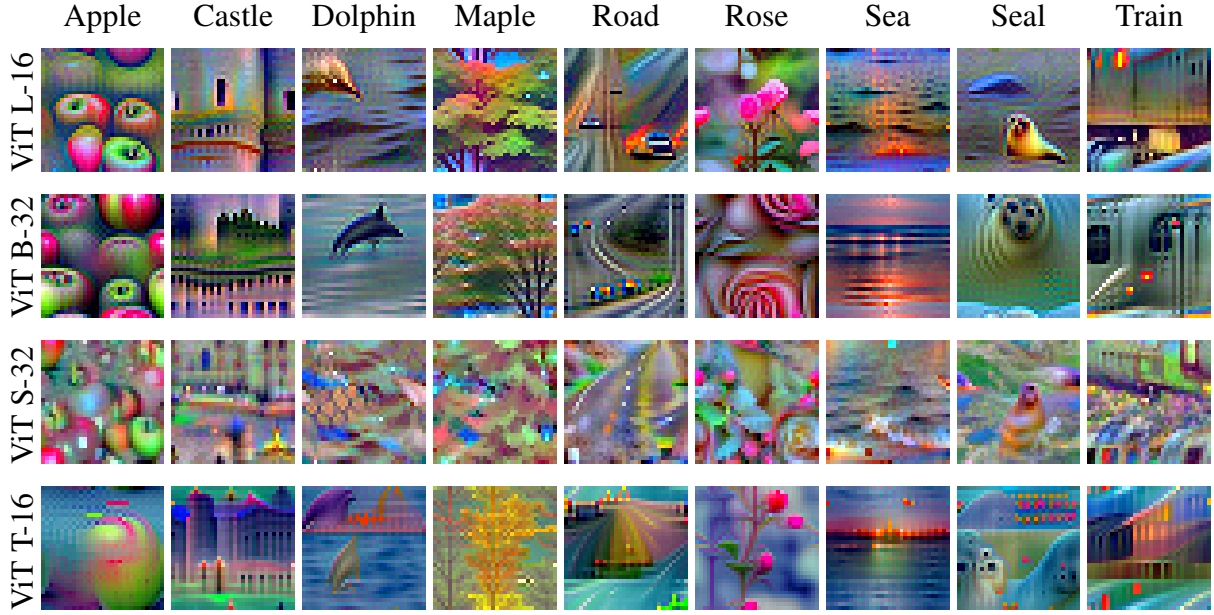


Figure 13.15: Inverting different CIFAR-100 model and class combinations using PII.

13.7 Additional baseline comparisons

Figures 13.17 and 13.18 depict images inverted from various models using the DeepInversion and DeepDream methods as outlined by Yin et al. [226]. More specifically, we use DeepInversion with the prescribed regularization weights for CNNs, and use the same procedure minus the feature regularizer for all other models (in which BatchNorm is not used), keeping the same weights on the remaining terms¹. We see less consistent performance across models using this method than when using PII, illustrating the need for model-specific tuning when using regularization-based approaches.

¹This is the implementation of DeepDream [137] considered by Yin et al. [226].

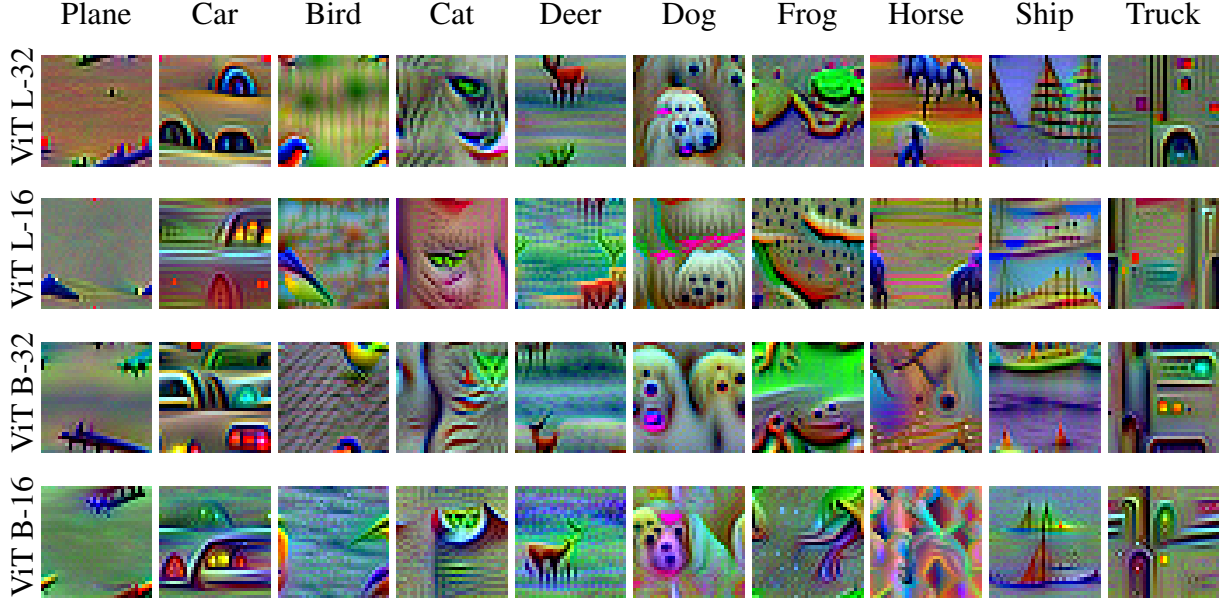


Figure 13.16: Inverting different CIFAR-10 model and class combinations using PII.

13.8 Quantitative Results

To quantitatively evaluate our method, we invert a pre-trained ViT model to produce one image per class using PII, and do the same using DeepDream (i.e., DeepInversion minus feature regularization, which is not available for this model). We then use a variety of pre-trained CNN, ViT, and MLP models to classify these images. We find that every model achieves strictly higher top-1 and top-5 accuracy on the PII-generated image set (excepting the ‘teacher’ model, which perfectly classifies both). We compile these results in figure 13.19. Additionally, we compute the Inception score [168] for both sets of images, which also favors PII over DeepDream, with scores of 28.17 ± 7.21 and 2.72 ± 0.23 , respectively.

We also perform the same evaluation for images generated from a pre-trained ResMLP model. These results are more mixed; DeepDream images are classified much better by a small number of models, but the majority of models classify PII images better, and the average accuracy

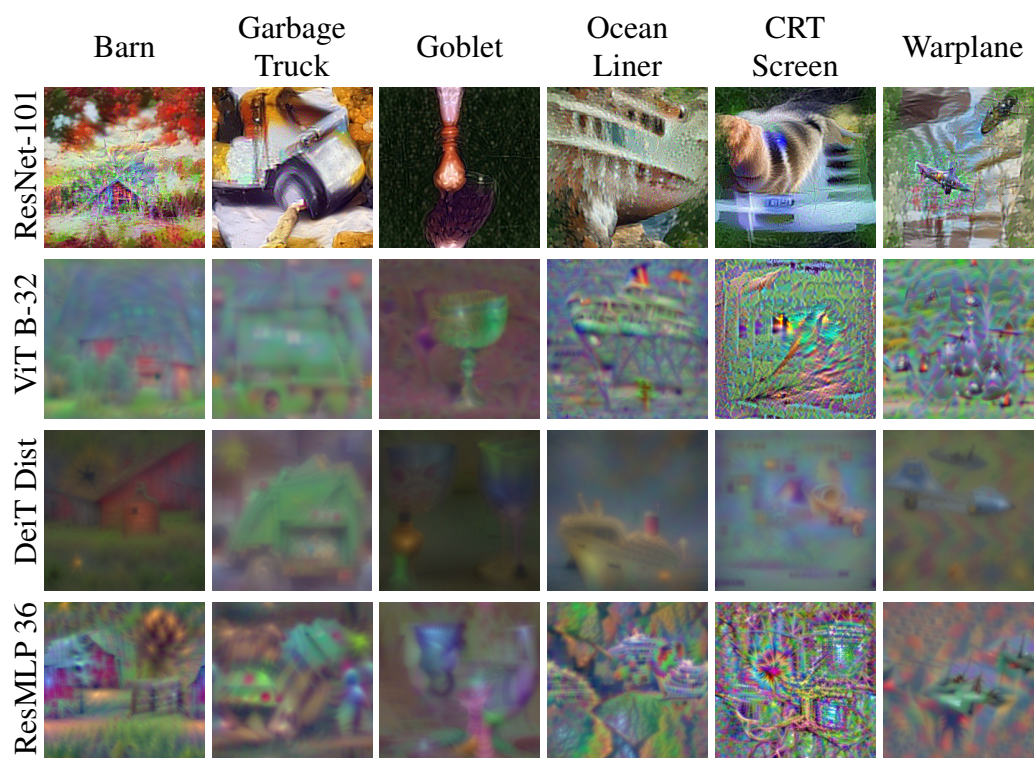


Figure 13.17: Inverting different model and class combinations for different classes using DeepInversion (top row) / DeepDream (other rows). Cross-reference figure 6.8.

across models is approximately equal for both methods. Inception score, however, once again clearly favors PII over DeepDream, with scores of 6.79 ± 2.18 and 3.27 ± 0.47 , respectively.

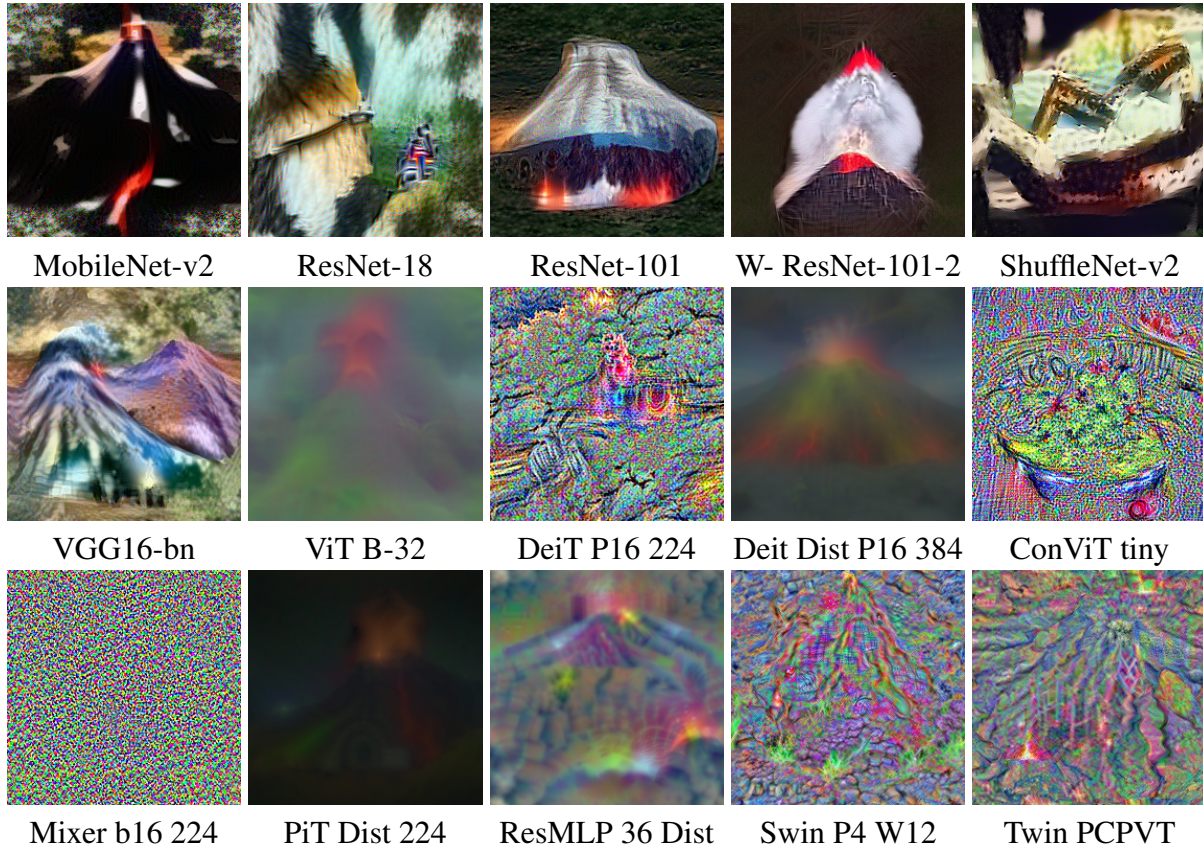


Figure 13.18: Images inverted from the Volcano class for various Convolutional, Transformer, and MLP-based networks using DeepInversion (CNN models) / DeepDream (non-CNN models). Cross-reference figure 6.7.

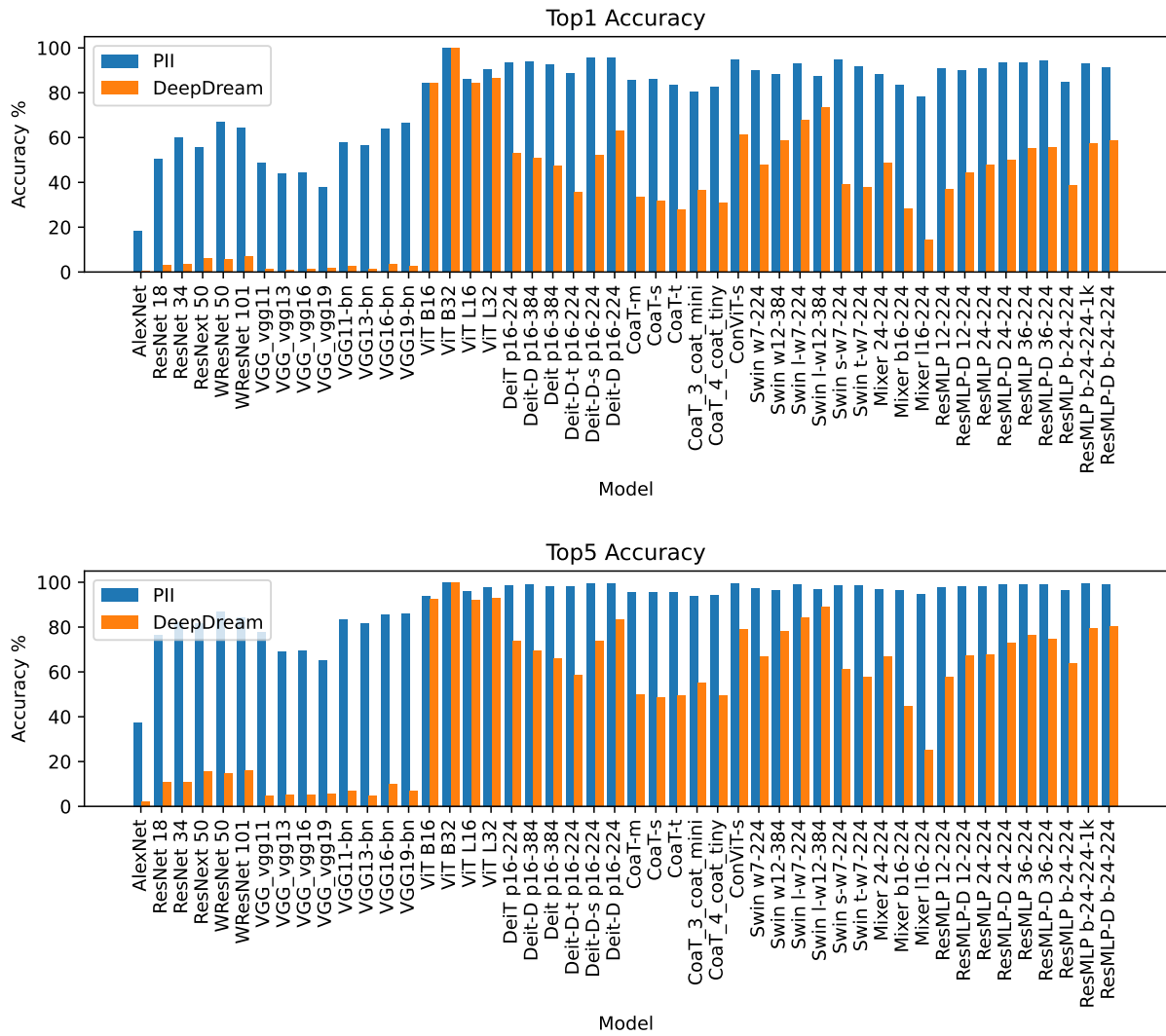


Figure 13.19: Top-1 (a) and top-5 (b) classification accuracy of various CNN, ViT, and MLP models evaluated on images generated from ViT B-32 using PII and DeepDream.

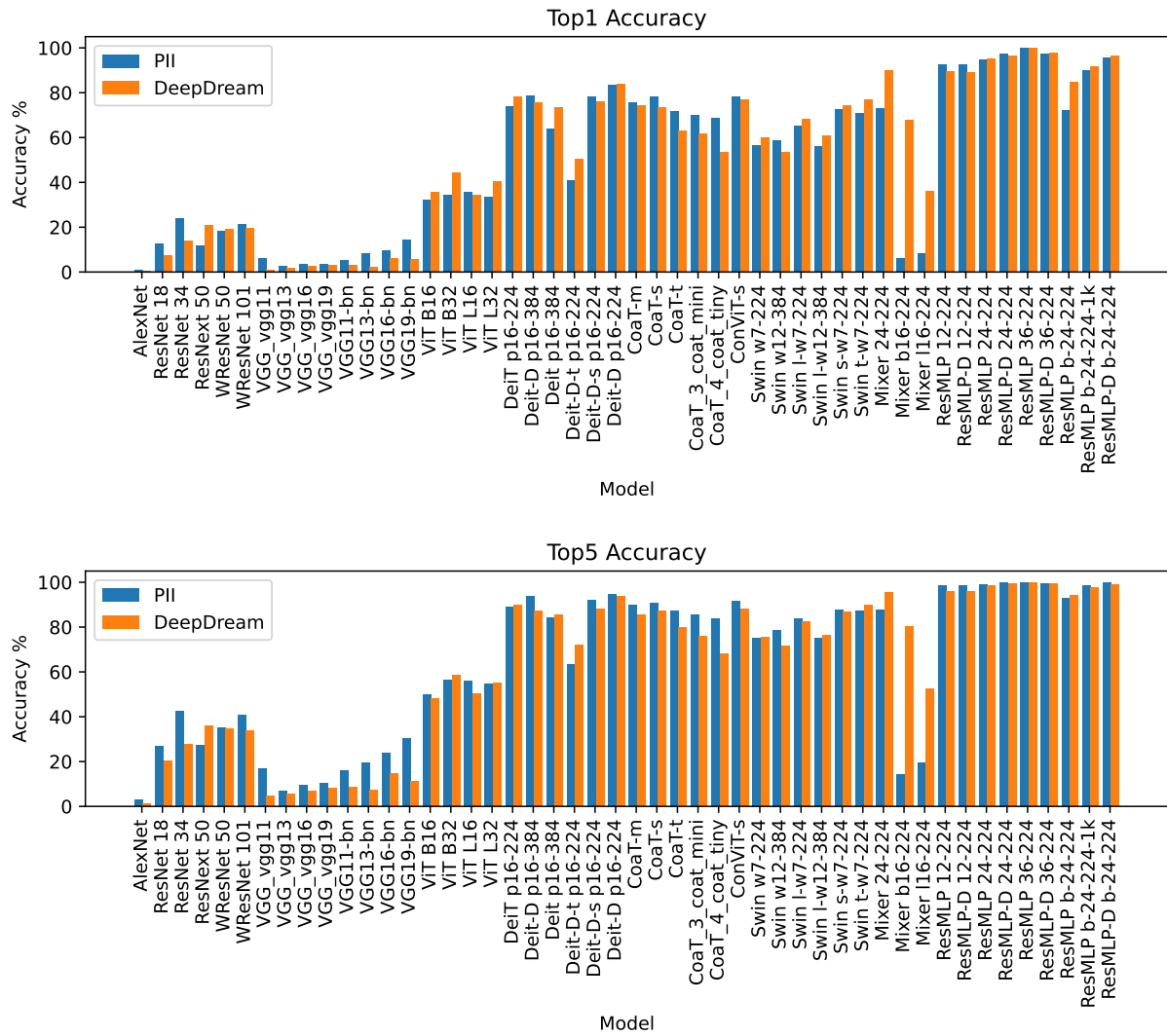


Figure 13.20: Top-1 (a) and top-5 (b) classification accuracy of various CNN, ViT, and MLP models evaluated on images generated from ResMLP 36-224 using PII and DeepDream.

Appendix 14: Appendix For Visualizing Vision Transformers

14.1 Failed Examples

Figure 14.1 shows few examples of our visualization method failing when applied on low dimensional spaces. We attribute this to entanglement of more than 768 features when represented by vectors of size 768. We note that, due to skip connections, activation in previous layers can cause activation in the next layer for the same feature, consequently, the visualizations of the same features in different layers can share visual similarities.

14.2 Experimental Setup and Hyperparameters

As mentioned before, we ensemble augmentations to the input. More specifically, we use $\mathcal{A}$ is $GS(CS(Jitter(x)))$ as our augmentation. The bound for *Jitter* is $(-32, 32)$ for both directions vertical and horizontal. The hyper parameters for *CS* are always $mean = 1.0$ and $std = 1.0$ in all of the experiments. For *GS*, the $mean$ is always 0, however, for the std we have a linear scheduling, where at the beginning of the optimization the $std = 0.5$ and at the end of the optimization $std = 0.0$. We use a batch-size $n = 8$ for all of our experiments. We use ADAM as our choice of optimizer with $\beta = (0.5, 0.99)$. Optimization is done in 400 steps and at every step, we re-sample the augmentations *Jitter*, *GS* and *CS*. We also use a CosineAnealing learning rate scheduling, starting from $lr = 0.1$ at the beginning and $l = 0$ the end. The hyper-parameter used for total variation $\lambda_{tv} = 0.00005$.

For all ViT experiments, we use the pretrained models from <https://github.com/lukemelas/PyTorch-Pretrained-ViT>. For clip models, we use pretrained models by [208]. The rest of models we use are from <https://github.com/pytorch/vision>.

For all of our experiments, we use GeForce RTX 2080 Ti GPUs with 12GB of memory. All inferences on ImageNet are done under 20 minutes on validation set and under 1 hour on training set using only 1 GPU. All visualization experiments take at most 90 seconds to complete.

14.3 Models

In our experiments, we use publicly available pre-trained models from various sources. The following tables list the models used from each source, along with references to where they are introduced in the literature.

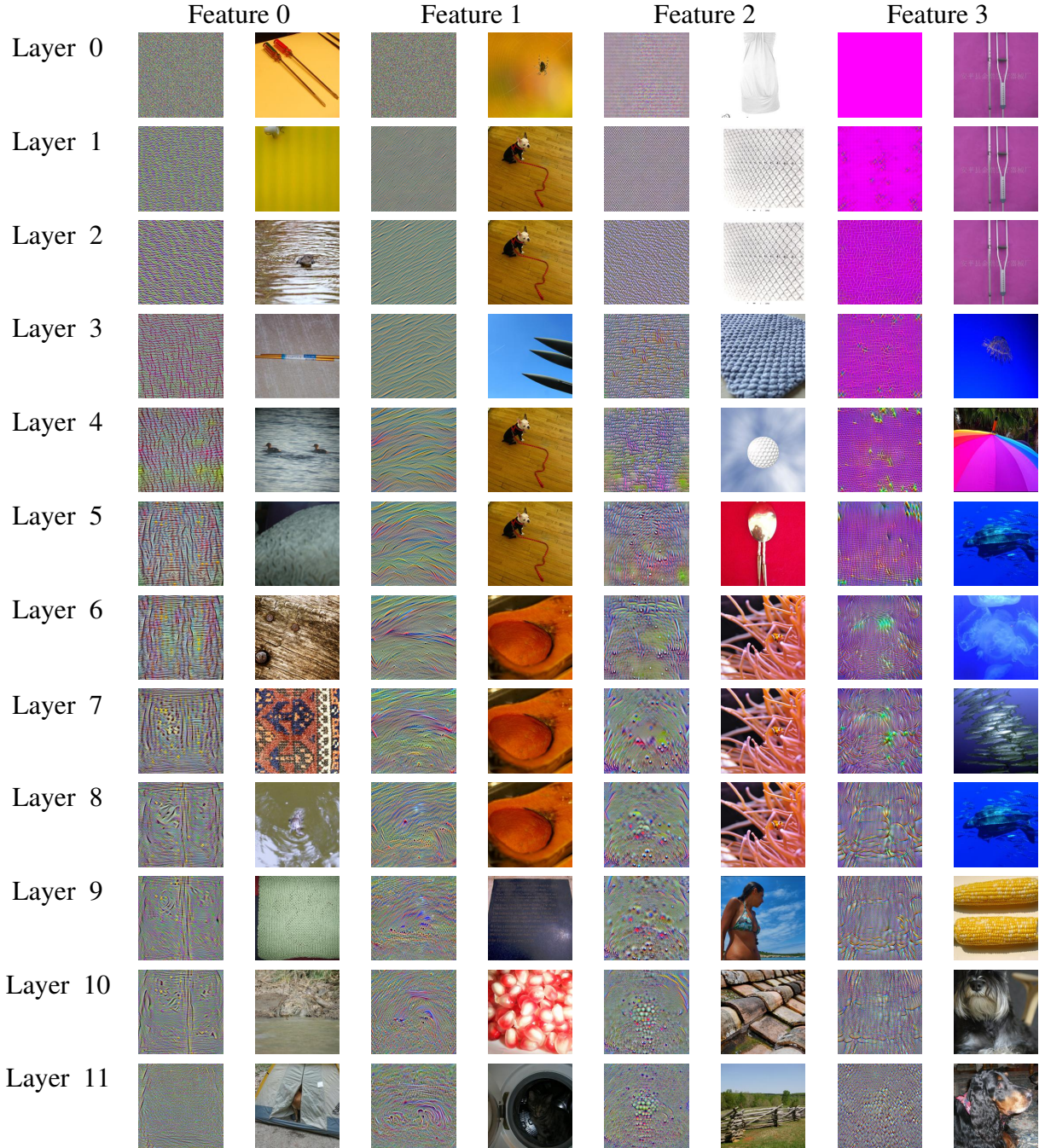


Figure 14.1: **Some examples of failed visualizations on the input of the attention layers.** Same visualization technique fails when applied on low dimensional (e.g. on key, query, value, etc) spaces. We believe that the visualization shows roughly meaningful and interpretable visualizations in early layers, since there are not many different features to be embedded. However, in deeper layers the features are entangled, so it is more difficult to visualize them. For every example, the picture on the left shows the results of optimization and the picture on the right shows the most activating image from ImageNet1k validation set.

Name	Paper
B_16_imagenet1k	[46]
B_32_imagenet1k	[46]

Figure 14.2: Pre-trained models used from : <https://github.com/lukemelas/PyTorch-Pretrained-ViT>.

Name	Paper
deit_base_patch16_224	[200]
deit_base_distilled_patch16_384	[200]
deit_base_patch16_384	[200]
deit_tiny_distilled_patch16_224	[200]
deit_small_distilled_patch16_224	[200]
deit_base_distilled_patch16_224	[200]

Figure 14.3: Pre-trained models from [199] .

Name	Paper
coat_lite_mini	[223]
coat_lite_small	[223]
coat_lite_tiny	[223]
coat_mini	[223]
coat_tiny	[223]
convit_base	[37]
convit_small	[37]
convit_tiny	[37]
pit_b_224	[80]
pit_b_distilled_224	[80]
pit_s_224	[80]
pit_s_distilled_224	[80]
pit_ti_224	[80]
pit_ti_distilled_224	[80]
swin_base_patch4_window7_224	[125]
swin_base_patch4_window7_224_in22k	[125]
swin_base_patch4_window12_384	[125]
swin_base_patch4_window12_384_in22k	[125]
swin_large_patch4_window7_224	[125]
swin_large_patch4_window7_224_in22k	[125]
swin_large_patch4_window12_384	[125]
swin_large_patch4_window12_384_in22k	[125]
swin_small_patch4_window7_224	[125]
swin_tiny_patch4_window7_224	[125]
twins_pcpvt_base	[28]
twins_pcpvt_large	[28]
twins_pcpvt_small	[28]
twins_svt_base	[28]
twins_svt_large	[28]
twins_svt_small	[28]

Figure 14.4: Pre-trained models used from: [208]

14.4 Effect of low-pass filtering

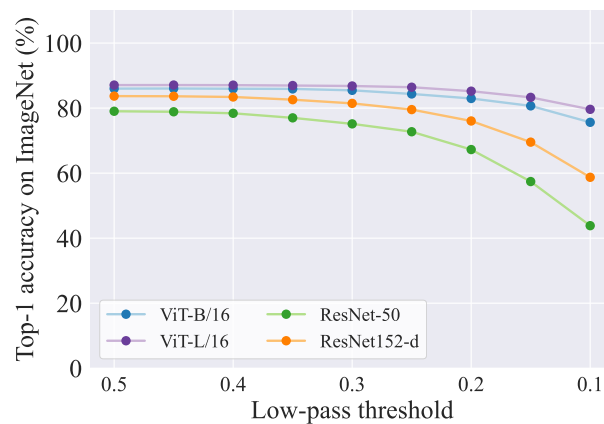


Figure 14.5: **Effect of low-pass filtering on top-1 ImageNet accuracy.** CNNs are *more* dependent on high frequency textural image information than ViTs.

14.5 Additional visualizations

Table 14.1: **ViTs more effectively correlate background information with correct class.** Both foreground and background data are normalized by full image top-1 accuracy.

Normalized Top-1 ImageNet Accuracy			
Architecture	Full Image	Foreground	Background
ViT-B32	89.25	91.53	15.04
ViT-L16	95.00	93.88	19.08
ViT-L32	94.64	90.63	17.67
ViT-B16	92.37	93.70	16.98
ResNet-50	87.67	85.59	9.25
ResNet-152	82.92	82.03	8.24
MobileNetv2	83.77	85.58	8.75
DenseNet121	90.58	86.53	9.72

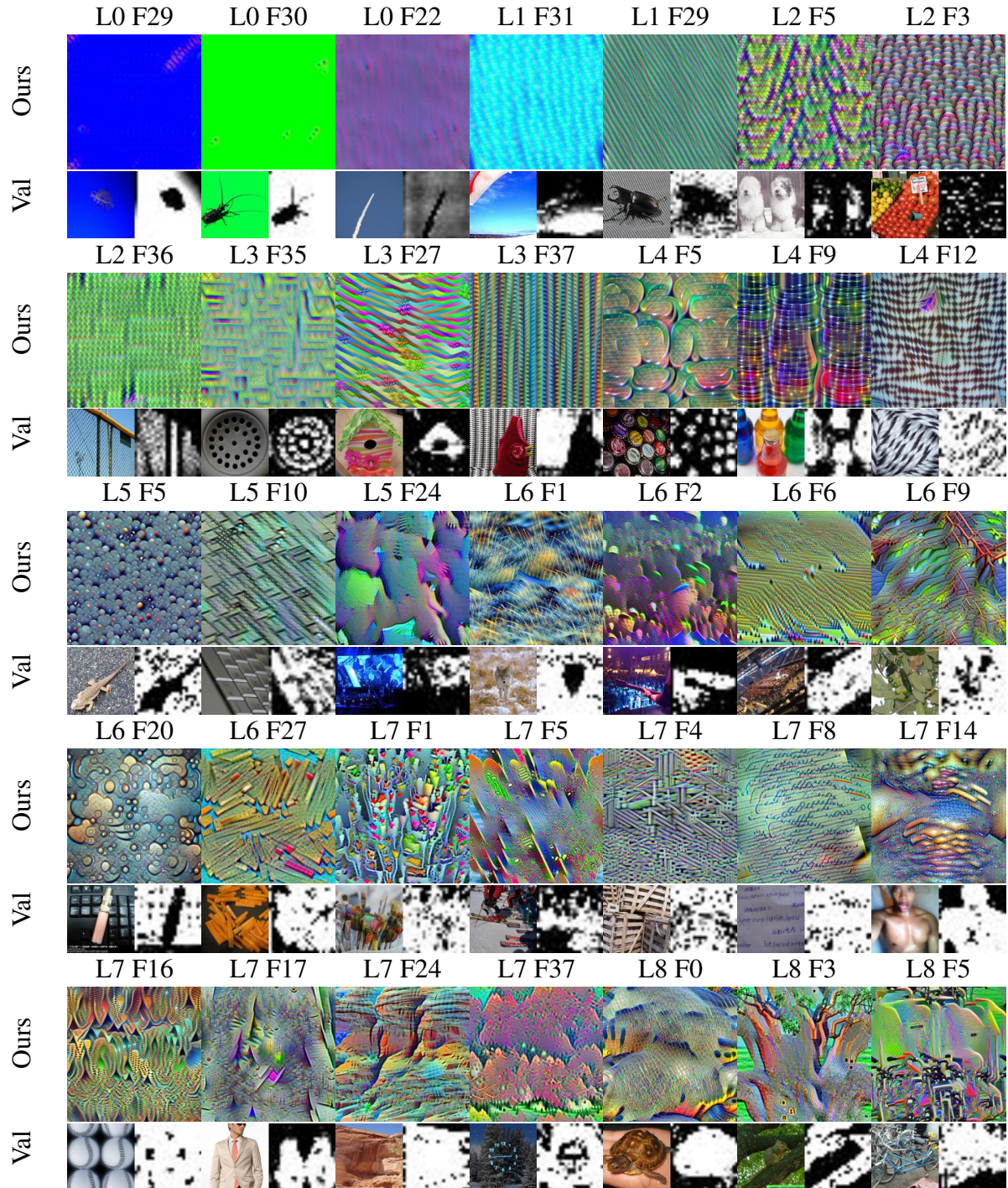


Figure 14.6: Visualization of ViT-base-patch16

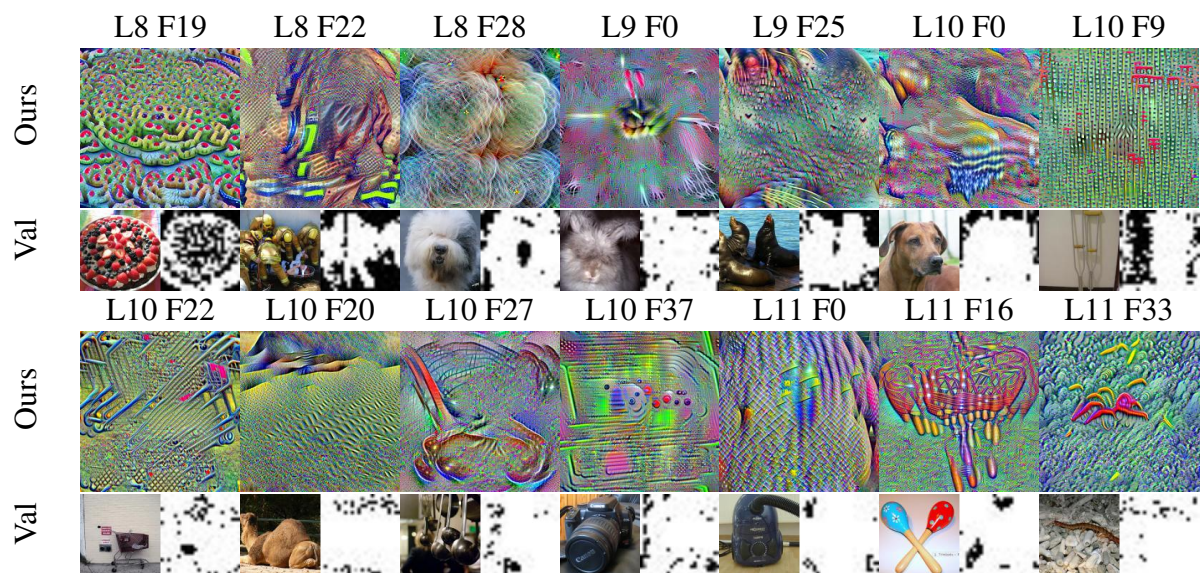


Figure 14.7: (Cont.) Visualization of ViT-base-patch16

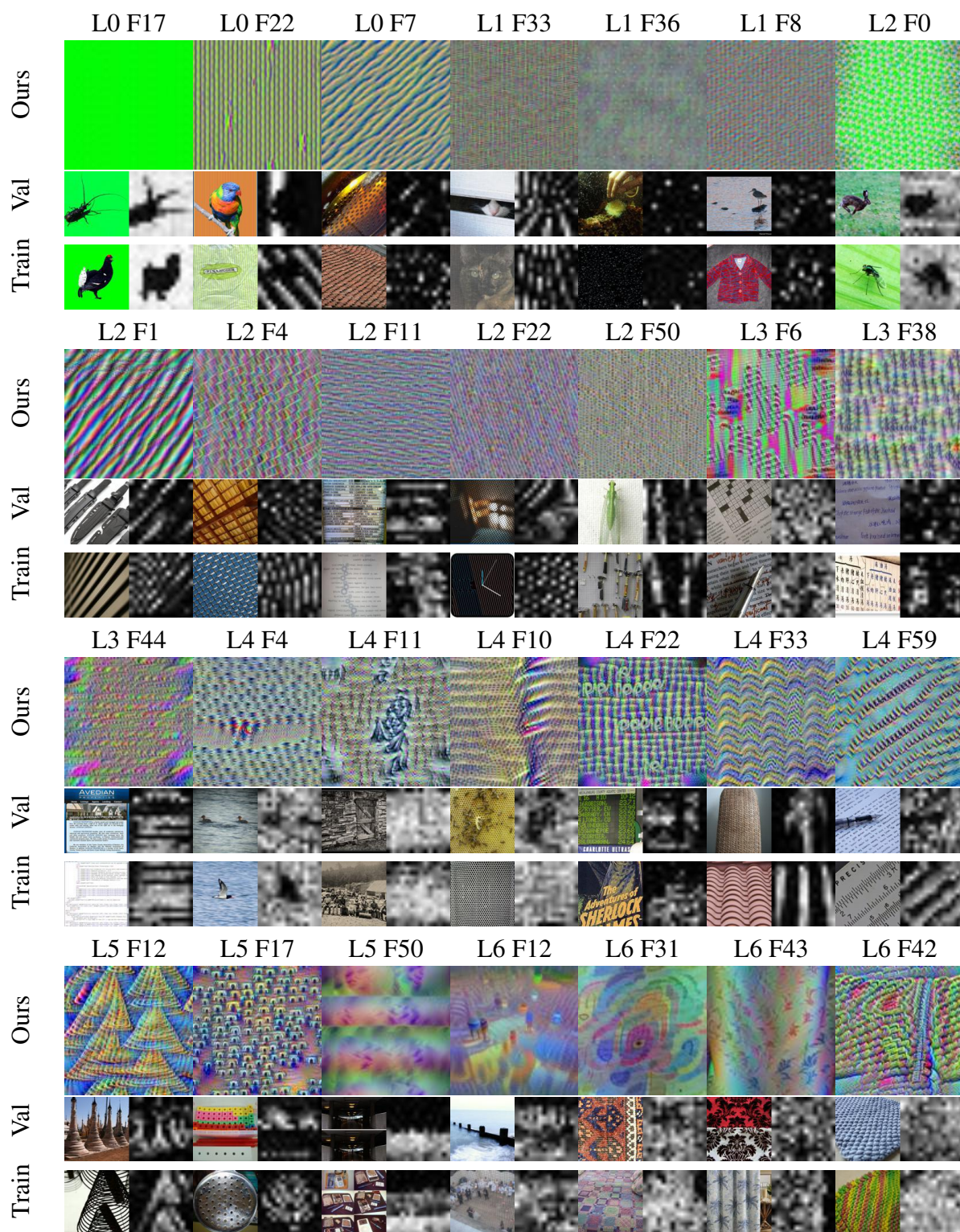


Figure 14.8: Visualization of a CLIP model with ViT-base-patch16 as its visual part.

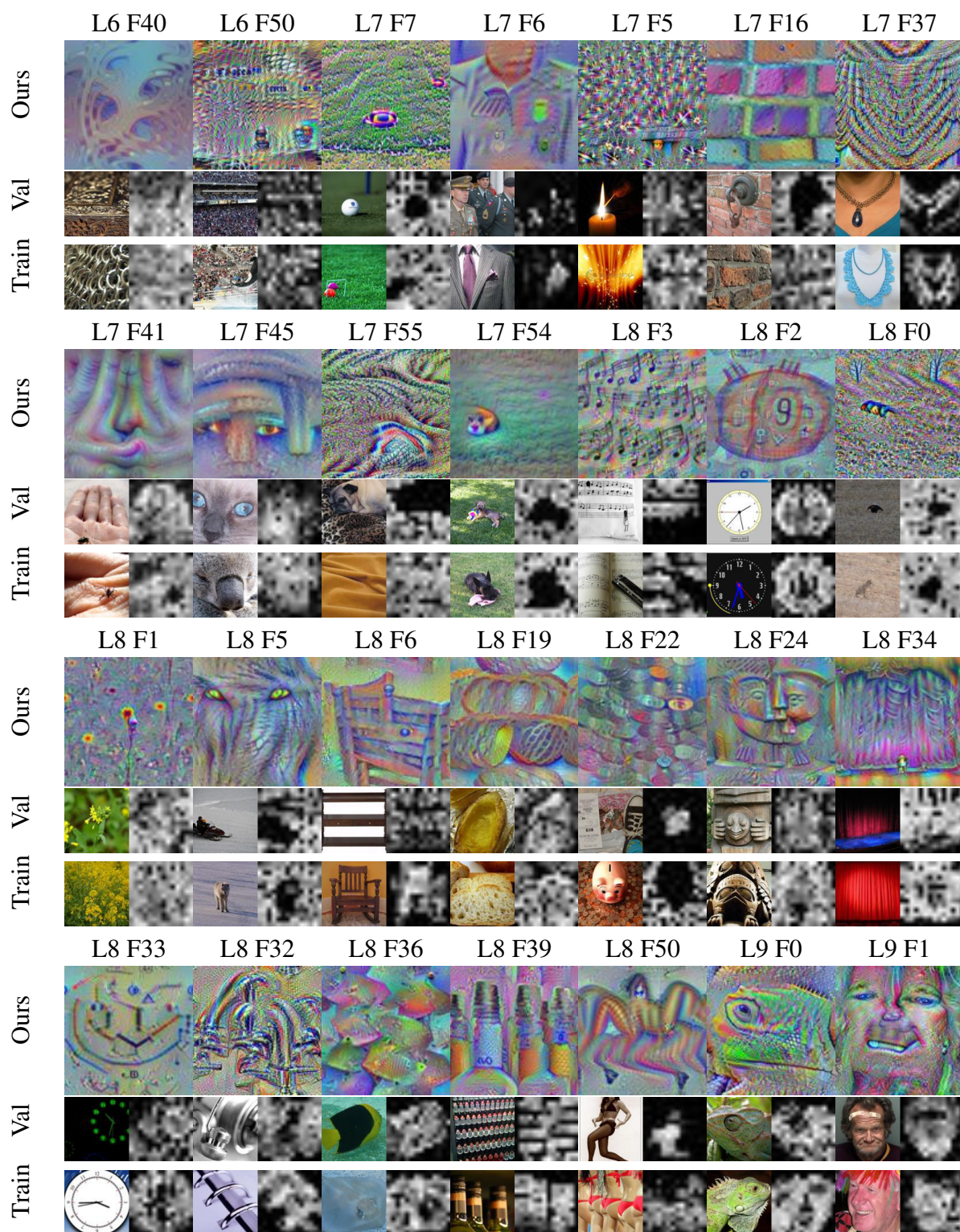


Figure 14.9: (Cont.) Visualization of a CLIP model with ViT-base-patch16 as its visual part.

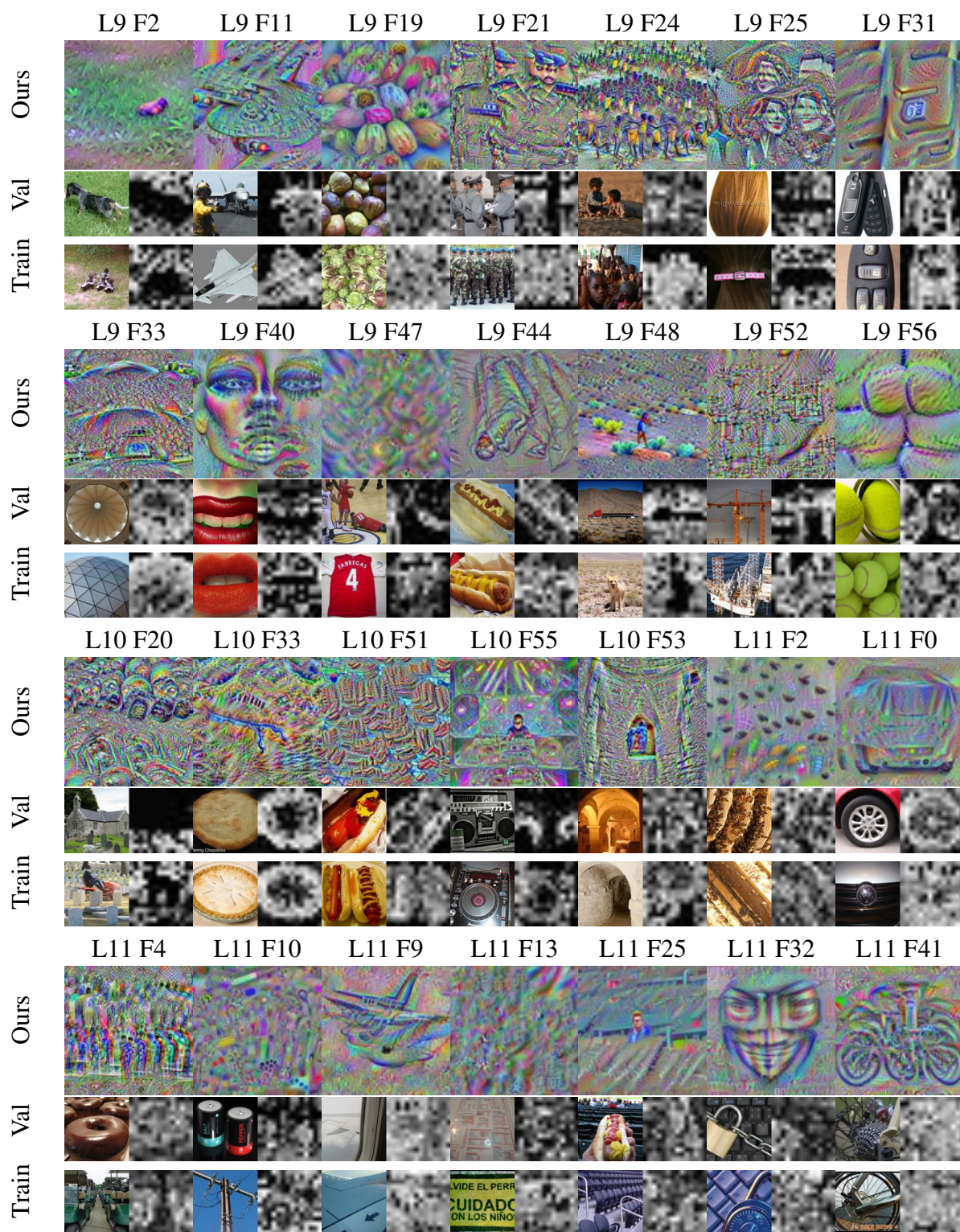


Figure 14.10: (Cont.) Visualization of a CLIP model with ViT-base-patch16 as its visual part.

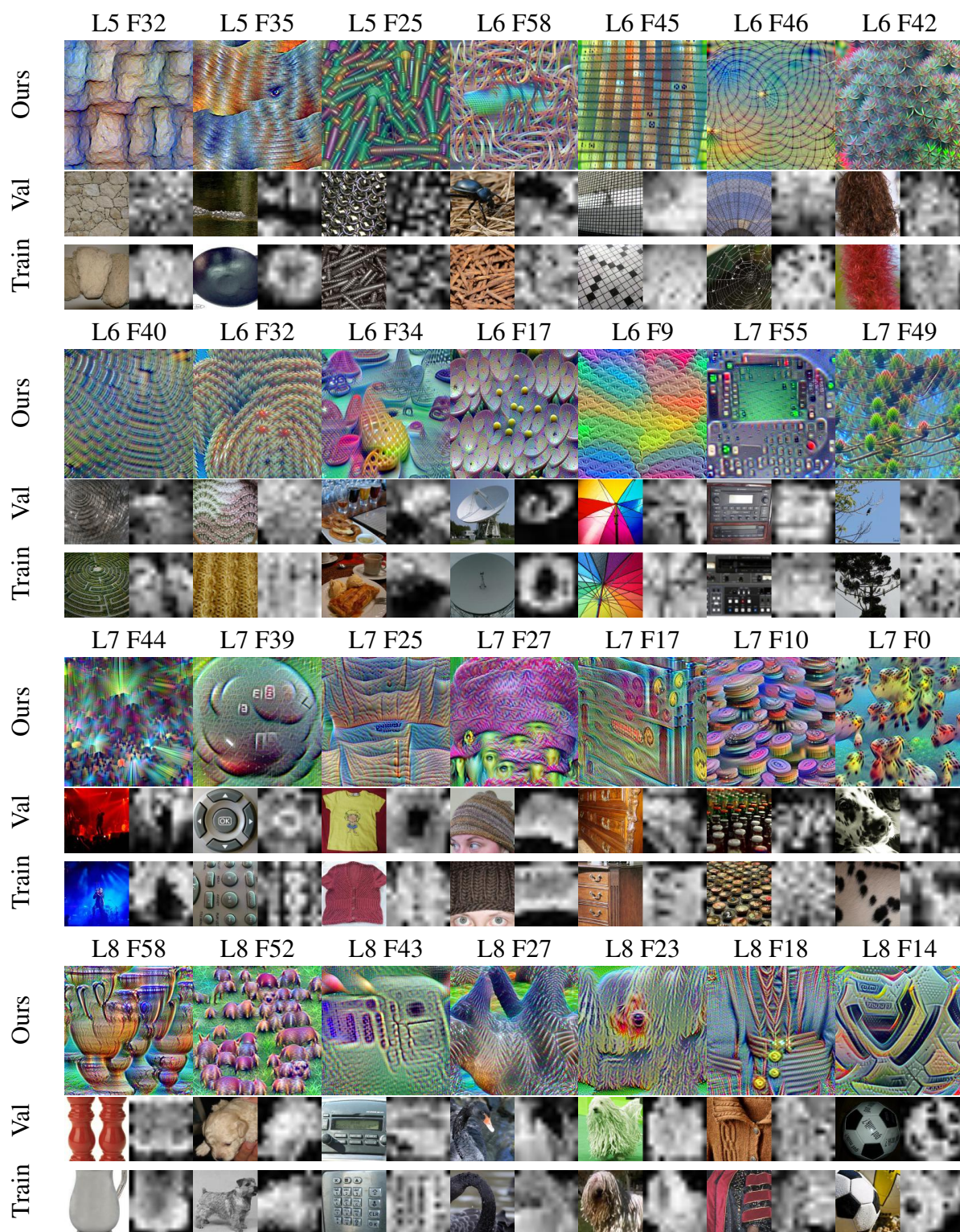


Figure 14.11: Visualization of ViT-base-patch32

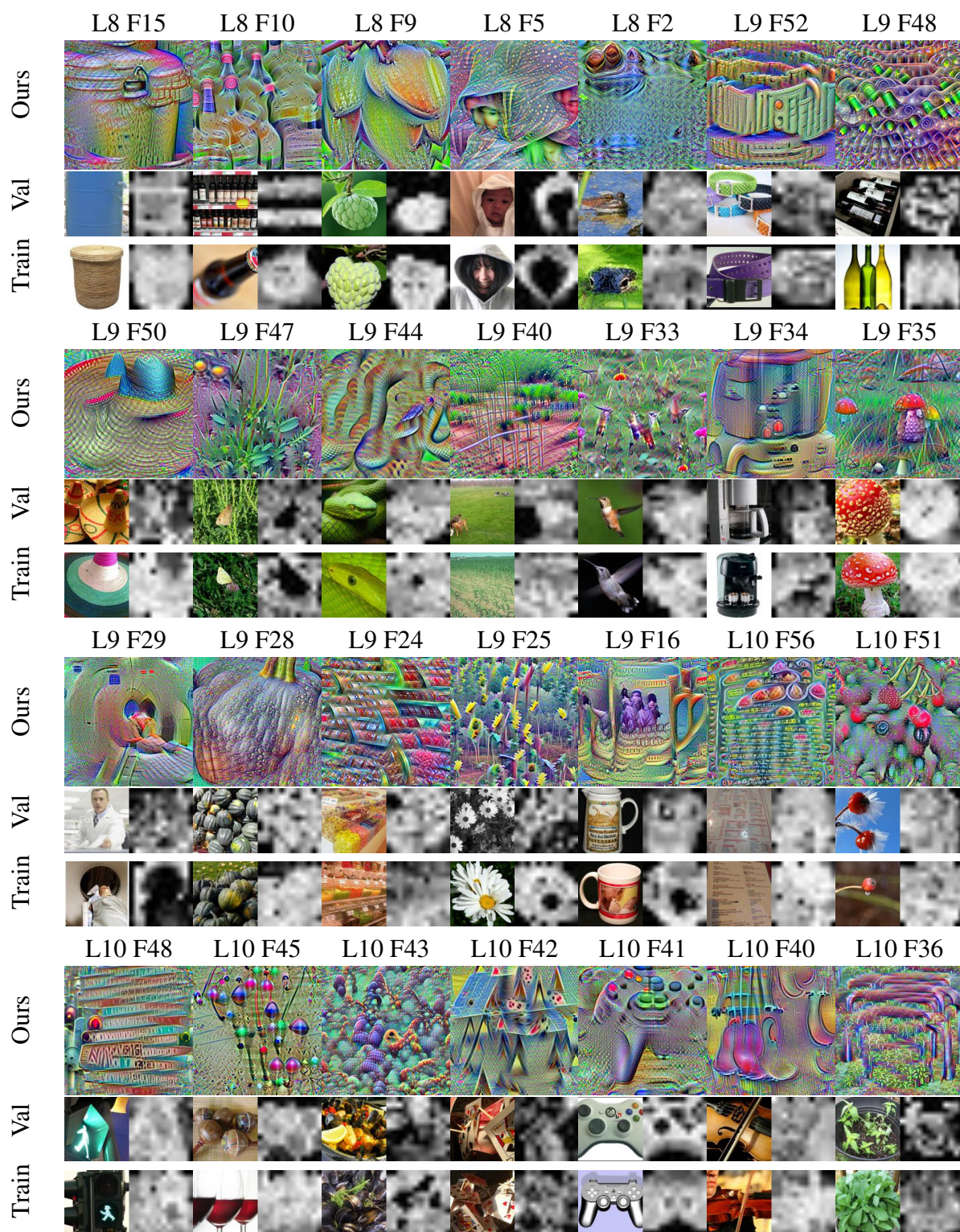


Figure 14.12: (Cont.) Visualization of ViT-base-patch32

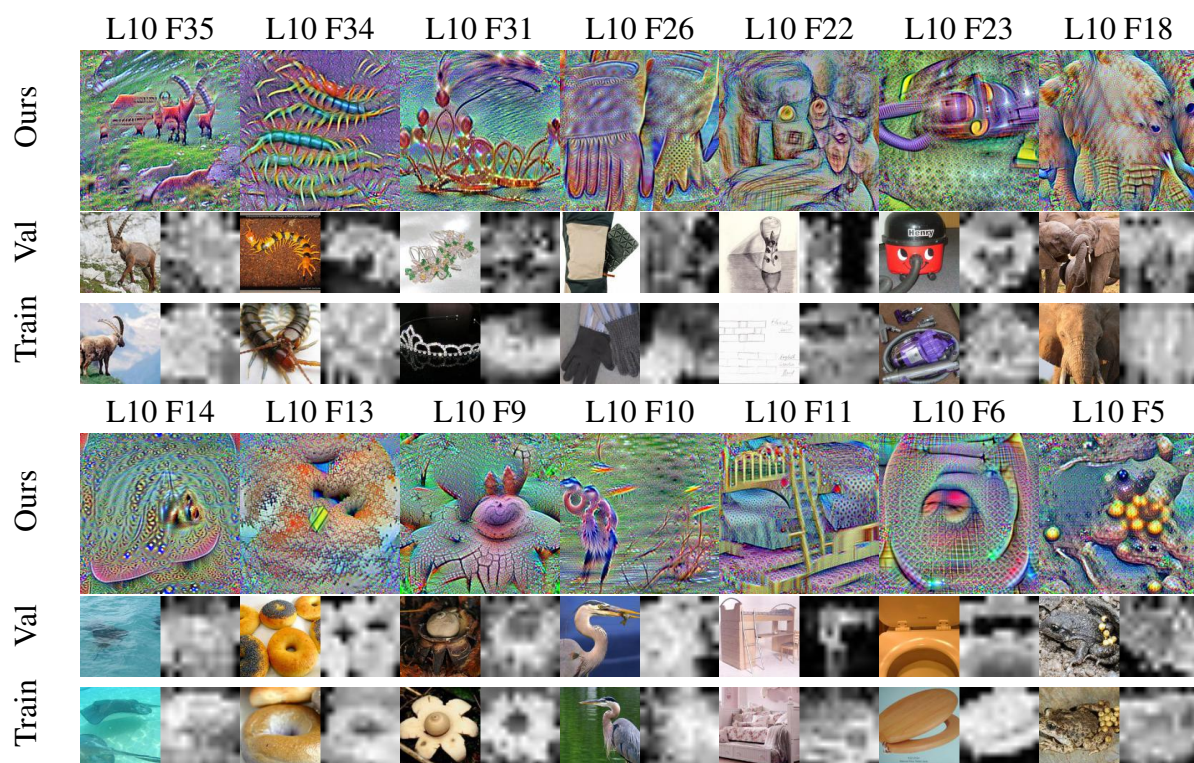


Figure 14.13: (Cont.) Visualization of ViT-base-patch32

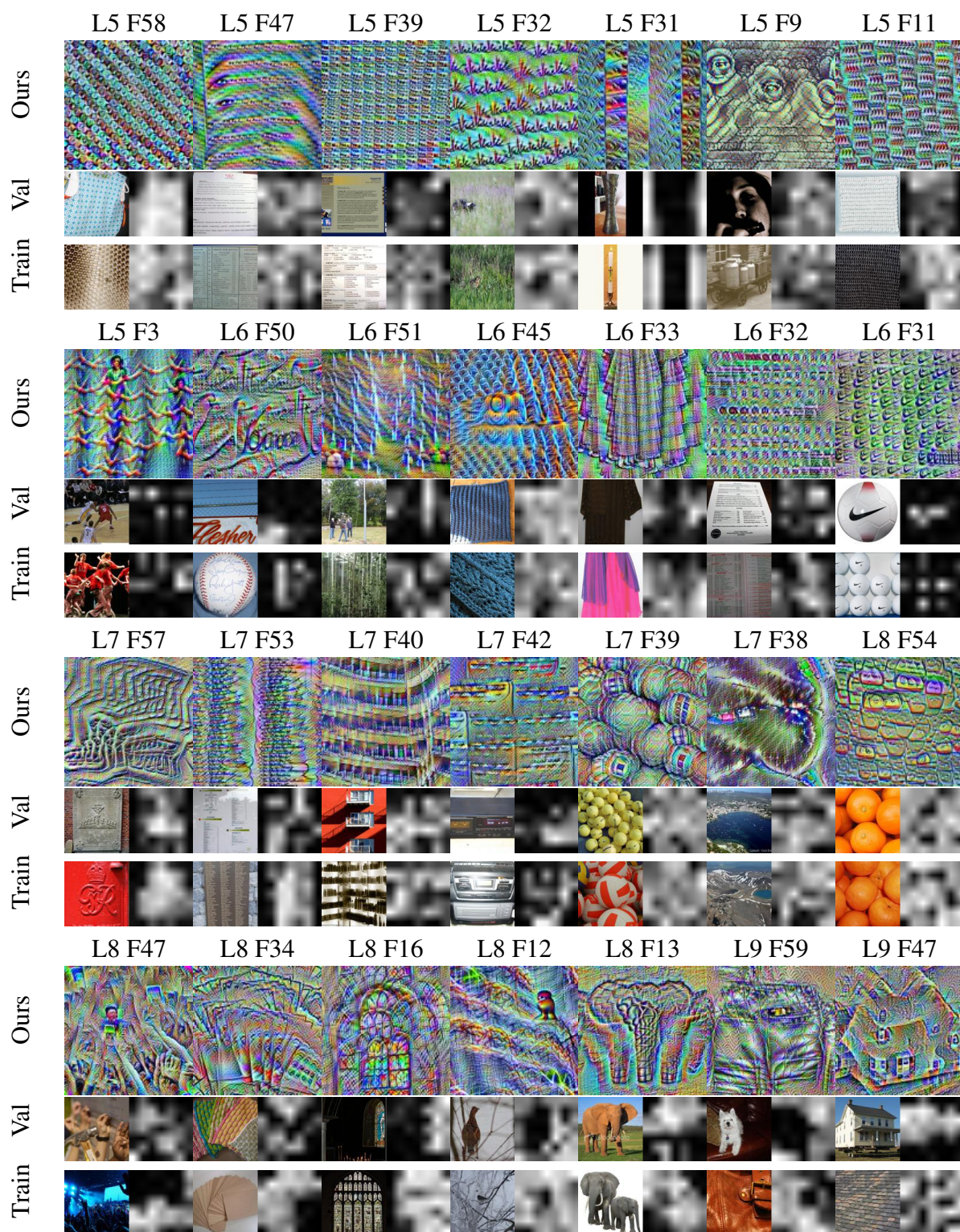


Figure 14.14: Visualization of a CLIP model with ViT-base-patch32 as its visual part.

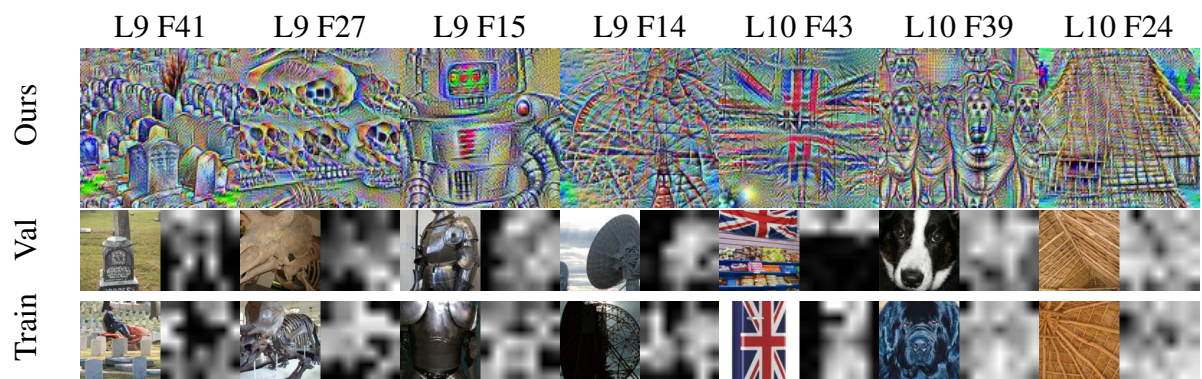


Figure 14.15: (Cont.) Visualization of a CLIP model with ViT-base-patch32 as its visual part.

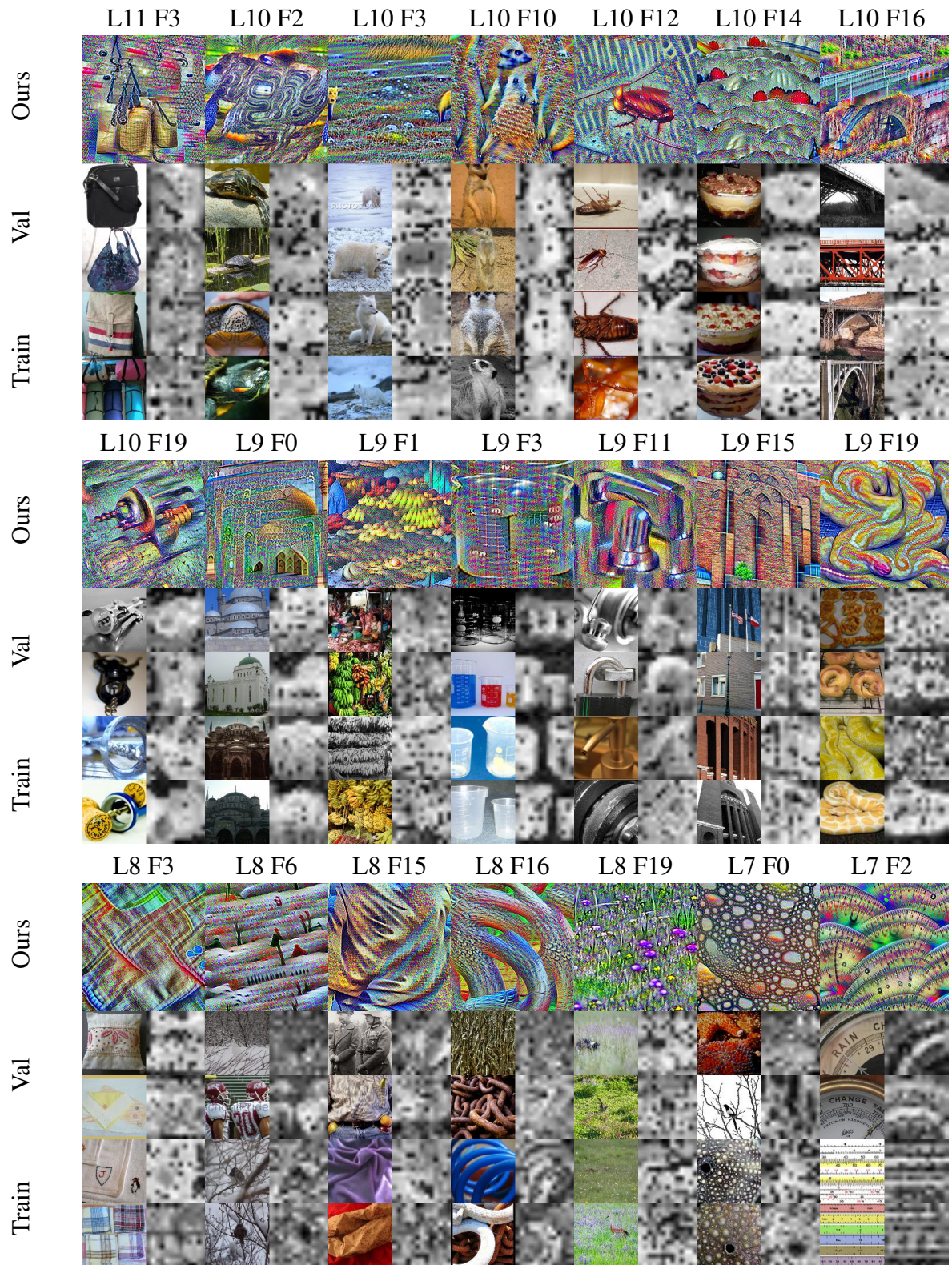


Figure 14.16: Visualization of features in Deit base p-16 im-224

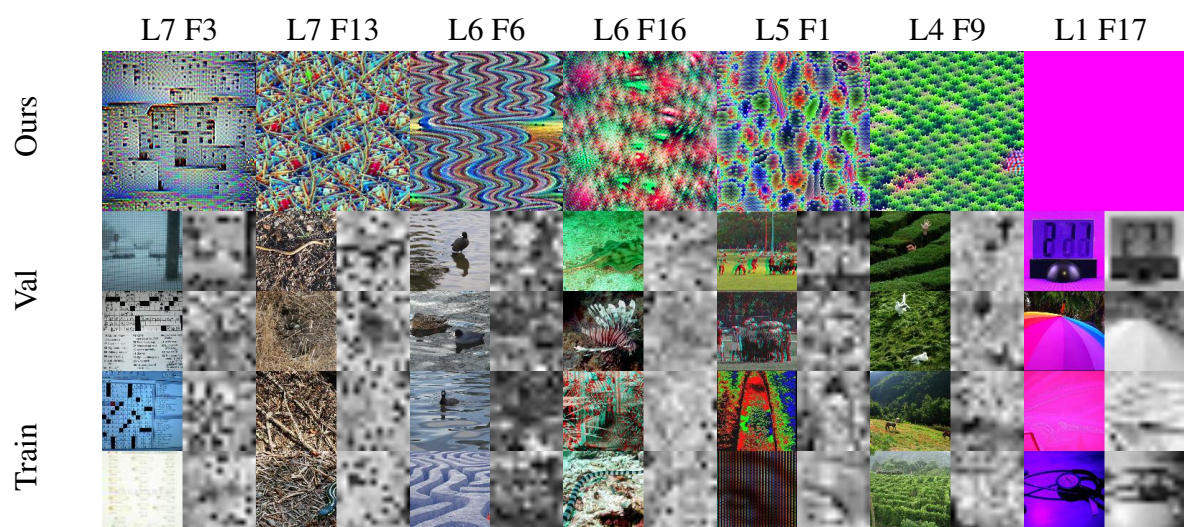


Figure 14.17: (Cont.) Visualization of features in Deit base p-16 im-224

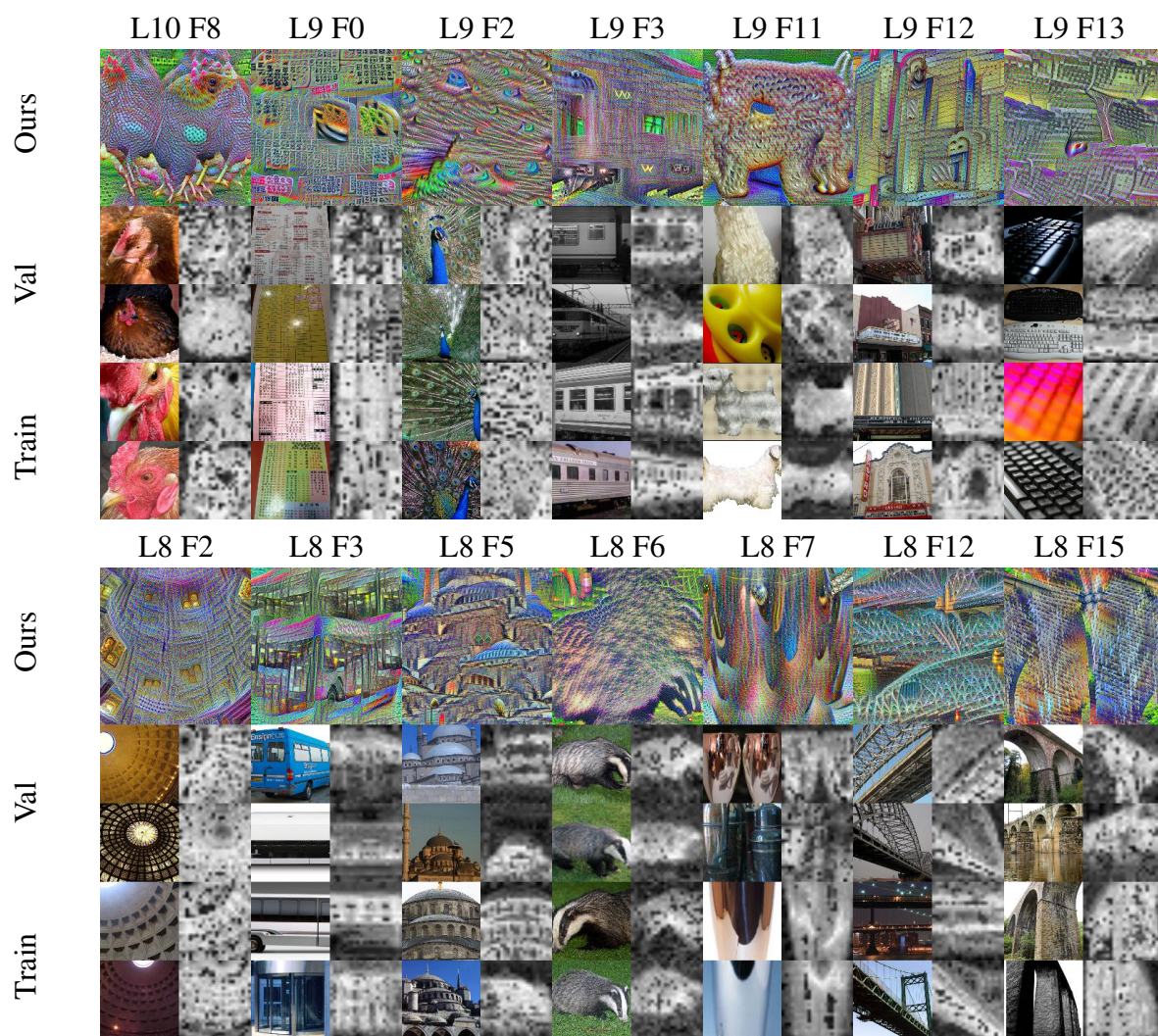


Figure 14.18: Visualization of features in DeiT base p-16 im-384

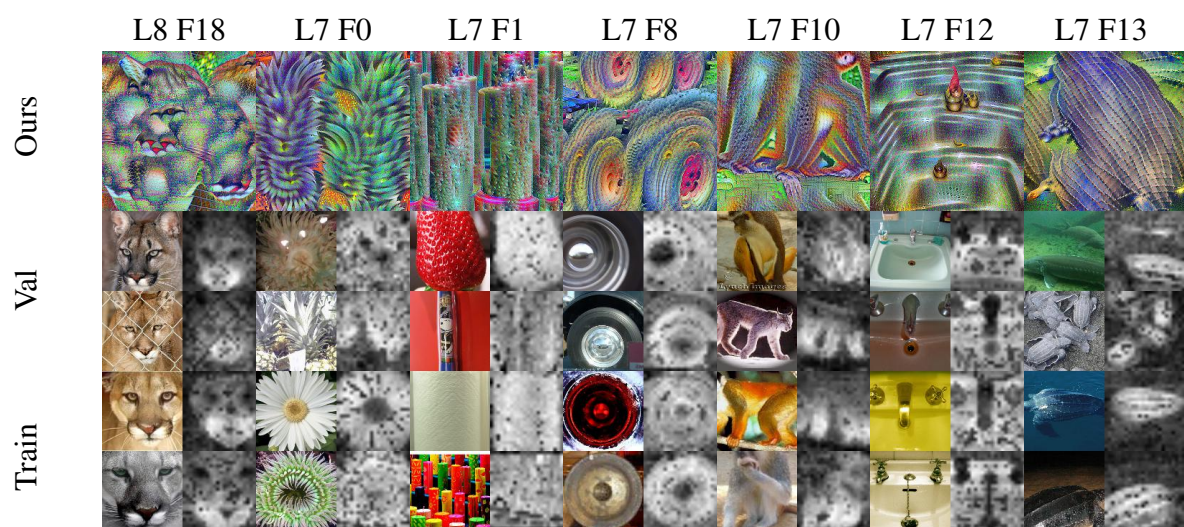


Figure 14.19: (Cont.) Visualization of features in DeiT base p-16 im-384

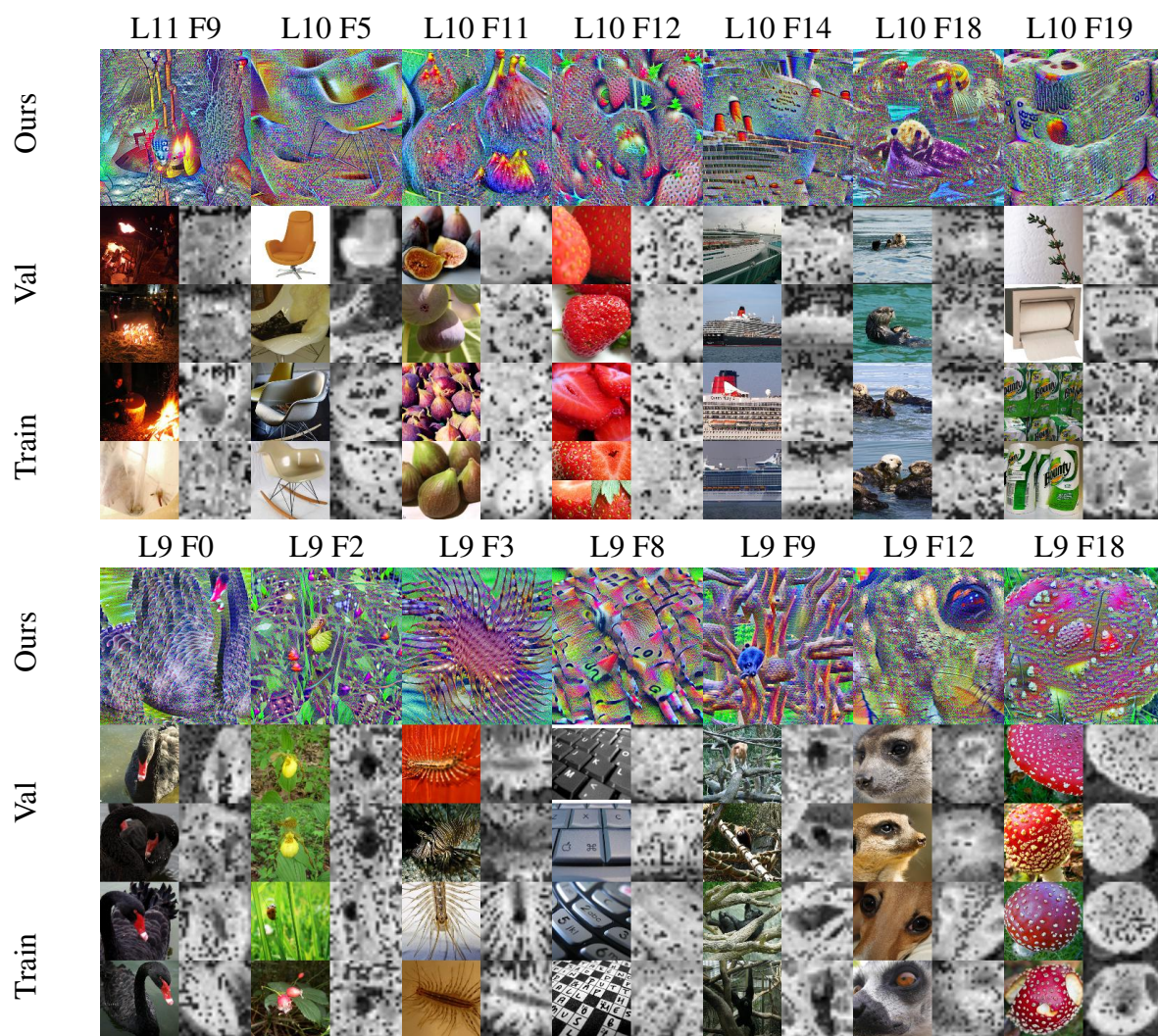


Figure 14.20: Visualization of features in DeiT base p-16 im-384



Figure 14.21: Visualization of features in DeiT tiny distilled p-16 im-224

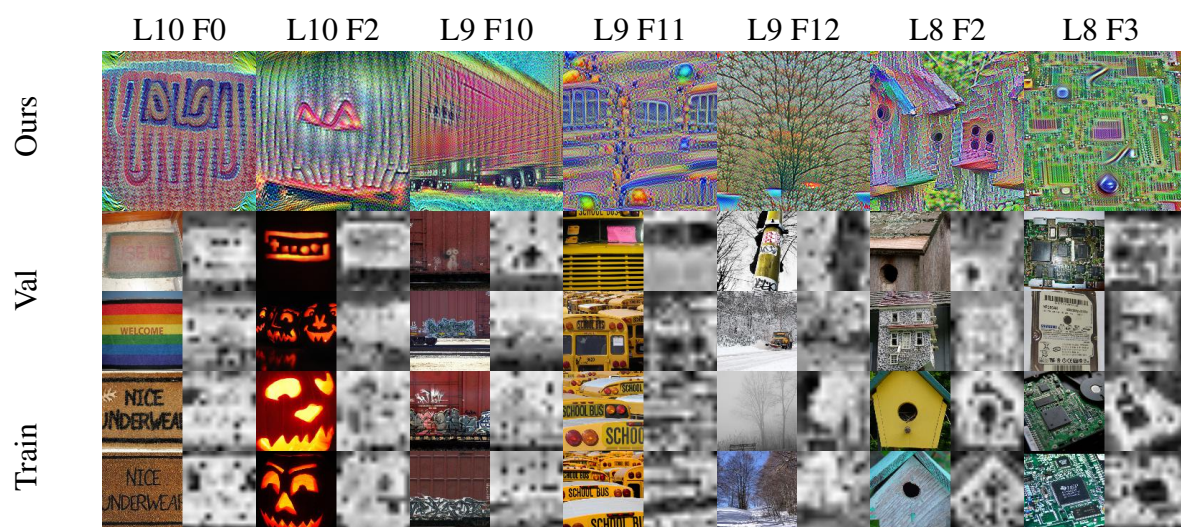


Figure 14.22: Visualization of features in DeiT small distilled p-16 im-224

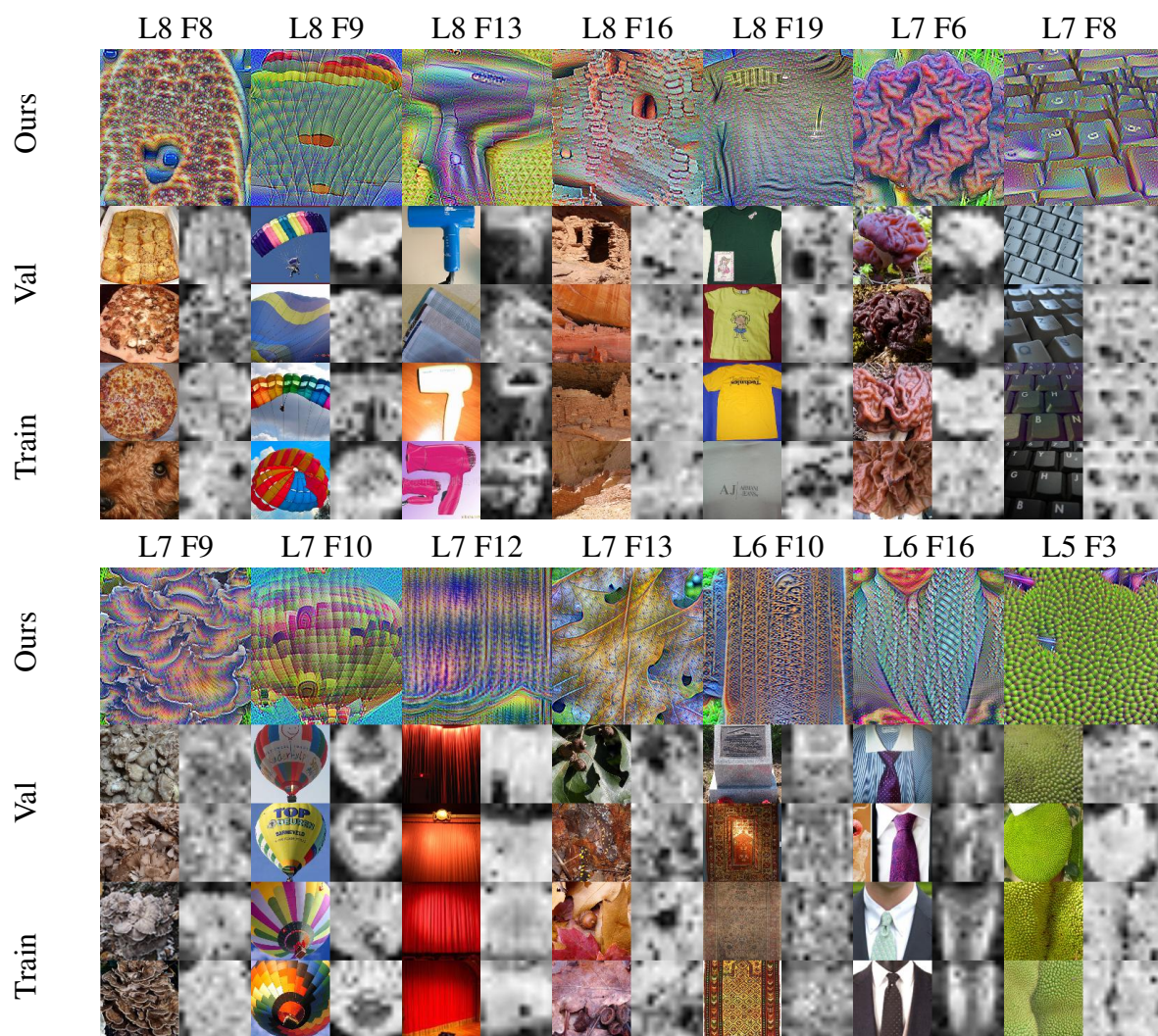


Figure 14.23: Visualization of features in DeiT small distilled p-16 im-224

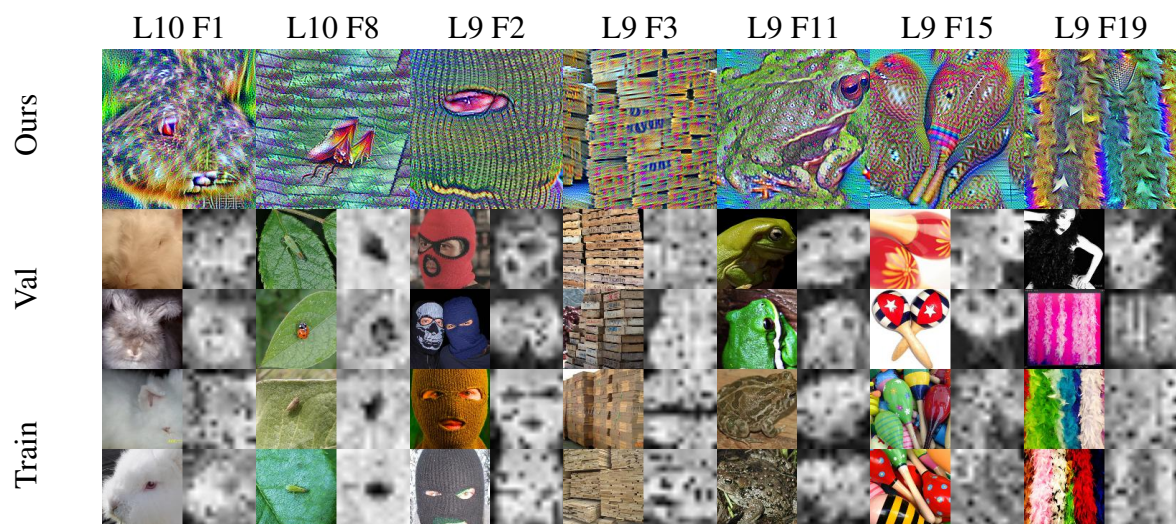


Figure 14.24: Visualization of features in DeiT base distilled p-16 im-224

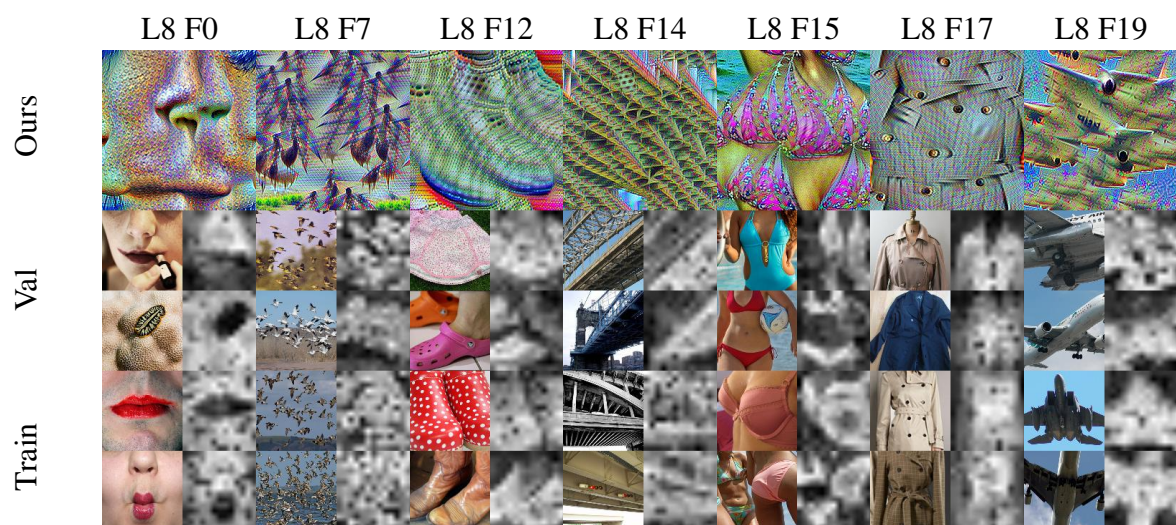


Figure 14.25: (Cont.) Visualization of features in DeiT base distilled p-16 im-224

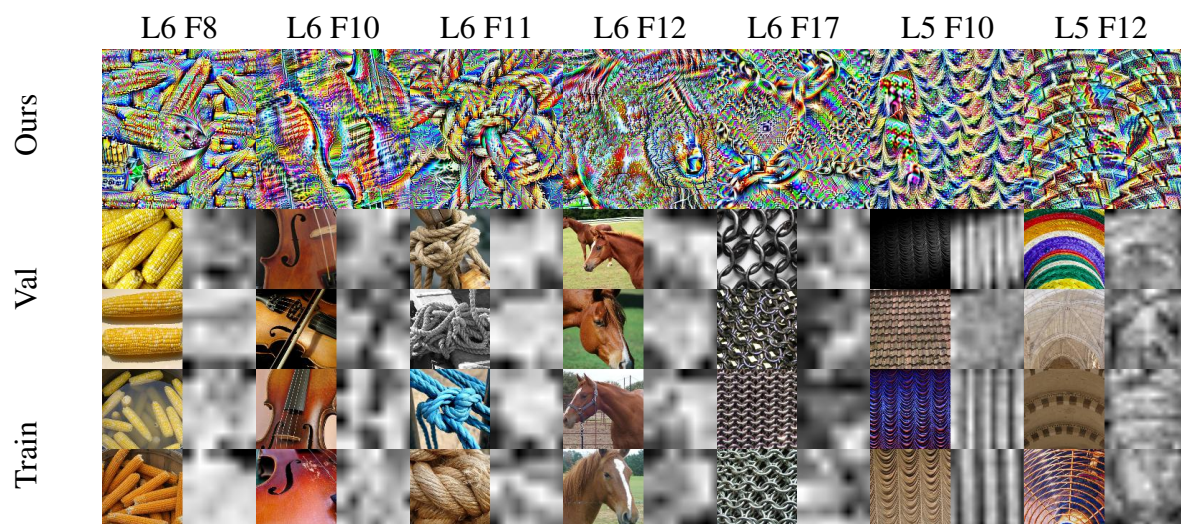


Figure 14.26: Visualization of features in Coat lite mini

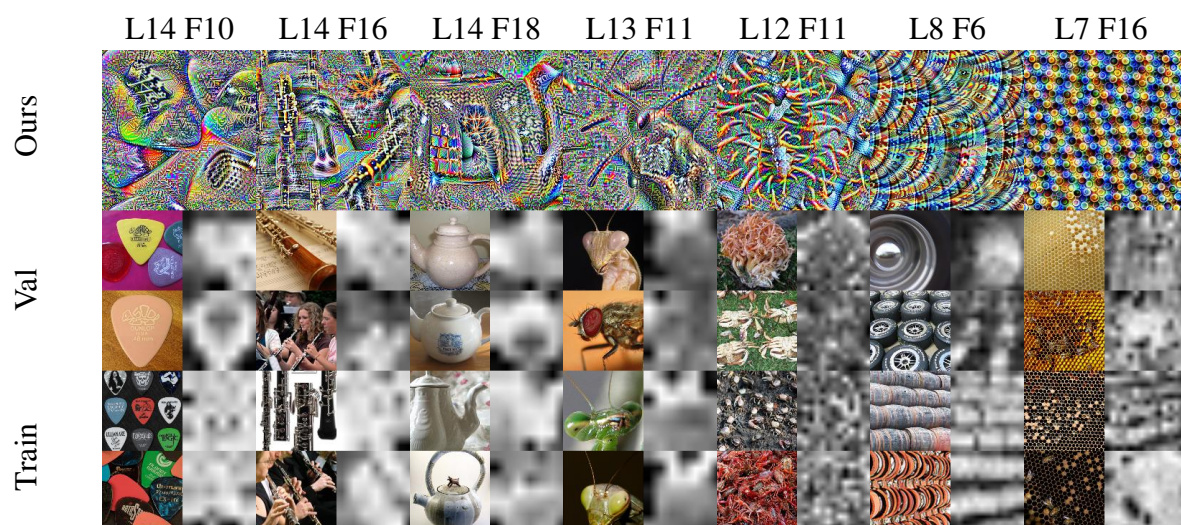


Figure 14.27: Visualization of features in Coat lite small



Figure 14.28: Visualization of features in ConViT base

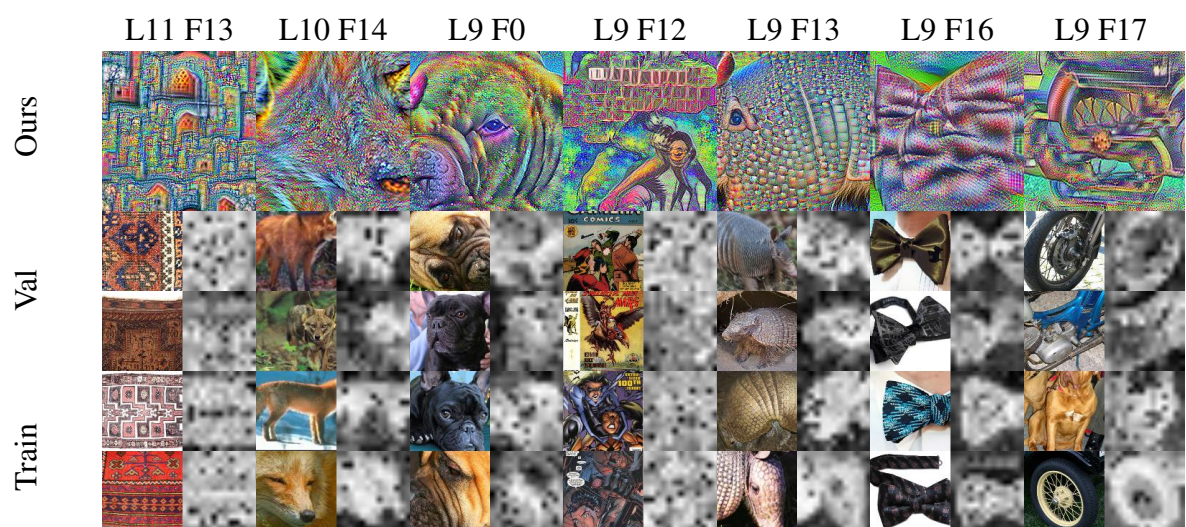


Figure 14.29: Visualization of features in ConViT small.

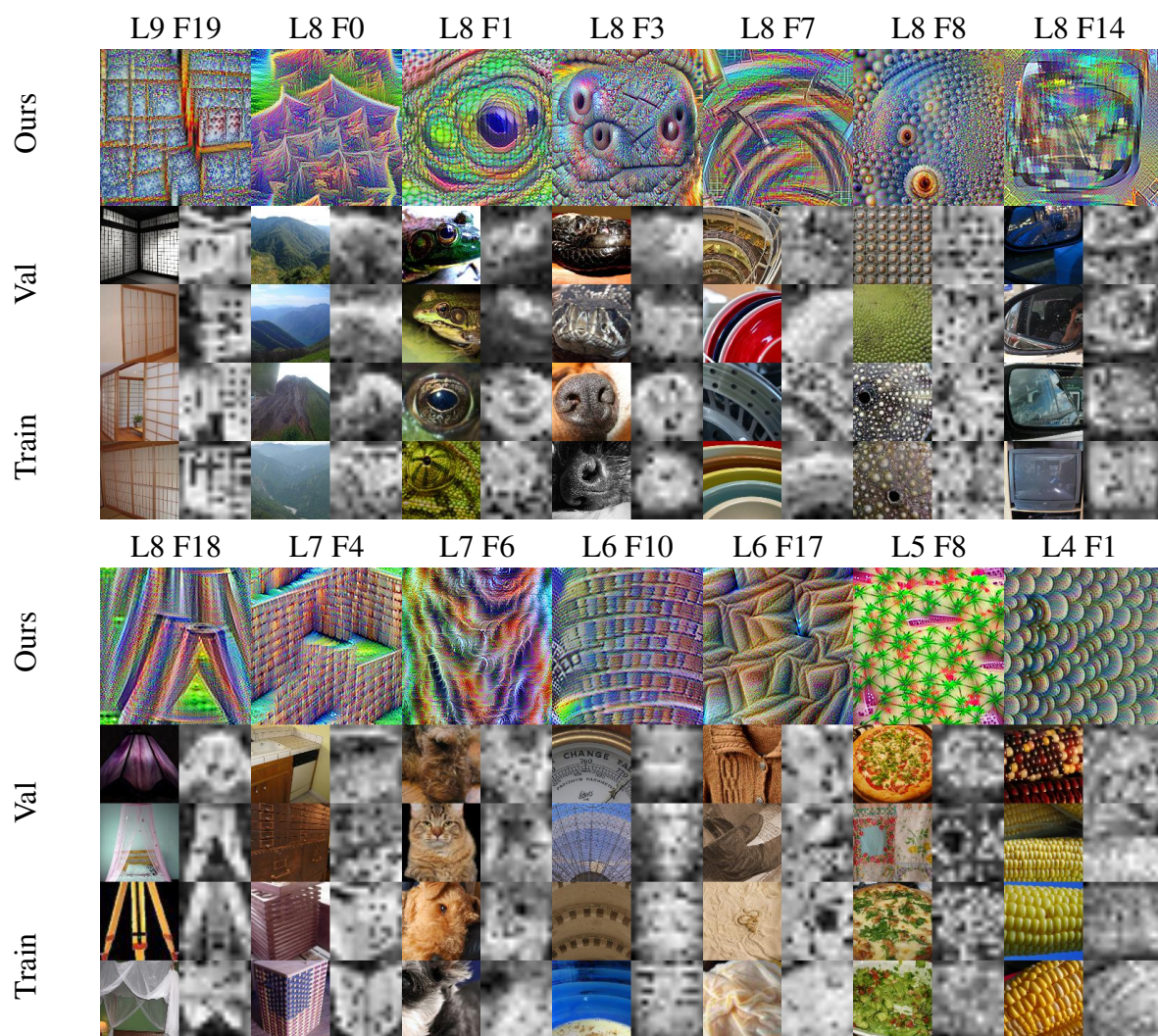


Figure 14.30: (Cont.) Visualization of features in ConViT small.

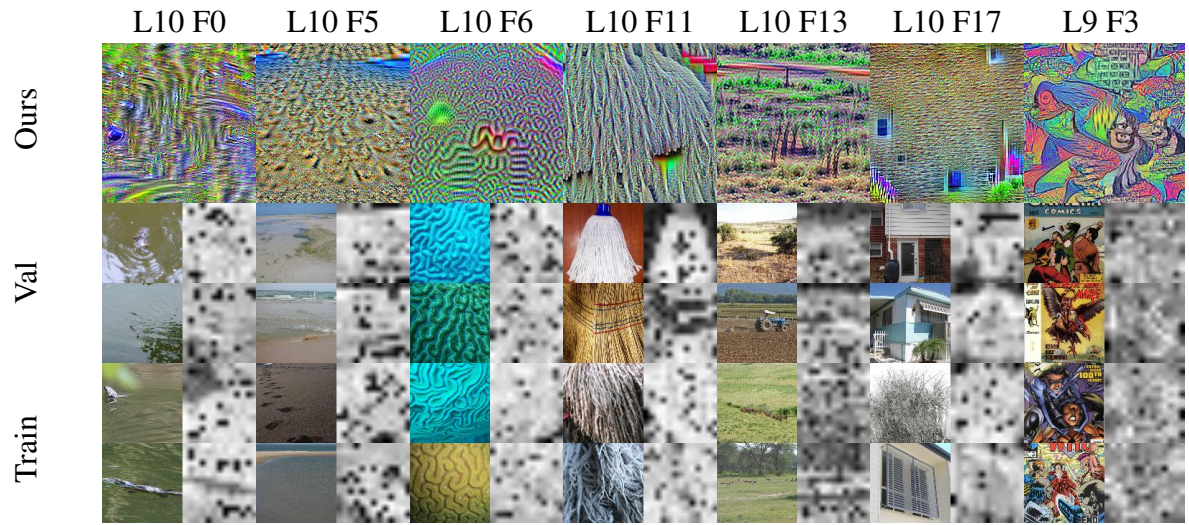


Figure 14.31: Visualization of features in ConViT tiny.

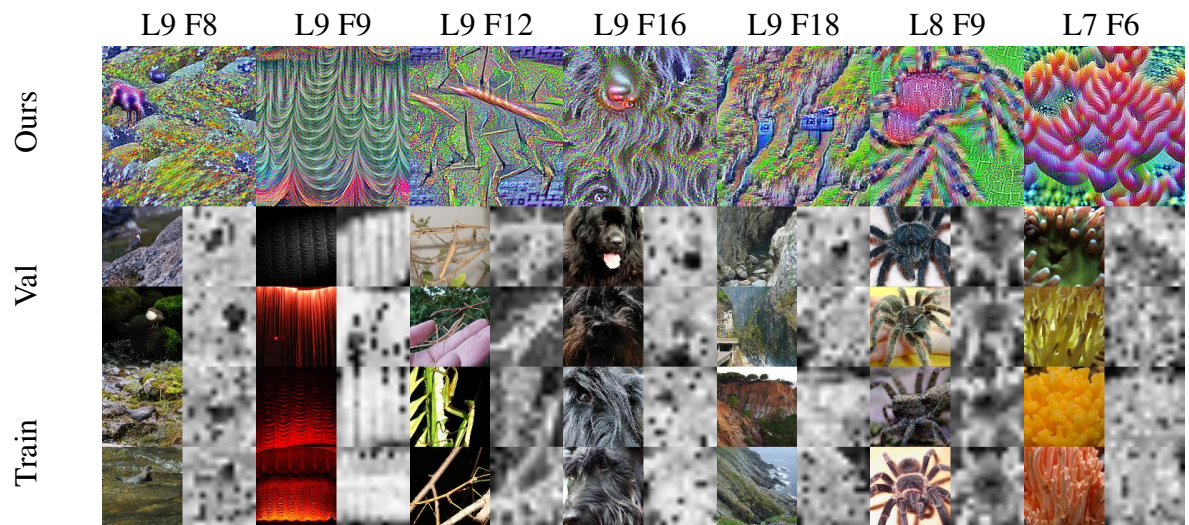


Figure 14.32: (Cont.) Visualization of features in ConViT tiny.

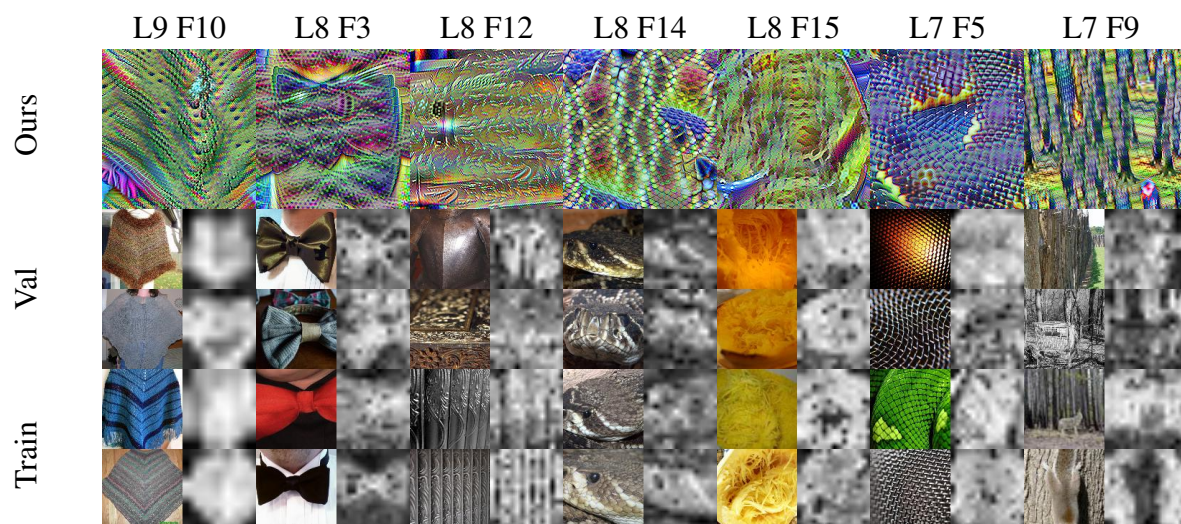


Figure 14.33: Visualization of features in Pit base im-224

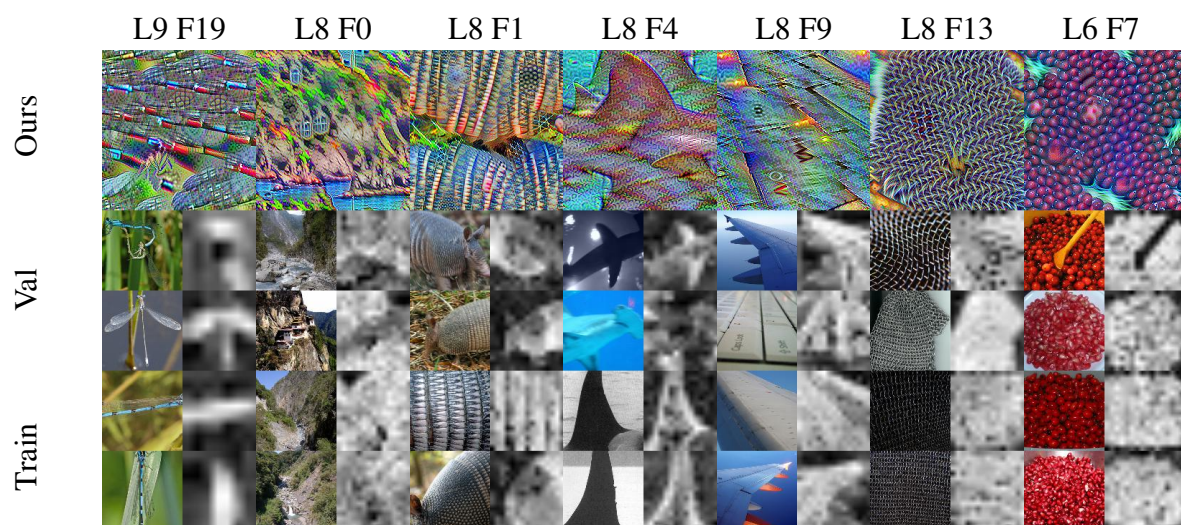


Figure 14.34: Visualization of features in Pit base distilled im-224.

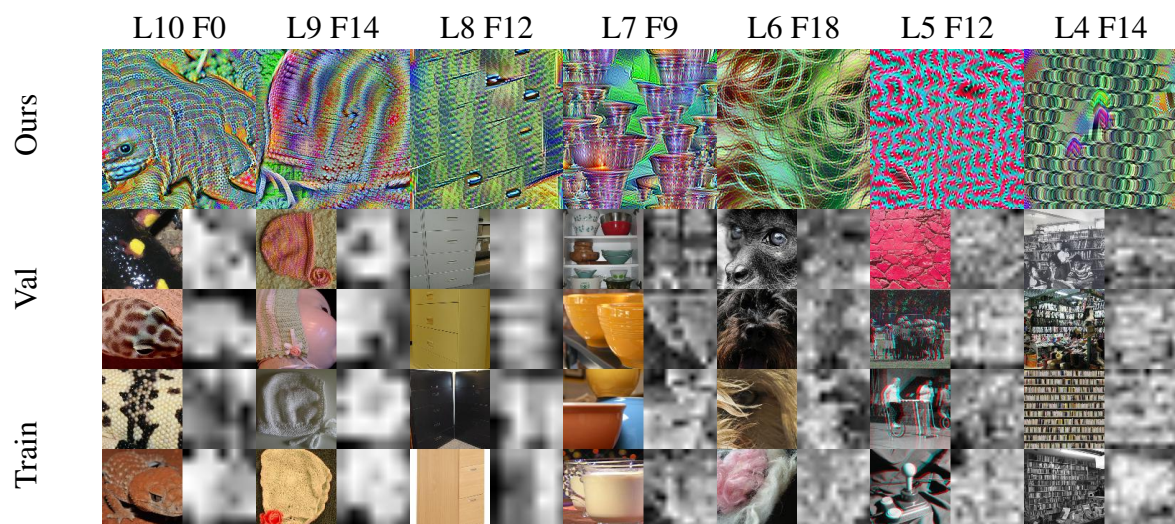


Figure 14.35: Visualization of features in Pit small im-224.

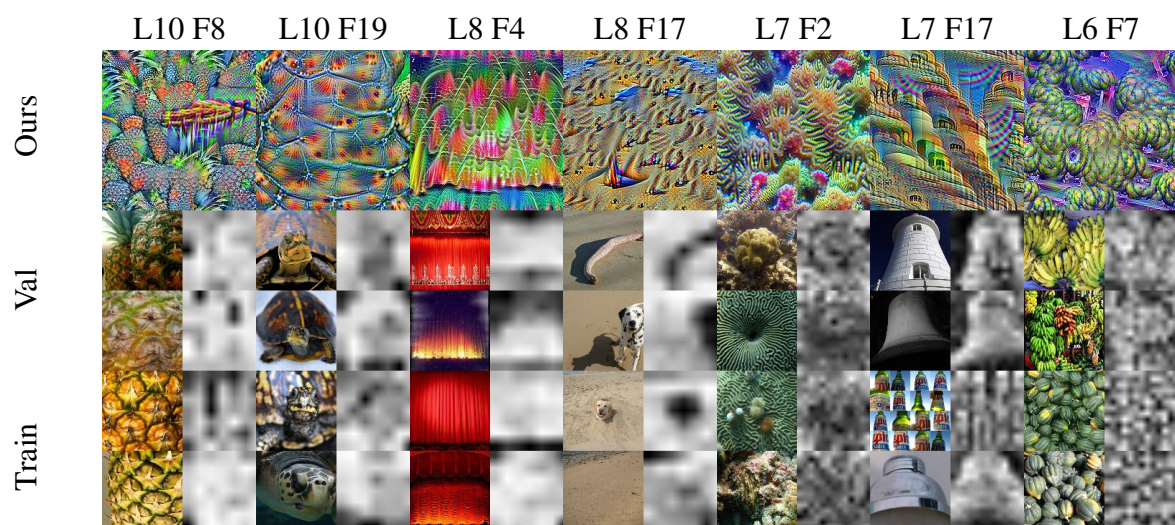


Figure 14.36: Visualization of features in Pit small distilled im-224.

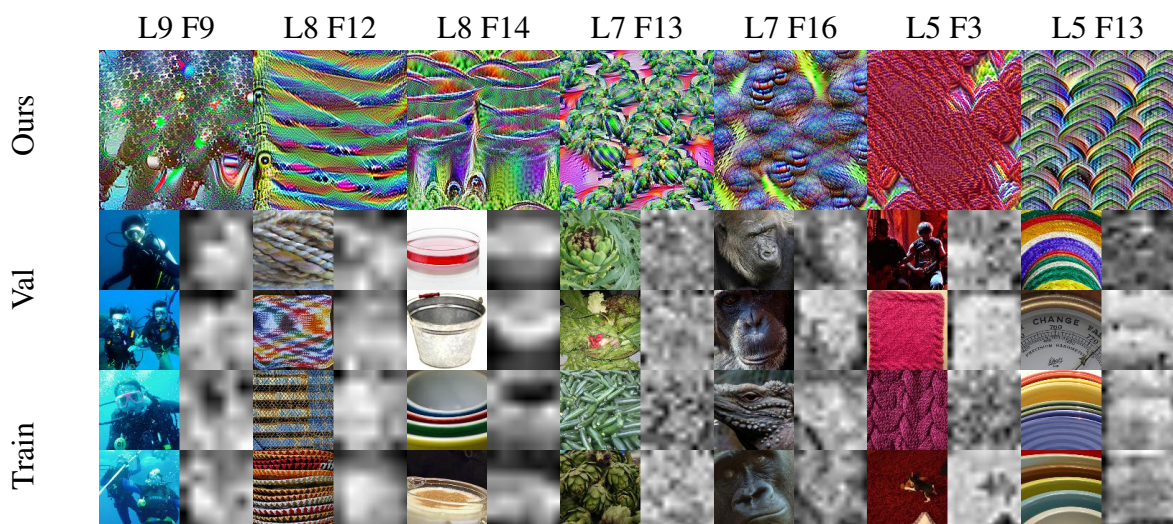


Figure 14.37: Visualization of features in Pit tiny im-224.

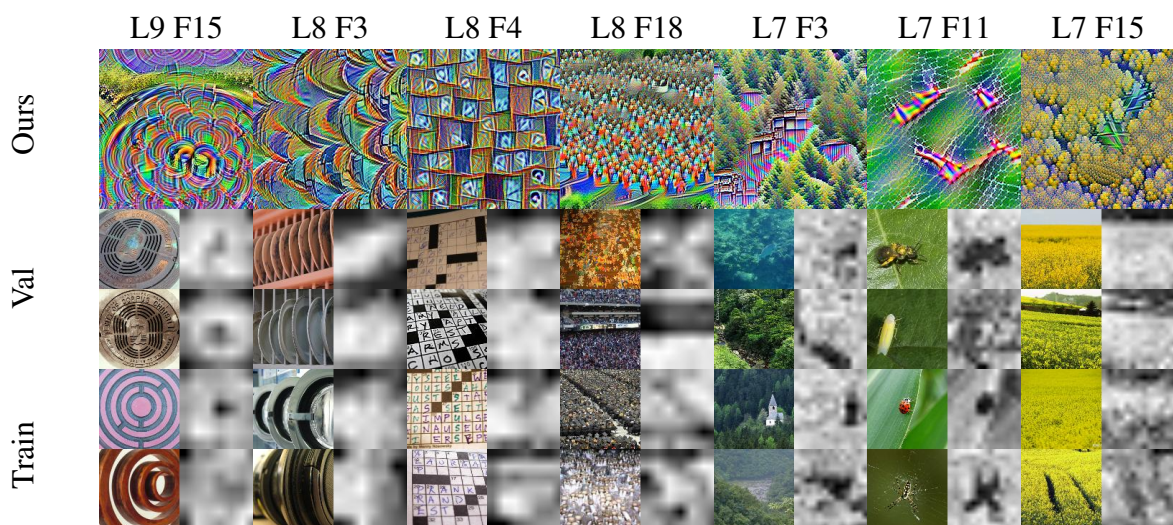


Figure 14.38: Visualization of features in Pit tiny distilled im-224.

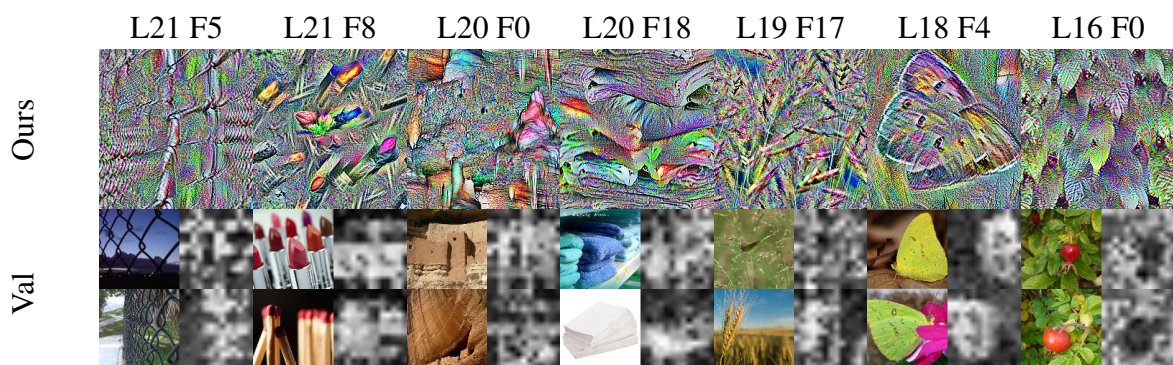


Figure 14.39: Visualization of features in Swin base p-4 w-7 im-224.

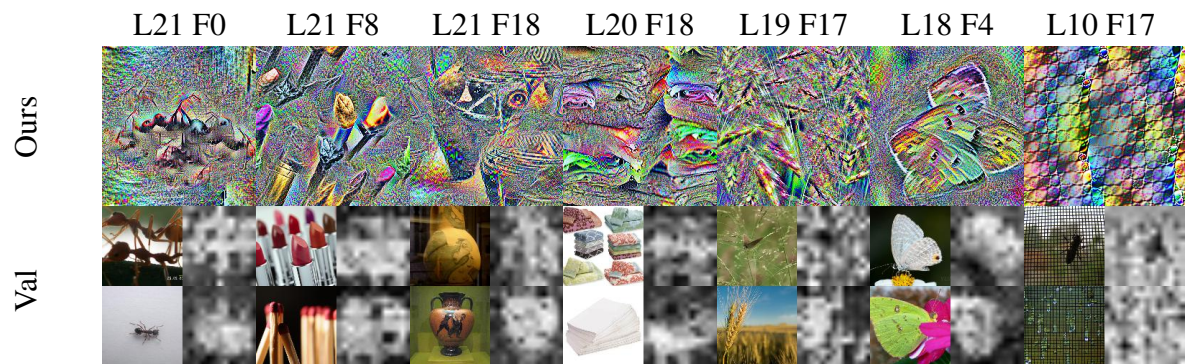


Figure 14.40: Visualization of features in Swing base base p-4 w-7 im-224 imagenet 22k.

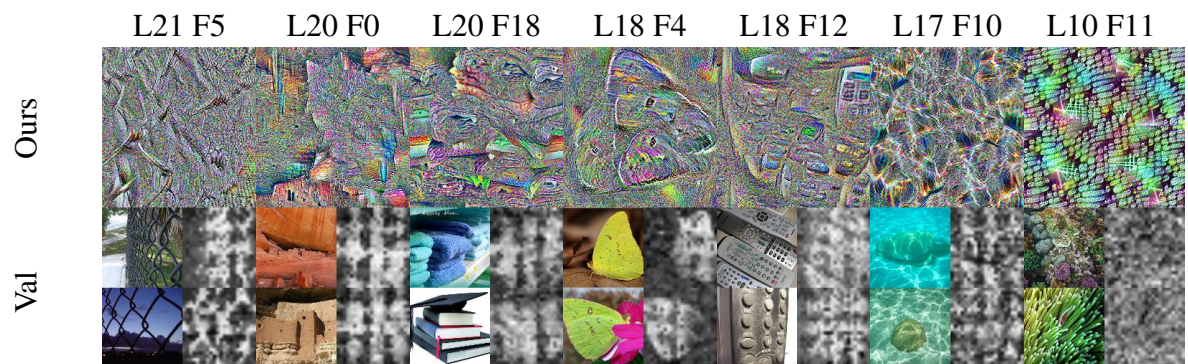


Figure 14.41: Visualization of features in Swin base p-4 w-12 im-384.

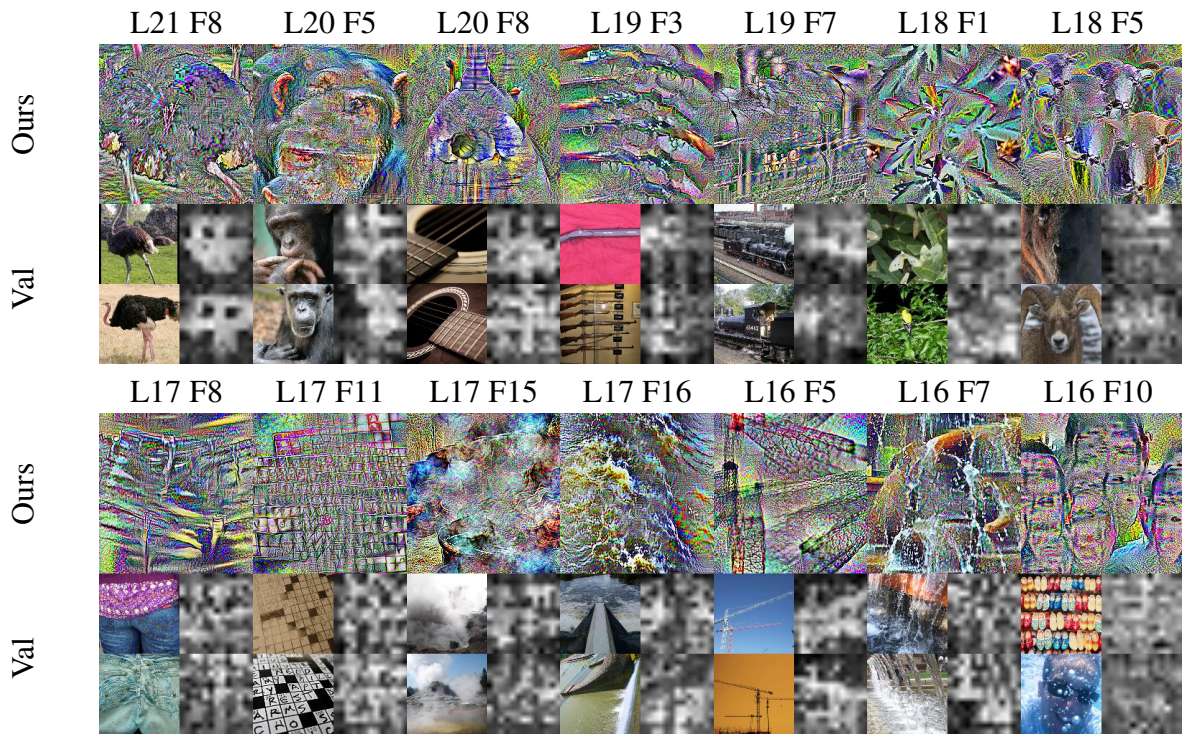


Figure 14.42: Visualization of features in Swin large p-4 w-7 im-224.

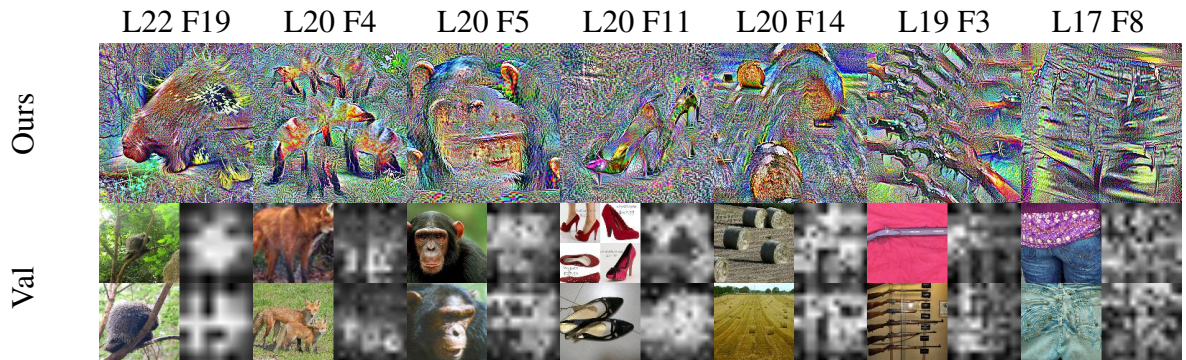


Figure 14.43: Visualization of features in Swin large p-4 w-7 im-224 imagenet 22k.

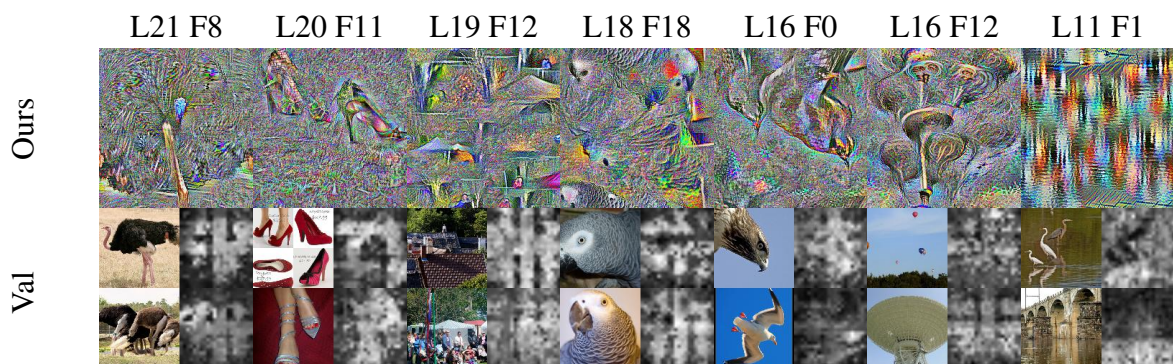


Figure 14.44: Visualization of features in Swin large p-4 w-12 im-384.

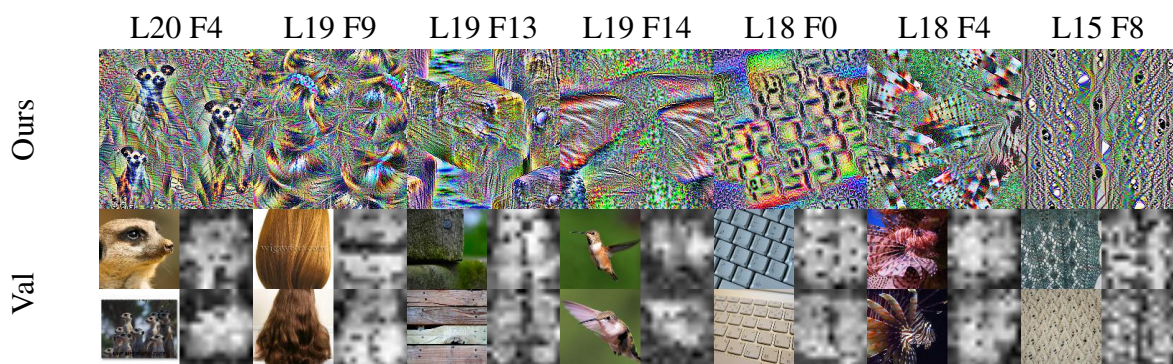


Figure 14.45: Visualization of features in Swin small p-4 w-7 im-224.

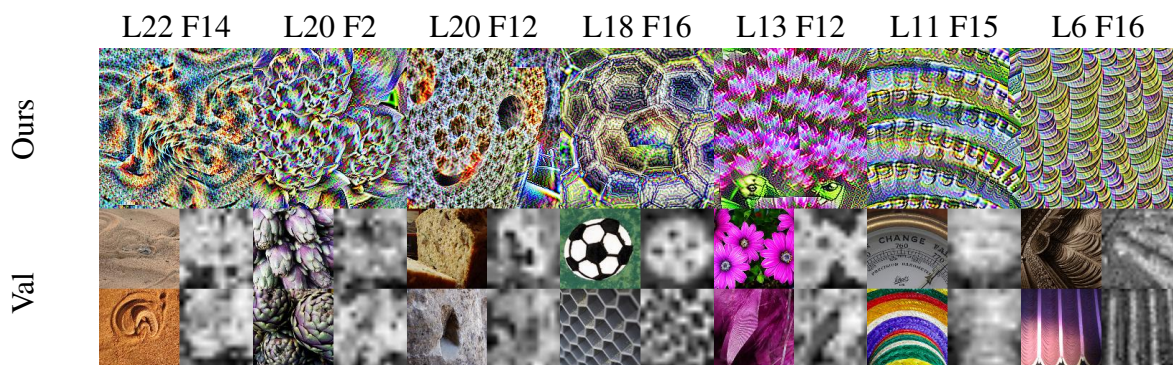


Figure 14.46: Visualization of features in Twins pcvt base.



Figure 14.47: Visualization of features in Twins pcvt large.

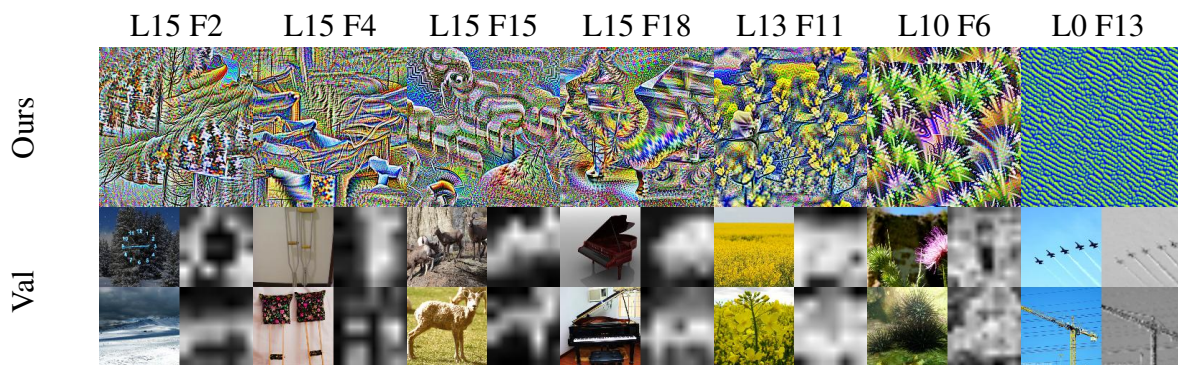


Figure 14.48: Visualization of features in Twins pcvt small.

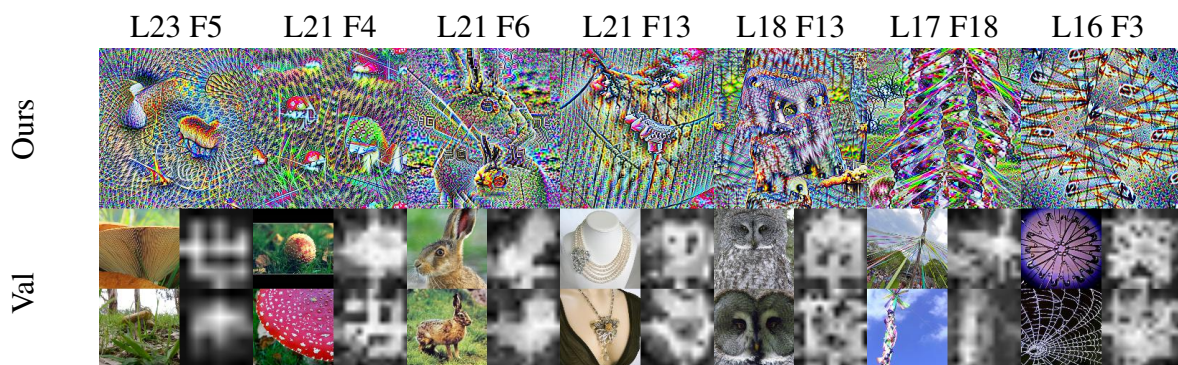


Figure 14.49: Visualization of features in Twins svt base.

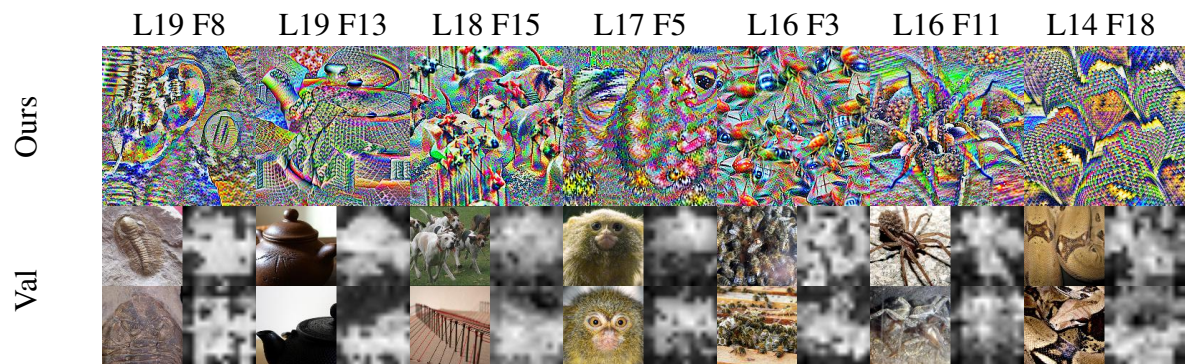


Figure 14.50: Visualization of features in Twins svt large.

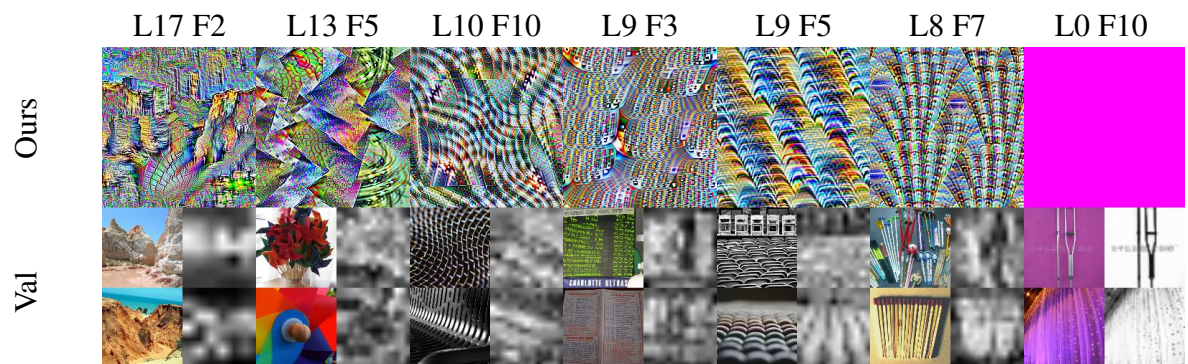


Figure 14.51: Visualization of features in Twins svt small.

Bibliography

- [1] Mohammed Al-Rawi and Dimosthenis Karatzas. On the labeling correctness in computer vision datasets. In IAL@ PKDD/ECML, pages 1–23, 2018.
- [2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In International conference on machine learning, pages 173–182. PMLR, 2016.
- [3] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. In European Conference on Computer Vision, pages 484–501. Springer, 2020.
- [4] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. arXiv preprint arXiv:1802.00420, 2018.
- [5] Arun Babu, Changan Wang, Andros Tjandra, Kushal Lakhota, Qiantong Xu, Naman Goyal, Kritika Singh, Patrick von Platen, Yatharth Saraf, Juan Pino, et al. Xls-r: Self-supervised cross-lingual speech representation learning at scale. arXiv preprint arXiv:2111.09296, 2021.
- [6] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE transactions on pattern analysis and machine intelligence, 39(12):2481–2495, 2017.
- [7] Alexei Baevski, Steffen Schneider, and Michael Auli. vq-wav2vec: Self-supervised learning of discrete speech representations. arXiv preprint arXiv:1910.05453, 2019.
- [8] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. arXiv preprint arXiv:2006.11477, 2020.
- [9] Alexei Baevski, Wei-Ning Hsu, Alexis Conneau, and Michael Auli. Unsupervised speech recognition. arXiv preprint arXiv:2105.11084, 2021.

- [10] Yogesh Balaji, Tom Goldstein, and Judy Hoffman. Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets. arXiv preprint arXiv:1910.08051, 2019.
- [11] Shumeet Baluja and Ian Fischer. Adversarial transformation networks: Learning to generate adversarial examples. AAAI, 2018.
- [12] Mislav Balunovic and Martin Vechev. Adversarial training and provable defenses: Bridging the gap. In International Conference on Learning Representations, 2019.
- [13] Peter L Bartlett, Michael I Jordan, and Jon D McAuliffe. Convexity, classification, and risk bounds. Journal of the American Statistical Association, 101(473):138–156, 2006.
- [14] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In International conference on machine learning, pages 115–123. PMLR, 2013.
- [15] Battista Biggio, Iginio 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, pages 387–402. Springer, 2013.
- [16] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.
- [17] Tom B Brown, Nicholas Carlini, Chiyuan Zhang, Catherine Olsson, Paul Christiano, and Ian Goodfellow. Unrestricted adversarial examples. arXiv preprint arXiv:1809.08352, 2018.
- [18] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. ICLR, 2018.
- [19] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, pages 3–14. ACM, 2017.
- [20] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In 2017 IEEE Symposium on Security and Privacy (SP), pages 39–57. IEEE, 2017.
- [21] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, John C Duchi, and Percy S Liang. Unlabeled data improves adversarial robustness. Advances in Neural Information Processing Systems, 32, 2019.
- [22] Hila Chefer, Shir Gur, and Lior Wolf. Transformer interpretability beyond attention visualization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 782–791, 2021.

- [23] 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. IEEE transactions on pattern analysis and machine intelligence, 40(4):834–848, 2017.
- [24] Liang-Chieh Chen, Maxwell D Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for efficient multi-scale architectures for dense image prediction. arXiv preprint arXiv:1809.04184, 2018.
- [25] Pengguang Chen, Shu Liu, Hengshuang Zhao, and Jiaya Jia. Gridmask data augmentation. arXiv preprint arXiv:2001.04086, 2020.
- [26] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. arXiv preprint arXiv:2003.04297, 2020.
- [27] Ping-yeh Chiang, Renkun Ni, Ahmed Abdelkader, Chen Zhu, Christoph Studer, and Tom Goldstein. Certified defenses for adversarial patches. arXiv preprint arXiv:2003.06693, 2020.
- [28] Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting the design of spatial attention in vision transformers. arXiv preprint arXiv:2104.13840, 1(2):3, 2021.
- [29] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. arXiv preprint arXiv:1902.02918, 2019.
- [30] Alexis Conneau, Alexei Baevski, Ronan Collobert, Abdelrahman Mohamed, and Michael Auli. Unsupervised cross-lingual representation learning for speech recognition. arXiv preprint arXiv:2006.13979, 2020.
- [31] Francesco Croce and Matthias Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack. In International Conference on Machine Learning, pages 2196–2205. PMLR, 2020.
- [32] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In International conference on machine learning, pages 2206–2216. PMLR, 2020.
- [33] Francesco Croce, Maksym Andriushchenko, Vikash Sehwal, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. arXiv preprint arXiv:2010.09670, 2020.
- [34] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 113–123, 2019.

- [35] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops, pages 702–703, 2020.
- [36] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. arXiv preprint arXiv:2106.04803, 2021.
- [37] Stéphane d’Ascoli, Hugo Touvron, Matthew Leavitt, Ari Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. arXiv preprint arXiv:2103.10697, 2021.
- [38] Simon Delecourt and Li Guo. Building a robust mobile payment fraud detection system with adversarial examples. In 2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), pages 103–106. IEEE, 2019.
- [39] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR09, 2009.
- [40] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009.
- [41] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552, 2017.
- [42] Guneet S Dhillon, Kamyar Azizzadenesheli, Zachary C Lipton, Jeremy Bernstein, Jean Kossaifi, Aran Khanna, and Anima Anandkumar. Stochastic activation pruning for robust adversarial defense. arXiv preprint arXiv:1803.01442, 2018.
- [43] Xiaoyi Dong, Jianmin Bao, Ting Zhang, Dongdong Chen, Weiming Zhang, Lu Yuan, Dong Chen, Fang Wen, and Nenghai Yu. Peco: Perceptual codebook for bert pre-training of vision transformers. arXiv preprint arXiv:2111.12710, 2021.
- [44] Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4829–4837, 2016.
- [45] 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. arXiv preprint arXiv:2010.11929, 2020.
- [46] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

- [47] Isaac Dunn, Tom Melham, and Daniel Kroening. Generating realistic unrestricted adversarial inputs using dual-objective gan training. arXiv preprint arXiv:1905.02463, 2019.
- [48] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. arXiv preprint arXiv:1712.02779, 2017.
- [49] Logan Engstrom, Andrew Ilyas, and Anish Athalye. Evaluating and understanding the robustness of adversarial logit pairing. arXiv preprint arXiv:1807.10272, 2018.
- [50] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. University of Montreal, 1341(3):1, 2009.
- [51] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Analysis of classifiers’ robustness to adversarial perturbations. Machine Learning, 107(3):481–508, 2018.
- [52] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, pages 1322–1333, 2015.
- [53] John S Garofolo. Timit acoustic phonetic continuous speech corpus. Linguistic Data Consortium, 1993, 1993. URL <https://catalog.ldc.upenn.edu/LDC93S1>.
- [54] Xavier Gastaldi. Shake-shake regularization. arXiv preprint arXiv:1705.07485, 2017.
- [55] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients—how easy is it to break privacy in federated learning? arXiv preprint arXiv:2003.14053, 2020.
- [56] 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. arXiv preprint arXiv:1811.12231, 2018.
- [57] Amin Ghiasi, Ali Shafahi, and Tom Goldstein. Breaking certified defenses: Semantic adversarial examples with spoofed robustness certificates. arXiv preprint arXiv:2003.08937, 2020.
- [58] Amin Ghiasi, Hamid Kazemi, Ronny Huang, Emily Liu, Micah Goldblum, and Tom Goldstein. Feature sonification: An investigation on the features learned for automatic speech recognition, 2021. URL <https://cs.umd.edu/~amin/apps/visxai/sonification/>.
- [59] Amin Ghiasi, Hamid Kazemi, Steven Reich, Chen Zhu, Micah Goldblum, and Tom Goldstein. Plug-in inversion: Model-agnostic inversion for vision with data augmentations. 2021.

- [60] Amin Ghiasi, Hamid Kazemi, Steven Reich, Chen Zhu, Micah Goldblum, and Tom Goldstein. Plug-in inversion: Model-agnostic inversion for vision with data augmentations. 2021.
- [61] Amin Ghiasi, Ali Shafahi, and Reza Ardekani. Adaptive weight decay: On the fly weight decay tuning for improving robustness. arXiv preprint arXiv:2210.00094, 2022.
- [62] Gabriel Goh, Nick Cammarata †, Chelsea Voss †, Shan Carter, Michael Petrov, Ludwig Schubert, Alec Radford, and Chris Olah. Multimodal neurons in artificial neural networks. Distill, 2021. doi: 10.23915/distill.00030. URL <https://distill.pub/2021/multimodal-neurons/>. <https://distill.pub/2021/multimodal-neurons>.
- [63] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572, 2014.
- [64] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. arXiv preprint arXiv:1810.12715, 2018.
- [65] Sven Gowal, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. Uncovering the limits of adversarial training against norm-bounded adversarial examples. arXiv preprint arXiv:2010.03593, 2020.
- [66] Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. arXiv preprint arXiv:1412.5068, 2014.
- [67] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. arXiv preprint arXiv:2005.08100, 2020. URL <https://arxiv.org/pdf/2005.08100.pdf>.
- [68] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. arXiv preprint arXiv:1711.00117, 2017.
- [69] Wei Han, Zhengdong Zhang, Yu Zhang, Jiahui Yu, Chung-Cheng Chiu, James Qin, Anmol Gulati, Ruoming Pang, and Yonghui Wu. Contextnet: Improving convolutional neural networks for automatic speech recognition with global context. arXiv preprint arXiv:2005.03191, 2020. URL <https://arxiv.org/pdf/2005.03191.pdf>.
- [70] Awni Hannun. Sequence modeling with ctc. Distill, 2017. doi: 10.23915/distill.00008. URL <https://distill.pub/2017/ctc/>. <https://distill.pub/2017/ctc>.
- [71] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567, 2014.

- [72] Jonathan Harrington and Anne Johnstone. The effects of equivalence classes on parsing phonemes into words in continuous speech recognition. Computer Speech & Language, 2(3-4):273–288, 1987. URL <https://www.sciencedirect.com/science/article/abs/pii/088523088790012X>.
- [73] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. Advances in neural information processing systems, 5, 1992.
- [74] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.
- [75] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In Proceedings of the IEEE international conference on computer vision, pages 2961–2969, 2017.
- [76] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. arXiv preprint arXiv:2111.06377, 2021.
- [77] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In NeurIPS, pages 2266–2276, 2017.
- [78] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. In International Conference on Machine Learning, pages 2712–2721. PMLR, 2019.
- [79] Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. arXiv preprint arXiv:1912.02781, 2019.
- [80] Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. Rethinking spatial dimensions of vision transformers. arXiv preprint arXiv:2103.16302, 2021.
- [81] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.
- [82] Hossein Hosseini and Radha Poovendran. Semantic adversarial examples. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 1614–1619, 2018.
- [83] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 1314–1324, 2019.
- [84] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700–4708, 2017.

- [85] 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, pages 7310–7311, 2017.
- [86] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size. arXiv preprint arXiv:1602.07360, 2016.
- [87] Andrew Ilyas, Ajil Jalal, Eirini Asteri, Constantinos Daskalakis, and Alexandros G Dimakis. The robust manifold defense: Adversarial training using generative models. arXiv preprint arXiv:1712.09196, 2017.
- [88] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning, pages 448–456. PMLR, 2015.
- [89] Pavel Izmailov, Dmitrii Podoprikin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. arXiv preprint arXiv:1803.05407, 2018.
- [90] Daniel Jakubovitz and Raja Giryes. Improving dnn robustness to adversarial attacks using jacobian regularization. In ECCV, pages 514–529, 2018.
- [91] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In International conference on machine learning, pages 2304–2313. PMLR, 2018.
- [92] Lu Jiang, Di Huang, Mason Liu, and Weilong Yang. Beyond synthetic noise: Deep learning on controlled noisy labels. In International Conference on Machine Learning, pages 4804–4815. PMLR, 2020.
- [93] Pengchong Jin, Vivek Rathod, and Xiangxin Zhu. Pooling pyramid network for object detection. arXiv preprint arXiv:1807.03284, 2018.
- [94] Nikola Jovanović, Mislav Balunović, Maximilian Baader, and Martin Vechev. Certified defenses: Why tighter relaxations may hurt training? arXiv preprint arXiv:2102.06700, 2021.
- [95] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. Adversarial logit pairing. arXiv preprint arXiv:1803.06373, 2018.
- [96] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.
- [97] Sunil Kumar Kopparapu and M Laxminarayana. Choice of mel filter bank in computing mfcc of a resampled speech. In 10th International Conference on Information Science, Signal Processing and their Applications (ISSPA 2010), pages 121–124. IEEE, 2010. URL <https://arxiv.org/abs/1410.6903>.

- [98] Samuel Krivan, Stanislav Beliaev, Boris Ginsburg, Jocelyn Huang, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, and Yang Zhang. Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions. In ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 6124–6128, 2020. doi: 10.1109/ICASSP40776.2020.9053889. URL <https://ieeexplore.ieee.org/abstract/document/9053889>.
- [99] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
- [100] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). 2010. URL <http://www.cs.toronto.edu/~kriz/cifar.html>.
- [101] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25:1097–1105, 2012.
- [102] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6):84–90, 2017.
- [103] Anders Krogh and John Hertz. A simple weight decay can improve generalization. Advances in neural information processing systems, 4, 1991.
- [104] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. ICLR, 2016.
- [105] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533, 2016.
- [106] Koichi Kuriyama. Autocleansing: Unbiased estimation of deep learning with mislabeled data. 2020.
- [107] Jungmin Kwon, Jeongseop Kim, Hyunseo Park, and In Kwon Choi. Asam: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. In International Conference on Machine Learning, pages 5905–5914. PMLR, 2021.
- [108] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL <http://yann.lecun.com/exdb/mnist/>.
- [109] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL <http://yann.lecun.com/exdb/mnist/>.
- [110] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4):541–551, 1989.
- [111] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. Advances in neural information processing systems, 2, 1989.

- [112] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.
- [113] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. arXiv preprint arXiv:1802.03471, 2018.
- [114] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In 2019 IEEE Symposium on Security and Privacy (SP), pages 656–672. IEEE, 2019.
- [115] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Second-order adversarial attack and certifiable robustness. CoRR, abs/1809.03113, 2018. URL <http://arxiv.org/abs/1809.03113>.
- [116] Chung-Yi Li, Pei-Chieh Yuan, and Hung-Yi Lee. What does a network layer hear? analyzing hidden representations of end-to-end asr through speech synthesis. In ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 6434–6438. IEEE, 2020. URL <https://arxiv.org/pdf/1911.01102.pdf>.
- [117] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In NeurIPS, pages 6389–6399, 2018.
- [118] Jason Li, Vitaly Lavrukhin, Boris Ginsburg, Ryan Leary, Oleksii Kuchaiev, Jonathan M Cohen, Huyen Nguyen, and Ravi Teja Gadde. Jasper: An end-to-end convolutional neural acoustic model. arXiv preprint arXiv:1904.03288, 2019. URL <https://arxiv.org/pdf/1904.03288.pdf>.
- [119] Zhizhong Li and Derek Hoiem. Learning without forgetting. IEEE transactions on pattern analysis and machine intelligence, 40(12):2935–2947, 2018.
- [120] Fangzhou Liao, Ming Liang, Yinpeng Dong, Tianyu Pang, Xiaolin Hu, and Jun Zhu. Defense against adversarial attacks using high-level representation guided denoiser. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1778–1787, 2018.
- [121] Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. ICLR, 2019.
- [122] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In Proceedings of the IEEE international conference on computer vision, pages 2980–2988, 2017.
- [123] Hanxiao Liu, Zihang Dai, David R So, and Quoc V Le. Pay attention to mlps. arXiv preprint arXiv:2105.08050, 2021.

- [124] Mason Liu, Menglong Zhu, Marie White, Yinxiao Li, and Dmitry Kalenichenko. Looking fast and slow: Memory-guided mobile video object detection. arXiv preprint arXiv:1903.10172, 2019.
- [125] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. arXiv preprint arXiv:2103.14030, 2021.
- [126] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101, 2017.
- [127] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In Proceedings of the European conference on computer vision (ECCV), pages 116–131, 2018.
- [128] Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E Houle, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. arXiv preprint arXiv:1801.02613, 2018.
- [129] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083, 2017.
- [130] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 5188–5196, 2015.
- [131] Felipe A Mejia, Paul Gamble, Zigfried Hampel-Arias, Michael Lomnitz, Nina Lopatina, Lucas Tindall, and Maria Alejandra Barrios. Robust or private? adversarial training makes models more vulnerable to privacy attacks. arXiv preprint arXiv:1906.06449, 2019.
- [132] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. In Workshop on automatic machine learning, pages 58–65. PMLR, 2016.
- [133] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 135–147. ACM, 2017.
- [134] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In 2016 fourth international conference on 3D vision (3DV), pages 565–571. IEEE, 2016.
- [135] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In International Conference on Machine Learning, pages 3575–3583, 2018.

- [136] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2574–2582, 2016.
- [137] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. 2015.
- [138] Marius Mosbach, Maksym Andriushchenko, Thomas Trost, Matthias Hein, and Dietrich Klakow. Logit pairing methods can fool gradient-based attacks. arXiv preprint arXiv:1810.12042, 2018.
- [139] Nicolas M Müller and Karla Markert. Identifying mislabeled instances in classification datasets. In 2019 International Joint Conference on Neural Networks (IJCNN), pages 1–8. IEEE, 2019.
- [140] Samuel G Müller and Frank Hutter. Trivialaugment: Tuning-free yet state-of-the-art data augmentation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 774–782, 2021.
- [141] Kensuke Nakamura and Byung-Woo Hong. Adaptive weight decay for deep neural networks. IEEE Access, 7:118857–118865, 2019.
- [142] Muzammal Naseer, Kanchana Ranasinghe, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Intriguing properties of vision transformers. arXiv preprint arXiv:2105.10497, 2021.
- [143] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 427–436, 2015.
- [144] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. Distill, 2017. doi: 10.23915/distill.00007. <https://distill.pub/2017/feature-visualization>.
- [145] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. Journal of artificial intelligence research, 11:169–198, 1999.
- [146] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10):1345–1359, 2009.
- [147] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In 2016 IEEE European symposium on security and privacy (EuroS&P), pages 372–387. IEEE, 2016.
- [148] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In 2016 IEEE symposium on security and privacy (SP), pages 582–597. IEEE, 2016.
- [149] Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le. Specaugment: A simple data augmentation method for automatic speech recognition. arXiv preprint arXiv:1904.08779, 2019.

- [150] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32:8026–8037, 2019.
- [151] Sayak Paul and Pin-Yu Chen. Vision transformers are robust learners. arXiv preprint arXiv:2105.07581, 2021.
- [152] David C Plaut et al. Experiments on learning by back propagation. 1986.
- [153] Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge Belongie. Generative adversarial perturbations. CVPR, 2018.
- [154] Haifeng Qian and Mark N Wegman. L2-nonexpansive neural networks. arXiv preprint arXiv:1802.07896, 2018.
- [155] 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. arXiv preprint arXiv:2103.00020, 2021.
- [156] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? Advances in Neural Information Processing Systems, 34, 2021.
- [157] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. arXiv preprint arXiv:1801.09344, 2018.
- [158] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In NeurIPS, pages 10877–10887, 2018.
- [159] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 3505–3506, 2020.
- [160] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. Neural computation, 29(9):2352–2449, 2017.
- [161] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Aging evolution for image classifier architecture search. In AAAI conference on artificial intelligence, volume 2, page 2, 2019.
- [162] Sylvestre-Alvise Rebuffi, Sven Gowal, Dan A Calian, Florian Stimberg, Olivia Wiles, and Timothy Mann. Fixing data augmentation to improve adversarial robustness. arXiv preprint arXiv:2103.01946, 2021.
- [163] Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In International Conference on Machine Learning, pages 8093–8104. PMLR, 2020.

- [164] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention, pages 234–241. Springer, 2015.
- [165] Andrew Slavin Ross and Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In AAAI, 2018.
- [166] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. IJCV, 115(3):211–252, 2015.
- [167] Parsa Saadatpanah, Ali Shafahi, and Tom Goldstein. Adversarial attacks on copyright detection systems. In International Conference on Machine Learning, pages 8307–8315. PMLR, 2020.
- [168] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. Advances in neural information processing systems, 29:2234–2242, 2016.
- [169] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4510–4520, 2018.
- [170] Shibani Santurkar, Andrew Ilyas, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Image synthesis with a single (robust) classifier. Advances in Neural Information Processing Systems, 32:1262–1273, 2019.
- [171] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. In Advances in Neural Information Processing Systems, pages 5014–5026, 2018.
- [172] Ali Shafahi, Mahyar Najibi, Zheng Xu, John Dickerson, Larry S Davis, and Tom Goldstein. Universal adversarial training. arXiv preprint arXiv:1811.11304, 2018.
- [173] Ali Shafahi, W Ronny Huang, Christoph Studer, Soheil Feizi, and Tom Goldstein. Are adversarial examples inevitable? ICLR, 2019.
- [174] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! Advances in Neural Information Processing Systems, 32, 2019.
- [175] Ali Shafahi, Parsa Saadatpanah, Chen Zhu, Amin Ghiasi, Christoph Studer, David Jacobs, and Tom Goldstein. Adversarially robust transfer learning. arXiv preprint arXiv:1905.08232, 2019.
- [176] Rulin Shao, Zhouxing Shi, Jinfeng Yi, Pin-Yu Chen, and Cho-Jui Hsieh. On the adversarial robustness of visual transformers. arXiv preprint arXiv:2103.15670, 2021.

- [177] Mahmood Sharif, Lujo Bauer, and Michael K Reiter. On the suitability of lp-norms for creating and preventing adversarial examples. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 1605–1613, 2018.
- [178] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
- [179] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In In Workshop at International Conference on Learning Representations, 2014.
- [180] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.
- [181] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. arXiv preprint arXiv:1706.03825, 2017.
- [182] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. arXiv preprint arXiv:2201.11990, 2022.
- [183] Hwanjun Song, Minseok Kim, and Jae-Gil Lee. Selfie: Refurbishing unclear samples for robust deep learning. In International Conference on Machine Learning, pages 5907–5915. PMLR, 2019.
- [184] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. arXiv preprint arXiv:1710.10766, 2017.
- [185] Yang Song, Rui Shu, Nate Kushman, and Stefano Ermon. Constructing unrestricted adversarial examples with generative models. In Advances in Neural Information Processing Systems, pages 8312–8323, 2018.
- [186] Hossein Souri, Micah Goldblum, Liam Fowl, Rama Chellappa, and Tom Goldstein. Sleeper agent: Scalable hidden trigger backdoors for neural networks trained from scratch. arXiv preprint arXiv:2106.08970, 2021.
- [187] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1):1929–1958, 2014.
- [188] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. In Proceedings of the 31st International Conference on Neural Information Processing Systems, pages 3520–3532, 2017.
- [189] 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 Proceedings of the European Conference on Computer Vision (ECCV), pages 631–648, 2018.

- [190] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. 2013.
- [191] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199, 2013.
- [192] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1–9, 2015.
- [193] Towaki Takikawa, David Acuna, Varun Jampani, and Sanja Fidler. Gated-scnn: Gated shape cnns for semantic segmentation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 5229–5238, 2019.
- [194] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In International conference on machine learning, pages 6105–6114. PMLR, 2019.
- [195] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2820–2828, 2019.
- [196] Shixin Tian, Guolei Yang, and Ying Cai. Detecting adversarial examples through image transformation. In Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
- [197] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. Mlp-mixer: An all-mlp architecture for vision. arXiv preprint arXiv:2105.01601, 2021.
- [198] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Gautier Izacard, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. Resmlp: Feedforward networks for image classification with data-efficient training, 2021.
- [199] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In International Conference on Machine Learning, volume 139, pages 10347–10357, July 2021.
- [200] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In International Conference on Machine Learning, pages 10347–10357. PMLR, 2021.

- [201] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. [arXiv preprint arXiv:1705.07204](#), 2017.
- [202] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. [ICLR](#), 1050:11, 2018.
- [203] Jonathan Uesato, Brendan O’Donoghue, Aaron van den Oord, and Pushmeet Kohli. Adversarial risk and the dangers of evaluating against weak attacks. [arXiv preprint arXiv:1802.05666](#), 2018.
- [204] 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](#), pages 5998–6008, 2017.
- [205] Changhan Wang, Yun Tang, Xutai Ma, Anne Wu, Dmytro Okhonko, and Juan Pino. Fairseq s2t: Fast speech-to-text modeling with fairseq. [arXiv preprint arXiv:2010.05171](#), 2020.
- [206] Shiqi Wang, Yizheng Chen, Ahmed Abdou, and Suman Jana. Mixtrain: Scalable training of formally robust neural networks. [arXiv preprint arXiv:1811.02625](#), 2018.
- [207] David Warde-Farley and Ian Goodfellow. 11 adversarial perturbations of deep neural networks. [Perturbations, Optimization, and Statistics](#), 311:5, 2016.
- [208] Ross Wightman. Pytorch image models. <https://github.com/rwightman/pytorch-image-models>, 2019.
- [209] Rey Wiyatno and Anqi Xu. Maximal jacobian-based saliency map attack. [arXiv preprint arXiv:1808.07945](#), 2018.
- [210] Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. [ICML](#), 2017.
- [211] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In [International Conference on Machine Learning](#), pages 5286–5295. PMLR, 2018.
- [212] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. In [Advances in Neural Information Processing Systems](#), pages 8400–8409, 2018.
- [213] Eric Wong, Leslie Rice, and J Zico Kolter. Fast is better than free: Revisiting adversarial training. [arXiv preprint arXiv:2001.03994](#), 2020.
- [214] Huikai Wu, Junge Zhang, Kaiqi Huang, Kongming Liang, and Yizhou Yu. Fastfcn: Rethinking dilated convolution in the backbone for semantic segmentation. [arXiv preprint arXiv:1903.11816](#), 2019.

- [215] 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, pages 1–17. Springer, 2020.
- [216] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. IJCAI, 2018.
- [217] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747, 2017.
- [218] Kai Y Xiao, Vincent Tjeng, Nur Muhammad Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing relu stability. arXiv preprint arXiv:1809.03008, 2018.
- [219] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. arXiv preprint arXiv:1711.01991, 2017.
- [220] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. CVPR, 2019.
- [221] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1492–1500, 2017.
- [222] Qiantong Xu, Alexei Baevski, Tatiana Likhomanenko, Paden Tomasello, Alexis Conneau, Ronan Collobert, Gabriel Synnaeve, and Michael Auli. Self-training and pre-training are complementary for speech recognition. In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 3030–3034. IEEE, 2021.
- [223] Weijian Xu, Yifan Xu, Tyler Chang, and Zhuowen Tu. Co-scale conv-attentional image transformers. arXiv preprint arXiv:2104.06399, 2021.
- [224] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. arXiv preprint arXiv:1704.01155, 2017.
- [225] Yoshihiro Yamada, Masakazu Iwamura, Takuya Akiba, and Koichi Kise. Shakedown regularization for deep residual learning. IEEE Access, 7:186126–186136, 2019.
- [226] Hongxu Yin, Pavlo Molchanov, Jose M Alvarez, Zhizhong Li, Arun Mallya, Derek Hoiem, Niraj K Jha, and Jan Kautz. Dreaming to distill: Data-free knowledge transfer via deepinversion. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8715–8724, 2020.
- [227] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Advances in neural information processing systems, pages 3320–3328, 2014.

- [228] Fuxun Yu, Chenchen Liu, Yanzhi Wang, and Xiang Chen. Interpreting adversarial robustness: A view from decision surface in input space. arXiv preprint arXiv:1810.00144, 2018.
- [229] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In Proceedings of the IEEE/CVF international conference on computer vision, pages 6023–6032, 2019.
- [230] Sangdoo Yun, Seong Joon Oh, Byeongho Heo, Dongyoon Han, Junsuk Choe, and Sanghyuk Chun. Re-labeling imagenet: from single to multi-labels, from global to localized labels. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2340–2350, 2021.
- [231] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. arXiv preprint arXiv:1605.07146, 2016.
- [232] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In European conference on computer vision, pages 818–833. Springer, 2014.
- [233] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. arXiv preprint arXiv:2106.04560, 2021.
- [234] Chiyan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. Communications of the ACM, 64(3):107–115, 2021.
- [235] Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization. arXiv preprint arXiv:1810.12281, 2018.
- [236] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. Theoretically principled trade-off between robustness and accuracy. arXiv preprint arXiv:1901.08573, 2019.
- [237] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412, 2017.
- [238] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. arXiv preprint arXiv:1906.06316, 2019.
- [239] Xinbin Zhang. A method of data label checking and the wrong labels in mnist and cifar10. Available at SSRN 3072167, 2017.
- [240] Tianhang Zheng, Di Wang, Baochun Li, and Jinhui Xu. A unified framework for randomized smoothing based certified defenses. 2019.
- [241] Yanqi Zhou, Siavash Ebrahimi, Serkan Ö Arık, Haonan Yu, Hairong Liu, and Greg Diamos. Resource-efficient neural architect. arXiv preprint arXiv:1806.07912, 2018.

- [242] Ligeng Zhu and Song Han. Deep leakage from gradients. In Federated learning, pages 17–31. Springer, 2020.
- [243] Roland Zimmermann, Judy Borowski, Robert Geirhos, Matthias Bethge, Thomas Wallis, and Wieland Brendel. How well do feature visualizations support causal understanding of cnn activations? Advances in Neural Information Processing Systems, 34, 2021.
- [244] Chandler Zuo. Regularization effect of fast gradient sign method and its generalization. arXiv preprint arXiv:1810.11711, 2018.