

ABSTRACT

Title of dissertation: Adversarial Vulnerabilities of Deep Networks and
Affordable Robustness

Ali Shafahi
Doctor of Philosophy, 2020

Dissertation directed by: Professor Tom Goldstein
Computer Science Department

Deep learning has improved the performance of many computer vision tasks. However, the features that are learned without extra regularization are not necessarily interpretable. While in terms of generalization, conventionally trained models seem to perform really well, they are susceptible to certain failure modes. Eventhough these catastrophic failure cases rarely happen naturally, an adversary can engineer them by having some knowledge about the design process. Based on the time that the adversary manipulates the system, we can classify threats into evasion attacks or data poisoning attacks.

First, we will cover a recently proposed data poisoning threat model that does not assume that the adversary has control over the labeling process. We call this attack the “targeted clean-label” poisoning attack. The proposed attack successfully causes misclassification of a target instance both under end-to-end training and transfer learning scenarios without degrading the overall performance of the classifier on non-target examples.

We will then shift our focus to evasion attacks. We will consider two types of

inference-time attacks: universal perturbations, and adversarial examples. For universal perturbations, we present an efficient method for perturbation generation. We also propose universal adversarial training for defending against universal perturbations.

In the last part of this dissertation, we will present methods for training per-instance robust models in settings where we have limited resources. One case of limited resources is the scarcity of computing power. In this case, we will present our algorithm called “Adversarial Training for Free!” which enables us to train robust models with the same computational cost of conventional/natural training. We achieve this efficiency by simultaneously updating the network parameters and the adversarial perturbation. Another case of limited resources is availability of training data per-class. For this case, we introduce adversarially robust transfer learning.

Adversarial Vulnerabilities of Deep Networks and Affordable Robustness

by

Ali Shafahi

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

Advisory Committee:

Professor John P. Dickerson

Professor Tudor Dumitras

Professor Tom Goldstein, Chair/Advisor

Professor Furong Huang

Professor Gavin Taylor

© Copyright by
Ali Shafahi
2020

Dedication

To my lovely parents, Raheleh and Yousef, and to my brother, Mohammad.

Acknowledgments

First and foremost, I am greatly indebted to my advisor, Prof. Tom Goldstein who has guided me throughout my research with his invaluable knowledge and feedback. He always had a pool of interesting ideas to pursue but still patiently allowed me to follow my interests, to freely fail time to time, and then helped me to learn from my failures. With Tom, I really felt that I have a great co-author who contributes a lot in all parts of the research. He also made me feel welcome to express my concerns and supported me a lot. He has been always there to advise me even during the chaotic moments just before the deadlines. I always felt to have his full support and have learned a lot from him.

I had the privilege to work with many bright and energetic student researchers throughout my Ph.D. I would like to express my strong gratitude to all my collaborators and co-authors who influenced my works with their diverse backgrounds and knowledge: Mahyar Najibi who taught me the very basics and also the very advanced concepts – has always helped me by sharing his invaluable knowledge and providing great suggestions, Ronny Huang who made my transition to ML a lot easier through our routine chats, Zheng Xu who taught me how a true scholar should be, Amin Ghiasi who was extremely helpful in coming up with interesting ideas and improving my CS knowledge, and Chen Zhu who always fascinated me with his extreme drive.

I would also like to extend my thanks to my committee members Prof. Tudor Dumitras, Prof. Furong Huang, Prof. John Dickerson, and Prof. Gavin Taylor for all the insights they shared with me in our research collaborations and their comments and suggestions, without which this dissertation and my status overall would not be in its

current shape.

I would also greatly thank all my old and new friends for their support during this journey. I would like to thank Kiana Roshan Zamir for helping me a lot during the busy and stressful times and giving valuable recommendations. I would also like to thank all my other fabulous friends Mahyar Najibi, Saeed Seddigin, Sina Dehghani, Amin Ghiasi, Saba Ahmadi, Mahsa Derakhshan, Soheil Behnezhad, Hadi Yami, Alireza Farhadi, Parsa Saadatpanah, Sepehr Ghader, and Hamed Saleh who helped my quick adaptation to this new field. I really look forward to our continued friendship.

Last but most importantly, I dedicate this dissertation to my family. To my lovely parents Raheleh and Yousef, without whom this journey was not even possible. My brother, Mohammad, who always backed me up unconditionally. I was greatly blessed to have them by my side through all these years and cannot find words adequate enough to truly thank them.

Table of Contents

Dedication	ii
Acknowledgements	iii
Table of Contents	v
List of Tables	viii
List of Figures	x
1 Introduction	1
1.1 Training-time Attacks	3
1.2 Inference-time Attacks	4
1.2.1 Adversarial Examples	4
1.2.2 Universal perturbations	5
1.3 Outline	6
2 Targeted Clean-Label Poisoning Attacks on Neural Networks	7
2.1 Introduction	7
2.1.1 Related work	8
2.1.2 Contributions	10
2.2 A simple clean-label attack	11
2.2.1 Crafting poison data via feature collisions	12
2.2.2 Optimization procedure	13
2.3 Poisoning attacks on transfer learning	14
2.3.1 A one-shot kill attack	14
2.4 Poisoning attacks on end-to-end training	18
2.4.1 Single poison instance attack	19
2.4.2 Watermarking: a method to boost the power of poison attacks	20
2.4.2.1 Multiple poison instance attacks	21
2.5 Conclusion	23
3 Universal Adversarial Training	27
3.1 Introduction	27
3.2 Related work	29
3.3 Optimization for universal perturbation	32

3.4	Universal adversarial training	36
3.4.1	Attacking hardened models	39
3.4.2	Low-cost universal adversarial training	41
3.5	Universal perturbations for ImageNet	41
3.5.1	Benefits of the proposed method	43
3.5.2	The effect of clipping	44
3.5.3	How much training data does the attack need?	44
3.6	Universal adversarial training on ImageNet	46
3.7	Conclusion	48
4	Adversarial Training for Free!	49
4.1	Introduction	49
4.2	Non-targeted adversarial examples	51
4.3	Adversarial training	53
4.4	“Free” adversarial training	54
4.4.1	The effect of mini-batch replay on natural training	55
4.5	Robust models on CIFAR-10 and 100	57
4.5.1	CIFAR-10	57
4.5.2	CIFAR-100	59
4.6	Does “free” training behave like standard adversarial training?	60
4.6.0.1	Generative behavior for largely perturbed examples	60
4.6.0.2	Smooth and flattened loss surface	61
4.7	Robust ImageNet classifiers	61
4.7.1	Free training results	63
4.7.2	Comparison with PGD-trained models	65
4.7.3	Free training on models with more capacity	65
4.8	Conclusions	66
5	Adversarially Robust Transfer Learning	67
5.1	Introduction	67
5.1.1	Contributions	68
5.2	Background	69
5.3	The robustness of deep features	71
5.4	Transfer learning: Recycling feature extractors	73
5.4.1	Transfer Learning with ImageNet models	75
5.4.1.1	Transfer learning using ImageNet	75
5.4.2	Low-data regime	77
5.4.2.1	Low-data regime transfer learning from ImageNet to CIFAR-10	78
5.4.2.2	Training deeper networks on top of robust feature extractors	78
5.5	Analysis: Robust feature extractors are filters	82
5.6	End-to-end training without forgetting	84
5.6.1	The distance between feature representations of natural images and augmented images	87

5.6.2	Decreasing generalization gap of adversarially trained networks .	88
5.6.3	Improving generalization of the CIFAR-10 adversarially trained model	89
5.6.4	LwF-based robust transfer learning for similar source and target datasets	89
5.7	Experiment details	91
5.7.1	LWF-based experiments	91
5.7.2	ImageNet to CIFAR experiments	91
5.7.3	Free training experiments	91
5.8	Conclusion	92
6	Conclusions and Future Works	93
	Bibliography	97

List of Tables

3.1	Validation accuracy of hardened WideResnet models trained on CIFAR-10. Note that Madry’s PGD training is significantly slower than the other training methods.	40
3.2	Top-1 accuracy on ImageNet for natural images, and adversarial images with universal perturbation.	44
3.3	Accuracy on ImageNet for nat and robust models.	47
4.1	Validation accuracy and robustness of CIFAR-10 models trained with various methods.	58
4.2	Validation accuracy and robustness of CIFAR-100 models trained with various methods.	58
4.3	ImageNet validation accuracy and robustness of ResNet-50 models trained with various replay parameters and $\epsilon = 2$	63
4.4	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.	65
4.5	Validation accuracy and robustness of free- $m = 4$ trained ResNets with various capacities.	66
5.1	Accuracy and robustness of natural and adversarially trained models on CIFAR-10+ and CIFAR-100+. The “+” sign denotes standard data augmentation ($\epsilon = 8$).	71
5.2	Transfer learning by freezing the feature extractor layers ($\epsilon = 8$).	75
5.3	Transfer learning from ImageNet ($\epsilon = 5$).	76
5.4	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.	87
5.5	Decreasing generalization gap by transferring with LwF. For reference, last row shows results from adversarial training CIFAR-100. The ‘+’ sign refers to using augmentation. ($\epsilon = 8$)	89
5.6	Decreasing generalization gap by transferring with LwF. For reference, last row shows results from adversarial training CIFAR-10. The ‘+’ sign refers to using augmentation. ($\epsilon = 8$)	90

5.7	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.	90
-----	--	----

List of Figures

1.1	Adversarial examples with different norm constraints formed via the projected gradient method [1] on Resnet50, along with the distance between the base image and the adversarial example, and the top class label. A variety of norms can be used to measure the size of adversarial perturbations. This figure is adapted from [2].	2
2.1	Transfer learning poisoning attack. (a) The top row contains 5 random target instances (from the “fish” class). The second row contains the constructed poison instance corresponding to each of these targets. We used the same base instance (second row, leftmost image) for building each poison instance. The attack is effective for any base, but fewer iterations are required if the base image has a higher resolution. We stopped the poison generation algorithm when the maximum iterations was met or when the feature representation of the target and poison instances were less than 3 units apart (in Euclidean norm). The stopping threshold of 3 was determined by the minimum distance between all pairs of training points. As can be seen, the poison instances are visually indistinguishable from the base instance (and one another). Rows 3 and 4 show samples from similar experiments where the target (fish) and base (dog) classes were swapped.	16
2.2	Angular deviation of the feature space decision boundary when trained with clean dataset + poison instance(s) versus when trained with clean dataset alone. (a) Histogram of the final (last epoch) angular deviation over all experiments. In transfer learning (blue), there is a significant rotation (average of 23 degrees) in the feature space decision boundary. In contrast, in end-to-end training (red) where we inject 50 poison instances, the decision boundary’s rotation is negligible. (b) Most of the parameter adjustment is done during the first epoch. For the end-to-end training experiments, the decision boundary barely changes.	18

2.3	Feature space visualization of end-to-end training poisoning attacks. (a) A single poison instance is unable to successfully attack the classifier. The poison instance's feature space position under the clean model is overlapped with that of the target instance. However, when the model is trained on the clean + poisoned data (i.e. the poisoned model), the feature space position of the poison instance is returned to the base class distribution, while target remains in the target class distribution. (b) To make the attack successful, we construct 50 poison instances from 50 random base instances that are "watermarked" with a 30% opacity target instance. This causes the target instance to be pulled out of the target class distribution (in feature space) into the base class distribution and get incorrectly classified as the base class.	25
2.4	12 out of 60 random poison instances that successfully cause a bird target instance to get misclassified as a dog in the end-to-end training scenario. An adversarial watermark (opacity 30%) of the target bird instance is applied to the base instances when making the poisons. More examples are in the supplementary material.	25
2.5	Success rates for attacks on outliers and random targets. While attacking non-outlier is still possible, attacking an outlier can increase the chances of success.	26
3.1	A universal perturbation made using a subset of ImageNet and the VGG-16 architecture. When added to the validation images, their labels usually change. The perturbation was generated using the proposed algorithm 3. Perturbation pixel values lie in $[-10, 10]$ (i.e. $\epsilon = 10$).	28
3.2	Classification accuracy on adversarial examples of universal perturbations generated by increasing the cross-entropy loss. PGD and ADAM converge faster. We use 5000 training samples from CIFAR-10 for constructing the universal adversarial perturbation for naturally trained Wide ResNet model from [1]. The batch-size is 128, $\epsilon=8$, and the learning-rate/step-size is 1.	35
3.3	Visualizations of universal perturbations after 160 iterations of the optimizers depicted in fig. 3.2.	36
3.4	The universal perturbations made for 4 different robust models trained on CIFAR-10: adversarially trained with FGSM or PGD, and universally adversarially trained with FGSM (uFGSM) or SGD (uSGD). Perturbations were made using 400 iterations.	39
3.5	Attack performance varies with clipping parameter β in eq. (3.2). Attacking Inception-V3 (with natural validation accuracy 78%) is more successful with clipping ($\beta = 9$) than without clipping ($\beta = \infty$).	45
3.6	The attack performance significantly improves when the number of training points is larger than the number of classes. For reference, Inception-V3's top-1 accuracy is 78%. Using only a small fraction of the training-data (4,000 / 1,281,167) is enough to degrade the validation and train accuracy to around 20%.	45

3.7	Training universal perturbation can fool naturally trained AlexNet on ImageNet, but fails to fool our robust AlexNets. We smoothed the curves in (a) for better visualization. The universal perturbations generated for the universal adversarial trained AlexNets on ImageNet have little geometric structure compared to that of the naturally trained network. (b) Universal perturbation of natural model. The accuracy of the validation images + noise is only 3.9% (c) Perturbation for our universally trained model using algorithm 4. The accuracy of the validation images + noise for our robust model is 42.0% . (d) Perturbation for the model trained with our low-cost universal training variant (algorithm 5). The accuracy of the validation images + noise is 28.3% . While the universal noise for the low-cost variant of universal adversarial training has some structure compared to the original variant, when compared to that of the natural model (b), it is structure-less.	46
4.1	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.	56
4.2	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.	60
4.3	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.	62
4.4	The effect of the perturbation bound ϵ and the mini-batch replay hyperparameter m on the robustness achieved by free training.	64
5.1	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>i.e.</i> robustness) after re-training deep layers. The robustness of the adversarially trained models if all layers are frozen are shown with dashed lines.	72
5.2	Wide Resnet 32-10 and the blocks used for freezing/retraining	73

5.3	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.	79
5.4	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.	80
5.5	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 5.4.1.1). 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>i.e.</i> training accuracy becomes 100%). We can slightly improve generalization using batch norm (BN) and dropout (DO).	82
5.6	Gradients of the loss w.r.t to input images for the CIFAR-100 transfer learning experiments of sections 5.4.1.1 & 5.4.2.2. 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.	84
5.7	Our LwF loss has a term that enforces the similarity of feature representations (<i>i.e.</i> penultimate layer activations) between the source model and the fine-tuned model.	85
5.8	Figures 5.8a and 5.8b 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.	88

Chapter 1: Introduction

Deep Neural Networks (DNNs) have been greatly popularized due to their performance on many benchmark tasks. DNNs have achieved state-of-the-art performance on many computer vision related tasks such as classification, localization, and segmentation. The DNNs used in computer vision tasks are often Convolutional Neural Networks (CNNs). CNNs are very good at extracting features for visual data where locality has an inherent meaning.

While CNNs are very good models when one looks at various evaluation metrics such as accuracy, they come with their downsides. One important downside is the lack of interpretability. In efforts to make sense of the CNNs predictions, many studies have tried to visualize their learned features. These feature visualization works often require additional regularizations such as the Total Variation penalty. Without these regularizers, the visualized features of conventionally trained models are not interpretable due to high-frequency artifacts.

CNNs, similar to most machine learning models, generalize very well to unseen data (test data) that are sampled from a similar distribution of data used for training them. If there is a distribution mismatch between the train and test data, conventionally trained CNNs perform poorly. For example, a detector trained on images from day time doesn't

perform well on images taken during night time. Cases like this can be expected by humans since visually a distribution shift can be observed. However, there are cases where a human cannot explain why a CNN is misclassifying something. Adversarial examples are among the most difficult to explain.

Given a natural example, an adversarial example is an example which looks very similar (i.e. quasi-identical) to the natural example but alters the classifiers prediction. Similarity between the natural example and its adversarial counterpart can be measured using many different metrics. The most common metrics are ℓ_p -norms which result in ℓ_p -bounded adversarial examples. Different p -norms result in different adversarial examples (See Fig. 1.1.) An ℓ_0 -bounded adversarial example limits the number of pixels which can be perturbed (sparsity). An ℓ_2 -bounded adversarial example limits the euclidean norm of the adversarial noise. And, an ℓ_∞ -bounded example, limits the intensity of changing any pixel.

Many cases of adversarial examples fall within the additive noise case where the adversarial example can be built by adding an adversarial noise to the clean example

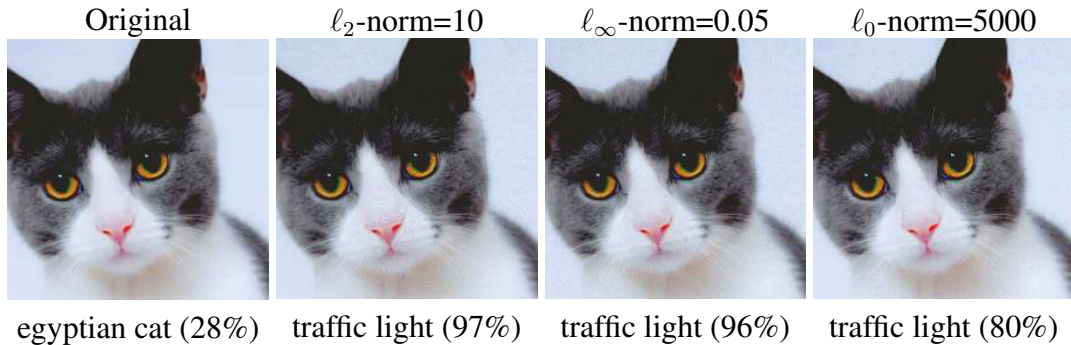


Figure 1.1: Adversarial examples with different norm constraints formed via the projected gradient method [1] on Resnet50, along with the distance between the base image and the adversarial example, and the top class label. A variety of norms can be used to measure the size of adversarial perturbations. This figure is adapted from [2].

(i.e., $x_{adv} = x_{nat} + \delta_{adv}$.) The δ_{adv} can be instance specific or instance independent. A universal perturbation is a case where δ_{adv} causes misclassification for many different unseen examples as opposed to the per-instance perturbation which is specially crafted for an individual target.

Manipulating a target test image (using a per-instance adversarial examples and universal perturbations) is not the only failure mode for CNNs. Another alternative failure mode is data poisoning which aims at adding maliciously perturbed instances to the training data. A poisoned dataset makes the network behave in a certain way chosen by the adversary at test time. As just mentioned, based on the time the adversary manipulates the data, attacks can be classified into *training-time* or *inference-time*.

1.1 Training-time Attacks

In supervised learning tasks such as classification, CNNs are trained on labeled data. In many realistic settings, the data is not synthetic and is often collected by scrapping the web. This allows an adversary to affect the data which will be used for training classifiers and to effectively perform data poisoning.

Unlike inference-time attacks, poisoning attacks often do not modify the target image at test time¹ but aim to poison the training dataset and cause the models to fail during test time. The failure in poisoning could be on an entire class as assumed by prior poisoning literature [3] or on particular examples [4, 5].

¹Some classes of poisoning attacks such as backdoor attacks do modify the target image at test time.

1.2 Inference-time Attacks

Per-instance adversarial examples and universal perturbations fall within the inference-time attacks category since they cause misclassification of a certain target by adding a quasi-imperceptible perturbation to the target at test time.

1.2.1 Adversarial Examples

One popular way to generate a per-instance attack (i.e., adversarial example) is to solve the following optimization problem:

$$\max_{\delta} J_{\theta}(x + \delta_{adv}, y) \quad \text{s.t.} \quad \|\delta_{adv}\|_p \leq \epsilon \quad (1.1)$$

where J in multi-class classification problems is often cross-entropy, θ is the frozen network parameters, and (x,y) is the clean example and its ground-truth label. This optimization problem can be solved using various methods which vary largely in terms of their computation costs. The most popular, and one of the most powerful methods is Projected Gradient Descent (PGD). In PGD, the perturbation is often updated in multiple (K) iterations.

One popular way to defend against adversarial examples is to augment the training dataset and include adversarial examples built on the fly. This method is known as adversarial training. It is often argued that adversarial training with weaker adversaries result in models which cannot defend against stronger adversaries. Consequently, to gain the strongest hardened model, adversarial training is often performed using multi-step PGD

methods. This results in the training time for robust training to be K -times more than that of conventional training.

K -PGD training is both costly *and* data-hungry compared to conventional training. Adversarial training with many popular datasets does not result in great generalization of robustness². In other words, after adversarial training, the robustness on the training examples is very high (often close to 100%) but the robustness on validation examples is considerably lower. The generalization gap does decrease with the addition of more data (both labeled and un-labeled.)

These two requirements for adversarial training (data and computation), leaves room for efficiency gains.

1.2.2 Universal perturbations

Unlike adversarial examples, universal perturbations are image-agnostic. They can be built using a subset of images and applied to a different subset which makes them applicable for situations where the attacker has last-minute access to test images or does not have the computational power to generate per-instance attacks at inference time.

Another characteristic of universal perturbations is their cross-model transferability. For example, a universal perturbation built for a ResNet model often transfers to a VGG model. The generation of universal perturbations using the convention approach in [6] has been seen as a cumbersome and time-consuming task and has caused researchers to use heuristics to build new but weaker universal perturbations from a small set of universal adversarial perturbations [7].

²The exceptions are toy datasets such as MNIST.

From a defense perspective, due to the expensive cost associated with generation of universal perturbations, adversarial training on universal perturbations is often seen as computationally impractical. Consequently, the few studies which propose defenses against universal perturbations, often use other methods such as training a noise removal head for hardening the classifier.

The computation cost associated with universal perturbation generation, and also adversarial training leaves room for computational efficiency.

1.3 Outline

The rest of this dissertation is organized as follows and is structured in a way that each chapter can be seen as a standalone read. Chapter 2 studies the problem of clean-label poisoning attacks which degrade the performance of a classifier on a target instance without tampering with the target and the poison label. Chapter 3 focuses on the problem of cheap methods to generate universal perturbations and affordable methods to train CNNs to be robust against these universal perturbations. Chapters 4 and 5 discuss the problem of training classifiers which can resist per-instance adversarial examples in settings where we have either limited compute power or limited number of training data per-class. Finally, chapter 6 concludes the dissertation.

Chapter 2: Targeted Clean-Label Poisoning Attacks on Neural Networks

2.1 Introduction

Before deep learning algorithms can be deployed in high stakes, security-critical applications, their robustness against adversarial attacks must be put to the test. The existence of adversarial examples in deep neural networks (DNNs) has triggered debates on how secure these classifiers are [8, 9, 10]. Adversarial examples fall within a category of attacks called *evasion attacks*. Evasion attacks happen at test time – a clean target instance is modified to avoid detection by a classifier, or spur misclassification. However, these attacks do not map to certain realistic scenarios in which the attacker cannot control test time data. For example, consider a retailer aiming to mark a competitor’s email as spam through an ML-based spam filter. Evasion attacks are not applicable because the attacker cannot modify the victim emails. Similarly, an adversary may not be able to alter the input to a face recognition engine that operates under supervised conditions, such as a staffed security desk or building entrance. Such systems are still susceptible to *data poisoning* attacks. These attacks happen at training time; they aim to manipulate the performance of a system by inserting carefully constructed *poison instances* into the training data.

This chapter studies poisoning attacks on neural nets that are *targeted*, meaning

they aim to control the behavior of a classifier on one specific test instance. For example, they manipulate a face recognition engine to change the identity of one specific person, or manipulate a spam filter to allow/deny a specific email of the attacker’s choosing. We propose *clean label* attacks that do not require control over the labeling function; the poisoned training data appear to be labeled correctly according to an expert observer. This makes the attacks not only difficult to detect, but opens the door for attackers to succeed without any inside access to the data collection/labeling process. For example, an adversary could place poisoned images online and wait for them to be scraped by a bot that collects data from the web. The retailer described above could contribute to a spam filter dataset simply by emailing people inside an organization.

2.1.1 Related work

Classical poisoning attacks indiscriminately degrade test accuracy rather than targeting specific examples, making them easy to detect. While there are studies related to poisoning attacks on support vector machines [3] or Bayesian classifiers [11], poisoning attacks on Deep Neural Networks (*DNN*) have been rarely studied. In the few existing studies, DNNs have been shown to fail catastrophically against data poisoning attacks. Steinhardt et al. reported that, even under strong defenses, there is an 11% reduction in test accuracy when the attacker is allowed 3% training set modifications [12]. Some studies propose a back-gradient based approach for generating poisons [13]. To speed up the process of generating poisoning instances, generators have been developed that produce poisons [14].

A more dangerous approach is for the attacker to target specific test instances. For example, the retailer mentioned above, besides achieving her target goal, does not want to render the spam filter useless or tip off the victim to the presence of her attack. Targeted backdoor attacks [15] with few resources (~ 50 training examples) have been recently shown to cause the classifier to fail for special test examples. For example some networks are trained using mislabeled images tagged with a special pattern, causing the classifier to learn the association between the pattern and the class label [16]. In [17] a network is trained to respond to a trojan trigger.

These attacks present the same shortcomings as evasion attacks; they require test-time instances to be modified to trigger the mispredictions. Moreover, in most prior work, the attacker is assumed to have some degree of control over the labeling process for instances in the training set. This inadvertently excludes real-world scenarios where the training set is audited by human reviewers who will label each example as it appears to the eye, or where the labels are assigned by an external process (such as malware detectors which often collect ground truth labeled by third party antiviruses). Assumed control over the labeling function leads to a straightforward one-shot attack wherein the target instance with a flipped label is added as poison. Overfitting on the poison would then ensure that the target instance would get misclassified during inference time. The most closely related work to our own is [18], which studies targeted attacks on neural nets. This attack, however, requires that poisons fill at least 12.5% (and up to 100%) of *every* minibatch, which may be unrealistic in practice. In contrast, our attacks do not require any control of the minibatching process, and assume a much smaller poisoning budget ($\leq 0.1\%$ vs. $\geq 12.5\%$).

Finally, we note that several works have approached poisoning from a theoretical perspective. [19, 20] studies poisoning threat models from a theoretical perspective, and the robustness of classifiers to training data perturbations was considered in [21].

2.1.2 Contributions

In this work, we study a new type of attack, henceforth called *clean-label* attacks, wherein the attacker’s injected training examples are cleanly labeled by a certified authority, as opposed to maliciously labeled by the attacker herself. Our strategy assumes that the attacker has no knowledge of the training data but has knowledge of the model and its parameters. This is a reasonable assumption given that many classic networks pre-trained on standard datasets, such as ResNet [22] or Inception [23] trained on ImageNet, are frequently used as feature extractors. The attacker’s goal is to cause the retrained network to misclassify a special test instance from one class (e.g. a piece of malware) as another class of her choice (e.g. benign application) after the network has been retrained on the augmented data set that includes poison instances. Besides the intended misprediction on the target, the performance degradation on the victim classifier is not noticeable. This makes state-of-the-art poisoning defenses that measure the performance impact of training instances (such as [24]) ineffective.

A similar type of attack was demonstrated using influence functions [4] for the scenario where only the final fully connected layer of the network was retrained on the poisoned dataset, with a success rate of 57%. We demonstrate an optimization-based clean-label attack under the *transfer learning* scenario studied in [4], but we achieve 100%

attack success rate on the same dog-vs-fish classification task. Further, we study – for the first time to our knowledge – clean-label poisoning in the *end-to-end training* scenario where all layers of the network are retrained. Through visualizations, we shed light on why this scenario is much more difficult due to the expressivity of deep networks. Informed by these visualizations, we craft a 50 poison instance attack on a deep network which achieves success rates of up to 60% in the end-to-end training scenario.

2.2 A simple clean-label attack

We now propose an optimization-based procedure for crafting poison instances that, when added to the training data, manipulate the test-time behavior of a classifier. Later, we'll discuss tricks to boost the power of this simple attack.

An attacker first chooses a *target instance* from the test set; a successful poisoning attack causes this target example to be misclassified during test time. Next, the attacker samples a *base instance* from the base class, and makes imperceptible changes to it to craft a *poison instance*; this poison is injected into the training data with the intent of fooling the model into labelling the target instance with the base label at test time. Finally, the model is trained on the poisoned dataset (clean dataset + poison instances). If, at test time, the model mistakes the target instance as being in the base class, then the poisoning attack is considered successful.

2.2.1 Crafting poison data via feature collisions

Let $f(\mathbf{x})$ denote the function that propagates an input $\mathbf{x}$ through the network to the penultimate layer (before the softmax layer). We call the activations of this layer the *feature space* representation of the input since it encodes high-level semantic features. Due to the high complexity and nonlinearity of f , it is possible to find an example $\mathbf{x}$ that “collides” with the target in feature space, while simultaneously being close to the base instance $\mathbf{b}$ in input space by computing

$$\mathbf{p} = \underset{\mathbf{x}}{\operatorname{argmin}} \quad \|f(\mathbf{x}) - f(\mathbf{t})\|_2^2 + \beta \|\mathbf{x} - \mathbf{b}\|_2^2 \quad (2.1)$$

The right-most term of Eq. 2.1 causes the poison instance $\mathbf{p}$ to appear like a base class instance to a human labeler (β parameterizes the degree to which this is so) and hence be labeled as such. Meanwhile, the first term of Eq. 2.1 causes the poison instance to move toward the target instance in feature space and get embedded in the target class distribution. On a clean model, this poison instance would be misclassified as a target. If the model is retrained on the clean data + poison instances, however, the linear decision boundary in feature space is expected to rotate to label the poison instance as if it were in the base class. Since the target instance is nearby, the decision boundary rotation may inadvertently include the target instance in the base class along with the poison instance (note that training strives for correct classification of the poison instance but not the target since the latter is not part of the training set). This allows the unperturbed target instance, which is subsequently misclassified into the base class during test time, to gain

a “backdoor” into the base class.

2.2.2 Optimization procedure

Our procedure for performing the optimization in Eq. 2.1 to obtain $\mathbf{p}$ is shown in Algorithm 1. The algorithm uses a forward-backward-splitting iterative procedure [25]. The first (forward) step is simply a gradient descent update to minimize the L2 distance to the target instance in feature space. The second (backward step) is a proximal update that minimizes the Frobenius distance from the base instance in input space. The coefficient β is tuned to make the poison instance look realistic in input space, enough to fool an unsuspecting human observer into thinking the attack vector image has not been tampered with.

Algorithm 1: Poisoning Example Generation

```

1 Input: target instance  $t$ , base instance  $b$ , learning rate  $\lambda$ 

2 Initialize  $x$ :  $x_0 \leftarrow b$ 

3 Define:  $L_p(x) = \|f(\mathbf{x}) - f(\mathbf{t})\|^2$ 

4 for  $i = 1$  to  $maxIters$  do
5     Forward step:  $\hat{x}_i = x_{i-1} - \lambda \nabla_x L_p(x_{i-1})$ 
6     Backward step:  $x_i = (\hat{x}_i + \lambda \beta b) / (1 + \beta \lambda)$ 
7 end
```

2.3 Poisoning attacks on transfer learning

We begin by examining the case of transfer learning, in which a pre-trained feature extraction network is used, and only the final network (softmax) layer is trained to adapt the network to a specific task. This procedure is common in industry where we want to train a robust classifier on limited data. Poisoning attacks in this case are extremely effective. In Section 2.4, we generalize these attacks to the case of end-to-end training.

We perform two poisoning experiments. First, we attack a pretrained InceptionV3 [26] network under the scenario where the weights of all layers excluding the last are frozen. Our network and dataset (ImageNet [27] dog-vs-fish) were identical to that of [4]. Second, we attack an AlexNet architecture modified for the CIFAR-10 dataset by [28] under the scenario where all layers are trained.¹

2.3.1 A one-shot kill attack

We now present a simple poisoning attack on transfer learned networks. In this case, a “one-shot kill” attack is possible; by adding just one poison instance to the training set (that is labeled by a reliable expert), we cause misclassification of the target with 100% success rate. Like in [4], we essentially leverage InceptionV3 as a feature extractor and retrain its final fully-connected layer weights to classify between dogs and fish. We select 900 instances from each class in ImageNet as the training data and remove duplicates from the test data that are present in the training data as a pre-processing step². After

¹The code is available at <https://github.com/ashafahi/inceptionv3-transferLearn-poison>

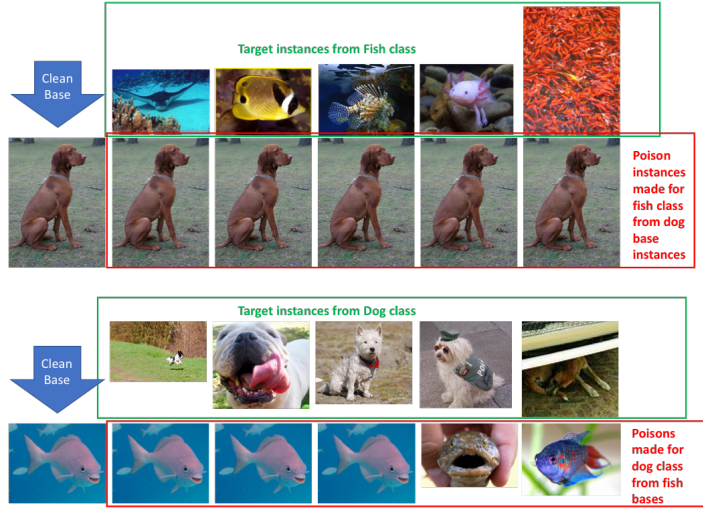
²If an identical image appears in both the train and test set, it could be chosen as both a base and target, in which case poisoning is trivial. We remove duplicate images to prevent this sort of “cheating.”

this, we are left with 1099 test instances (698 test instances for the dog class and 401 test instances for the fish class).

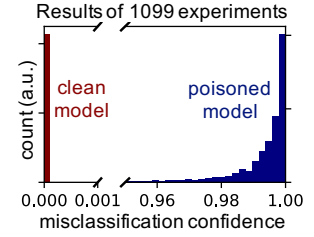
We select both target and base instances from the test set and craft a poison instance using Algorithm 1 with $maxIters = 1000$. Since the images in ImageNet have different dimensions, we calculate β for Eq. 2.1 using $\beta = \beta_0 \cdot 2048^2 / (dim_b)^2$ which takes the dimensionality of the base instance (dim_b) and the dimension of InceptionV3’s feature space representation layer (2048) into account. We use $\beta_0 = 0.25$ in our experiments. We then add the poison instance to the training data and perform cold-start training (all unfrozen weights initialized to random values). We use the Adam optimizer with learning rate of 0.01 to train the network for 100 epochs.

The experiment is performed 1099 times – each with a different test-set image as the target instance – yielding an attack success rate of 100%. For comparison, the influence function method studied in [4] reports a success rate of 57% . The median misclassification confidence was 99.6% (Fig. 2.1b). Further, the overall test accuracy is hardly affected by the poisoning, dropping by an average of 0.2%, with a worst-case of 0.4%, from the original 99.5% over all experiments. Some sample target instances and their corresponding poison instances are illustrated in Fig. 2.1a.

Note that it is not generally possible to get 100% success rate on transfer learning tasks. The reason that we are able to get such success rate using InceptionV3 on the dog-vs-fish task is because there are more trainable weights (2048) than training examples (1801). As long as the data matrix contains no duplicate images, the system of equations that needs to be solved to find the weight vector is under-determined and overfitting on all of the training data is certain to occur.



(a) Sample target and poison instances.



(b) Incorrect class's probability histogram predicted for the target image by the clean (dark red) and poisoned (dark blue) models. When trained on a poisoned dataset, the target instances not only get misclassified; they get misclassified with high confidence.

Figure 2.1: Transfer learning poisoning attack. (a) The top row contains 5 random target instances (from the “fish” class). The second row contains the constructed poison instance corresponding to each of these targets. We used the same base instance (second row, leftmost image) for building each poison instance. The attack is effective for any base, but fewer iterations are required if the base image has a higher resolution. We stopped the poison generation algorithm when the maximum iterations was met or when the feature representation of the target and poison instances were less than 3 units apart (in Euclidean norm). The stopping threshold of 3 was determined by the minimum distance between all pairs of training points. As can be seen, the poison instances are visually indistinguishable from the base instance (and one another). Rows 3 and 4 show samples from similar experiments where the target (fish) and base (dog) classes were swapped.

To better understand what causes the attacks to be successful, we plot the angular deviation between the decision boundary (i.e. the angular difference between the weight vectors) of the clean and poisoned networks in Fig. 2.2 (blue bars and lines). The angular deviation is the degree to which retraining on the poison instance caused the decision boundary to rotate to encompass the poison instance within the base region. This deviation occurs mostly in the first epoch as seen in Fig. 2.2b, suggesting that the attack may succeed even with suboptimal retraining hyperparameters. The final deviation of 23 degrees on average (Fig. 2.2a) indicates that a substantial alteration to the final layer decision boundary is made by the poison instance. These results verify our intuition that misclassification of the target occurs due to changes in the decision boundary.

While our main formulation (Eq. 2.1) promotes similarity between the poison and base images via the ℓ_2 metric, the same success rate of 100% is achieved when we promote similarity via an ℓ_∞ bound of 2 (out of a dynamic range of 255) as is done in [4]. Details of the experiment are presented in the supplementary material.

The experiments here were on a binary classification task (“dog” vs. “fish”). There is, however, no constraint on applying the same poisoning procedure to many-class problems. In the supplementary material, we present additional experiments where a new class, “cat”, is introduced and we show 100% poisoning success on the 3-way task is still achieved while maintaining an accuracy of 96.4% on clean test images.

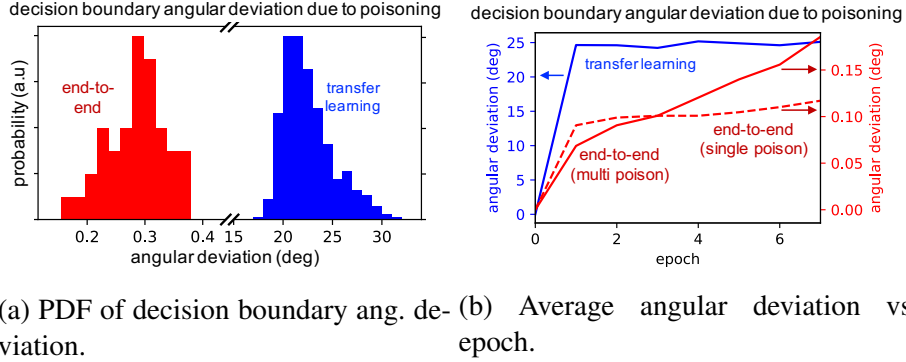


Figure 2.2: Angular deviation of the feature space decision boundary when trained with clean dataset + poison instance(s) versus when trained with clean dataset alone. (a) Histogram of the final (last epoch) angular deviation over all experiments. In transfer learning (blue), there is a significant rotation (average of 23 degrees) in the feature space decision boundary. In contrast, in end-to-end training (red) where we inject 50 poison instances, the decision boundary’s rotation is negligible. (b) Most of the parameter adjustment is done during the first epoch. For the end-to-end training experiments, the decision boundary barely changes.

2.4 Poisoning attacks on end-to-end training

We saw in Section 2.3 that poisoning attacks on transfer learning are extremely effective. When all layers are trainable, these attacks become more difficult. However, using a “watermarking” trick and multiple poison instances, we can still effectively poison end-to-end networks.

Our end-to-end experiments focus on a scaled-down AlexNet architecture for the CIFAR-10 dataset³ (architectural details in appendix), initialized with pretrained weights (warm-start), and optimized with Adam at learning rate 1.85×10^{-5} over 10 epochs with batch size 128. Because of the warm-start, the loss was constant over the last few epochs after the network had readjusted to correctly classify the poison instances.

³We do this to keep runtimes short since quantifying performance of these attacks requires running each experiment (and retraining the whole network) hundreds of times.

2.4.1 Single poison instance attack

We begin with an illustrative example of attacking a network with a single poison instance. Our goal is to visualize the effect of a poison on the network’s behavior, and explain why poisoning attacks under end-to-end training are more difficult than under transfer learning. For the experiments, we randomly selected “airplane” as the target class and “frog” as the base class. For crafting poison instances, we used a β value of 0.1 and iteration count of 12000. Figure 2.3a shows the target, base, and poison feature space representations visualized by projecting the 193-dimensional deep feature vectors onto a 2-dimensional plane. The first dimension is along the vector joining the centroids of the base and target classes ($\mathbf{u} = \mu_{base} - \mu_{target}$), while the second dimension is along the vector orthogonal to $\mathbf{u}$ and in the plane spanned by $\mathbf{u}$ and θ (the weight vector of the penultimate layer, i.e. the normal to the decision boundary). This projection allows us to visualize the data distribution from a viewpoint best representing the separation of the two classes (target and base).

We then evaluate our poisoning attack by training the model with the clean data + single poison instance. Fig. 2.3a shows the feature space representations of the target, base, and poison instances along with the training data under a clean (unfilled markers) and poisoned (filled markers) model. In their clean model feature space representations, the target and poison instances are overlapped, indicating that our poison-crafting optimization procedure (Algorithm 1) works. Oddly, unlike the transfer learning scenario where the final layer decision boundary rotates to accommodate the poison instance within the base region, the decision boundary in the end-to-end training scenario is *un-*

changed after retraining on the poisoned dataset, as seen through the red bars and lines in Fig. 2.2.

From this, we make the following important observation: *During retraining with the poison data, the network modifies its lower-level feature extraction kernels in the shallow layers so the poison instance is returned to the base class distribution in the deep layers.*

In other words, the poison instance generation exploits imperfections in the feature extraction kernels in earlier layers such that the poison instance is placed alongside the target in feature space. When the network is retrained on this poison instance, because it is labeled as a base, those early-layer feature kernel imperfections are corrected and the poison instance is returned to the base class distribution. This result shows that the objectives of poison instance generation and of network training are mutually opposed and thus a single poison may not be enough for compromising even extreme outlier target examples. To make the attack successful, we must find a way to ensure that the target and poison instances do not get separated in feature space upon retraining.

2.4.2 Watermarking: a method to boost the power of poison attacks

To prevent the separation of poison and target during training, we use a simple but effective trick: add a low-opacity watermark of the target instance to the poisoning instance to allow for some inseparable feature overlap while remaining visually distinct. This blends some features of the target instance into the poison instance and should cause the poison instance to remain within feature space proximity of the target instance even

after retraining. Watermarking has been previously used in [15], but their work required the watermark to be applied during inference time, which is unrealistic in situations where the attacker cannot control the target instance.

A base watermarked image with target opacity γ is formed by taking a weighted combination of the base b and the target images t : $\mathbf{b} \leftarrow \gamma \cdot \mathbf{t} + (1 - \gamma) \cdot \mathbf{b}$. Some randomly selected poison instances are shown in the supplementary material. Watermarks are not visually noticeable even up to 30% opacity for some target instances. Fig. 2.4 illustrates 60 poison instances used for successfully attacking a “bird” target instance.

2.4.2.1 Multiple poison instance attacks

Poisoning in the end-to-end training scenario is difficult because the network learns feature embeddings that optimally distinguish the target from the poison. But what if we introduce multiple poison instances derived from different base instances into the training set?

For the classifier to resist multiple poisons, it must learn a feature embedding that separates *all* poison instances from the target while also ensuring that the target instance remains in the target distribution. We show in our experiments that using a high diversity of bases prevents the moderately-sized network from learning features of the target that are distinct from those of the bases. Consequently, when the network is retrained, the target instance is pulled along with the poison instances toward the base distribution, and attacks are frequently successful. These dynamics are shown in Fig. 2.3b.

In Fig. 2.2, we observe that even in the multiple poison experiments, the decision

boundary of the final layer remains unchanged, suggesting that there’s a fundamentally different mechanism by which poisoning succeeds in the transfer learning vs. end-to-end training scenarios. Transfer learning reacts to poisons by rotating the decision boundary to encompass the target, while end-to-end training reacts by pulling the target into the base distribution (in feature space). The decision boundary in the end-to-end scenario remains stationary (varying by fractions of a degree) under retraining on the poisoned dataset, as shown in Fig. 2.2.

To quantify how the number of poison instances impacts success rate, we ran experiments for each number of poison instances between 1 and 70 (increments of 5). Experiments used randomly chosen target instances from the test set. Each poison was generated from a random base in the test set (resulting in large feature diversity among poisons). A watermarking opacity of 30% or 20% was used to enhance feature overlap between the poisons and targets. The attack success rate (over 30 random trials) is shown in Fig. 2.5b. The set of 30 experiments was repeated for a different target-base class pair within CIFAR-10 to verify that the success rates are not class dependent. We also try a lower opacity and observe that the success rate drops. The success rate increases monotonically with the number of poison instances. With 50 poisons the success rate is about 60% for the bird-vs-dog task. Note we declare success only when the target is classified as a base; the attack is considered unsuccessful even when the target instance is misclassified to a class other than the base.

We can increase the success rate of this attack by targeting data outliers. These targets lie far from other training samples in their class, and so it should be easier to flip their class label. We target the 50 “airplanes” with the lowest classification confidence

(but still correctly classified), and attack them using 50 poison frogs per attack. The success rate for this attack is 70% (Fig. 2.5a), which is 17% higher than for randomly chosen targets.

To summarize, clean-label attacks under the end-to-end scenario require multiple techniques to work: (1) optimization via Algorithm 1, (2) diversity of poison instances, and (3) watermarking. In the supplementary material, we provide a leave-one-out ablation study with 50 poisons which verifies that all three techniques are required for successful poisoning.

2.5 Conclusion

We studied targeted clean-label poisoning methods that attack a net at training time with the goal of manipulating test-time behavior. These attacks are difficult to detect because they involve non-suspicious (correctly labeled) training data, and do not degrade the performance on non-targeted examples. The proposed attack crafts poison images that collide with a target image in feature space, thus making it difficult for a network to discern between the two. These attacks are extremely powerful in the transfer learning scenario, and can be made powerful in more general contexts by using multiple poison images and a watermarking trick.

Training with poison instances is akin to the *adversarial training* technique for defending against evasion attacks [9]. The poison instance can here be seen as an adversarial example to the base class. While our poisoned dataset training does indeed make the network more robust to base-class adversarial examples designed to be misclassified as the

target, it also has the effect of causing the unaltered target instance to be misclassified as a base. This side effect of adversarial training was exploited in this chapter, and is worth further investigation.

Many neural networks are trained using data sources that are easily manipulated by adversaries. We hope that this work will raise attention for the important issue of data reliability and provenance.

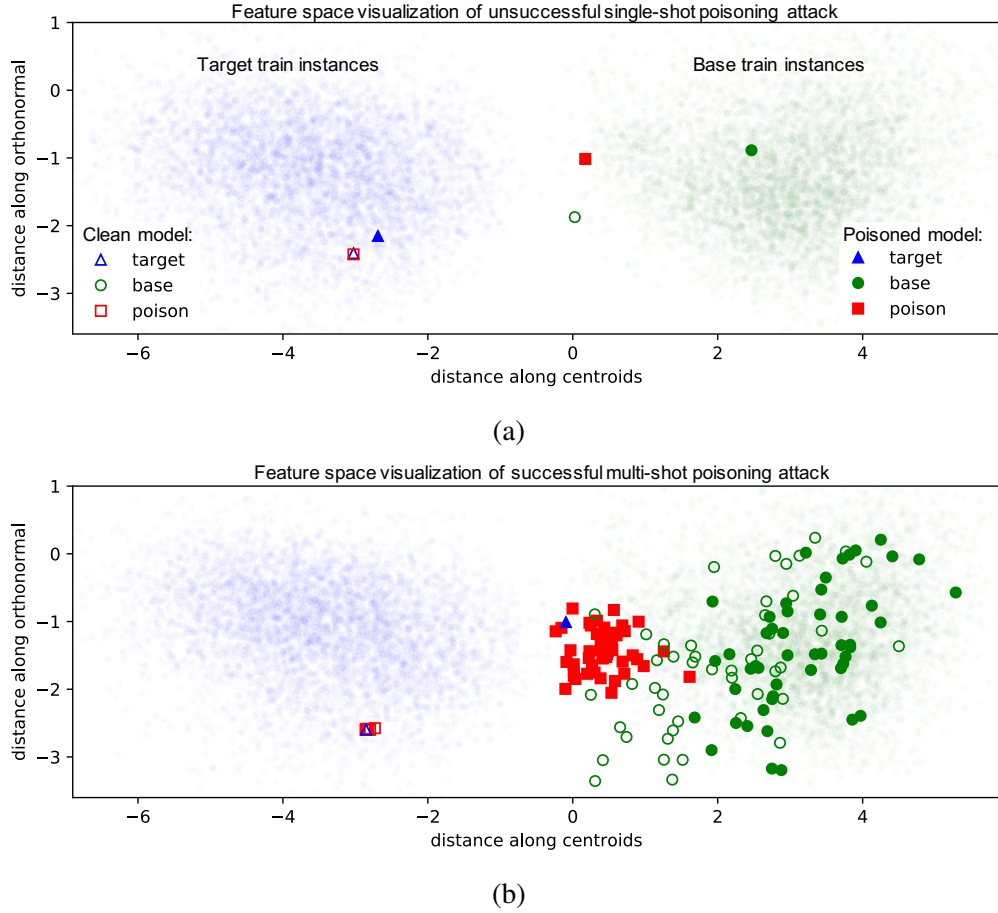


Figure 2.3: Feature space visualization of end-to-end training poisoning attacks. (a) A single poison instance is unable to successfully attack the classifier. The poison instance’s feature space position under the clean model is overlapped with that of the target instance. However, when the model is trained on the clean + poisoned data (i.e. the poisoned model), the feature space position of the poison instance is returned to the base class distribution, while target remains in the target class distribution. (b) To make the attack successful, we construct 50 poison instances from 50 random base instances that are “watermarked” with a 30% opacity target instance. This causes the target instance to be pulled out of the target class distribution (in feature space) into the base class distribution and get incorrectly classified as the base class.

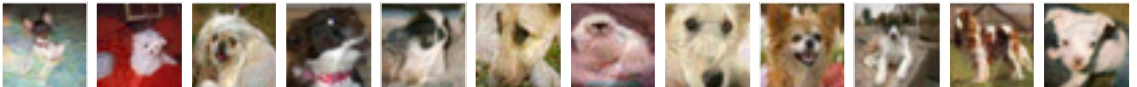
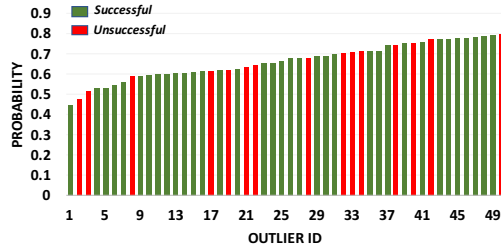
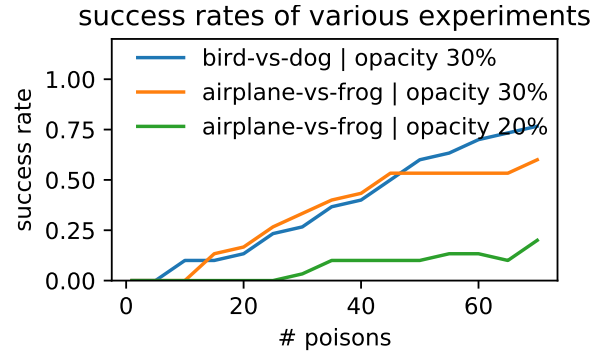


Figure 2.4: 12 out of 60 random poison instances that successfully cause a bird target instance to get misclassified as a dog in the end-to-end training scenario. An adversarial watermark (opacity 30%) of the target bird instance is applied to the base instances when making the poisons. More examples are in the supplementary material.



(a) Attacks on the most outlier target airplanes. The bars indicate the probability of the target instance before the attack (calculated using the pre-trained network). The coloring, denotes whether the attack was successful or unsuccessful. Each experiment utilizes a watermark opacity of 30% and 50 poisons. Out of these 50 outliers, the attack succeeds 70% of the time (compare to 53% for a random target).



(b) Success rate of attacks on different targets from different bases as a function of number of poison instances used and different target opacity added to the base instances.

Figure 2.5: Success rates for attacks on outliers and random targets. While attacking non-outlier is still possible, attacking an outlier can increase the chances of success.

Chapter 3: Universal Adversarial Training

3.1 Introduction

Deep neural networks (DNNs) are vulnerable to adversarial examples, in which small and often imperceptible perturbations change the class label of an image [8, 29, 30, 31]. Because of the security concerns this raises, there is an increasing interest in studying these attacks and also designing mechanisms to defend against them.

Adversarial examples were originally formed by selecting a single base image, and then sneaking it into a different class using a small perturbation [29, 32, 1]. This is done most effectively using (potentially expensive) iterative optimization procedures [33, 1, 34].

Different from per-instance perturbation attacks, Moosavi-Dezfooli et al. [6, 35] show there exists “universal” perturbations that can be added to any image to change its class label (fig. 3.1) with high probability. Universal perturbations empower attackers who cannot generate per-instance adversarial examples on the go, or who want to change the identity of an object to be selected later in the field. What’s worse, universal perturbations have good cross-model transferability, which facilitates black-box attacks.

Among various methods for hardening networks to per-instance attacks, *adversarial training* [1] is known to dramatically increase robustness [34]. In this process, adversarial

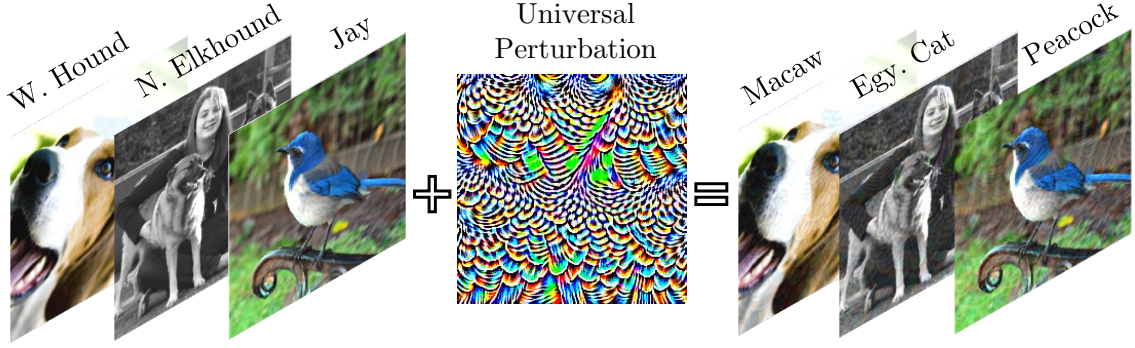


Figure 3.1: A universal perturbation made using a subset of ImageNet and the VGG-16 architecture. When added to the validation images, their labels usually change. The perturbation was generated using the proposed algorithm 3. Perturbation pixel values lie in $[-10, 10]$ (i.e. $\epsilon = 10$).

examples are produced for each mini-batch during training, and injected into the training data. While effective at increasing robustness against small perturbations, it is not effective for larger perturbations which are often the case for universal perturbations. Also, the high cost of this process precludes its use on large and complex datasets.

Contributions This chapter studies effective methods for producing and deflecting universal adversarial attacks. First, we pose the creation of universal perturbations as an optimization problem that can be effectively solved by stochastic gradient methods. This method dramatically reduces the time needed to produce attacks as compared to [6]. The efficiency of this formulation empowers us to consider universal adversarial training. We formulate the adversarial training problem as a min-max optimization where the minimization is over the network parameters and the maximization is over the universal perturbation. This problem can be solved quickly using alternating stochastic gradient methods with no inner loops, making it far more efficient than per-instance adversarial training with a strong adversary (which requires a PGD inner loop to generate adversar-

ial perturbations). Prior to our work, it was argued that adversarial training on universal perturbations is infeasible because the inner optimization requires the generation of a universal perturbation from scratch, which requires a lot of computation and many iterations [36]. We further improve the defense efficiency by providing a “low-cost” algorithm for defending against universal perturbations. Through experiments on CIFAR-10 and ImageNet, we show that this “low-cost” method works well in practice.

3.2 Related work

We briefly review *per-instance* perturbation attack techniques that are closely related to this chapter and can be used during the universal perturbation update step of universal adversarial training. The Fast Gradient Sign Method (FGSM) [29] is one of the most popular one-step gradient-based approaches for ℓ_∞ -bounded attacks. FGSM applies one step of gradient ascent in the direction of the sign of the gradient of the loss function with respect to the input image. When a model is FGSM adversarially trained, the gradient of the loss function may be very small near unmodified images. In this case, the R-FGSM method remains effective by first using a random perturbation to step off the image manifold, and then applying FGSM [37]. Projected Gradient Descent (PGD) [38, 1] iteratively applies FGSM multiple times, and is one of the strongest per-instance attacks [1, 34]. The PGD version of [1] applies an initial random perturbation before multiple steps of gradient ascent and projection of perturbation on to the norm-ball of interest. Finally, DeepFool [39] is an iterative method based on a linear approximation of the training loss objective. This method formed the backbone of the original method for

producing universal adversarial examples [6].

Adversarial training, in which adversarial examples are injected into the dataset during training, is an effective method to learn a robust model resistant to per-instance attacks [1, 34, 40, 41, 42]. Robust models adversarially trained with FGSM can resist FGSM attacks [38], but can be vulnerable to PGD attacks [1]. Madry et al. [1] suggest strong attacks are important, and they use the iterative PGD method in the inner loop for generating adversarial examples when optimizing the min-max problem. PGD adversarial training is effective but time-consuming when the perturbation is small. The cost of the inner PGD loop is high, although this can sometimes be replaced with neural models for attack generation [43, 44, 45]. These robust models are adversarially trained to fend off per-instance perturbations and have not been designed for, or tested against, universal perturbations.

Unlike per-instance perturbations, *universal* perturbations can be directly added to any test image to fool the classifier. In [6], universal perturbations for image classification are generated by iteratively optimizing the per-instance adversarial loss for training samples using DeepFool [39]. In addition to classification tasks, universal perturbations are also shown to exist for semantic segmentation [46]. Robust universal adversarial examples are generated as a universal targeted adversarial patch in [47]. They are targeted since they cause misclassification of the images to a given target class. Moosavi-Dezfooli et al. [35] prove the existence of small universal perturbations under certain curvature conditions of decision boundaries. Data-independent universal perturbations are also shown to exist and can be generated by maximizing spurious activations at each layer. These universal perturbations are slightly weaker than the data dependent approaches [48, 49]. As

a variant of universal perturbation, unconditional generators are trained to create perturbations from random noises for attack [50, 51]. Universal perturbations are often larger than per-instance perturbation. For example on ImageNet, universal perturbations generated in prior works have ℓ_∞ perturbations of size $\epsilon = 10$ [6] while non-targeted per-instance perturbations as small as $\epsilon = 2$ are often enough to considerably degrade the performance of conventionally trained classifiers. As a consequence, from a defense perspective, per-instance defenses on ImageNet focus on smaller perturbations compared to universal perturbations.

There has been very little work on defending against universal attacks. To the best of our knowledge, the only dedicated study is by Akhtar et al., who propose a perturbation rectifying network that pre-processes input images to remove the universal perturbation [7]. The rectifying network is trained on universal perturbations that are built for the downstream classifier. While other methods of data sanitization exist [52, 53], it has been shown (at least for per-instance adversarial examples) that this type of defense is easily subverted by an attacker who is aware that a defense network is being used [54].

Two recent preprints [36, 55] model the problem of defending against universal perturbations as a two-player min-max game. However, unlike us, and similar to per-instance adversarial training, after each gradient descent iteration for updating the DNN parameters, they generate a universal adversarial example in an iterative fashion. Since the generation of universal adversarial perturbations is very time-consuming [7], this makes their approach very slow in practice and prevents them from training the neural network parameters for many iterations.

Algorithm 2: Iterative solver for universal perturbations [6]

```
1 Initialize  $\delta \leftarrow 0$ 
2 while  $\text{Prob}(X, \delta) < 1 - \xi$  do
3   for  $x_i$  in  $X$  do
4     if  $f(w, x_i + \delta) \neq f(w, x_i)$  then
5       Solve  $\min_r \|r\|_2$  s.t.  $f(w, x_i + \delta + r) \neq f(w, x_i)$       by DeepFool
6       Update  $\delta \leftarrow \delta + r$ 
7       project  $\delta$  to  $\ell_p$  ball
8     end
9   end
10 end
```

3.3 Optimization for universal perturbation

Given a set of training samples $X = \{x_i, i = 1, \dots, N\}$ and a network $f(w, \cdot)$ with frozen parameter w that maps images onto labels, Moosavi-Dezfooli et al. [6] propose to find universal perturbations δ that satisfy,

$$\|\delta\|_p \leq \epsilon \text{ and } \text{Prob}(X, \delta) \geq 1 - \xi, \quad (3.1)$$

$\text{Prob}(X, \delta)$ represents the “fooling ratio,” which is the fraction of images x whose perturbed class label $f(w, x + \delta)$ differs from the original label $f(w, x)$. The parameter ϵ controls the ℓ_p diameter of the bounded perturbation, and ξ is a small tolerance hyperparameter. Problem (3.1) is solved by the iterative method in algorithm 2 [6]. This solver relies on an inner loop to apply DeepFool [39] to each training instance, which makes the solver slow. Moreover, the outer loop of algorithm 2 is not guaranteed to converge. Different from [6], we consider the following optimization problem for building universal

perturbations,

$$\max_{\delta} \mathcal{L}(w, \delta) = \frac{1}{N} \sum_{i=1}^N l(w, x_i + \delta) \text{ s.t. } \|\delta\|_p \leq \epsilon, \quad (3.2)$$

where $l(w, \cdot)$ represents the loss used for training DNNs. This simple formulation (3.2) searches for a universal perturbation that maximizes the training loss, and thus forces images into the wrong class.

The naive formulation (3.2) suffers from a potentially significant drawback; the cross-entropy loss is unbounded from above, and can be arbitrarily large when evaluated on a single image. In the worst-case, a perturbation that causes misclassification of just a single image can maximize (3.2) by forcing the average loss to infinity. To force the optimizer to find a perturbation that fools many instances, we propose a “clipped” version of the cross entropy loss,

$$\hat{l}(w, x_i + \delta) = \min\{l(w, x_i + \delta), \beta\}. \quad (3.3)$$

We cap the loss function at β to prevent any single image from dominating the objective in (3.2), and giving us a better surrogate of misclassification accuracy. In section 3.5.2, we investigate the effect of clipping with different β .

We directly solve eq. (3.2) by a stochastic gradient method described in algorithm 3. Each iteration begins by using gradient ascent to update the universal perturbation δ so that it maximizes the loss. Then, δ is projected onto the ℓ_p -norm ball to prevent it from growing too large. We experiment with various optimizers for this ascent step, including Stochastic Gradient Descent (SGD), Momentum SGD (MSGD), Projected Gradient

Algorithm 3: Stochastic gradient for universal perturbation

```
1 for  $epoch = 1 \dots N_{ep}$  do
2   for minibatch  $B \subset X$  do
3     Update  $\delta$  with gradient variant  $\delta \leftarrow \delta + g$ 
4     Project  $\delta$  to  $\ell_p$  ball
5   end
6 end
```

Descent (PGD), and ADAM [56].

We test this method by attacking a naturally trained WideResnet 28-10 architecture on the CIFAR-10 dataset. We use $\epsilon = 8$ for the ℓ_∞ constraint for CIFAR-10 following [1]. Stochastic gradient methods that use “normalized” gradients (ADAM and PGD) are less sensitive to learning rate and converge faster, as shown in fig. 3.2. We visualize the generated universal perturbation from different optimizers in fig. 3.3. Compared to the noisy perturbation generated by SGD, normalized gradient methods produced stronger attacks with more well-defined geometric structures and checkerboard patterns. The final evaluation accuracies (on test-examples) after adding universal perturbations with $\epsilon = 8$ were 42.56% for the SGD perturbation, 13.08% for MSGD, 13.30% for ADAM, and 13.79% for PGD. The clean test accuracy of the WideResnet is 95.2%.

Our proposed method of universal attack using a clipped loss function has several advantages. It is based on a standard stochastic gradient method that comes with convergence guarantees when a decreasing learning rate is used [57]. Also, each iteration is based on a minibatch of samples instead of one instance, which accelerates computation on a GPU. Finally, each iteration requires a simple gradient update instead of the complex DeepFool inner loop; we empirically verify fast convergence and good performance of the proposed method (see section 3.5).

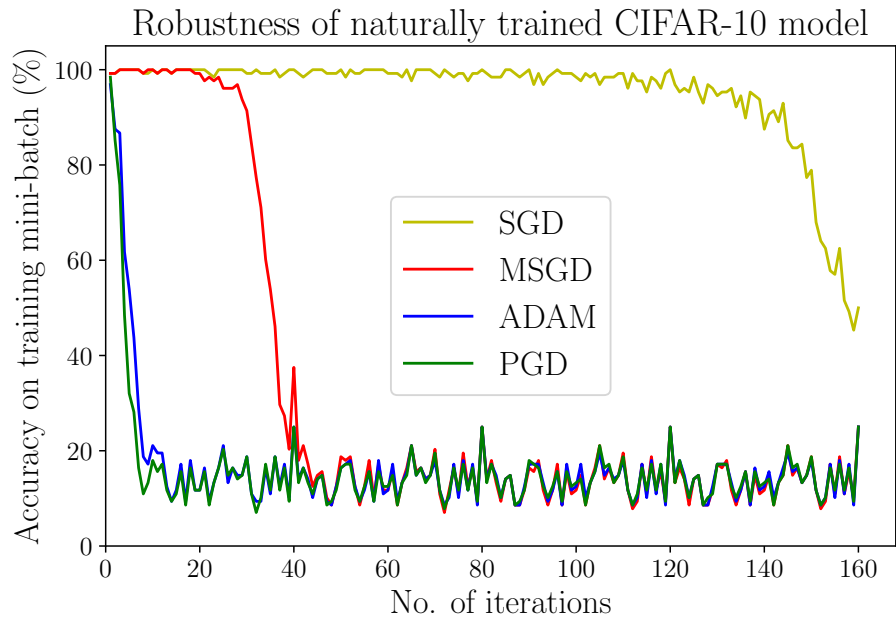


Figure 3.2: Classification accuracy on adversarial examples of universal perturbations generated by increasing the cross-entropy loss. PGD and ADAM converge faster. We use 5000 training samples from CIFAR-10 for constructing the universal adversarial perturbation for naturally trained Wide ResNet model from [1]. The batch-size is 128, $\epsilon=8$, and the learning-rate/step-size is 1.

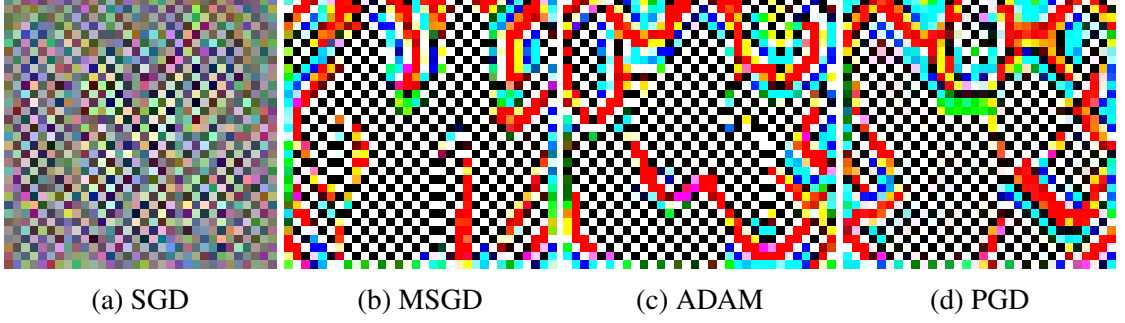


Figure 3.3: Visualizations of universal perturbations after 160 iterations of the optimizers depicted in fig. 3.2.

3.4 Universal adversarial training

We now consider training robust classifiers that are resistant to universal perturbations. Similar to [1], we borrow ideas from robust optimization. We use robust optimization to build robust models that can resist universal perturbations. In particular, we consider universal adversarial training, and formulate this problem as a min-max optimization problem,

$$\begin{aligned} \min_w \max_{\delta} \mathcal{L}(w, \delta) &= \frac{1}{N} \sum_{i=1}^N l(w, x_i + \delta) \\ \text{s.t. } \|\delta\|_p &\leq \epsilon, \end{aligned} \tag{3.4}$$

where w represents the neural network weights, $X = \{x_i, i = 1, \dots, N\}$ represents training samples, δ represents universal perturbation noise, and $l(\cdot)$ is the loss function. Here, unlike conventional adversarial training, our δ is a universal perturbation (or, more accurately, mini-batch universal). Previously, solving this optimization problem directly was deemed computationally infeasible due to the large cost associated with generating a universal perturbation [36], but we show that eq. (3.4) is efficiently solvable by alternating

Algorithm 4: Alternating stochastic gradient method for adversarial training against universal perturbation

```

1 Input: Training samples  $X$ , perturbation bound  $\epsilon$ , learning rate  $\tau$ , momentum  $\mu$ 
  for  $epoch = 1 \dots N_{ep}$  do
2   for minibatch  $B \subset X$  do
3     Update  $w$  with momentum stochastic gradient
4      $g_w \leftarrow \mu g_w - \mathbb{E}_{x \in B} [\nabla_w l(w, x + \delta)]$ 
5      $w \leftarrow w + \tau g_w$ 
6     Update  $\delta$  with stochastic gradient ascent
7      $\delta \leftarrow \delta + \epsilon \text{sign}(\mathbb{E}_{x \in B} [\nabla_\delta l(w, x + \delta)])$ 
8     Project  $\delta$  to  $\ell_p$  ball
9   end
10 end

```

stochastic gradient methods shown in algorithm 4. We show that unlike [1], updating the universal perturbation only using a simple step is enough for building universally hardened networks. Each iteration alternatively updates the neural network weights w using gradient descent, and then updates the universal perturbation δ using ascent.

We compare our formulation (3.4) and algorithm 4 with PGD-based adversarial training in [1], which trains a robust model by optimizing the following min-max problem,

$$\min_w \max_Z \frac{1}{N} \sum_{i=1}^N l(w, z_i) \text{ s.t. } \|Z - X\|_p \leq \epsilon. \quad (3.5)$$

The standard formulation (3.5) searches for per-instance perturbed images Z , while our formulation in (3.4) maximizes using a universal perturbation δ . Madry et al. [1] solve (3.5) by a stochastic method. In each iteration, an adversarial example z_i is generated for an input instance by the PGD iterative method, and the neural network parameter w is updated once [1]. Our formulation (algorithm 4) only maintains one single perturbation that is used and refined across all iterations. For this reason, we need only to update w

and δ once per step (i.e., there is no expensive inner loop), and these updates accumulate for both w and δ through training.

We consider different rules for updating δ during universal adversarial training,

$$\text{FGSM } \delta \leftarrow \delta + \epsilon \cdot \text{sign}(\mathbb{E}_{x \in B}[\nabla_{\delta} l(w, x + \delta)]), \quad (3.6)$$

$$\text{SGD } \delta \leftarrow \delta + \tau_{\delta} \cdot \mathbb{E}_{x \in B}[\nabla_{\delta} l(w, x + \delta)], \quad (3.7)$$

and ADAM [56]. We found that the FGSM update rule was most effective when combined with the SGD optimizer for updating neural network weights w .

We use fairly standard training parameters in our experiments. In our CIFAR experiments, we use $\epsilon = 8$, batch-size of 128, and we train for 80,000 steps. For the optimizer, we use Momentum SGD with an initial learning rate of 0.1 which drops to 0.01 at iteration 40,000 and drops further down to 0.001 at iteration 60,000. One way to assess the update rule is to plot the model accuracy before and after the ascent step (i.e., the perturbation update). It is well-known that adversarial training is more effective when stronger attacks are used. In the extreme case of a do-nothing adversary, the adversarial training method degenerates to natural training. As illustrated in the supplementary, we see a gap between the accuracy curves plotted before and after gradient ascent. We find that the FGSM update rule leads to a larger gap, indicating a stronger adversary. Correspondingly, we find that the FGSM update rule yields networks that are more robust to attacks as compared to SGD update.

3.4.1 Attacking hardened models

We evaluate the robustness of different models by applying algorithm 3 to find universal perturbations. We attack universally adversarial trained models (produced by eq. (3.4)) using the FGSM universal update rule (uFGSM), or the SGD universal update rule (uSGD). We also consider robust models from per-instance adversarial training (eq. (3.5)) with adversarial steps of the FGSM and PGD type [1].

Robust models adversarially trained with weaker attackers such as uSGD and (per-instance) FGSM are relatively vulnerable to universal perturbations, while robust models from (per-instance) PGD [1] and uFGSM can resist universal perturbations. We plot the universal perturbations generated using algorithm 4 in fig. 3.4. When we visually compare the universal perturbations of robust models (fig. 3.4) and those of a naturally trained model (fig. 3.3), we can see a drastic change in structure. Similarly, even among hardened models, universal perturbations generated for weaker robust models have more geometric textures, as shown in fig. 3.4 (a,d).

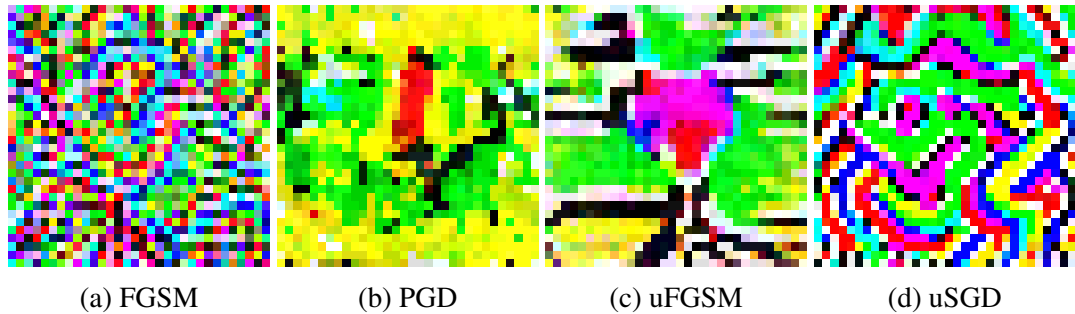


Figure 3.4: The universal perturbations made for 4 different robust models trained on CIFAR-10: adversarially trained with FGSM or PGD, and universally adversarially trained with FGSM (uFGSM) or SGD (uSGD). Perturbations were made using 400 iterations.

While an ϵ -bounded per-instance robust model is also robust against ϵ -bounded uni-

versal attacks since the universal attack is a more constrained version of the per-instance attack, training robust per-instance models are only partially possible for small datasets such as CIFAR and with small choices of ϵ . For larger datasets such as ImageNet, we cannot train a per-instance robust model with such a large ϵ which is common for universal attacks. However, we include the per-instance adversarially trained model as a candidate universally robust model for CIFAR-10 in our comparisons to allow comparisons in settings where it is possible to train per-instance robust models unlike ImageNet.

We apply the strongest attack to the validation images of the natural model and various universal adversarially trained models using different update steps. The results are summarized in table 3.1. Our models become very robust against universal perturbations and have higher accuracies on natural validation examples compared to per-instance adversarially trained models. Compared to the per-instance FGSM trained model which has the same computational cost as ours, our universal model is much more robust. Note that the PGD trained model is trained on a 7-step per-instance adversary and requires about $4\times$ more computation than ours. Our universally adversarially trained models are more robust than this strong and more computationally expensive defense.

		Validation Accuracy on	
		UnivPert	Natural
(Robust) models trained with	Natural	9.2%	95.2%
	FGSM	51.0%	91.42%
	PGD	86.1%	87.25%
	uADAM (ours)	91.6%	94.28%
	uFGSM (ours)	91.8%	93.50%

Table 3.1: Validation accuracy of hardened WideResnet models trained on CIFAR-10. Note that Madry’s PGD training is significantly slower than the other training methods.

3.4.2 Low-cost universal adversarial training

As shown in table 3.1, our proposed algorithm 4 was able to harden the CIFAR-10 classification network. This comes at the cost of doubling the training time. Adversarial training in general should have some cost since it requires the generation or update of the adversarial perturbation of the mini-batch *before* each minimization step on the network’s parameters. However, since universal perturbations are approximately image-agnostic, results should be fairly invariant to the order of updates. For this reason, we propose to compute the image gradient needed for the perturbation update during the same backward pass used to compute the parameter gradients. This results in a simultaneous update for network weights and the universal perturbation in algorithm 5, which backprops only once per iteration and produces approximately universally robust models at almost no cost in comparison to natural training. The “low-cost universal adversarially trained” model of CIFAR-10 is 86.1% robust against universal perturbations and has 93.5% accuracy on the clean validation examples. When compared to the original version in table 3.1, the robustness has only slightly decreased. However, the training time is cut by half. This is a huge improvement in efficiency, in particular for large datasets like ImageNet with long training times.

3.5 Universal perturbations for ImageNet

To validate the performance of our proposed optimization on different architectures and more complex datasets, we apply algorithm 3 to various popular architectures designed for classification on the ImageNet dataset [27]. We compare our method of uni-

Algorithm 5: Simultaneous stochastic gradient method for adversarial training against universal perturbation

```
1 Input: Training samples  $X$ , perturbation bound  $\epsilon$ , learning rate  $\tau$ , momentum  $\mu$ 
2 Initialize  $w, \delta$ 
3 for  $epoch = 1 \dots N_{ep}$  do
4   for minibatch  $B \subset X$  do
5     Compute gradient of loss with respect to  $w$  and  $\delta$ 
6      $d_w \leftarrow \mathbb{E}_{x \in B} [\nabla_w l(w, x + \delta)]$ 
7      $d_\delta \leftarrow \mathbb{E}_{x \in B} [\nabla_\delta l(w, x + \delta)]$ 
8     Update  $w$  with momentum stochastic gradient
9      $g_w \leftarrow \mu g_w - d_w$ 
10     $w \leftarrow w + \tau g_w$ 
11    Update  $\delta$  with stochastic gradient ascent
12     $\delta \leftarrow \delta + \epsilon \text{sign}(d_\delta)$ 
13    Project  $\delta$  to  $\ell_p$  ball
14  end
15 end
```

versal perturbation generation with the current state-of-the-art method, Iterative DeepFool (iDeepFool for short) [6]. We use the authors’ code to run the iDeepFool attack on these classification networks. For fair comparison, we execute both our method and iDeepFool on the exact same 5000 training data points and terminate both methods after 10 epochs. We use $\epsilon = 10$ for ℓ_∞ constraint following [6], use a step-size of 1.0 for our method, and use suggested parameters for iDeepFool. Similar conclusions could be drawn when we use ℓ_2 bounded attacks. We independently execute iDeepFool since we are interested in the accuracy of the classifier on attacked images – a metric not reported in their paper ¹.

¹They report “fooling ratio” which is the ratio of examples who’s label prediction changes after applying the universal perturbation. This has become an uncommon metric since the fooling ratio can increase if the universal perturbation causes an example that was originally miss-classified to become correctly classified.

3.5.1 Benefits of the proposed method

We compare the performance of our stochastic gradient method for eq. (3.2) and the iDeepFool method for eq. (3.1) in [6]. We generate universal perturbations for Inception [26] and VGG [58] networks trained on ImageNet [27], and report the top-1 accuracy in table 3.2. Universal perturbations generated by both iDeepFool and our method can fool networks and degrade the classification accuracy. Universal perturbations generated for the training samples generalize well and cause the accuracy of the validation samples to drop. However, when given a fixed computation budget such as number of passes on the training data (i.e., epochs), our method outperforms iDeepFool by a large margin. Our stochastic gradient method generates the universal perturbations at a much faster pace than iDeepFool. About $20\times$ faster on InceptionV1 and $6\times$ on VGG16 ($13\times$ on average).

After verifying the effectiveness and efficiency of our proposed stochastic gradient method², we use our algorithm 3 to generate universal perturbations for more advanced architectures such as ResNet-V1 152 [59] and Inception-V3 [26] (and for other experiments in the remaining sections). Our attacks degrade the validation accuracy of ResNet-V1 152 and Inception-V3 from 76.8% and 78% to 16.4% and 20.1%, respectively. The final universal perturbations used for the results presented in this section are illustrated in the supplementary.

²Unless otherwise specified, we use the sign-of-gradient PGD for our stochastic gradient optimizer in algorithm 3.

		InceptionV1	VGG16
Natural	Train	76.9%	81.4 %
	Val	69.7%	70.9%
iDeepFool	Train	43.5%	39.5%
	Val	40.7%	36.0%
Ours	Train	17.2%	23.1%
	Val	19.8%	22.5%
iDeepFool time (s)		9856	6076
our time (s)		482	953

Table 3.2: Top-1 accuracy on ImageNet for natural images, and adversarial images with universal perturbation.

3.5.2 The effect of clipping

In this section we analyze the effect of the “clipping” loss parameter β in eq. (3.2). For this purpose, similar to our other ablation experiments, we generate universal perturbations by solving eq. (3.2) using PGD for Inception-V3 on ImageNet.

Since the results and performance could slightly vary with different random initializations, we run each experiment with 5 random subsets of training data. The accuracy reported is the classification accuracy on the entire validation set of ImageNet after adding the universal perturbation. The results are summarized in fig. 3.5. The results showcase the value of our proposed loss function for finding universal adversarial perturbations.

3.5.3 How much training data does the attack need?

As in [6], we analyze how the number of training points ($|X|$) affects the strength of universal perturbations in fig. 3.6. In particular, we build δ using varying amounts of training data. For each experiment, we report the accuracy on the entire validation set after we add the perturbation δ . We consider four cases for $|X|$: 500, 1000, 2000, and

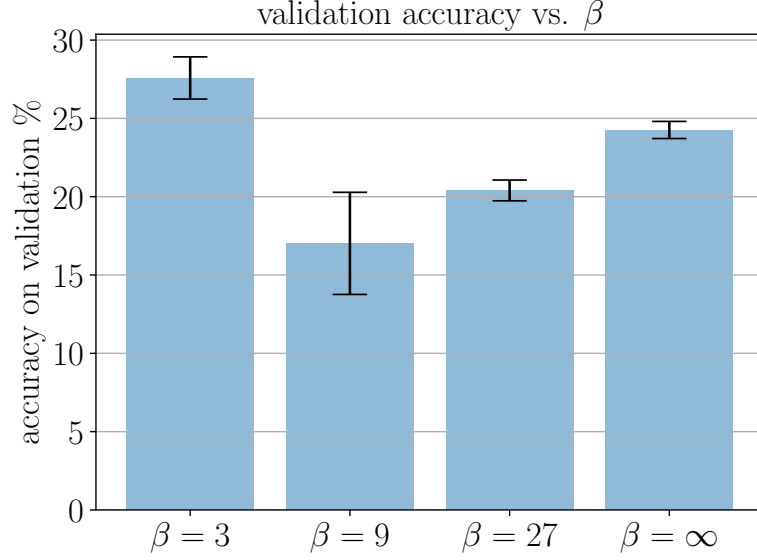


Figure 3.5: Attack performance varies with clipping parameter β in eq. (3.2). Attacking Inception-V3 (with natural validation accuracy 78%) is more successful with clipping ($\beta = 9$) than without clipping ($\beta = \infty$).

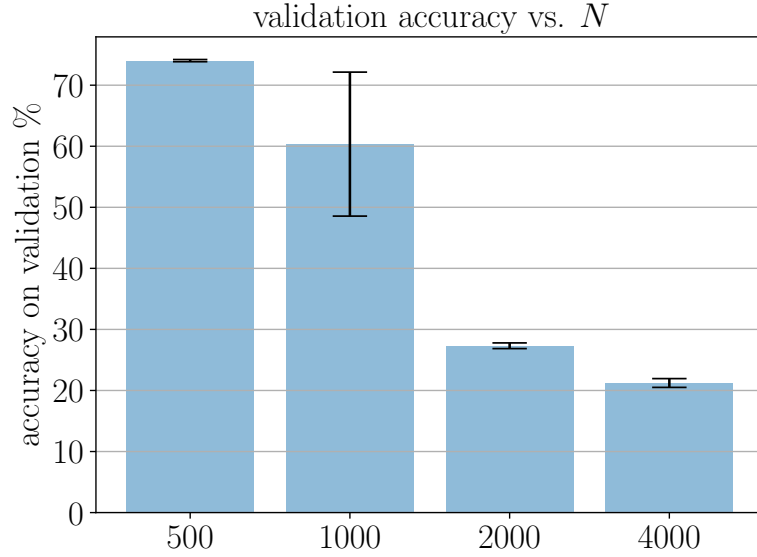


Figure 3.6: The attack performance significantly improves when the number of training points is larger than the number of classes. For reference, Inception-V3’s top-1 accuracy is 78%. Using only a small fraction of the training-data (4,000 / 1,281,167) is enough to degrade the validation and train accuracy to around 20%.

4000³.

³The number of epochs (N_{ep} in algorithm 3) was 100 epochs for 500 data samples, 40 for 1000 and 2000 samples, and 10 for 4000 samples.

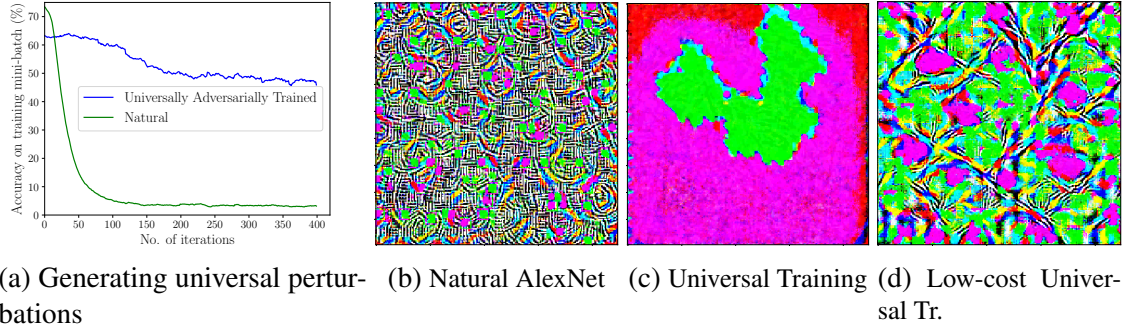


Figure 3.7: Training universal perturbation can fool naturally trained AlexNet on ImageNet, but fails to fool our robust AlexNets. We smoothed the curves in (a) for better visualization. The universal perturbations generated for the universal adversarial trained AlexNets on ImageNet have little geometric structure compared to that of the naturally trained network. (b) Universal perturbation of natural model. The accuracy of the validation images + noise is only **3.9%** (c) Perturbation for our universally trained model using algorithm 4. The accuracy of the validation images + noise for our robust model is **42.0%**. (d) Perturbation for the model trained with our low-cost universal training variant (algorithm 5). The accuracy of the validation images + noise is **28.3%**. While the universal noise for the low-cost variant of universal adversarial training has some structure compared to the original variant, when compared to that of the natural model (b), it is structure-less.

3.6 Universal adversarial training on ImageNet

In this section, we analyze our robust models that are universal adversarially trained by solving the min-max problem (section 3.4) using algorithm 4. We use $\epsilon = 10$ for ImageNet following [6]. Note that unlike CIFAR where we were able to train a per-instance PGD-7 robust model with $\epsilon = 8$, for ImageNet, there exists no model which can resist per-instance non-targeted perturbations with such large ϵ . For ImageNet, we again use fairly standard training parameters (90 epochs, batch-size 256).

Since our universal adversarial training algorithm (algorithm 4) is cheap, it scales to large datasets such as ImageNet. We first train an AlexNet model on ImageNet. We use the natural training hyper-parameters for universal adversarially training our AlexNet

Training	Evaluated Against			
	Natural Images		Universal Attack	
	Top-1	Top-5	Top-1	Top-5
Natural	56.4%	79.0 %	3.9 %	9.4 %
Universal	49.5%	72.7%	42.0%	65.8 %
Low-cost U.	48.4%	72.4%	28.3%	48.3 %

Table 3.3: Accuracy on ImageNet for nat and robust models.

model. Also, we separately use our “low-cost universal training” algorithm to train a robust AlexNet with no overhead cost. We then attack the natural, universally trained, and no-cost universally trained versions of AlexNet using universal attacks.

As seen in fig. 3.7 (a), the AlexNet trained using our universal adversarial training algorithm (algorithm 4) is robust against universal attacks generated using both algorithm 2 and algorithm 3. The naturally trained AlexNet is susceptible to universal attacks. The final attacks generated for the robust and natural models are presented in fig. 3.7 (b,c,d). The universal perturbation generated for the robust AlexNet model has little structure compared to the universal perturbation built for the naturally trained AlexNet. This is similar to the trend we observed in fig. 3.3 and fig. 3.4 for the WideResnet models trained on CIFAR-10.

The accuracy of the universal perturbations on the validation examples are summarized in table 3.3. Similar to CIFAR-10, the low-cost version of universal adversarial training is robust but not as robust as the main method.

We also train a universally robust ResNet-101 ImageNet model. While a naturally trained ResNet-101 achieves only 7.23% accuracy on universal perturbations, our ResNet-101 achieves 74.43% top1 and 92.00% top5 accuracies.

3.7 Conclusion

We proposed using stochastic gradient methods and a “clipped” loss function as an effective universal attack that generates universal perturbations much faster than previous methods. To defend against universal perturbations, we proposed to train robust models by optimizing a min-max problem using alternating or simultaneous stochastic gradient methods. We show that this is possible using certain universal noise update rules that use “normalized” gradients. The simultaneous stochastic gradient method comes at almost no extra cost compared to natural training and has almost no additional cost compared to conventional training.

Chapter 4: Adversarial Training for Free!

4.1 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 [10, 8], 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 [60, 53, 61]. Recently, Athalye et al. broke a complete suite of allegedly robust defenses, leaving *adversarial training*, in which the defender augments each minibatch of training data with adversarial examples [1], among the few that remain resistant to attacks [34]. 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 [43, 44, 45].

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 [62, 63, 64, 65, 66, 67]. 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 [68, 69, 70, 71, 72]. These methods were mostly demonstrated for small networks, low-res datasets, and relatively small perturbation budgets (ϵ). Randomized smoothing was proposed as a certified defense [73] and was later improved [74]. Cohen et al. prove a tight robustness guarantee under the ℓ_2 norm for smoothing with Gaussian noise [75]. Their study was the first certifiable defense for the ImageNet dataset [76]. 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 datasets, such as CIFAR-10 and CIFAR-100, adversarial training is time consuming and can take multiple days.

¹[77] use 128 V100s and [78] use 53 P100s for targeted adv training ImageNet.

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 [1, 77]. 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 [78, 77].

4.2 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

$$\max_{\delta} \quad l(x + \delta, y, \theta), \quad \text{subject to} \quad \|\delta\|_p \leq \epsilon, \quad (4.1)$$

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 [9] is one of the most popular non-targeted methods that uses the sign of the gradients to construct an adversarial example in one iteration:

$$x_{adv} = x + \epsilon \cdot \text{sign}(\nabla_x l(x, y, \theta)). \quad (4.2)$$

The Basic Iterative Method (BIM) by [79] is an iterative version of FGSM. The PGD attack is a variant of BIM with uniform random noise as initialization, which is recognized by [34] to be one of the most powerful first-order attacks. The initial random noise was first studied by [37] 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 chapter we will refer to a K -step

PGD attack as PGD- K .

4.3 Adversarial training

Adversarial training can be traced back to [9], 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 [38, 1]. Consequently, Madry et al. propose training on multi-step PGD adversaries, achieving state-of-the-art robustness levels against ℓ_∞ attacks on MNIST and CIFAR-10 datasets [1].

While many defenses were broken by [34], 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) [78], Feature Denoising [77], Defensive Quantization [80], Thermometer Encoding [81], PixelDefend [82], Robust Manifold Defense [83], L2-nonexpansive nets [84], Jacobian Regularization [66], Universal Perturbation [85], and Stochastic Activation Pruning [86].

We focus on the min-max formulation of adversarial training [1], 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.

4.4 “Free” adversarial training

K -PGD adversarial training [1] is generally slow. For example, the 7-PGD training of a WideResNet [87] on CIFAR-10 in [1] takes about four days on a Titan X GPU. To scale the algorithm to ImageNet, [77] and [78] 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. 6) 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 multi-

Algorithm 6: “Free” Adversarial Training (Free- m)

```
1 Require: Training samples  $X$ , perturbation bound  $\epsilon$ , learning rate  $\tau$ , hop steps  $m$ 
2 Initialize  $\theta$ 
3  $\delta \leftarrow 0$ 
4 for  $epoch = 1 \dots N_{ep}/m$  do
5   for minibatch  $B \subset X$  do
6     for  $i = 1 \dots m$  do
7       Update  $\theta$  with stochastic gradient descent
8        $g_\theta \leftarrow \mathbb{E}_{(x,y) \in B} [\nabla_\theta l(x + \delta, y, \theta)]$ 
9        $g_{adv} \leftarrow \nabla_x l(x + \delta, y, \theta)$ 
10       $\theta \leftarrow \theta - \tau g_\theta$ 
11      Use gradients calculated for the minimization step to update  $\delta$ 
12       $\delta \leftarrow \delta + \epsilon \cdot \text{sign}(g_{adv})$ 
13       $\delta \leftarrow \text{clip}(\delta, -\epsilon, \epsilon)$ 
14    end
15  end
16 end
```

ple 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.

4.4.1 The effect of mini-batch replay on natural training

While the hope for alg. 6 is to build robust models, we still want models to perform well on natural examples. As we increase m in alg. 6, 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

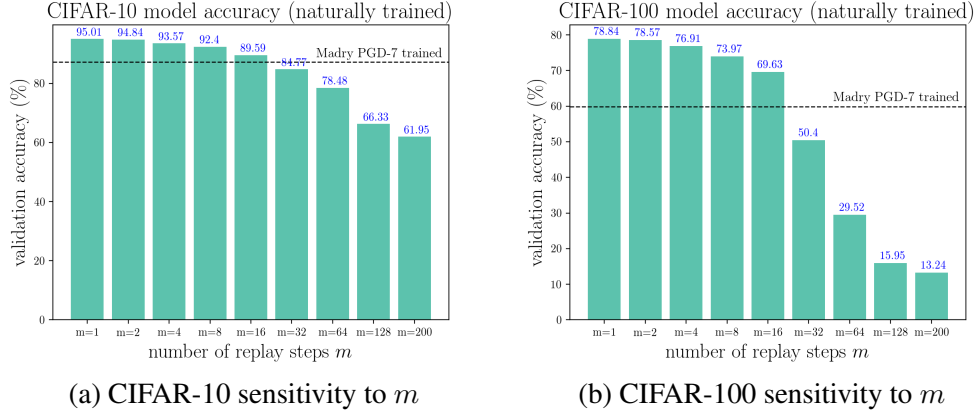


Figure 4.1: Natural validation accuracy of Wide Resnet 32-10 models using varied mini-batch replay parameters m . Here $m = 1$ corresponds to natural training. For large m 's, validation accuracy drops drastically. However, small m 's have little effect. For reference, CIFAR-10 and CIFAR-100 models that are 7-PGD adversarially trained have natural accuracies of 87.25% and 59.87%, respectively.

and CIFAR-100 using different levels of replay. Fig. 4.1 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 [88, 89, 2]. 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 4.5 that good robustness can be achieved using “free” adversarial training with just $m \leq 10$.

4.5 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. 6) and compare them to K-PGD adversarial training²³. We find that free training is able to achieve state-of-the-art robustness on the CIFARs without the overhead of standard PGD training.

4.5.1 CIFAR-10

We train various CIFAR-10 models using the Wide-Resnet 32-10 model and standard hyper-parameters used by [1]. In the proposed method (alg. 6), 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 4.1. 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 [1]⁴, 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) [32]. We test using the PGD-20 attack following [1], and also increase the number of attack iterations and employ random restarts to verify robustness under stronger attacks. To measure the sensitivity of our method to initialization, we perform five trials for the Free- $m = 8$ case and find that our results are insensitive.

²Adversarial Training for Free code for CIFAR-10 in Tensorflow can be found here: https://github.com/ashafahi/free_adv_train/

³ImageNet Adversarial Training for Free code in Pytorch can be found here: <https://github.com/mahyarnajibi/FreeAdversarialTraining>

⁴Results based on the “adv_trained” model in Madry’s CIFAR-10 challenge repo.

Training	Evaluated Against					Train Time (min)
	Nat.	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 4.1: Validation accuracy and robustness of CIFAR-10 models trained with various methods.

Training	Evaluated Against			Training Time (minutes)
	Natural	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 <i>et al.</i> (2-PGD trained)	67.94%	17.08%	16.50%	2053
Madry <i>et al.</i> (7-PGD trained)	59.87%	22.76%	22.52%	5157

Table 4.2: Validation accuracy and robustness of CIFAR-100 models trained with various methods.

The natural accuracy is 85.95 ± 0.14 and robustness against a 20-random restart PGD-20 attack is 46.49 ± 0.19 . 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 in [90]. Gradient-free attacks are superior in settings where 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 reason-

able 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. Recently, a new method called YOPO [91] has been proposed for speeding up adversarial training, in their CIFAR-10 results they use a wider networks (WRN-34-10) with larger batch-sizes (256). As shown in our supplementary, both of these factors increase robustness. To do a direct comparison, we train WRN-34-10 using $m = 10$ and batch-size=256. We match their best reported result (48.03% against PGD-20 attacks for “Free” training *v.s.* 47.98% for YOPO 5-3).

4.5.2 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. 4.4.1, 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 [1], which produces state-of-the-art robust models on CIFAR-10. We summarize the results in table. 4.2.

We see that “free training” exceeds the accuracy on both natural images and adver-



Figure 4.2: 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.

arial 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.

4.6 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.

4.6.0.1 Generative behavior for largely perturbed examples

In [88] it was 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. 4.2 plots “weakly bounded” adversarial examples for the CIFAR-10 7-PGD adversarially trained model [1] 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.

4.6.0.2 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 [92] argues that gradient masking adds little security. We show in fig. 4.3a that free training does not operate by masking gradients using a rough loss surface. In fig. 4.3 we plot the cross-entropy loss projected along two directions in image space for the first few validation examples of CIFAR-10 [93]. In addition to the loss of the free $m = 8$ model, we plot the loss of the 7-PGD adversarially trained model for comparison.

4.7 Robust ImageNet classifiers

ImageNet is a large image classification dataset of over 1 million high-res images and 1000 classes ([27]). 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. [38] first hardened ImageNet classifiers by adversarial training with non-iterative attacks.⁵

⁵Training using a non-iterative attack such as FGSM only doubles the training cost.

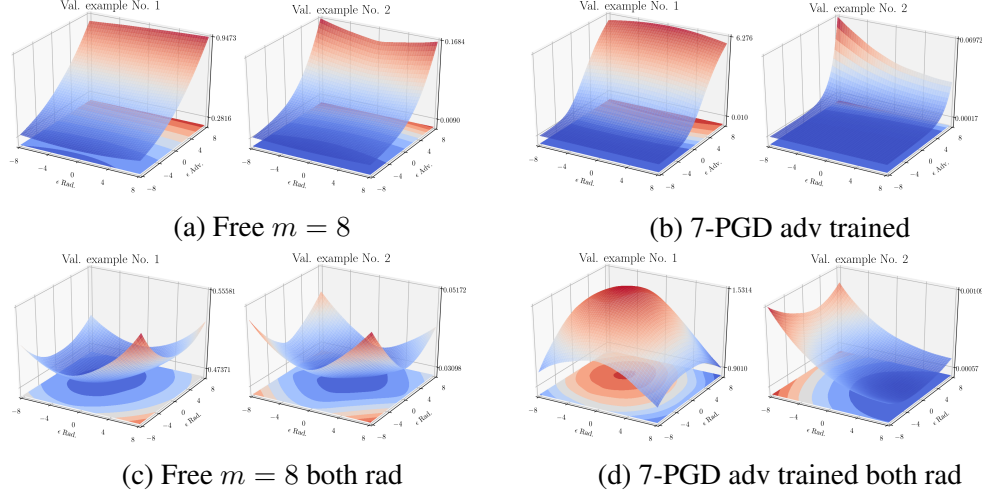


Figure 4.3: The loss surface of a 7-PGD adversarially trained model and our “free” trained model for CIFAR-10 on the first 2 validation images. In (a) and (b) we display the cross-entropy loss projected on one random (Rademacher) and one adversarial direction. In (c) and (d) we display the the cross 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.

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.

Later, [78] 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, [77] 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.

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%

Table 4.3: ImageNet validation accuracy and robustness of ResNet-50 models trained with various replay parameters and $\epsilon = 2$.

4.7.1 Free training results

Our alg. 6 is designed for non-targeted adversarial training. As [34] 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, [90] broke three defenses for $\epsilon = 2$ non-targeted attacks on ImageNet [94, 95, 96], 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 4.3 and fig. 4.4. 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 4.3 shows the robustness of Resnet-50 on ImageNet with $\epsilon = 2$. The validation accuracy for natural images decreases when we increase the minibatch replay m , just like it did for CIFAR in section 4.5.

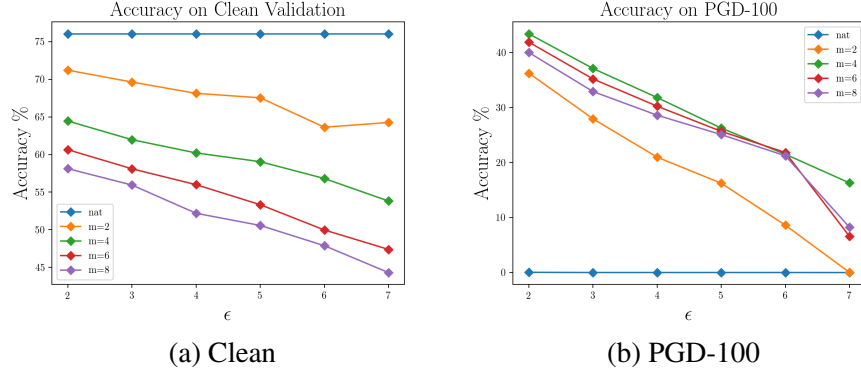


Figure 4.4: The effect of the perturbation bound ϵ and the mini-batch replay hyperparameter m on the robustness achieved by free training.

The naturally trained model is vulnerable to PGD attacks (first row of table 4.3), while free training produces robust models that achieve over 40% accuracy vs PGD attacks ($m = 4, 6, 8$ in table 4.3). 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. 4.4 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 [78].

Model & Training	Evaluated Against				Train time (minutes)
	Natural	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

Table 4.4: 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.**

4.7.2 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 4.4. 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).

4.7.3 Free training on models with more capacity

It is believed that increased network capacity leads to greater robustness from adversarial training [1, 38]. 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-

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%

Table 4.5: Validation accuracy and robustness of free- $m = 4$ trained ResNets with various capacities.

50 is summarized in table 4.5. 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.

4.8 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 5: Adversarially Robust Transfer Learning

5.1 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 [97, 98].

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 [10, 8]. 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 [99]. To make things worse, high-capacity models are often needed to achieve good robustness [1, 38, 100], but these models may over-fit

badly on small datasets.

5.1.1 Contributions

The purpose of this dissertation 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.

5.2 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

$$\max_{\delta} \quad l(x + \delta, y, \theta) \quad \text{s.t.} \quad \|\delta\|_p \leq \epsilon, \quad (5.1)$$

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. 5.1 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 [39], L-BFGS [8], and CW [32].

Many researchers have studied methods for building a robust network which have been later shown to be ineffective when attacked with stronger adversaries [34]. Adversarial training [8] is one of the defenses that was not broken by [34]. While adversarial training using a weak adversary such as the FGSM attack [9] can be broken even by single step attacks which add a simple random step prior to the FGSM step [37], adversarial training using a strong attack has successfully improved robustness. [1] showed that a PGD attack

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

(which is a BIM attack [38] 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 [101] and acceptable robustness on CIFAR-10 [28] classifiers.

[88] 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 [89], [102], and [2].

While, to the best of our knowledge, the transferability of robustness has not been studied in depth, [103] 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 [103] 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 measured in terms of both robustness and clean validation accuracy.

Dataset	model	val. accuracy	acc. on PGD-20	acc. 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%

Table 5.1: Accuracy and robustness of natural and adversarially trained models on CIFAR-10+ and CIFAR-100+. The “+” sign denotes standard data augmentation ($\epsilon = 8$).

5.3 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 [28], 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 [87] for CIFAR-10 from [1] as our robust model for CIFAR-10. We also adversarially train our own robust classifier for CIFAR-100 using the code from [1]. To keep things consistent, we use the same hyper-parameters used by [1] 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 5.1. 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. 5.2. In each experiment, we first re-initialize the k deepest blocks (blocks 1 through k) and then

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.

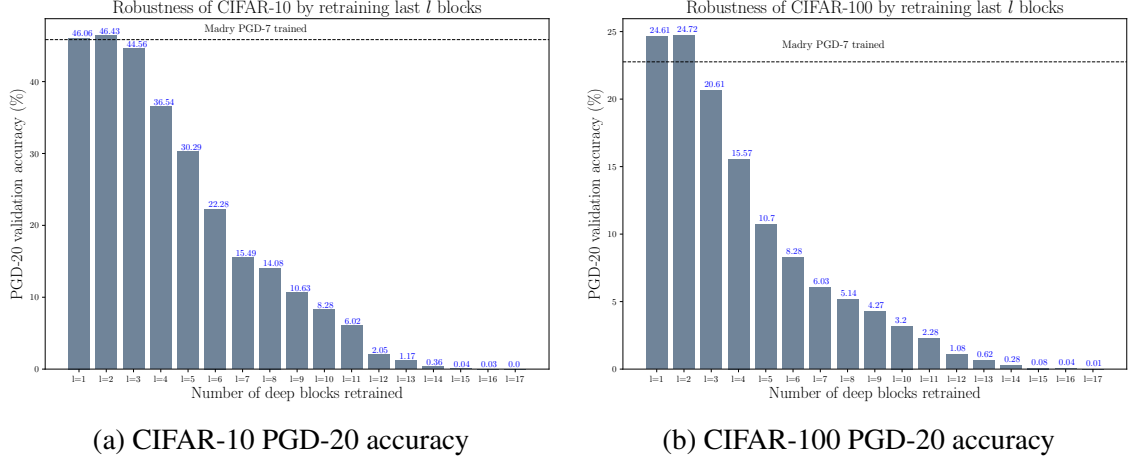


Figure 5.1: Robustness is preserved when we retrain only the deepest block(s) of robust CIFAR-10 and CIFAR-100 models using natural examples. The vertical axis is the accuracy on PGD-20 generated adversarial examples (*i.e.* robustness) after re-training deep layers. The robustness of the adversarially trained models if all layers are frozen are shown with dashed lines.

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. 5.1 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.

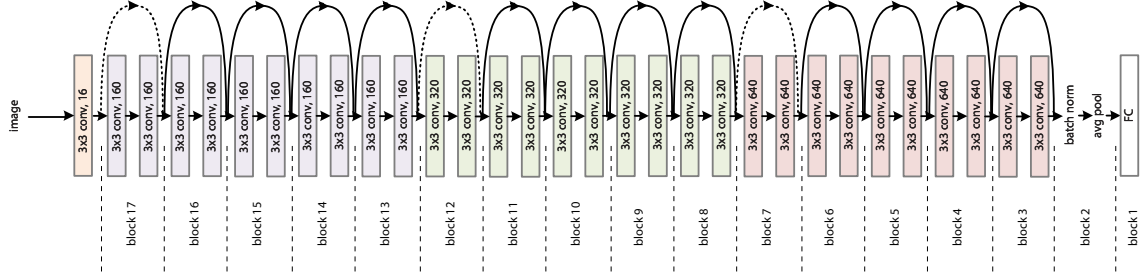


Figure 5.2: Wide Resnet 32-10 and the blocks used for freezing/retraining

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.

5.4 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:

$$\min_w l(z(x, \theta^*), y, w) \quad (5.2)$$

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 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 5.2. 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 5.1 to the first row of Table 5.2). This difference is wider when the source and target data distributions are dissimilar [97].

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 5.2 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 5.2 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.

We make the following observations from the transfer-learning results in Table 5.2.

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

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 $\frac{25.16\%+15.86\%}{2} = 20.51\%$.

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

Table 5.2: Transfer learning by freezing the feature extractor layers ($\epsilon = 8$).

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.

5.4.1 Transfer Learning with ImageNet models

Transfer learning using models trained on ImageNet [27] 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.

5.4.1.1 Transfer learning using ImageNet

Starting from both a natural and robust ImageNet model, we perform the same set of experiments we did in section 5.4. 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 [100], we “free train” a robust ResNet-50 on ImageNet using replay hyper-parameter $m = 4$. The hardened ImageNet classifier withstands attacks bounded by

Arch. & Source Data	Target Dataset	Source	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%	

Table 5.3: Transfer learning from ImageNet ($\epsilon = 5$).

$\epsilon = 5$. Our robust ImageNet achieves 59.05% top-1 accuracy and roughly 27% accuracy against PGD-20 ℓ_∞ $\epsilon = 5$ attacks on validation examples. We experiment with using this robust ImageNet model and a 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 [100] 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 5.3 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.

5.4.2 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. [103] 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 [100]; (2) fine-tuning using adversarial examples of the target task starting from the Free-4 robust ImageNet model similar to [103]; 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 per-

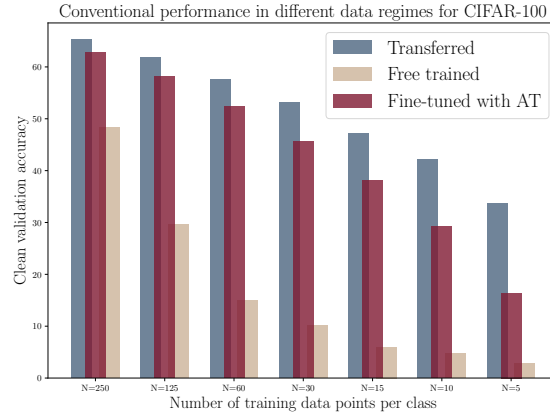
formance $((a)+(b))/2$.) The results are summarized in Fig. 5.3. In the regimes where transfer learning is more common, adversarially robust transfer learning results in the best overall performance. Adversarially/Free training the target domain results in less robustness and validation accuracy compared to fine-tuning which highlights the importance of pre-training [103]. 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 $k = 3$).

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

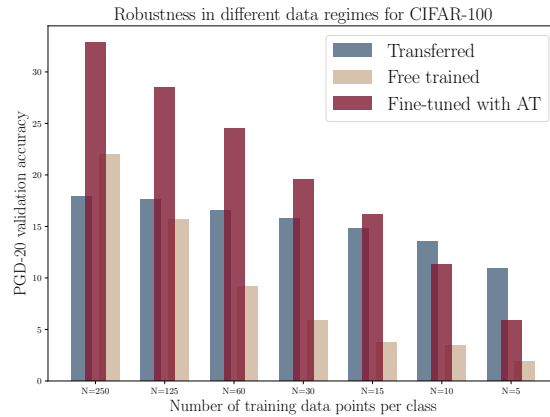
In section 5.4.2, we illustrated that robust transfer learning is most beneficial where we have limited data (i.e., limited number of training data per class). We used CIFAR-100 as an illustrative target dataset. However, the overall results are not data set dependent. When we have limited number of training data, robust transfer learning results in the highest overall performance (see Fig. 5.4 for transferring from a robust ImageNet model to smaller instances of CIFAR-10).

5.4.2.2 Training deeper networks on top of robust feature extractors

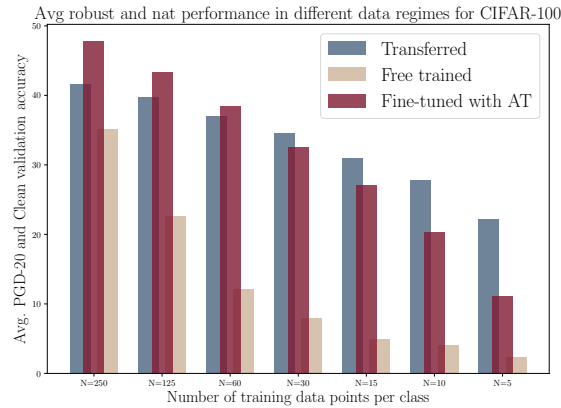
The basic transfer learning setting of section 5.4.1.1 only re-trains one layer for the new task. In section 5.4.1.1, when we transferred from the robust ImageNet to CIFAR-100, the natural *training* accuracy was 88.84%. Given the small number of trainable parameters left for the network ($\approx 2048 \times 100$) and the fixed feature extractor, the network



(a) Clean validation

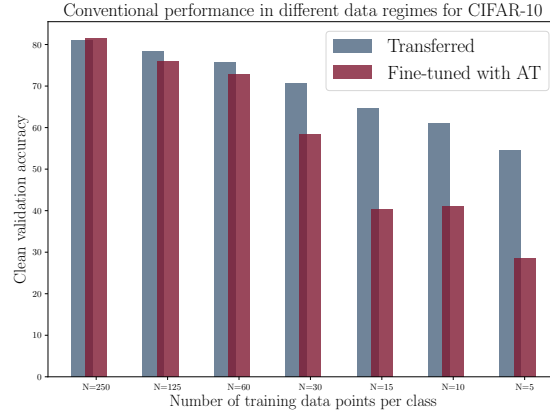


(b) Adversarial validation

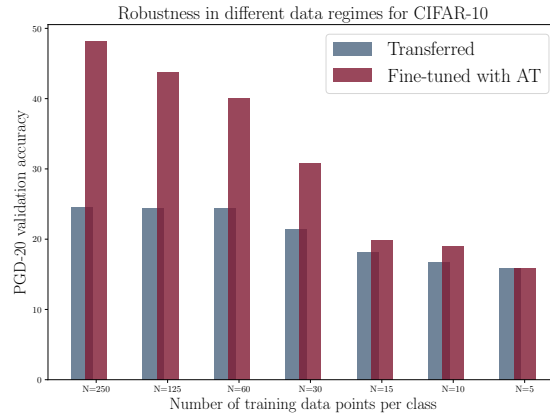


(c) Average validation

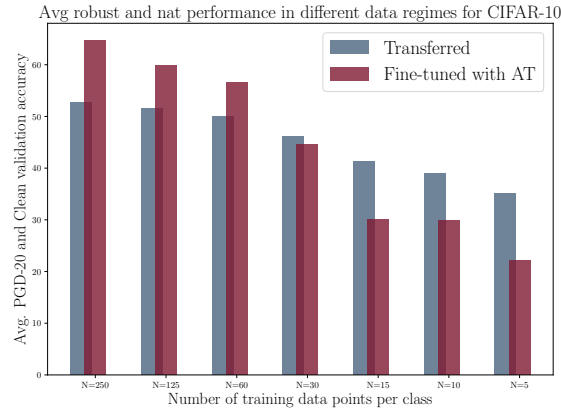
Figure 5.3: When the number of training data per-class is very limited (right bars), adversarially robust transfer learning [Transferred] is better in all metrics. However, as the number of training data increases (left bars), fine-tuning with adversarial examples of the target domain [Fine-tuned with 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.



(a) Clean validation



(b) Adversarial validation



(c) Average validation

Figure 5.4: 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.

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. 5.5 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 [104] (with probability 0.25) and batch-normalization [105].

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 [34]. Therefore, for this case we change our robustness measure and use the

CW attack [32] 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.

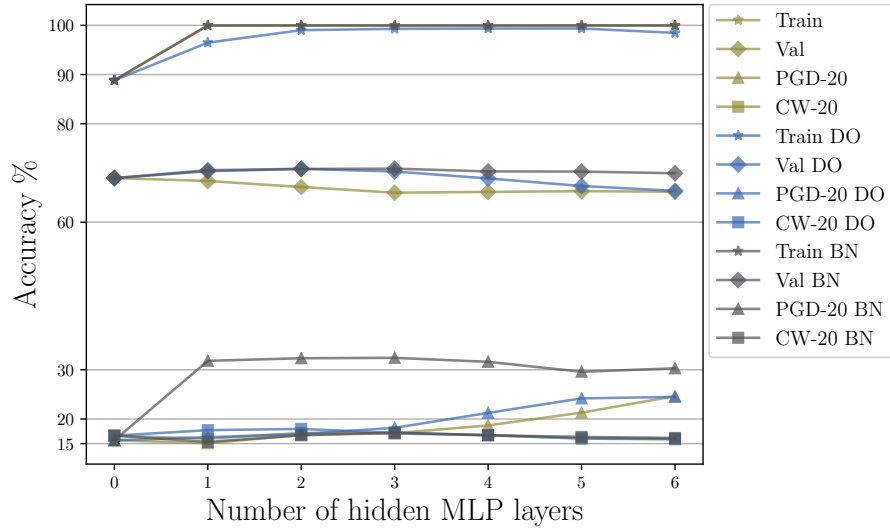


Figure 5.5: Training an MLP for CIFAR-100 on top of the robust feature extractors from ImageNet. The x-axis corresponds to the number of hidden layers (0 is a linear classifier and corresponds to experiments in section 5.4.1.1). Robustness stems from robust feature extractors. Adding more layers on top 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).

5.5 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 into what it means to have a robust feature extractor. Through visualizations, [88] 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. 5.6 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. 5.5).

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.

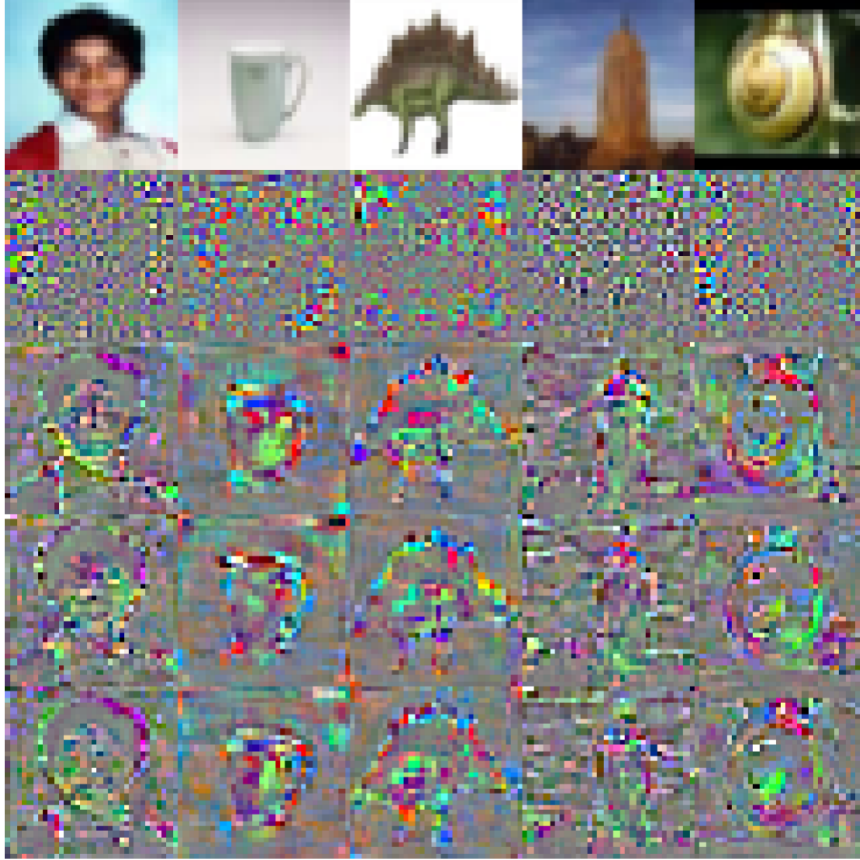


Figure 5.6: Gradients of the loss w.r.t to input images for the CIFAR-100 transfer learning experiments of sections 5.4.1.1 & 5.4.2.2. The top row contains sample CIFAR-100 images. Other rows contain image gradients of the model loss. The second row is for a model transferred from a naturally trained 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.

5.6 End-to-end training without forgetting

As discussed in section 5.4, 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 [88, 89, 2]. 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) [106] 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.

$$\min_{w, \theta} l(z(x, \theta), y, w) + \lambda_d \cdot d(z(x, \theta), z_0(x, \theta_r^*)) \quad (5.3)$$

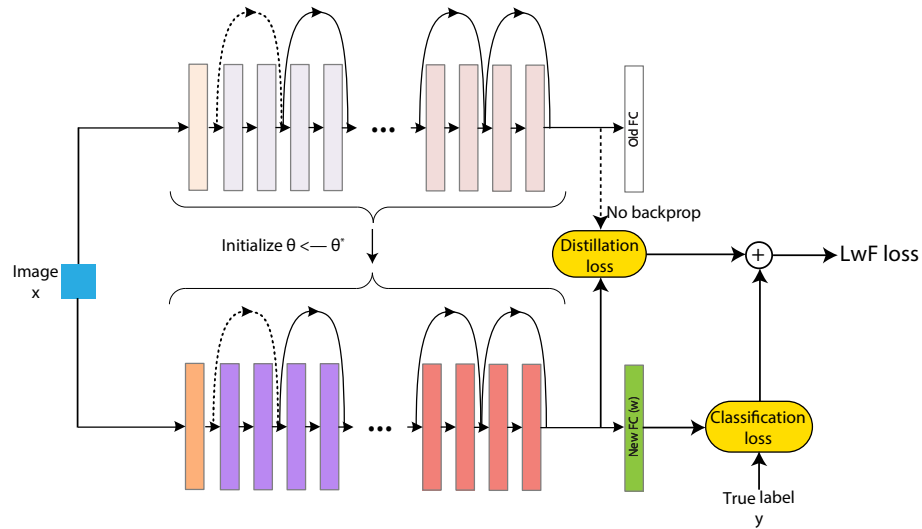


Figure 5.7: Our LwF loss has a term that enforces the similarity of feature representations (*i.e.* penultimate layer activations) between the source model and the fine-tuned model.

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 [107] 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. 5.7). 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) 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, below, we verified that $d(z(x, \theta_r^*), z(x_a, \theta_r^*))$ was not negligible.

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 5.4.⁶

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

⁵We do so since in section 5.3 we found the source of robustness to be the feature extractors and this observation was later reinforced due to the empirical results in section 5.4

⁶Source code for LwF-based experiments: <https://github.com/ashafahi/RobustTransferLWF>

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 5.4. 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.

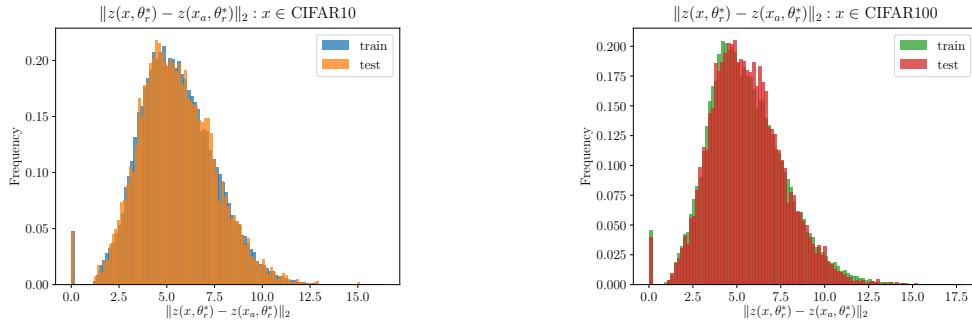
Source $\rightarrow$ Target Dataset	Source Model	λ_d	val.	PGD-20 ($\epsilon = 8$)
CIFAR-100+ $\rightarrow$ CIFAR-10	robust	1e-7	89.07%	0.61%
		0.001	86.15%	4.70%
		0.0025	81.90%	15.42%
		0.005	79.35%	17.61%
		0.01	77.73%	17.55%
		0.1	73.39%	18.62%
CIFAR-100+ $\rightarrow$ CIFAR-10+	natural	NA	83.05%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

Table 5.4: 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.

5.6.1 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 5.8, this quantity is not often negligible.



(a) Histogram of $\|z(x, \theta_r^*), z(x_a, \theta_r^*)\|_2$ for CIFAR-10 dataset, given θ_r^* for CIFAR-100 dataset. The mean for both training and test examples is $\simeq 5.53$

(b) Histogram of $\|z(x, \theta_r^*), z(x_a, \theta_r^*)\|_2$ for CIFAR-100 dataset, given θ_r^* for CIFAR-100 dataset. The mean for both training and test examples is $\simeq 5.57$

Figure 5.8: Figures 5.8a and 5.8b 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.

5.6.2 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 5.5.

Source $\rightarrow$ Target Dataset	Source Model	λ_d	val.	PGD-20	CW-20
CIFAR-100+ $\rightarrow$ 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%

Table 5.5: Decreasing generalization gap by transferring with LwF. For reference, last row shows results from adversarial training CIFAR-100. The ‘+’ sign refers to using augmentation. ($\epsilon = 8$)

5.6.3 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 5.6.

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

In Table 5.7 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

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%

Table 5.6: Decreasing generalization gap by transferring with LwF. For reference, last row shows results from adversarial training CIFAR-10. The ‘+’ sign refers to using augmentation. ($\epsilon = 8$)

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 5.1).

Source $\rightarrow$ Target Dataset	Source	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%

Table 5.7: 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.

5.7 Experiment details

5.7.1 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.

5.7.2 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 $\epsilon = 5$.

5.7.3 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 $\epsilon = 5$.

5.8 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 6: Conclusions and Future Works

In this dissertation, we studied how an adversary can cause deep neural networks to behave how she likes. In particular, we introduced a new class of data poisoning attacks. The proposed attacks use “clean-labels”; they don’t require the attacker to have any control over the labeling of training data. They are also targeted; they control the behavior of the classifier on a *specific* test instance without degrading overall classifier performance. For example, an attacker could add a seemingly innocuous image (that is properly labeled) to a training set for a face recognition engine, and control the identity of a chosen person at test time. Because the attacker does not need to control the labeling function, poisons could be entered into the training set simply by leaving them on the web and waiting for them to be scraped by a data collection bot.

We presented an optimization-based method for crafting targeted clean-label poisons, and showed that just one single poison image can control classifier behavior when transfer learning is used. For full end-to-end training, we presented a “watermarking” strategy that makes poisoning reliable using multiple (≈ 50) poisoned training instances. We demonstrated our method by generating poisoned frog images from the CIFAR dataset and using them to manipulate image classifiers.

We then studied universal perturbations which is an update that can be added to

any image in a broad class of images, while still changing the predicted class label. We studied the efficient generation of universal adversarial perturbations, and also efficient methods for hardening networks to these attacks. We proposed a simple optimization-based universal attack that reduces the top-1 accuracy of various network architectures on ImageNet to less than 20%, while learning the universal perturbation $13\times$ faster than the standard method.

To defend against these universal perturbations, we proposed *universal adversarial training*, which models the problem of robust classifier generation as a two-player min-max game. We solved the min-max problem by alternating stochastic gradient descent. The training time for the robust model is $2\times$ of the natural training. We also proposed a simultaneous stochastic gradient method that is almost free of extra computation which allows us to do universal adversarial training on ImageNet. Both algorithms train models that are robust against universal perturbations. We empirically verified the effectiveness of our attack and defense by testing them on various datasets and architectures.

For improving the efficiency of training robust models that can withstand per-instance perturbations, we presented 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.

Finally, we studied how one can use adversarially robust transfer learning to train a robust model on a target dataset. 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 considered robust transfer learning, in which we transfer not only performance but also robustness from a source model to a target domain. We started by observing that robust networks contain robust feature extractors. By training classifiers on top of these feature extractors, we produced new models that inherit the robustness of their parent networks. We then considered 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.

Going forward, studying other failure modes of neural networks can help both in terms of robustness and also in terms of improving interpretability. Some future directions that are not explored in this work are:

- Studying poisoning attacks for other tasks such as object detection and segmentation.
- Studying poisoning attacks in semi-supervised learning settings where some of the

training data has labels and some doesn't.

- Improving the generalization of cheap methods such as "Adversarial Training for Free!"
- Finding efficient methods for defending against targeted attacks which allow the adversary to have access to larger perturbation budgets.
- Improving the performance on natural validation examples for robust models.
- Improving robustness through architectural changes and other regularization which do not require the explicit generation of adversarial examples on the fly.

Bibliography

- [1] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *ICLR*, 2017.
- [2] Ali Shafahi, W Ronny Huang, Christoph Studer, Soheil Feizi, and Tom Goldstein. Are adversarial examples inevitable? *ICLR*, 2019.
- [3] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.
- [4] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730*, 2017.
- [5] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*, pages 6103–6113, 2018.
- [6] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *CVPR*, pages 1765–1773, 2017.
- [7] Naveed Akhtar, Jian Liu, and Ajmal Mian. Defense against universal adversarial perturbations. *CVPR*, 2018.
- [8] 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.
- [9] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *International Conference on Learning Representation*, 2015.
- [10] 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.

- [11] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 7:1–7:9, Berkeley, CA, USA, 2008.
- [12] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified Defenses for Data Poisoning Attacks. *arXiv preprint arXiv:1706.03691*, (i), 2017.
- [13] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38. ACM, 2017.
- [14] Chaofei Yang, Qing Wu, Hai Li, and Yiran Chen. Generative poisoning attack method against neural networks. *arXiv preprint arXiv:1703.01340*, 2017.
- [15] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted Back-door Attacks on Deep Learning Systems Using Data Poisoning. *arXiv preprint arXiv:1712.05526*, 2017.
- [16] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.
- [17] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojanning attack on neural networks. 2017.
- [18] Octavian Suci, Radu Mărginean, Yiğitcan Kaya, Hal Daumé III, and Tudor Dumitraş. When does machine learning fail? generalized transferability for evasion and poisoning attacks. *arXiv preprint arXiv:1803.06975*, 2018.
- [19] Saeed Mahloujifar and Mohammad Mahmoody. Blockwise p -tampering attacks on cryptographic primitives, extractors, and learners. Cryptology ePrint Archive, Report 2017/950, 2017. <https://eprint.iacr.org/2017/950>.
- [20] Saeed Mahloujifar, Dimitrios I. Diochnos, and Mohammad Mahmoody. Learning under p -tampering attacks. *CoRR*, abs/1711.03707, 2017.
- [21] Ilias Diakonikolas, Daniel M Kane, and Alistair Stewart. Efficient robust proper learning of log-concave distributions. *arXiv preprint arXiv:1606.03077*, 2016.
- [22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385*, 7(3):171–180, 2015.
- [23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *arXiv:1409.4842*, 2014.

- [24] Marco Barreno, Blaine Nelson, Anthony D Joseph, and JD Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.
- [25] Tom Goldstein, Christoph Studer, and Richard Baraniuk. A field guide to forward-backward splitting with a fasta implementation. *arXiv preprint arXiv:1411.3406*, 2014.
- [26] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.
- [27] 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. *International Journal of Computer Vision*, 115(3):211–252, 2015.
- [28] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.
- [29] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *ICLR*, 2014.
- [30] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, pages 427–436, 2015.
- [31] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.
- [32] 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.
- [33] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Xiaolin Hu, Jianguo Li, and Jun Zhu. Boosting adversarial attacks with momentum. *CVPR*, 2017.
- [34] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *ICML*, 2018.
- [35] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, Pascal Frossard, and Stefano Soatto. Analysis of universal adversarial perturbations. *arXiv preprint arXiv:1705.09554*, 2017.

- [36] Julien Perolat, Mateusz Malinowski, Bilal Piot, and Olivier Pietquin. Playing the game of universal adversarial perturbations. *arXiv preprint arXiv:1809.07802*, 2018.
- [37] 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.
- [38] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *ICLR*, 2016.
- [39] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deep-fool: 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.
- [40] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. Learning with a strong adversary. *arXiv preprint arXiv: 1511.03034*, 2015.
- [41] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432*, 2015.
- [42] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifying some distributional robustness with principled adversarial training. *ICLR*, 2018.
- [43] Shumeet Baluja and Ian Fischer. Adversarial transformation networks: Learning to generate adversarial examples. *AAAI*, 2018.
- [44] Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge Belongie. Generative adversarial perturbations. *CVPR*, 2018.
- [45] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. *IJCAI*, 2018.
- [46] Jan Hendrik Metzen, Mummadi Chaithanya Kumar, Thomas Brox, and Volker Fischer. Universal adversarial perturbations against semantic image segmentation. In *ICCV*, 2017.
- [47] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.
- [48] Konda Reddy Mopuri, Utsav Garg, and R Venkatesh Babu. Fast feature fool: A data independent approach to universal adversarial perturbations. *BMVC*, 2017.
- [49] Konda Reddy Mopuri, Aditya Ganeshan, and Venkatesh Babu Radhakrishnan. Generalizable data-free objective for crafting universal adversarial perturbations. *IEEE TPAMI*, 2018.

- [50] Konda Reddy Mopuri, Phani Krishna Uppala, and R Venkatesh Babu. Ask, acquire, and attack: Data-free uap generation using class impressions. In *ECCV*, pages 19–34, 2018.
- [51] Konda Reddy Mopuri, Utkarsh Ojha, Utsav Garg, and R Venkatesh Babu. Nag: Network for adversary generation. In *CVPR*, pages 742–751, 2018.
- [52] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. *ICLR*, 2018.
- [53] 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.
- [54] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM, 2017.
- [55] Chaithanya Kumar Mummadi, Thomas Brox, and Jan Hendrik Metzen. Defending against universal perturbations with shared adversarial training. *arXiv preprint arXiv:1812.03705*, 2018.
- [56] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2014.
- [57] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.
- [58] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint*, 2014.
- [59] 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.
- [60] 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.
- [61] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.
- [62] Ali Shafahi, Amin Ghiasi, Furong Huang, and Tom Goldstein. Label smoothing and logit squeezing: A replacement for adversarial training?, 2019.
- [63] 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.

- [64] 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.
- [65] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *NeurIPS*, pages 2266–2276, 2017.
- [66] Daniel Jakubovitz and Raja Giryes. Improving dnn robustness to adversarial attacks using jacobian regularization. In *ECCV*, pages 514–529, 2018.
- [67] 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.
- [68] Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *ICML*, 2017.
- [69] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. In *NeurIPS*, pages 8400–8409, 2018.
- [70] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *NeurIPS*, pages 10877–10887, 2018.
- [71] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.
- [72] Shiqi Wang, Yizheng Chen, Ahmed Abdou, and Suman Jana. Mixtrain: Scalable training of formally robust neural networks. *arXiv preprint arXiv:1811.02625*, 2018.
- [73] 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.
- [74] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Second-order adversarial attack and certifiable robustness. *CoRR*, abs/1809.03113, 2018.
- [75] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918*, 2019.
- [76] 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.
- [77] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. *CVPR*, 2019.
- [78] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*, 2018.

- [79] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.
- [80] Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. *ICLR*, 2019.
- [81] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. *ICLR*, 2018.
- [82] 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.
- [83] 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.
- [84] Haifeng Qian and Mark N Wegman. L2-nonexpansive neural networks. *arXiv preprint arXiv:1802.07896*, 2018.
- [85] Ali Shafahi, Mahyar Najibi, Zheng Xu, John Dickerson, Larry S Davis, and Tom Goldstein. Universal adversarial training. *arXiv preprint arXiv:1811.11304*, 2018.
- [86] 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.
- [87] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.
- [88] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *ICLR*, 1050:11, 2018.
- [89] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. Theoretically principled trade-off between robustness and accuracy. *ICML*, 2019.
- [90] 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.
- [91] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Painless adversarial training using maximal principle. *arXiv preprint arXiv:1905.00877*, 2019.
- [92] Logan Engstrom, Andrew Ilyas, and Anish Athalye. Evaluating and understanding the robustness of adversarial logit pairing. *arXiv preprint arXiv:1807.10272*, 2018.

- [93] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *NeurIPS*, pages 6389–6399, 2018.
- [94] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.
- [95] 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.
- [96] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.
- [97] 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.
- [98] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.
- [99] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Alexander Madry. Adversarially robust generalization requires more data. In *Advances in Neural Information Processing Systems*, pages 5014–5026, 2018.
- [100] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *arXiv preprint arXiv:1904.12843*, 2019.
- [101] 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.
- [102] 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.
- [103] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. *arXiv preprint arXiv:1901.09960*, 2019.
- [104] 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.
- [105] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

- [106] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2018.
- [107] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.