ABSTRACT

Title of Dissertation: RELIABILITY OF MACHINE LEARNING MODELS IN THE REAL WORLD
Ping-yeh Chiang
Doctor of Philosophy, 2023

Dissertation Directed by: Professor Tom Goldstein
Department of Computer Science

Neural networks have consistently showcased exceptional performance in various applications. Yet, their deployment in adversarial settings is limited due to concerns about reliability. In this paper, we'll first explore methods to verify a model's reliability in diverse scenarios, including classification, detection, auctions, and watermarking. We'll then discuss the challenges and limitations of these verification techniques in real-world situations and suggest potential remedies. We'll wrap up by examining the reliability of neural networks in the context of the model's implicit bias.

Our initial research investigated three critical areas where deep learning model reliability is crucial: object detection, deep auctions, and model watermarking. We found that without rigorous verification, systems could be vulnerable to accidents, manipulation of auction systems, and potential intellectual property theft. To counteract this, we introduced verification algorithms tailored to these respective scenarios.

However, while certificates affirm the resilience of our models within a predefined threat

framework, they don't guarantee real-world infallibility. Hence, in the subsequent section, we explored strategies to improve model's adaptability to domain shifts. While the pyramid adversarial training technique is effective in improving reliability with respect to domain shift, it is very computationally intensive. In response, we devised an alternative technique, universal pyramid adversarial training, which offers comparable advantages while being 30-70% more efficient.

Finally, we try to understand the inherent non-robustness of neural networks through the lens of the model's implicit bias. Surprisingly, we found that the generalization ability of deep learning models comes almost entirely from the architecture and not the optimizer as commonly believed. This architectural bias might be a crucial factor in explaining the inherent non-robustness of neural networks.

Looking ahead, we intend to probe deeper into how neural networks' innate biases can lead to their frailties. Moreover, we posit that refining these implicit biases could offer avenues to enhance model reliability.

RELIABILITY OF MACHINE LEARNING MODELS IN THE REAL WORLD

by

Ping-yeh Chiang

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
2023

Advisory Committee:
Professor Tom Goldstein, Chair/Advisor
Professor Min Wu, Dean's Representative
Professor John Dickerson
Professor Rachael Rudinger
Professor Jia-bin Huang

Acknowledgments

I owe my gratitude to all the people who have made this thesis possible and because of whom my graduate experience has been one that I will cherish forever.

First and foremost I'd like to thank my advisor, Professor Tom Goldstein for giving me an invaluable opportunity to work on challenging and extremely interesting projects over the past five years. It really was the most fun I have ever had in my life.

I'd like to thank all of my colleagues throughout the years. Ahmed Abdelkader, you helped me get my first couple papers into the conference when I was still junior in the program. You continued supporting me when I had some difficulty towards the end of my PhD. You mentorship is invaluable through this journey. Chen Zhu, while we have not collaborated heavily, you have always been a role model for me from afar. I am always inspired by your work ethics, and I often remind myself to work harder because of you.

It is impossible to remember all, and I apologize to those I've inadvertently left out.

Lastly, thank you all and thank God!

Table of Contents

Chapter 1: Introduction

Parts of the introduction were adapted from work described in other chapters, so coauthors credited there also deserve some credit.

Deep learning is becoming increasingly powerful tool for solving real world problem. Deep learning models are now pervasive in areas such as autonomous driving, medical image analysis, and virtual assistant. Despite the success of neural networks for these problems, its use has been limited in safety critical areas due to the lack of reliability in both adversarial scenarios or domain shift.

In this report, I will first describe how we verify neural networks to have desirable reliability in different applications including classification, detection, auctions, and neural network watermarking. In Chapter 2, we discuss how we improve verifiable robustness of classifiers with respect to adversarial patch attack, an attack that is considered more realistic compared to the norm bounded attack. In Chapter 3, we discuss how we adapt prior defenses, which are tailored for classifier, to detectors , which are more frequently found in more sophisticated computer vision systems. In Chapter 4, we discuss how we apply neural verification network technique to certify regret of an auction network. In Chapter 5, we study how to make neural network watermark certifiably unremovable. These four chapters presented various technique to generate certificates for model's reliability with respect to different settings.

While certificates proves resilience of the model with respect to a specified threat model, they do not imply infallibility in the real world, so in Chapter 6, we study model's general robustness with respect to distribution shifts. Specifically, we study the technique pyramid adversarial training [1], which is previously shown to achieve the state of the art performance on out of distribution generalization, but is very expensive computationally. As a result, we propose an alternative technique, universal pyramid adversarial training, which brings similar benefit while reducing the compute budget by 30% to 70%.

At the end, in Chapter 7, we study the non-robustness of neural networks through the lens of the implicit bias of neural network. We found that the implicit bias of neural network arises more from the architecture than the optimizer, a surprising finding that differs from common beliefs within the machine learning community.

Chapter 2: Certified Patch Defenses

2.1 Introduction

Despite the great success of neural networks for vision problems, they are easily fooled by adversarial attacks in which the input to a machine learning model is modified with the goal of manipulating its output. Research in this area is largely focused on norm-bounded attack [2, 3, 4], where the adversary is allowed to perturb *all* pixels in an image provided that the ℓ_p-norm of the perturbation is within prescribed bounds. Other adversarial models were also proposed, such as functional [5], rotation/translation [6], and Wasserstein [7], all of which allow modification to all pixels.

Whole-image perturbations are unrealistic for modeling "physical-world" attacks, in which a real-world object is modified to evade detection. A physical adversary usually modifies an object using stickers or paint. Because this object may only occupy a small portion of an image, the adversary can only manipulate a limited number of pixels. As such, the more practical *patch attack* model was proposed [8]. In a patch attack, the adversary may only change the pixels in a confined region, but is otherwise free to choose the values yielding the strongest attack. The threat to real-world computer vision systems is well-demonstrated in recent literature where carefully crafted patches can fool a classifier with high reliability [8, 9], or make objects invisible to an object detector [10]. Moreover, special glasses with an adversarial frames were used to fool a

face recognition system [11]. In light of such effective physical-world patch attacks, very few defenses are known to date.

In this chapter, we study principled defenses against patch attacks. We begin by looking at existing defenses in the literature that claim to be effective against patch attacks, including Local Gradient Smoothing (LGS) [12] and Digital Watermarking (DW) [13]. Similar to what has been observed for whole-image attacks by [14], we show that these patch defenses are easily broken by stronger adversaries. Concretely, we demonstrate successful white-box attacks, where the adversary designs an attack against a known model, including any pre-processing steps. To cope with such potentially stronger adversaries, we train a robust model that produces a lower-bound on adversarial accuracy. In particular, we propose the first *certifiable* defense against patch attacks by extending interval bound propagation (IBP) defenses [15, 16]. We also propose modifications to IBP training to make it faster in the patch setting. Finally, we study the generalization of certified patch defenses to patches of different shapes, and observe that robustness transfers well across different patch types.

2.2 Problem Setup

We consider a white-box adversary that is allowed to choose the location of the patch (chosen from a set $\mathbb{L}$ of possible locations) and can modify pixels within the particular patch (chosen from the set $\mathbb{P}$) similar to [9]. An attack is successful if the adversary changes the classification of the network to a wrong label. Throughout this chapter, we are interested in the

patch attack robust accuracy (adversarial accuracy for short) as defined by

$$\mathop{\mathbb{E}}_{x\sim X} \min_{p\in\mathbb{P},l\in\mathbb{L}} \mathcal{X}[f(A(x,p,l);\theta) = y], \tag{2.1}$$

where the operator A places the adversarial patch p on a given image x at location l, f is a neural network with parameter θ, X is a distribution of images, and $\mathcal{X}$ is a characteristic function that takes value 1 if its argument is true, and 0 otherwise.

In this model, the strength of the adversary can vary depending on the set of possible patches allowed, and the type of perturbation allowed within the patch. In what follows, we assume the standard setup in which the adversary is allowed any perturbation that maintains pixel intensities in the range $[0, 1]$. Unless otherwise noted, we also assume the patch is restricted to a square of prescribed size. We consider two different options for the set $\mathbb{L}$ of possible patch locations. First, we consider a weak adversary that can only place patches at the corner of an image. We find that even this weak model is enough to break existing patch defenses. Then, we consider a stronger adversary with no restrictions on patch location, and use this model to evaluate our proposed defenses. Note that an adversary, when restricted to modify only a square patch at location l in the image, has the freedom to modify any non-square subset of these pixels. In other words, a certified defense against square patch attacks also provably subverts any non-square patch attack that fits inside a small enough square.

In general, calculating the adversarial accuracy (2.1) is intractable due to non-convexity. Common approaches try to approximate it by solving the inner minimization using a gradient-based method. However, in Section 3, we show that depending on how the minimization is solved, the upper bound could be very loose: a model may appear to be very robust, but fail when faced

with a stronger attack. To side-step the arms race between attacks and defenses, in Section 4, we extend the work of [15] and [16] to train a network that produces a lower bound on adversarial accuracy. We will refer to approximations of the upper bound as *empirical adversarial accuracy* and the lower bound as *certified accuracy*.

2.3 Vulnerability of Existing Defenses

We start by examining existing defense strategies that claim to be effective against patch attacks. Similar to what has been observed for whole-image attacks by [14], we show that these patch defenses can easily be broken by white-box attacks, where the adversary optimizes against a given model including any preprocessing steps.

2.3.1 Existing Defenses

Under our threat model, two defenses have been proposed that each use input transformations to detect and remove adversarial patches.

The first defense is based on the observation that the gradient of the loss with respect to the input image often exhibits large values near the perturbed pixels. In [13], the proposed digital watermarking (DW) approach exploits this behavior to detect unusually dense regions of large gradient entries using saliency maps, before masking them out in the image. Despite a 12% drop in accuracy on clean (non-adversarial) images, this defense method supposedly achieves an empirical adversarial accuracy of 63% for non-targeted patch attacks of size 42×42 (2% of the image pixels), using 400 randomly picked images from ImageNet [17] on VGG19 [18].

The second defense, Local Gradient Smoothing (LGS) by [12] is based on the empirical

observation that pixel values tend to change sharply within these adversarial patches. In other words, the image gradients tend to be large within these adversarial patches. Note that the image gradient here differs from the gradient in [13], the former is with respect the changes of adjacent pixel values and the later is with respect to the classification loss. [12] propose suppressing these adversarial noise by multiplying each pixel with one minus its image gradient as in (2.2). The λ here is a smoothing hyper-parameter. To make their methods more effective, [12] also preprocess the image gradient with a normalization and a thresholding step.

$$\hat{x} = x \odot (1 - \lambda g(x)) \tag{2.2}$$

[12] claim the best adversarial accuracy on ImageNet with respect to patch attacks among all of the defenses we studied. They also claim that their defense is resilient to Backward Pass Differential Approximation (BPDA) from [14], one of the most effective methods to attack models that includes a non-differentiable operator as a preprocessing step.

2.3.2 Breaking Existing Defenses

Using a similar setup as in [12, 13], we are able to mostly replicate the reported empirical adversarial accuracy for Iterative Fast Gradient Sign Method (IFGSM), a common gradient based attack, but we show that when the preprocessing step is taken into account, the empirical adversarial accuracy on ImageNet quickly drops from $\tilde{7}0\%(\tilde{5}0\%)$ for LGS(DW) to levels that are almost the same as random guessing ($\tilde{1}0\%$) as shown in Table 2.1.

Specifically, we break DW [13] by applying BPDA, in which the non-differentiable operator is approximated with an identity mapping during the backward pass. We break LGS [12] by

Table 2.1: Empirical adversarial accuracy of ImageNet classifiers defended with Local Gradient Smoothing and Digital Watermarking. We consider two types of adversaries, one that takes the defense into account during backpropagation and one that does not

		Patch Size		
Attack	Defense	42×42	52×52	60×60
IFGSM	LGS	78%	75%	71%
IFGSM + LGS	LGS	14%	5%	3%
IFGSM	DW	56%	49%	45%
IFGSM + DW	DW	13%	8%	5%

directly incorporating the smoothing step during backpropagation. Even though the windowing and thresholding steps are non-differentiable, the smoothing operator provides enough gradient information for the attack to be effective.

To make sure that our evaluation is fair, we used the exact same models as [13] (VGG19) and [19] (Inception V3). We also consider a weaker set of attackers that can only attack the corners, the same as their setting. Further, we ensure that we were able to replicate their reported result under similar setting.

2.4 Certified Defenses

Given the ease with which these supposedly strong defenses are broken, it is natural to seek methods that can rigorously guarantee robustness of a given model to patch attacks. With such *certifiable* guarantees in hand, we need not worry about an adversary with a stronger optimizer, or a more clever algorithm for choosing patch locations.

2.4.1 Background on certified defenses

Certified defenses have been intensely studied with respect to norm-bounded attacks [15, 16, 20, 21, 22]. In all of these methods, in addition to the prediction model, there is also a verifier. Given a model and an input, the verifier outputs a certificate if it is guaranteed that the image can not be adversarially perturbed. This is done by checking whether there exists any nearby image (within a prescribed ℓ_p distance) with a different label than the image being classified. Alternatively, the verifier may output a lower bound on the distance to the nearest image of a different label. This latter distance is referred to as the *certifiable radius*. Most of these verifiers provide a rather loose bound on the certifiable radius. However, if the verifier is differentiable, then the network can be trained with a loss that promotes tightness of this bound. We use the term *certificate training* to refer to the process of training with a loss that promotes strong certificates.

Interval bound propagation (IBP) [15, 16] is a very simple verifier that uses layer-wise interval arithmetic to produce a certificate. Even though the IBP certificate is generally loose, after certificate training, it yields state-of-the-art certifiably-robust models for l_∞-norm bounded attacks [15, 22]. In this chapter, we extend IBP to train certifiably-robust networks resilient to patch attacks. We first introduce some notation and basic algorithms for IBP training.

Notation We represent a neural network with a series of transformations $h^{(k)}$ for each of its k layers. We use $z^{(k)} \in \mathbb{R}^{n_k}$ to denote the output of layer k, where n_k is the number of units in the k^{th} layer and $z^{(0)}$ stands for the input. Specifically, the network computes

$$z^{(k)} = h^{(k-1)}(z^{(k-1)}) \qquad \forall k = 1, \ldots, K.$$

Certification Problem To produce a certificate for an input x_0, we want to verify that the following condition is true with respect to all possible labels y:

$$(e_{y_{true}} - e_y)^T z^{(K)} = \mathbf{m}_y \geq 0 \qquad \forall z^{(0)} \in \mathbb{B}(x_0) \qquad \forall y. \tag{2.3}$$

Here, e_i is the i^{th} basis vector, and $\mathbf{m}_y$ is called the margin following [21]. Note that $\mathbf{m}_{y_{true}}$ is always equal to 0. The vector $\mathbf{m}$ contains all margins corresponding to all labels. $\mathbb{B}(x_0)$ is the constraint set over which the adversarial input image may range. In a conventional setting, this is an ℓ_∞ ball around x_0. In the case of patch attack, the constraint set contains all images formed by applying a patch to x_0;

$$\mathbb{B}(x_0) = \{A(x_0, p, l) | p \in \mathbb{P} \text{ and } l \in \mathbb{L}\}. \tag{2.4}$$

The Basics of Interval Bound Propagation (IBP) We now describe how to produce certificates using interval bound propagation as in [15]. Suppose that for each component in $z^{(k-1)}$ we have an interval containing all the values which this component reaches as $z^{(0)}$ ranges over the ball $\mathbb{B}(x_0)$. If $z^{(k)} = h^{(k)}(z^{(k-1)})$ is a linear (or convolutional) layer of the form $z^{(k)} = W^{(k)} z^{(k-1)} + b^{(k)}$, then we can get an *outer approximation* of the reachable interval range of activations by the next layer $z^{(k)}$ using the formulas below

$$\overline{z}^{(k)} = W^{(k)} \frac{\overline{z}^{(k-1)} + \underline{z}^{(k-1)}}{2} + |W^{(k)}| \frac{\overline{z}^{(k-1)} - \underline{z}^{(k-1)}}{2} + b^{(k)}, \tag{2.5}$$

$$\underline{z}^{(k)} = W^{(k)} \frac{\overline{z}^{(k-1)} + \underline{z}^{(k-1)}}{2} - |W^{(k)}| \frac{\overline{z}^{(k-1)} - \underline{z}^{(k-1)}}{2} + b^{(k)}. \tag{2.6}$$

Here $\overline{z}^{(k-1)}$ denotes the upper bound of each interval, $\underline{z}^{(k-1)}$ the lower bound, and $|W^{(k)}|$ the element-wise absolute value. Alternatively, if $h^{(k)}(z^{(k-1)})$ is an element-wise monotonic activation (e.g., a ReLU), then we can calculate the *outer approximation* of the reachable interval range of the next layer using the formulas below.

$$\overline{z}^{(k)} = h^{(k)}(\overline{z}^{(k-1)}) \tag{2.7}$$

$$\underline{z}^{(k)} = h^{(k)}(\underline{z}^{(k-1)}). \tag{2.8}$$

When the feasible set $\mathbb{B}(x_0)$ represents a simple ℓ_∞ attack, the range of possible $z^{(0)}$ values is trivially characterized by an interval bound $\overline{z}^{(0)}$ and $\underline{z}^{(0)}$. Then, by iteratively applying the above rules, we can propagate intervals through the network and eventually get $\overline{z}^{(K)}$ and $\underline{z}^{(K)}$. A certificate can then be given if we can show that (2.3) is always true for outputs in the range $\overline{z}^{(K)}$ and $\underline{z}^{(K)}$ with respect to all possible labels. More specifically, we can check that the following holds for all y

$$\underline{\mathbf{m}}_y = e_{y_{true}}^T \underline{z}^{(K)} - e_y^T \overline{z}^{(K)} = \underline{z}_{y_{true}}^{(K)} - \overline{z}_y^{(K)} \geq 0 \quad \forall y \tag{2.9}$$

Training for Interval Bound Propagation To train a network to produce accurate interval bounds, we simply replace standard logits with the $-\underline{\mathbf{m}}$ vector in (2.3). Note that all elements of $\underline{\mathbf{m}}$ need to be larger than zero to satisfy the conditions in (2.3), and $\underline{\mathbf{m}}_{y_{true}}$ is always equal to zero. Put simply, we would like $\underline{\mathbf{m}}_{y_{true}} = 0$ to be the least of all margins. We can promote this

condition by training with the loss function

$$\text{Certificate Loss} = \text{Cross Entropy Loss}(-\underline{\mathbf{m}}, y). \tag{2.10}$$

Unlike regular neural network training, stochastic gradient descent for minimizing equation 2.10 is unstable, and a range of tricks are necessary to stabilize IBP training [15]. The first trick is merging the last linear weight matrix with $(e_y - e_{y_{true}})$ before calculating $-\underline{\mathbf{m}}_y$. This allows a tighter characterization of the interval bound that noticeably improves results. The second trick uses an "epsilon schedule" in which training begins with a perturbation radius of zero, and this radius is slowly increased over time until a sentinel value is reached. Finally, a mixed loss function containing both a standard natural loss and an IBP loss is used.

In all of our experiments, we use the merging technique and the epsilon schedule, but we do not use a mixed loss function containing a natural loss as it does not increase our certificate performance.

2.4.2 Certifying against patch attacks

We can now describe the extension of IBP to patches. If we specify the patch location, one can represent the feasible set of images with a simple interval bound: for pixels within the patch, the upper and lower bound is equal to 1 and 0. For pixels outside of the patch, the upper and lower bounds are both equal to the original pixel value. By passing this bound through the network, we would be able to get $\underline{\mathbf{m}}^{\text{single location}}$ and verify that they satisfy the conditions in (2.3).

However, we have to consider not just a single location, but all possible locations $\mathbb{L}$ to give a certificate. To adapt the bound to all possible location, we pass each of the possible patches

through the network, and take the worst case margin. More specifically,

$$\underline{\mathbf{m}}^{\text{es}}(\mathbb{L})_y = \min_{l \in \mathbb{L}} \underline{\mathbf{m}}^{\text{single patch}}(l)_y \quad \forall y. \tag{2.11}$$

Similar to regular IBP training, we simply use $\underline{\mathbf{m}}^{\text{es}}(\mathbb{L})$ to calculate the cross entropy loss for training and backpropagation,

$$\text{Certificate Loss} = \text{Cross Entropy Loss}(-\underline{\mathbf{m}}^{\text{es}}(\mathbb{L}), y). \tag{2.12}$$

Unfortunately, the cost of producing this naïve certificate increases quadratically with image size. Consider that a CIFAR-10 image is of size 32×32, requiring over a thousand interval bounds, one for each possible patch location. To alleviate this problem, we propose two certificate training methods: *Random Patch* and *Guided Patch*, so that the number of forward passes does not scale with the dimension of the inputs.

Random Patch Certificate Training In this method, we simply select a random set of patches out of the possible patches and pass them forward. A level of robustness is achieved even though a very small number of random patches are selected compared to the total number of possible patches

$$\underline{\mathbf{m}}^{\text{random patches}}(\mathbb{L})_y = \underline{\mathbf{m}}^{\text{es}}(S)_y \tag{2.13}$$

where S is a random subset of $\mathbb{L}$. Similarly, the random patch certificate loss is calculated as

below.

$$\text{Random Patch Certificate Loss} =$$

$$\text{Cross Entropy Loss}(-\underline{\mathbf{m}}^{\text{random patches}}(\mathbb{L}), y) \tag{2.14}$$

Guided Patch Certificate Training In this method, we propose using a U-net [23] to predict $\underline{\mathbf{m}}^{\text{single patch}}$, and then randomly select a couple of locations based on the predicted $\underline{\mathbf{m}}^{\text{single patch}}$ so that fewer patches need to be passed forward.

Note that very few patches contribute to the worst case bound $\underline{\mathbf{m}}^{\text{es}}$ in (2.11). In fact, the number of patches that yield the worst case margins will be no more than the number of labels. If we know the worst-case patches beforehand, then we can simply select the few worst-case patches during training.

We propose to use U-net as the number of locations and margins is very large. For a square patch of size $n \times n$ and an image of size $m \times m$, the total number of possible locations is $(m - n + 1)^2$, and for each location the number of margins is equal to the number of possible labels.

$$\underline{\mathbf{m}}^{\text{pred}} = \text{U-net(image)} \tag{2.15}$$

$$\dim(\underline{\mathbf{m}}^{\text{pred}}) = (m - n + 1, m - n + 1, \#\text{ of labels}).$$

Given the U-net prediction of $\underline{\mathbf{m}}^{\text{pred}}$, we then randomly select a single patch for each label based on the softmax of the predicted $\underline{\mathbf{m}}^{\text{pred}}$. The number of selected patches is equal to the number of labels. After these patches are passed forward, the U-net is then updated with a mean-

squared-error loss between the predicted margins $\underline{\mathbf{m}}^{\text{pred}}$ and the actual margins $\underline{\mathbf{m}}^{\text{actual}}$. Note that only a few patches are selected at a time, so that the mean-squared-error only passes through the selected patches.

$$\text{U-net Loss} = \text{MSE}(\underline{\mathbf{m}}^{\text{pred}}, \underline{\mathbf{m}}^{\text{actual}}). \tag{2.16}$$

The network is trained with the following loss:

$$\begin{aligned} &\text{Guided Patch Certificate Loss} = \\ &\text{Cross Entropy Loss}(-\underline{\mathbf{m}}^{\text{guided patches}}(\mathbb{L}), y). \end{aligned} \tag{2.17}$$

Certification Process In all our experiments, we check that equation (2.3) is satisfied by iterating over all possible patches and forward-passing the interval bounds generated for each patch; this overhead is tolerable at evaluation time.

2.5 Experiments

In this section, we compare our certified defenses with exiting algorithms on two datasets and three model architectures of varying complexity. We consider a strong attack setting in which adversarial patches can appear anywhere in the image. Different training strategies for the certified defense are also compared, which shows a trade-off between performance and training efficiency. Finally, we evaluate the transferability of a model trained using square patches to other adversarial shapes, including shapes that do not fit in any certified square. The training and architectural details can be found in Appendix 2.7.1.

2.5.1 Comparison against existing defenses

In this section, we study the effectiveness of our proposed IBP certified models against an adversary that is allowed to place patches anywhere in the image, even on top of the salient object. If the patch is sufficiently small, and does not cover a large portion of the salient object, then the model should still classify correctly, and defense against the perturbation should be possible.

In the best case, our IBP certified model is able to achieve 91.6% certified (Table 2.2) with respect to a 2×2 patch ($\sim$ .5% of image pixels) adversary on MNIST. For more challenging cases, such as a 5 $\times$ 5 ($\sim$ 2.5% of image pixels) patch adversary on CIFAR-10, the certified adversarial accuracy is only 24.9% (Table 2.2). Even though these existing defenses appear to achieve better or comparable adversarial accuracy as our IBP certified model when faced with a weak adversary, when faced with a stronger adversary their adversarial accuracy dropped to levels below our certified accuracy for all cases that we analyzed.

When evaluating existing defenses, we only report cases where non-trivial adversarial accuracy is achieved against a weaker adversary. We do not explore cases where LGS and DW perform so poorly that no meaningful comparison can be done. LGS and DW are highly dependent on hyperparameters to work effectively against naive attacks, and yet neither [12] nor [13] proposed a way to learn these hyperparameters. By trial and error, we were able to increase the adversarial accuracy against a weaker adversary for some settings, but not all. In addition, we also notice a peculiar feature of DW: when we increase the adversarial accuracy, the clean accuracy degrades, sometimes so much that it is even lower than the empirical adversarial accuracy. This happens because DW always removes a patch from the prediction. When an adversarial patch is detected, it is likely to be removed, enabling correct prediction. On the other

hand, when there are no adversarial patches, DW removes actual salient information, resulting in lower clean accuracy.

Here we did not compare our results with adversarial training, because even though it produces some of the most adversarially robust models, it does not offer any guarantees on the empirical robust accuracy, and could still be decreased further with stronger attacks. For example, [24] proposed a stronger attack that could find 47% more adversarial examples compared to gradient based method. Further, adversarial training on all possible patches would be even more expensive compared to certificate training, and is slightly beyond our computational budget.

Compared to state-of-the-art certified models for CIFAR with L_∞-perturbation, where [25] proposed a deterministic algorithm that achieves clean accuracy of 34.0%, our clean accuracy for our most robust CIFAR 5×5 model is 47.8% when using a large model (Table 2.2).

2.5.2 Comparison of training strategies

We find that given a fixed architecture all-patch certificate training achieves the best certified accuracy. However, given a fixed computational budget, random and guided training significantly outperform all-patch training. Finally, guided-patch certificate training consistently outperforms random-patch certificate training by a slim margin, indicating that the U-net is learning how to predict the minimum margin $\underline{\mathbf{m}}$.

In Table 2.3, we see that given a fixed architecture all-patch certificate training significantly outperforms both random-patch certificate training and guided-patch certificate training in terms of certified accuracy, outperforming the second best certified defenses in each task by 2.6% (MNIST, 2×2), 7.3% (MNIST, 5×5), 3.9% (CIFAR-10, 2×2), and 3.4% (CIFAR-10, 5

Table 2.2: Comparison of our IBP certified patch defense against existing defenses. Empirical adversarial accuracy is calculated for 400 random images in both datasets. All results are averaged over three different models.

Dataset	Patch Size	Adversary	Defense	Clean Accuracy	Empirical Adversarial Accuracy	Certified Accuracy
MNIST	2×2	IFGSM	None	98.4%	80.1%	-
	2×2	IFGSM	LGS	97.4%	90.0%	-
	2×2	IFGSM + LGS	LGS	97.4%	60.7%	-
	2×2	IFGSM	IBP	98.5%	93.9%	91.6%
	5×5	IFGSM	None	98.5%	3.3%	-
	5×5	IFGSM	IBP	92.9%	66.1%	62.0%
CIFAR	2×2	IFGSM	None	66.3%	25.4%	-
	2×2	IFGSM	LGS	64.9%	31.3%	-
	2×2	IFGSM + LGS	LGS	64.9%	24.2%	-
	2×2	IFGSM	DW	47.1%	43.3%	-
	2×2	IFGSM + DW	DW	47.1%	20.2%	-
	2×2	IFGSM	IBP	48.6%	45.2%	41.6%
	5×5	IFGSM	None	66.5%	0.4%	-
	5×5	IFGSM	LGS	51.2%	22.11%	-
	5×5	IFGSM + LGS	LGS	51.2%	0.5%	-
	5×5	IFGSM	DW	45.3%	59.3%	-
	5×5	IFGSM + DW	DW	45.3%	15.6%	-
	5×5	IFGSM	IBP	33.9%	29.1%	24.9%

$\times$ 5). However, all-patch certificate training is very expensive, taking on average 4 to 15 times longer than guided-patch certificate training and over 30 to 70 times longer than random-patch certificate training.

On the other hand, given a limited computational budget, random-patch and guided-patch training significantly outperforms all-patch training. Due to the efficiency of random-patch and guided-patch training, they scale much better to large architectures. By switching to a large architecture (5 layer wide convolutional network), we are able to boost the certified accuracy by over 10% compared to the best performing all-patch small model (Table 2.2). Note that we are unable to all-patch train the same architecture as it will take almost 15 days to complete, and is

Table 2.3: Trade-off between certified accuracy and training time for different strategies. The numbers next to training strategies indicate the number of patches used for estimating the lower bound during training. Most training times are measured on a single 2080Ti GPU, with the exception of all-patch training which is run on four 2080Ti GPUs. For that specific case, the training time is multiplied by 4 for fair comparison. See Appendix 2.7.5 for more detailed statistics. *indicates the performance of the best performing large model trained with either random or guided patch. Detailed performance of the large models can be found in Appendix 2.7.4

		2×2			5×5		
Dataset	Training Strategy	Clean Accuracy	Certified Accuracy	Training Time(h)	Clean Accuracy	Certified Accuracy	Training Time(h)
MNIST	All Patch	98.5%	91.5%	9.3	92.0%	60.4%	8.4
	Random(1)	98.5%	82.9%	0.2	96.9%	24.1%	0.4
	Random(5)	98.6%	86.6%	0.3	95.8%	42.1%	0.3
	Random(10)	98.6%	87.7%	0.3	95.6%	49.6%	0.3
	Guided(10)	98.6%	88.9%	2.2	95.0%	53.1%	2.6
CIFAR	All Patch	50.9%	39.9%	56.4	33.5%	22.0%	45.8
	Random(1)	53.6%	21.6%	0.6	43.6%	6.1%	0.6
	Random(5)	52.9%	32.3%	0.7	39.0%	14.6%	0.7
	Random(10)	51.9%	35.6%	0.8	38.8%	18.6%	0.8
	Guided(10)	52.4%	36.0%	3.7	37.9%	18.8%	3.7
	Large Model*	**65.8%**	**51.9%**	22.4	**47.8%**	**30.3%**	15.4

out of our computational budget.

Guided-patch certificate training is slightly more expensive compared to random patch, due to overhead from the U-net architecture. However, given the 10 patches picked, guided-patch certificate training consistently outperforms random-patch certificate training, indicating that the U-net is learning how to predict the minimum margin $\underline{\mathbf{m}}$.

2.5.3 Transferability to patches of different shapes

Since real-world adversarial patches may not always be square, the robust transferability of the model to shapes other than the square is important.

Table 2.4: Certified accuracy for square-patch trained model for different shapes

Dataset	Pixel Count	Square	Rectangle	Line	Diamond	Parallelogram
MNIST	4	91.6%	-	92.5%	91.6%	92.3%
	16	69.4%	55.4%	46.7%	68.13%	70.2%
	25	59.7%	50.9%	32.4%	53.6%	55.2%
CIFAR	4	50.8%	-	46.1%	48.6%	49.8%
	16	36.9%	29.0%	32.1%	35.7%	36.3%
	25	30.3%	25.1%	29.0%	30.1%	30.7%

Therefore, we evaluate the robustness of the square-patch-trained model to adversarial patches of different shapes while fixing the number of pixels. In all these experiments, we evaluate the certified accuracy for our largest model, on both MNIST and CIFAR datasets. We evaluate the transferability to various shapes including rectangle, line, parallelogram, and diamond. With the exception of rectangles, all the shapes have the exact same pixel count as the patches used for training. For rectangles, we use multiple choices of width and length, obtaining some combinations with slightly more pixels, and the worst accuracy is reported in Table 2.4. The exact shapes used can be found in Appendix 2.7.2.

The certified accuracy of our models generalize surprisingly well to other shapes, losing no more than than 5% in most cases for MNIST and no more than 6% for CIFAR-10 (Table 2.4). The largest degradation of accuracy happens for rectangles and lines, and it is mostly because the rectangle considered has more pixels compared to the square, and the line has less overlaps. However, it is still interesting that the certificate even generalizes to a straight line, even though the model was never trained to be robust to lines. In the case of MNIST with small patch size, the certified accuracy even improves when transferred to lines.

2.6 Chapter Summary and Future Work

In this chapter, we established the vulnerability of known defenses against adversarial patch attacks. To remedy this, we proposed the first certified defense against patch attacks. We demonstrated the effectiveness of our certified defense on two datasets, and proposed two training strategies to speed it up while trading-off efficiency and robustness. Finally, we considered the robust transferability of our trained certified models to different shapes, obtaining good generalization. In its current form, the proposed certified defense is unlikely to scale to ImageNet, and we hope the presented experiments will encourage further work along this direction.

2.7 Chapter Appendix

2.7.1 Experimental Settings and Network Structure

We evaluate the proposed certified patch defense on three neural networks: a multilayer perceptron (MLP) with one 255-neuron hidden layer, and two convolutional neural networks (CNN) with different depths. The small CNN has two convolutional layers (kernel size 4, stride 2) of 4 and 8 output channels each, and a fully connected layer with 256 neurons. The large CNN has four convolutional layers with kernel size (3, 4, 3, 4), stride (1, 2, 1, 2), output channels (4, 4, 8 ,8), and two fully connected layer with 256 neurons. We run experiments on two datasets, MNIST and CIFAR10, with two different patch sizes 2×2 and 5×5. For all experiments, we are using Adam [26] with a learning rate of $5e-4$ for MNIST and $1e-3$ for CIFAR10, and with no weight decay. We also adopt a warm-up schedule in all experiments like [22], where the input interval bounds start at zero and grow to [-1,1] after 61/121 epochs for MNIST/CIFAR10

respectively. We train the models for a total of 100/200 epochs for MNIST/CIFAR10, where in the first 61/121 epochs the learning rate is fixed and in the following epochs, we reduce the learning rate by one half every 10 epochs.

In addition, following [15], we further evaluate the CIFAR10 on a larger model which has 5 convolutional layers with kernel size 3 and a fully connected layer with 512 neurons. This deeper and wider model achieves the clean accuracy around 89%, and has 17M parameters in total. Table 2.6 in Appendix 2.7.4 describes the full certified patch results for this large model.

2.7.2 Sample shapes for generalization experiments

We demonstrate generalization to other patch shapes that were not considered in training, obtaining surprisingly good transfer in robust accuracy; see the figure below and the results in Table 2.4.

2.7.3 Bound pooling

Besides random-patch certificate training and guided-patch certificate training, we also experimented with the idea of bound pooling. All-patch training is very expensive as bounds generated by each potential patch has to be forward passed through the *complete* network. Bound pooling partially relieves the problem be pooling the interval bounds in intermediate layers thus reducing the forward pass in subsequent layers.

Specifically, given a set of patches $\mathbb{P}$, the interval bounds in the ith layer are $\bar{Z}^{(i)}(\mathbb{P}) = \{\bar{z}^{(i)}(p) : p \in \mathbb{P}\}$ and $\underline{Z}^{(i)}\mathbb{P} = \{\underline{z}^{(i)}(p) : p \in \mathbb{P}\}$. We can reduce the number of interval bounds by partitioning $\mathbb{P}$ into n subsets $\{\mathbb{S}^1, ..., \mathbb{S}^n\}$ and calculate a new set of bounds $\bar{Z}^{(i)}_{pool}(\mathbb{P}) =$

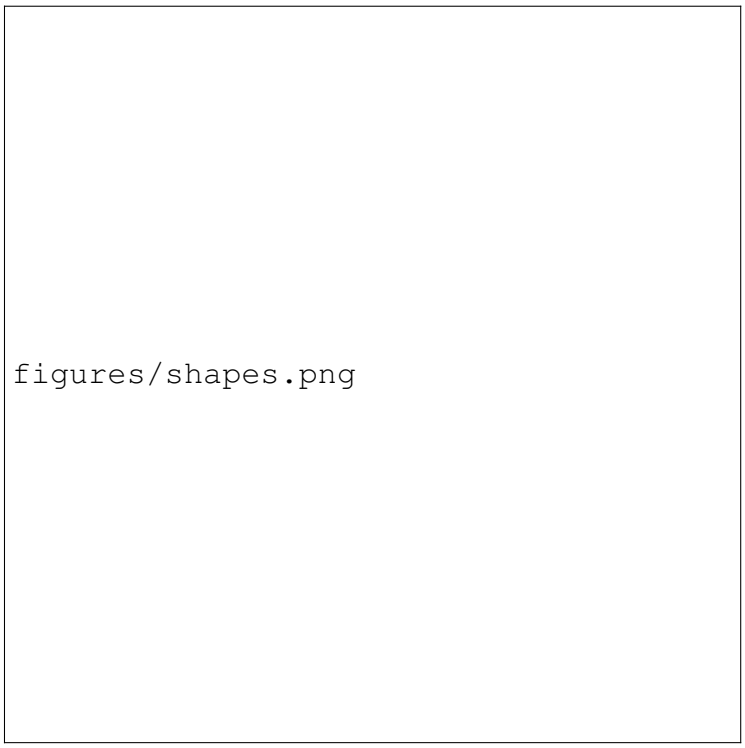


Figure 2.1: Examples of shapes with pixels number 4 and 25. From left to right are square, parallelogram, diamond and rectangle (line) respectively.

$\{\max_{p\in\mathbb{S}_i} \bar{z}^{(i)}(p) : i \in [n]\}$ and $\underline{Z}^{(i)}_{pool}(\mathbb{P}) = \{\min_{p\in\mathbb{S}_i} \underline{z}^{(i)}(p) : i \in [n]\}$. Depending on how $\mathbb{P}$ is partitioned, the bound pooling would work differently. In our experiments, we always select adjacent patches for each $\mathbb{S}_i$ with the assumption that adjacent patches tend to generate similar bounds thus resulting in tighter certificate.

Bound pooling, similar to random- and guided- patch training, trades performance for efficiency compared to all-patch certificate training. However, the trade off is not as favorable compared to random-patch and guided-patch training. For example, in Table 2.5, Pooling 16 (4×4) patches in the first layer reduces training time by 35% while loosing 0.7% in performance (on MNIST 2×2), but a similar level of performance can be achieved with guided-patch training with almost 90% reduction in training time. The trade off becomes greater when the model becomes larger. Overall, bound pooling is still quite expensive, and cannot scale to larger models

like random-patch or guided-patch training.

Table 2.5: Comparing bound pooling with the guided-patch and random-patch training. Pool 4 means that the adjacent 4×4 patches (16 patches) are pooled together in the first layer. Pool 2-2 means that the adjacent 2×2 bounds are pooled together in the first layer and then another 2×2 bound pooling happens at the second layer. This is similar to 4×4 pooling except the pooling operation is distributed between the first and second layer. All experiments are performed on a 4-layer convolutional network.

		2×2			5×5		
Dataset	Training Strategy	Clean Accuracy	Certified Accuracy	Training Time(h)	Clean Accuracy	Certified Accuracy	Training Time(h)
MNIST	All Patch	98.5%	91.6%	20.1	90.0%	59.7%	16.3
	Pool 2	98.0%	91.1%	15.8	85.2%	54.2%	11.6
	Pool 4	97.2%	89.9%	13.2	70.4%	38.3%	10.2
	Random(1)	98.5%	81.9%	0.3	96.8%	24.8%	0.4
	Random(5)	98.6%	86.5%	0.3	94.9%	42.0%	0.5
	Random(10)	98.6%	87.5%	0.5	94.7%	50.4%	0.6
	Guided(10)	98.7%	88.9%	2.2	94.0%	53.2%	3.4
CIFAR	All Patch	49.6%	41.6%	22.5	34.0%	25.0%	18.6
	Pool 2	48.1%	39.4%	17.3	32.4%	24.2%	14.5
	Pool 4	44.9%	37.1%	16.3	28.3%	20.6%	13.6
	Pool 2-2	45.0%	37.4%	16.5	25.3%	19.1%	13.8
	Random(1)	53.2%	32.4%	0.6	42.7%	11.0%	0.6
	Random(5)	52.2%	39.5%	0.9	37.8%	19.6%	0.9
	Random(10)	50.8%	38.6%	1.0	38.4%	21.9%	1.0
	Guided(10)	53.0%	39.8%	4.0	36.1%	23.0%	3.9

2.7.4 Training with larger models

Recall that all-patch training considers all possible patches during training, which can be too expensive for larger models and/or images. The proposed random- and guided-patch training methods aim to reduce the training cost by considering only a subset of patches; please see Section 2.4.2 for more details.

Table 2.6: The random and guided training strategy could yield significantly stronger model compared to all-patch training given a fixed computational budget. The random and guided training strategy allows us to train a larger model that would be infeasible to train otherwise. The guided-patch large model is able to boost the certified accuracy by over 10% compared to the best performing all-patch small model.

Dataset	Patch Size	Training Strategy	Model	Clean Accuracy	Certified Accuracy	Training Time(h)
CIFAR	2× 2	All Patch	mlp	50.8%	35.5%	9.1
			2 layer conv	52.4%	42.6%	10.7
			4 layer conv	49.6%	41.6%	22.5
			5 layer conv (wide)	-	-	∼360.0
		Random(10)	5 layer conv (wide)	64.7%	49.0%	9.5
		Random(20)	5 layer conv (wide)	64.4%	50.8%	15.8
		Guided(10)	5 layer conv (wide)	**66.5%**	49.2%	12.2
		Guided(20)	5 layer conv (wide)	**65.8%**	**51.9%**	22.4
CIFAR	5× 5	All Patch	mlp	31.1%	18.8%	7.1
			2 layer conv	35.5%	22.3%	8.7
			4 layer conv	34.0%	25.0%	18.6
			5 layer conv (wide)	-	-	∼360.0
		Random(10)	5 layer conv (wide)	**48.6%**	29.9%	9.4
		Random(20)	5 layer conv (wide)	47.8%	**30.3%**	15.4
		Guided(10)	5 layer conv (wide)	48.4%	29.0%	12.4
		Guided(20)	5 layer conv (wide)	47.6%	29.6%	23.8

2.7.5 Detailed Statistics on Training Strategies

Dataset	Training Strategies	Model Architecture	Clean Accuracy	Certified Accuracy	Training Time
MNIST	All Patch	2 layer convolution	98.63/%	91.38%	21.0
		4 layer convolution	98.48%	91.63%	80.3
		fully connected (255,10)	98.46%	91.47%	9.8
	Random (1)	2 layer convolution	98.69%	82.57%	0.2
		4 layer convolution	98.45%	81.87%	0.3
		fully connected (255,10)	98.48%	84.32%	0.2
	Random (5)	2 layer convolution	98.75%	85.87%	0.3
		4 layer convolution	98.57%	86.50%	0.3
		fully connected (255,10)	98.62%	87.32%	0.2
	Random (10)	2 layer convolution	98.73%	87.31%	0.3
		4 layer convolution	98.63%	87.54%	0.5
		fully connected (255,10)	98.49%	88.13%	0.2
	Guided (10)	2 layer convolution	98.60%	88.49%	2.3
		4 layer convolution	98.70%	88.85%	2.2
		fully connected (255,10)	98.63%	89.44%	2.2
CIFAR	All Patch	2 layer convolution	52.42%	42.57%	42.6
		4 layer convolution	49.58%	41.57%	89.8
		fully connected (255,10)	50.83%	35.49%	36.6
	Random (1)	2 layer convolution	54.93%	29.13%	0.6
		4 layer convolution	53.22%	32.35%	0.6
		fully connected (255,10)	52.76%	03.21%	0.5
	Random (5)	2 layer convolution	54.15%	37.30%	0.6
		4 layer convolution	52.19%	39.45%	0.9
		fully connected (255,10)	52.38%	20.17%	0.6
	Random (10)	2 layer convolution	53.08%	39.32%	0.7
		4 layer convolution	50.80%	38.57%	1.0
		fully connected (255,10)	51.90%	28.97%	0.6
	Guided (10)	2 layer convolution	53.04%	38.81%	3.7
		4 layer convolution	52.97%	39.84%	4.0
		fully connected (255,10)	51.32%	29.44%	3.6

Table 2.7: Detailed statistics for the comparison of training strategies - 2×2

Dataset	Training Strategies	Model Architecture	Clean Accuracy	Certified Accuracy	Training Time
MNIST	All Patch	2 layer convolution	91.88%	59.59%	28.4
		4 layer convolution	90.03%	59.72%	65.2
		fully connected (255,10)	93.96%	61.97%	7.2
	Random (1)	2 layer convolution	96.27%	18.57%	0.2
		4 layer convolution	96.83%	24.79%	0.4
		fully connected (255,10)	97.60%	29.04%	0.2
	Random (5)	2 layer convolution	95.82%	38.47%	0.2
		4 layer convolution	94.85%	42.02%	0.5
		fully connected (255,10)	96.73%	45.89%	0.2
	Random (10)	2 layer convolution	95.55%	46.13%	0.3
		4 layer convolution	94.76%	50.43%	0.6
		fully connected (255,10)	96.40%	52.30%	0.2
	Guided (10)	2 layer convolution	95.28%	50.28%	2.3
		4 layer convolution	93.98%	53.17%	3.4
		fully connected (255,10)	95.82%	55.89%	2.2
CIFAR	All Patch	2 layer convolution	35.48%	22.31%	34.8
		4 layer convolution	33.95%	24.96%	74.4
		fully connected (255,10)	31.05%	18.78%	28.4
	Random (1)	2 layer convolution	45.71%	07.14%	0.6
		4 layer convolution	42.65%	10.99%	0.6
		fully connected (255,10)	42.34%	00.10%	0.5
	Random (5)	2 layer convolution	42.85%	17.29%	0.6
		4 layer convolution	37.80%	19.63%	0.9
		fully connected (255,10)	36.23%	06.99%	0.6
	Random (10)	2 layer convolution	41.90%	21.40%	0.7
		4 layer convolution	38.41%	21.90%	1.0
		fully connected (255,10)	36.04%	12.46%	0.6
	Guided (10)	2 layer convolution	42.08%	20.77%	3.6
		4 layer convolution	36.08%	23.01%	3.9
		fully connected (255,10)	35.51%	12.56%	3.5

”

Table 2.8: Detailed statistics for the comparison of training strategies - 5×5

Joint work with Renkun Ni, Ahmed Abdelkader, Chen Zhu, Christoph Studer, Tom Goldstein

first published in ICLR 2020

Chapter 3: Detection as Regression: Certified Object Detection by Median Smoothing

3.1 Introduction

Adversarial examples are seemingly innocuous neural network inputs that have been deliberately modified to produce unexpected or malicious outputs. Early work on adversarial examples was highly focused on image classifiers, which assign a single label to an entire image [27, 28, 29, 30]. A large literature has rapidly emerged on defenses against classifier attacks, which includes both heuristic defenses [2] and certified methods with theoretical guarantees of robustness [15, 16, 21, 31, 32]. However, most realistic vision systems crucially rely on *object detectors*, rather than *image classifiers*, to identify and localize multiple objects within an image [33, 34, 35].

Over time, attacks on object detection have become more sophisticated, as has been successfully demonstrated both digitally and in the physical world using a range of perturbation techniques, as well as attacks that break both the object localization and classification parts of the detection pipeline [10, 36, 37, 38, 39]. As of this writing, we are only aware of one recent paper on the adversarial robustness of object detectors [22]. This lack of defenses is likely because (i) the complexity of the multi-stage detection pipeline [33] is difficult to analyze, and (ii) detectors are far more expensive to train than classifiers. Furthermore, (iii) object detectors output bounding-box coordinates, and are thus *regression* networks to which many standard defenses for *classifier* networks cannot be readily applied.

In this chapter, we present, to the best of our knowledge, the first certified defense against adversarial attacks on object detectors. To avoid the difficulties discussed above, we treat the complex detection pipeline as a black box without requiring specialized re-training schemes. In particular, we present a reduction from object detection to a single regression problem which envelopes the proposal, classification, and non-maximum suppression stages of the detection pipeline. Then, we endow this regression with certified robustness using the Gaussian smoothing approach [20], originally proposed for the defense of classifiers. To this end, we develop a new variant of smoothing specifically for regression based on the *medians* of the smoothed predictions, rather than their mean values. The proposed *median smoothing* approach enjoys a number of useful properties, and we expect it to find further applications in certified robustness. Finally, we implement our method to obtain a *certifiably-robust* wrapper of the YOLOv3 [34], Mask RCNN [40], and Faster RCNN [33] object detectors. We use the MS-COCO dataset [41] to test the resulting detector, obtaining the first detector to achieve non-trivial ℓ_2-norm certified average precision on large scale image dataset.

3.2 Background

We briefly review attacks on object detectors, and the certified classification methods we build upon.

Attacks and defense on object detection and semantic segmentation. Attacks exist that interfere with different components of the detection pipeline. Dense Adversary Generation (DAG) is an early attack that interferes with the classifier stage of detection [42], and was later extended to videos [43]. In contrast, region proposal networks can be manipulated by decreasing

Figure 3.1: Samples of object detection certificates using the proposed method. Dotted lines represent the farthest a bounding box could move under an adversarial perturbation δ of bounded ℓ_2-norm. If the predicted bounding box can be made to disappear, or if the label can be made to change, after a perturbation with $\|\delta\|_2 < 0.36$, then we annotate the bounding box with a red X.

the confidence of proposals [44]. The DPatch attack causes misclassification by placing a patch that does not overlap with the objects of interest [36], while the attack in [45] contaminated images with "imperceptible" patches.

The attacks described above are all digital. Early physical-world attacks on detectors fooled stop sign detectors by modifying the entire stop sign [46, 47]. While it was shown that detectors are much more robust to attacks than classifiers [33], later works successfully broke detectors using patch attacks that do not require whole-object modification. This includes printed adversarial patches that deceive person detectors [48], and adversarial clothing that makes its wearer invisible to a range of detection systems [38].

Despite a plethora of attacks, we are only aware of a single paper studying adversarial defenses for object detectors [22]. In [22], they adversarially train the object detector base on both the classification and localization loss. While empirically effective, such an approach could fail against stronger, more sophisticated attackers whereas as our approach can guarantee robustness against all possible attackers within the threat model.

In this chapter, we present a *certified* defense for object detectors that is robust regardless of the method used to craft the attack. Our work is motivated by recent progress on certified image classification by the randomized smoothing approach, as we review next.

Certified defenses for image classification. Several methods of obtaining robustness certificates for classification problems have been proposed [16, 32, 49, 50, 51, 52, 53]. In addition, [21, 54, 55, 56, 57] proposed methods to both defend the model while enabling better certificates. For our purposes, we focus on randomized smoothing defenses [20, 54]. These first convert a base classifier to a smoothed classifier by labeling many images sampled from a Gaussian ball around

the input, and taking a majority vote [20]. The effect of image perturbations on this smoothed classifier can be bounded using either the Neyman-Pearson Lemma [20] or properties of the Weierstrass transform [56].

Since [54] proposed randomized smoothing for robustness, it has been improved by tightening the certificate [20], improving the training process [56], adapting it to new threat models [55], or incorporating confidence level [58] – all in the context of classification tasks. In particular, the voting scheme in [20, 58] requires a global bound on the output of the base classifier, which naturally holds for binary classifiers with a 0/1 output.

We will see later that existing certificates based on randomized smoothing become weaker when applied to regression problems over a large range of output values. Towards the intended application of robust detection, we propose a new certificate for regression problems based on the median of the smoothed predictions, rather than their mean values, which is of independent interest.

3.3 Detection Certificates

In this section, we introduce the proposed certificate in the context of modern detection pipelines. We begin by describing the black box interface that allows us to certify the outputs of detector networks without the need for re-training. Then, we motivate and define the proposed certificate at a high level, where the next two sections fill in the details.

Detection interface. Typical object detectors take an input image and output a variable-length list of bounding boxes and associated class labels $\{(b_1, \ell_1), (b_2, \ell_2), \dots\}$. State-of-the-art detectors, such as Faster-RCNN [33], YOLO [34], and RetinaNet [35] usually have many output heads,

each of which is responsible for a single bounding box. As the number of output heads is usually much larger than the number of objects, redundant boxes need to be filtered out, e.g., by *Non-Maximum Suppression* (NMS). First, NMS discards any box with an *objectness score* below a threshold τ. Then, the box b^* with the highest score is taken out as output, and any remaining boxes overlapping b^* significantly are filtered out. This process is repeated until there are no more boxes left.

Certifying detector outputs. Adversarial attacks on object detectors attempt to distort the location and appearance of objects. We certify detector outputs by bounding the positions and sizes of the detected objects. In addition, we ensure that the associated class labels stay the same.

Representation. We represent a bounding box b using the coordinates of its corners (x_1, y_1, x_2, y_2), with $x_1 \leq x_2$ and $y_1 \leq y_2$, along with the associated class label ℓ and objectness score. Then, to measure the overlap between two boxes b_1 and b_2, we use the *Intersection over Union* defined as $\text{IoU}(b_1, b_2) = \text{Area}(b_1 \cap b_2)/\text{Area}(b_1 \cup b_2)$.

Bounding-box certificate. Given a predicted bounding box b, we aim to certify its size and location. Assume for now that we obtained certified lower and upper bound for each coordinate of b as $(\underline{x}_1, \underline{y}_1, \underline{x}_2, \underline{y}_2)$ and $(\overline{x}_1, \overline{y}_1, \overline{x}_2, \overline{y}_2)$. We say that a box is "certifiably correct" if the IoU between the ground truth bounding box and the worst-case bounding box, with coordinates respecting the certified bounds, is above a certain threshold. The worst-case bounding box is the box with coordinates satisfying the certified upper and lower bounds which realizes the lowest IoU with the ground-truth box. If $\underline{x}_2 \leq \overline{x}_1$ or $\underline{y}_2 \leq \overline{y}_1$, then the worst-case IoU is zero. Otherwise, the worst-case bounding box can always be found in the set $\{\underline{x}_1, \overline{x}_1\} \times \{\underline{y}_1, \overline{y}_1\} \times \{\underline{x}_2, \overline{x}_2\} \times$

$\{\underline{y}_2, \overline{y}_2\}$. Hence, we simply enumerate all 16 boxes, and take the smallest IoU. As long as the worst-case IoU is larger than the threshold τ, then the box b is considered certifiably correct.

Label certificates. We treat the label $\ell \in \mathbb{N}$ as an additional coordinate. Again, assuming we obtained certified lower and upper bounds as $\overline{\ell}$ and $\underline{\ell}$, then ℓ is only considered certified when $\overline{\ell} = \underline{\ell}$.

In the next section, we describe the smoothing approach we use to obtain the required certified bounds on each coordinate.

3.4 Median Smoothing for Regression

A number of strategies have been proposed for certifying classifiers, many based on Gaussian means. We will see below that the bounds provided become fairly weak for regression problems. For this reason, we propose smoothing based on Gaussian *medians*, which provide considerably stronger bounds for regression networks such as object detectors.

Mean smoothing. Given a base function $f : \mathbb{R}^d \to \mathbb{R}$, its Gaussian smoothed analog is [20, 54, 56]

$$g(x) = \mathbb{E}[f(x + G)], \quad \text{where } G \sim N(0, \sigma^2 I). \tag{3.1}$$

In the context of classifier networks, where output heads take the form $f : \mathbb{R}^d \to [0, 1]$, a certificate can be obtained by bounding the gap between the highest and second-highest class probabilities [20]. However, we aim to apply the smoothing technique to a general regression problem, such as bounding box regression, with $f : \mathbb{R}^d \to [l, u]$. In that case, the bound on g can be rather loose, which follows by invoking Lemma 2 from [56] (see their appendix)

with the normalized function $\frac{f(x)-l}{u-l}$ as stated below; see Appendix A for further discussion. Throughout, we use Φ to denote the cumulative distribution function (CDF) of the standard Guassian distribution.

Corollary 3.0.1. *[56] For any $f : \mathbb{R}^d \to [l, u]$, the map $\eta(x) = \sigma \cdot \Phi^{-1}(\frac{g(x)-l}{u-l})$ is 1-Lipschitz, implying*

$$l + (u-l) \cdot \Phi\left(\frac{\eta(x) - \|\delta\|_2}{\sigma}\right) \leq g(x+\delta) \leq l + (u-l) \cdot \Phi\left(\frac{\eta(x) + \|\delta\|_2}{\sigma}\right) \tag{3.2}$$

Median smoothing. The issue with Gaussian smoothing is that the mean values it computes can be highly skewed by extreme values of the base function. Hence, the resulting bounds are rather loose when applied to functions with large variations in their outputs. This is not a problem for classifiers (which output values between 0 and 1), but is highly problematic for general regression problems.

To obtain tighter certificates, we utilize the percentiles of the output random variable instead of its mean. In particular, as the *median* is almost unaffected by outliers, a global bound on the base function is no longer required. Formally, we propose the following formulation.

Definition 1. Given $f : \mathbb{R}^d \to \mathbb{R}$ and $G \sim N(0, \sigma^2 I)$, we define the *percentile smoothing* of f as

$$\underline{h}_p(x) = \sup\{y \in \mathbb{R} \mid \mathbb{P}[f(x+G) \leq y] \leq p\} \tag{3.3}$$

$$\overline{h}_p(x) = \inf\{y \in \mathbb{R} \mid \mathbb{P}[f(x+G) \leq y] \geq p\} \tag{3.4}$$

While the two forms $\underline{h}_p$ and $\overline{h}_p$ are equivalent for continuous distributions, the distinction is needed to handle edge cases with discrete distributions. In the remainder of the chapter, we will

use h_p to denote the percentile-smoothed function when either definition can be applied. While h_p may not admit a closed form, we can approximate it by Monte Carlo sampling [20], as we explain in Section 3.5.

A useful property of percentile smoothing is that it always outputs a realizable output of the base function f. This can be useful when f produces discrete values or labels (as is the case for classifiers), or bounding boxes. In contrast, mean smoothing is a weighted average of the outputs, and it is more susceptible to outliers. For example, when two distinct bounding boxes are predicted, we select one or the other rather than their average.

Regression certificates. To certify a percentile-smoothed function h_p for input x under adversarial perturbations of bounded ℓ_2-norm, we evaluate the function at x with two appropriate percentiles $\underline{p}$ and $\overline{p}$. The basic idea is to first bound the probability that the output of the base function f will fall below a particular threshold Λ. A key observation is that this is equivalent to bounding a mean-smoothed indicator function $\mathbb{E}(\mathbf{1}_{f(x+G)<\Lambda})$, which satisfies the assumption of Corollary 3.0.1. We can then use this bound to further bound the change in the percentiles output by h_p; see Appendix B for the full proof.

Lemma 3.1. *A percentile-smoothed function h_p with adversarial perturbation δ can be bounded as*

$$\underline{h}_{\underline{p}}(x) \leq h_p(x+\delta) \leq \overline{h}_{\overline{p}}(x) \quad \forall \, \|\delta\|_2 < \epsilon, \tag{3.5}$$

where $\underline{p} := \Phi\left(\Phi^{-1}(p) - \frac{\epsilon}{\sigma}\right)$ and $\overline{p} := \Phi\left(\Phi^{-1}(p) + \frac{\epsilon}{\sigma}\right)$, with Φ being the standard Gaussian CDF.

The immediate benefit of h_p is that the tightness of the bound now depends on the concentration

of f around the p-th percentile of $f(x+G)$, per the local gap between the percentiles $\underline{h}_{\underline{p}}(x)$ and $\overline{h}_{\overline{p}}(x)$. In contrast, the bound obtained by g depends on the position of $\mathbb{E}[f(x+G)]$ relative to the global bounds per the extreme values l and u.

In addition, percentile smoothing can be applied without specifying the bounds l and u, which may be unknown a priori. Even when one or both of the bounds l and u is infinite, percentile smoothing provides a non-vacuous certificate where mean smoothing fails. For example, take f so that $f(x+G) \sim N(0,1)$. In this case, it is easy to see that the bounds on $g(x+\delta)$ per Corollary 3.0.1 are vacuous, while $h_{50\%}(x+\delta)$ can be bounded between $\pm\|\delta\|_2/\sigma$ by Lemma 5.0.1.

Now that we have a mechanism for bounding the individual coordinates output by an object detector, we proceed to describe the specifics of our approach and its implementation.

3.5 Implementing Detection Certificates

In order to certify the predictions output by the object detector, we aggregate multiple predictions made under randomly perturbed inputs using the median smoothing approach presented in Section 3.4. Roughly speaking, the more the aggregated predictions agree, the stronger the certificate.

Recall that each prediction consists of four coordinates (x_1, y_1, x_2, y_2) representing a bounding box, with the associated label $\ell \in \mathbb{N}$ treated as the fifth coordinate. As outlined in Section 3.3, the proposed certificate requires lower and upper bounds for each coordinate of each prediction.

Our certification strategy is based on treating each coordinate c as the output of a dedicated function f_c. Then, we apply median smoothing to certify each coordinate independently. The

challenge to implementing this strategy is to consolidate multiple predictions, each with five coordinates, as a single vector so that certified regression can be applied. The main complication is that object detectors typically produce a variable-length list of predictions in no particular order. In contrast, our regression model requires a single vector with a consistent assignment of indices to prediction coordinates.

Certification pipeline. First, we compose a base detector b into a sequence $f = r \circ b \circ d$, where r encodes the predictions produced by b into one or more vectors, and d is a denoiser to improve concentration. Then, we work with the median smoothing of f, see Definition 2, and use Monte Carlo sampling to approximate $h_p(x)$ along with lower and upper bounds. See Figure 3.2 for the overall workflow and Algorithm 3 for the pseudocode.

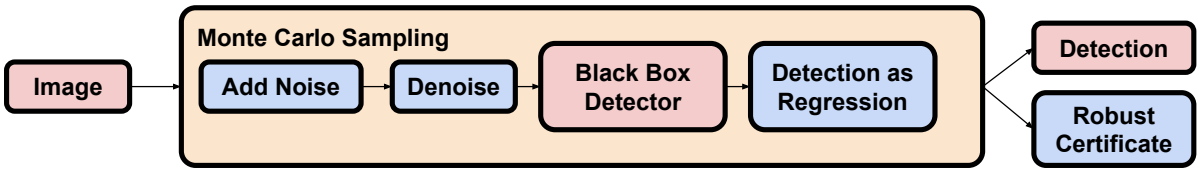


Figure 3.2: Converting a base detector to a certified robust detector.

Denoising. The denoiser d is applied to the input of the base detector b. The idea is that, since the Gaussian smoothing certificate can be applied to any pipeline, we might as well apply it to one that begins with a denoiser that removes most of the Gaussian noise. This makes $f(x + G)$ more concentrated [57], resulting in a stronger certificate without re-training on noisy data.

Encoding detectors as regressors. Given n detections of potential objects, the corresponding predictions can be represented as $u \in \mathbb{R}^{n \times 5}$, where n may vary across random perturbations. We aim to convert u into a suitable vector $v = r(u)$. A naive approach to implement the encoding r

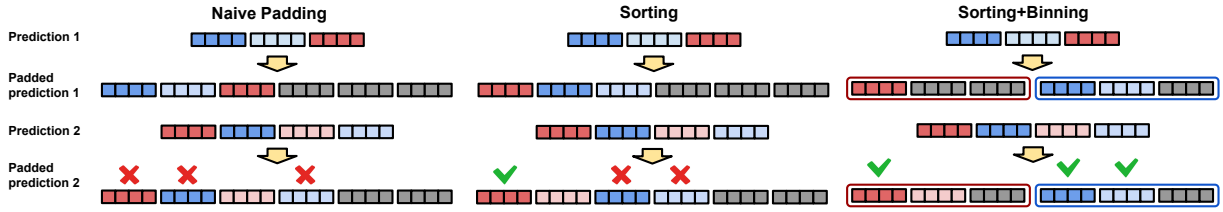


Figure 3.3: Even when the bounding boxes predicted for one image appear among the predictions for another noisy sample, the outputs will appear completely different if the boxes are not ordered in the same way. In the first column, we convert the detected output into a fixed length vector by directly padding the output, and none of the boxes are aligned. In the second column, we sort the boxes by location first before padding, allowing us to align one box out of three. In the third column, we place the boxes into corresponding bins before sorting, and this approach aligns all three boxes as desired.

is to simply copy u into v before possibly padding v with sentinel entries up to the desired length. However, even if the detector produces the same predictions under different noises, their ordering may be different leading to inconsistencies that weaken the certificate.

To improve the consistency of the encoded regression vectors, we propose two operations. The first operation is based on *sorting* the predictions in u either by their objectness scores or by the centers of their bounding boxes.[1] The second operation is based on partitioning u into independent *bins* based on the labels or the locations[2] of the predictions, with each bin b_i encoded separately as a vector v_i. Detecting a different number of objects in some bin does not affect the other bins; see Figure 3.3. For example, when binning by label, a dog which is only detected under some perturbations would not impact the certificates produced for any cars; see Figure 3.1. As the random perturbations may produce vectors of varying lengths, we implicitly assume that all vectors are padded with sentinel values, taken as ∞, such that every coordinate has a corresponding realization in all outputs.

[1]When sorting by centers, we first sort vertically then horizontally. We found that this is important in achieving better results as in the object detection task, the horizontal location of a detected object seems to be more informative than the vertical location.

[2]For location binning, we split the image into 3x3 grid cells, and gather the corresponding boxes into bins based on the center of the box.

Algorithm 1 Prediction and Certified Detection

function DETECT(f, x, σ, n)
  $\hat{x} \leftarrow$ AddGaussianNoise(x, σ, n) ▷ $\hat{x}$ is n times as large as x
  $\hat{y} \leftarrow$ toRegression($f(\hat{x})$) ▷ Convert detection output into regression vectors
  $\hat{y} \leftarrow$ Sort($\hat{y}$) ▷ Sort each coordinates along the batch dimension
  $y_{median} \leftarrow \hat{y}_{\lfloor 0.5n \rfloor}$ ▷ Take the median
  $bbox, \ell \leftarrow$ toDetection(y_{median})
  return $bbox, \ell$
function CERTIFYDETECT(f, x, σ, n, ϵ, c)
  $q_u, q_l \leftarrow$ GetEmpiricalPerc(n, ϵ, c)
  $\hat{x} \leftarrow$ AddGaussianNoise(x, σ, n) ▷ $\hat{x}$ is n times as large as x
  $\hat{y} \leftarrow$ toRegression($f(\hat{x})$)
  $\hat{y} \leftarrow$ Sort($\hat{y}$) ▷ Sort each coordinates along the batch dimension
  $y_{median} \leftarrow \hat{y}_{\lfloor 0.5n \rfloor}$ ▷ Take the median
  $y_u, y_l \leftarrow \hat{y}_{q_u}, \hat{y}_{q_l}$ ▷ Take the qth order statistics
  $bbox, \ell, \overline{bbox}, \overline{\ell}, \underline{bbox}, \underline{\ell} \leftarrow$ toDetection(y_{median}, y_u, y_l)
  return $bbox, \ell, \overline{bbox}, \overline{\ell}, \underline{bbox}, \underline{\ell}$

Evaluation of h_p (`GetEmpiricalPerc`). In practice, we resort to using Monte Carlo sampling to approximate the upper bound of $\overline{h}_{\overline{p}}(x)$ and lower bound of $\underline{h}_{\underline{p}}(x)$ similar to [20]. Given n draws $\{G_1, G_2, \ldots, G_n\}$, we evaluate $X_i = f(x + G_i) \in \mathbb{R}$. Then, we find the corresponding order statistics $0 = K_0 \leq K_1 \leq K_2 \cdots \leq K_n \leq K_{n+1} = \infty$. We want to find the empirical order statistic K_{q^u} and K_{q^l}, such that $P(K_{q^u} \geq \overline{h}_{\overline{p}}) \geq \alpha$ and $P(K_{q^l} \leq \underline{h}_{\underline{p}}) \geq \alpha$. Specifically, given an order statistic K_{q^u}, we can evaluate $P(K_{q^u} \geq \overline{h}_{\overline{p}})$ explicitly using the binomial formula below.

$$P(\overline{h}_{\overline{p}} \leq K_{q^u}) = \sum_{i=1}^{i=q^u} P(K_{i-1} < \overline{h}_{\overline{p}} \leq K_i) = \sum_{i=1}^{j=q^u} \binom{n}{i} (\overline{p})^i (1-\overline{p})^{n-i} \tag{3.6}$$

A similar formula can be derived for $P(\underline{h}_{\underline{p}} \geq K_{q^l})$. We can then use binary search to find the smallest q^u and largest q^l such that $P(\overline{h}_{\overline{p}} \leq K_{q^u}) \geq \alpha$ and $P(\underline{h}_{\underline{p}} \geq K_{q^l}) \geq \alpha$ are satisfied.

3.6 Experiments

We mainly use YOLOv3 [34], pretrained on the MS-COCO dataset [41], as our black-box detector where IoU thresholds for NMS is set to 0.4. The evaluation is done on all 5000 images from the test set, with adversarial perturbations $\|\delta\|_2 < \epsilon = 0.36$. We set the IoU threshold for certification $\tau = 0.5$, as it is a common setting for evaluation on the MS-COCO dataset [41]. To perform the smoothing, we inject Gaussian noise with standard deviation $\sigma = 0.25$, and we use 2000 noise samples for each image. The estimated upper and lower bound for each coordinate are selected such that they bound the true $\overline{h}_{\overline{p}}(x)$ and $\underline{h}_{\underline{p}}(x)$ with confidence $\alpha = 99.999\%$. For denoising, we use the DNCNN denoiser [59] pretrained by [57] (with $\sigma = 0.25$).

We find all three operations – sorting, binning, and denoising – are helpful in mitigating the impact of smoothing on the clean performance as well as increasing the certified performance. All of these methods complement each other for the most part, and we are able to achieve the most robust and accurate smoothed object detector by using all three methods. Specifically, we find that sorting by box location works best, and that prepending a denoiser is indeed very important for both the clean and certified performance.

Robustness metrics and performance evaluation. We evaluate the smoothed models based on two metrics: AP and certified AP. AP is indicative of the smoothed model's performance in absence of an adversary, and ideally, we would like to avoid drops in AP when converting the base detector to a smoothed one. On the other hand, certified AP tells us the guaranteed lower bound on the AP when faced with the specified adversary.

More specifically, *certified precision and recall* are calculated as follows. To get the number

of certifiably correct boxes, we count the number of detections whose worst-case IoU with the corresponding ground-truth box exceeds the threshold τ. To calculate the certified precision, we also need to know the maximum possible number of detections under any perturbed input. We upper bound this number by counting all certifiably non-empty entries across all regression vectors.

$$\text{Certified Recall} = \frac{\text{\# certifiably correct detections}}{\text{\# ground-truth detections}} \tag{3.7}$$

$$\text{Certified Precision} = \frac{\text{\# certifiably correct detections}}{\text{max \# predicted detections}} \tag{3.8}$$

Due to the binning and sorting processes associated with smoothing, it is difficult to calculate the exact precision/recall curve at all objectness thresholds. We instead evaluate certified precision and recall at 5 different objectness thresholds $\{0.1, 0.2, 0.4, 0.6, 0.8\}$, and use the area under the steps to lower-bound the true certified AP.

Binning Method	**Sorting Method**	**AP @ 50**	**Certified AP @ 50**
Blackbox detector		48.66%	-
None	Objectness	17.60%	1.24%
	Location	21.51%	1.24%
Label	Objectness	25.27%	2.67%
	Location	29.75%	3.32%
Location	Objectness	27.48%	3.23%
	Location	28.90%	2.67%
Location+Label	Objectness	30.32%	3.97%
	Location	32.04%	4.18%

Table 3.1: Clean and certified AP using various sorting and binning methods with YOLOv3 as base detector. Detailed precision/recall statistics can be found in Appendix D.

Sorting and binning methods. We find that sorting bounding boxes by location consistently achieves better clean AP and certified AP than sorting them by objectness score, as shown in Table 3.1. The only exception is when location binning is used together with location sorting where the clean AP improves, but certified AP decreases. In the best case, switching to location sorting improves AP by 4.52% and certified AP by 0.64%.

Both label binning and location binning also consistently boost the clean AP and certified AP. Compared to no binning, label binning alone could increase the AP by 7-8% and the certified AP by 1-2%, as shown in Table 3.1. Furthermore, the two binning methods complement each other, and we are able to achieve the best AP and certified AP by combining both location binning and label binning.

Denoising. Because we do not retrain the detector under noise, denoising the image first is extremely important in achieving good clean AP and certified AP. In Table 3.2, we present performance with and without the denoiser – given the results in Table 3.1 as summarized above, we restrict experiments to the location sorting method. Without the denoiser, the clean AP drops by an average of 20.07% and the certified AP drops by an average of 2.55%.

Architecture. Since our proposed approach treats the base detector as a black box, we also experimented with Mask-RCNN and Faster-RCNN as the base detector. We use the best combination of settings from Table 1 for all three architectures. Surprisingly, even though the base Mask-RCNN and Faster-RCNN perform better compared to YOLOv3, after smoothing, they both perform consistently worse. Certified AP is decreased by almost 2/3 after switching to the alternative architecture.

Binning Method	Denoise	AP @ 50	Certified AP @ 50
None	Yes	21.51%	1.24%
	No	5.85%	0.14%
Label	Yes	29.75%	3.32%
	No	8.58%	0.32%
Location	Yes	28.90%	2.67%
	No	8.03%	0.32%
Location+Label	Yes	32.04%	4.18%
	No	9.49%	0.44%

Table 3.2: Denoising significantly improves the clean AP and certified AP with YOLOv3 as base detector. Sorting method: location. Detailed precision/recall statistics can be found in Appendix D.

Architecture	Base Detector AP @ 50	Smoothed Detector AP @ 50	Smoothed Detector Certified AP @ 50
YOLOv3	48.66%	31.93%	4.21%
Mask RCNN	51.28%	30.53%	1.67%
Faster RCNN	50.47%	29.89%	1.54%

Table 3.3: Robustness comparison using different base detectors.

Tightness of certification. To understand the tightness of our certification, we implemented the DAG attack [42] against our most robust smoothed detector. We take 20 PGD steps and draw 5 random samples to estimate the gradient of the smoothed model. Surprisingly, the smoothed model is quite robust within the desired radius. The DAG attack was only able to decrease recall by 1.1%. This illustrates that the bound we obtained is likely quite loose with respect to the true robustness of the smoothed object detector.

Inference speed. We note that the Monte Carlo sampling process is inherently costly. In our experiments, we used 2000 samples to approximate the smooth model for each image, which makes our evaluation 2000 times as expensive. This is a common problem in the randomized

smoothing approach [20], and reducing the sample complexity is still an active area of research [60].

While the certified AP are still far below the requirements of real-world applications, the proposed smoothing approach is able to achieve non-trivial certified AP *without* any re-training of the base detector. We think this is a promising direction for eventually obtaining practical verifiably robust object detectors.

Broader Impact

Neural networks are very powerful tools, and society will benefit greatly if they can be used in a broader range of safety critical applications. In particular, object detectors can be used in many systems that must visually perceive and interact with the real world – perhaps most strikingly, in autonomous vehicles. Ensuring the safety and predictability of neural networks is critical in enabling the application of neural networks in these areas, and certificates associated with safety with respect to a particular model are very useful in providing assurance that the neural network cannot be exploited by malicious actors. At the same time, there is real concern about the privacy impact of widespread deployment of modern computer vision systems, including systems like object detectors and face recognition systems that produce bounding boxes. If individuals wish to use physical or digital adversarial examples to protect their privacy, the techniques we present might make it more difficult for them to do so, although it is not actually clear that adversarial examples will ultimately prove effective or useful for protecting privacy. In any case, we are not aware of any real-world adversarial attacks being performed "in the wild", for good or ill. We believe the concrete positive impact on safety is probably greater than a

hypothetical negative impact on privacy.

3.7 Chapter Summary and Future Work

We propose a new type of randomized smoothing using a percentile instead of an expectation, such that certificates can be easily generated for regression-type problems. We then apply it to obtain the first certified defense for object detectors. Some potential future work includes decreasing the sample complexity of our randomized certificate, extending our robust detector to defend against other threat models like patch attacks leveraging the method from [55], or using a learning approach to find a better reduction from detection to regression compared to our binning/sorting approach. We note that the machinery we have developed for certifying regression problems is largely application agnostic, and we hope it can find use in certifying a range of other regression tasks.

Joint work with Michael J. Curry, Ahmed Abdelkader, Aounon Kumar, John Dickerson, Tom Goldstein first published in NeurIPS 2020

Chapter 4: Certifying Strategyproof Auction Networks

4.1 Introduction

Auctions are an important mechanism for allocating scarce goods, and drive billions of dollars of revenue annually in online advertising [61], sourcing [62], spectrum auctions [63, 64], and myriad other verticals [65, 66]. In the typical auction setting, agents who wish to bid on one or more items are presumed to have private valuations of the items, which are drawn at random from some prior valuation distribution. They then bid in the auction, possibly lying strategically about their valuation while attempting to anticipate the strategic behavior of others. The result is a potentially complicated Bayes-Nash equilibrium; even predicting these equilibria can be difficult, let alone designing auctions that have good equilibria.

This motivates the design of **strategyproof** auctions: these are auctions where players are incentivized to simply and truthfully reveal their private valuations to the auctioneer. Subject to the strategyproofness constraint, which makes player behavior predictable, it is then possible to optimize the mechanism to maximize revenue. A classic strategyproof auction design is the second-price auction—coinciding with the celebrated Vickrey-Clarke-Groves (VCG) mechanism [67, 68, 69]—in which participants bid only once and the highest bidder wins, but the price paid by the winner is only the amount of the second-highest bid.

In a groundbreaking work, [70] characterized the revenue-maximizing strategyproof auction

for selling one item to many bidders. However, there has been very little progress in characterizing optimal strategyproof auctions in more general settings. Optimal mechanisms are known for some cases of auctioning two items to a single bidder [71, 72, 73, 74]. The optimal strategyproof auction even for 2 agents buying 2 items is still unknown.

A more recent set of approaches involves formulating the auction design problem as a learning problem. [75] provide the general end-to-end approach which we build on in this chapter. In brief, they design a neural network architecture to encode an auction mechanism, and train it on samples from the valuation distribution to maximize revenue. Their goal is to enforce, at least approximately, dominant-strategy incentive compatibility: a stronger notion than the Bayesian incentive compatibility of some other mechanism design approaches [76, 77, 78]. While they describe a number of network architectures which work in restricted settings, we focus on their RegretNet architecture, which can be used in settings with any number of agents or items.

The training procedure for RegretNet involves performing gradient ascent on the network inputs, to find a good strategic bid for each player; the network is then trained to minimize the difference in utility between strategic and truthful bids—this quantity is the eponymous "regret". The process is remarkably similar to adversarial training [79], which applies robust optimization schemes to neural networks, and the desired property of strategyproofness can be thought of as a kind of adversarial robustness. Motivated by this connection, we use techniques from the adversarial robustness literature to compute **certifiable** upper bounds on the amount by which a player with a specific valuation profile can improve their utility by strategically lying.

While the adversarial training approach seems effective in *approximating* strategyproof auction mechanisms, neural network training is fraught with local minima and suboptimal stationary points. One can discover strategic behaviors by using simple gradient methods on RegretNet

auctions, but we note that it is known from the adversarial examples literature that such results are often sub-optimal [80] and depend strongly on the optimizer [81]. For this reason, it is unclear how strategyproof the results of RegretNet training are, and how much utility can be gained through strategic behavior in such auctions.

Our goal here is to learn auction mechanisms that are not only approximately strategyproof, but that come with rigorous bounds on how much they can be exploited by strategic agents, regardless of the strategy used. We achieve this by leveraging recent ideas developed for certifying adversarial robustness of neural classifiers [82, 83, 84], and adapting them to work within an auction framework.

Our contributions.

- We initiate the study of certifiably strategyproof learned auction mechanisms. We see this as a step toward achieving the best of both worlds in auction design—maintaining *provable properties* while expanding to more *general settings* than could be analyzed by traditional methods.

- We develop a method for formulating an integer-programming-based certifier for general learned auctions with additive valuations. This requires changes to the RegretNet architecture. We replace its softmax activation with the piecewise linear sparsemax [85], and we present two complementary techniques for dealing with the requirement of individual rationality: either formulating a nonconvex integer program, or removing this constraint from the network architecture and adding it as a learned penalty instead.

- We provide the first *certifiable* learned auctions for several settings, including a 2 agent, 2 item case where no known optimal auction exists; modulo computational scalability, our

techniques for learning auctions and producing certificates for a given valuation profile work for settings with any number of items or (additive) bidders.

4.2 Background

Below, we introduce the general problem of automated mechanism design. We then describe the RegretNet approach for learning auction mechanisms, as well as the neural network verification techniques that we adapt to the auction setting. The RegretNet architecture originated the idea of parameterizing mechanisms as neural networks and training them using techniques from modern deep learning. This approach has been termed "differentiable economics", and several other papers have expanded on this approach in various settings beyond revenue-maximizing sealed-bid auctions [86, 87, 88, 89, 90].

4.2.1 Previous work

Automated mechanism design is the problem of finding good mechanisms for specific valuation distributions. In this area, one strand of work involves discretizing the space of types and solving a linear program to find the best auction in a family of mechanisms [91, 92]. For Bayesian incentive compatible revenue-maximizing auctions with additive bidders, [76, 77, 78] give techniques for finding the optimal mechanism, although Bayesian incentive compatibility is a weaker requirement than dominant-strategy incentive compatibility. Other work requires only access to samples from the valuation distribution over which revenue must be maximized [93, 94]. In this way, auction design becomes a learning problem, to which the tools of learning theory can be applied [95]. RegretNet falls into this latter family of techniques. In particular, it

is an approach that learns from samples to approximate a DSIC mechanism.

4.2.2 RegretNet

In the standard auction setting, it is assumed that there are n agents (indexed by i) buying k items (indexed by j), and that the agents value the items according to values drawn from some distribution $P(\boldsymbol{v}_i)$. This distribution is assumed to be public, common knowledge (it is essentially the data-generating distribution). However, the specific sampled valuations are assumed to be private to each agent.

The auctioneer solicits a bid b_{ij} from all agents on all items. The auction mechanism $f(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n)$ aggregates bids and outputs the results of the auction. This consists of an allocation matrix a_{ij}, representing each player's probability of winning each item, and a payment vector p_i, representing the amount players are charged. Players receive some utility u_i based on the results; in this work, we focus on the case of additive utility, where $u_i = \sum_j a_{ij} v_{ij} - p_i$.

As previously mentioned, players are allowed to choose bids strategically to maximize their own utility, but it is often desirable to disincentivize this and enforce strategyproofness. The auctioneer also wants to maximize the amount of revenue paid. [75] present the RegretNet approach: the idea is to represent the mechanism f as a neural network, with architectural constraints to ensure that it represents a valid auction, and a training process to encourage strategyproofness while maximizing revenue.

(We note that [75] presents other architectures, RochetNet and MyersonNet, which are completely strategyproof by construction, but only work in specific settings: selling to one agent, or selling only one item. In our work, we focus only on certifying the general RegretNet

architecture.)

4.2.2.1 Network architecture

The RegretNet architecture is essentially an ordinary feedforward network that accepts vectors of bids as input and has two output heads: one is the matrix of allocations and one is the vector of payments for each agent.

The network architecture is designed to make sure that the allocation and payments output are feasible. First, it must ensure that no item is overallocated: this amounts to ensuring that each column of the allocation matrix is a valid categorical distribution, which can be enforced using a softmax activation.

Second, it must ensure that no bidder (assuming they bid their true valuations) is charged more than the expected value of their allocation. It accomplishes this by using a sigmoid activation on the payment output head to make values lie between 0 and 1 – call these $\tilde{p}_i$. Then the final payment for each player is $\left(\sum_j v_{ij} a_{ij}\right) \tilde{p}_i$; this guarantees that utility can at worst be 0.

Both of these architectural features pose problems for certification, which we describe below.

4.2.2.2 Training procedure

The goal of the auctioneer is to design a mechanism that maximizes the expected sum of payments received $\mathbb{E}_{\boldsymbol{v} \sim P(\boldsymbol{v})} \left[\sum_i p_i(\boldsymbol{v})\right]$, while ensuring that each player has low regret, defined as the difference in utility between the truthful bid and their best strategic bid:

$$\mathrm{rgt}_i(\boldsymbol{v}) = \max_{\boldsymbol{b}_i} u_i(\boldsymbol{b}_i, \boldsymbol{v}_{-i}) - u_i(\boldsymbol{v}_i, \boldsymbol{v}_{-i}) \tag{4.1}$$

Note that this definition of regret allows only player i to change their bid. However, if $\mathbb{E}_{\boldsymbol{v}}[\mathrm{rgt}_i(\boldsymbol{v})]$ is low for all players then the mechanism must be approximately strategyproof; this is because every possible strategic bid by players other than i could also be observed as a truthful bid from the support of $P(\boldsymbol{v})$.

[75] approximates regret using an approach very similar to adversarial training [79]. They define a quantity $\widehat{\mathrm{rgt}_i}$ by approximately solving the maximization problem using gradient ascent on the input – essentially finding an adversarial input for each player. Given this approximate quantity, they can then define an augmented Lagrangian loss function to maximize revenue while forcing $\widehat{rgt}$ to be close to 0:

$$\mathcal{L}(\boldsymbol{v}, \boldsymbol{\lambda}) = -\sum_i p_i + \sum_i \lambda_i \widehat{\mathrm{rgt}_i}(\boldsymbol{v}) + \frac{\rho}{2} \sum_i \left(\widehat{\mathrm{rgt}_i}(\boldsymbol{v})\right)^2 \tag{4.2}$$

They then perform stochastic gradient descent on this loss function, occasionally increasing the Lagrange multipliers λ, ρ and recomputing $\widehat{\mathrm{rgt}}$ at each iteration using gradient ascent. At test time, they compute revenue under the truthful valuation and regret against a stronger attack of 1000 steps. A number of high probability generalization bounds are provided for estimating regret and revenue from pointwise values on samples. With regards to the estimation of regret, we note that their generalization bound refers to the true regret at a single point (equation 4.1) – a quantity which a gradient-based method can only approximate but not compute exactly.

4.2.3 Mixed integer programming for certifiable robustness

Modern neural networks with ReLU activations are piecewise linear, allowing the use of integer programming techniques to verify properties of these networks. [82] present a good overview of various techniques use to certify adversarial robustness, along with some new methods. The general approach they describe is to define variables in the integer program representing neural network activations, and constrain them to be equal to each network layer's output:

$$\begin{aligned} \hat{\boldsymbol{x}}_{i+1} &= W_i \boldsymbol{x}_i + \boldsymbol{b}_i \\ \boldsymbol{x}_{i+1} &= \max(0, \hat{\boldsymbol{x}}_{i+1}) \end{aligned} \tag{4.3}$$

With input constraints $x_0 \in S$ representing the set over which the adversary is allowed to search, solving the problem to maximize some quantity will compute the actual worst-case input. In most cases, this is some proxy for the classification error, and the input set is a ball around the true input; in our case, computing a certificate for player i involves maximizing $u_i(\boldsymbol{b}_i, \boldsymbol{v}_{-i})$ over all $\boldsymbol{b}_i \in \text{Supp}(P(\boldsymbol{v}_i))$, i.e. explicitly solving (4.1).

The program is linear except for the ReLU term, but this can be represented by adding some auxiliary integer variables. In particular, [84] present the following set of constraints (supposing a d-dimensional layer output), which they show are feasible iff $\boldsymbol{x}_i = \max(\hat{\boldsymbol{x}}_i, 0)$:

$$\begin{aligned} \boldsymbol{\delta}_i \in \{0,1\}^d, \quad \boldsymbol{x}_i \geq 0, \quad \boldsymbol{x}_i \leq \boldsymbol{u}_i \boldsymbol{\delta}_i \\ \boldsymbol{x}_i \geq \hat{\boldsymbol{x}}_i, \quad \boldsymbol{x}_i \leq \hat{\boldsymbol{x}}_i - \boldsymbol{l}_i(1 - \boldsymbol{\delta}_i) \end{aligned} \tag{4.4}$$

The $\boldsymbol{u}_i, \boldsymbol{l}_i$ are upper and lower bounds on each layer output that are known a priori – these can be derived, for instance, by solving some linear relaxation of the program representing the

network. In particular, an approach called Planet due to [96] involves resolving the relaxation to compute tighter bounds for each layer in turn. [82] provide a Gurobi-based [97] integer program formulation that uses the Planet relaxations, later updated for use by [83]; we modify that version of the code for our own approach. These methods output a solution which will at least be a lower bound on the true regret. Under the assumption that the chosen integer programming solver accurately reports when it has solved programs to global optimality, this will also be an upper bound – thus we will know the true maximum regret. Using bounds from [75], by computing true expected regret at many sampled points, we can estimate the overall regret of the mechanism with high probability, and we can bound the probability of sampling a point with high regret.

4.3 Techniques

These neural network verification techniques cannot be immediately applied to the RegretNet architecture directly. We describe modifications to both the network architecture and the mathematical programs that allow for their use: a replacement for the softmax activation that can be exactly represented via a bilevel optimization approach, and two techniques for coping with the individual rationality requirement. We also use a regularizer from the literature to promote ReLU stability, which empirically makes solving the programs faster.

4.3.1 Sparsemax

The RegretNet architecture applies a softmax to the network output to produce an allocation distribution where no item is overallocated. In an integer linear program, there is no easy way to exactly represent the softmax. While a piecewise linear overapproximation might be possible,

we elect instead to replace the softmax with the *sparsemax* [85]. Both softmax and sparsemax project vectors onto the simplex, but the sparsemax performs a Euclidean projection:

$$\text{sparsemax}(x) = \arg\min_z \frac{1}{2}\|x - z\|_2^2 \text{ s. t. } 1^T z - 1 = 0, 0 < z < 1 \tag{4.5}$$

([85] describes a cheap exact solution to this optimization problem and its gradient which are used during training. We use a PyTorch layer provided in [98, 99].)

In order to encode this activation function in our integer program, we can write down its KKT conditions and add them as constraints (a standard approach for bilevel optimization [100]), as shown in (4.6).

These constraints are all linear, except for the complementary slackness constraints – however, these can be represented as SOS1 constraints in Gurobi and other solvers.

The payment head also uses a sigmoid nonlinearity; we simply replace this with a piecewise linear function similar to a sigmoid.

$$\begin{aligned} (z - x) + \mu_1 - \mu_2 + \lambda 1 &= 0 \\ z - 1 \leq 0, -z \leq 0, 1^T z - 1 &= 0 \\ \mu_1 \geq 0, \mu_2 &\geq 0 \\ \mu_1(z - 1) = 0, \mu_2(-z) &= 0 \end{aligned} \tag{4.6}$$

4.3.2 Enforcing individual rationality

The RegretNet architecture imposes individual rationality – the requirement that no agent should pay more than they win – by multiplying with a fractional payment head, so that each player's payment is always some fraction of the value of their allocation distribution.

When trying to maximize utility (in order to maximize regret), this poses a problem. The

utility for player i, with input bids $\boldsymbol{b}_i$, is $u_i(\boldsymbol{b}_i) = \sum_j a_{ij} v_{ij} - p_i$. The value of the allocation is a linear combination of variables with fixed coefficients. But $p_i = \tilde{p}_i \left(\sum_j a_{ij} \boldsymbol{b}_{ij} \right)$ – this involves products of variables, which cannot be easily represented in standard integer linear programs.

We propose two solutions: we can either formulate and solve a nonconvex integer program (with bilinear equality constraints), or remove the IR constraint from the architecture and attempt to enforce it via training instead.

Nonconvex integer programs The latest version of Gurobi can solve programs with bilinear optimality constraints to global optimality. By adding a dummy variable, we can chain together two such constraints to represent the final payment: $p_i = \tilde{p}_i y$, and $y_i = \sum_j a_{ij} b_{ij}$. It is desirable to enforce IR constraints at the architectural level, but as described in the experiments section, it can potentially be much slower.

Individual rationality penalty As opposed to constraining the model architecture to enforce individual rationality constraint, we also experiment with enforcing the constraint through an additional term in the Lagrangian (a similar approach was used in an earlier version of [75]). We can compute the extent to which individual rationality is violated:

$$\mathrm{irv}_i = \max(p_i - \sum_j a_i b_i, 0) \tag{4.7}$$

We then allow the network to directly output a payment, but add another penalty term to encourage individual rationality:

$$\mathcal{L}(\boldsymbol{v}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = -\sum_i p_i + \sum_i \lambda_i \widehat{\mathrm{rgt}}_i(\boldsymbol{v}) + \frac{\rho}{2} \sum_i \left(\widehat{\mathrm{rgt}}_i(\boldsymbol{v}) \right)^2 + \sum_i \mu_i \, \mathrm{irv}_i^2 \tag{4.8}$$

With this approach, the final payment no longer involves a product between allocations, bids, and payment head, so the MIP formulation does not have any quadratic constraints.

Distillation loss Training becomes quite unstable after adding the individual rationality penalty; we stabilize the process using distillation [101]. Specifically, we train a teacher network using the original RegretNet architecture, and use a mean squared error loss between the student and the teacher's output to train our network. The teacher may have an approximately correct mechanism, but is difficult to certify; using distillation, we can train a similar student network with an architecture more favorable for certification. Additionally, combined with the individual rationality penalty in the Lagrangian, distilling from a teacher network which enforces IR by construction also allows us to learn a student network which is discouraged from violating IR.

We allow the payments to vary freely during distillation training, to avoid vanishing gradients, and simply clip the payments to the feasible range after the training is done. Through this method, empirically, we are able to train student networks that are comparable to the teachers in performance.

4.3.3 Regularization for fast certificates

[102] point out that a major speed limitation in integer-programming based verification comes from the need to branch on integer variables to represent the ReLU nonlinearity (see Equation 4.4). However, if a ReLU unit is *stable*, meaning its input is always only positive or only negative, then there is no need for integer variables, as its output is either linear or constant respectively.

We adopt the approach in [102], which at train time uses interval bound propagation [103]

to compute loose upper and lower bounds on each activation, and adds a regularization term $-\tanh(1+ul)$ to encourage them to have the same sign. At verification time, variables where upper and lower bounds (computed using the tighter Planet relaxation) are both positive or both negative do not use the costly formulation of Equation 4.4.

4.4 Experiments

We experiment on two auction settings: 1 agent, 2 items, with valuations uniformly distributed on $[0, 1]$ (the true optimal mechanism is derived analytically and presented by [71]); and 2 agents, 2 items, with valuations uniformly distributed on $[0, 1]$, which is unsolved analytically but shown to be empirically learnable in [75].

For each of these settings, we train 3 networks:

- A network with a sparsemax allocation head which enforces individual rationality using the fractional payment architecture, and uses the ReLU stability regularizer of [102]
- The same architecture, without ReLU regularization
- A network that does not enforce IR, trained via distillation on a network with the original RegretNet architecture

Additionally, to investigate how solve time scales for larger auctions, we consider settings with up to 3 agents and 3 items for the architecture without IR enforcement. All training code is implemented using the PyTorch framework [104].

4.4.1 Training procedure

We generate 600,000 valuation profiles as training set and 3,000 valuation profiles as the testing set. We use a batch size of 20,000 for training, and we train the network for a total of 1000 epochs. At train time, we generate misreports through 25 steps of gradient ascent on the truthful valuation profiles with learning rate of .02; at test time, we use 1000 steps. Architecturally, all our networks use a shared trunk followed by separate payment and allocation heads; we find the use of a shared embedding makes the network easier to certify. We generally use more layers for larger auctions, and the detailed architectures, along with hyperparameters of the augmented Lagrangian, are reported in Chapter 4.6.1.

Auction Setting	**IR**	**Relu Reg.**	**Solve time (s)**	**Revenue**	**Empirical Regret**	**Certified Regret**	**Emp./Cert. Regret**
1x2	Yes	No	25.6 (72.0)	0.593 (0.404)	0.014 (0.012)	0.019 (0.016)	0.731
1x2	Yes	Yes	7.2 (17.5)	0.569 (0.390)	0.003 (0.002)	0.004 (0.003)	0.700
1x2	No	Yes	0.034 (0.007)	0.568 (0.398)	0.009 (0.005)	0.011 (0.004)	0.839
2x2	Yes	No	13.9 (37.0)	0.876 (0.286)	0.009 (0.013)	0.014 (0.016)	0.637
2x2 (2nd)	Yes	No	17.4 (51.9)	—	0.007 (0.011)	0.011 (0.013)	0.676
2x2	Yes	Yes	5.8 (16.3)	0.874 (0.285)	0.008 (0.012)	0.013 (0.015)	0.626
2x2 (2nd)	Yes	Yes	7.520 (24.2)	—	0.008 (0.012)	0.012 (0.014)	0.680
2x2	No	Yes	5.480 (5.577)	0.882 (0.334)	0.006 (0.007)	0.011 (0.011)	0.533
2x2 (2nd)	No	Yes	2.495 (2.271)	—	0.011 (0.010)	0.017 (0.017)	0.666

Table 4.1: Summary of experimental results. Empirical regret is computed on 3000 random points and certified regret is computed on 1000 different points. (2nd) denotes the second agent in a multi-agent auction. Note that average empirical regret is only about 60-80% of the average true regret. The number in the parenthesis represents the standard deviation.

4.4.2 Results

Our results for regret, revenue and solve time are summarized in Table 4.1. We show the relationship between truthful and strategic bids for a learned 1 agent, 2 item mechanism in Figure

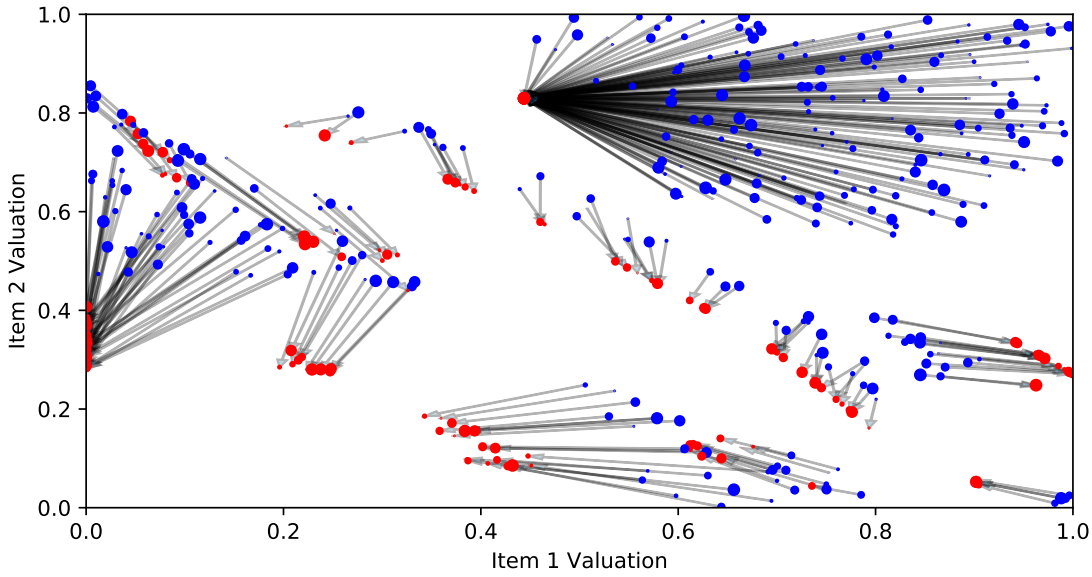


Figure 4.1: For the 1 agent, 2 item setting (regularized, IR enforced), this plot shows truthful bids (blue circle), with an arrow to the best strategic bid computed by the certifier (red filled). Only points with regret at least 0.005 are shown; the size of markers is proportional to the magnitude of regret. While the truthful and strategic bids are often far apart, this does not necessarily mean that violations of strategyproofness are large; in this plot, the highest regret of any point is still only 0.014.

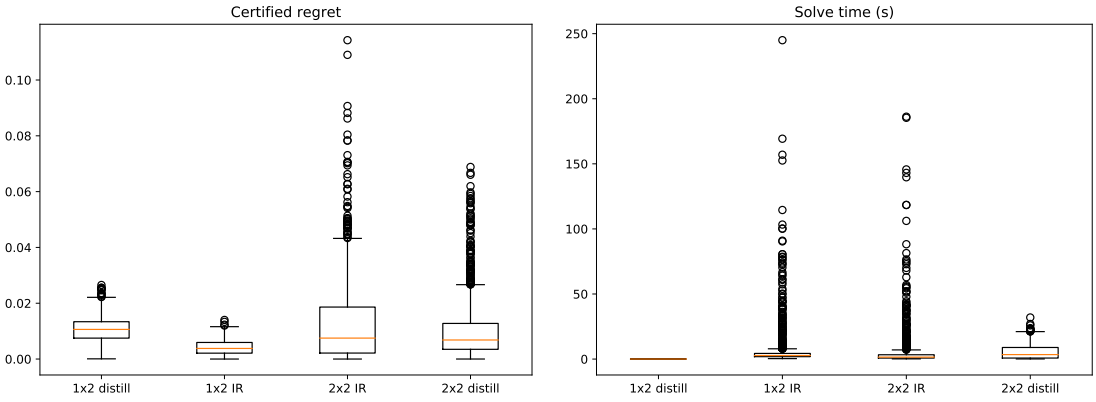


Figure 4.2: Certified regret and solve time for 1000 random points for IR and non-IR network architectures (regularized). Maximum utility in these settings is 2.0, so regrets are relatively small in most regions. At points with high regret, our certificates are able to detect this deficiency.

4.1.

Regret certificate quality We are able to train and certify networks with reasonably low regret – usually less than one percent of the maximum utility an agent can receive in these settings. Although mean regrets are small, the distributions are right skewed (particularly in the 2 agent, 2 item case) and there are a few points with high (approximately 0.1) regret. Crucially, we find that our certified regrets tend to be larger on average than PGD-based empirical regret, suggesting that our method reveals strategic behaviors that gradient-based methods miss.

Trained revenue As a baseline we consider the mean revenue from selling each item individually in a Myerson auction – in the 1 agent 2 item setting, this is 0.5; in the 2 agent 2 item setting, it is 0.8333. Our trained revenues exceed these baselines. For the 1 agent 2 item settings, the optimal mechanism [71] has a revenue of 0.55; our mechanisms can only exceed this because they are not perfectly strategyproof.

Individual rationality Empirically, the individual rationality penalty is very effective. On average, less than 5.53% of points give an allocation violating the individual rationality constraint, and even if it is violated, the magnitude of violation is on average less than .0002 (Table 4.4, Appendix 4.6.2). Filtering out IR-violating points after the fact results in lower revenue but by less than one percent.

Solve time The time required to solve the MIP is also quite important. In general, we find that ReLU stability regularization helps speed up solve time, and that solving the bilinear MIP (required for architectural enforcement of IR) is much harder than solving the mixed-integer

linear program for the other architecture.

Auction setting	Mean solve time (s)	Solve time std	Regret	Regret std
2x3	160.66	142.86	0.0342	0.0169
3x2	5.039	3.40	0.0209	0.0152
3x3	71.81	54.24	0.0243	0.0204

Table 4.2: Solve times and regrets for non-IR architecture without clipped payments in larger settings on 250 random points. In general, increasing the number of items significantly slows down certification.

To investigate scalability, we also consider solve times and certified regrets for settings with larger numbers of agents and items; results are summarized in Table 4.2. Our experiments use the non-IR-enforcing architecture; additionally, for these experiments we do not apply hard clipping of payments. In general, increasing the number of items significantly increases the solve time – this is not too surprising, as increasing the number of items increases the dimensionality of the space that must be certified (while the same is not true for increasing the number of agents, because certificates are for one agent only). The larger solve time for 2 rather than 3 agents is harder to explain – it may simply be the result of different network properties or a more complex learned mechanism.

We note that both solve time and regret are heavily right-skewed, as shown in Figure 4.2. We also find that the difference between allocations, payments, and utilities computed by the actual network and those from the integer program is on the order of 10^{-6} – the constraints in the model correctly represent the neural network.

4.5 Chapter Summary and Future Work

Our MIP solution method is relatively naive. Using more advanced techniques for presolving, and specially-designed heuristics for branching, have resulted in significant improvements in speed and scalability for certifying classifiers [83, 84]. Our current work serves as strong proof-of-concept validation that integer-programming-based certifiability can be useful in the auction setting, and it is likely that these techniques could be applied in a straightforward way to yield a faster certifier.

The performance of our learned mechanisms is also not as strong as those presented in [75], both in terms of regret and revenue. It is unclear to us whether this is due to differences in the architecture or to hyperparameter tuning. We observe that our architecture has the capacity to represent the same class of functions as RegretNet, so we are hopeful that improved training might close the gap. The recent paper [105] finds that RegretNet is indeed very sensitive to hyperparameters, and presents an improved algorithm for auction learning which is less sensitive. The neural architectures used with this new algorithm are essentially the same as RegretNet and can also be modified to allow for certificates.

In addition to generalization bounds provided by [75], other work has dealt with the problem of estimating expected strategyproofness given only regret estimated on samples from the valuation distribution [106]. The methods presented in this work for solving the utility maximization problem have the potential to complement these bounds and techniques.

In this chapter, we have described a modified version of the RegretNet architecture for which we can produce certifiable bounds on the maximum regret which a player suffers under a given valuation profile. Previously, this regret could only be estimated empirically using a

gradient ascent approach which is not guaranteed to reach global optimality. We hope that these techniques can help both theorists and practitioners have greater confidence in the correctness of learned auction mechanisms.

Broader Impact

The immediate social impact of this work will likely be limited. Learned auction mechanisms are of interest to people who care about auction theory, and may eventually be used as part of the design of auctions that will be deployed in practice, but this has not yet happened to our knowledge. We note, however, that the design of strategyproof mechanisms is often desirable from a social good standpoint. Making the right move under a non-strategyproof mechanism may be difficult for real-world participants who are not theoretical agents with unbounded computational resources. The mechanism may impose a real burden on them: the cost of figuring out the correct move. By contrast, a strategyproof mechanism simply requires truthful reports—no burden at all.

Moreover, the knowledge and ability to behave strategically may not be evenly distributed, with the result that under non-strategyproof mechanisms, the most sophisticated participants may game the system to their own benefit. This has happened in practice: in Boston, some parents were able to game the school choice assignment system by misreporting their preferences, while others were observed not to do this; on grounds of fairness, the system was replaced with a redesigned strategyproof mechanism [107].

Thus, we believe that in general, the overall project of strategyproof mechanism design is likely to have a positive social impact, both in terms of making economic mechanisms easier to participate in and ensuring fair treatment of participants with different resources, and we hope

we can make a small contribution to it.

4.6 Chapter Appendix

4.6.1 Architectural and Training Details

We initialized the Lagrange multiplier of regret (λ_i) as 5, and update it every 6 batches, and we experiment with values for the constant ρ^{rgt} ranging between 0.5 to 2 (reporting the choice that gave the lowest regret). For the IR violation penalty, we initialize the Lagrange multiplier of IR violation (μ_i) as 20, and update the Lagrange multiplier every 6 iterations. μ is initialized as 5, and then incremented by 5 every 5 batches. For distillation, we take a mean squared error loss between the student and teacher's output, and use a multiplier of $\frac{1}{400}$. Specifically, the Lagrange multipliers are updated as follows.

$$\lambda_{i+1} = \lambda_i + \rho^{\text{rgt}} \widehat{\text{rgt}}_i \qquad \rho^{\text{rgt}}_{i+1} = \rho^{\text{rgt}}_i + \rho^{\text{rgt}}_{inc}$$

$$\mu_{i+1} = \mu_i + \rho^{\text{irv}} \, \text{irv}_i \qquad \rho^{\text{irv}}_{i+1} = \rho^{\text{irv}}_i + \rho^{\text{irv}}_{inc}$$

Auction Setting	**Inner Product**	**Relu Stability Regularizer**	**Embedding Layer**
1 Agent x 2 Items	Yes	No	1 hidden layer x 128 units
1 Agent x 2 Items	Yes	Yes	1 hidden layer x 128 units
1 Agent x 2 Items	No	Yes	1 hidden layer x 128 units
2 Agents x 2 Items	Yes	No	2 hidden layer x 128 units
2 Agents x 2 Items	Yes	Yes	2 hidden layer x 128 units
2 Agents x 2 Items	No	Yes	2 hidden layer x 128 units

4.6.2 Additional Experimental Information

Hardware All certification experiments were conducted on an AMD Ryzen 3600X CPU with 32GB RAM. Training of the network was conducted with a 2080 GPU on a university compute cluster.

Additional experiments Table 4.4 shows more detailed results for the non-IR-enforcing architecture. IR violations are relatively small, and filtering out these cases (sacrificing revenue) does not harm overall revenue too much.

Table 4.3 shows the results of scaling experiments for settings with more agents and items, in a setting where payment clipping is applied. Again, increasing the dimensionality of the input space by increasing the number of items seems to impose a greater cost than increasing the number of agents.

Auction setting	**Mean solve time (s)**	**Regret**
2x3	109.749 (159.212)	0.027 (0.016)
3x2	3.033 (2.377)	0.019 (0.016)
3x3	59.173 (53.431)	0.022 (0.020)

Table 4.3: Solve times and regrets for non-IR architecture with clipped payments in larger settings on 250 random points. In general, increasing the number of items significantly slows down certification. Standard deviations are in parentheses.

Joint work with Michael Curry, Tom Goldstein, John Dickerson first published in NeurIPS 2020

Auction Setting	% of IR violation	Max IR violation	Mean IR violation	Revenue before enforcing IR	Revenue after enforcing IR
1x2	5.53%	0.0053	0.0001 (0.0003)	0.5738	0.5681
2x2	4.60%	0.0083	0.0002 (0.0007)	0.8874	0.8824

Table 4.4: IR violation for the 1x2/2x2 auction settings. Note that the mean IR violation is small, and revenue after enforcing IR drops only slightly. The number in parenthesis represents the standard deviation.

Chapter 5: Certified Neural Network Watermarks with Randomized Smoothing

5.1 Introduction

With the rise of deep learning, there has been an extraordinary growth in the use of neural networks in various computer vision and natural language understanding tasks. In parallel with this growth in applications, there has been exponential growth in terms of the cost required to develop and train state-of-the-art models [108]. For example, the latest GPT-3 generative language model [109] is estimated to cost around 4.6 million dollars [110] in TPU cost alone. This does not include the cost of acquiring and labeling data or paying engineers, which may be even greater. With up-front investment costs growing, if access to models is offered as a service, the incentive is strong for an adversary to try to steal the model, sidestepping the costly training process. Incentives are equally strong for companies to protect such a significant investment.

Watermarking techniques have long been used to protect the copyright of digital multimedia [111]. The copyright holder hides some imperceptible information in images, videos, or sound. When they suspect a copyright violation, the source and destination of the multimedia can be identified, enabling appropriate follow-up actions [111]. Recently, watermarking has been extended to deter the theft of machine learning models [112, 113]. The model owner either imprints a predetermined signature into the parameters of the model [112] or trains the model to give predetermined predictions [113] for a certain trigger set (e.g. images superimposed with a

predetermined pattern).

A strong watermark must also resist removal by a motivated adversary. Even though the watermarks in [112, 113, 114] initially claimed some resistance to various watermark removal attacks, it was later shown in [115, 116] that these watermarks can in fact be removed with more sophisticated methods, using a combination of distillation, parameter regularization, and finetuning. To avoid the cat-and-mouse game of ever-stronger watermark techniques that are only later defeated by new adversaries, we propose a certifiable watermark: unless the attacker changes the model parameters by more than a certain ℓ_2 distance, the watermark is guaranteed to remain.

To the best of our knowledge, our proposed watermarking technique is the first to provide a certificate against an ℓ_2 adversary. We also analyzed whether ℓ_2 adversary is a reasonable threat model as well as the magnitude of appropriate defense radius. Surprisingly, we find our certified radius to be quite substantial relative to the range of meaningful radius that one could certify. Additionally, we empirically find that our certified watermark is more resistant to previously proposed watermark removal attacks [115, 116] compared to its counterparts – it is thus valuable even when a certificate is not required.

5.2 Related Work

Watermark techniques [112] proposed the first method of watermarking neural networks: they embed the watermark into the parameters of the network during training through regularization. However, the proposed approach requires explicit inspection of the parameters for ownership verification. Later, [113, 117] improved upon this approach, such that the watermark can be

verified through API-only access to the model. Specifically, they embed the watermark by forcing the network to deliberately misclassify certain "backdoor" images. The ownership can then be verified through the adversary's API by testing its predictions on these images.

In light of later and stronger watermark removal techniques [115, 116, 118], several papers have proposed methods to improve neural network watermarking. [119] propose an improved white-box watermark that avoids the detection and removal techniques from [118]. [120] propose using values outside of the range of representable images as the trigger set pattern. They show that their watermark is quite resistant to a finetuning attack. However, since their trigger set does not consist of valid images, their method does not allow for black-box ownership verification against a realistic API that only accepts actual images, while our proposed watermark is effective even in the black-box setting.

[121] proposed watermarking methods for models housed behind an API. Unlike our method, their method does not embed a watermark into the model weights itself, and so cannot work in scenarios where the weights of the model may be stolen directly, e.g. when the model is housed on mobile devices.

Finally, [122] propose using a particular type of adversarial example ("conferrable" adversarial examples) to construct the trigger set. This makes the watermark scheme resistant even to the strongest watermark removal attack: ground-up distillation which, starting from a random initialization, trains a new network to imitate the stolen model [115]. However, for their approach to be effective, they need to train a large number of models (72) on a large amount of data (e.g. requiring CINIC as opposed to CIFAR-10). While our approach does not achieve this impressive resistance to ground-up distillation, it is also much less costly.

Watermark removal attacks However, one concern for all these watermark methods is that a sufficiently motivated adversary may attempt to remove the watermark. Even though [112, 113, 114, 117] all claim that their methods are resistant to watermark removal attacks, such as finetuning, other authors [115, 116] later show that by adding regularization, finetuning and pruning, their watermarks can be removed without compromising the prediction accuracy of the stolen model. [118] shows that the watermark signals embedded by [112] can be easily detected and overwritten; [123] shows that by leveraging both labeled and unlabeled data, the watermark can be more efficiently removed without compromising the accuracy. Even if the watermark appears empirically resistant to currently known attacks, stronger attacks may eventually come along, prompting better watermark methods, and so on. To avoid this cycle, we propose a certifiably unremovable watermark: given that parameters are not modified more than a given threshold ℓ_2 distance, the watermark will be preserved.

Certified defenses for adversarial robustness Our work is inspired by recent work on certified adversarial robustness, [124, 125, 126, 127, 128, 129, 130, 131]. Certified adversarial robustness involves not only training the model to be robust to adversarial attacks under particular threat models, but also proving that no possible attacks under a particular constraint could possibly succeed. Specifically, in this chapter, we used the randomized smoothing technique first developed by [54, 124] for classifiers, and later extended by [132] to deal with regression models. However, as opposed to defending against an ℓ_2-bounded threat models in the image space, we are now defending against an ℓ_2-bounded adversary in the parameter space. Surprisingly, even though the certificate holds only when randomized smoothing is applied, empirically, when our watermark is evaluated in a black-box setting on the non-smoothed model, it also exhibits stronger persistence

compared to previous methods.

Certified watermark The only other work that we have found that proposes certified watermarks is [133]. In [133], they propose a technique to find the minimal modification required to remove watermark in a neural network. Our proposal differs from theirs in two ways. First, they do not propose methods to embed a watermark that would be more resilient, rather they simply find the minimal change required to remove a watermark. On the other hand, our proposed watermark is empirically more resistant compared to previous approaches. Second, their approach is based on solving mixed integer linear programs and thus does not scale well to larger networks. For example, in their experiment, they were only able apply their technique on a network with 150 hidden neurons for MNIST [133]. In contrast, our method can be easily applied to any modern architecture: we use ResNet-18 for all of our experiments.

5.3 Methods

Below, we introduce the formal model for neural network watermarking, and the watermark removal adversaries that we are concerned with. Then, we describe some background related to randomized smoothing, and show that by using randomized smoothing we can create a watermark that provably cannot be removed by an ℓ_2 adversary.

5.3.1 Watermarking

White box vs black box We first introduce the distinction between black box and white box settings from the perspective of the owner of the model. In a white box setting, parameters are known. In a black box setting, the model parameters are hidden behind an API. We consider

cases where the owner may have either black box or white box access to verify their watermarks.

Black-box watermarking In backdoor-based watermarking, the owner employs a "trigger set" of specially chosen images that has disjoint distribution compared to the original dataset. If another model makes correct predictions on this trigger set, then this is evidence that the model has been stolen. A backdoor-based watermark can be verified in a black-box setting.

The trigger set may be chosen in various ways. [113] considered three different methods of generating the trigger set: embedded content, pre-specified noise, and abstract images. Embedded content methods embed text over existing datasets and assigns all examples with the text overlay the same fixed label. Pre-specified noise overlays Gaussian noise on top of existing dataset and again assigns the examples with the same fixed label. For abstract images, a set of images from a different domain is additionally used to train the network. For example, MNIST images could form the trigger set for a CIFAR-10 network, so if an adversary's model performs exceedingly well on MNIST images, then the adversary must have used the stolen model. Examples of trigger set images are presented can be found in Appendix - Figure 5.2.

Our proposed method builds upon such backdoor-based watermarks, so our marked model can also naturally be verified in the black-box manner even though our certificate is only valid in the white-box setting described in the next section.

White-box watermarking White-box watermarks in general embed information directly into the parameters. Our proposed watermark does not directly embed information into parameters, but parameter access is required for verification, so it is still a white-box watermark. The rationale for using such a white-box watermark is detailed below.

In the black-box setting, to verify model ownership, we generally check that the trigger set accuracy function from parameters to accuracy $f(\theta)$ is larger than a threshold [115]. The trigger set accuracy function takes in model parameter as input and outputs the accuracy on the trigger set. Since directly certifying the function is hard, we first convert the trigger set accuracy function $f(\theta)$ to its smoothed counterpart $h(\theta)$, and then check that $h(\theta)$ is greater than the threshold t for ownership verification. Practically, one converts the base function to the smoothed function by injecting random noise into the parameters during multiple trigger set evaluations, and then taking the median trigger set accuracy as $\hat{h}$. Note that this verification process *requires* access to parameters, so ownership verification using $\hat{h}$ is considered a *white-box* watermark.

Watermark Removal Threat Model In our experiments, we consider three different threat models to the watermark verification: 1) distillation, 2) finetuning, and 3) an ℓ_2 adversary.

In the distillation threat model (1), we assume that the adversary initializes their model with our original model, and then trains their model with distillation using unlabeled data that comes from the same distribution. In other words, the adversary uses our original model to label the unlabeled data for finetuning. [115] propose first adding some regularization during the initial part of the attack to remove the watermark, and then later turning off the regularization to fully recover the test accuracy of the model. We experiment with this distillation attack both with and without regularization.

In the finetuning threat model (2), the adversary has its own labeled dataset from the original data-generating distribution. This adversary is strictly stronger compared to the distillation threat model. In our experiments, we make the conservative assumption that the adversary has exactly the same amount of data as the model owner.

The ℓ_2 adversary (3) obtains the original model parameters, and then is allowed to move the parameters at most a certain ℓ_2 distance to maximally decrease trigger set accuracy. Even though the ℓ_2 adversary is not a completely realistic threat model, we argue similarly to the adversarial robustness literature [134] that being able to defend against a small ℓ_2 adversary is a requirement for defending against more sophisticated attacks. In our experiments, we empirically find that a large shift of parameters in ℓ_2 distance is indicative of the strength of the adversary. For example, training the models for more time, with a larger learning rate, or using ground truth labels as opposed to distillation are all stronger attacks, and as expected, they both remove the watermark faster and move the parameters by a greater ℓ_2 distance (Table 5.4). Additionally, given a local Lipschitz constant of L and a learning rate of r, the number of steps required to move outside of the ϵ-ℓ_2 ball can be upper bounded by $\epsilon/(rL)$, and we think the number of steps is a good proxy for the computational budget of the adversary.

5.3.2 Watermark Certification

For our certificates, we focus on the ℓ_2 adversary described above: the goal of certification is to bound the worst-case decrease in trigger set accuracy, given that the model parameters do not move too far in ℓ_2 distance. Doing this directly is in general quite difficult [135], but using techniques from [124, 132], we show that by adding random noise to the parameters it is possible to define a smoothed version of the model and bound the change in its trigger set accuracy.

Deriving the certificate Before we start describing the watermark certificate, we will first introduce the percentile smoothed function from [132].

Definition 2. Given $f : \mathbb{R}^d \to \mathbb{R}$ and $G \sim N(0, \sigma^2 I)$, we define the *percentile smoothing* of f as

$$\underline{h}_p(x) = \sup\{y \in \mathbb{R} \mid \mathbb{P}[f(x+G) \leq y] \leq p\} \tag{5.1}$$

$$\overline{h}_p(x) = \inf\{y \in \mathbb{R} \mid \mathbb{P}[f(x+G) \leq y] \geq p\} \tag{5.2}$$

As mentioned in [132], the two forms $\underline{h}_p$ and $\overline{h}_p$ are needed to handle edge cases with discrete distributions. While h_p may not admit a closed form, we can approximate it by Monte Carlo sampling [124].

There are some differences from existing adversarial robustness work in how we apply these bounds. First, while the robustness literature applies the smoothing results to bound outputs of the classifier itself, we apply smoothing over the trigger set accuracy function to bound changes in trigger set accuracy. Second, we are applying smoothing over parameters as opposed to input. Our trigger set accuracy function $f(X, \theta)$ in general takes in two arguments: X, a set of images, and θ, the model parameters. In the case of adversarial robustness, the model parameters θ are constant after training while the attacker perturbs the image x. But in our case, the trigger set X remains constant and the adversary can only change θ. Therefore, to defend against our specific adversary, we apply smoothing over θ as opposed to X. Since the trigger set X is constant for our case, we simply write the trigger set accuracy function as $f(\theta)$ for the remaining part of the chapter.

In our proposed watermark, we use the median smoothed version ($h_{50\%}$) of the trigger set accuracy function for ownership verification. Empirically evaluating $h_{50\%}$ essentially involves adding noise to several copies of the model parameters, calculating trigger set accuracy for all of them, and taking the median trigger set accuracy. The details of evaluating smoothed trigger set

accuracy are described in Algorithm 3 in Appendix 5.6.1.

Even though the evaluation process of $h_{50\%}$ is more involved compared to the base trigger set accuracy function, the smoothed version allows us to use Lemma 1 from [132] to bound the worst case change in the trigger set accuracy given bounded change in parameters, as shown in Corollary 1. We have delegated the proof of the corollary to Appendix 5.6.3.

Corollary 5.0.1. *Given a measureable trigger set accuracy function $f(\theta)$, the median smoothed trigger set accuracy function $h_{50\%}(\theta)$ can be lower bounded as follows*

$$\underline{h}_{\Phi(-\epsilon/\sigma)}(\theta) \leq h_{50\%}(\theta + \delta) \quad \forall \, \|\delta\|_2 < \epsilon, \tag{5.3}$$

when the adversary does not modify the model parameters θ by more than ϵ in terms of ℓ_2 norm. Φ is the standard Gaussian CDF.

Using the above corollary, we can then bound the worst case trigger set accuracy given the ϵ adversary by evaluating $\underline{h}_{\Phi(-\epsilon/\sigma)}(x)$. Even though $\underline{h}_{\Phi(-\epsilon/\sigma)}(x)$ does not have a closed form, we can calculate an empirical estimator that would lower bound it with sufficient confidence c. We detail steps for calculating the estimator in Algorithm 3 in Appendix 3.

Trigger set accuracy and model ownership In this chapter, we assume a sufficiently high trigger set accuracy implies ownership with high probability. However, there are some scenarios where the assumption does not hold, which we will clarify below. Whether high trigger set accuracy implies ownership depends heavily on the trigger set selected. For example, if the trigger set (X, Y) selected has labels corresponding to what most people would consider to be correct classes, then a model developed independently by someone else would likely classify such trigger sets

correctly, leading to incorrect ownership assignment. However, if the trigger set consists of wrong or meaningless labels (such as dog images paired withi cat labels), then an independently developed model is very unlikely to classify such trigger sets correctly. In this chapter, we assume that the trigger set examples selected have a probability less than random chance of being classified correctly by a random model, and that a sufficiently high trigger set accuracy implies ownership. Our certificate is only focused on proving the preservation of trigger set accuracy when the adversary is allowed to move parameters within a certain ℓ_2 norm ball.

Algorithm 2 Embed Certifiable Watermark

Required: training samples X, trigger set samples $X_{trigger}$, learning rate τ, maximum noise level ϵ, replay count k, noise sample count t

for epoch = 1, ... , N **do**
  for $B \subset X$ **do**
    $g_\theta \leftarrow E_{(x,y)\in B}[\nabla_\theta l(x, y, \theta)]$
    $\theta \leftarrow \theta - \tau g_\theta$
  for $B \subset X_{trigger}$ **do**
    $g_\theta = 0$
    for i = 1 to k **do**
      $\sigma \leftarrow \frac{i}{k}\epsilon$
      for j = 1 to t **do**
        $G \sim N(0, \sigma^2 I)$
        $g_\theta \leftarrow g_\theta + E_{(x,y)\in B}[\nabla_\theta l(x, y, \theta + G)]$
      $g_\theta \leftarrow g_\theta/(kt)$
      $\theta \leftarrow \theta - \tau g_\theta$

Embedding the Certifiable Watermark To embed the watermark during training, we add Gaussian noise and train on the trigger set images with the desired labels. For a given trigger set image, we average gradients across several (in our experiments, 100) draws of noise to better approximate the gradient of the smoothed classifier. Directly adding a large amount of noise into all parameters makes training unstable, so we incrementally increase the levels of noise within each epoch. In our experiments, we inject Gaussian noise with a range of standard deviations σ ranging from

0 to 1. Empirically, we notice that the test accuracy drops when using this technique to embed the watermark, so to recover some of the lost test accuracy, we warm up the model with regular training and only begin embedding the watermark after the fifth epoch. We note that using warm-up epochs to recover clean accuracy is a common practice in the robustness literature [136, 137]. The detailed training method is described in Algorithm 2.

5.4 Experiments

In our first set of experiments, we investigate the strength of our certificate under two datasets and three watermark schemes. In our second set of experiments, we evaluate the watermark's empirical robustness to removal compared to previous methods that claimed resistance to removal attacks. The code for all these experiments is publically available [1].

5.4.1 Experimental Settings

To produce the trigger sets themselves, we consider the three schemes from [113]: images with embedded content (superimposed text), images with random noise, or images from an unrelated dataset (CIFAR-10 for MNIST and vice versa) (Figure 5.2). While we generated certificates for all three schemes, we focus on embedded content watermark for empirical persistency evaluation.

To train the watermarked model, we used ResNet-18, SGD with learning rate of .05, momentum of .9, and weight decay of 1e-4. The model is trained for 100 epochs, and the learning rate is divided by 10 every 30 epochs. Only 50% of the data is used for training, since we reserve

[1] A PyTorch implementation of Certified Watermarks is available at `https://github.com/arpitbansal297/Certified_Watermarks`

Attack Radius	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8
Worst Case Accuracy	85.8%	82.5%	80.5%	76.2%	67.1%	56.1%	32.0%	18.4%	8.4%

Table 5.1: Attack Radius vs Worst Case Accuracy of the Model. It becomes meaningless to defend against a threat model with a radius larger than 1.8 because these models are indistinguishable from any randomly initialized model.

the other half for the adversary. For our watermark models, we select σ of 1, replay count of 20, and noise sample count of 100. Given these training parameters, embedding the watermark increase the compute time by two times compared to regular training. For certification, we use 10000 instances of Monte Carlo sampling to perform smoothing.

To attack the model, we used Adam with learning rates of .1, .001 or .0001 for 50 epochs. We test three different types of attacks: finetuning, hard-label distillation, and soft-label distillation. Soft-label distillation takes the probability distribution of the original model as labels, whereas hard-label distillation takes only the label with maximum probability. We always give the adversary the same amount of data as the owner (labeled for finetuning, unlabeled for distillation) to err on the conservative side for our evaluation.

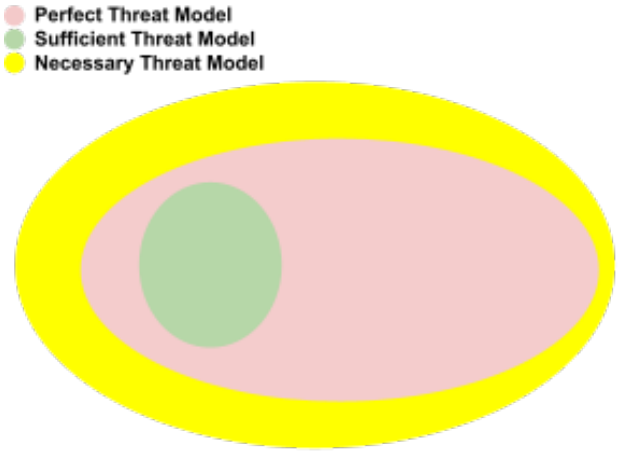


Figure 5.1: Graphical illustration of perfect, sufficient, and necessary threat models.

5.4.2 Properties of a Good Threat Model

Before we present our experimental results for watermark certification, we first briefly discuss properties of a good threat model and what an appropriate radius to certify would be.

In [138], the author defines a *perfect threat model* as being able to capture all items that are similar to the current example. In the case of adversarial attacks, we would like to capture all modified images that are similar to the original image. In the case of watermark removal, we would like to cover all attacked models that have similar test accuracy as the original model.

However, specifying a perfect threat model is often impossible for an obvious reason: we do not have an oracle measure of human perceptual similarities nor can we easily specify constraints that capture all models with similar test accuracy. As a result, researchers often have to trade off between two different imperfect threat models: a *sufficient threat model* and a *necessary threat model*. The sufficient threat model is a subset of the perfect threat model whereas the necessary threat model is a superset of the perfect threat model as illustrated in the Figure 5.1.

Most prior works prioritize the sufficiency criteria over the necessary criteria since being able to defend against a sufficient threat model is a requirement for being able to defend against the perfect threat model. In the watermark setting, being able to retain the trigger set accuracy (watermark) within the ℓ_2 norm ball of the parameter space is thus a prerequisite for defending against the perfect threat model which would include models outside of the ℓ_2 norm ball.

Appropriate Radius to Certify Here, we analyze the appropriate ℓ_2 norm constraint based on the sufficiency condition and find that a certified radius between 0 and 1.8 is appropriate and that our certified radius is indeed at a similar magnitude. In Table 5.1 on the right, we use a standard

PGD attack in the parameter space to gauge how much the accuracy of the model could decrease within the specified radius on the CIFAR-10 dataset. We ran 40 steps of PGD with a learning rate of 0.001, and the parameter gradient is calculated over 2560 examples. If we consider all models with higher than 80% test accuracy to be similar, then an ℓ_2 radius of 0.6 would satisfy the sufficiency condition. This is similar in magnitude to our certified radius presented in the next section, which is between 0.2 to 1.2. The determination of the appropriate radius is still dependent on some subjective judgement on what one would consider to be a well performing test accuracy. However, what we do know is that the appropriate radius should lie somewhere between 0 and 1.8 – defending against a radius larger than 1.8 is meaningless, as a model with only 8% accuracy on CIFAR-10 is indistinguishable from any randomly initialized model.

5.4.3 Watermark Certificate Evaluation

In this section, we investigate the certified trigger set accuracy that our watermarking is able to guarantee against ℓ_2 adversaries of various strengths. To further contextualize the meaning of a certified ℓ_2 radius, we consider the size of the empirical changes in parameters observed after performing various watermark removal attacks. Finally, we also study how one can increase the certificate by modifying the noise level.

As shown in Table 5.2, we are able to certify trigger set accuracy for radii up to 0.4 for all datasets and watermark schemes considered. This is quite a substantial radius when considering the sufficiency condition, which suggests a meaningful certificate does not exceed a radius of 1.8. Our certificate seems to be similarly effective across all trigger set types. In the best scenario for CIFAR-10, we can certify that the trigger set accuracy does not drop below 51% as long as parameters do not move more than an ℓ_2 distance of 1.

Dataset	Watermark	ℓ_2 Radius (ϵ) 0.2	0.4	0.6	0.8	1	1.2
MNIST	Embedded content	100%	95%	47%	3%	0%	0%
MNIST	Noise	100%	91%	7%	0%	0%	0%
MNIST	Unrelated	100%	94%	45%	4%	0%	0%
CIFAR-10	Embedded content	100%	100%	100%	93%	51%	5%
CIFAR-10	Noise	100%	100%	100%	100%	47%	0%
CIFAR-10	Unrelated	100%	100%	100%	97%	35%	0%

Table 5.2: Certified trigger set accuracy at different radius

Noise Level (σ)	Test Accuracy	Certified Watermark Accuracy ℓ_2 radius (ϵ) 0.2	0.4	0.6	0.8	1	1.2	1.4
1	86.00%	100.00%	100.00%	100.00%	93.00%	51.00%	5.00%	0.00%
1.1	84.56%	100.00%	100.00%	100.00%	97.00%	63.00%	13.00%	0.00%
1.2	84.18%	100.00%	100.00%	100.00%	100.00%	98.00%	74.00%	24.00%

Table 5.3: Trade-off between certified trigger set accuracy and noise level (σ) for CIFAR-10

To see how long our certificates can persist in the face of attack, we measure the approximate amount of ℓ_2 parameter change in the first epoch under different attack settings. In Table 5.4, with learning rate 0.0001, parameters change by ℓ_2 distance of approximately 2-3. In other words, it would require approximately 1/3 to 1/2 of an epoch to move outside of a certified radius of 1. (We focus here on the first epoch because changes are relatively small in succeeding epochs; see Appendix.)

Interestingly, attacks considered to be stronger correspond to changes of a greater distance. This relationship helps support the use of ℓ_2 radius as a proxy for the strength of the adversary. For example, fine-tuning has been found to be a stronger attack compared to hard label distillation, and correspondingly [115], fine-tuning moves the network by a larger distance in the first epoch compared to hard label distillation. Similarly, an attack that is stronger due to a higher learning rate moves the parameters much faster compared to an attack with a lower learning rate.

In Table 5.3, we show that one can obtain larger certificates by increasing the noise level. However, as one makes the model more robust against watermark removal, the model's test accuracy also decreases. This trade-off is similar to the trade-off observed in the adversarial robustness literature [2]. As the level of noise increases, training also becomes more unstable. For example, using the same hyperparameters as our other experiments, we were unable to train models with $\sigma = 1.5$. However, this is not to say that it is impossible to train a model with $\sigma = 1.5$. We did find an alternative setting where $\sigma = 1.5$ is trainable and offers higher robustness compared to $\sigma = 1.0 - 1.2$. However, since the hyperparameters are not the same, we do not list the results here as we don't think they are directly comparable.

Attack Type	Finetuning	Distillation Hard Label	Distillation Soft Label	Finetuning	Distillation Hard Label	Distillation Soft Label
Learning Rate	0.0001	0.0001	0.0001	0.001	0.001	0.001
MNIST	2.67	2.39	1.56	19.39	17.58	20.35
CIFAR-10	2.85	2.41	2.06	19.93	19.40	19.29

Table 5.4: ℓ_2 distance change in the first epoch

Overall, it would take approximately 0.03 to 0.3 epochs for the attacker to escape the certified radius, depending on the type of attack, watermark schemes, and dataset. Our certified bounds are substantial when considering the sufficiency criteria, but they are still quite small when compared to a non-ℓ_2 bounded attack. In the next section, we show that even though our certificates are not large when considering the optimization trajectory, the watermarks are empirically stronger than the certificate is able to guarantee: in most cases, our watermarks are more resistant to removal attacks compared to previous methods in both the white-box and black-box settings.

Dataset	Attack	lr	Baseline Watermark	Black-box Watermark	White-box Watermark
MNIST	Finetuning	0.0001	45.31%	59.38%	100.00%
MNIST	Finetuning	0.001	50.00%	54.70%	100.00%
MNIST	Hard-Label Distillation	0.001	42.19%	50.00%	100.00%
MNIST	Soft-Label Distillation	0.001	96.88%	100.00%	100.00%
CIFAR-10	Finetuning	0.0001	17.20%	9.40%	100.00%
CIFAR-10	Finetuning	0.001	14.06%	10.94%	100.00%
CIFAR-10	Hard-Label Distillation	0.001	29.69%	81.25%	100.00%
CIFAR-10	Soft-Label Distillation	0.001	81.25%	100.00%	100.00%
CIFAR-100	Finetuning	0.0001	18.75%	23.44%	100.00%
CIFAR-100	Finetuning	0.001	0.00%	0.00%	0.00%
CIFAR-100	Hard-Label Distillation	0.001	7.81%	12.5%	5.00%
CIFAR-100	Soft-Label Distillation	0.001	96.88%	96.88%	98.44%
MNIST	Hard-Label Distillation + Reg	0.1	40.63%	32.81%	0.00%
CIFAR-10	Hard-Label Distillation + Reg	0.1	8.00%	27.00%	0.00%
CIFAR-100	Hard-Label Distillation + Reg	0.1	0.00%	0.00%	0.00%

Table 5.5: Trigger set accuracy after 50 epochs of removal attacks. We note that this is only a snapshot of the trigger set accuracy. During training, trigger set accuracies could sometimes fluctuate significantly (see figures in Appendix). We use watermarks from [113] as the baseline watermark.

5.4.4 Empirical Watermark Persistence Evaluation

In this section, we evaluate the persistence of our proposed watermarking methods and the model's performance on the original dataset. For all experiments in this section, we use the embedded content method to produce the trigger set. We compare our watermark method with the baseline method from [113], which is the same as our watermark method but without noise injection during training. We further conduct additional attack evaluations in Appendix 5.6.7.

For persistence evaluation, we focus on two main attacks: the distillation attack and the finetuning attack, as both of these have been shown to be very effective in [115, 116]. In addition, we tested the effect of different learning rates and label smoothing levels, which have also been shown to influence the effectiveness of watermark removal techniques [115]. To make our attacks

more similar to [115], we also experimented with adding parameter regularization during attack.

We first evaluate our proposed watermark against finetuning attacks. In Table 5.5, we see that our proposed watermark is much more robust with respect to finetuning attacks than the baseline method on CIFAR-10, and is comparably resistant on MNIST. In the case of CIFAR-10, the baseline watermark is completely removed within less than 10 epochs (See Figure 1 in Appendix), but our white-box watermark is still visible after finetuning for up to 50 epochs. In the case of MNIST, both the proposed method and the baseline are quite resistant. However, our proposed method achieves slightly higher trigger set accuracy for both white-box watermarks and black-box watermarks throughout the 50 epochs of the finetuning attack. In the case of CIFAR-100, neither watermark is very resistant to removal. However, our blackbox watermark slightly outperforms the baseline method.

In the face of the distillation attack, we find our white-box watermark to be extremely resistant. The trigger set accuracy remains 100% even after 50 epochs of attack. However, our black-box watermark works more effectively on CIFAR-10 than MNIST. In the case of CIFAR-10, the black-box watermark remains at 81.25% after 50 epochs of distillation attack, whereas only 50.00% of trigger set accuracy remains for MNIST. In the case of CIFAR-100, our proposed watermark slightly outperforms the baseline method. However, they are both quite susceptible to removal attack.

When regularization is added in addition to distillation, we find that our white-box watermark is completely removed. This could be due to regularization moving the parameters further in terms of ℓ_2 norm. However, we note that our black-box watermark still persists similarly to the baseline.

In some cases, the baseline watermark persists quite strongly. For example, in the case

of soft-label distillation, the baseline watermark still achieves higher than 75% accuracy after attack. We tried a variety of settings, but we had difficulty completely removing the watermark as described in [115]. Differences in performance could be due to architecture, regularization, or other factors – experimental code was not released by [115], so it is hard to know exactly what might be the cause. However, we note that our main goal is to show that our proposed watermark is more resistant to removal, and our trigger set accuracy is consistently higher compared to the baseline throughout the attack.

Even though our watermark is generally more resistant in both the white-box and black-box settings, our proposed method does slightly decrease the accuracy of the model on the original dataset. Test accuracies are decreased by 0.1% (from 99.5% to 99.4%), 3.3% (from 89.3% to 86.0%), 1.1% (from 68.28% to 67.23%) for MNIST, CIFAR-10, and CIFAR-100 respectively. The decrease in clean accuracy has been historically observed for other forms of robust training [2], and recovery of the test accuracy in robust training is still an active area of research [136]. However, it is worth noting that the decrease in accuracy does not scale with the difficulty of the dataset. For example, even though CIFAR-100 is a much more challenging dataset compared to CIFAR-10, we actually observe smaller accuracy decrease for CIFAR-100.

5.5 Chapter Summary

We present a certifiable neural network watermark – trigger set accuracy is provably maintained unless the network parameters are moved by more than a given ℓ_2 distance. We see this as the first step towards guaranteed persistence of watermarks in the face of adversaries – a valuable property in real-world applications. We also analyzed the size of our certificate with respect to

the sufficiency criteria, and found that our certificates are indeed quite meaningful.

At the same time, we find that our certifiable watermarks are empirically far more resistant to removal than the certified bounds can guarantee. Indeed in the face of the removal attacks from the literature, our watermarks are more persistent than previous methods. Our randomized-smoothing-based training scheme is therefore a watermarking technique of interest even where a certificate is not needed. We are hopeful that our technique represents a contribution to both the theory and practice of neural network watermarking, and that this approach can lead to watermarks that are both empirically useful while coming with provable guarantees.

5.6 Chapter Appendix

5.6.1 Algorithm for evaluating the smoothed model

Algorithm 3 Evaluate and Certify the Median Smoothed Model

function TRIGGERSETACCURACY(f, θ, σ, n)
  $\hat{w} \leftarrow$ AddGaussianNoise(θ, σ, n) ▷ n simulations of noised parameter w
  $\hat{a} \leftarrow f(\hat{\theta})$ ▷ evaluate trigger accuracy for each simulation of w
  $\hat{a} \leftarrow$ Sort($\hat{a}$) ▷ Sort simulated accuracies
  $a_{median} \leftarrow \hat{a}_{\lfloor 0.5n \rfloor}$ ▷ Take the median
  return a_{median}
function TRIGGERSETACCURACYLOWERBOUND(f, θ, σ, ϵ, n, c)
  $\hat{\theta} \leftarrow$ AddGaussianNoise(θ, σ, n) ▷ n simulations of noised parameter w
  $\hat{a} \leftarrow f(\hat{\theta})$ ▷ evaluate trigger accuracy for each simulation of θ
  $\hat{a} \leftarrow$ Sort($\hat{a}$) ▷ Sort simulated accuracies
  $k \leftarrow$ EmpiricalPercentile(n, c, σ, ϵ) ▷ Algorithm 1 in Appendix
  $\underline{a} \leftarrow \hat{a}_k$ ▷ $\hat{a}_k$ Lower bound $\underline{h}_{\Phi(-\epsilon/\sigma)}(\theta)$ with confidence c
  return $\underline{a}$

(a) Original (b) Embedded Content (c) Gaussian Noise (d) Unrelated

Figure 5.2: Samples of the backdoor images used for watermarking.

5.6.2 Samples of watermark images

5.6.3 Proof of Corollary 1

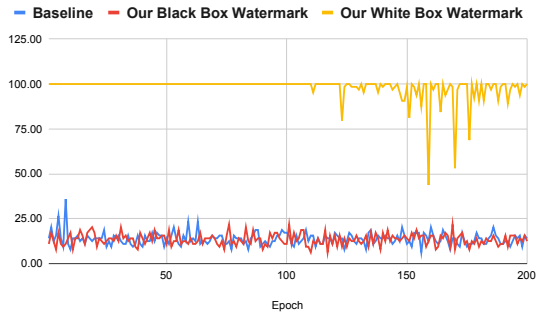
$$\underline{h}_{\underline{p}}(x) \leq h_p(x+\delta) \leq \overline{h}_{\overline{p}}(x) \qquad \forall \|\delta\|_2 < \epsilon \quad \text{Lemma 1 from [132]}$$

$$\Rightarrow \underline{h}_{\underline{p}}(x) \leq h_p(x+\delta) \qquad \forall \|\delta\|_2 < \epsilon$$

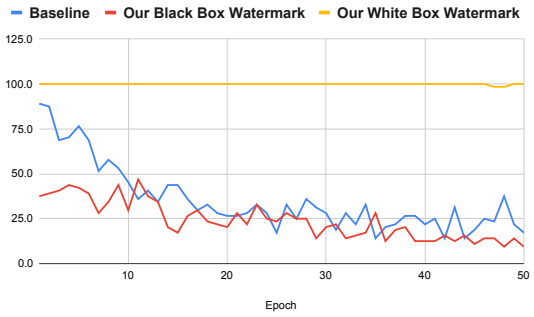
$$\Rightarrow \underline{h}_{\Phi\left(\Phi^{-1}(p)-\frac{\epsilon}{\sigma}\right)}(x) \leq h_p(x+\delta) \qquad \forall \|\delta\|_2 < \epsilon \quad \text{Definition of } \underline{p}$$

$$\Rightarrow \underline{h}_{\Phi\left(\Phi^{-1}(50\%)-\frac{\epsilon}{\sigma}\right)}(x) \leq h_{50\%}(x+\delta) \qquad \forall \|\delta\|_2 < \epsilon \quad \text{Plug in 50\% for } p$$

$$\Rightarrow \underline{h}_{\Phi\left(-\frac{\epsilon}{\sigma}\right)}(x) \leq h_{50\%}(x+\delta) \qquad \forall \|\delta\|_2 < \epsilon$$

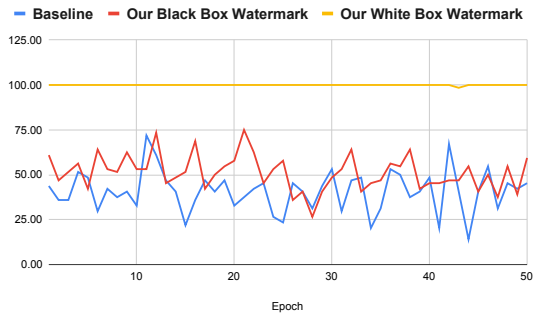


(a) lr=.001

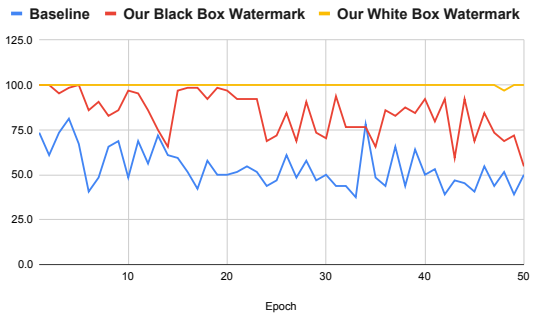


(b) lr=.0001

Figure 5.3: CIFAR-10 trigger set accuracy when faced with finetuning attacks



(a) finetuning attack with lr=.001



(b) finetuning attack with lr=.0001

Figure 5.4: MNIST trigger set accuracy when faced with finetuning attacks

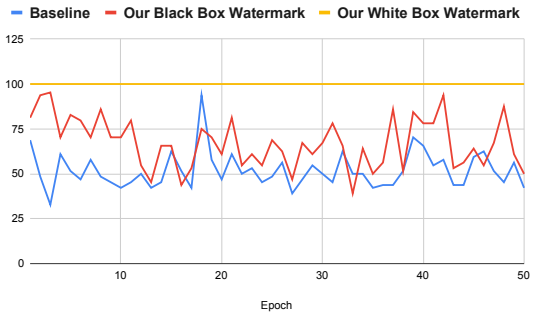
5.6.4 Trigger set trajectories during attack

5.6.5 Algorithm for empirical order statistic

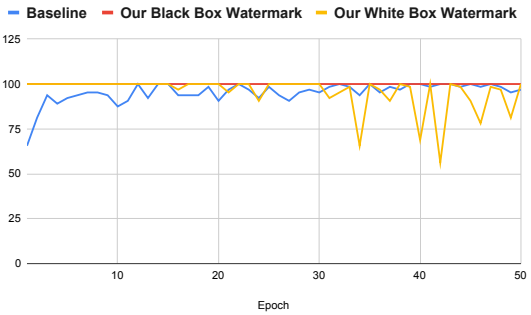
5.6.6 ℓ_2 norm change during attack

5.6.7 Additional Persistence Evaluation

In this section, we evaluated our watermark scheme with respect to 11 more attacks from [139]. We allow the adversary to have a time budget of 1 hour to remove the watermark as this is approximately the amount of time needed to train the model from scratch. With a budget any

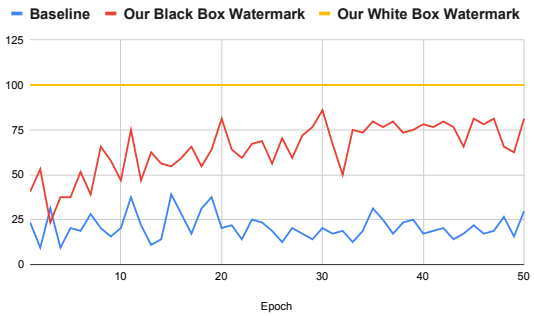


(a) hard-label distillation with lr=1e-3

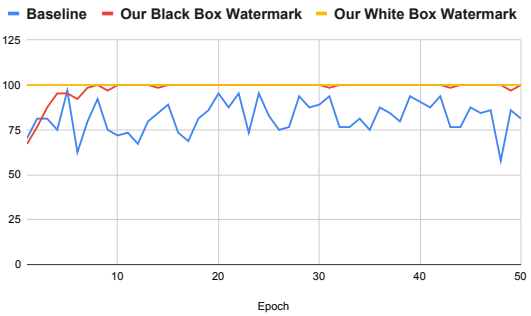


(b) soft-label distillation with lr=1e-3

Figure 5.5: MNIST trigger set accuracy when faced with distillation attacks



(a) hard-label distillation with lr=1e-3



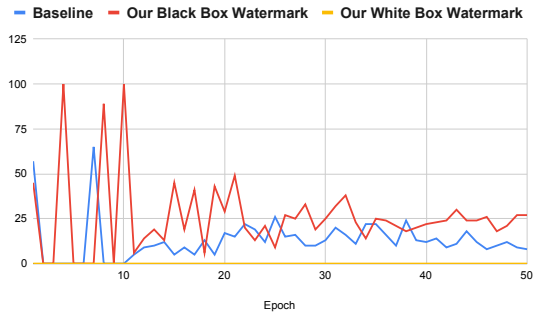
(b) soft-label distillation with lr=1e-3

Figure 5.6: CIFAR-10 trigger set accuracy when faced with distillation attacks

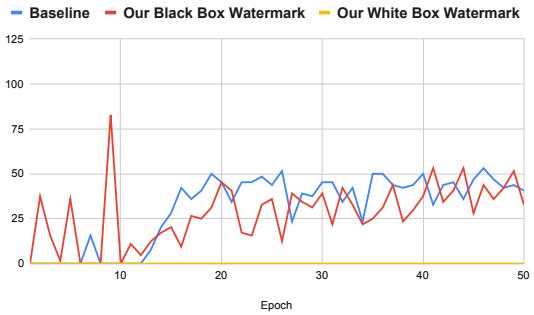
larger than 1 hour, the adversary will be better off training his/her own model.

We consider a watermark removed if the adversary obtains a model with higher than 82% test accuracy with less than 30% of watermark accuracy. We follow conventions from [139] where they consider an attack successful only if the accuracy of the model does not degrade by more than a certain amount and than the watermark accuracy remains above a decision threshold. We selected 82% following the convention in [139] where they consider an attack unsuccessful if the model's accuracy has been degraded by more than 5%. On the other hand, we chose 30% as a decision threshold as specified by [139] to err on the conservative side.

We used the Watermark-Robustness-Toolbox to conduct the additional persistence evaluation.



(a) CIFAR-10



(b) MNIST

Figure 5.7: Trigger set accuracy when faced with distillation+regularization attacks

Algorithm 4 Choosing the empirical order statistics that sufficiently lower bound the theoretical percentile

function EMPIRICALPERCENTILE(n, c, σ, ϵ)
  $p_{lower} \leftarrow \Phi(-\frac{\epsilon}{\sigma})$ ▷ calculate theoretical percentile that we should be lower bounding
  $\hat{\underline{K}}_{lower}, \hat{\overline{K}}_{lower} \leftarrow 0, \lfloor n \cdot p_{lower} \rfloor$ ▷ initialized empirical order statistics for lower bound
  while $\hat{\overline{K}}_{lower} - \hat{\underline{K}}_{lower} > 1$ **do**
    $\dot{K}_{lower} \leftarrow \lfloor (\hat{\overline{K}}_{lower} + \hat{\underline{K}}_{lower})/2 \rfloor$
    if 1-Binomial(n, $\dot{K}_{lower}$, p_{lower}) $> c$ **then**
      $\hat{\underline{K}}_{lower} \leftarrow \dot{K}_{lower}$
    else
      $\hat{\overline{K}}_{lower} \leftarrow \dot{K}_{lower}$
  if $\hat{\underline{K}}_{lower} > 0$ **then**
    return $\hat{\underline{K}}_{lower}$
  else
    return null

For each of the attack, as discussed in the paper the defender has half of the original training dataset while the attacker has the other half. For different attacks discussed in 5.7, we used the default hyper-parameters present in /configs/cifar10/attack_configs/ in which we simply changed the number of epochs to 100 in order to restrict the adversary within the time budget of approximately 1 hour.

Within the time constraint, all the methods tested fail to remove both the black-box watermark and white-box watermark simultaneously. Even though neural cleanse and neural laundering

Method	1st	2	3	4	5	6	7	8	9	10
CIFAR										
Hard label 10^{-4}	2.41	3.07	3.56	4.00	4.37	4.71	5.00	5.32	5.64	5.88
Hard label 10^{-3}	19.4	21.33	23.45	25.71	27.95	30.02	32.06	34.06	36.12	38.04
Soft label 10^{-4}	2.06	2.47	2.73	2.95	3.2	3.47	3.73	3.97	4.16	4.38
Soft label 10^{-3}	19.29	20.19	21.00	21.9	22.75	23.7	24.64	25.5	26.36	27.34
Finetune 10^{-4}	2.85	3.47	4.18	4.79	5.48	6.13	6.76	7.37	7.92	8.45
Finetune 10^{-3}	19.93	22.57	25.54	28.41	31.34	34.31	37.31	40.18	42.98	45.73
MNIST										
Hard label 10^{-4}	2.39	3.14	3.71	4.17	4.66	5.04	5.32	5.63	5.92	6.25
Hard label 10^{-3}	17.58	19.34	21.2	22.87	24.77	26.73	28.77	30.33	32.12	33.83
Soft label 10^{-4}	1.56	2.23	2.86	3.46	3.98	4.45	4.94	5.35	5.76	6.15
Soft label 10^{-3}	20.35	22.51	25.00	28.12	30.29	32.31	34.35	36.58	38.84	41.1
Finetune 10^{-4}	2.67	3.44	4.08	4.61	5.12	5.67	6.03	6.45	6.87	7.22
Finetune 10^{-3}	19.4	21.33	23.43	25.53	27.59	29.78	31.96	34.15	36.33	38.1

Table 5.6: Difference in ℓ_2 norm from previous parameters after each epoch of attack. After the first epoch, the increase is general small on each successive epoch.

are showing some effects at removing the watermark, these two methods did not successfully remove the watermark within the time limit. The two approaches also result in greater loss of test accuracy. The fine-tuning based approach (FTAL FTLL, RTAL, and RTLL) surprisingly increases the test accuracy. This is consistent with the results in [139] where the finetuning based approaches increase the test accuracy due to the use of additional training data.

Joint work with Arpit Bansal, Michael Curry, Rajiv Jain, Curtis Wigington, Varun Manjunatha, John Dickerson, Tom Goldstein first published in ICML 2022

	BB WM removal time	BB Acc	WB WM removal time	WB Acc	Accuracy Loss
FTAL	99	15	3600+	96	-4.01
FTLL	3600+	62	813	0	-2.24
RTAL	3600+	56	34	0	-1.22
RTLL	3600+	100	32	0	-0.33
Adversarial Training	530	28	3600+	100	3.72
Neural Cleanse	3600+	47	3600+	100	0.88
Neural Laundering	3600+	55	3600+	82	2.53
Weight Quantization	3600+	46	3600+	100	1.83
Feature Shuffling	0.97	100	-	100	0.05
Weight Pruning	3.27	100	-	100	0.05
Weight Shifting	3600+	52	3600+	100	2.49

Table 5.7: Evaluation against most attacks with new metrics

Chapter 6: Universal Pyramid Adversarial Training for Improved ViT Performance

6.1 Introduction

Human intelligence is exceptional at generalizing to previously unforeseen circumstances. While deep learning models have made great strides with respect to clean accuracy on a test set drawn from the same distribution as the training data, a model's performance often significantly degrades when confronted with distribution shifts that are qualitatively insignificant to a human. Most notably, deep learning models are still susceptible to adversarial examples (perturbations that are deliberately crafted to harm accuracy) and out-of-distribution samples (images that are corrupted or shifted to a different domain).

Adversarial training has recently been shown to be a promising avenue for improving both clean accuracy and robustness to distribution shifts. While adversarial training was historically used for enhancing adversarial robustness, recent works [1, 140] found that properly adapted adversarial training regimens could be used to achieve state-of-the-art results (at the time of publication) on Imagenet [140] and out-of-distribution robustness [1].

However, both proposed techniques [1, 140] use up to 7 times the standard training compute due to the sample-wise and multi-step procedure for generating adversarial samples. The expensive cost has prevented it from being incorporated into standard training pipelines and more widespread adoptions. In this chapter, we seek to improve the efficiency of the adversarial training technique

so that it can become more accessible for practitioners and researchers.

Several prior works [141, 142, 143, 144, 145] have proposed methods to increase the efficiency of adversarial training in the context of adversarial robustness, where they try to make models robust to deliberate malicious attacks. [141] proposed reusing the parameter gradient for training during the sample-wise adversarial step for faster convergence. Later, [145] proposed making adversarial training more efficient with a single-step adversary rather than the expensive multi-step adversary. However, all prior works focus on the efficiency trade-off concerning adversarial robustness rather than clean accuracy or out-of-distribution robustness. In the setting of adversarial robustness, one often assumes a deliberate and all-knowing adversary. Security is crucial, yet in reality, deep learning systems already exhibit a significant number of errors without adversaries, such as self-driving cars making mistakes in challenging environments. Consequently, clean accuracy and robustness to out-of-distribution data are typically prioritized in most industrial settings. Yet, few works seek to improve the efficiency trade-off for the out-of-distribution metric. Our work aims to fill this gap.

By shifting the context from adversarial robustness to clean accuracy and out-of-distribution robustness, we can free ourselves from certain constraints, such as the need to train the model on *sample-wise* adversaries, which is very expensive to compute. Instead, we can leverage *universal* perturbations, which are shared across the whole dataset. By leveraging this simple idea, we can generate adversarial samples for *free* while getting more performance improvement on clean accuracy compared to prior work [1].

In this chapter, we focus our experiments on the Vision Transformer architecture [146]. We focus on this architecture as it is the most general and scalable architecture that applies to many domains, including vision, language, and audio, while simultaneously achieving SOTA on many

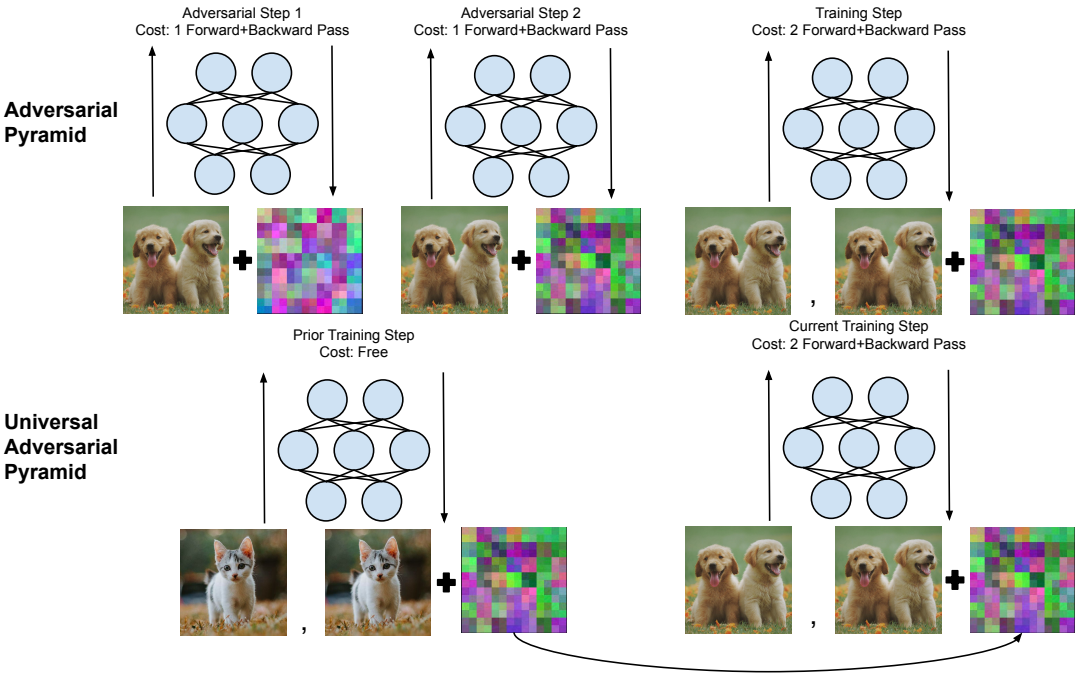


Figure 6.1: A graphical illustration of how our Universal Pyramid Adversarial training is more efficient compared to Pyramid Adversarial training. In the specific examples, the two-step Pyramid Adversarial training requires 4 forward and backward passes for a single example, where 2 passes are used for generating the adversarial sample and 2 additional passes are used for training. On the other hand, our proposed approach only requires 2 forward and backward passes. These two passes are needed because the batch size has been doubled since we optimize both the clean and the adversarial objectives.

of them. We believe focusing on this architecture will lead to more valuable techniques for the community.

In summary, here are our three main contributions:

- We propose Universal Pyramid Adversarial training that is 70% more efficient than the multi-step approach while increasing ViT's clean accuracy more than Pyramid Adversarial training.
- We evaluate our technique on 5 out-of-distribution datasets and find that Universal Pyramid Adversarial training effectively increases the distributional robustness and is competitive with Pyramid Adversarial Training while being efficient.

- To the best of our knowledge, we are the first to identify universal adversarial training as a viable technique for improving clean performance and out-of-distribution robustness on Imagenet 1K. In our ablations, , we found that the pyramid structure is critical for the performance gain and plain universal adversarial training is detrimental to performance, unlike [1], which found both instance wise adversarial training and pyramid adversarial training to be beneficial.

6.2 Related Work

Improving the efficiency of adversarial training has been widely studied [141, 142, 144, 145], but they have mainly been in the context of adversarial robustness. [141] proposed reusing the parameter gradients from the adversarial step. By reusing the free parameter gradients for training, they were able to achieve much faster convergence. Even though the proposed approach was much more efficient, [141] could not reach the same robustness level as the original multi-step training on Imagenet-1K. [142] proposed making the iterative attack cheaper by updating the noise based on the Hamiltonian functions of the first few layers. [145] proposed using a single-step adversary for training instead of a multi-step adversary. They found that random initialization and early stopping could prevent adversarial over-fitting, where label leakage happens from using adversaries with fewer steps. [144] proposed reusing the adversarial perturbations between epochs with the observation that adversarial noises are often transferable. The downside of the method is that the memory requirement grows with the data size, which can be quite large given the size of modern datasets. We differ from all the prior works in that we aim to investigate the efficiency gain of adversarial training in the context of clean accuracy. By focusing on

clean accuracy and out-of-distribution robustness, we gained more flexibility concerning the formulation of the min-max problem.

[140] was the first paper that showed adversarial training could improve clean performance of convolutional networks. To achieve this, [140] employed split batchnorms (AdvProp) for adversarial and clean samples. They argued that clean and adversarial samples have very different distributions and that split batchnorms are needed to make optimization easier. Before [140], the community commonly believed that adversarial training leads to a decrease in clean accuracy [147].

In a similar line of work, [143] proposed a faster variant of AdvProp [140] that makes the training speed comparable to standard training while retaining some of Advprop's benefits. Even though the proposed method is more efficient, it substantially trades off the performance of the original multi-step method. Our work is similar to [143] because we also focus on improving the efficiency of the adversarial training process, but we differ in that we focus on ViT architecture with Pyramid Adversarial training where their proposed approach is not applicable. Also, we can achieve a performance gain that is *comparable or better* than the multi-step approach, whereas the previous method trades off performance for efficiency.

Several recent approaches have shown that adversarial training can be used to improve the performance of vision transformers. [148] showed that ViT relies more on low-frequency signals than high-frequency signals. By adversarially training the model on high-frequency signals, [148] further boosted ViT's performance. [149] showed that by converting images to discrete tokens, adversarial training could further increase the performance of ViTs. Later, [1] showed that by incorporating the pyramid structure into standard adversarial training, they could boost the performance of ViT, where the split batchnorms idea introduced in [140] were not directly

applicable to ViT models. In our work, we focus on the Pyramid Adversarial training technique proposed by [1] since it is the best-performing method that achieves SOTA on multiple fronts while being applicable to ViT, a more modern architecture. The main drawback of [1] is the significantly higher training time which can go up to 7x of the standard training time. In this work, we propose Universal Pyramid Adversarial training to improve the efficiency of Pyramid Adversarial training while retaining its effectiveness.

While universal adversarial samples have been used in prior work [150] for training to defend against universal adversarial attacks, our proposed approach differs from them in that it leverages these samples to improve clean model performance. [151] finds that universal perturbations tend to slide images into some classes more than others. They find that by updating universal perturbations in a class-wise manner, they can achieve better robustness compared to [141]. Both prior works [150, 151] show that universal adversarial training consistently decreases the performance of the model, similar to standard adversarial training. Our ablation study shows that incorporating both clean loss and pyramid structures are crucial for the performance gain observed in Universal Pyramid Adversarial training. Without our proposed modifications, universal adversarial training consistently decreases the clean performance of the model. To the best of our knowledge, we are the first to show that universal adversarial training can be leveraged for improved model performance.

6.3 Method

In this section, we will go over the formulation of the proposed adversarial training objective, the pyramid structure that we leveraged from [1], and our more efficient Universal Pyramid

Adversarial training.

6.3.1 Adversarial Training

Adversarial training remains one of the most effective methods for defending against adversarial attacks [152]. Adversarial training is aimed at solving the following min-max optimization problem:

$$\min_{\theta} E_{(x,y)\sim D}\left[\max_{\delta\in B} L(f(x+\delta;\theta),y)\right], \tag{6.1}$$

where θ is the model parameter, δ is the adversarial perturbation, L is the loss function, D is the data distribution, and B is the constraint for the adversarial perturbation, which is often an ℓ_∞ ball. The inner objective seeks to find an adversarial perturbation within the constraint, and the outer objective aims to minimize the worst-case loss by optimizing the model parameters. While the method effectively improves robustness to adversarial attacks, it often reduces clean performance. However, the loss of clean performance is often not acceptable for most practical applications.

Since our goal is to improve performance as opposed to worst case robustness, we train the model on the following formulation (similar to [1, 140]) instead where the clean loss is optimized in addition to the adversarial loss:

$$\min_{\theta} \underset{(x,y)\sim D}{E}\left[L(f(x;\theta),y) + \lambda \max_{\delta\in B} L(f(x+\delta;\theta),y)\right]. \tag{6.2}$$

Here, the λ controls the trade-off between adversarial and clean loss. However, adding clean loss alone is often not sufficient for improving the performance of the model [140]. Additional techniques such as split batchnorms [140] and pyramid structures [1] are necessary for performance gain.

The main problem with the adversarial training formulation is that the inner maximization is often expensive to compute, requiring several steps to approximate accurately [147]. Specifically, for each iteration of the adversarial step, one needs a full forward and backward pass on all of the examples in a batch (see Figure 6.1). For example, if five steps are used, which is the setting in both [1, 140], then five forward and backward passes are needed. The generation of adversarial samples is already five times more expensive than regular training. In addition, one needs to use both the clean and the generated adversarial samples for training, so now we have doubled the batch size. The larger batch size increases the cost by another factor of two. When training with a 5-step adversary, the total computational cost will be seven times more expensive than standard training. In Section 6.3.3, we describe our proposed Universal Pyramid Adversarial training, where we can substantially reduce the computational cost.

6.3.2 Pyramid Structure

Adversarial training alone even when coupled with the clean loss does not typically increase performance of the model [140]. In order to increase clean accuracy, certain techniques have to be used. Here we leverage the pyramid structure from [1]. [1] aimed to endow the adversarial perturbation with more structure so that the adversary can make larger edits without changing an image's class. The pyramid adversarial noise is parameterized with different levels of scales as follows:

$$\delta = \sum_{s \in S} m_s \cdot C_B(\delta_s), \tag{6.3}$$

where C_B clips the noise within the constraint set B, S is all of the scales used, m_s is the multiplicative constant, and δ_s is perturbation at scale s. For the δ_s at a given scale, $s \times s$ number

of pixels within a square tile share a single parameter giving greater structure to the noise. Since the larger scale can often tolerate more changes, larger m_s at the coarser scales allow us to update the coarser noises more quickly relative to the granular noise.

6.3.3 Universal Pyramid Adversarial Training

While Pyramid Adversarial training [1] is effective at increasing clean model performance, it is seven times more expensive compared to standard training. To address this, we propose Universal Pyramid Adversarial training, an efficient adversarial training approach to improve model performance on clean and out-of-distribution data. Our proposed approach learns a universal adversarial perturbation with pyramid structure, thus unifying both the effectiveness of Pyramid Adversarial training and the efficiency of universal adversarial training [150]. Specifically, we attempt to solve the following objective:

$$\min_{\theta} \max_{\delta \in B} \underset{(x,y)\sim D}{E} \left[L(f(x;\theta), y) + \lambda L(f(x+\delta;\theta), y) \right]. \tag{6.4}$$

With this objective, we only have to solve for a *single* universal adversarial pattern that can be shared across the whole dataset, and we do not have to optimize a new adversary for each sample. Even though the objective looks similar, they are not the same. Due to Jensen's inequality, Equation 6.4 is always strictly upper-bounded by Equation 6.2. We have described the complete method in Algorithm 5. This yields up to 70% saving compared to the 5-step sample-wise approach (see Table 6.1). Further, we update the universal adversarial pattern during the backward pass of training, where we can get the gradients of δ for free (see Figure 6.1 for an illustration of how universal adversarial training can help save compute). However, since we still need to train the model on twice the number of samples, our proposal is still twice as expensive as standard

Algorithm 5 Universal Pyramid Adversarial Training

τ is the step size of adversarial attack
λ controls the regularization strength
Initialize δ_s as zeros
for epoch = 1 ... N_{ep} **do**
  for $x, y \in D$ **do**
    $\delta \leftarrow \sum_{s \in S} m_s C_B(\delta_s)$
    $\mathcal{L}(x, y, \delta; \theta) \leftarrow \ell(x, y; \theta) + \lambda \ell(x + \delta, y; \theta)$
    $\theta \leftarrow \theta - lr \cdot \nabla_\theta \mathcal{L}(x, y, \delta; \theta)$
    \\Update model parameters with some optimizer
    for $s \in S$ **do**
      $\delta_s \leftarrow \delta_s + \tau \cdot sign(\nabla_{\delta_s} \mathcal{L}(x, y, \delta; \theta))$
      \\Update noise with the free gradients

training, but it is already 33% cheaper than the fastest (one-step) sample-wise adversarial training approach.

More concretely, the generation step for the one-step sample-wise adversarial training costs a single forward-backward pass, and the training step is twice as expensive as standard training. Overall, one-step sample-wise adversarial training is 3x the cost of standard training making our method 33% faster. This is because in case of the one-step adversarial training, the gradient from the first generation step cannot be reused because the patterns are randomly initialized and the induced gradient is different from the clean training gradient.

6.3.4 Radius Schedule

In our experiments, we find that a radius schedule occasionally benefits performance. The radius dictates the extent to which an adversary is permitted to alter the image, as measured by the ℓ_∞ distance between the original and perturbed images. A larger radius permits greater perturbations, thus strengthening the adversary, while a smaller radius restricts the perturbation, rendering the adversary weaker. We propose this schedule as we observe that a more aggressive/larger

Augmentation	Method	Radius	Radius Schedule	# of Steps	Training Time (hrs)	Top 1 Accuracy	Gain
Weak	Baseline	-	-	-	12.7	72.90%	-
	Pyramid Adversarial Training[1]	6/255	-	1	36.5	73.52%	0.62%
		6/255	-	2	49.3	73.95%	1.05%
		6/255	-	3	61.8	74.68%	1.78%
		6/255	-	4	75.4	74.80%	1.90%
		6/255	-	5	88.9	74.18%	1.28%
	Universal Pyramid Adversarial Training	8/255	Yes	-	26.6	74.87%	**1.97%**
		8/255	-	-	26.6	74.57%	1.67%
Strong	Baseline	-	-	-	12.7	79.85%	-
	Pyramid Adversarial Training[1]	6/255	-	1	36.5	79.46%	-0.38%
		6/255	-	2	49.3	79.87%	0.03%
		6/255	-	3	61.8	80.19%	0.35%
		6/255	-	4	75.4	80.04%	0.22%
		6/255	-	5	88.9	80.10%	0.26%
	Universal Pyramid Adversarial Training	8/255	Yes	-	26.6	80.15%	0.31%
		8/255	-	-	26.6	80.28%	**0.44%**

Table 6.1: Comparison of the effectiveness of universal and sample-wise pyramid adversarial training. Universal Pyramid Adversarial training improves performance compared to sample-wise Pyramid Adversarial training while being much more efficient.

radius tends to promote faster convergence at the beginning, but these images are very far out of distribution, resulting in poor performance. By using a linearly decreasing radius schedule, we are sometimes able to get a considerable performance boost while maintaining fast convergence. Precisely, we calculate the radius at a given epoch as follows

$$r(e) = r_{\text{start}} + (r_{\text{end}} - r_{start})\frac{\max(e - e_{start}, 0)}{e_{end} - e_{start}}, \tag{6.5}$$

where r_{start}, r_{end} are the starting and ending radius with $r_{start} > r_{end}$, e_{start}, e_{end} are the starting and ending epochs for the radius schedule, and $r(e)$ is the radius at a given epoch e.

6.4 Experiments

6.4.1 Experimental Set-up

In all of our experiments, we focus on the training setup in [153] since it allows us to achieve a competitive 79.8% on Imagenet-1K with a ViT-S/16. The setup allows us to study ViT in a computationally feasible setting.

Following [153], we use the AdamW optimizer, with a batch size of 1024, a learning rate of 0.001 with a linear warm-up for the first 8 epochs, and weight decay of 0.1. We train the model for a total of 300 epochs across all settings. For augmentation, we apply a simple inception crop and horizontal flip. For experiments with strong data augmentation, we apply RandomAugment [154] of level 10 and MixUp [155] of probability 0.2. We used strong data augmentation in all of our experiments except for the first part of experiments in Table 6.1.

For Pyramid Adversarial training, following [1] we use $S = [32, 16, 1]$, $M = [20, 10, 1]$, and radius of 6/255. For step size, we simply divide the radius by the number of steps used. For our proposed Universal Pyramid Adversarial training, we use the same M and S as Pyramid Adversarial training, but with a radius of 8/255. When a radius schedule is used, we linearly decrease the radius by 90% in increments starting from epoch 30.

In addition to Imagenet-1K, we also evaluated our models on five out-of-distribution datasets: Imagenet-C [156], Imagenet-A [157], Imagenet-Rendition, Imagenet Sketch [158], and Stylized Imagenet [159]. Using a diverse set of out-of-distribution datasets, we can more thoroughly evaluate the model's robustness to unexpected distribution shifts.

Method	Radius	Training Time (hrs)	Clean ↑	C ↓	A ↑	Rendition ↑	Sketch ↑	Stylized ↑
Universal Pyramid Adversarial Training	8/255	26.6	80.28%	41.02%	29.08%	35.45%	16.89%	17.74%
Gain/Loss Relative to								
- Regular Training	-	12.7	0.44%	-1.16%	1.56%	1.80%	1.81%	0.68%
- Pyramid Adversarial Training 1 steps	6/255	36.5	0.82%	0.62%	2.60%	0.11%	0.33%	-1.26%
- Pyramid Adversarial Training 2 steps	6/255	49.3	0.41%	1.25%	0.59%	-0.71%	-0.06%	-2.21%
- Pyramid Adversarial Training 3 steps	6/255	61.8	0.09%	2.10%	-1.01%	-2.29%	-0.77%	-3.22%
- Pyramid Adversarial Training 4 steps	6/255	75.4	0.24%	1.80%	-1.25%	-2.95%	-1.13%	-3.28%
- Pyramid Adversarial Training 5 steps	6/255	88.9	0.18%	2.17%	-0.96%	-3.17%	-0.91%	-3.47%
Universal Pyramid Adversarial training	12/255	26.6	80.04%	40.37%	28.21%	37.05%	17.42%	19.53%
Gain/Loss Relative to								
- Regular Training	-	12.7	0.20%	-1.81%	0.69%	3.39%	2.34%	2.48%
- Pyramid Adversarial Training 1 steps	6/255	36.5	0.58%	-0.02%	1.73%	1.70%	0.87%	0.53%
- Pyramid Adversarial Training 2 steps	6/255	49.3	0.17%	0.60%	-0.28%	0.88%	0.48%	-0.41%
- Pyramid Adversarial Training 3 steps	6/255	61.8	-0.15%	1.46%	-1.88%	-0.70%	-0.23%	-1.42%
- Pyramid Adversarial Training 4 steps	6/255	75.4	0.00%	1.15%	-2.12%	-1.36%	-0.59%	-1.48%
- Pyramid Adversarial Training 5 steps	6/255	88.9	-0.06%	1.52%	-1.83%	-1.58%	-0.38%	-1.67%

Table 6.2: Evaluating models trained with Universal Pyramid Adversarial training (across two radius) on 5 additional out-of-distribution datasets and comparing them with regular and Pyramid Adversarial training. Universal Pyramid Adversarial training consistently boosts out-of-distribution robustness and is competitive with 1-step and 2-step Pyramid Adversarial training. We report the gain/loss relative to our method, i.e., the values in the colored cells are calculated by the following formula: (our method - the alternative method). For all columns except the Imagenet-C column, positive numbers mean that our method performs better and vice versa. We report mean Corruption Error (mCE) for Imagenet-C where lower is better. For the rest of the datasets, we simply report the top-1 accuracy. The complete table can be found in the Appendix.

6.4.2 Experimental Results

Overall, we find that Universal Pyramid Adversarial training effectively increases the clean accuracy and out-of-distributional robustness similar to the original Pyramid Adversarial training while being much more efficient.

Clean Accuracy Here we first analyze the effectiveness of our proposed Universal Pyramid Adversarial training when applied to ViT with weak data augmentation. Pyramid Adversarial training, as expected, significantly increases the performance of the ViT by up to 1.9% (Table 6.1) when a 4-step attack is used. As we increase the step count to 5, the benefit of Pyramid Adversarial training starts diminishing. On the other hand, our Universal Pyramid Adversarial training increased the performance even further. With the radius schedule, we can obtain a

performance increase of 1.97%, exceeding the performance benefit of the Pyramid Adversarial training for all step counts. Without the radius schedule, we still obtain a competitive gain of 1.67%. In addition to better performance, our method is much more efficient than the original Pyramid Adversarial training. In Table 6.1, we also reported the training time of each method on 8 Nvidia A100 GPUs in hours, and our approach is 70% faster compared to the 5-step Pyramid Adversarial training.

To further verify our proposal's effectiveness, we analyze our method's performance when coupled with strong data augmentations. The setting with strong data augmentation is a more challenging setting since all components of the training pipelines are heavily tuned. It is worth noting that our baseline ViT-S performs comparably with the baseline ViT-B/16 in [1, 160]. Again, we continue to see the benefit of Universal Pyramid Adversarial training compared to standard training.

In the more challenging setting, we see less benefit from both approaches, as expected. As we increase the number of steps for the Pyramid Adversarial training, the accuracy first increases, reaching the maximum at 3 steps, and starts decreasing with 4 or more steps. On the other hand, our Universal Pyramid Adversarial training achieves more performance gain than all of the step count tested while being more efficient.

Out-of-Distribution Robustness We found that Universal Pyramid Adversarial training effectively increases models' out-of-distribution robustness and is comparable to 1-step and 2-step Pyramid Adversarial training. In Table 6.2, we see that our Universal Pyramid Adversarial training consistently increases models' performance on all five out-of-distribution datasets relative to the baseline. When compared with Pyramid Adversarial training, we find that Universal Pyramid

	Radius 2/255	4/255	6/255	8/255	10/256	12/256
Baseline	79.85%					
Universal Adversarial Training (w/o Pyramid Structure)	79.73%	79.81%	79.64%	79.59%	79.65%	79.75%
- Gain Relative to Baseline	-0.12%	-0.04%	-0.21%	-0.26%	-0.20%	-0.10%
Universal Pyramid Adversarial Training	79.89%	80.23%	80.15%	80.28%	80.13%	80.04%
- Gain Relative to Baseline	0.04%	0.38%	0.30%	0.43%	0.28%	0.19%

Table 6.3: Comparing universal adversarial training with and without the pyramid structure. We find that the pyramid structure is indeed crucial for the performance gain observed. Without the pyramid structure, universal adversarial training consistently decreases the model's performance relative to the baseline.

Method	Radius	Top 1 Accuracy	Gain
Baseline	-	79.84%	-
Uni. Pyramid Adv.	2/255	79.89%	0.05%
	4/255	80.23%	0.39%
	6/255	80.15%	0.31%
	8/255	80.28%	**0.44%**
	10/255	80.13%	0.29%
	12/255	80.04%	0.20%

Table 6.4: Universal Pyramid Adversarial training consistently increases the performance of ViT across a range of hyperparameters but achieves the best performance at radius 8/255.

Adversarial training with a radius of 12/255 consistently improves performance with respect 1-step Pyramid Adversarial training and is comparable with the 2-step Pyramid Adversarial training. Note that both 1-step and 2-step Pyramid Adversarial training are already 50% and 100% more expensive than our proposed Universal Pyramid Adversarial training. However, unlike the case of clean accuracy, Universal Pyramid Adversarial training still underperforms relative to the more costly Pyramid Adversarial training with 3 or more steps.

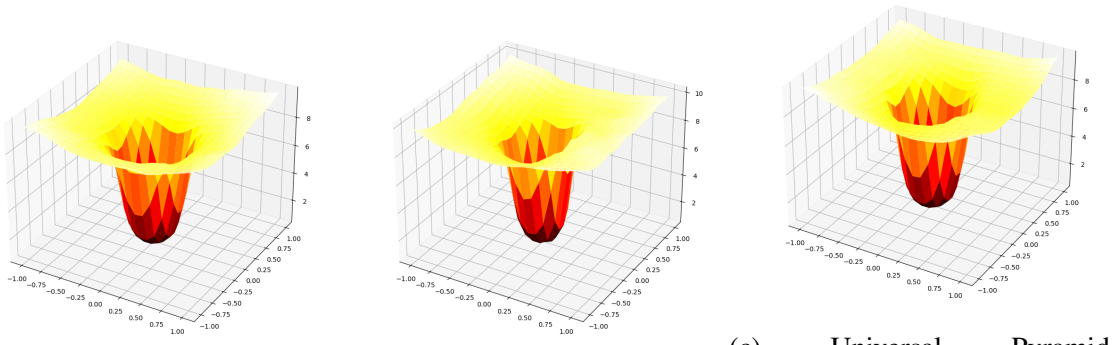
	Top 1 Accuracy
Baseline	79.84%
Uni. Pyramid Adv.	
w/o clean loss	78.08% (-1.76%)
w/ clean loss	80.28% (+0.44%)

Table 6.5: Comparing Universal Pyramid Adversarial training with and without adding the clean loss. We found that the addition of clean loss is critical for performance improvements.

6.4.3 Ablations

In this section, we ablated several components of Universal Pyramid Adversarial training, including its sensitivity to the selected radius, the importance of the pyramid structure, and the benefit of incorporating clean loss.

Sensitivity to Radius Hyperparameter sensitivity is crucial for a method's practicality, and we find that our Universal Pyramid Adversarial training is consistent and stable with respect to the selected radius. In Table 6.4, we see that Universal Pyramid Adversarial training consistently increases the model's performance across a wide range of radii from 2/255 to 12/255. This consistency is important because it allows us to benefit from the method without finely tuning the radius. We also find that the performance varies in a predictable upside-down U-shape. As we increase the radius, we see that the performance steadily increases until radius 8/255 after which the performance decreases. The way that performance changes with respect to the radius gives practitioners a clear signal on whether to increase or decrease the radius and makes hyperparameter tuning easier.

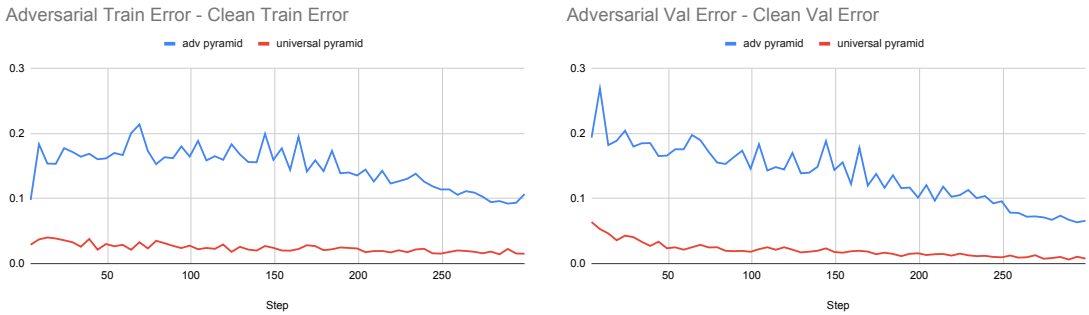


(a) Baseline (b) Pyramid Adversarial training (c) Universal Pyramid Adversarial training

Figure 6.2: Analyzing the loss landscape of the final trained models. We employed the filter normalization method from [161] for visualization. Pyramid Adversarial training seems to induce minima that are sharper compared to the baseline and Universal Pyramid Adversarial training. However, despite landing the model in sharper minima, Pyramid Adversarial training still produces a better performing model than the baseline. On the other hand, Universal Pyramid Adversarial training does not seem to change the sharpness of minima while attaining the best performance. This observation shows that both adversarial pyramid approaches do not improve performance through flattening the loss landscape.

Pyramid Structure We also found that the pyramid structure is crucial for performance gain with our proposed universal adversarial approach. We experimented with naively combining clean loss with universal adversarial training as in [150] as an additional regularizer. However, in Table 6.3, we see that without using the pyramid structure, the model consistently performs worse after adding the adversarial samples.

Clean Loss In addition to using pyramid structure, we found that incorporating clean loss to the Universal Pyramid Adversarial training is vital to obtain the performance gain we see. In Table 6.5, we see that removing the clean loss results in a performance decrease of 1.76%, making the model much worse than the baseline.



(a) Attack Strength on the Training Set (b) Attack Strength on the Validation Set

Figure 6.3: Comparing the strength of the universal adversary with the sample-wise adversary. The x-axis is the number of epochs, and the y-axis is the increased error rate after adversarial attack. We see that instance wise adversary is much stronger compared to universal adversary even though universal adversary achieves competitive performance on clean and ood tasks.

6.4.4 Analysis

In this section, we try to understand the similarities between universal and sample-wise adversarial pyramid training, given their similar benefits and formulation. We analyzed the attack strength, the noise pattern, and the loss landscape and found that despite their similarities, they are quite different in many aspects. The findings point to the need to further understand the mechanism that both universal and sample-wise adversarial pyramid training use to increase model performance.

Attack Strength Given that the Universal Pyramid Adversarial training achieves similar performance gain compared to the Pyramid Adversarial training, one may expect that their attack strength is similar. However, we found that the sample-wise adversary is significantly stronger than the universal adversary even though the universal adversary uses a larger radius (8/255 vs. 6/255). In Figure 6.3, the multi-step adversary consistently achieves a much higher adversarial error rate throughout the training process. The observation shows that we don't necessarily need to make

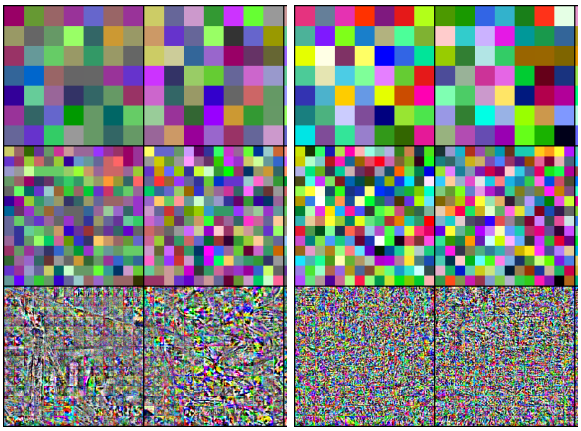
the attack very strong to benefit from adversarial training.

Qualitative Differences in Perturbation Pattern In Figure 6.4, we visualize the universal and sample-wise adversarial patterns used during training. We find the perturbations to have qualitatively different patterns despite their similar effectiveness in improving clean accuracy.

For the perturbation at the coarser scales, universal perturbation has a more diverse color than sample-wise perturbation. The diversity may be because universal perturbations need to transfer between images, and large brightness changes may be effective in removing some information from the samples. On the other hand, the sample-level perturbations may exploit the color cues to move an image to the adversarial class by consistently changing the color of the image.

For the perturbation at the pixel level, sample-wise perturbation is much more salient compared to universal perturbation. We can see that the sample-wise perturbations have some resemblance to objects. Even though the universal perturbations have some salient patterns, they are less obvious than the pattern from the sample-wise perturbations.

Loss Landscape To understand how Pyramid Adversarial training and Universal Pyramid Adversarial training improve the performance of a model, we visualize the loss landscape of models trained with both approaches to see whether it achieves the performance by implicitly inducing a flatter minimum [161]. Surprisingly, we found this not to be the case. Sample-wise Pyramid Adversarial training produces sharper minima compared to regular training and yet has better performance (see Figure 6.2). On the other hand, Universal Pyramid Adversarial training does not noticeably change the sharpness of the minimum and yet produces the greatest performance improvement. This finding suggests that both adversarial pyramid approaches rely on different underlying



(a) (b)

Figure 6.4: Adversarial patterns generated by sample-wise pyramid adversarial training in 6.4a and universal pyramid adversarial training in 6.4b. We plotted each level of the pyramid separately, so that we can visually inspect the differences between each level. We found that (1) universal pyramid seems to rely on more diverse perturbation values at the coarser scale and (2) sample-wise pyramid adversarial seems to rely more on the pixel level perturbations according to the more salient pixel level perturbations.

mechanisms to improve the model's performance compared to an optimizer, such as SAM [162] that explicitly searches for flatter minima.

6.5 Chapter Summary

In this chapter, we propose Universal Pyramid Adversarial training to improve the clean performance and out-of-distribution robustness of ViT. It obtains similar accuracy gain as sample-wise Pyramid Adversarial training while being up to 70% faster than the original approach. To the best of our knowledge, we are also the first to identify universal adversarial training as a possible technique for improving the model's clean accuracy. We hope that the proposed method will help make the adversarial technique more accessible to practitioners and future researchers.

6.6 Chapter Appendix

6.6.1 Further Details for Table 2

Method	Radius	Clean ↑	C ↓	A ↑	Rendition ↑	Sketch ↑	Stylized ↑
Universal Pyramid Adversarial Training	12/255	80.04%	40.37%	28.21%	37.05%	17.42%	19.53%
Universal Pyramid Adversarial Training	8/255	80.28%	41.02%	29.08%	35.45%	16.89%	17.74%
Regular Training		79.84%	42.18%	27.52%	33.65%	15.08%	17.06%
Pyramid Adversarial Training 1 steps	6/255	79.46%	40.39%	26.48%	35.34%	16.56%	19.00%
Pyramid Adversarial Training 2 steps	6/255	79.87%	39.76%	28.49%	36.17%	16.95%	19.95%
Pyramid Adversarial Training 3 steps	6/255	80.19%	38.91%	30.09%	37.75%	17.66%	20.96%
Pyramid Adversarial Training 4 steps	6/255	80.04%	39.22%	30.33%	38.40%	18.02%	21.02%
Pyramid Adversarial Training 5 steps	6/255	80.10%	38.84%	30.04%	38.63%	17.80%	21.21%

Table 6.6: Here we provide further details for the performance of each of the adversarial training methods on different corruption datasets.

6.6.2 Visualization of Attention Operations

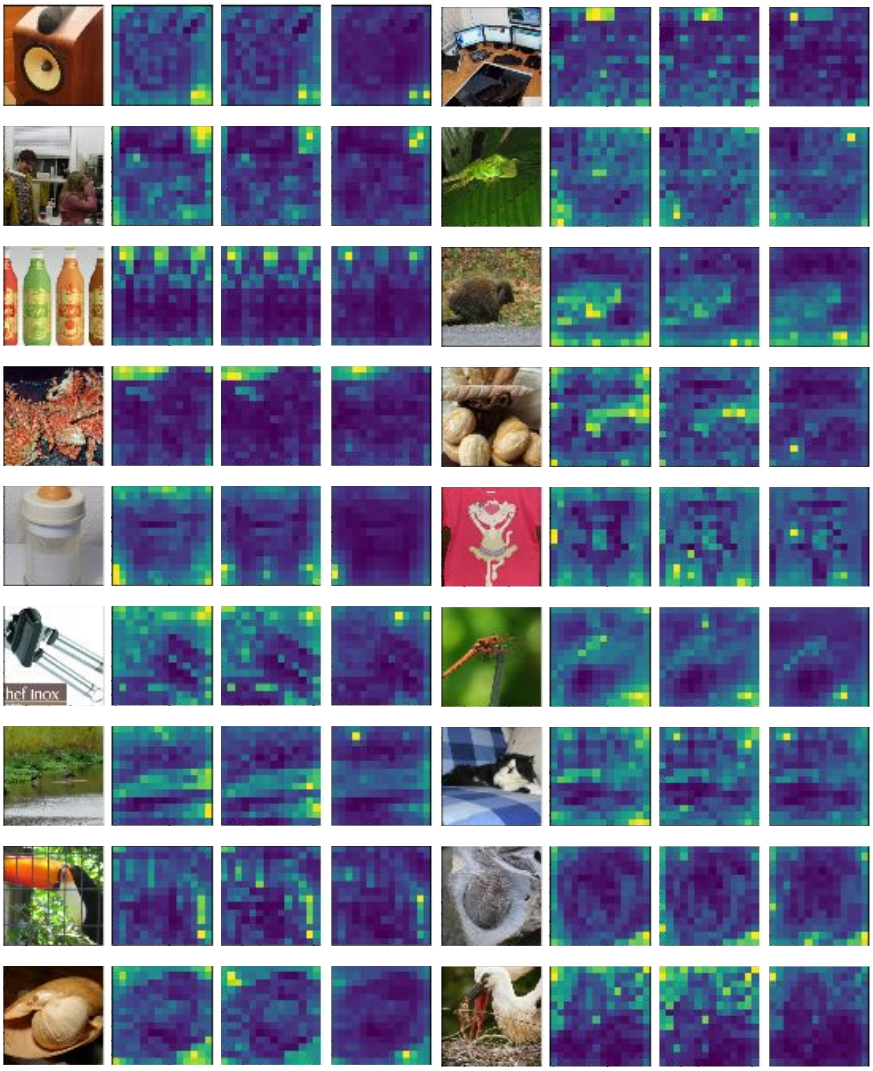


Figure 6.5: The first column displays the image, followed by attention visualizations for the baseline model, UPAT model, and PAT model in the next three columns. Despite exhibiting similar attention patterns to the baseline model, the UPAT model consistently outperforms it. In contrast, the PAT model demonstrates sparse attention, indicating that UPAT and PAT models improve performance through different mechanisms.

6.6.3 Radius Ablation with Respect to Imagenet V2

Method	Radius	Top 1 Accuracy (Imagenet v2)	Gain
Baseline	-	79.81%	-
Uni. Pyramid Adv.	2/255	79.93%	0.12%
	4/255	79.91%	0.10%
	6/255	79.94%	0.13%
	8/255	80.24%	**0.43%**
	10/255	79.91%	0.10%
	12/255	80.20%	0.39%

Table 6.7: During training, we lacked a separate validation set, raising concerns that our model's hyperparameters may overfit to the test accuracy. To mitigate this, we employed additional evaluation using ImageNet-V2, an independent dataset not used for hyperparameter selection in our experiments. Ultimately, we observed similar evaluation results on ImageNet-V2 and the test set, alleviating concerns of hyperparameter overfitting.

Joint work with Yipin Zhou, Omid Poursaeed, Satya Narayan Shukla, Ashish Shah, Tom Goldstein, Ser-Nam Lim

Chapter 7: Loss Landscapes Are All You Need: Neural Network Generalization Can Be Explained Without the Implicit Bias of Gradient Descent

7.1 Introduction

The impressive generalization of deep neural networks continues to defy prior wisdom, where overparameterization relative to the number of data points is thought to hurt model performance. From the perspective of classical learning theory, using measures such as Rademacher complexity and VC dimension, as one increases the complexity of a model class, the generalization performance of learned models should eventually deteriorate. However, in the case of deep learning models, we observe the exact opposite phenomenon – as one increases the number of model parameters, the performance continues to improve. This is particularly surprising since deep neural networks were shown to easily fit random labels in the overparameterized regime [163]. This combination of empirical and theoretical pointers shows a large gap in our understanding of deep learning, which has sparked significant interest in studying various forms of implicit bias which could explain generalization phenomena.

Perhaps the most widely-held hypothesis posits that gradient-based optimization gives rise to implicit bias in the final learned parameters, leading to better generalization [164, 165, 166, 167]. For example, [164] showed that deep matrix factorization, which can be viewed as a highly

simplified neural network, is biased towards solutions with low rank when trained with gradient flow. Indeed, [167] shows theoretically and empirically that stochastic gradient descent (SGD) with a small batch size can implicitly bias neural networks towards matrices of low rank. A related concept was used by [166] to show that gradient agreement between examples is indicative of generalization in the learned model.

In this chapter, we empirically examine the hypothesis that gradient dynamics is a necessary source of implicit bias for neural networks. Our investigation is based on a comparison of several zeroth order optimizers, which require no gradient computations, with the performance of SGD. We focus our studies on the small sample regime where zeroth order optimizations are tractable. Interestingly, we find that all the gradient-free optimizers we try generalize well compared to SGD in a variety of settings, including MNIST [168], CIFAR-10 [169], and few-shot problems [170, 171].

Even though we use fewer samples in our experiments compared to standard settings, this low-data regime highlights the role of model bias, where the generalization behavior of neural networks is particularly intriguing. The model we test has more than $10,000$ parameters, but it has to generalize with fewer than $1,000$ training samples. Without implicit bias, such a feat is nearly impossible in realistic use cases like the ones we consider. Our work shows empirically that generalization does not require the implicit regularization of gradient dynamics, at least in the low-data regime. It is still an open question whether gradient dynamics play a larger role in other regimes, namely, where more data is available.

We need to caution that we are not claiming that gradient dynamics have no effect on generalization, as it has been clearly shown both theoretically and empirically that it has a regularizing effect [164, 167]. Instead, we argue that the implicit regularization of gradient

dynamics is only secondary to the observed generalization performance of neural networks, at least in the low-data regimes we study.

The observations in this chapter support the idea that implicit bias can come from properties of the loss landscape rather than the optimizer. In particular, they support the *volume hypothesis* for generalization: The implicit bias of neural networks may arise from the volume disparity of different basins in the loss landscape, with good hypothesis classes occupying larger volumes. The conjecture is empirically supported by the observation that even a "guess & check" algorithm, which randomly samples solutions from parameter space until one is found with low training error, can generalize well. The success of this optimizer strongly suggests that generalizing minima occupy a much larger volume than poorly generalizing minima in neural loss functions, and that this volume disparity alone is enough to explain generalization in the low-shot regime.

Finally, we show in a previously studied toy example that volume implicitly biases the learned function towards good minima, regardless of the choice of optimizer.

7.2 Related Work

The capability of highly overparametrized neural networks to generalize remains a puzzling topic of theoretical investigations. Despite their high model complexity and lack of strong regularization, neural networks do not overfit to badly generalizing solutions. From a classical perspective, this is surprising. Bad global solutions do exist [163, 172], yet usual training routines which optimize neural networks with stochastic gradient descent never find such worst-case solutions. This has led a flurry of work re-characterizing and investigating the source of the generalization ability of neural networks. In the following we highlight a few angles.

High-dimensional optimization Before reviewing the literature on gradient dynamics, we want to review the underlying reasons why gradient-based (first-order) optimization is so central to deep neural networks: The core reasons for this is often dubbed the *curse of dimensionality*: For arbitrary optimization problems (with minimal conditions, i.e. [173]) a first-order optimizer will converge to a local minimal solution in polynomial time in the worst-case, independent of the dimensionality of the problem. However, a zeroth order algorithm without gradient information will have to, in the worst-case, evaluate a number of queries that increases exponentially with the dimensionality of the problem, even for smooth, convex optimization problems [174]. However as we will discuss, neural networks are far from a worst-case scenario, given that many solutions exist due to the flatness of basins and the inter-connectedness of minima in neural networks.

Gradient dynamics Here we briefly review literature that argues for gradient descent as the main implicit bias for generalization of neural networks. In Liu et al. [166], they argue that deep networks generalize well because of the large agreement of gradients among training examples using a quantity called gradient signal-to-noise ratio (GSNR). They found both empirically and theoretically that a large GSNR would lead to better generalization and that deep networks induce a large GSNR during training, leading to better generalization. Arora et al. [164] show that the dynamics of gradient-based optimization induce implicit bias that is stronger than typical norm-based bias in the setting of deep matrix factorization, and raise the question whether implicit biases can be induced from first-order optimization that cannot be captured by any explicit regularization. Advani et al. [165] argues that in the overparameterized regime, the gradient dynamics prevent learning from happening in a certain subspace of the weights, which effectively works as implicit regularization. A recent paper by [167] proves that SGD trained networks have

a low-rank implicit bias and hypothesizes that such an implicit bias may be the source of superior generalization for deep neural networks.

Non-gradient based explanation of implicit bias Several works have tried to explain the generalization behavior of neural networks with other forms of implicit regularization. Neyshabur et al. [175] argues weight norms to be the main measure of capacity control that allows neural networks to generalize. Keskar et al. [176] suggests that flatness in the parameter space corresponds to simpler functions, thus allowing neural networks to generalize. However, Dinh et al. [177] later show that when the flatness measure is not scale-invariant, sharp solutions can generalize just as well with appropriate rescaling of the network parameters. Valle-Perez et al. [178] argue that the parameter-function map is exponentially biased towards simple functions. Rahaman et al. [179] shows that neural networks are biased toward low frequency functions that vary globally without local fluctuation. Among all works that try to explain neural network generalization, most recent works argue gradient descent or stochastic gradient descent as the main implicit bias of neural network training that allows deep overparameterized networks to generalize.

Volume and Bayesian modeling From a Bayesian perspective, flat minima of the loss surface are highly represented in the Bayesian model average, especially when they contain functional diversity [180]. The size of a posterior peak has also been connected to Occam factors indicating that they represent simpler solutions which are more compressible and generalize well [181]. Smith and Le [182] studies generalization behavior of overparameterized linear models where they find that the Bayesian evidence or marginal likelihood, which is connected to generalization, strongly favors flat minima. A line of work on PAC-Bayes generalization bounds, which is related to compressibility and the Bayesian evidence, uses compressibility to guarantee generalization

and finds the flat minima are more compressible as they yield more bits back from the KL-divergence term in the bound [183]. In contrast to these works, our findings focus not on why flat minima generalize well but rather how their large volume makes them likely to be found by optimizers.

Similar Lines of Inquiry Mingard et al. [184] empirically show that when sampling from wide networks conditioned on the training set, the sampled models behave similarly to finite width networks trained with SGD. They approximate the posterior with Neural Network Gaussian Process, which is not exact in finite width networks. Geiping et al. [185] show that full batch gradient descent, when coupled with explicit regularization can perform comparably to model trained with SGD, thus bringing into question the importance of SGD for generalization. Similar in spirit, we argue that SGD and all gradient-based optimizers are not the main source of generalization behavior of neural networks. [172] provide intuitive explanations for the volume hypothesis, and empirically measure the volume of both good and bad minima. While they show that individual good minima tend to have much larger volume than individual bad ones, their experiments do not show that the total volume of all good minima is large. Experiments below address this weakness.

7.3 The Mystery of Generalization with Overparameterization

In this section, we illustrate how the complexity of a hypothesis class increases with the number of parameters in the context of a simple classification problem. Specifically, we increase the number of hidden units of a two layer neural network and showcase the increasing complexity of the model class. Despite this increased complexity, SGD often consistently finds good classifiers. Then, we proceed to show that a similar generalization behavior can be achieved without any

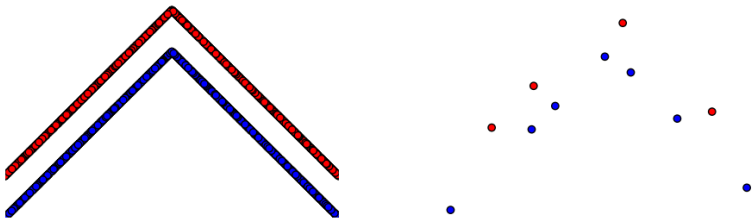


Figure 7.1: On the left, we have the true underlying distribution of the toy problem. On the left, we have the sampled training data.

gradient dynamics.

We begin with a toy classification problem defined over two classes where the data distribution is a wedge "Λ" with a vertical margin separating the two classes (see Figure 7.1). Throughout this section, our training and testing data consist of 11 points and 5 points, respectively, each sampled uniformly at random.

7.3.1 Overparameterization increases model complexity

To illustrate how the model complexity increases with number of parameters, we first poison a model by minimizing the loss on the training data while maximizing the loss on the testing data. Given we only examine cases where the model (trained by SGD) achieves 100% training accuracy, this represents the worst-case decision boundary for the unpoisoned loss. As we increase the number of hidden units from 2 to 20, the decision boundary becomes much more ill-behaved (see Figure 7.2a). When the hypothesis class is restricted to 2 hidden units (the left most plots in Figure 7.2a), the model can only fit the data by using a single kink in the decision

boundary, so it has to trade off either fitting the training examples to 100% accuracy or performing badly on the poison objective. Given that the model fits training data, it has to perform well on testing data. This is consistent with the under-parameterized regime and with classical learning theory.

As we increase the model size, the model class now contains strange decision boundaries that can fit the training data while performing poorly on the testing data. From the perspective of classical learning theory, we would expect models with 20 hidden units to perform much worse than the model with 2 hidden units. Surprisingly, even though more complicated decision boundaries are available as we increase the model size, we never see such boundaries when optimizing the (unpoisoned) training objective with SGD. For example, in Figure 7.2b, as we increase the number of hidden units, the decision boundary remains relatively consistent. Given that both the weird and nice decision boundaries exist in the model class, it is natural to ask what biases the learned network towards good vs. bad optima.

Due to the consistent behavior of SGD trained networks in the overparameterized regime, it is only reasonable that people started investigating gradient dynamics as a source of implicit regularization [164, 165, 166, 167]. However, we show below that, rather surprisingly, we can obtain similar generalization behavior on the toy problem by using a Guess & Check algorithm that is completely free of gradient dynamics.

7.3.2 Generalizing on toy problem without gradients

In our toy setting, we find that generalization is surprisingly generic with respect to the dynamics of optimizers. To avoid using optimizers with the same inductive biases as gradient

methods, we experiment with *Guess & Check*: we repeatedly sample parameters until a model achieves 100% training accuracy with train loss below a certain threshold. Unlike other optimizers, we do not use any gradient information, and we also do not take any iterative steps. Surprisingly, even with Guess & Check, we often end up with a well-behaving decision boundary like the one that we trained with regular SGD, see Figure 7.2c.

From the simple two class toy problem, we can see clearly that Guess & Check solutions already endow the learned model with a very strong implicit bias that does not originate from gradient dynamics. In the next section, we extend a similar analysis to common datasets such as MNIST & CIFAR10 to see whether this observation continues to hold in more practical settings.

7.4 Experiments

7.4.1 Non-gradient Based Optimizers

In our experiments, we test three different non-gradient based optimizers: Guess & Check, Pattern Search, and Greedy Random Search on varying scales of MNIST & CIFAR-10 and on different architectures. Here, we explain each optimizer.

7.4.1.1 Guess & Check

The Guess & Check algorithm optimizer randomly generates parameter vectors with entries sampled independently and uniformly from $[-1, 1]$[1]. If the randomly sampled model achieves 100% training accuracy and has training loss below a chosen threshold, then the model is kept and the optimizer terminates. If not, the vector is thrown away and we keep guessing new vectors

[1] See Appendix C for experiments with other sampling intervals.

until our conditions are met.

Guess & Check is of theoretical value because its only implicit bias comes from the geometry of the loss landscape, and its success implies the volume hypothesis. With this optimizer, the likelihood that a set of solutions are selected is exactly proportional to the volume of the set in parameter space. If a model consistently generalizes well when trained with Guess & Check, then this means the set of "good" minima has large volume among low-loss parameters. We do want to make a distinction between flat solutions [176] and solutions with large volumes. It is possible that a collection of solutions has a very large volume but is not itself a flat basin but rather a collection of many small volume regions that have large volume in aggregate.

When we train with Guess & Check, we can be confident that gradient-based implicit regularization plays no role in the final performance – the volume hypothesis is the only source of implicit regularization. Unfortunately, naïve guess-and-check suffers from the problem that the cost of interpolating the training data grows exponentially as the number of training examples or classes increase, so we have restricted experiments with Guess & Check to few-shot problems with smaller sample sizes.

7.4.1.2 Local Non-gradient Based Optimizer

Due to the difficulty of scaling Guess & Check to large problems, we explore two alternative non-gradient based optimizers, Pattern Search and Random Greedy Search, that work for bigger datasets. Like SGD, both approaches update the model using a local search and may have biases that originate from factors other than volume alone. The success of Pattern Search and Random Greedy demonstrates that gradient optimization is not *strictly* needed to observe implicit

regularization, but they may exploit regularization properties of local search that are also exploited by SGD.

Pattern Search Pattern Search randomly selects a parameter in the model and takes a step of fixed size that is randomly chosen to be either positive or negative, if the model achieves lower loss after taking the step, then the parameter is accepted as the new starting point. If Pattern Search fails to find a step that decreases the loss after going through all the parameters, then the step size is decreased by a constant factor. We repeat this procedure until a solution is found that achieves 100% training accuracy. In our experiments, we use a starting radius of 1, and we decrease the radius by a factor of 2 when it fails to find a descent direction.

Random Greedy Search Random Greedy Search adds Gaussian noise to the initial parameter vector with standard deviation of σ. If the noised solution improves training loss, then the noised solution is accepted as a new starting point. If no solution is found after a fixed number of steps, then σ is decreased by a chosen factor before the search continues. Again, we repeat this procedure until a parameter is found that achieves 100% training accuracy. In our experiment, we start the procedure with $\sigma = 1$. If we fail to find a perturbation that decreases loss after 30000 random steps, then we decrease σ by a factor of 2.

7.4.2 Results on 2-Class CIFAR-10/MNIST

In this section, we apply the Guess & Check algorithm on a conventional LeNet model on MNIST [168] and CIFAR-10 [169]. Due to the exponential time complexity of the Guess & Check algorithm, we stick with 2-class problems with fewer than 32 total training samples. To

MNIST # Samples	Val. Acc.	CIFAR # Samples	Val. Acc.
32	0%	24	0%
16	0%	16	0%
8	0%	8	0%
4	0%	4	0%
2	0%	2	0%

Table 7.1: Comparing poisoned validation error. In this table, we attempt to fit the training data of various sizes while poisoning LeNet with the wrong validation labels. We find that the LeNet we use is of sufficient capacity that it can completely fit the training data while failing to classify the validation set.

Table 7.2: On the two class MNIST problem, G&C performs comparably to SGD across different train loss level and number of samples. This shows us that despite the large number of parameters, G&C solutions are implicitly regularized. To show that the degree of generalization of G&C is indeed substantial, we train an additional linear model, which has a much more restricted hypothesis class, but has on average 10% worse generalization performance. The empty cells correspond to linear models where we could not find solutions with 100% training accuracy. We also show the estimated standard deviations of the averages computed over 175 random data split and training seeds. For most cells, the standard deviation is less than 1%.

Sample Count	**Arch**	**Optimizer**	**Best Test Acc**	**Train Loss** (0.3, 0.35)	(0.35, 0.4)	(0.4, 0.45)	(0.45, 0.5)	(0.5, 0.55)	(0.55, 0.6)	(0.6, 0.65)
32	LeNet	G&C	93.02%±0.27%	93.02%±0.27%	92.39%±0.29%	90.59%±0.34%	89.18%±0.38%	87.22%±0.43%	86.23%±0.44%	83.15%±0.51%
	LeNet	SGD	**94.04%±0.25%**	-	-	94.04%±0.25%	93.49%±0.28%	92.54%±0.28%	91.63%±0.33%	88.60%±0.35%
	Linear	SGD	84.75%±0.47%	84.75%±0.47%	82.69%±0.43%	81.24%±0.44%	79.04%±3.14%	78.94%±4.74%	-	-
16	LeNet	G&C	89.21%±0.47%	89.21%±0.47%	87.01%±0.50%	85.18%±0.56%	84.69%±0.54%	81.91%±0.62%	78.61%±0.65%	75.37%±0.63%
	LeNet	SGD	**91.24%±0.40%**	91.24%±0.40%	90.87%±0.41%	90.84%±0.38%	88.77%±0.48%	87.93%±0.48%	86.98%±0.47%	83.90%±0.49%
	Linear	SGD	80.68%±0.55%	80.68%±0.55%	78.50%±0.56%	75.69%±0.60%	72.09%±0.56%	67.16%±0.67%	69.51%±3.40%	-
8	LeNet	G&C	83.05%±0.67%	83.05%±0.67%	80.72%±0.75%	78.23%±0.81%	78.05%±0.72%	76.40%±0.79%	70.76%±0.74%	67.48%±0.78%
	LeNet	SGD	**84.82%±0.63%**	83.63%±0.63%	84.82%±0.63%	82.62%±0.74%	81.85%±0.72%	79.70%±0.70%	79.74%±0.63%	76.51%±0.71%
	Linear	SGD	74.29%±0.72%	74.29%±0.72%	71.72%±0.75%	67.79%±0.69%	67.36%±0.76%	63.46%±0.75%	58.65%±0.79%	54.87%±0.75%
4	LeNet	G&C	76.28%±0.90%	76.28%±0.90%	73.93%±0.92%	72.63%±0.86%	70.89%±0.90%	68.27%±0.83%	65.63%±0.92%	62.38%±0.91%
	LeNet	SGD	**77.35%±0.81%**	77.35%±0.81%	75.01%±0.85%	75.61%±0.83%	73.95%±0.85%	73.28%±0.85%	69.15%±0.84%	67.65%±0.84%
	Linear	SGD	65.12%±0.81%	65.12%±0.81%	61.94%±0.82%	62.14%±0.78%	58.11%±0.88%	57.21%±0.91%	55.38%±0.88%	53.60%±0.83%
2	LeNet	G&C	66.89%±1.04%	66.89%±1.04%	65.87%±1.05%	64.03%±0.92%	62.81%±0.90%	61.02%±0.84%	59.90%±0.91%	56.82%±0.95%
	LeNet	SGD	**69.67%±0.98%**	69.67%±0.98%	67.11%±0.93%	64.94%±0.95%	63.42%±0.87%	64.38%±0.88%	63.82%±0.89%	62.33%±0.87%
	Linear	SGD	58.93%±0.94%	58.93%±0.94%	58.45%±0.92%	56.59%±0.89%	54.11%±0.91%	54.21%±0.87%	53.13%±0.93%	51.59%±0.89%

enable fair comparisons between G&C and SGD optimized models, we compare the performance of the models across different train loss levels after the model's weights have been normalized. This is crucial for a fair comparison because it has been observed that lower loss levels corresponds to better generalization even after train accuracy has reached 100%.

We find that given the same loss level and number of samples, Guess & Check performs comparably to SGD, especially at lower loss levels. In the case of CIFAR-10, Guess & Check even outperforms SGD solutions by a substantial margin. This result is made even more interesting given that the models are capable of pathological overfitting: they are able to completely misclassify the validation set while achieving 100% training accuracy (see Table 7.1).

To illustrate how well G&C performs, we also train a linear model on MNIST for comparison. Even though the hypothesis class is now restricted to only linear solutions, a significantly smaller hypothesis class compared to LeNet, the linear model still underperforms the Guess & Check solution on LeNet by more than 10% in many cases. Note that, despite being convex, solutions of the linear problem vary because we use SGD and apply early stopping when the desired loss level is achieved.

Even though the number of samples is small, we do note that this regime highlights the effects of overparametrization. For example, in our LeNet for MNIST, we have 11074 parameters, which is orders of magnitude larger than the number of examples. Yet the model continues to generalize well relative to SGD, showing us that the large volume of the good solution set is on its own enough to bias the optimizer towards favorable generalization.

Even though the generalization performance is similar between the SGD and G&C solutions, we do note that the test accuracies are not exactly the same between models trained with both methods, implying that SGD may have additional bias that G&C does not take into account.

Table 7.3: On the two class CIFAR10 problem, G&C performs comparably to SGD across different training losses and numbers of samples. This shows us that, despite the large number of parameters, G&C solutions are implicitly regularized. We do note that G&C in this low data regime consistently performs better than SGD though. We computed the standard deviation over 75 random data splits and training seeds.

Sample Count	**Optimizer**	**Best Test Acc**	**Train Loss** (0.55, 0.57)	(0.57, 0.59)	(0.59, 0.61)	(0.61, 0.63)	(0.63, 0.65)	(0.65, 0.67)
24	G&C	**66.59%±0.74%**	66.59%±0.74%	65.91%±0.80%	64.09%±0.96%	61.08%±0.89%	59.33%±0.88%	57.18%±0.89%
	SGD	63.16%±0.87%	63.16%±0.87%	62.02%±0.84%	60.74%±0.73%	58.21%±0.75%	57.62%±0.69%	56.24%±0.55%
16	G&C	**61.10%±0.98%**	61.10%±0.98%	59.54%±0.98%	59.21%±0.90%	57.53%±0.86%	57.71%±0.81%	55.06%±0.70%
	SGD	58.98%±0.69%	58.58%±0.77%	58.98%±0.69%	57.86%±0.79%	57.11%±0.61%	56.77%±0.62%	53.90%±0.50%
8	G&C	**57.17%±0.94%**	54.39%±0.80%	53.99%±0.76%	57.17%±0.94%	54.61%±0.68%	52.66%±0.66%	52.82%±0.62%
	SGD	56.76%±0.71%	56.76%±0.71%	55.02%±0.62%	54.79%±0.72%	54.62%±0.68%	53.39%±0.66%	53.53%±0.55%
4	G&C	**55.51%±0.84%**	55.51%±0.84%	53.59%±0.96%	52.78%±0.82%	52.30%±0.67%	52.38%±0.63%	54.07%±0.72%
	SGD	53.75%±0.62%	53.49%±0.68%	52.14%±0.51%	53.75%±0.62%	51.53%±0.63%	52.18%±0.66%	50.44%±0.55%
2	G&C	**52.39%±0.67%**	51.66%±0.74%	52.39%±0.67%	52.00%±0.60%	51.37%±0.56%	50.01%±0.71%	50.66%±0.62%
	SGD	51.98%±0.59%	51.93%±0.66%	51.39%±0.47%	51.98%±0.59%	51.16%±0.48%	50.65%±0.45%	50.05%±0.43%

However, our main argument is that optimizer-specific bias is not needed to explain generalization, and may not even be the primary cause of generalization behavior; in our experiments here, the bulk of generalization can be explained by the geometry of the loss landscape.

7.4.3 Results on 10-class CIFAR-10/MNIST

In this section, we evaluate the importance of gradient-based optimizers in the setting where more classes are involved. However, the Guess & Check algorithm is no longer feasible due to the exponential time complexity. Instead of Guess & Check, we employ Pattern Search and Greedy Random Search to evaluate the dependence of generalization on gradient based optimizers. Again, we find that these non-gradient based optimizers offer similar levels of generalization benefits as SGD despite not using any gradient information at all.

In Table 7.4, we see that Greedy Random Search and Pattern Search both generalize comparably to SGD. The average performance difference is only 0.9% across different sample sizes, datasets, and optimizer combinations. In several cases, Pattern Search even performs better

Table 7.4: In this table, we trained the same LeNet, but with more examples and more classes. We found that the generalization performance is still fairly similar between SGD and alternative zeroth order optimizers that do not use any gradient information. The empty cells indicate the experiment has timed out, and we failed to find models achieving 100% training accuracy within a reasonable time limit.

	Sample Count	1000	500	300	100
MNIST	SGD	93.46%±0.11%	90.15%±0.22%	**87.48%±0.26%**	**78.67%±0.51%**
	Pattern Search	**93.68%±0.12%**	90.33%±0.12%	87.26%±0.30%	78.43%±0.46%
	Random Greedy	93.34%±0.08%	**90.35%±0.10%**	87.33%±0.21%	78.51%±0.50%
CIFAR-10	SGD	**36.01%±0.25%**	29.91%±0.31%	**25.88%±0.34%**	**19.86%±0.27%**
	Pattern Search	-	**30.00%±0.69%**	25.04%±0.66%	18.70%±1.22%
	Random Greedy	34.44%±0.54%	27.06%±0.75%	24.04%±0.58%	16.80%±0.13%

than SGD. Even though 0.9% may seem large when viewed from the perspective of achieving state-of-the-art accuracy, we note that a performance difference of 0.9% is within the margin that can be expected from hyperparameter tuning, and that we have not tuned either of the zeroth order optimizers, and yet they still achieve a comparable level of generalization to SGD.

7.4.4 Few-Shot Learning with ResNets

In this section, we evaluate the importance of gradient-based optimizers in the few-shot setting. This setting enables us to compare gradient methods to zeroth order optimization using industrial-scale models. For the most part, we find that the gradient-free Pattern Search optimizer performs comparably to SGD in the 1-shot setting.

Few-shot learning is usually used to test the ability of models to generalize to unseen tasks given limited training examples. This is a perfect evaluation task for our hypothesis for the following reasons: First, in few-shot learning, we only use 1 or 5 training images per class during the evaluation stage, which makes zeroth order optimization possible. Second, few-shot learners usually utilize a pre-trained feature extractor on the base classes, and only learn a new

Table 7.5: 1-shot-5-way classification performance on both CIFAR-FS and mini-ImageNet with ResNet-12 backbone. We provide mean test accuracy over 600 episodes and the one standard error. Compared to SGD, Pattern Search can always achieve better performance by a large margin.

Optimizer	CIFAR-FS	mini-ImageNet
SGD	68.35 ± 0.46	55.76 ± 0.42
Pattern Search	$\mathbf{70.25 \pm 0.45}$	$\mathbf{58.53 \pm 0.41}$

classification head by SGD or other solvers such as SVM and ridge regression [170, 186] given the unseen tasks. This setting limits the dimension of learnable parameters, thus making training deeper networks such as ResNet possible with these non-gradient based optimizers. Finally, although we only attempt to learn a single layer, due to the few training examples (1 or 5 per class), we will still have an overparameterized model, which is the setting we are interested in.

We evaluate the effectiveness of non-gradient based optimizers on CIFAR-FS [170] and mini-ImageNet [171] with ResNet-12, a commonly used architecture in the few-shot classification literature. During the training stage, we pre-train a feature extractor on the base classes and evaluate the generalization on unseen tasks via 1-shot-5-way episodes, where each episode is a 5-way classification problem and each class contains 1 training image. During the evaluation stage, given the unseen episodes and pre-trained feature extractor, we learn a new classification head with a specific optimizer and evaluate the performance on the testing images. We compare the testing accuracy on 600 different episodes between SGD, and Pattern Search in Table 7.5. For Pattern Search, instead of stopping optimization immediately after fitting all the training examples, we keep updating the model until t steps, where we set $t = 3000$ for both CIFAR-FS and mini-ImageNet. As showed in Table 7.5, Pattern Search always outperforms SGD by a large margin, i.e., over 2% for both CIFAR-FS and mini-ImageNet, which suggests that gradient-based

	Loss Level						
width	(0.3, 0.35)	(0.35, 0.4)	(0.4, 0.45)	(0.45, 0.5)	(0.5, 0.55)	(0.55, 0.6)	(0.6, 0.65)
1	n/a	n/a	n/a	n/a	n/a	92.11%±1.35%	93.44%±n/a
0.9	n/a	n/a	n/a	n/a	90.89%±n/a	86.33%±6.81%	90.90%±3.03%
0.8	n/a	n/a	93.34%±n/a	83.89%±8.97%	87.71%±n/a	93.29%±n/a	81.94%±7.35%
0.7	97.73%±n/a	96.32%±n/a	95.20%±0.42%	92.30%±1.21%	86.10%±6.24%	80.59%±2.96%	87.37%±n/a
0.6	91.20%±1.40%	89.66%±1.88%	87.76%±2.22%	85.45%±2.07%	83.64%±2.78%	82.76%±3.33%	79.43%±3.06%
0.5	89.21%±0.47%	87.01%±0.50%	85.18%±0.56%	84.69%±0.54%	81.91%±0.62%	78.61%±0.65%	75.37%±0.63%
0.4	85.82%±0.53%	83.78%±0.61%	81.43%±0.61%	79.63%±0.65%	77.50%±0.64%	75.98%±0.66%	72.30%±0.77%
0.3	79.55%±0.60%	79.13%±0.60%	76.63%±0.72%	75.43%±0.63%	74.22%±0.66%	72.28%±0.72%	71.75%±0.67%
0.2	77.39%±0.56%	76.08%±0.63%	74.70%±0.56%	73.40%±0.60%	71.84%±0.60%	69.66%±0.64%	68.49%±0.58%

Table 7.6: Performance of G&C on MNIST with 16 samples as we scale up the model. The n/a indicates that a model has not been found for the cell.

optimizers are not necessary in the few-shot setting.

7.5 How does G&C behave as we scale up the model?

People have observed that increasing the size of neural networks trained with SGD can lead to either double descent behavior [187] or increasing performance [188]. This phenomenon has been previously attributed to the regularization effect of SGD. However, given the similarity of performance between G&C models and SGD-trained models, it is natural to ask whether we observe similar behaviors in G&C models as we increase the number of parameters. Here, we show that G&C models continue to improve as we increase their size, without using SGD.

To investigate this further, we conducted experiments on 2-class MNIST using G&C with varying widths of LeNet (Table 7.6). Surprisingly, we found that as we increased the width of the model, its validation accuracy also increased. This observation contradicts generalization theories based on model capacity, which suggest that increasing model size beyond a certain point leads to overfitting and reduced generalization performance.

The observation points to the hypothesis that increasing the width of a neural network can expand the volume of the good function class. If we can identify the specific function within

this class that experiences an increase in volume with more parameters, it may be possible to achieve similar benefits with a smaller model that captures the favorable properties of the function. This could lead to more efficient and effective deep learning models that perform better without unnecessary parameter bloat. However, further research is required to investigate this hypothesis and identify the specific functions that contribute to the observed increase in volume.

7.6 A Toy Example: Simplicity bias may originate from the volume bias as opposed to SGD

In this section, we study whether volume may explain the simplicity bias previously observed in [189].

While we have mostly measured bias in terms of generalization in this chapter, we think it is a promising future direction to quantify whether other forms of bias can be attributed to the volume hypothesis instead. One example is simplicity bias, where trained neural networks strongly prefer linear decision boundaries compared to robust ones. We provide a toy illustration on why this may be attributed to the volume hypothesis, but leave further exploration of this as future work.

Consider the following example: a trained neural network on the slab dataset ignores the more complex y-axis, as shown on the left of Figure 7.3, and uses a linear decision boundary drawn along the x-axis only as opposed to the robust decision boundary shown on the right of Figure 7.3. While [189] has attributed the simplicity bias in this example to SGD, we found that the simplicity bias may simply originate from the large disparity in volumes between the linear and robust functions in the loss landscape. In fact, when we used G&C to measure the

volume of the two respective decision boundaries in the parameter space, we found that the linear decision boundary has volume that is 6 orders of magnitude larger than that of the robust decision boundary. Specifically, we estimate the volume of the solution by taking the reciprocal of the number of guesses before a solution is obtained. The volume disparity may explain why the simple decision boundary is strongly preferred compared to the alternative.

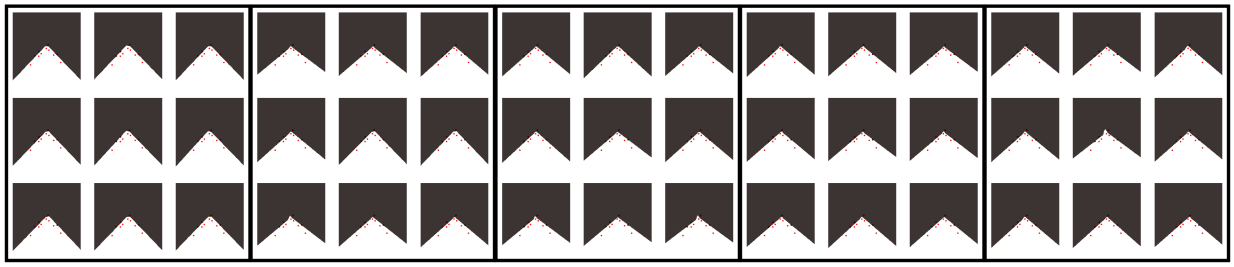
7.7 Chapter Summary

In this chapter, we empirically show that gradient-based implicit regularization of training dynamics is not required for generalization. Instead, we consider non-gradient optimizers that lack gradient dynamics, yet still perform well. The strong performance of gradient-free optimizers, in particular Guess & Check, strongly suggests that the disparate volume of good and bad hypothesis classes is the main implicit bias that enables these optimizers to succeed. For future work, we think more critically examining the role of volume as implicit bias in neural networks will be a fruitful and interesting direction.

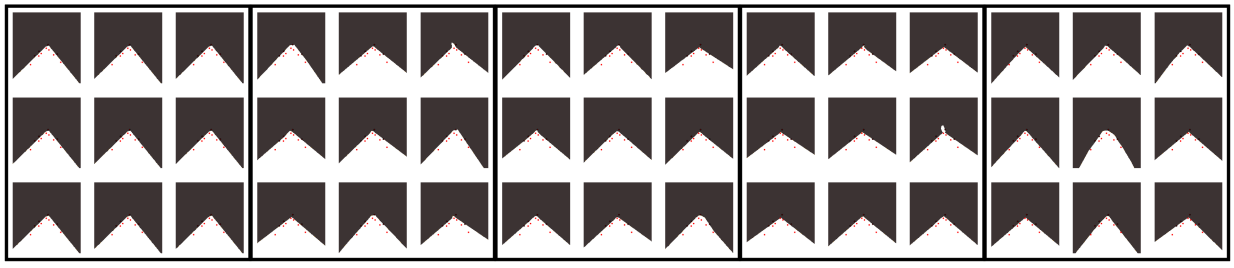
Joint work with Renkun Ni, David Miller, Arpit Bansal, Jonas Geiping, Micah Goldblum, Tom Goldstein first published in ICLR 2023

Figure 7.2: In this figure, we show that even though the model becomes much more expressive as we increase the number of parameters, as shown in the possible decision boundaries of the poisoned network of various sizes, *both SGD and Guess & Check produce decision boundaries that are relatively stable* as we increase the number of parameters. From left to right, we have decision boundaries produced by 2, 4, 10, 15, 20 hidden units single layer neural networks with different training methods. For each (training method, model size) pair, we show 9 randomly sampled decision boundaries of the trained network. We showed even more samples of the decision boundaries in Appendix A

(a) Decision boundaries of a poisoned neural network



(b) Decision boundaries of SGD trained models



(c) Decision boundaries of Guess & Check trained models

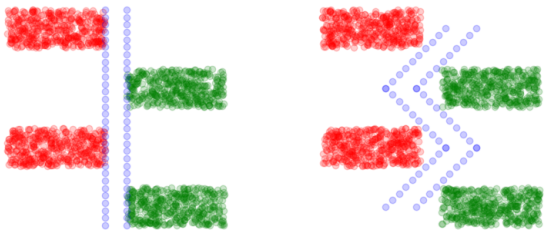


Figure 7.3: The volume of the decision boundary on the left as measured by G&C is 10^{-4} whereas the volume of the robust/complex decision boundary has volume smaller than 10^{-10}. The large volume disparity may explain trained network's strong preference for the linear solution.

Bibliography

[1] Charles Herrmann, Kyle Sargent, Lu Jiang, Ramin Zabih, Huiwen Chang, Ce Liu, Dilip Krishnan, and Deqing Sun. Pyramid adversarial training improves vit performance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13419–13429, 2022.

[2] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJzIBfZAb`.

[3] Florian Tramer and Dan Boneh. Adversarial training and robustness for multiple perturbations. *Advances in neural information processing systems*, 32, 2019.

[4] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *Advances in Neural Information Processing Systems*, pages 3353–3364, 2019.

[5] Cassidy Laidlaw and Soheil Feizi. Functional adversarial attacks. *Advances in neural information processing systems*, 32, 2019.

[6] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. *arXiv preprint arXiv:1712.02779*, 2017.

[7] Eric Wong, Frank Schmidt, and Zico Kolter. Wasserstein adversarial examples via projected sinkhorn iterations. In *International Conference on Machine Learning*, pages 6808–6817. PMLR, 2019.

[8] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.

[9] Danny Karmon, Daniel Zoran, and Yoav Goldberg. Lavan: Localized and visible adversarial noise. In *International Conference on Machine Learning*, pages 2507–2515. PMLR, 2018.

[10] Mark Lee and J. Zico Kolter. On Physical Adversarial Patches for Object Detection, 2019.

[11] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. A general framework for adversarial examples with objectives. *ACM Transactions on Privacy and Security (TOPS)*, 22(3):1–30, 2019.

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

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

[14] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pages 274–283. PMLR, 2018.

[15] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[16] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3578–3586, 2018.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[18] K Simonyan and A Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR 2015)*. Computational and Biological Learning Society, 2015.

[19] 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.

[20] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *international conference on machine learning*, pages 1310–1320. PMLR, 2019.

[21] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International conference on machine learning*, pages 5286–5295. PMLR, 2018.

[22] Haichao Zhang and Jianyu Wang. Towards adversarially robust object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 421–430, 2019.

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[24] Shiqi Wang, Yizheng Chen, Ahmed Abdou, and Suman Jana. Enhancing gradient-based attacks with symbolic intervals, 2019.

[25] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=Skxuk1rFwB`.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *ICLR Workshop*, 2017.

[28] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning models. In *CVPR*, 2018.

[29] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *CVPR*, 2017.

[30] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[31] Rudy Bunel, P Mudigonda, Ilker Turkaslan, Philip Torr, Jingyue Lu, and Pushmeet Kohli. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(2020), 2020.

[32] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018.

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[34] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[35] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

[36] Xin Liu, Huanrui Yang, Ziwei Liu, Linghao Song, Hai Li, and Yiran Chen. DPatch: An Adversarial Patch Attack on Object Detectors, 2018.

[37] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *ACM CCS*, 2016.

[38] Zuxuan Wu, Ser-Nam Lim, Larry S Davis, and Tom Goldstein. Making an invisibility cloak: Real world adversarial attacks on object detectors. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*, pages 1–17. Springer, 2020.

[39] Quanyu Liao, Xin Wang, Bin Kong, Siwei Lyu, Bin Zhu, Youbing Yin, Qi Song, and Xi Wu. Transferable adversarial examples for anchor free object detection. In *2021 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2021.

[40] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017.

[41] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755, 2014.

[42] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection. In *ICCV*, 2017.

[43] Xingxing Wei, Siyuan Liang, Ning Chen, and Xiaochun Cao. Transferable adversarial attacks for image and video object detection. In *IJCAI*, 2019.

[44] Yuezun Li, Daniel Tian, Ming-Ching Chang, Xiao Bian, and Siwei Lyu. Robust adversarial perturbation on deep proposal-based models. In *BMVC*, 2018.

[45] Yuezun Li, Xiao Bian, Ming-Ching Chang, and Siwei Lyu. Exploring the vulnerability of single shot module in object detectors via imperceptible background patches. In *BMVC*, 2019.

[46] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Florian Tramer, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Physical adversarial examples for object detectors. In *WOOT*, 2018.

[47] Shang-Tse Chen, Cory Cornelius, Jason Martin, and Duen Horng Polo Chau. Shapeshifter: Robust Physical Adversarial Attack on Faster R-CNN Object Detector. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 52–68. Springer, 2018.

[48] Simen Thys, Wiebe Van Ranst, and Toon Goedemé. Fooling automated surveillance cameras: adversarial patches to attack person detection. In *CVPR Workshop*, 2019.

[49] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[50] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems*, pages 10877–10887, 2018.

[51] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2018.

[52] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in neural information processing systems*, pages 4939–4948, 2018.

[53] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3 (POPL):1–30, 2019.

[54] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified Robustness to Adversarial Examples with Differential Privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672, 2019.

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

[56] Hadi Salman, Jerry Li, Ilya Razenshteyn, Pengchuan Zhang, Huan Zhang, Sebastien Bubeck, and Greg Yang. Provably Robust Deep Learning via Adversarially Trained Smoothed Classifiers. In *Advances in Neural Information Processing Systems*, 2019.

[57] Hadi Salman, Mingjie Sun, Greg Yang, Ashish Kapoor, and J Zico Kolter. Denoised smoothing: A provable defense for pretrained classifiers. *Advances in Neural Information Processing Systems*, 33:21945–21957, 2020.

[58] Aounon Kumar, Alexander Levine, Soheil Feizi, and Tom Goldstein. Certifying confidence via randomized smoothing. *Advances in Neural Information Processing Systems*, 33:5165–5177, 2020.

[59] Wangmeng Zuo, Kai Zhang, and Lei Zhang. *Convolutional Neural Networks for Image Denoising and Restoration*, pages 93–123. Springer International Publishing, 2018.

[60] Anonymous. Tight second-order certificates for randomized smoothing. In *Submitted to International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=1AyPW2Emp6`. under review.

[61] Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American Economic Review*, 97(1):242–259, 2007.

[62] Tuomas Sandholm. Expressive commerce and its application to sourcing: How we conducted $35 billion of generalized combinatorial auctions. *AI Magazine*, 28(3):45–45, 2007.

[63] Peter Cramton. The FCC spectrum auctions: An early assessment. *Journal of Economics & Management Strategy*, 6(3):431–495, 1997.

[64] Kevin Leyton-Brown, Paul Milgrom, and Ilya Segal. Economics and computer science of a radio spectrum reallocation. *Proceedings of the National Academy of Sciences (PNAS)*, 114(28):7202–7209, 2017.

[65] Paul Milgrom. *Discovering prices: auction design in markets with complex constraints*. Columbia University Press, 2017.

[66] Alvin E Roth. Marketplaces, markets, and market design. *American Economic Review*, 108(7):1609–58, 2018.

[67] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1):8–37, 1961.

[68] Edward H Clarke. Multipart pricing of public goods. *Public choice*, pages 17–33, 1971.

[69] Theodore Groves. Incentives in teams. *Econometrica: Journal of the Econometric Society*, pages 617–631, 1973.

[70] Roger B Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1): 58–73, 1981.

[71] Alejandro M. Manelli and Daniel R. Vincent. Bundling as an optimal selling mechanism for a multiple-good monopolist. *J. Econ. Theory*, 127(1):1–35, 2006.

[72] Gregory Pavlov. Optimal mechanism for selling two goods. *The BE Journal of Theoretical Economics*, 11(1), 2011.

[73] Nima Haghpanah and Jason D. Hartline. Reverse mechanism design. *CoRR*, abs/1404.1341, 2014.

[74] Constantinos Daskalakis, Alan Deckelbaum, and Christos Tzamos. Strong duality for a multiple-good monopolist. In *Economics and Computation (EC)*, 2015.

[75] Paul Duetting, Zhe Feng, Harikrishna Narasimhan, David C. Parkes, and Sai Srivatsa Ravindranath. Optimal auctions through deep learning. In *International Conference on Machine Learning (ICML)*, 2019.

[76] Yang Cai, Constantinos Daskalakis, and S Matthew Weinberg. An algorithmic characterization of multi-dimensional mechanisms. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 459–478, 2012.

[77] Yang Cai, Constantinos Daskalakis, and S Matthew Weinberg. Optimal multi-dimensional mechanism design: Reducing revenue to welfare maximization. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 130–139. IEEE, 2012.

[78] Yang Cai, Constantinos Daskalakis, and S Matthew Weinberg. Understanding incentives: Mechanism design becomes algorithm design. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 618–627. IEEE, 2013.

[79] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018.

[80] Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning (ICML)*, 2018.

[81] Shiqi Wang, Yizheng Chen, Ahmed Abdou, and Suman Jana. Enhancing gradient-based attacks with symbolic intervals. *CoRR*, abs/1906.02282, 2019.

[82] Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and Pawan Kumar Mudigonda. A unified view of piecewise linear neural network verification. In *Neural Information Processing Systems (NeurIPS)*, 2018.

[83] Jingyue Lu and M. Pawan Kumar. Neural network branching for neural network verification. In *International Conference on Learning Representations (ICLR)*, 2020.

[84] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations (ICLR)*, 2019.

[85] André F. T. Martins and Ramón Fernández Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International Conference on Machine Learning (ICML)*, 2016.

[86] Noah Golowich, Harikrishna Narasimhan, and David C. Parkes. Deep learning for multi-facility location mechanism design. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[87] Zhe Feng, Harikrishna Narasimhan, and David C. Parkes. Deep learning for revenue-optimal auctions with budgets. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2018.

[88] Padala Manisha, CV Jawahar, and Sujit Gujar. Learning optimal redistribution mechanisms through neural networks. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2018.

[89] Weiran Shen, Binghui Peng, Hanpeng Liu, Michael Zhang, Ruohan Qian, Yan Hong, Zhi Guo, Zongyao Ding, Pengjun Lu, and Pingzhong Tang. Reinforcement mechanism design, with applications to dynamic pricing in sponsored search auctions. *CoRR*, abs/1711.10279, 2017.

[90] Andrea Tacchetti, DJ Strouse, Marta Garnelo, Thore Graepel, and Yoram Bachrach. A neural architecture for designing truthful and efficient auctions. *CoRR*, abs/1907.05181, 2019.

[91] Vincent Conitzer and Tuomas Sandholm. Applications of automated mechanism design. 2003.

[92] Tuomas Sandholm. Automated mechanism design: A new application area for search algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pages 19–36. Springer, 2003.

[93] Anton Likhodedov and Tuomas Sandholm. Methods for boosting revenue in combinatorial auctions. In *AAAI*, pages 232–237, 2004.

[94] Tuomas Sandholm and Anton Likhodedov. Automated design of revenue-maximizing combinatorial auctions. *Operations Research*, 63(5):1000–1025, 2015.

[95] Maria-Florina F Balcan, Tuomas Sandholm, and Ellen Vitercik. Sample complexity of automated mechanism design. In *Advances in Neural Information Processing Systems*, pages 2083–2091, 2016.

[96] Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis*. Springer, 2017.

[97] Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2020. URL `http://www.gurobi.com`.

[98] Kris Korrel. sparsemax-pytorch, May 2020. URL `https://doi.org/10.5281/zenodo.3860669`.

[99] Kris Korrel, Dieuwke Hupkes, Verna Dankers, and Elia Bruni. Transcoding compositionally: Using attention to find more generalizable solutions. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 2019.

[100] Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256, April 2007.

[101] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[102] Kai Y. Xiao, Vincent Tjeng, Nur Muhammad (Mahi) Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing relu stability. In *International Conference on Learning Representations (ICLR)*, 2019.

[103] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy A. Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *CoRR*, abs/1810.12715, 2018.

[104] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems (NeurIPS)*. 2019.

[105] Jad Rahme, Samy Jelassi, and S Matthew Weinberg. Auction learning as a two-player game. *arXiv preprint arXiv:2006.05684*, 2020.

[106] Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Estimating approximate incentive compatibility. In *Economics and Computation (EC)*, 2019.

[107] Atila Abdulkadiroglu, Parag Pathak, Alvin E Roth, and Tayfun Sonmez. Changing the boston school choice mechanism. Working Paper 11965, National Bureau of Economic Research, January 2006.

[108] Dario Amodei and Danny Hernandez. Ai and compute. *Heruntergeladen von https://blog. openai. com/aiand-compute*, 2018.

[109] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[110] Chuan Li. *OpenAI's GPT-3 Language Model: A Technical Overview*, 2020. URL `https://lambdalabs.com/blog/demystifying-gpt-3/#1`.

[111] Frank Hartung and Martin Kutter. Multimedia watermarking techniques. *Proceedings of the IEEE*, 87(7):1079–1107, 1999.

[112] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, pages 269–277, 2017.

[113] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 159–172, 2018.

[114] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1615–1631, 2018.

[115] Masoumeh Shafieinejad, Nils Lukas, Jiaqi Wang, Xinda Li, and Florian Kerschbaum. On the robustness of backdoor-based watermarking in deep neural networks. In *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security*, pages 177–188, 2021.

[116] William Aiken, Hyoungshick Kim, Simon Woo, and Jungwoo Ryoo. Neural network laundering: Removing black-box backdoor watermarks from deep neural networks. *Computers & Security*, 106:102277, 2021.

[117] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: A generic watermarking framework for ip protection of deep learning models. *arXiv preprint arXiv:1804.00750*, 2018.

[118] T. Wang and F. Kerschbaum. Attacks on digital watermarks for deep neural networks. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2622–2626, 2019.

[119] Tianhao Wang and Florian Kerschbaum. Robust and undetectable white-box watermarks for deep neural networks. *arXiv preprint arXiv:1910.14268*, 2019.

[120] Fang-Qi Li and Shi-Lin Wang. Persistent watermark for image classification neural networks by penetrating the autoencoder. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 3063–3067. IEEE, 2021.

[121] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N Asokan. Dawn: Dynamic adversarial watermarking of neural networks. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 4417–4425, 2021.

[122] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. Deep neural network fingerprinting by conferrable adversarial examples. *arXiv preprint arXiv:1912.00888*, 2019.

[123] Xinyun Chen, Wenxiao Wang, Yiming Ding, Chris Bender, Ruoxi Jia, Bo Li, and Dawn Song. Leveraging unlabeled data for watermark removal of deep neural networks. In *ICML workshop on Security and Privacy of Machine Learning*, 2019.

[124] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918*, 2019.

[125] Ping-yeh Chiang, Renkun Ni, Ahmed Abdelkader, Chen Zhu, Christoph Studor, and Tom Goldstein. Certified defenses for adversarial patches. In *International Conference on Learning Representations*, 2019.

[126] Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.

[127] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3575–3583, 2018.

[128] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S. Dhillon, and Luca Daniel. Towards fast computation of certified robustness for relu networks, 2018.

[129] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks, 2019.

[130] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning models. *arXiv preprint arXiv:1707.08945*, 2017.

[131] Alexander Levine and Soheil Feizi. Robustness certificates for sparse adversarial attacks by randomized ablation. *arXiv preprint arXiv:1911.09272*, 2019.

[132] Ping-yeh Chiang, Michael J Curry, Ahmed Abdelkader, Aounon Kumar, John Dickerson, and Tom Goldstein. Detection as regression: Certified object detection by median smoothing. *arXiv preprint arXiv:2007.03730*, 2020.

[133] Ben Goldberger, Guy Katz, Yossi Adi, and Joseph Keshet. Minimal modifications of deep neural networks using verification. In Elvira Albert and Laura Kovacs, editors, *LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 73 of *EPiC Series in Computing*, pages 260–278. EasyChair, 2020. doi: 10.29007/699q. URL `https://easychair.org/publications/paper/CWhF`.

[134] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness, 2019.

[135] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer, 2019.

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

[137] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[138] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Adversarial generative nets: Neural network attacks on state-of-the-art face recognition. *CoRR*, abs/1801.00349, 2018. URL `http://arxiv.org/abs/1801.00349`.

[139] Nils Lukas, Edward Jiang, Xinda Li, and Florian Kerschbaum. Sok: How robust is image classification deep neural network watermarking?(extended version). *arXiv preprint arXiv:2108.04974*, 2021.

[140] Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L Yuille, and Quoc V Le. Adversarial examples improve image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 819–828, 2020.

[141] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *Advances in Neural Information Processing Systems*, 32, 2019.

[142] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. *Advances in Neural Information Processing Systems*, 32, 2019.

[143] Jieru Mei, Yucheng Han, Yutong Bai, Yixiao Zhang, Yingwei Li, Xianhang Li, Alan Yuille, and Cihang Xie. Fast advprop. *arXiv preprint arXiv:2204.09838*, 2022.

[144] Haizhong Zheng, Ziqi Zhang, Juncheng Gu, Honglak Lee, and Atul Prakash. Efficient adversarial training with transferable adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1181–1190, 2020.

[145] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. *CoRR*, abs/2001.03994, 2020. URL `https://arxiv.org/abs/2001.03994`.

[146] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[147] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJzIBfZAb`.

[148] Jiawang Bai, Li Yuan, Shu-Tao Xia, Shuicheng Yan, Zhifeng Li, and Wei Liu. Improving vision transformers by revisiting high-frequency components. *arXiv preprint arXiv:2204.00993*, 2022.

[149] Xiaofeng Mao, Yuefeng Chen, Ranjie Duan, Yao Zhu, Gege Qi, Shaokai Ye, Xiaodan Li, Rong Zhang, and Hui Xue. Enhance the visual representation via discrete adversarial training. *arXiv preprint arXiv:2209.07735*, 2022.

[150] Ali Shafahi, Mahyar Najibi, Zheng Xu, John Dickerson, Larry S Davis, and Tom Goldstein. Universal adversarial training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5636–5643, 2020.

[151] Philipp Benz, Chaoning Zhang, Adil Karjauv, and In So Kweon. Universal adversarial training with class-wise perturbations. In *2021 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2021. doi: 10.1109/ICME51207.2021.9428419.

[152] Tao Bai, Jinqi Luo, Jun Zhao, Bihan Wen, and Qian Wang. Recent advances in adversarial training for adversarial robustness. *arXiv preprint arXiv:2102.01356*, 2021.

[153] Lucas Beyer, Xiaohua Zhai, and Alexander Kolesnikov. Better plain vit baselines for imagenet-1k. *arXiv preprint arXiv:2205.01580*, 2022.

[154] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 702–703, 2020.

[155] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[156] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[157] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. *CVPR*, 2021.

[158] Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. Learning robust global representations by penalizing local predictive power. In *Advances in Neural Information Processing Systems*, pages 10506–10518, 2019.

[159] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bygh9j09KX.

[160] Andreas Peter Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=4nPswr1KcP.

[161] Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *CoRR*, abs/1712.09913, 2017. URL http://arxiv.org/abs/1712.09913.

[162] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=6Tm1mposlrM.

[163] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations (ICLR) 2017*, 2017. URL https://openreview.net/forum?id=Sy8gdB9xx.

[164] Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/c0c783b5fc0d7d808f1d14a6e9c8280d-Paper.pdf.

[165] Madhu S. Advani, Andrew M. Saxe, and Haim Sompolinsky. High-dimensional dynamics of generalization error in neural networks. *Neural Networks*, 132:428–446, 2020. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2020.08.022. URL https://www.sciencedirect.com/science/article/pii/S0893608020303117.

[166] Jinlong Liu, Yunzhi Bai, Guoqing Jiang, Ting Chen, and Huayan Wang. Understanding why neural networks generalize well through gsnr of parameters. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HyevIJStwH.

[167] Tomer Galanti and Tomaso Poggio. Sgd noise and implicit low-rank bias in deep neural networks. 03/2022 2022. URL https://cbmm.mit.edu/publications/sgd-noise-and-implicit-low-rank-bias-deep-neural-networks.

[168] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[169] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[170] Luca Bertinetto, Joao F. Henriques, Philip Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyxnZh0ct7.

[171] Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. *CoRR*, abs/1606.04080, 2016. URL http://arxiv.org/abs/1606.04080.

[172] W. Ronny Huang, Zeyad Emam, Micah Goldblum, Liam Fowl, Justin K. Terry, Furong Huang, and Tom Goldstein. Understanding generalization through visualizations. In Jessica Zosa Forde, Francisco Ruiz, Melanie F. Pradier, and Aaron Schein, editors, *Proceedings on "I Can't Believe It's Not Better!" at NeurIPS Workshops*, volume 137 of *Proceedings of Machine Learning Research*, pages 87–97. PMLR, 12 Dec 2020. URL https://proceedings.mlr.press/v137/huang20a.html.

[173] Dominikus Noll. Convergence of non-smooth descent methods using the kurdyka–łojasiewicz inequality. *Journal of Optimization Theory and Applications*, 160:553–572, 2014.

[174] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

[175] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pages 1376–1401. PMLR, 2015.

[176] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016. URL http://arxiv.org/abs/1609.04836.

[177] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In Doina Precup and Yee Whye Teh, editors, *Proceedings of*

the 34th International Conference on Machine Learning, volume 70 of *Proceedings of Machine Learning Research*, pages 1019–1028. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/dinh17b.html.

[178] Guillermo Valle-Perez, Chico Q. Camargo, and Ard A. Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rye4g3AqFm.

[179] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.

[180] Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33: 4697–4708, 2020.

[181] David JC MacKay, David JC Mac Kay, et al. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[182] Samuel L Smith and Quoc V Le. A bayesian perspective on generalization and stochastic gradient descent. In *International Conference on Learning Representations*, 2018.

[183] Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *CoRR*, abs/1703.11008, 2017. URL https://arxiv.org/abs/1703.11008.

[184] Chris Mingard, Guillermo Valle-Pérez, Joar Skalse, and Ard A. Louis. Is sgd a bayesian sampler? well, almost. *Journal of Machine Learning Research*, 22(79):1–64, 2021. URL http://jmlr.org/papers/v22/20-676.html.

[185] Jonas Geiping, Micah Goldblum, Phil Pope, Michael Moeller, and Tom Goldstein. Stochastic training is not necessary for generalization. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=ZBESeIUB5k.

[186] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10657–10665, 2019.

[187] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

[188] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[189] Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. *Advances in Neural Information Processing Systems*, 33:9573–9585, 2020.