University of California
Santa Barbara

Structural Defense Techniques in Adversarial Machine Learning

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Can Bakiskan

Committee in charge:

Professor Upamanyu Madhow, Chair
Professor Ramtin Pedarsani
Professor Kenneth Rose
Professor Ambuj Singh

September 2022

The Dissertation of Can Bakiskan is approved.

Professor Ramtin Pedarsani

Professor Kenneth Rose

Professor Ambuj Singh

Professor Upamanyu Madhow, Committee Chair

September 2022

Structural Defense Techniques in Adversarial Machine Learning

by

Can Bakiskan

To my wife Laurel, my parents Emine and Faruk, and my sister Evrim

Acknowledgements

First and foremost, I would like to thank my advisor, Professor Upamanyu Madhow, for accepting me as a student, his support and patience with me throughout the years, his technical expertise and willingness to guide me, and his understanding nature in difficult situations. I consider myself lucky to have been your student. I want to also express my gratitude for the members of my dissertation committee, Professors Ramtin Pedarsani, Kenneth Rose and Ambuj Singh, for taking the time to review and help by giving their constructive feedback.

I want to thank all past and present members of the Wireless Communication and Sensornets Lab for helpful discussions and their friendship: Metehan Cekic, Ahmet Dundar Sezer, Bhagyashree Puranik, Soorya Gopalakrishnan, Maryam Eslami Rasekh, Mohammed Abdelghany, Lalitha Giridhar, Anant Gupta and Zhinus Marzi. I am especially thankful to Metehan Cekic and Ahmet Dundar Sezer for many late night studies, Zoom meetings, stimulating discussions and many collaborations.

I thank all my past teachers and instructors for their knowledge and enthusiasm to teach me what they know. I would not be here if it was not for you.

Special thanks go to Army Research Office and National Science Foundation for the financial support for part of my Ph.D. studies.

Last but not least, I want to thank my parents Emine and Faruk, my wife Laurel and my sister Evrim from the bottom of my heart for their unconditional support during challenging times.

Curriculum Vitæ
Can Bakiskan

Education

2022	Ph.D. in Electrical and Computer Engineering, University of California, Santa Barbara, California, USA
2019	M.S. in Electrical and Computer Engineering, University of California, Santa Barbara, California, USA
2017	B.S. in Electrical and Electronics Engineering, Boğaziçi University, İstanbul, Turkey
2017	B.S. in Mathematics, Boğaziçi University, İstanbul, Turkey

Publications

1. **Can Bakiskan**, Metehan Cekic, Upamanyu Madhow. "Early Layers Are More Important For Adversarial Robustness", *in International Conference on Machine Learning (ICML) Workshop – New Frontiers in Adversarial Machine Learning*, 2022
2. Metehan Cekic*, **Can Bakiskan***, Upamanyu Madhow. "Layerwise Hebbian/anti-Hebbian (HaH) Learning In Deep Networks: A Neuro-inspired Approach To Robustness", *in International Conference on Machine Learning (ICML) Workshop – New Frontiers in Adversarial Machine Learning*, 2022
3. Metehan Cekic*, **Can Bakiskan***, Upamanyu Madhow. "Neuro-Inspired Deep Neural Networks with Sparse, Strong Activations", *in IEEE International Conference on Image Processing (ICIP)*, 2022
4. Metehan Cekic, **Can Bakiskan**, Upamanyu Madhow, "Towards Robust, Interpretable Neural Networks via Hebbian/anti-Hebbian Learning: A Software Framework for Training with Feature-Based Costs", *in Software Impacts Journal*, 2022
5. **Can Bakiskan**, Metehan Cekic, Ahmet Dundar Sezer, Upamanyu Madhow. "Sparse Coding Front End for Robust Neural Networks", *in International Conference on Learning Representations (ICLR) Workshop on Security and Safety in Machine Learning Systems*, 2021
6. **Can Bakiskan**, Metehan Cekic, Ahmet Dundar Sezer, Upamanyu Madhow. "A Neuro-Inspired Autoencoding Defense Against Adversarial Attacks", *in IEEE International Conference on Image Processing (ICIP)*, 2021
7. **Can Bakiskan**, Soorya Gopalakrishnan, Metehan Cekic, Upamanyu Madhow. "Polarizing Front Ends for Robust CNNs", *in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020

Abstract

Structural Defense Techniques in Adversarial Machine Learning

by

Can Bakiskan

Over the last decade, deep neural networks (DNNs) have become an increasingly popular choice for researchers looking to take on previously unsolved problems. With the popularity of these networks come concerns about their security and reliability. In particular, DNNs have been shown to be vulnerable to carefully crafted perturbations that make the networks yield incorrect outcomes while indicating high confidence in those outcomes. There have been many defense mechanisms proposed to combat these attacks, among which adversarial training and its variants have stood the test of time. While adversarial training of DNNs yields state-of-the-art empirical performance, it does not provide insight into the mechanism of robustness, or explicit control over the features being extracted by the network layers. In this dissertation, we seek to address these drawbacks by incorporating bottom-up structural blocks into DNNs, with the aim of providing robustness and extracting interpretable features in a principled manner. Specifically, we use guiding principles from signal processing, sparse representation theory and neuroscience to design network components to incorporate robust features into neural networks.

We begin by presenting an analysis of adversarial training that motivates and justifies further research into shaping the earlier layers of neural networks. Through partial adversarial training and perturbation statistics tracking, we show that early layers play a crucial role in adversarial training.

We then focus our attention on front end based techniques, which process the input to reduce the impact of perturbations before feeding it to a DNN. In one technique, we design and evaluate a front end which polarizes and quantizes the data. We observe that polarization and subsequent quantization eliminates most perturbations and develop algorithms to learn approximately polarizing bases for data. We investigate the effectiveness of the proposed strategy on simple image classification datasets. However, it is more difficult to learn polarizing bases for more complex datasets. This motivates the design of a front end based defense inspired by existing sparse coding techniques. We construct an encoder that uses a sparse overcomplete dictionary, lateral inhibition and drastic nonlinearity, characteristics commonly observed in biological vision, in order to reduce the effects of adversarial perturbations.

Finally, we introduce a promising neuro-inspired approach to DNNs with sparser and stronger activations. We complement the end-to-end discriminative cost function with layer-wise costs promoting Hebbian ("fire together wire together") updates for highly active neurons, and anti-Hebbian updates for the remaining neurons. Instead of batch norm, we use divisive normalization of activations to suppress weak outputs using strong outputs, and L^2 normalization of neuronal weights to provide scale invariance. Experiments demonstrate that, relative to standard end-to-end trained architectures, our proposed architecture leads to sparser activations, exhibits more robustness to noise and other common corruptions, and demonstrates more robustness to adversarial perturbations without adversarial training.

The material in this dissertation is partly based on the following publications.

- **Can Bakiskan**, Metehan Cekic, Upamanyu Madhow. "Early Layers Are More Important For Adversarial Robustness", *in International Conference on Machine Learning (ICML) Workshop – New Frontiers in Adversarial Machine Learning*, 2022

- © 2020 IEEE. Reprinted, with permission from **Can Bakiskan**, Soorya Gopalakrishnan, Metehan Cekic, Upamanyu Madhow. "Polarizing Front Ends for Robust CNNs", *in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020

- © 2021 IEEE. Reprinted, with permission from **Can Bakiskan**, Metehan Cekic, Ahmet Dundar Sezer, Upamanyu Madhow. "A Neuro-Inspired Autoencoding Defense Against Adversarial Attacks", *in IEEE International Conference on Image Processing (ICIP)*, 2021

- **Can Bakiskan**, Metehan Cekic, Ahmet Dundar Sezer, Upamanyu Madhow. "Sparse Coding Front End for Robust Neural Networks", *in International Conference on Learning Representations (ICLR) Workshop on Security and Safety in Machine Learning Systems*, 2021

- © 2022 IEEE. Reprinted, with permission from Metehan Cekic*, **Can Bakiskan***, Upamanyu Madhow. "Neuro-Inspired Deep Neural Networks with Sparse, Strong Activations", *in IEEE International Conference on Image Processing (ICIP)*, 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Deep Learning Revolution and Its Achilles' Heel

The original, groundbreaking paper of Krizhevsky et al. [14] drew attention to the potential of machine learning in general, and deep neural networks (DNNs) in particular, more than a decade ago. Since then, DNNs have been in the spotlight and at the cutting edge of almost any application (spanning science, engineering and beyond) for which large amounts of data are available. There has been an accompanying Cambrian explosion of papers published that focus on using and improving deep neural networks. DNNs can now beat humans in the game of Go [15], predict protein folding [16], control robotic hands [17], beat humans in the video games of DoTA [18] and StarCraft [19], produce surprisingly realistic faces [20], create believable pictures from absurd prompts [1] (see Figure 1.1), write and summarize text comparable to a highly proficient human [21], do multi-modal reasoning [22], and even converse well enough to fool Google employees into thinking they possess consciousness [23]. More importantly, deep neural networks are in the core of many everyday tools and products such as search engines, product recommendation tools, speech recognition and virtual assistant systems or language translation websites, influencing and greatly affecting most people's lives.

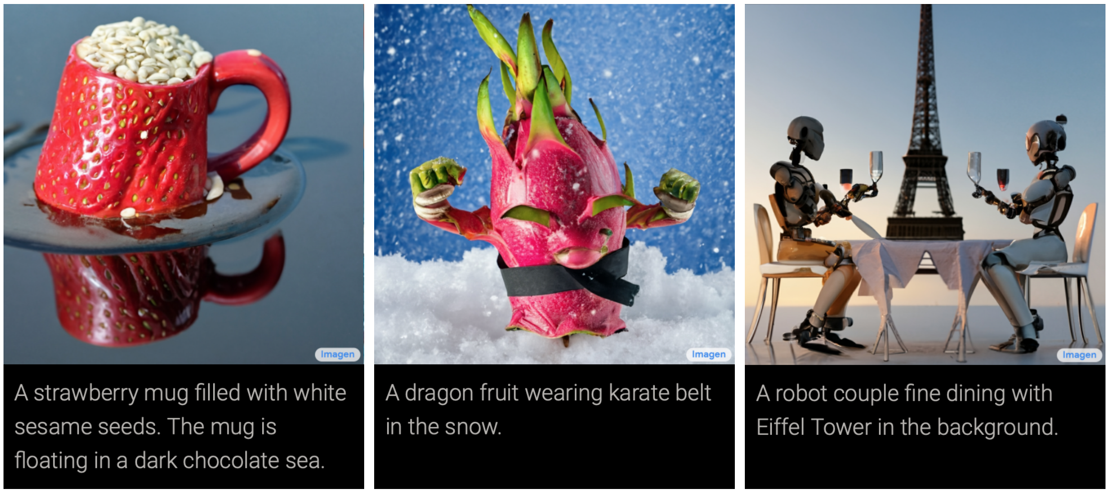
Imagen
A strawberry mug filled with white sesame seeds. The mug is floating in a dark chocolate sea.
Imagen
A dragon fruit wearing karate belt in the snow.
Imagen
A robot couple fine dining with Eiffel Tower in the background.

Figure 1.1: Realistic images generated for absurd prompts. Images taken from [1]

Concomitant with the growing interest in machine learning, however, researchers brought into question the robustness and security of these models. Biggio et al. [24] have exposed the vulnerability of machine learning models such as support vector machines (SVMs) and neural networks to adversarial attacks. These adversarial attacks work by changing the inputs ever so slightly, such that they are imperceptible to humans, to make the models fail at their task and have high confidence while doing so. Other researchers showed that the same kind of attacks are also applicable to DNNs [25] and found even more powerful adversarial attack methods [9]. Kurakin et al. [26] demonstrated that these adversarial examples are not limited to carefully controlled lab conditions but can be implemented in the real world. Eykholt et al. [2] showed that not only can adversarial attacks be implemented physically, but they can affect hypothetical safety critical systems. An example for this is traffic sign recognition models for autonomous driving being fooled using inconspicuous looking alterations like stickers and graffiti (see Figure 1.2).

The issue of adversarial vulnerability is not limited to the visual domain; DNNs that

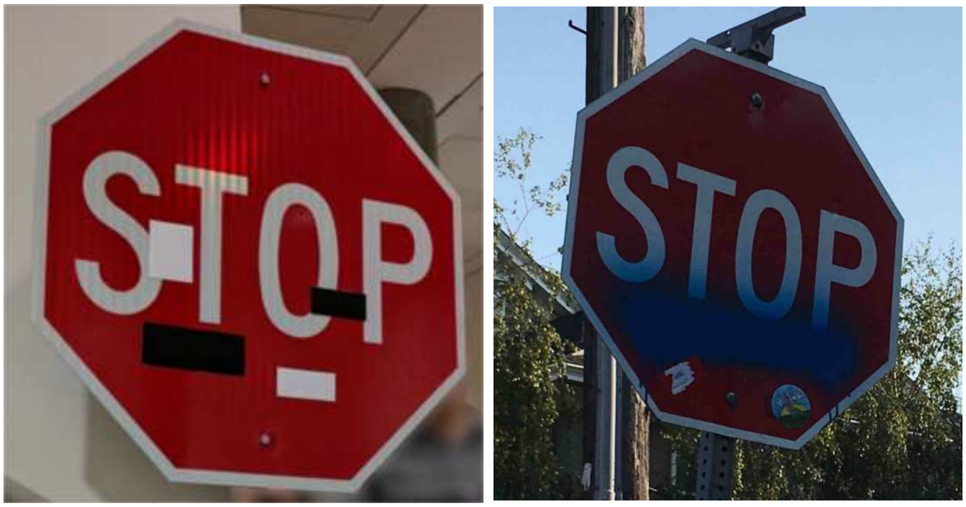

Figure 1.2: Stop signs altered by adversarial perturbations to look innocuous. They appear as 45 mph signs to a DNN trained to recognize traffic signs. Images taken from [2]

process other types of inputs can also be attacked effectively. For example, a model that predicts the sentiment of a review can be fooled by a few changes in the order of the letters, or what a text to speech recognition model understands can be completely changed by a noise imperceptible to our ears (see Figure 1.3). In addition, neural networks are notoriously hard to explain and interpret [27]. Such demonstrations of vulnerability and uninterpretability raise concerns for the reliability of neural networks, especially in security sensitive applications, and pose a significant challenge for the future of the role played by neural networks in our daily lives.

1.2 Tale of Two Paradigms

In order to understand the adverse effects mentioned in Section 1.1 better, we need to take a step back from the topic of adversarial vulnerability, and discuss two approaches in signal and data processing. Before the widespread adoption of machine learning methods, most signal processing was done using domain-specific methods. Researchers would hand-

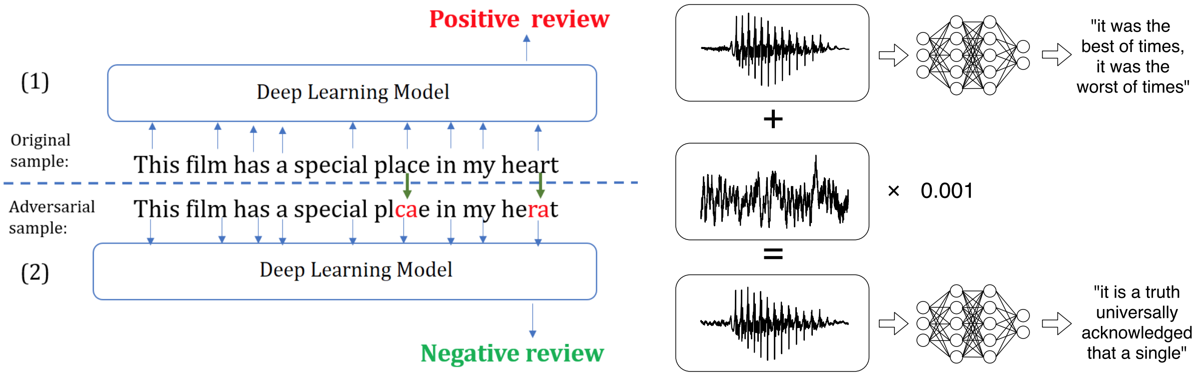
Positive review
(1)
Deep Learning Model
Original sample:
This film has a special place in my heart
Adversarial sample:
This film has a special plcae in my herat
(2)
Deep Learning Model
Negative review
+
× 0.001
=
"it was the best of times, it was the worst of times"
"it is a truth universally acknowledged that a single"

Figure 1.3: Adversarial vulnerability is not limited to image domain. Left: a model that predicts sentiment of reviews can be fooled by a few changes in the order of the letters. Image taken from [3] Right: text to speech recognition model understands can be fooled by a noise imperceptible to our ears. Image taken from [4].

craft specialized solutions for each task at hand, such as Pyramid Histogram of Oriented Gradients (PHOG) in image processing or dynamic time warping in speech recognition. In contrast, the promise of deep learning was to replace all of this manual research labor by top-down, end-to-end trained models that require minimal intervention other than preprocessing (“massaging”) of the data — sometimes even using the same architecture for different tasks. The famous argument asserted by Rich Sutton in [28] is that in this dichotomy of hand-crafted vs end-to-end, heavily data-reliant approaches, the latter wins in the long run — or did so far. Arguably, training extremely large DNNs on such end-to-end cost functions with variants of stochastic gradient descent is one of the key contributors to the recent explosive growth of data processing.

We argue that while the assertion in [28] may be true, it is the exclusive reliance on these top-down methods which results in unintended and dangerous consequences such as susceptibility to adversarial perturbations, lack of interpretability or vulnerability against common corruptions like blur, defocus and weather effects (Figure 1.4). Within the existing training paradigm, the main recourse to remedy these negative effects is more sophisticated top-down training, including modification of the end-to-end cost function and/or adversarial augmentation of the input data. On the other hand, we claim that it may be beneficial or even necessary to incorporate more carefully designed “bottom-up” components and structures into DNNs (instead of exclusive reliance on top-down solutions) to achieve desired outcomes.

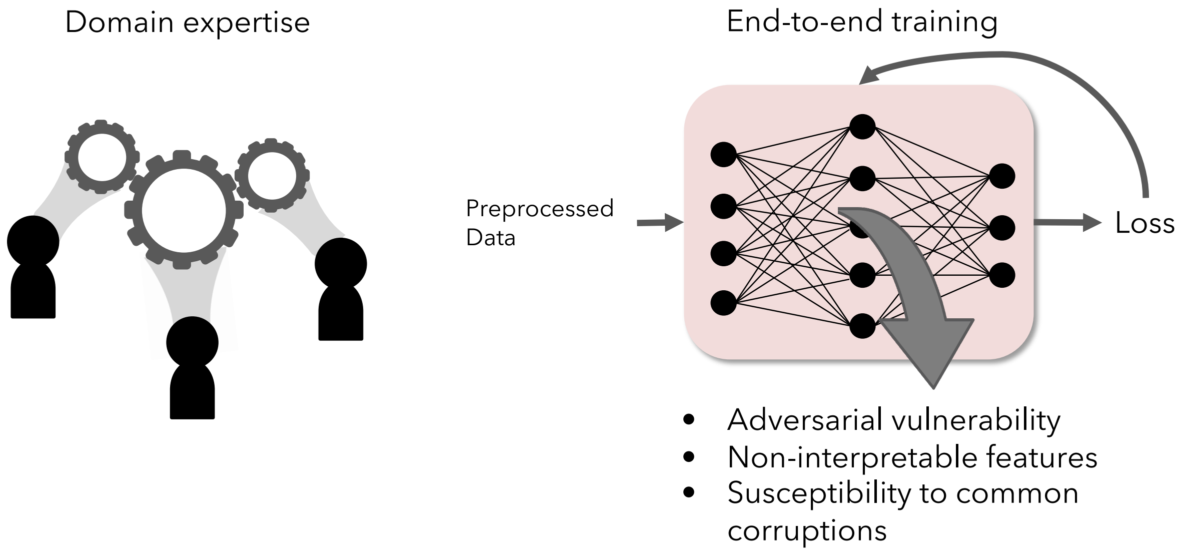
Domain expertise
End-to-end training
Preprocessed Data
Loss
• Adversarial vulnerability
• Non-interpretable features
• Susceptibility to common corruptions

Figure 1.4: Comparison of two approaches in signal processing: hand-crafted versus top-down, end-to-end methods. End-to-end methods provide convenience but over-reliance on them results in unintended adverse effects.

It is important to emphasize that we do not advocate going back to the domain expertise days. Rather, we seek to incorporate some key principles from other areas to build more carefully designed structural elements, integrating them as part of neural networks to alleviate or reduce these unintended ill effects caused by over reliance on end-to-end methods. Our bottom-up elements are inspired by the areas of signal processing, sparse representation theory and neuroscience. The term “bottom-up” refers to structural elements that are designed with a principle in mind, pieced together to give rise to a more complex system — a system which “controls” the output signal representations from those blocks. (Figure 1.5).

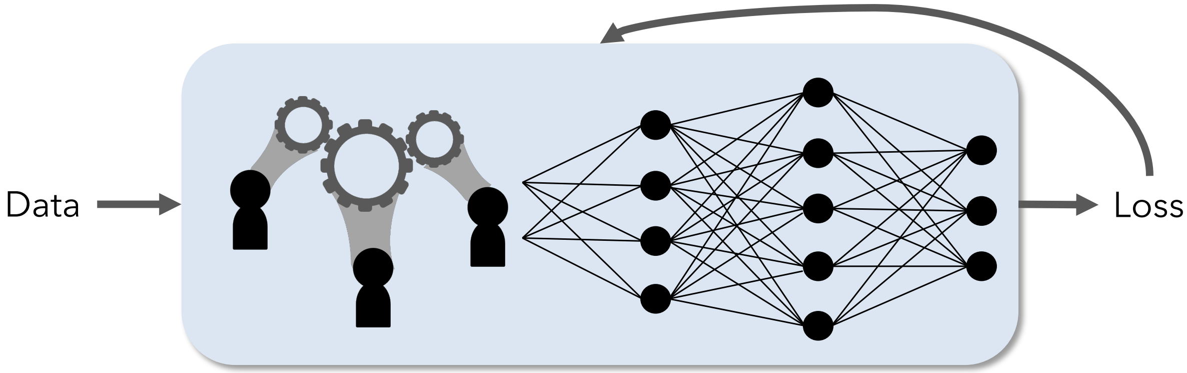

Figure 1.5: Illustration for our approach. We incorporate bottom-up elements inspired by the principles from traditional signal processing, sparse representation theory and neuroscience into DNNs to enhance negative aspects.

1.3 Dissertation Organization

This work proposes principled structural elements and network components that are designed “bottom up” to suppress adversarial perturbations at each step and provide explainable mechanisms while doing so. The organization of the rest of this dissertation is as follows. Chapter 2 gives a brief background into the field of adversarial machine learning and lays out some of the attacks and defenses used to compare with our approaches. Chapter 3 explains the motivation behind some of the design choices we made in our approaches. Chapter 4 describes two front end based approaches. The first uses polarization and quantization to mitigate adversarial attacks, while the second uses techniques and ideas from sparse coding theory and neuroscience. Chapter 5 outlines a more general approach that can be used to complement any layer of a convolutional neural network to produce sparser and stronger activations which could potentially enhance robustness and interpretability. Chapter 6 concludes by discussing high-level ideas and further research directions.

Chapter 2

Background

Since the vulnerability of machine learning models was first pointed out [24, 25], there have been a vast number of research papers written on how to generate these perturbations (adversarial attacks) [7, 9] and how to defend against them [9, 29, 30, 31]. In this cat-and-mouse game, attackers try to identify the smallest potential distortion that can fool a DNN (flipping the prediction in a classification context) or the distortion within some bounds that fools the DNN the best. Defenders, on the other hand, aim to defend DNNs against such threats. The fundamental importance of this contest is well understood by the research community: given the pervasive impact of DNNs, it is crucial to understand and address their vulnerabilities. In this section, we give a brief background on the most commonly used attack types and some of the proposed defenses from the literature that we use to compare our defenses with.

2.1 Adversarial Attacks

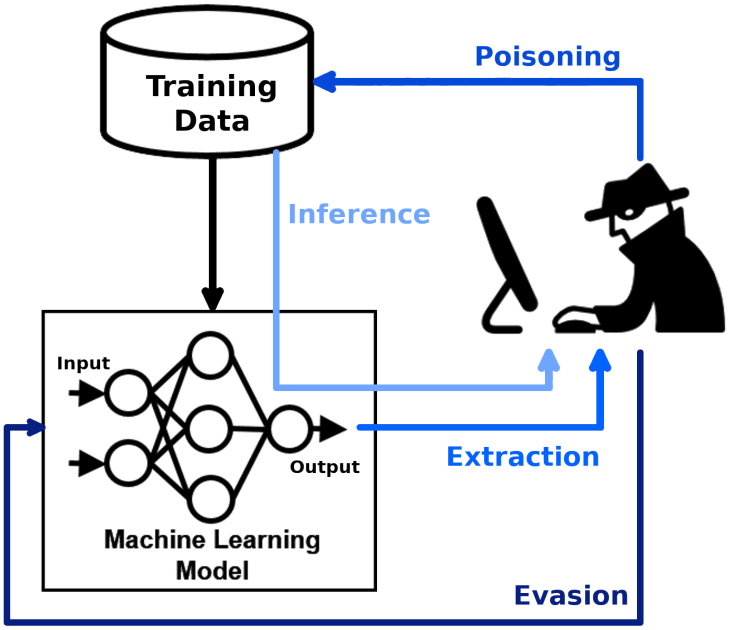

Figure 2.1: There are four threat models in adversarial attacks: poisoning attacks, model extraction attacks, inference attacks, and evasion attacks. Image taken from the GitHub page of [5].

Adversarial attacks can be broadly grouped into four threat models: *poisoning attacks*, in which the adversary mixes bad data into training data to hamper training; *extraction attacks*, in which, the adversary tries to learn about or decode the weights of the network by querying; *inference attacks*, in which, the adversary tries to find out more about the training data; and *evasion attacks*, in which, the attacker tries to fool the network by changing the input (see Figure 2.1). Our focus in this dissertation is evasion attacks. When we use the term “adversarial attacks”, we mean evasion attacks in general.

Evasion attacks are typically grouped into two categories: white-box attacks, in which the attacker has access to both the structure and the parameters of the neural network; and black-box attacks, in which the attacker has access only to the network outputs (scores) or sometimes only the decision (prediction).

Orthogonal to the assumption about access to the network parameters and gradients, are assumptions about the “noticeability” (power) or about the geometry of the attacks. In terms of the geometry of the attacks, a minority of papers consider a specific shape for the attack [2, 6] (see Figures 1.2, 2.2). Most of the research, however, considers attacks that are distributed throughout the input. In terms of the power of the attack, most papers assume an L^p bound on the perturbation, with p most commonly chosen as ∞ because it can be tuned to be imperceptible to humans [7, 26]. An exception to this is exemplified by [32], which uses “perceptual” distance to compute the power of the attack. In this work we mostly study L^∞ bounded attacks, but in Section 4.2 we also test our defense against L^1 and L^2 bounded attacks.

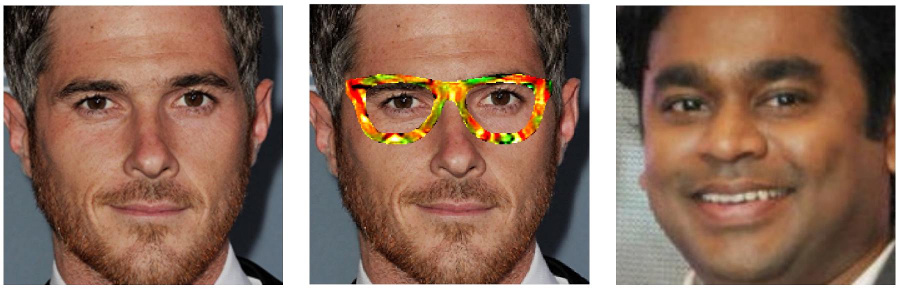

Figure 2.2: Example of an attack restricted to the shape of an eyeglass frame. Left: clean image, Middle: perturbed image, Right: Individual predicted by the recognition model after the attack. Images taken from [6].

2.1.1 White-Box Attacks

Most of the commonly used white-box attacks generate their perturbation by using gradient information. As is well known, during the training of a neural network, the gradient of the loss with respect to network weights is used to update the weights to minimize the loss. In contrast, while generating gradient-based white-box adversarial attacks, the *gradient with respect to the input* is computed, and the input is modified using this to *maximize* the loss in order to fool the neural network (Figure 2.3). While doing this, the attacks are constrained to be “small” or “imperceptible”. In other words, while successfully fooling the DNNs, they are restricted to have no effect on decisions made by humans.

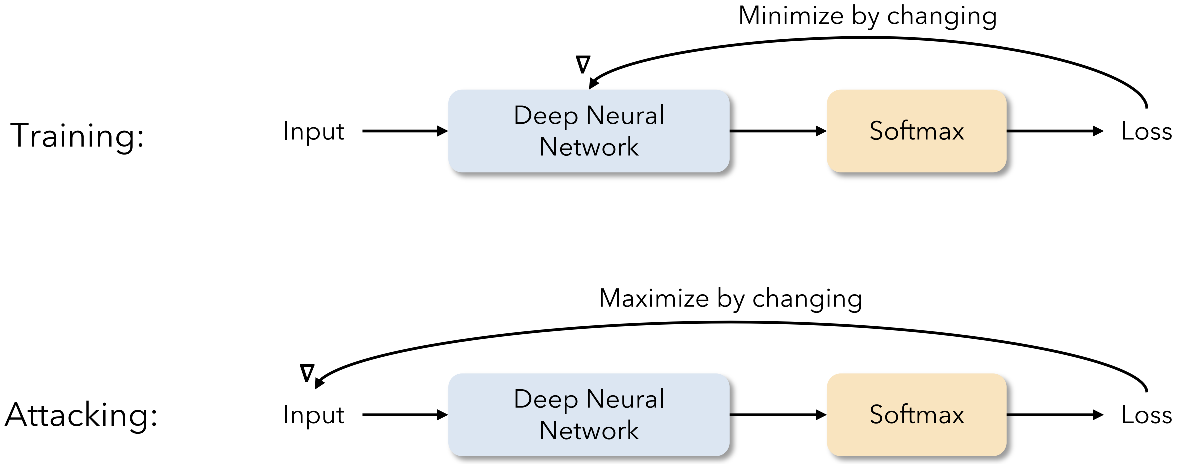

Figure 2.3: White-box attacks are usually computed using the gradient information. Contrary to how training minimizes the loss by changing network parameters using their gradients, attacking tries to maximize the loss by changing the inputs using input gradients.

The formulation of gradient based white-box attacks on classifiers can be written as follows. Given a classifier with parameters $\boldsymbol{\theta}$, $\boldsymbol{f}_{\boldsymbol{\theta}} : \boldsymbol{x} \to \boldsymbol{y}$ that takes in inputs $\boldsymbol{x} \in \mathbb{R}^N$, and outputs predictions (confidence scores for M classes) $\boldsymbol{y} \in [0,1]^M$, the goal of an adversary is to find a perturbation $\boldsymbol{e} \in \mathbb{R}^N$ that maximizes the given loss function $\mathcal{L}$ for classification subject to some constraints:

$$\max_{\boldsymbol{e} \in \mathcal{S}} \mathcal{L}(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x} + \boldsymbol{e}), \boldsymbol{y}_{\text{true}}), \tag{2.1}$$

where $\mathcal{L}$ is a loss function, $\boldsymbol{y}_{\text{true}}$ is the vector of true labels and the constraint is that $\boldsymbol{e}$ is chosen from a set $\mathcal{S}$, which is typically constrained in L^p norm, with $p = \infty$ receiving the greatest attention as mentioned.

The loss $\mathcal{L}$ maximized for the attack is usually taken as the cross-entropy loss between the network outputs $\mathbf{f}(\mathbf{x})$ and label $\mathbf{y}$. This corresponds to an untargeted attack, i.e. the wrong label that is easiest to switch to is sought. However, the attack can also be targeted such that a particular target prediction is promoted [33].

Next, we describe the key L^∞ bounded attacks in the literature, in chronological order of invention.

Fast Gradient Sign Method (FGSM) [7]: the sign of the gradient with respect to the input is taken and scaled by the L^∞ budget ϵ:

$$\boldsymbol{e} = \epsilon \cdot \mathrm{sign}(\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}_{\mathrm{true}})) \tag{2.2}$$

Basic Iterative Method (BIM) [8]: This is an iterative version of FGSM where the overall L^p budget ϵ is divided into smaller steps of size δ. In each step, gradient of the loss function $\mathcal{L}$ with respect to the input is normalized and scaled by δ, and added to the perturbation calculated in the previous step. After each step, the computed attack is projected to conform to the L^p bound. Gradient ascent over the loss in the input space is thus achieved.

$$\boldsymbol{e}_{i+1} = \mathrm{clip}_{\epsilon}\Big[\boldsymbol{e}_i + \delta \cdot \mathrm{sign}(\nabla_{\boldsymbol{e}} \mathcal{L}(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}+\boldsymbol{e}), \boldsymbol{y}_{\mathrm{true}}))\Big] \tag{2.3}$$

where $\mathbf{e}_i$ corresponds to the value of the perturbation at iteration i with $\mathbf{e}_0 = \mathbf{0}$ or $\mathbf{e}_0$ with each element drawn from uniform distribution $\mathcal{U}(-\epsilon, \epsilon)$.

Projected Gradient Descent (PGD): This attack [9] employs BIM with multiple random starting points sampled from a uniform distribution in the ϵ box around the data point. The goal is to optimize over a greater area in the input space because the loss surface being navigated is complicated. It was noted in [9] that BIM is a formulation of Projected Gradient Descent (PGD), a well-known method in convex optimization. Among the many attack methods, Projected Gradient Descent (PGD) is regarded to be among the most effective L^{∞} bounded attacks, and is therefore generally used to evaluate defense methods. Figure 2.4 illustrates these three methods.

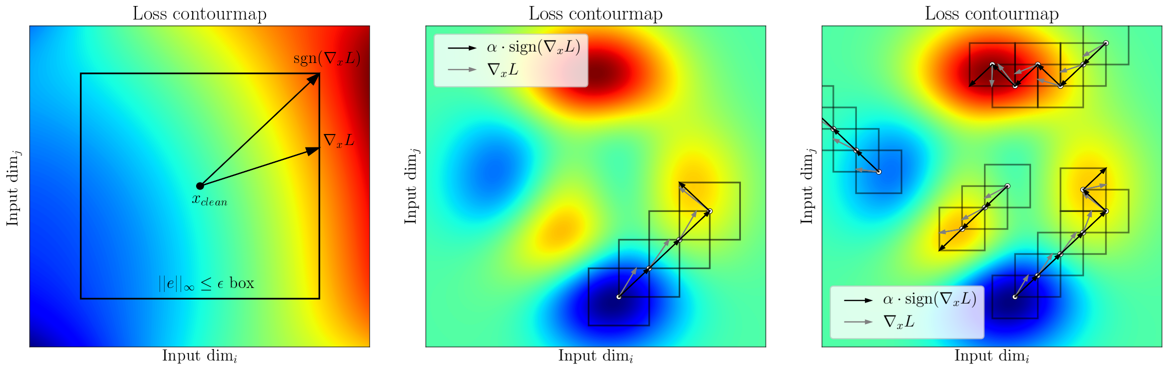

Figure 2.4: Illustration of three different gradient based attacks over the loss surface. Left: FGSM [7], which computes the gradient and projects it onto the L^p ball. Middle: BIM [8], which divides the gradient ascent into smaller steps of size α. Right: PGD [9], which performs the multi-step gradient ascent for different initializations and chooses the best among them.

Carlini-Wagner (C&W) [33] is a modified version of PGD. In this attack, the loss function $\mathcal{L}$ is taken as the difference between the logit that corresponds to the correct label and the maximum of logits among the incorrect labels.

Expectation Over Transformation (EOT) is suggested in [34] to make attacks robust against transformations, and [35] suggests using this method to evaluate defenses utilizing stochasticity. As the name suggests, it computes the mean of gradients over the random transformations that are part of some networks to better approximate the gradients. With EOT, PGD becomes:

$$\boldsymbol{e}_{i+1} = \mathrm{clip}_{\epsilon}\big[\boldsymbol{e}_i + \delta \cdot \mathrm{sign}(\sum_{r=0}^{N_E-1} \nabla_{\mathbf{e}} \mathcal{L}_r(\boldsymbol{f}(\boldsymbol{x} + \boldsymbol{e}_i), \boldsymbol{y}))\big] \tag{2.4}$$

where $\mathbf{e}_0 = \mathbf{0}$ and N_E corresponds to the number of runs used to average the gradients. We use this technique in evaluating our defense that contains randomness (Section 4.2).

AutoAttack [10] is a recent attack that improves some of the shortcomings of PGD by automating the choice of the step size hyperparameter by adaptively changing it using optimization trends. This circumvents a bottleneck in attack hyperparameter selection. It also replaces the cross-entropy loss with a shift and rescaling invariant loss to avoid vanishing gradients and gradient masking problems encountered in defense evaluation (see Section 2.2.3).

2.1.2 Black-Box Attacks

Black-box attacks are gradient-free, not relying on the knowledge of weights or gradients. There are three main types of black-box attacks. Score-based black-box attacks assume access to the softmax outputs, i.e. scores [11, 36], while decision-based black-box attacks assume access to the decision of the model (argmax of scores) [37, 38]. The third type of black-box attacks assumes access to the training dataset only. Such attacks typically use a surrogate model trained on the same dataset in order to generate attacks [39, 40] and apply (transfer) it to the actual network in question.

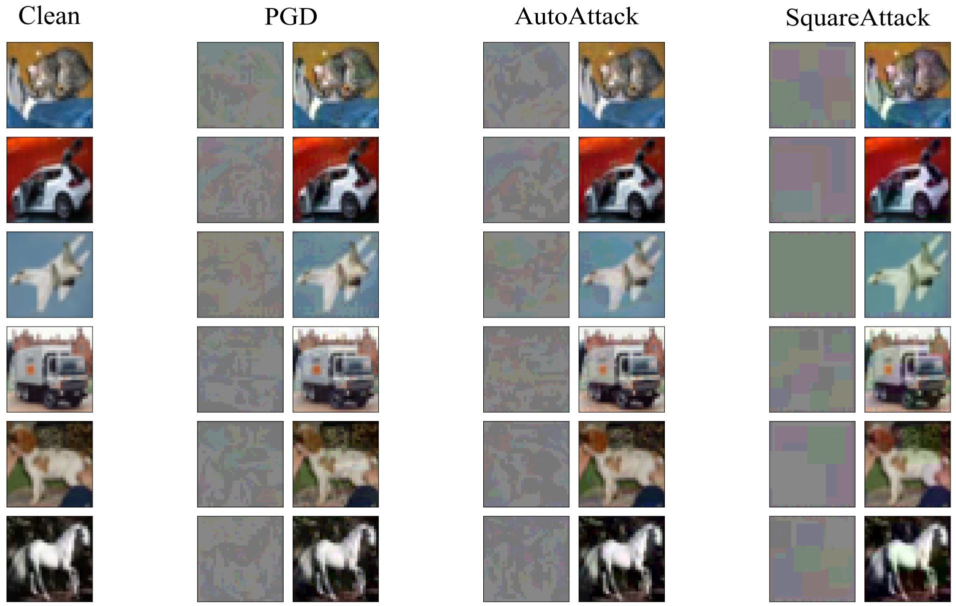

Figure 2.5: Illustration of three different attack types (PGD [9], AutoAttack [10], and SquareAttack [11]) and the corresponding perturbed images. Attacked images are not easily recognized by humans.

Figure 2.5 demonstrates some of the white-box and black-box attacks and the images perturbed with them.

2.2 Defenses

Since the vulnerability of neural networks to adversarial examples was first pointed out [24, 25], there has been significant research effort in developing more sophisticated defenses. Over the years many defense mechanisms have been proposed, only to be defeated by stronger adversaries [41, 42]. Nowadays most of the state-of-the-art defenses use either extra unlabeled data [43] that is not present in the original dataset or an extreme amount of data augmentation, both in conjunction with *adversarial training*.

2.2.1 Adversarial Training and Variants

In adversarial training, we train the neural network with attacked inputs generated during training. In each training step, adversarial attacks for the network at that stage are calculated, inputs are perturbed and fed into the network to train it. Some of the more recent successful defenses that use adversarial training include [44], which uses 2000-fold synthetic data augmentation of the training dataset combined with adversarial training, [45], which uses 80 million additional images from TinyImages dataset and other data augmentation techniques, [46], which combines adversarial training with parameterizable activation functions and 6 million synthetic images, and [47], which uses a modified cost function aiming to trade off clean and adversarial accuracy without any augmentations or extra data.

Even though it is used in state-of-the-art defenses, adversarial training is not without disadvantages. First, it is computationally intensive: computing as many gradients as attack steps in each training step considerably slows down the training, by at least an order of magnitude. An even bigger issue is that adversarial training, like any top-down method, results in a black box. It is unclear how the model is made more robust or what exactly changed in the weights of the network to provide the enhanced robustness. Therefore it is very difficult to have an intuitive understanding of why and how perturbations are being controlled as they flow up the network. This opaqueness and lack of insight makes it challenging to improve on the adversarial robustness without ad hoc methods like augmentations or additional data. As a result, most of the improvements in the past few years are achieved through combining adversarial training with other approaches, rather than by improving the adversarial training itself. Further, there is still a significant robust generalization gap between clean and adversarial accuracies for adversarially trained networks, showing that the perturbation is not completely rejected by the adversarially trained network. While the last issue is not unique to adversarial training, it does show that it is not a "silver bullet" against adversarial vulnerability, and there remain fundamental security concerns about DNNs.

2.2.2 Other Defenses

Provably robust defenses have also been studied extensively [48, 49, 50]. These methods provide a lower bound for adversarial accuracies; however, guarantees are provided mostly for small datasets, models, and low attack budgets. [51, 52, 53] report certified robustness for L^2 bounded attacks that is able to scale to larger datasets such as ImageNet. Unfortunately, these certified defenses do not perform as well as adversarial training against current attack methods.

There are also bio-inspired defenses. [54] has a network structure similar to the V1 cortex of the primate brain in the first few layers of CNNs. This aids the networks in increasing adversarial robustness with adversarial training. Their approach is distinct from our neuro-inspired approach that is described in Chapter 5 in that their defense uses Gabor type filters and injects noise adapted from observations from physiology.

Many other sophisticated defense mechanisms have been proposed employing detection techniques [55, 56, 57, 58], preprocessing methods [29, 30, 59], modification of the optimization objective [56, 60, 61], and biological constraints [54, 62, 63, 64]. However, these methods have either eventually been beaten by an adaptive attack [41, 42, 65] or do not match the empirical success of adversarial training. Adversarial training and its variants therefore continue to be the state-of-the-art defense against adversarial attacks.

2.2.3 Defense Evaluation and Gradient Masking:

The use of non-differentiable functions or functions with a saturation region can cause state-of-the-art gradient-based attacks to falter. However, defenses that rely on such "gradient masking" are not robust: they are easily circumvented by the attacker, as shown by Athalye et al. [42], by replacing the non-differentiable function with a differentiable approximation. Indeed, our own front end based defenses in Section 4.1 and Section 4.2 employ non-differentiable functions. We therefore test these defenses using the gradient approximation methods of [42], replacing non-differentiable functions with their differentiable approximations (or identity function if it makes the attack stronger) in the gradient calculations.

Chapter 3

Analysis of Adversarial Training

In view of its success, adversarial training is naturally under the spotlight for researchers looking to improve the state-of-the-art and/or looking for an answer to how the learned model differs from a model trained in a standard fashion. Previous works in the literature attempt to analyze adversarial training in terms of loss landscape [66], decision boundary [67], class-wise robustness [68], smoothness [69] and algorithmic stability [70]. However, prior works do not provide structural insight into the contribution of different layers in the network towards robustness. Nor do they investigate what statistical differences exist between the weights of adversarially trained networks and those of naturally trained networks. In this chapter, we aim to fill this gap and analyze adversarially trained networks in these two regards to obtain detailed structural insights and distill principles to guide us in designing bottom-up techniques.

3.1 Layers' Role in Adversarial Robustness

3.1.1 Introduction

In this section, we seek to quantify the importance of each layer in handling adversarial perturbations by comparing adversarial training versus natural training on a layer-wise basis. Our goal is to develop architectural insights to guide further research into the structural properties of robust neural networks. We demonstrate that earlier layers play a crucial role in defending against adversarial perturbations. We also provide a novel approach for inspecting how adversarial perturbations flow through the layers of the network, thus providing a fine-grained analysis of a model's robustness. We report experiments on CIFAR-10 image classification dataset using two popular architectures, namely, VGG and ResNet.

Our contributions are:

- We develop a partial adversarial training method in order to quantify the role of each layer in providing robustness against adversarial attacks.
- We introduce the concept of "perturbation-to-signal ratio" (PSR), defined in terms of different L^p norms, to track the flow of adversarial perturbations through the network.
- Using these two techniques we conclude that earlier layers play a crucial role in adversarial robustness.

The code repository associated with this section can be found at https://github.com/canbakiskan/layersnotequalinAT

3.1.2 Setup and Definitions

We now describe the methodology that was used to evaluate and understand the importance of each layer.

3.1.2.1 Partial Adversarial Training

Since our goal is to analyze the role of each layer of the network in adversarial training, we found the most straightforward method to be restricting the contribution of each layer to the adversarial training process. We achieve this by a two step process (see Figures 3.1 through 3.5). In the first training stage, we train the whole network adversarially or naturally. Then, we freeze either the earlier or the latter part of the network and reinitialize the remaining unfrozen layers. In the second training stage, we train the unfrozen layers with the other training method from scratch. To keep track of and refer to various experiments of these two kinds, we employ a 4 character code. The first two characters of the code refer to how the earlier layers are trained and whether they are trained in the first or second stage; and the last two characters refer to how the latter layers are trained and in which stage. "A" means adversarial and "N" means natural training. "1" means that part of the network was trained first, then frozen, whereas "2" means that part of the network was reinitialized and retrained a second time. Putting it all together, an experiment with the code "A1N2" is where we train all layers in the network adversarially first, then freeze the earlier layers and reinitialize the latter layers, and retrain the latter layers naturally. An experiment with the code "A2N1" means we train all the layers in the network naturally first. Following that, we freeze the latter layers and reinitialize the earlier layers. Then, we retrain the earlier layers adversarially. We use this encoding scheme to refer to specific instances of partial adversarial training throughout the section. For completeness, we train all combinations of partial adversarial training (A1N2, N1A2, A2N1, N2A1).

We also experimented with retraining the reinitialized layers both adversarially and naturally in parallel and taking the differences between them (e.g. N1A2–N1N2 or N2A1–A2A1). Doing this removes the confounding effects caused by the reinitialization of parameters and the mismatch of the optimizer state after the reinitialization, from the actual effect of partial adversarial training. However, we omit this process in this section because we observed that the aforementioned effects were minimal and this process made the reporting unnecessarily more convoluted.

In order to reduce the training time of the substantial number of different experiments, we opted for a smaller ResNet [71] and VGG [72] models with 16 units. The number of parameters is 11.2 million for ResNet and 14.7 million VGG.

We should mention that since ResNet has skip connections, it is not entirely linear. However, in order to plot the adversarial losses in a two dimensional plot, we need to use a linear list of layers. While linearizing the layers of ResNet, we chose to place the shortcut layers after the main branch layers that it's summed with, but the placement of these shortcut layers is somewhat arbitrary.

Another important point to make here is that the distributional shift in layer inputs caused by adversarial training necessitates the running mean and running variance (non-trainable parameters) of batch norm layers be kept unfrozen after the reinitialization. To be specific, freezing and reinitializing works in a straightforward way for trainable parameters (both convolutional and batch norm layers). For a batch norm layer whose trainable parameters *are not frozen* (therefore they are reinitialized), non-trainable statistics (i.e. running means and variances) are also *not frozen* and reinitialized. On the other hand, for a batch norm layer whose trainable parameters *are frozen*, the non-trainable statistics *are not frozen*, but not reinitialized either. This is done because freezing the running statistics would significantly hamper the training due to the different distributions of batch norm layer inputs corresponding to clean and adversarial images.

Retraining Latter Layers The first type of partial training experiment we conduct is where we freeze the earlier layers, then reinitialize and retrain the latter layers. This type of training is the most natural of the three partial adversarial training setups because it resembles the regular procedure followed in transfer learning.

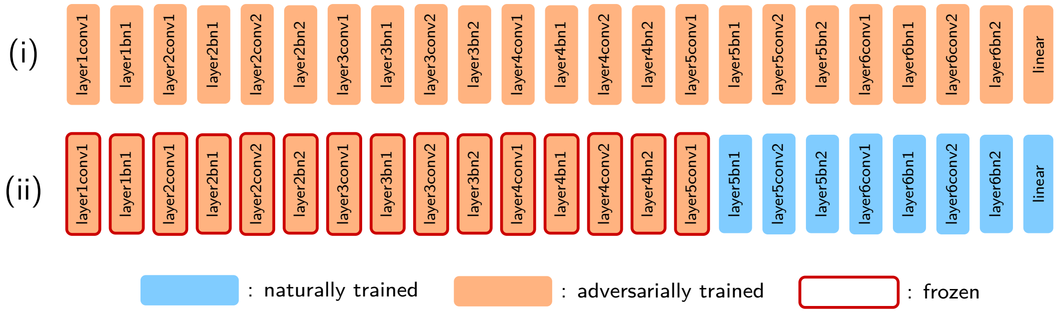

Figure 3.1: Visualization of (i) initial adversarial training, (ii) freezing of the early (adversarially trained) layers, and reinitialization and natural retraining of the latter layers for the example scenario with the code A1N2. Cutoff point is before block 5 convolutional layer 1. Note that skip connections have been omitted and network is shortened for illustration purposes.

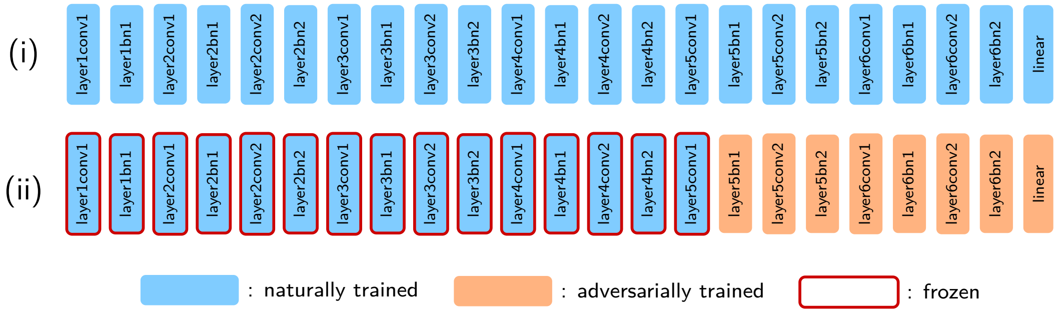

Figure 3.2: Visualization of (i) initial natural training, (ii) freezing of the early (naturally trained) layers, and reinitialization and adversarial retraining of the latter layers for the example scenario with the code N1A2. Cutoff point is before block 5 convolutional layer 1. Note that skip connections have been omitted and network is shortened for illustration purposes.

Retraining Earlier Layers The second type of partial training experiment is where we freeze the latter layers, then reinitialize and retrain the earlier layers.

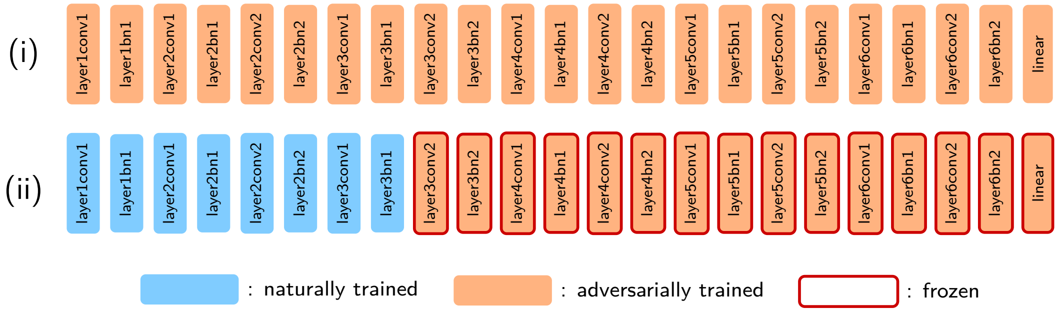

Figure 3.3: Visualization of (i) initial adversarial training, (ii) freezing of the latter (adversarially trained) layers, and reinitialization and natural retraining of the early layers for the example scenario with the code N2A1. Cutoff point is before block 3 convolutional layer 2. Note that skip connections have been omitted and network is shortened for illustration purposes.

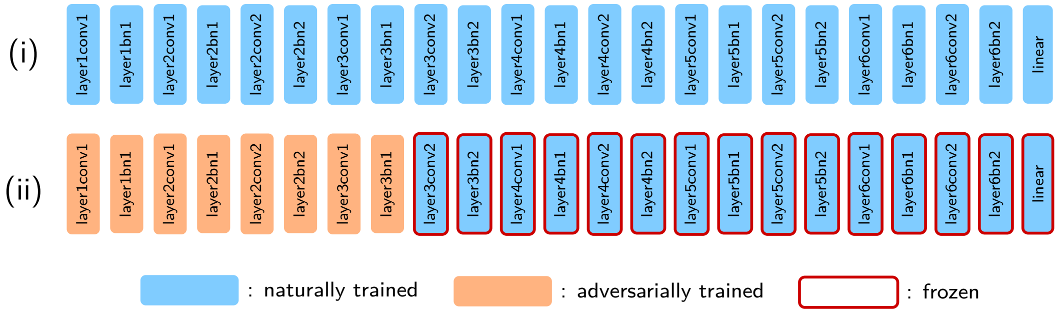

Figure 3.4: Visualization of (i) initial natural training, (ii) freezing of the latter (naturally trained) layers, and reinitialization and adversarial retraining of the early layers for the example scenario with the code A2N1. Cutoff point is before block 3 convolutional layer 2. Note that skip connections have been omitted and network is shortened for illustration purposes.

Retraining A Single Layer Similar to the previous two techniques, one can also freeze all but one layer, and then reinitialize and retrain that single layer. This makes a small but noticeable change in the robustness of the neural network.

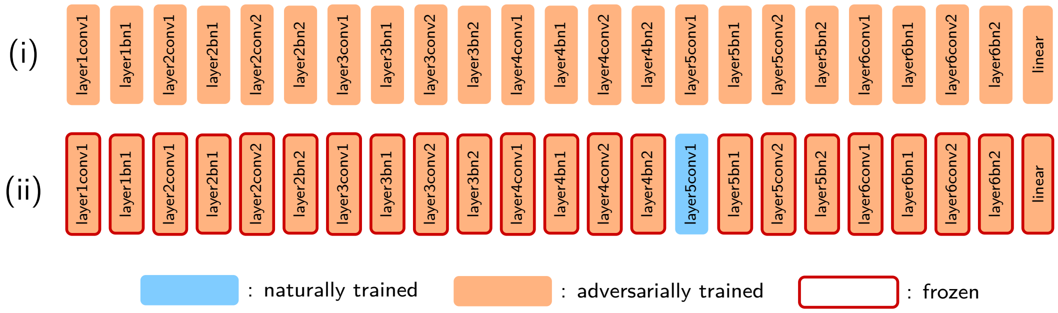

Figure 3.5: Visualization of (i) initial adversarial training, (ii) freezing of all layers except one, and reinitialization and natural retraining of the single layer. The single natural layer is block 5 convolutional layer 1. Note that skip connections have been omitted and network is shortened for illustration purposes.

3.1.2.2 Tracking Perturbations Across Layers

Separate from partial adversarial training, we also use another method to visualize how different layers operate on the adversarial perturbations by plotting the L^p norm of the difference between layer outputs for attacked images and layer outputs for clean images. We divide this by the L^p norm of the layer outputs for clean images to obtain perturbation-to-signal-ratio (Equation 3.1). This serves two purposes: we get a better measure of the strength of the attack at that layer with respect to the layer outputs for clean images, and the effect of number of dimensions on the L^p norm is eliminated since both the numerator and denominator have it. While L^p norms do not capture the full complexity of the adversarial perturbations, they do equip us with an approximate measure of remaining power of the perturbation at a given layer.

$$\mathrm{PSR}_l = \frac{\left\|\boldsymbol{f}_l(\boldsymbol{x}) - \boldsymbol{f}_l(\boldsymbol{x}+\boldsymbol{e})\right\|_p}{\left\|\boldsymbol{f}_l(\boldsymbol{x})\right\|_p} \tag{3.1}$$

3.1.3 Experiments & Results

At first glance, accuracy might seem like a reasonable statistic to measure the effectiveness of layers in adversarial training with. However, when we look at the plots for the accuracy, (Figure 3.6) we notice a shortcoming of using accuracy as a performance measure. Namely, both of the plots show a saturation of accuracy at the beginning. The saturation at the beginning means that even if the first quarter of the network (for ResNet) is adversarially trained, there is no observed gain in robustness. This is somewhat counterintuitive, and indeed, when we look at the same plots for cross-entropy loss, which is what the adversarial attack tries to maximize in the first place, we see a different picture. The first layers certainly do increase robustness, in the form of reduced false-confidence in the wrong label. To give a concrete example, for a hypothetical two label task, both $[0.51, 0.49]$ and $[0.99, 0.01]$ softmax outputs give 0% accuracy against label of $[0, 1]$. However, clearly the second softmax output is much worse. Therefore, when comparing points with accuracies close to zero, the better approach to measure effectiveness is to use the cross-entropy loss itself. If, on the other hand, accuracies are not close to zero, then accuracy can also be a viable metric.

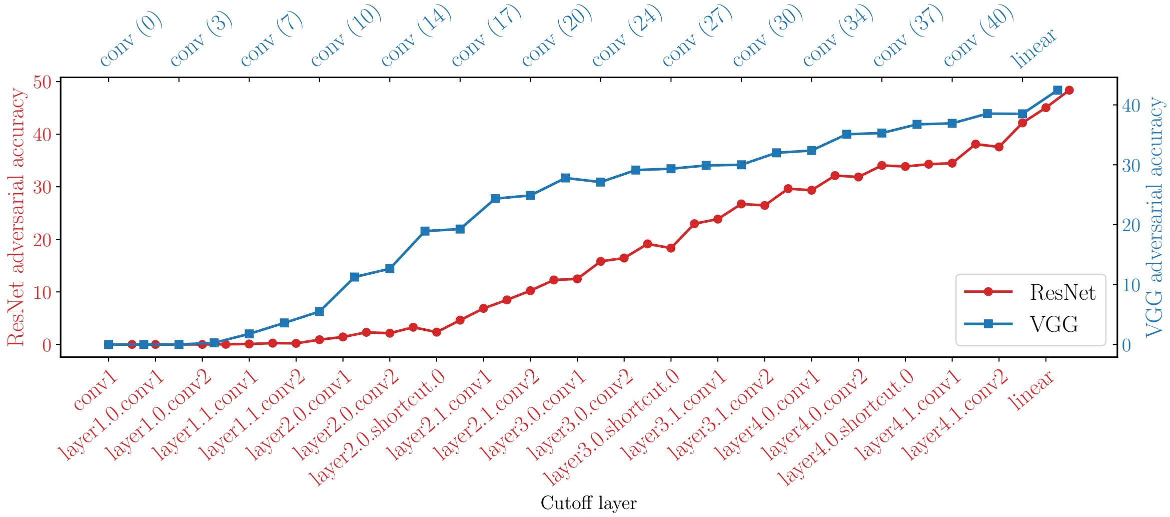

Figure 3.6: Adversarial **accuracy** for the case A1N2 as the number of adversarially trained layers increases. Note the plateau at the beginning.

3.1.3.1 Details

To reduce the amount of hyperparameter search and hand-tailoring, and for consistency, we use the same optimizer, scheduler and optimization hyperparameters for all experiments of the same model. We train ResNet-16 with momentum SGD optimizer with learning rate 0.1 for 75 epochs and 0.01 for a further 5 epochs. We use a momentum coefficient of 0.9, a weight decay coefficient of 2×10^{-4} and batch size of 128. We start training VGG-16 with Adam optimizer with learning rate 1×10^{-3} for 100 epochs. We scale down the learning rate by a factor of 10 at epochs 50 and 75. We do not use weight decay for VGG and use a batch size of 128. The choice of using different optimizers for different models was made to ensure diversity and reduce the likelihood of observations being caused by the choice of a certain optimizer/hyperparameter. The hyperparameters for the SGD optimizer were taken from [9] and the hyperparameters for Adam were determined after a hyperparameter search in a limited number of combinations. For ResNet training we stop early after reducing the learning rate to reduce the chance of catastrophic overfitting [73]. We do not use this for VGG, to reduce the reliance of observations on a particular method, as mentioned before. Following [9], we use L^{∞} bounded PGD attack with attack budget $\epsilon = 8/255$, step size $2/255$, and 10 steps for both of the models' adversarial training. Perturbations are initialized with each element drawn from uniform distribution $\mathcal{U}(-\epsilon, \epsilon)$. For adversarial testing, we use L^{∞} bounded PGD attack with attack budget $\epsilon = 8/255$, step size $1/255$, and 100 steps. Perturbations are initialized with each element drawn from uniform distribution $\mathcal{U}(-\epsilon, \epsilon)$. Total training time is 15 days for ResNet experiments and 10 days for VGG experiments, running on an in-house cluster with 4 Nvidia GTX 1080Ti GPUs.

3.1.3.2 Retraining Latter Layers

We observe that in Figure 3.7 as the number of adversarially trained layers increases from zero to number of layers in the network, the adversarial loss goes from *all natural* model's loss value to *all adversarial* model's loss value, as expected. What is striking is how this transition occurs. The evolution of adversarial loss very closely follows an exponential function for both of the models. This is the first indication that the first few layers play a major role in reducing the adversarial loss.

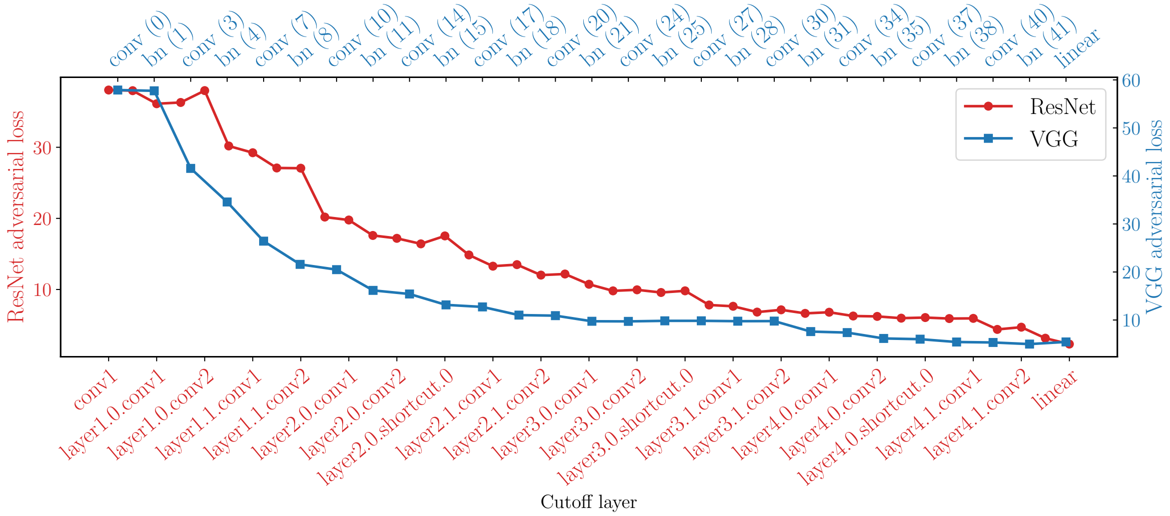

Figure 3.7: Adversarial **loss** for the case of A1N2. The number of adversarially trained layers increases to the right. Higher loss means model is performing worse.

Another key observation we made is that if the earlier layers are trained naturally in the first stage, frozen and the rest is adversarially trained (N1A2), the network has a more difficult time converging. In fact, after some cutoff point (layer3.0.bn2 for ResNet and layer 10 for VGG) the network cannot learn and collapses to random guessing (see Figure 3.8). A likely explanation is that the perturbations grow to such high levels in the earlier layers that the latter layers do not have the capacity to learn from the distributions they see.

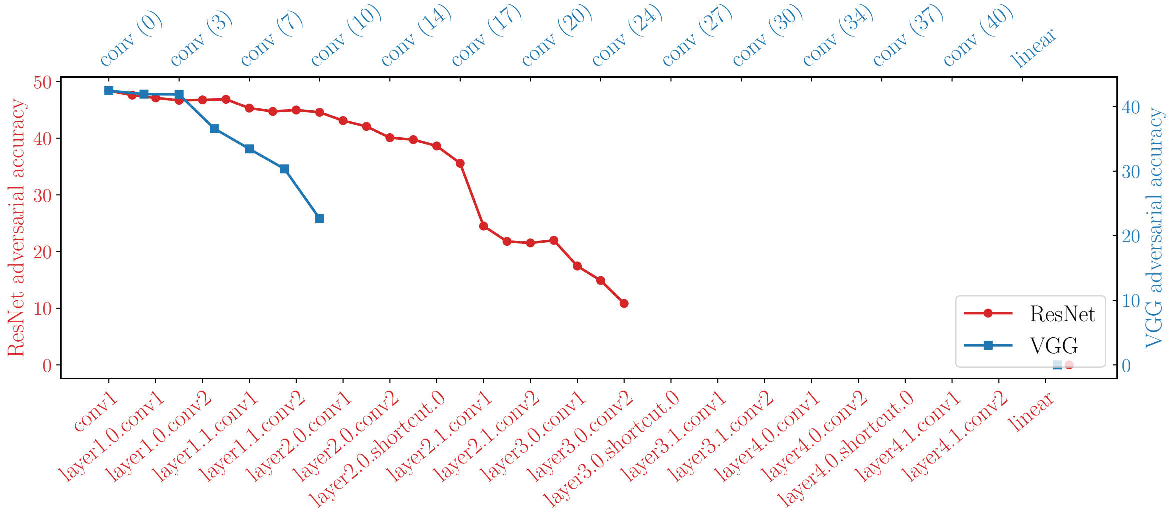

Figure 3.8: Adversarial **accuracy** for the case of N1A2. The number of adversarially trained layers decreases to the right. After a particular cutoff point, the networks cannot learn from adversarial training and collapse to random guessing. Such data points are omitted from the plot.

3.1.3.3 Retraining Earlier Layers

In Figure 3.9, similar to what we observe in latter layer retraining, we see a change in adversarial loss from *all adversarial* model's loss value to *all natural* model's loss value as the number of adversarially trained layers decreases. The transition for ResNet again follows an exponential curve wheres the the transition for VGG follows a non-monotonic trajectory. It is unclear what causes this behavior and why it is only observed when earlier layers are retrained.

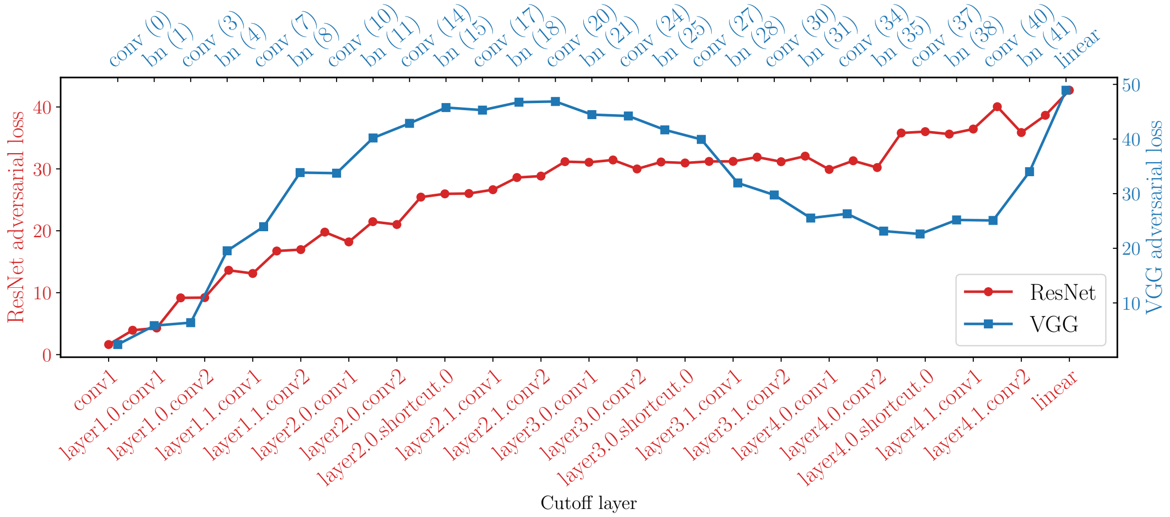

Figure 3.9: Adversarial **loss** for the case of N2A1. The number of adversarially trained layers decreases to the right. Higher loss means model is performing worse.

In Figure 3.10, when the early layers are retrained adversarially, we observe a very rapid and significant decrease in loss, to the level of *all adversarial* model's loss value, with as few as 4 adversarial layers.

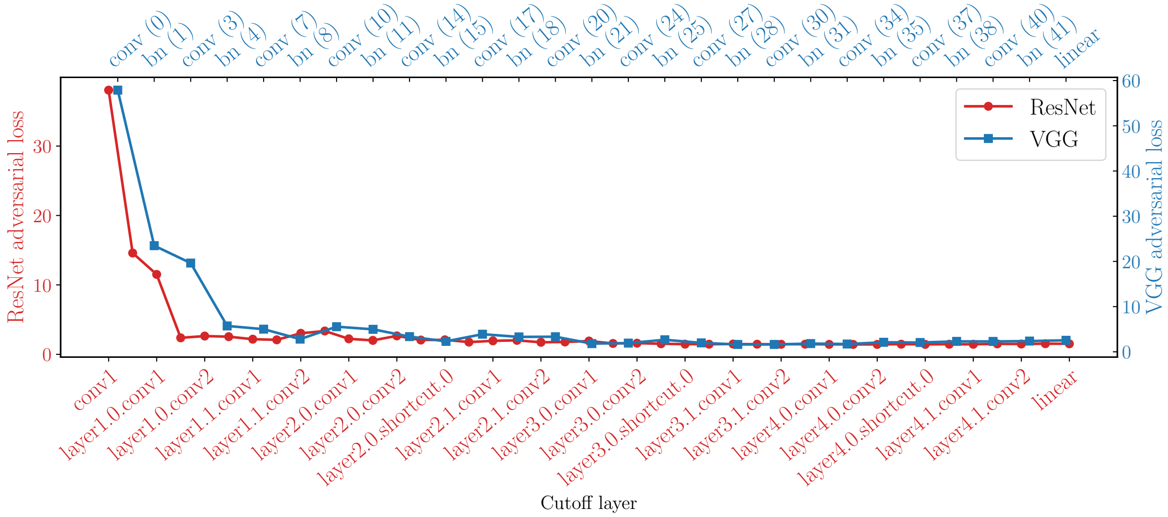

Figure 3.10: Adversarial **loss** for the case of A2N1. The number of adversarially trained layers increases to the right. Higher loss means model is performing worse.

3.1.3.4 Retraining A Single Layer

When we freeze the whole network and retrain a single layer (Figure 3.11), we see a similar trend in the change in statistics as in Sections 3.1.3.2 and 3.1.3.3. Specifically, the change in accuracy induced by each individual early layer is much greater than the change induced by the individual latter layers. Interestingly, these effects of individual layers follow a more linear trend. In this experiment, we can safely look at the accuracies since changing only a single layer does not bring the adversarial accuracy close to zero, where the aforementioned saturation occurs.

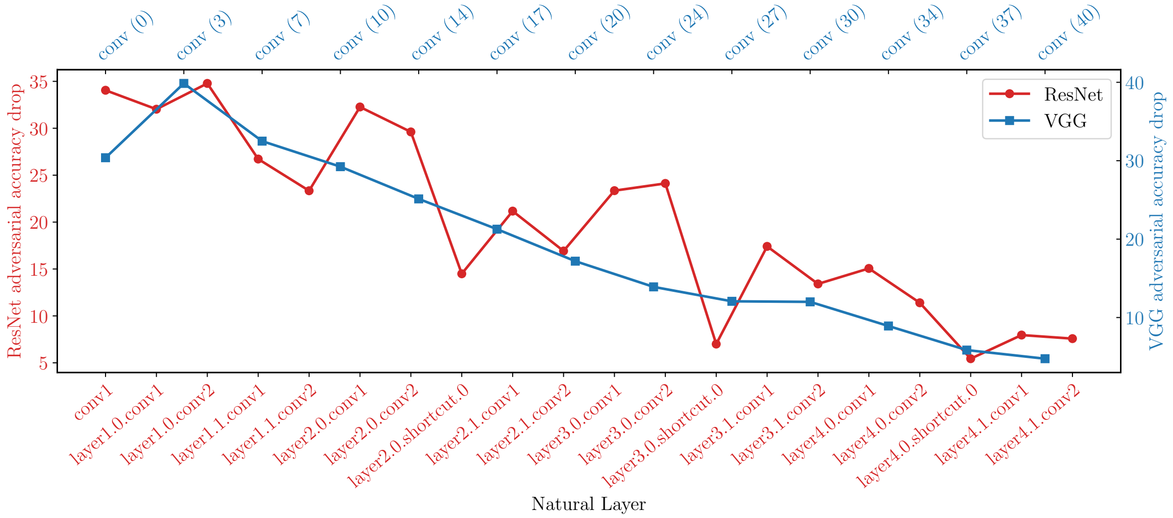

Figure 3.11: The difference in adversarial **accuracy** between an all adversarially trained network and network with a single natural layer, indicating the singular contribution of that layer to adversarial robustness.

3.1.3.5 Tracking Perturbations Across Layers

The perturbation-to-signal ratio (Equation 3.1) indicates the sustained power of the adversarial perturbation at each layer. Figure 3.12 shows this ratio computed for each image and the resulting ratio distributions.

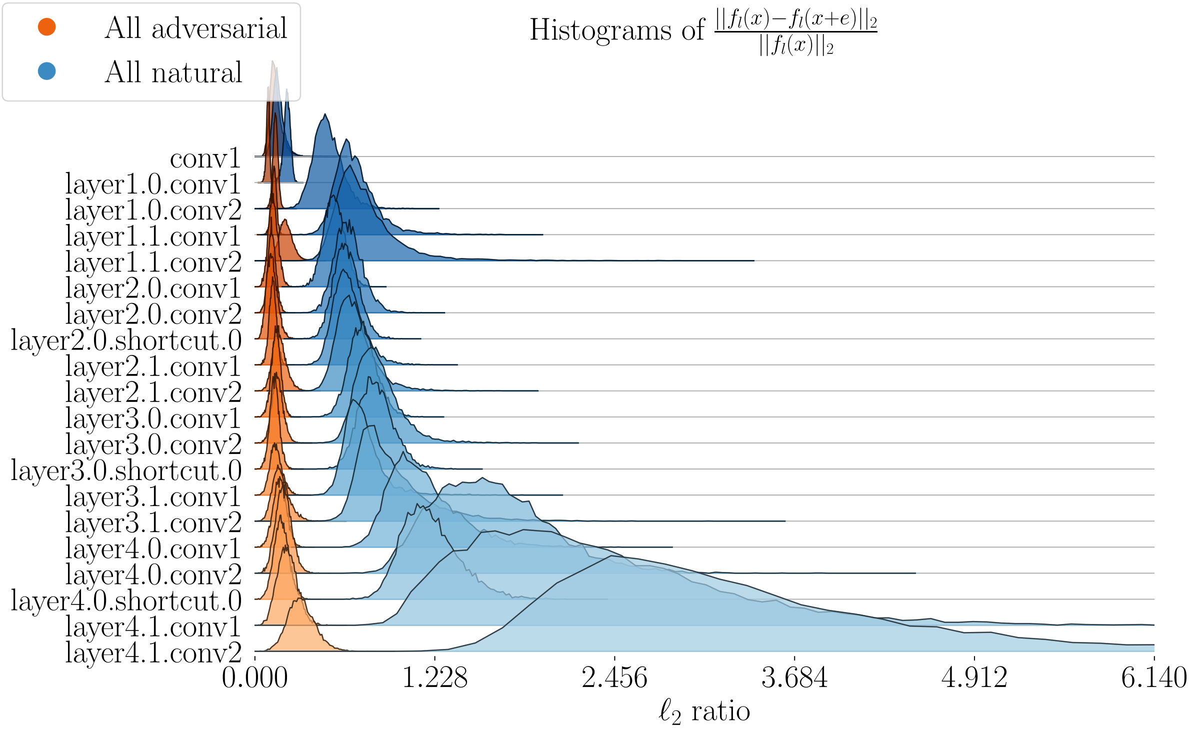

Figure 3.12: Distributions of the ratio of adversarial perturbations' L^2 norm to signal's L^2 norm after each convolutional layer. Histogram is created by computing this value for each image. The layer names are on the y-axis and histogram bin values are on the x-axis.

Figure 3.13 shows the means of these distributions converted to dB. This value is, in essence, the negative of the signal-to-noise ratio (SNR dB) widely used in the signal processing field, if we view the perturbation as noise.

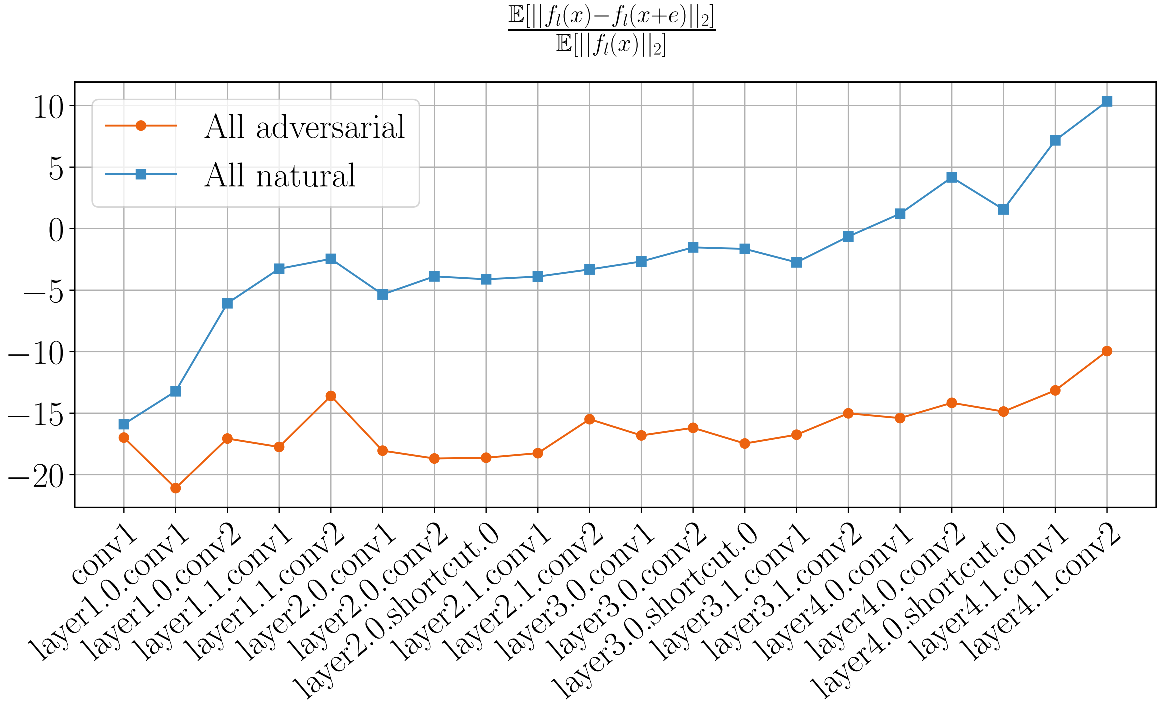

Figure 3.13: The means of the distributions of perturbation-to-signal ratio from Figure 3.12, expressed in dB.

3.1.4 Discussion

Our partial adversarial training approach clearly reveals that earlier layers play a larger role in providing robustness than the latter layers. This is demonstrated both by our evaluations of accuracy under adversarial attacks and by the trends in the loss function. An intuitive explanation is that, unless adversarial perturbations are attenuated early on, they can have ever increasing impacts as they flow through the network, since DNNs are sequential models. This intuition is borne out by our comparisons of perturbation-to-signal ratio between adversarially trained and naturally trained networks: we see from Figure 3.12 how the perturbation expands in size as it flows through the layers in the naturally trained network, and how it is kept under check with adversarial training. Furthermore, Figure 3.13 shows that the PSR difference increases rapidly in the earlier layers, and continues to grow but at a slower pace.

This observation is made even starker if we consider the number of parameters in each layer. Since the latter layers have many more channels than early layers but have the same number of spatial dimensions in the kernels, they have more parameters. Therefore the difference in contribution to adversarial robustness *per parameter* is even greater than the nominal difference between early and latter layers.

Yet another piece of evidence regarding the importance of the early layers is an experiment in which we freeze the early layers after *natural* training, and then try to train latter layers adversarially. We find that, if enough early layers are frozen after natural training, then adversarial training simply does not converge: this is because the perturbation is allowed to increase in size too much as it flows through the naturally trained layers.

3.1.4.1 Limitations

Due to the sheer number of experiments that were executed to obtain these results, they were only run once and there are no known error bars associated with the experimental results. Therefore there is some statistical variation in the results which is apparent in the minor roughness that is evident in some of the graphs. It also means that if the experiments were conducted again, slightly different results would be obtained. However, since the adjustment of cutoff points is very granular (one layer at a time), this statistical variation should not significantly affect the conclusions from this work.

3.2 Statistical Differences in Parameters

Along with the analysis to determine which layers are more important for adversarial robustness, we analyze the parameters themselves. We compare various statistics of layers' parameters of adversarially trained networks and naturally trained networks to see if there are clear structural differences that we can extract and use in our structural defenses.

3.2.1 Statistics of Batch Norm Layers

As reported by Frankle et al. [74], batch norm layers pack a great deal of capacity and expressivity, even when used in conjunction with random weights. So we first compare the parameters (both trainable and non-trainable) of batch norm layers of adversarially and naturally trained ResNets. Figures 3.14, 3.15, 3.16, and 3.17, plot the histograms of running means, running variances, γ (scale), and β (bias) for each batch norm layer of these networks. We can observe differences in some of these statistics. Namely, running variance distributions of some middle layers and the γ and β distributions of the latter layers differ between adversarially and naturally trained networks. However, these trends are not consistent across layers, nor do they give clear insights.

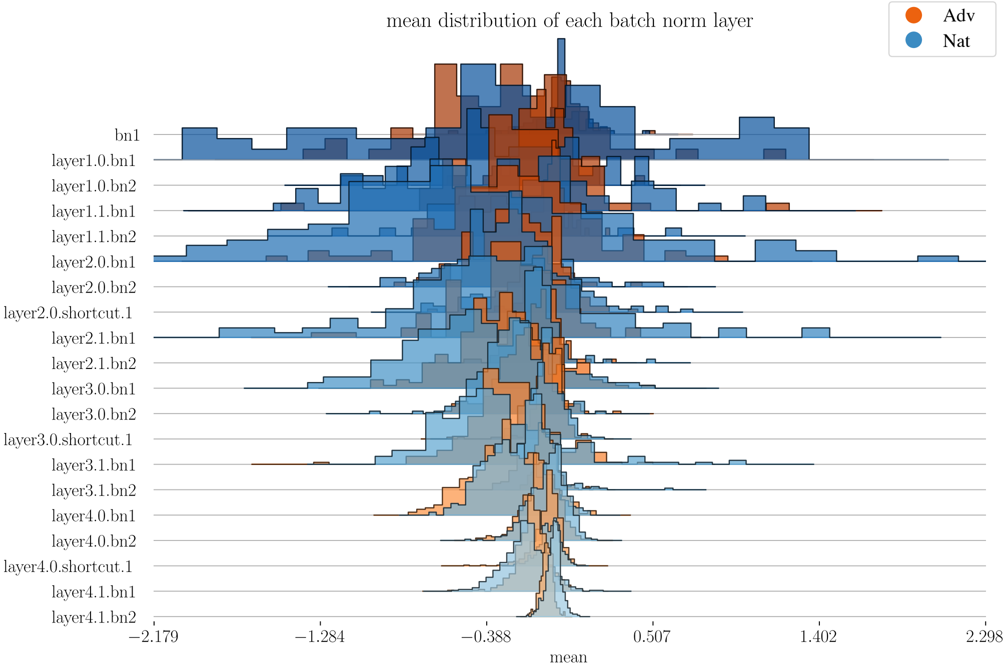

Figure 3.14: Histograms of running means in each batch norm layer in **ResNet**. Orange histograms are for the adversarially trained network and blue histograms are for the naturally trained network.

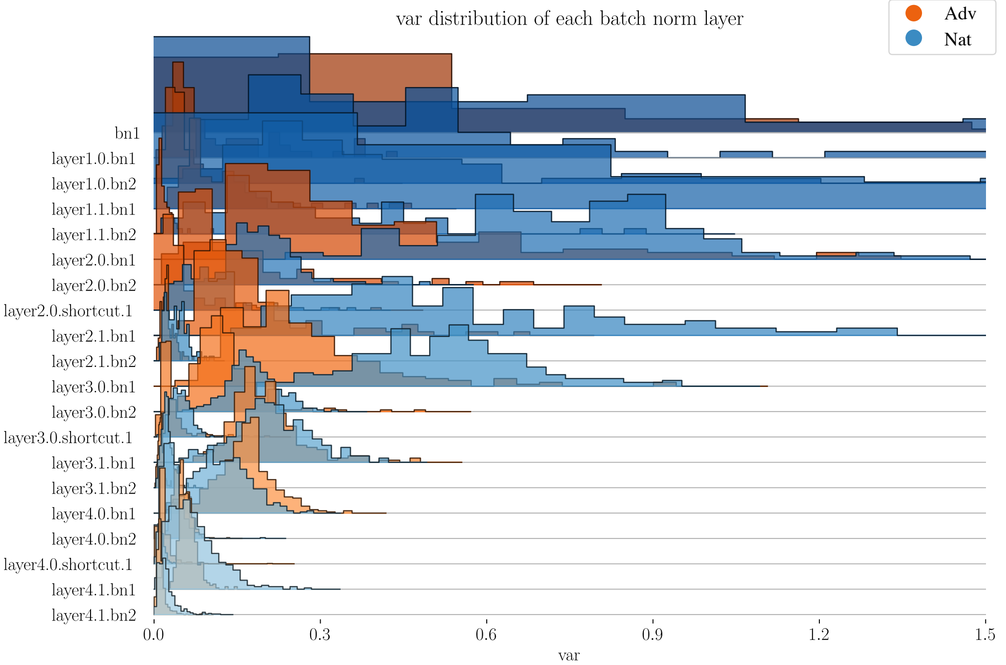

Figure 3.15: Histograms of running variances in each batch norm layer in **ResNet**. Orange histograms are for the adversarially trained network and blue histograms are for the naturally trained network.

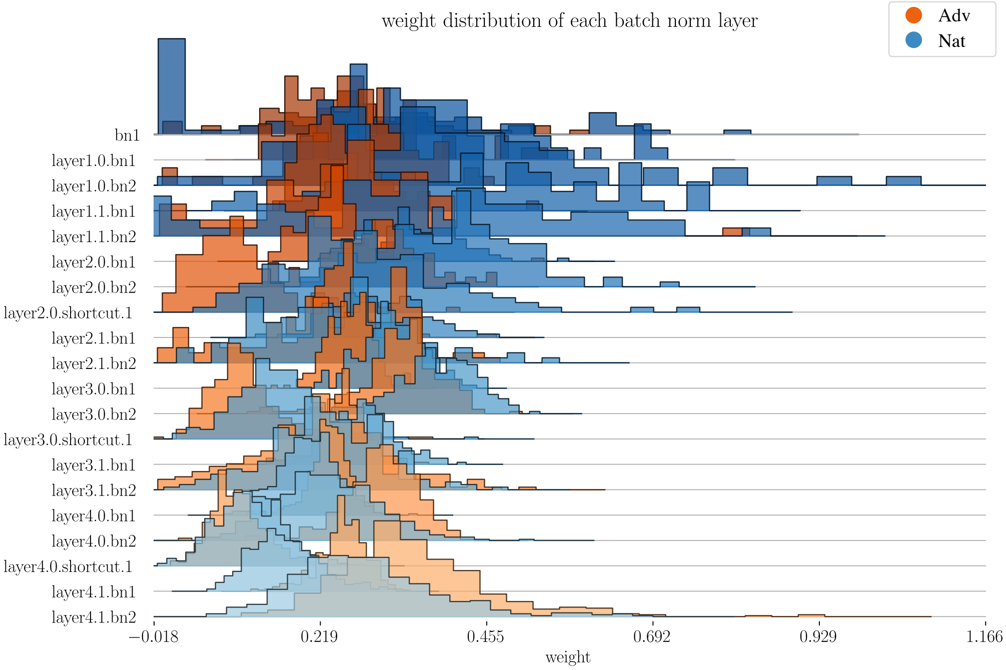

Figure 3.16: Histograms of γ (scale) in each batch norm layer in **ResNet**. Orange histograms are for the adversarially trained network and blue histograms are for the naturally trained network.

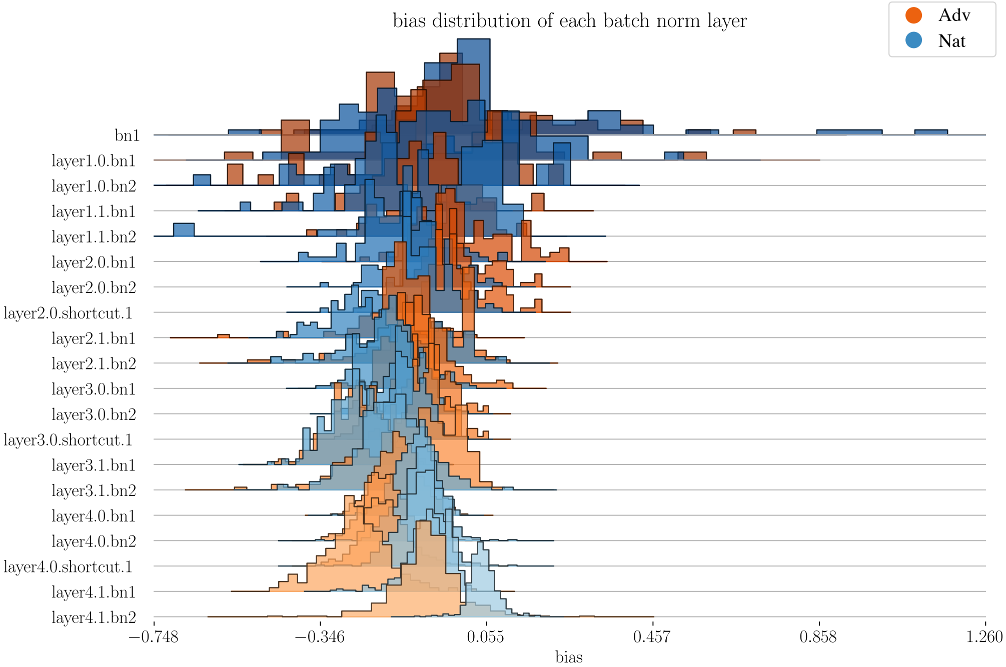

Figure 3.17: Histograms of β (bias/offset) in each batch norm layer in **ResNet**. Orange histograms are for the adversarially trained network and blue histograms are for the naturally trained network.

3.2.2 L^p Norms of Convolutional Layers

Similarly, we consider L^p norms of the filters of the convolutional layers. Specifically, we consider $\frac{(\|\cdot\|_1/\|\cdot\|_2)^2-1}{d-1}$ ratio (see Figures 3.18 and 3.19), which serves two purposes. First, it provides scale invariance. Comparing nominal L^p norms of two networks can yield incorrect conclusions because there's an interplay between weights and the batch norm running statistics, and different networks can converge to filters with different L^p norms by pure chance. Second, as reported in [75], $\frac{(\|\cdot\|_1/\|\cdot\|_2)^2-1}{d-1}$ ratio gives a measure of sparsity level – in fact, better than other proxy measures. 0 means that the filter has a single nonzero element and 1 means all elements in the filter are equal. However in this inquiry too, we run into greatly overlapping distributions of statistics. One outlier is the first convolutional layer; we can see that the adversarially trained network's first layer's filters are consistently sparser in both ResNet and VGG. For other layers, the histograms overlap significantly and it is hard to draw statistically significant conclusions.

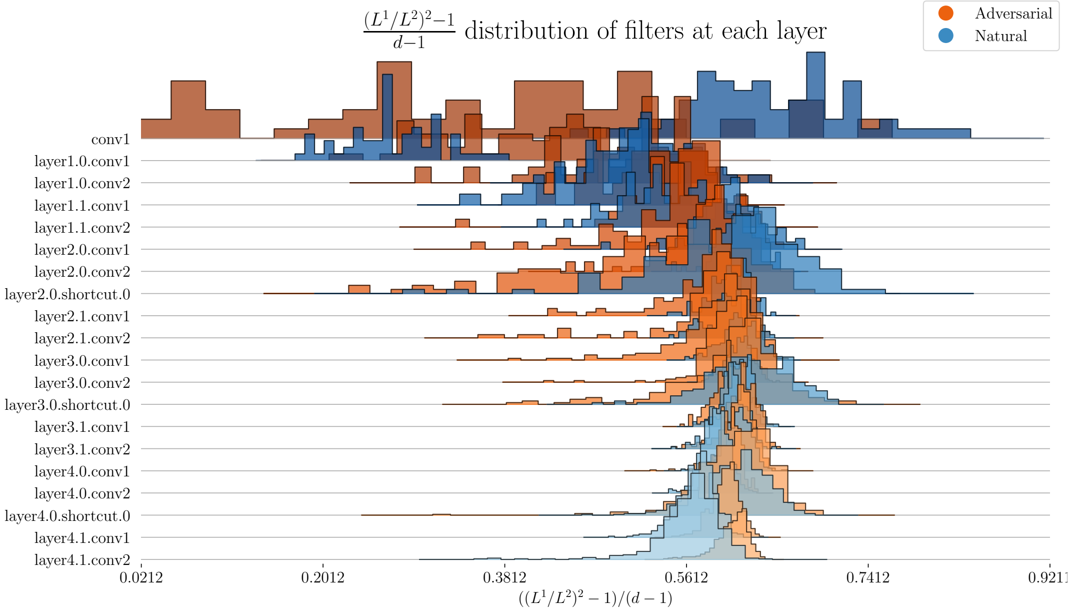

Figure 3.18: Histograms of $\frac{(\|\cdot\|_1/\|\cdot\|_2)^2-1}{d-1}$ ratio of the filters in each convolutional layer in **ResNet**. Orange histograms are for adversarially trained network, blue histograms are for naturally trained network. d is the number of elements in each filter of that layer. 0 means a single non-zero element and 1 means all elements are equal.

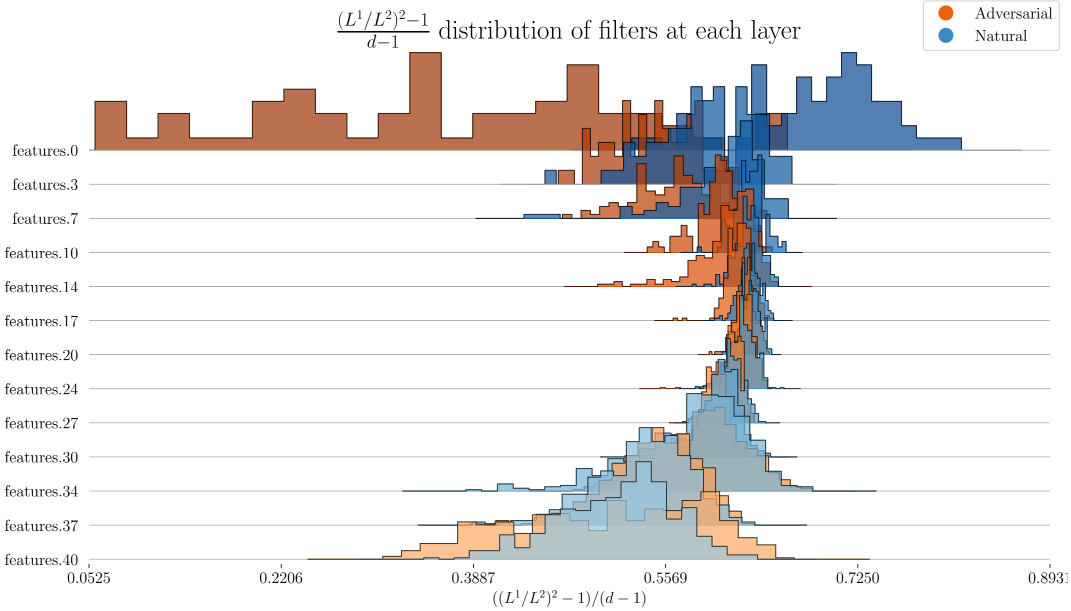

Figure 3.19: Histograms of $\frac{(\|\cdot\|_1/\|\cdot\|_2)^2-1}{d-1}$ ratio of the filters in each convolutional layer in **VGG**. Orange histograms are for adversarially trained network, blue histograms are for naturally trained network. d is the number of elements in each filter of that layer. 0 means a single non-zero element and 1 means all elements are equal.

3.2.3 Singular Value Distributions of Convolutional Layers

Lastly, we consider the linear operation that the convolutional layer applies on the image. We convert each convolutional layer into a very large matrix acting on the entire image, and then plot singular values of this matrix in decreasing order, normalized by the largest singular value. The plots for these singular value distributions are given in Figures 3.20 and Figures 3.21.

From these plots the following conclusions can be drawn:

- For most layers other than the first, adversarially trained models have slightly fatter-tailed singular value distributions than naturally trained models. This indicates that they project their inputs to a more diverse set of principal components.

- For the first layer singular values, the distributions are inconsistent; they vary between architectures but also depend on the optimizer used.

- For both naturally and adversarially trained models, singular values are sparser for the latter layers than the earlier layers.

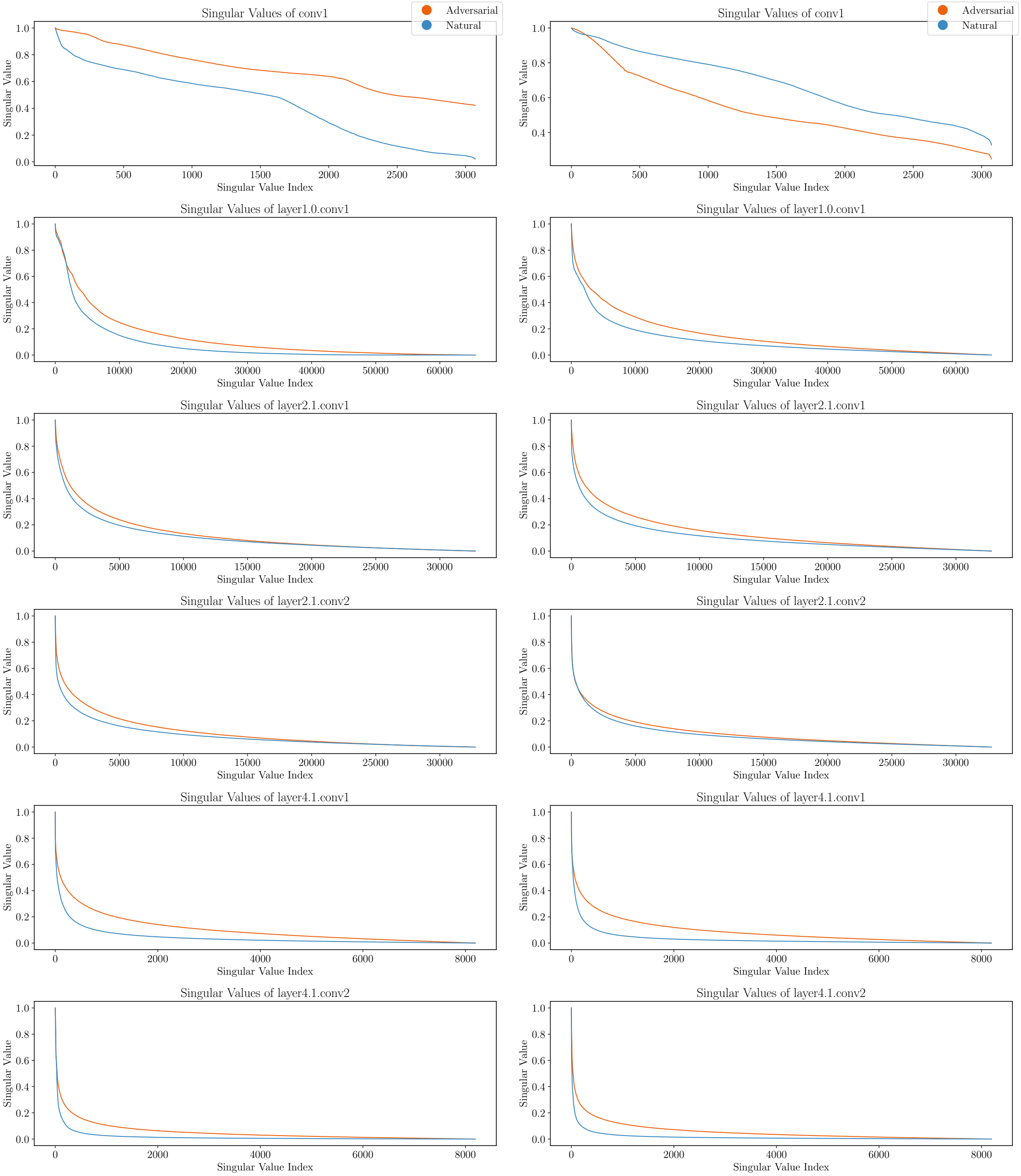

Figure 3.20: Singular value distributions of linear representations of various convolutional layers of **ResNet**, normalized by their top singular value. Top two rows are the first two layers, middle two rows are middle two layers, and bottom two rows are last two layers. Orange plots are for adversarially trained network, blue plots are for naturally trained network. Left: Distributions for ResNet-34 trained with SGD optimizer. Right: Distributions for ResNet-34 trained with Adam optimizer [12].

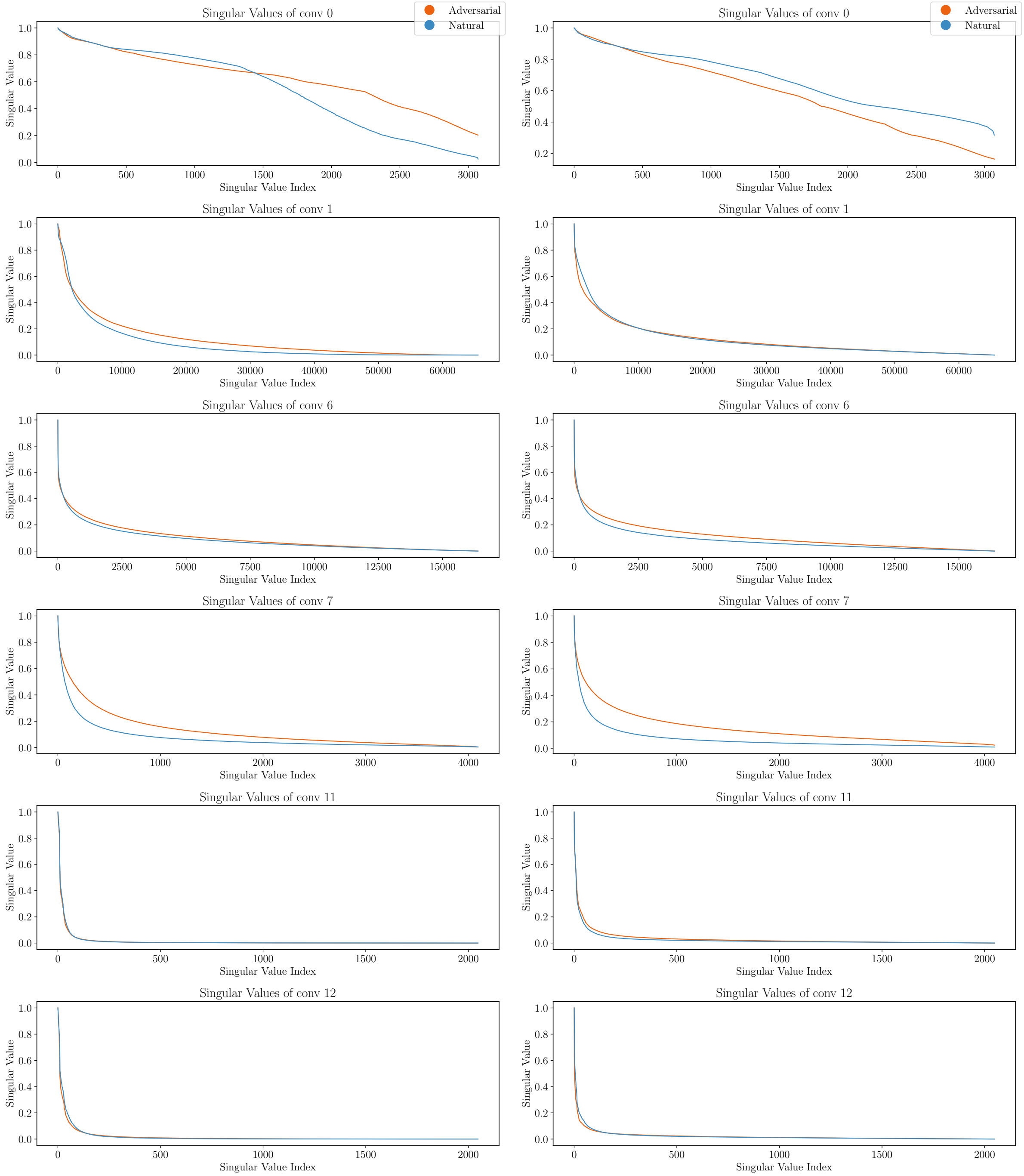

Figure 3.21: Singular value distributions of linear representations of various convolutional layers of **VGG**, normalized by their top singular value. Top two rows are the first two layers, middle two rows are middle two layers, and bottom two rows are last two layers. Orange plots are for adversarially trained network, blue plots are for naturally trained network. Left: Distributions for VGG-16 trained with SGD optimizer. Right: Distributions for VGG-16 trained with Adam optimizer [12].

3.3 Conclusion

The importance of continued research on systematic approaches to the design of robust neural networks, preferably in a manner that we can interpret, cannot be overstated. While adversarial training is the state-of-the-art defense, it is difficult to interpret the resulting networks, and they offer far less robustness under attack than is acceptable in many applications. We hope that our observations, and the methods we have developed to obtain them, open the door to further efforts at dissecting adversarial training, and in developing principled approaches to designing in robustness. For example, it may be possible to be more aggressive about adversarial training in the early layers as we continue to seek methods which trade off clean and attacked accuracy. As another example, while defenses based on preprocessing input data have been defeated, a closer analysis of the early layers may provide insights for the design of improved preprocessing techniques. In terms of measuring robustness, quick computations of perturbation-to-signal ratios for early layers may be more informative about the operation of a layer than exhaustive evaluations of end-to-end accuracy.

An open issue is how the structure of the weights and the patterns of the activations are different for adversarially trained networks than for naturally trained networks. While we perform extensive experiments and observe some differences between the statistics of naturally trained model parameters and adversarially trained model parameters, they lack coherence and discernible trends. Therefore we were unable to extract clear guidelines that apply to the different network architectures we have considered. Thus, in our approaches we focus on directly shaping the activations of layers instead. We believe that continued efforts to gain detailed structural insight are of great value in the quest to develop principled, interpretable approaches to robustness.

Chapter 4

Front End Based Defenses

As discussed in Chapter 3, most of the robustness in adversarial training is achieved by the earlier layers in a neural network. In this chapter we investigate the approach that uses front ends to suppress adversarial attacks before they reach the core neural network. The first approach we develop relies on polarization of first layer outputs and quantization. It is limited to simple datasets in its capability due to its relatively simple design and assumptions. The second front end approach we develop involves a similar polarization idea but achieved through the use of an overcomplete dictionary. It further employs techniques from other fields to alleviate the effects of adversarial attacks before they reach the core classifier.

4.1 Polarizing Front Ends for Robust CNNs

4.1.1 Introduction

In this section, we investigate a systematic, bottom-up approach to robustness, studying a defense based on a nonlinear front end for attenuating adversarial perturbations before they reach the deep network. We focus on L^∞ bounded attacks: $\|\boldsymbol{e}\|_\infty \leq \epsilon$

for an “attack budget” $\epsilon > 0$. Furthermore, we assume a “white-box” attack, in which the adversary has full knowledge of the network structure and weights. Our approach consists of *polarizing* the input data into well-separated clusters by projecting onto an appropriately selected basis (implemented using convolutional filters), and then quantizing the output using thresholds that scale with the L^1 norm of the basis functions. For ideal polarization, we prove that perturbations are completely eliminated. We introduce a regularization technique to learn polarizing bases from data, and demonstrate the efficacy of the proposed defense for the MNIST and Fashion MNIST datasets. The code associated with this section can be found at https://github.com/canbakiskan/polarizing-frontend.

4.1.2 Related Work

A number of quantization-based defense methods have been proposed in the literature, within the neural network [76, 77] and as a front end [78, 79, 80, 81]. The key difference in our proposed strategy is that we employ *polarization* prior to quantization, which enables theoretical guarantees on robustness (Section 4.1.3). We do not claim robustness due to non-differentiability of quantization; we test our defense using the gradient approximation methods of [42] (as discussed in Section 2.2.3).

4.1.3 Polarizing Front End

We investigate a defense based on a front end which preprocesses the inputs via a linear transformation followed by a nonlinear activation f. Following convention, the linear operation of a particular filter is termed a *neuron*. Consider a typical front end neuron with weights $\boldsymbol{w}$ and scalar output a. For perturbed input $\boldsymbol{x} + \boldsymbol{e}$ with L^∞ bound $\|\boldsymbol{e}\|_\infty < \epsilon$, a contains two components: desired signal $\boldsymbol{w}^T\boldsymbol{x}$, and an output perturbation $\boldsymbol{w}^T\boldsymbol{e}$ that is constrained in magnitude: $\left|\boldsymbol{w}^T\boldsymbol{e}\right| \leq \|\boldsymbol{e}\|_\infty \|\boldsymbol{w}\|_1 \leq \epsilon \|\boldsymbol{w}\|_1$ due to Hölder’s inequality. For the defense to be successful, the nonlinearity f must be chosen such that $f(a = \boldsymbol{w}^T(\boldsymbol{x} + \boldsymbol{e})) \approx f(\boldsymbol{w}^T\boldsymbol{x})$.

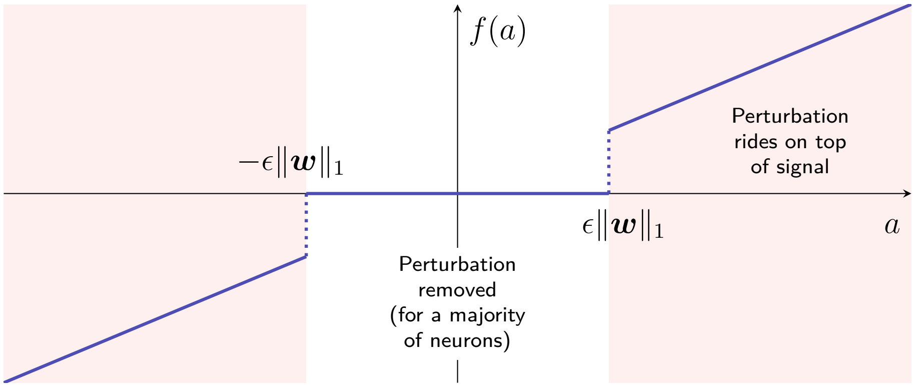

Figure 4.1: Activation sparsity (Equation 4.1) alone is not sufficient to achieve robustness: perturbations can ride on top of strongly activated neurons (shaded region).

One design approach is to promote sparse activations by increasing the threshold for neurons to fire, which makes it difficult for a small perturbation to induce firing:

$$f(a) = \begin{cases} 0 & |a| \leq \epsilon\|\boldsymbol{w}\|_1 \\ a & \text{otherwise} \end{cases} \tag{4.1}$$

While this method helps (see [82, 83, 84] for a similar approach), Figure 4.1 shows why it cannot be completely successful. When a neuron resides near the middle of the unshaded region, no perturbation can change the signal output ($f(a) = 0$). However, neurons with a strong desired signal component (large $|\boldsymbol{w}^T\boldsymbol{x}|$) can serve as hosts for the perturbation, allowing it to propagate through the defense. Hence activation sparsity can only be a part of the solution.

What if we could somehow *polarize* neural activity to obtain well-separated clusters of neurons? Consider for instance the three clusters of activations shown in Figure 4.2. In such a scenario, we can completely eliminate perturbations by using a quantized nonlinearity (in this case, ternary quantization). Note that it is important for neurons to

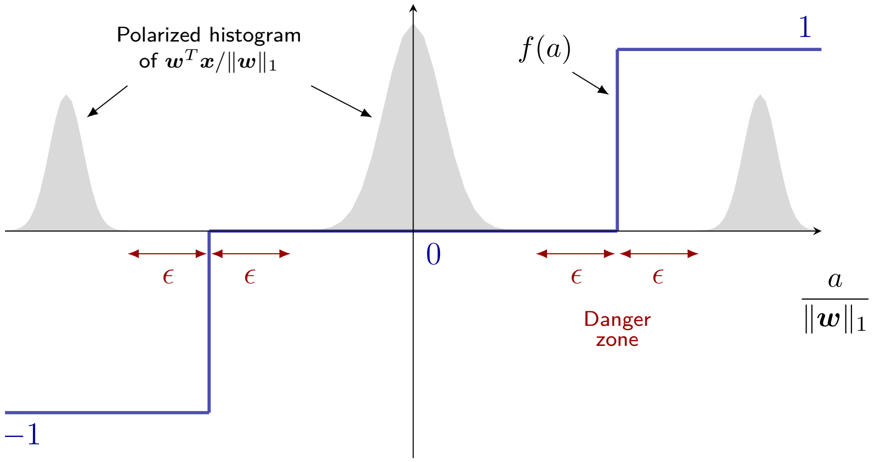
Polarized histogram of $\boldsymbol{w}^T\boldsymbol{x}/\|\boldsymbol{w}\|_1$
$f(a)$
1
0
-1
ϵ
ϵ
ϵ
ϵ
Danger zone
$\frac{a}{\|\boldsymbol{w}\|_1}$

Figure 4.2: Polarization of neural activity can fully eliminate perturbations. For the shown hypothetical histogram (gray) of $\boldsymbol{w}^T\boldsymbol{x}/\|\boldsymbol{w}\|_1$, a ternary activation (blue) is effective.

avoid the "danger zones" of width 2ϵ shown in the figure: this ensures that perturbations cannot switch data from one quantization level to the next. These observations are formalized in the following proposition.

Proposition 1. *Suppose the front end polarizes activations into a multimodal distribution with L clusters, with minimum inter-cluster separation $d > 2\epsilon\|\boldsymbol{w}\|_1$. Let $c_1 < c_2 < \ldots c_{L-1}$ denote the midpoints between adjacent clusters. Then the following L-level quantizer (with thresholds at c_i) completely eliminates perturbations with L^∞ norm smaller than ϵ:*

$$f(a) = \frac{1}{2}\sum_{i=1}^{L-1} \operatorname{sign}(a - c_i). \tag{4.2}$$

Proof. Since we use a quantizing nonlinearity, perturbations can cause distortion only if the output switches quantization levels. We know that for a perturbation $\boldsymbol{e}$ with L^∞ budget ϵ, the maximum output distortion is $\epsilon\|\boldsymbol{w}\|_1$. Therefore, if clusters are separated by a distance of $2\epsilon\|\boldsymbol{w}\|_1$, perturbations cannot propagate through the defense. □

This result motivates a second design approach, where we seek a neural basis in which outputs are well-polarized for clean inputs, with clusters of $\boldsymbol{w}^T\boldsymbol{x}/\|\boldsymbol{w}\|_1$ separated by at least 2ϵ, as shown in Figure 4.2. We can then choose a piecewise constant nonlinearity (Equation 4.2) to eliminate the effect of perturbations. Equipped with these design principles, we now detail training procedures to learn polarizing bases from data.

4.1.3.1 Implementing a Polarizing Front End

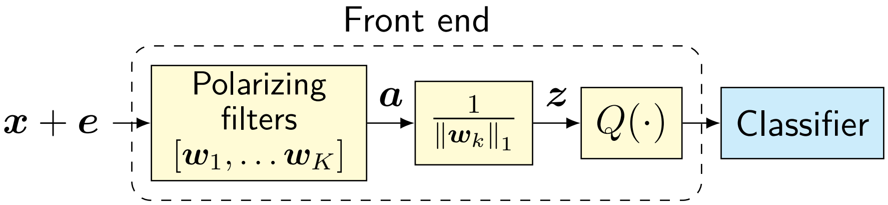
Front end
$\boldsymbol{x} + \boldsymbol{e}$
Polarizing filters $[\boldsymbol{w}_1, \dots \boldsymbol{w}_K]$
$\boldsymbol{a}$
$\frac{1}{\|\boldsymbol{w}_k\|_1}$
$\boldsymbol{z}$
$Q(\cdot)$
Classifier

Figure 4.3: Block diagram of the polarizing front end defense convolution with polarizing filters followed by ℓ_1 normalization and quantization.

We employ a front end (shown in Figure 4.3) which uses convolutional filters to learn polarized and quantized latent representations of data. For a front end neuron $\boldsymbol{w}_k$, let $z_k = a_k/\|\boldsymbol{w}_k\|_1$ denote the normalized activation. We seek a multimodal distribution for $\boldsymbol{z}$, with clusters separated by at least 2ϵ. We achieve this by training with *bump regularizers* $B_1(\cdot)$ and $B_2(\cdot)$ which promote polarization of data. We train in three stages by minimizing the modified loss function:

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{y}_{\text{true}}, \boldsymbol{z}) = \mathcal{L}_{CE}(\boldsymbol{y}, \boldsymbol{y}_{\text{true}}) + \frac{\lambda}{K} \sum_{k=1}^{K} B(z_k)$$

where $\mathcal{L}_{CE}$ is the cross-entropy loss determined by the true label and outputs of the classifier, K is the number of neurons, $\boldsymbol{z}$ is the vector of activations of all neurons $[z_1, \dots z_K]$, B is the regularizer and λ is a scaling coefficient which is adjusted at every epoch as explained in Section 4.1.4. These stages can be described as follows:

1. We start by training the polarizer without using any quantization. The front end filters are initialized from a uniform distribution described in [71]. Due to the

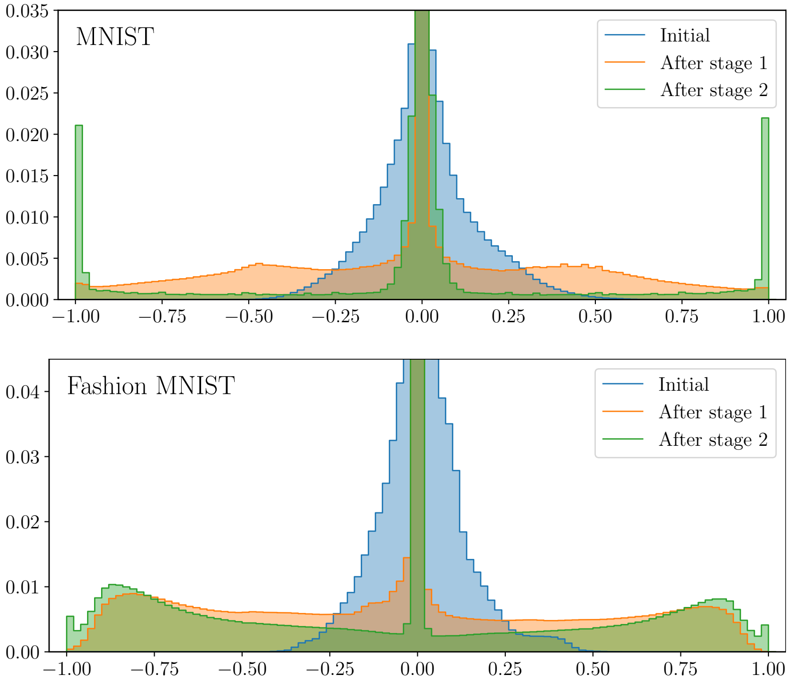

Figure 4.4: Histograms of normalized front end filter outputs $a_k/\|\boldsymbol{w}_k\|_1$ after each training stage for MNIST and Fashion MNIST

random initialization, normalized activations are typically clustered around zero initially (shown in blue in Figure 4.4). Next, we incorporate a bump regularizer $B(\cdot) = B_1(\cdot)$ (Figure 4.5) to drive the normalized activations away from the origin, pushing $\boldsymbol{z}$ towards the endpoints -1 and 1:

$$B_1(z_k) = e^{-z_k^2/2\sigma_1^2}.$$

2. After achieving a sufficiently even level of distribution throughout the interval $[-1, 1]$, we switch to the second bump regularizer $B(\cdot) = B_2(\cdot)$, aimed at pushing the normalized activations away from the quantization thresholds $\pm c$ and polarizing $\boldsymbol{z}$ into three clusters centered at -1, 0 and 1:

$$B_2(z_k) = e^{-(z_k-c)^2/2\sigma_2^2} + e^{-(z_k+c)^2/2\sigma_2^2}.$$

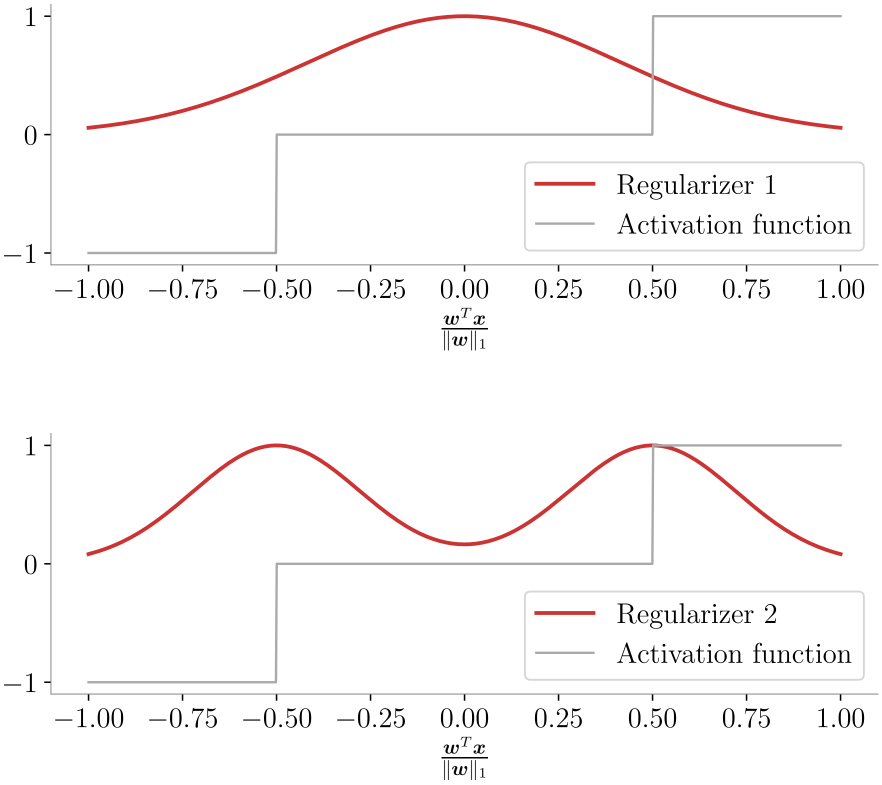

Figure 4.5: Illustration of regularizers B_1 and B_2 superposed onto the quantization function. B_1 ensures even distribution of first layer outputs and B_2 drives the even distribution away from the danger zones around the thresholds.

3. Then we introduce the quantization function $f_2(\cdot)$ described in Equation 4.3. We also stop training and freeze the filters in the front end, and remove the regularizer. We train the classifier to let the weights adapt to the quantization:

$$f_2(z_k) = 0.5\,\mathrm{sgn}(z_k - c) + 0.5\,\mathrm{sgn}(z_k + c). \tag{4.3}$$

For testing, we continue using the quantized activation in Equation 4.3 to eliminate perturbations. Details regarding the choice of parameters such as λ, c, σ_1 and σ_2 are given in Section 4.1.4. We find that these three stages suffice for Fashion MNIST and MNIST, but depending on the dataset, one could potentially repeat Stage 2 with an in-

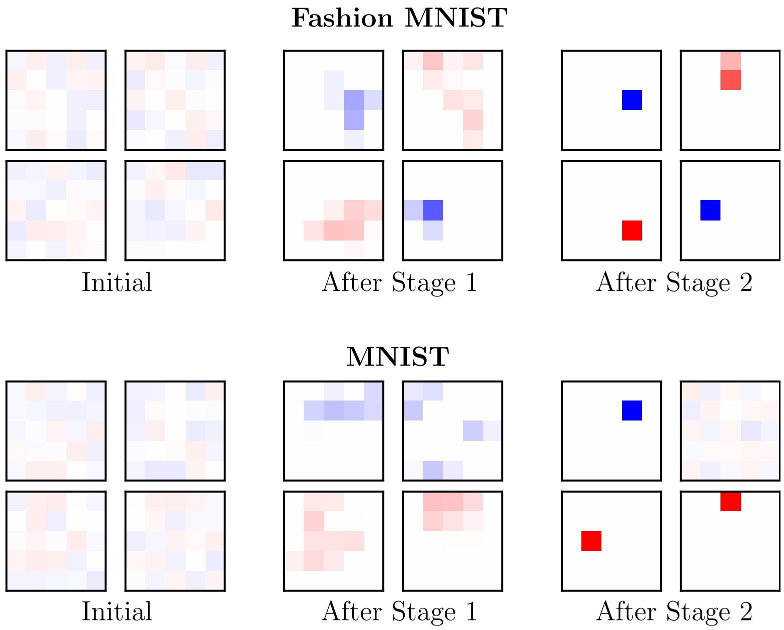

Figure 4.6: Typical progression of front end filters over the training stages.

creasing number of clusters until the desired level of polarization is achieved. Figure 4.4 demonstrates the effects bump regularizers have on the distribution of normalized activations. Figure 4.6 shows the filters obtained after each stage for MNIST and Fashion MNIST. Interestingly, we find that the learned filters appear similar to pixel bases. This is consistent with the observations in [9] about first-layer filters learned by adversarial training on MNIST.

4.1.4 Experiments and Results

4.1.4.1 Training Details

For a fair comparison we use the small convolutional neural network from [9], consisting of two convolutional layers and two fully connected layers. Convolutional layers have 32 and 64 filters that are 5x5 in size. Each convolutional layer is followed by 2x2 maxpooling operation. Every layer except the last uses ReLU activation function. The outputs of the last layer are fed into a softmax function to generate classification probabilities. In every run, the model is trained for 20 epochs in each stage for a total of 60 epochs. Gradient descent is achieved using the Adam optimizer [12] with learning rate 10^{-3} and default hyperparameters in PyTorch library.

During training with bump regularizers, stage 1 and stage 2 bump widths are picked to be $\sigma_1 = 0.35$ and $\sigma_2 = 0.15$, respectively. To make the adaptation of weights smoother, we increase the bump coefficient λ linearly from 0 to 1 in each stage, as the stages progress. The quantization threshold is chosen to be $c = 0.3$ for Fashion MNIST and $c = 0.5$ for MNIST. When adversarially training using the methods of Madry et al. [9] we use 10 restarts and 20 steps in each restart.

Attack Setup: We evaluate our defense against the white-box attacks described in Chapter 2: FGSM, BIM and PGD with Restarts. We use attack budget $\epsilon = 0.3$ for MNIST and $\epsilon = 0.1$ for Fashion MNIST. In iterative methods, we use step size $\alpha = \epsilon/10$. In BIM, we use 20 steps. In PGD, we choose the best performing attack from 20 random restarts, with 100 steps in each restart.

	Fashion MNIST ($\epsilon = 0.1$)				MNIST ($\epsilon = 0.3$)			
	Clean	FGSM	BIM	PGD	Clean	FGSM	BIM	PGD
No defense	91.6	19.7	1.49	0.11	99.4	21.9	0.47	0.00
Adv. Training	83.8	77.9	75.6	74.1	97.5	93.1	90.0	86.7
Ours	87.3	69.4	54.9	44.5	99.1	93.2	86.5	70.8

Table 4.1: Comparison of accuracies for the standard model (no defense), adversarially trained model, and our defense for MNIST and Fashion MNIST datasets.

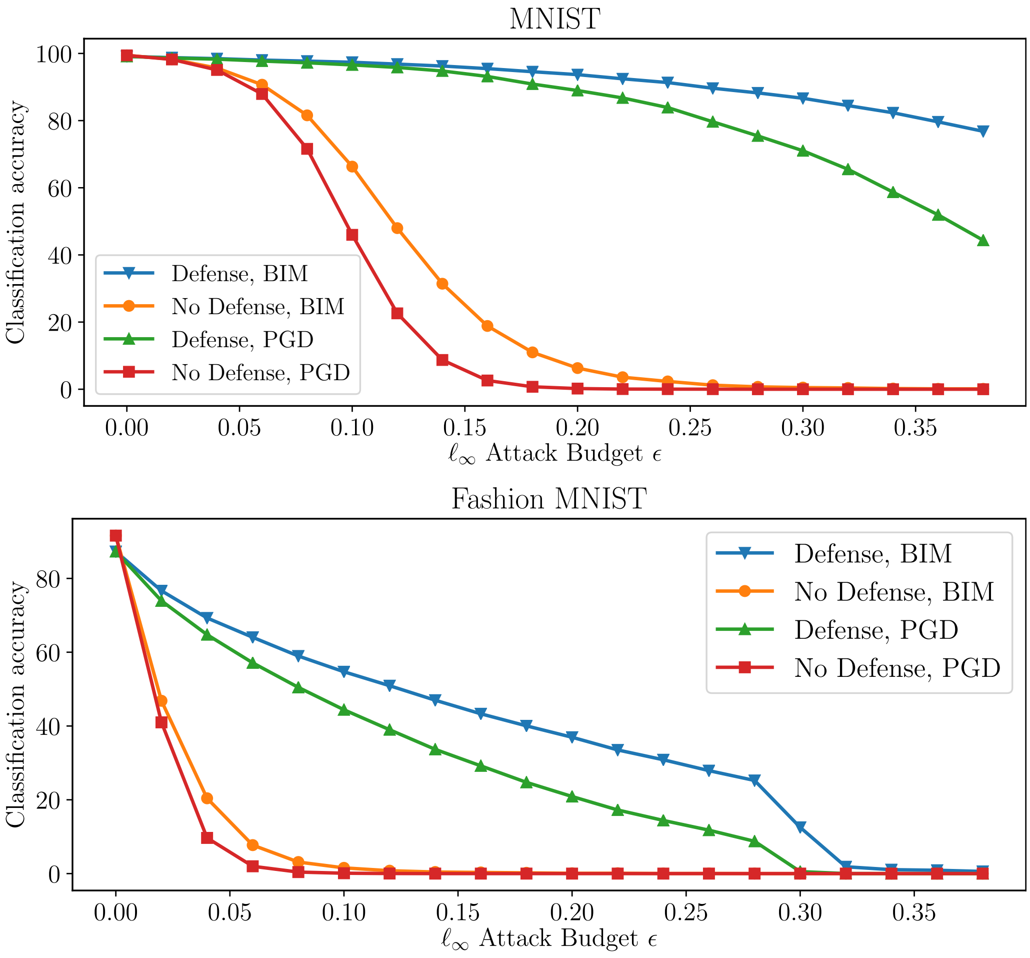

Figure 4.7: Classification accuracy versus L^∞ attack budget for MNIST and Fashion MNIST.

4.1.4.2 Results and Discussion

Figure 4.7 shows the effect of attack budget on accuracy, showing that our front end increases adversarial accuracy across a wide range of ϵ. Table 4.1 details our results against different attacks, with a comparison with methods from the literature.

Our defense significantly improves robustness against a variety of attacks, but falls short of the accuracies obtained by adversarial training. This is because perfect polarization is not possible in practice, leading to some leakage of adversarial perturbations through the front end. However, the polarization approach is amenable to interpretation, and provides an avenue for further efforts in systematic bottom-up design. In contrast, empirical experiments are the only means of verifying the efficacy of state-of-the-art adversarial training.

4.1.5 Conclusions

In this section, we have shown that polarization is a promising tool for defense against adversarial attacks: when data is perfectly polarized, quantization can provably eliminate perturbations. Our training procedures for learning polarizing bases indicate that pixel bases are effective for polarizing datasets like MNIST and Fashion MNIST, which is consistent with the first-layer filters learned in adversarially trained models for these datasets [9].

There are a few key limitations that make this approach inapplicable to more complicated datasets. First, this approach consists of a single layer, which in itself is very limiting, but it also considers first layer outputs to be independent of each other while learning the filters. It solely aims to obtain a polarized distribution at the end of the first layer. Furthermore, quantization combined with these filters results in a heavy loss of information. Therefore, this approach does not scale well to more complicated datasets like CIFAR-10. For such datasets, most of the activations remain in the danger zone due to a high variance in patches in these datasets, which translates to lower adversarial accuracies. For those datasets we need approaches that consist of more layers and take the complexities into account.

4.2 Sparse Coding Front End for Robust Neural Networks

After observing the shortcomings of the polarizing front end in addressing more complex datasets, we now describe our sparse coding based front end. It uses a similar polarization idea, achieved through sparse representation theory. It consists of multiple operations and layers, and generalizes to more complicated datasets.

4.2.1 Introduction

Olshausen & Field [13] showed that when you try to represent images by a distributed code in a sparse way (i.e. approximate an image as a sum of scaled constituent simple images, with as few nonzero coefficients as possible), you obtain basis filters that are similar to those observed in the visual cortex of mammals (Figure 4.8). The field that [13] was part of is called sparse representation theory. In this section, we turn to sparse overcomplete representation theory for defending against adversarial attacks, inspired by the observation that sparse representations are known to have greater robustness in the presence of noise. Specifically, we use the overcomplete basis obtained by sparse coding to have a polarization effect, similar to what we had in our previous defense. In addition, we also incorporate ideas from neuroscience into our defense. While neuro-inspiration could ultimately provide a general framework for designing DNNs which are robust to a variety of perturbations, for this defense we take a first step by focusing on the well-known L^∞ bounded attack, which captures the concept of "barely noticeable" perturbation. Our architecture, illustrated in Figure 4.9, does not require adversarial training. Rather, it consists of the following: (a) a neuro-inspired encoder learned in purely unsupervised fashion, (b) a decoder which produces an output of the same size as the original image,

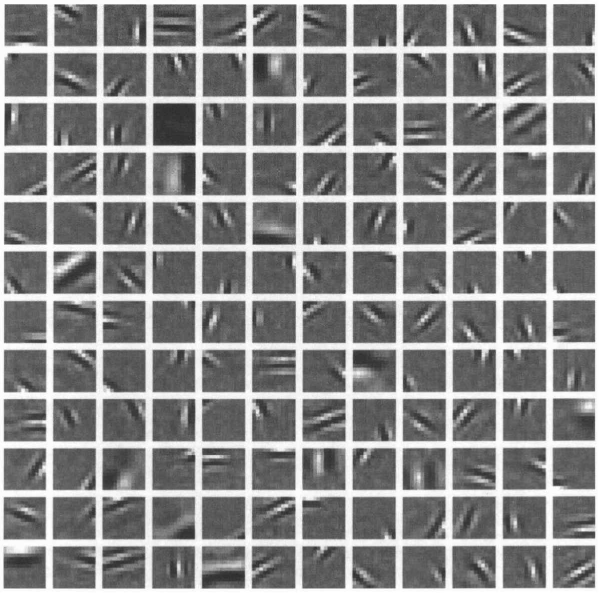

Figure 4.8: Basis filters discovered when Olshausen & Field [13] tried to represent images by a distributed code in a sparse manner. They are similar to filters observed in the visual cortex of mammals. Image taken from [13].

and (c) a standard CNN for classification. The decoder and classifier are trained jointly in standard supervised fashion using *clean* images passed through our encoder.

The key features we incorporate into our encoder design are sparsity and overcompleteness, long conjectured to be characteristic of the human visual system [13], lateral inhibition [85], synaptic noise [86], and drastic nonlinearity [87]. We use standard unsupervised dictionary learning [88] to learn sparse, highly overcomplete (5-10X relative to ambient dimension) patch-level representations. However, we use the learned dictionary in a non-standard manner in the encoder, not attempting patch-level reconstructions. Instead, we take the top T coefficients of the basis projection from each patch (lateral inhibition), thus restricting the attack space, randomly drop a fraction p of them (synaptic noise and lateral inhibition), and threshold and quantize them, retaining only their sign (drastic nonlinearity), thus not allowing perturbations to ride on top of them. We use overlapping patches, providing an additional degree of overcompleteness. The patch-

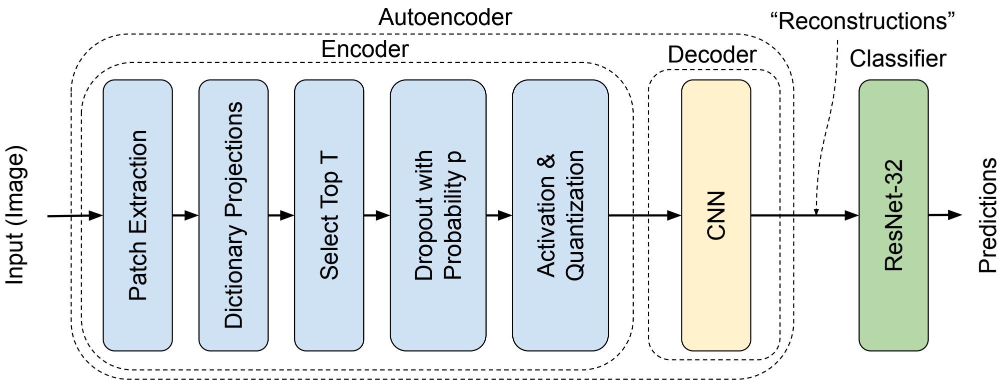

Figure 4.9: Proposed autoencoding defense. Decoder restores input size but does not attempt to reconstruct the input in our nominal design (supervised decoder+classifier training).

level outputs, which have ternary quantized entries, are fed to a multi-layer CNN decoder whose output is the same size as the original RGB image input. This is then fed to a standard classifier DNN.

We report on experiments on the CIFAR-10 and a subset of the ImageNet dataset ("Imagenette"), demonstrating the promise of a "bottom-up" neuro-inspired approach, in contrast to the top-down approaches that currently dominate adversarial machine learning. For state-of-the-art PGD attacks tailored to our architecture, our attacked accuracy is slightly worse than that of adversarial training [9, 47] for CIFAR-10, while it is on par or slightly better than these methods for ImageNette, showing that our approach scales to larger image sizes. The defense parameters are chosen to combat L^∞ bounded attacks, but we demonstrate its general-purpose nature by showing that it also provides robustness against L^2 and L^1 bounded attacks.

We invest significant effort into attacking our own defense: following the guidelines in [35], our strongest attack is tailored specifically to account for the structure of our defense while avoiding the gradient obfuscation problem exposed in [42]. The software for our defense, including the attack library we have created, is available at github.com/canbakiskan/neuro-inspired-defense.

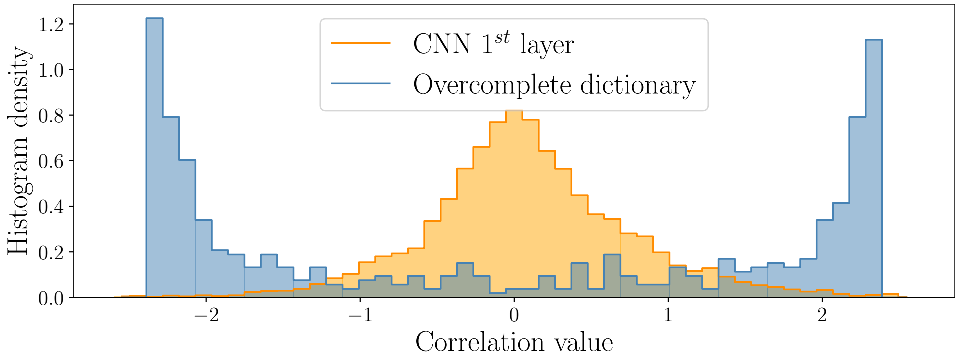

Figure 4.10: Histogram of correlations for a typical patch with atoms of an overcomplete dictionary vs. that of activations through layer 1 filters of a standard classifier CNN.

4.2.2 Sparse Overcomplete Front End

The rationale behind our encoder design is as follows: An overcomplete dictionary for sparse coding results in large activations for a fraction of the atoms, in contrast with filters learned in the first layer of a traditional convolutional neural network where activations are clustered around zero (see Figure 4.10). We can therefore drop most of the activations, reducing the effective subspace available to the attacker. An attacker can still perturb the subset of top T coefficients in each patch. We combat this by randomly dropping a large fraction p of these coefficients, thereby allowing the decoder and classifier to learn to be resilient to randomness in the sparse code, and to an attacker knocking a coefficient out of the top T. The thresholds for ternary quantization of the selected coefficients are selected to provably guarantee that the attacker cannot flip the sign of any nonzero entry in the sparse code. The hard thresholding ensures that the perturbation cannot add to a coefficient which would have been selected for a clean image. Rather, the attacker must invest the effort in pushing a smaller coefficient into the top T, and gamble on it being randomly selected.

4.2.2.1 Patch-Level Overcomplete Dictionary

We consider images of size $N \times N$ with 3 RGB channels, processed using $n \times n$ patches with stride S, so that we process $M = m \times m$ patches, where $m = \lfloor (N-n)/S \rfloor + 1$. Learning at the patch level allows for the extraction of sparse local features, effectively allowing reduction of the dimension of the space over which the adversary can operate for each patch.

We use a standard algorithm [88] (implemented in Python library `scikit-learn`), which is a variant of K-SVD [89], to learn the patch-level overcomplete dictionary (number of basis tensors much larger than the patch dimension $3n^2$) . Given a set of clean training images $\mathcal{X} = \{\mathbf{X}^{(k)}\}_{k=1}^{K}$, an overcomplete dictionary $\mathbf{D}$ with L atoms can be obtained by solving the following optimization problem [88]:

$$\min_{\mathbf{D}\in\mathcal{C},\{\boldsymbol{\alpha}^{(k)}\}_{k=1}^{K}} \sum_{k=1}^{K}\sum_{i,j}\Big(\frac{1}{2}\left\|\mathbf{R}_{ij}\mathbf{X}^{(k)} - \mathbf{D}\boldsymbol{\alpha}_{ij}^{(k)}\right\|_2^2 + \lambda\left\|\boldsymbol{\alpha}_{ij}^{(k)}\right\|_1\Big) \tag{4.4}$$

where $\mathcal{C} \triangleq \big\{\mathbf{D} = [\mathbf{d}_1, \ldots, \mathbf{d}_L] \in \mathbb{R}^{\bar{n}\times L} \mid \|\mathbf{d}_l\|_2 = 1\,, \forall l \in \{1, \ldots, L\}\big\}$, λ is a regularization parameter, $\boldsymbol{\alpha}^{(k)}$ is an $m \times m \times L$ tensor containing the coefficients of the sparse decomposition, and $\mathbf{R}_{ij} \in \mathbb{R}^{\bar{n}\times\bar{N}}$ with $\bar{n} \triangleq 3n^2$ and $\bar{N} \triangleq 3N^2$ extracts the (ij)-th patch from image $\mathbf{X}^{(k)}$. The optimization problem in Equation 4.4 is not convex, but its convexity with respect to each of the two variables $\mathbf{D}$ and $\{\boldsymbol{\alpha}^{(k)}\}_{k=1}^{K}$ allows for efficient alternating minimization [88, 89].

4.2.2.2 Sparse Randomized Encoder

Based on the overcomplete dictionary obtained from Equation 4.4, we encode the image patch by patch. For a given image $\mathbf{X}$, patch $\mathbf{x}_{ij} \in \mathbb{R}^{\bar{n}}$ is extracted based on the (ij)-th block of $\mathbf{X}$ (that is, $\mathbf{x}_{ij} = \mathbf{R}_{ij}\mathbf{X}$), and then projected onto dictionary $\mathbf{D}$ in order to obtain projection vector $\bar{\mathbf{x}}_{ij}$, where $\bar{\mathbf{x}}_{ij} = \mathbf{D}^T \mathbf{x}_{ij}$. Since the dictionary is highly overcomplete, a substantial fraction of coefficients typically take large values, and a sparse reconstruction of the patch can be constructed from a small subset of these. However, our purpose is robust image-level inference rather than patch-level reconstruction. Hence we use the dictionary to obtain a discrete sparse code for each patch using random "population coding," as follows:

1. **Top T selection:** We keep only T elements of the projection vector with largest absolute values and zero out the remaining elements. The surviving coefficients are denoted by $\check{\mathbf{x}}_{ij}$.

2. **Dropout:** Each of the top T coefficients is dropped with probability p, leaving surviving outputs

$$\tilde{\mathbf{x}}_{ij}(l) = \begin{cases} 0, & \text{with probability } p \\ \check{\mathbf{x}}_{ij}(l), & \text{with probability } 1 - p \end{cases} \tag{4.5}$$

   for all $l \in \{1, \ldots, L\}$. Note that we use dropout for both train and test time, as opposed to only during training in its standard usage.

3. **Activation/Quantization:** Finally, we obtain sparse codes with discrete values by applying binary quantization with a dead zone designed to reject perturbations.

$$\hat{\mathbf{x}}_{ij}(l) = \begin{cases} \operatorname{sgn}\left(\tilde{\mathbf{x}}_{ij}(l)\right) \left\|\mathbf{d}_l\right\|_1, & \text{if } \frac{|\tilde{\mathbf{x}}_{ij}(l)|}{\epsilon \|\mathbf{d}_l\|_1} \geq \beta \\ 0, & \text{otherwise} \end{cases}, \tag{4.6}$$

   for all $l \in \{1, \ldots, L\}$, where $\beta > 1$ is a hyperparameter. The scaling of the surviving ± 1 outputs by $\left\|\mathbf{d}_l\right\|_1$ allows basis coefficients surviving a larger L^1 norm based threshold to contribute more towards the decoder input, but these could be omitted,

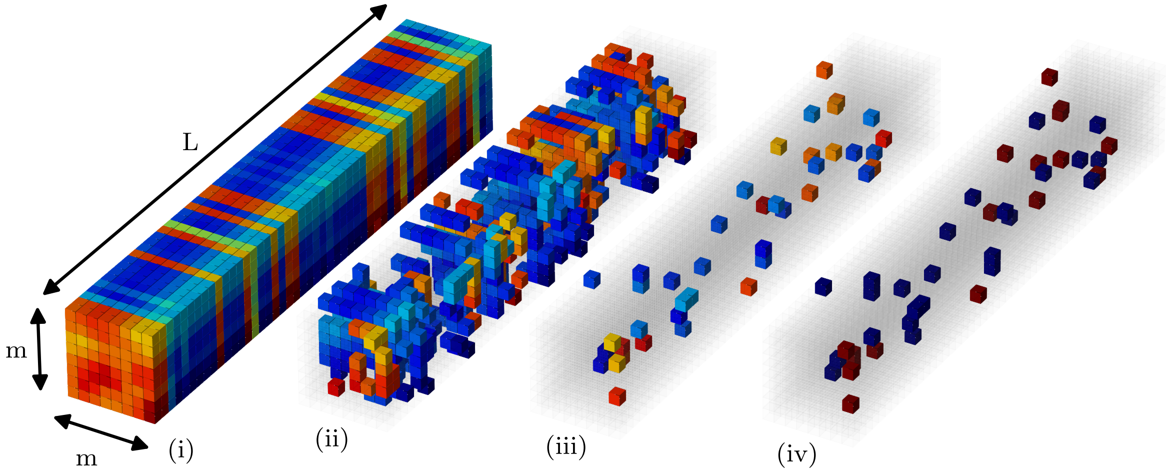

Figure 4.11: Progression of coefficients after each operation: (i) Each voxel shows projection onto a dictionary atom, (ii) Projections after taking top T, (iii) Remaining projections after dropout, (iv) Projections after activation and quantization. Notice the saturation in color.

since the decoder can learn the appropriate weights.

Rationale: By Hölder's inequality, an attacker with L^∞ budget ϵ can perturb the kth basis coefficient by at most $\epsilon \left\|\mathbf{d}_k\right\|_1$. By choosing $\beta > 1$, we guarantee that an attacker can never change the sign of a nonzero element of the sparse code. Thus, the attacker can only demote a nonzero element to zero, or promote a zero element to a nonzero value. As discussed, a large dropout probability alleviates the impact of both demotions and promotions. Even though this is designed with L^∞ bounded attacks in mind, it provides robustness against L^2 and L^1 bounded perturbations as well, as will be shown later.

Another consequence of choosing $\beta > 1$ is that weak patches whose top T coefficients are not large enough compared to the maximum perturbation $\epsilon \left\|\mathbf{d}_k\right\|_1$ get killed, thereby denying the adversary the opportunity to easily perturb the patch-level sparse code. Following patch-level processing with stride S, the encoder outputs an image level sparse code which is an $m \times m \times L$ tensor (Figure 4.11).

4.2.2.3 Sparse Encoder Without Randomization

Another encoder scheme we try eliminates the dropout component altogether. In this scheme, for each patch, we keep only the T elements of the projection vector with largest absolute values, zeroing out the remaining elements. Then, we apply the activation function same as in the randomized encoder scheme. We report the results for this encoder type separately in Section 4.2.4.1.

4.2.2.4 CNN-based Decoder

The discretized sparse codes $\bar{\mathbf{x}}_{ij}$ are fed into a CNN-based decoder with three transposed convolutional layers, each followed by ReLU activation function, that produces output with dimension $N \times N \times 3$ and restores the original RGB image size. The goal of the CNN-based decoder is to restore the output dimension rather than to reconstruct the image.

4.2.2.5 Ensemble Processing

In order to utilize the full potential of the randomization employed in the encoder, we allow for ensemble processing in which an input image is processed using E random realizations of our encoder during inference, with classifier softmax outputs averaged across the realizations.

4.2.3 Evaluation

State-of-the-art PGD attacks are based on gradient computation, and an important criticism of prior proposals of preprocessing-based defense is that they were masking/obfuscating/shattering gradients, without adapting the attacks to compensate for this [42]. Indeed, many such defenses do fail when attacks are suitably adapted, leading a group of prominent researchers to propose guidelines for evaluating defenses [90]. Following such guidelines, we devote substantial effort to devising attacks adapted to our defense, as follows:

- We consider several different smooth backward pass replacements for the parts of our defense that are non-differentiable:
  - for the activation function, we test two backward passes: identity and a smooth approximation.
  - for top T taking operation, we test identity backward pass as well as propagating gradients through top U coefficients where $U > T$.
- We check that there exist some ϵ values that reduce the adversarial accuracy to 0 (see Figure 4.13).
- We test our defense against attacks with a large number of iterations or restarts.
- We test against different threat models such as the following: L^∞, L^2, and L^1 bounded attacks; decision and query based black-box attacks that do not rely on gradients; transfer black-box attacks with different surrogate models.
- To ensure that the gradients are computed appropriately, we test computing smoothed gradients where gradients are averaged over many different points in the neighborhood of the original point, in each step of the attack (EOT [34]). However, we observe that this does not significantly increase the attack strength.

4.2.4 Experiments, Results and Discussion

Our main focus is on evaluating our defense on the CIFAR-10 dataset ($N = 32$), for which there are well-established benchmarks in adversarial ML. In order to verify that

Dictionary learned from CIFAR10

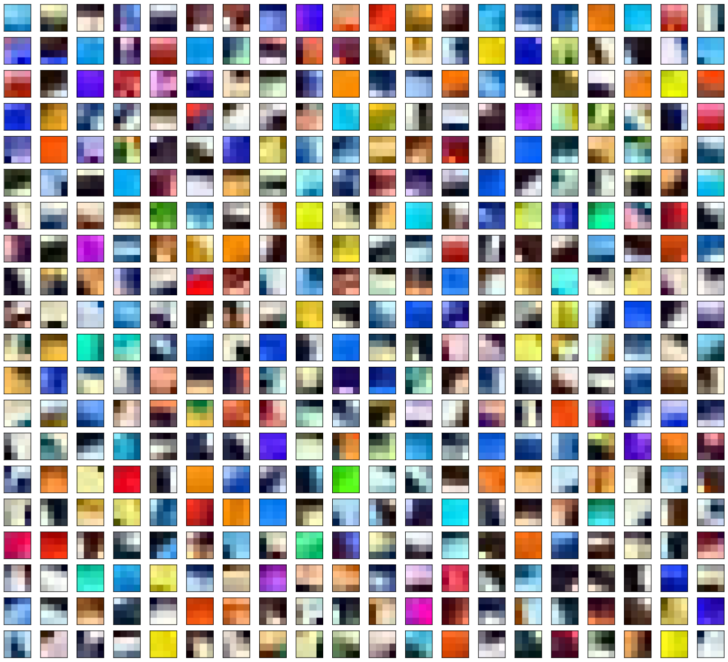

Figure 4.12: Dictionary learned through the optimization in Equation 4.4 from patches extracted from the training images of CIFAR-10 dataset.

our approach scales to larger images, we also consider the Imagenette dataset: 9469 train and 3925 validation RGB images, cropped to size 160 × 160 ($N = 160$). Both datasets contain images from 10 classes. For CIFAR-10, we use 4×4 patches ($n = 4$) and an overcomplete dictionary with $L = 500$ atoms. The stride $S = 2$, so the encoder output is a $15 \times 15 \times 500$ tensor ($m = 15$, $L = 500$). The regularization parameter in Equation 4.4 is set to $\lambda = 1$ and the number of iterations is chosen as 1000 to ensure convergence. The hyperparameters for Imagenette are 8×8 ($n = 8$) patches and an overcomplete dictionary with $L = 1000$ atoms, stride $S = 4$, which gives encoder outputs of size $38 \times 38 \times 1000$ ($m = 38$, $L = 1000$). The number L of dictionary atoms is 10 times the ambient dimension for CIFAR-10, and 5 times the ambient dimension for ImageNette. The number of iterations in dictionary learning is set to 10000, and to promote sparsity, the regularization parameter λ is set to 0.5, in the upper range of values resulting in convergence. A subset of the resulting atoms are shown in Figure 4.12.

We set $T = 50$, $p = 0.95$, $E = 10$ for our nominal defense based on ablation studies, with hyperparameter $\beta = 3$ for the threshold in Equation 4.6. We train the CNN-based decoder in supervised fashion in tandem with the classifier, using the standard cross-entropy loss. We use a cyclic learning rate scheduler [91] with a minimum and maximum learning rate of $\eta_{min} = 0$ and $\eta_{max} = 0.05$, respectively. In order to provide a consistent evaluation, we employ the ResNet-32 classifier used in [9] for CIFAR-10, and use EfficientNet-B0 [92] for Imagenette. The number of epochs for supervised training is 70 for CIFAR-10 and 100 for Imagenette.

Attacks: We report on three attacks on our nominal defense: (1) *White-box*, where every differentiable operation is differentiated. For the non-differentiable activation/quantization, we take a smooth backward pass approximation. (2) *Pseudo-white-box - transfer (PW-T)*, where we generate white-box attacks for an *unsupervised-trained decoder* with the same encoder, with standard supervised training of the classifier. This attack is adapted specifically for our defense and does not apply to the benchmark defenses that we compare against. *Black-box*, where the adversarial attack is generated based on a standard adversarially trained surrogate classifier. For attacks, we consider PGD and PGD with EOT, if it is applicable. Different from the existing EOT implementation, we use $\delta \cdot \text{sign}\left(\mathbf{E}_r\left[\nabla_x / ||\nabla_x||_2\right]\right)$ in each step to compute the expectation, which we find yields stronger attacks. Unless otherwise stated, we use the following parameters for L^∞ bounded PGD with EOT for CIFAR-10 trained models: an attack budget of $\epsilon = 8/255$, a step size of $\delta = 1/255$, a number of $N_S = 20$ steps, a number of $N_R = 1$ restarts, and a number of $N_E = 40$ realizations for EOT. The same default attack parameters are used for attacking models trained on Imagenette, but given the lack of standard benchmarks, we test several attack budgets $\epsilon \in \{2/255, 4/255, 8/255\}$.

Benchmarks: Our benchmarks are the PGD adversarially trained (AT) [9], R+FGSM

	PGD with EOT			
	Clean	White-Box	PW-T	Black-Box
Our defense	80.1	61.3	**39.5**	57.8

Table 4.2: Accuracies for our defense method (stochastic encoder) under different attacks (CIFAR-10, $\epsilon = 8/255$)

adversarially trained [93], and TRADES [47] defenses for the same classifier architecture. We reimplement these, to enable stress-testing these defenses with attacks of varying computational complexity. We train these models for 100 epochs with the same cyclic learning rate that we use for our models, and verify, for ResNet-32 classifier for CIFAR-10 and EfficientNet-B0 for Imagenette, that we can reproduce results obtained using the original code. For both PGD AT and TRADES, training hyperparameters are $\epsilon = 8/255$, $\delta = 1/255$, $N_S = 10$, and $N_R = 1$. In addition, for TRADES $\lambda_{\text{TRADES}} = 1/6$. For RFGSM AT, they are $\epsilon = 8/255$, $\alpha = 10/255$. We also report on naturally trained (NT) networks (i.e., no defense).

Note that the classifier CNN used in our experiments is "simpler" ResNet-32 rather than the *wide* ResNet-32, both of which are utilized in [9] and other studies in the literature. The choice of the smaller ResNet-32 network makes evaluation of attacks computationally more feasible.

Robustness against Defense-Adapted Attacks: We first investigate the performance of our defense under the different attack types specified earlier. Table 4.2 provides clean and adversarial accuracies for the different attack types. We note that the worst-case attack for it is *not* a white-box attack, rather, it is the pseudo-white-box transfer (PW-T) attack. While this result is surprising at first, it is intuitively pleasing. An attack succeeds only to the extent that it can change the identities of the top T coefficients in the encoder. Since the latter is designed to preserve information about the original

	Clean	Adversarial (Worst case)
NT	**93.1**	0.00
PGD AT [9]	79.4	42.1
RFGSM AT [93]	80.9	42.4
TRADES [47]	75.2	**45.8**
Our defense	80.1	39.5

Table 4.3: Comparison of our defense (stochastic encoder) with other defense techniques (CIFAR-10, $\epsilon = 8/255$). Attack details are: PGD with $N_S = 100$, $N_R = 50$ for the first 4 rows and PGD EOT with $N_S = 20$, $N_R = 1$, $N_E = 40$ for the last row.

image, providing an unsupervised decoder might provide better guidance to the attacker by giving it a reproduction of the original image to work with.

Comparison with benchmarks: Table 4.3 lists *worst-case* accuracies for each defense, where we vary the computational burden of attack on the benchmarks up to a point that is comparable to the default settings for our own EOT/PGD attack. NT denotes natural training (no defense). The worst-case adversarial accuracy for our defense is 39.5%, a little worse than the worst-case accuracies of 42-46% for the benchmark defenses. On the other hand, the worst-case accuracy of our defense (again achieved by the PW-T attack) is slightly better than for the benchmark defenses for Imagenette, as reported

	Clean	Adversarial ($\epsilon = x/255$)		
		$x = 2$	$x = 4$	$x = 8$
NT	**89.35**	11.44	0.28	0.00
PGD AT	80.97	75.31	68.81	53.32
TRADES	80.08	75.67	70.75	59.46
Our defense	79.36	**76.03**	**72.81**	**65.45**

Table 4.4: Comparison of our defense (stochastic encoder) with other defense techniques for Imagenette dataset.

in Table 4.4. We conjecture that the reason results are better for Imagenette is that a patch for an Imagenette image encompasses less of the total image area than that of a CIFAR-10 image. Hence, patches have less variability and are represented better with the sparse coding framework. This leads to a smaller loss of information for Imagenette after the drastic quantization.

4.2.4.1 Non-random encoder

We also test our defense with non-random encoder (i.e. without dropout) against a wide range of attacks and compare the results with benchmark defenses from the literature using the CIFAR-10 dataset. (See Appendix B.3 for the choice of hyperparameters and training settings for both our defense and benchmarks.) Table 4.5 compares our defense with the benchmarks from the literature. In all white-box attacks, we use the following attack settings: number of steps $N_s = 40$, number of random restarts $N_r = 100$ and step size $\delta = \epsilon/8$. For the attacks against our defense, we also use $U = 2T$ and the backward pass smoothing mentioned in Section 4.2.3 to make it stronger. (See Appendix B.2 for discussion of these choices and accuracies for other attack variations.) Column 1 reports on clean accuracy. Columns 2 and 3, report results for the L^∞ bounded white-box attack with cross-entropy loss and Carlini-Wagner loss, respectively. Columns

	Clean	White-Box (L^∞, $\epsilon = \frac{8}{255}$)	White-Box (C&W) (L^∞, $\epsilon = \frac{8}{255}$)	White-Box (L^2, $\epsilon = 0.6$)	White-Box (L^1, $\epsilon = 30$)
Natural	**92.66**	0.00	0.00	0.00	0.00
Adv. Training	79.41	43.27	42.03	52.09	19.98
TRADES	75.17	**45.79**	**42.87**	51.35	21.15
$T = 15$ (Ours)	85.45	37.33	37.35	**60.47**	**47.13**

Table 4.5: Performance comparison of different defense methods with our defense with non-random encoder (CIFAR-10).

4 and 5, report results for L^2 and L^1 bounded attacks, respectively.

We highlight the following observations from Table 4.5: (a) the clean accuracy (column 1) is better than that of the adversarially trained benchmarks, (b) the adversarial accuracy (columns 2 and 3) for L^∞ bounded perturbations falls short of the adversarially trained benchmarks, (c) the adversarial accuracy for L^2 (column 4) and L^1 (column 5) bounded perturbations is substantially better than that of the benchmarks.

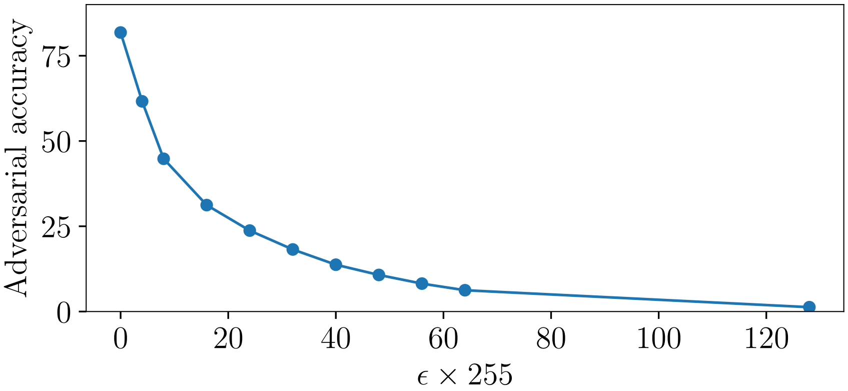

Figure 4.13: Accuracy vs ϵ plot for the sparse coding defense without dropout ($T = 15$) under L^∞ PGD attack with various ϵ values.

The robustness of our defense to L^p attacks with different p is because the impact of different attacks is similar: they flip ±1 to 0 and 0 to ±1 in the sparse code. In contrast, standard adversarial training approaches are optimized for L^p bounded attacks for a specific p and do not perform as well under other types of attacks.

4.2.5 Conclusion

The promising results obtained with our sparse coding front end open up an exciting new direction for building bottom-up defenses to make neural networks more robust for a wider range of attacks. We can tune our defense to approach the performance of state-of-the-art adversarial training for L^∞ bounded attacks, while also providing robustness against L^2 and L^1 bounded attacks. While our results demonstrate the potential of sparse overcomplete representations, neuro-inspiration and bottom-up design of robust DNNs, there is significant scope for further improvement. We attenuate perturbations in a single, rather drastic, encoding step, but spreading the burden across more layers may help with both clean and attacked accuracy. Such an approach that distributes the workload of combatting attacks across layers, making the networks more robust from within, is described in Chapter 5. Our separation of decoder and classifier enables reuse of standard classifier architectures, but there might be better options. Finally, the efficacy of the transfer attack designed specifically for our defense highlights the need for further research on adaptive attacks for novel defenses.

Chapter 5

Incorporating Neuro-inspired Bottom-up Principles for Robustness

The previous two front end based defenses are restricted in their power by the fact that they try to mitigate adversarial attacks before they reach the core network. A more general approach would be structural elements that can be used to complement any layer of a neural network. In this sense, the next defense is more general than the previous two in that the techniques used in the model can be used to replace any layer of a neural network, not necessarily be used in a front end. For this approach we take inspiration from neuroscience and supplement our model with Hebbian rule based updates which allows us to have more control over the features extracted.

5.1 Introduction

In this chapter, we explore a complementary approach to robustness, based on supplementing the end-to-end cost function with layer-wise costs aimed at shaping the features extracted by intermediate layers of the DNN. Specifically, while standard DNNs produce a large fraction of small activations at each layer, we seek architectures which produce a small fraction of strong activations, while continuing to utilize existing network architectures for feedforward inference and existing software infrastructure for stochastic gradient training. To this end, we introduce neuro-inspired mechanisms creating competition between neurons during both training and inference. The code for this bottom-up approach is available at https://github.com/metehancekic/SparseStrongActivations.

5.1.1 Approach and Contributions

In order to attain sparse, strong activations at each layer, we employ the following neuro-inspired strategy for modifying standard DNN training and architecture:

Hebbian/anti-Hebbian (HaH) Training: We augment the standard end-to-end discriminative cost function with layer-wise costs at each layer, promoting neurons with large activations and demoting neurons with small activations. The objective is to create a neural basis that generates a distributed sparse code without the reconstruction cost associated with traditional sparse coding. [13].

Neuronal Competition via Normalization: We increase sparsity even more by using *Divisive Normalization* (DN), which allows larger activations to suppress smaller activations. We use *Implicit L^2 Normalization* of the neuronal weights to ensure a fair competition amongst neurons. This way, each activation can be seen as a geometric projection of the layer input onto the "direction" of the neuron. (Using implicit rather than explicit weight normalization in our inference architecture simplifies training.)

We report on experiments using the CIFAR-10 image classification dataset, comparing a baseline VGG-16 network trained with HaH training and DN against the same architecture trained with end-to-end training. Both architectures use implicit weight normalization, which we have verified does not adversely impact accuracy. We show that, compared to the baseline network, the activations in our suggested architecture are indeed more sparse. We do not use noise augmentation or adversarial training in our experiments in order to separate the effects of our training approach and inference architecture. We demonstrate that our model is significantly more resistant to noise and adversarial perturbations than the baseline model for CIFAR-10 classification. Our model is more robust than both the baseline model and an adversarially trained model against the broader collection of corruptions in the CIFAR-10-C dataset (common corruptions dataset), in general.

5.1.2 Related Work

In artificial neural networks, Hebbian learning has a long history that dates back to the neocognitron [94] and includes recent attempts to incorporate it into deep architectures [95]. However, to the best of our knowledge, ours is the first approach to clearly demonstrate an increase in robustness from its integration in DNNs. A widely accepted concept in neuroscience [96, 97], divisive normalization has been demonstrated to be competitive with other normalization methods in deep neural networks. Our novel contribution is to demonstrate that divisive normalization can be engineered to improve sparsity and robustness. Finally, sparse coding with a reconstruction objective was shown to lead to neuro-plausible outcomes in a seminal paper decades ago [13]. Our HaH-based training targets strong sparse activations in a way that is compatible with conventional stochastic gradient training, as opposed to the iterative sparse coding and dictionary learning in such an approach.

Recent research demonstrating possible robustness improvements by incorporating well-known elements of mammalian vision into DNNs includes [54], which employs Gabor filter blocks and stochasticity, and [98], which employs neural activity measurements from

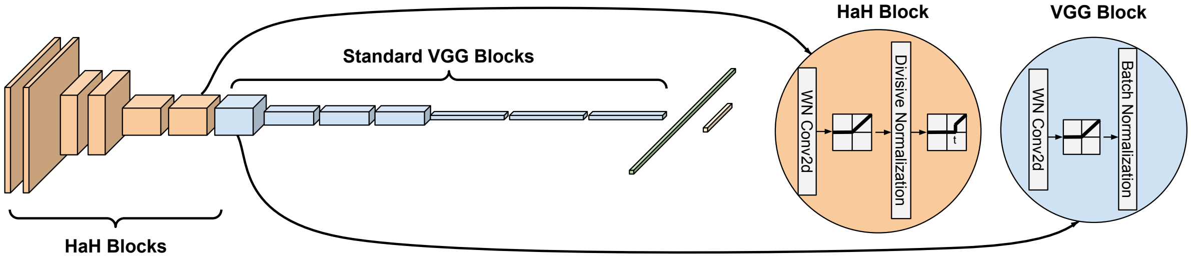

Figure 5.1: Our model consists of two different types of blocks: first 6 blocks are Hebbian-anti-Hebbian (HaH), while the rest are regular VGG blocks. HaH blocks use a weight normalized convolutional layer, followed by ReLU, divisive normalization and thresholding. Regular VGG blocks use a weight normalized convolutional layer followed by ReLU and batch norm.

mice for regularizing DNNs. Instead of including specific biological vision components, we draw broad ideas from neuro-inspiration that can be incorporated into data-driven learning and inference in DNNs.

5.2 Model

This section describes the way we incorporate HaH training and divisive normalization into a standard CNN for image classification. Since the residual connections of ResNet ([71]) variants complicate our interpretation of building models from the bottom-up using HaH learning, we use a linear CNN for our experiments (VGG-16 [72]), applied to the CIFAR-10 dataset. We alter the initial few convolutional blocks to include neuro-inspired principles since we want to create robustness from the bottom up. We name these modified blocks “HaH blocks” (see Figure 5.1).

Each HaH block uses convolution with implicit weight normalization, then ReLU, followed by divisive normalization, and then thresholding. Implicit weight normalization allows us to understand the convolution outputs for each filter as projections. We have ensured that using it in all blocks of a baseline VGG-16 architecture does not have an adverse effect on accuracy (indeed, it slightly improves it). Therefore, each standard (non-HaH) block in our architecture also uses convolution with implicit weight normalization, followed by ReLU, but employs batch norm rather than divisive normalization. Each HaH block contributes a HaH loss element for training, so that the overall loss function used for training is the standard discriminative cross-entropy loss and the sum of the HaH losses from each HaH block.

We now describe the core components of our architecture, shown in Figure 5.2.

5.2.1 Inference in a HaH block

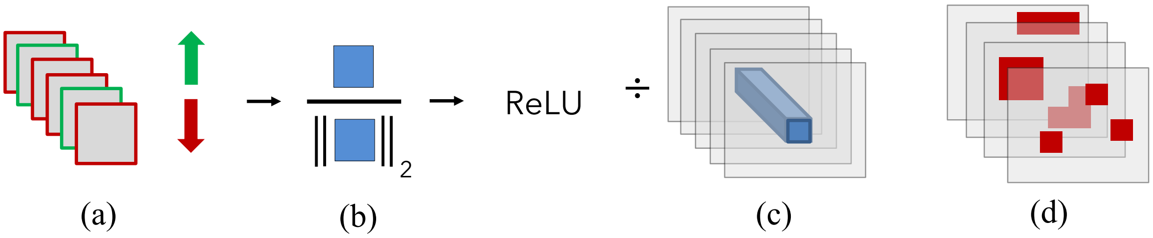

Figure 5.2: Illustrations for each component of a HaH block: (a) convolution with layer filters learned partly through Hebbian–anti-Hebbian updates, (b) division of layer outputs by the L^2 norm of their corresponding filters, effectively normalizing each filter, (c) divisive normalization, which divides each pixel by the mean across channels at that location to both provide scale invariance and suppress the weak "noise" elements, and (d) per-channel thresholding, which kills a fixed percentage of all activations in each channel helping attenuate the impact of noise on smaller activations.

Implicit weight normalization: If we think of the convolution output at a given spatial location from a given filter as a vector inner product $\langle \cdot, \cdot \rangle$ between the filter weights $\boldsymbol{w}$ vector and the input vector $\boldsymbol{x}$, the output of the ReLU unit following the convolution is computed by

$$y = \text{ReLU}\left(\frac{\langle \boldsymbol{w}, \boldsymbol{x} \rangle}{||\boldsymbol{w}||_2}\right) \tag{5.1}$$

This effectively means the weight tensor of each filter is normalized to unit L^2 norm, without actually having to enforce an L^2 norm constraint in the cost as a regularizer or explicitly normalizing the filters after each optimization step.

Divisive normalization: Let $y_1(loc), ..., y_N(loc)$ denote the activations corresponding to the N filters in a given HaH block computed as in Equation 5.1 for a given spatial location loc. Let $M(loc) = \frac{1}{N}\sum_{k=1}^{N} y_k(loc)$ represent the mean of the activations at a given location, and let $M_{max} = \max_{loc} M(loc)$ denote the maximum of this mean over all locations. The divisive normalization we use for each activation is computed as follows:

$$z_k(loc) = \frac{y_k(loc)}{\sigma M_{max} + (1-\sigma)M(loc)} \ , \ \ k = 1, ..., N \tag{5.2}$$

where $0 \leq \sigma \leq 1$ is a hyperparameter which can be separately tuned for each HaH block. As we can see, in addition to the competition among neurons at a given location achieved by dividing by $M(loc)$, M_{max} in the denominator suppresses the contributions at locations for which the input is "noise" rather than a strong enough "signal" well-aligned with one or more of the filters. The output of a HaH-block is scale-invariant (i.e., we obtain the same output if we scale the input to the block by any positive scalar) thanks to this specific implementation of divisive normalization.

Per-channel Adaptive Thresholding: Finally, channel-specific thresholding following the divisive normalization allows us to ensure that each neuron is generating significant outputs. The output of the kth neuron at location loc is given by

$$o_k(loc) = \begin{cases} z_k(loc) & \text{if } z_k(loc) \geq \tau_k \\ 0, & \text{otherwise} \end{cases} \tag{5.3}$$

where the threshold τ_k is neuron and image specific. For example, we may set τ_k to the 90th percentile of the statistics of $z_k(loc)$ in order to get an activation rate of 10% for each channel for every image. Another simple choice that works well, but gives higher activation rates, is to set τ_k to the mean of $z_k(loc)$ for each image.

5.2.2 HaH Training

For an N-channel HaH block with activations $y_k(loc)$, $k = 1, ..., N$ at location loc, the purpose of Hebbian/anti-Hebbian loss is to maximize the average of the top K activations and minimize the average of the remaining $N - K$ activations, where K is a hyperparameter. Therefore, sorting the activations $\{y_k(loc)\}$ so that $y^{(1)}(loc) \geq y^{(2)}(loc) \geq ... \geq y^{(N)}(loc)$, the contribution to HaH loss at this location is given by

$$L_{block}(loc) = \frac{1}{K}\sum_{k=1}^{K} y^{(k)}(loc) - \lambda \frac{1}{N-K}\sum_{k=K+1}^{N} y^{(k)}(loc) \tag{5.4}$$

where $\lambda \geq 0$ is a hyperparameter determining the importance of the anti-Hebbian component of the adaptation. The overall HaH cost for the block, L_{block}, which we wish to maximize, is obtained by taking the mean over all locations and batch images.

The overall loss function to be minimized is then given by

$$L = L_{disc} - \sum_{\text{HaH blocks}} \alpha_{block} L_{block} \tag{5.5}$$

where L_{disc} is the standard discriminative loss (cross-entropy) and $\{\alpha_{block} \geq 0\}$ are the hyper-parameters determining the relative weight of the HaH loss for each block.

Cheating: When naively implemented, we discovered that HaH costs lead to a form of cheating by the network layers. The network learns to adjust the earlier layer weights such that for almost all images a designated subset of channels are very active (shown in yellow/green in Figure 5.3), satisfying the Hebbian loss, and the rest is used for the actual classification task. To overcome the issue of cheating, we do not backpropagate the gradients coming from the Hebbian loss, thereby restricting how much earlier layers can influence the activity of latter layers, since this highly active subset is the result of an avalanche effect building up from earlier layers. Weights of the earlier layers would be arranged such that they result in these always active subsets. With this setup of non-propagation, the most a layer can do is to try to "match" its own inputs, which is what we want. Alongside this, division of the convolutional layer outputs by the 2-norm of their corresponding filters (weight normalization) also serves to prevent cheating by eliminating the factor of weight norm in the layer outputs.

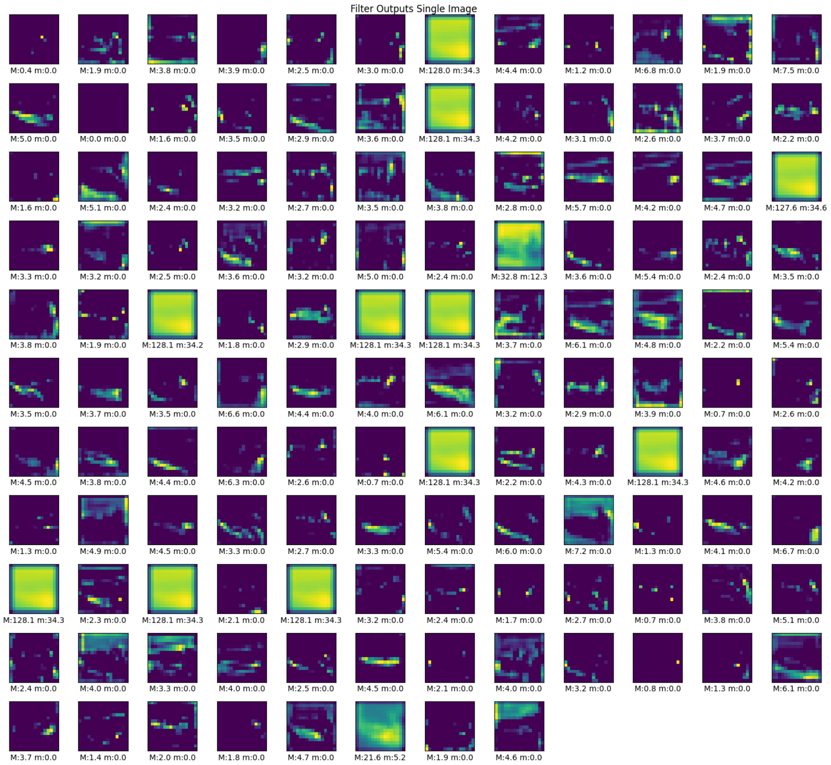

Figure 5.3: HaH block convolutional layer outputs illustrating the cheating in the naive implementation. The network learns to adjust the earlier layer weights such that for almost all images a designated subset of channels are very active (shown in yellow/green), satisfying the Hebbian loss, and the rest is used for the actual classification task.

5.3 Experiments

We replace the first 6 blocks (each block includes conv, ReLU, batch norm) of VGG-16 with the HaH blocks (each block includes normalized conv, ReLU, divisive norm, thresholding). For training, we use Adam optimizer [12] with an initial learning rate of 10^{-3}, multiplied by 0.1 at epoch 60 and again at epoch 80. We train all models for 100 epochs on CIFAR-10. We choose τ_k in Equation 5.3 to keep 20% of activations. We use $[4.5 \times 10^{-3}, 2.5 \times 10^{-3}, 1.3 \times 10^{-3}, 1 \times 10^{-3}, 8 \times 10^{-4}, 5 \times 10^{-4}]$ for α in Equation 5.5. We use 0.1 for λ and set K to 10% of number of filters in each layer in Equation 5.4 and set $\sigma = 0.1$ in Equation 5.2. Details about other hyper-parameters can be found in our code repository.

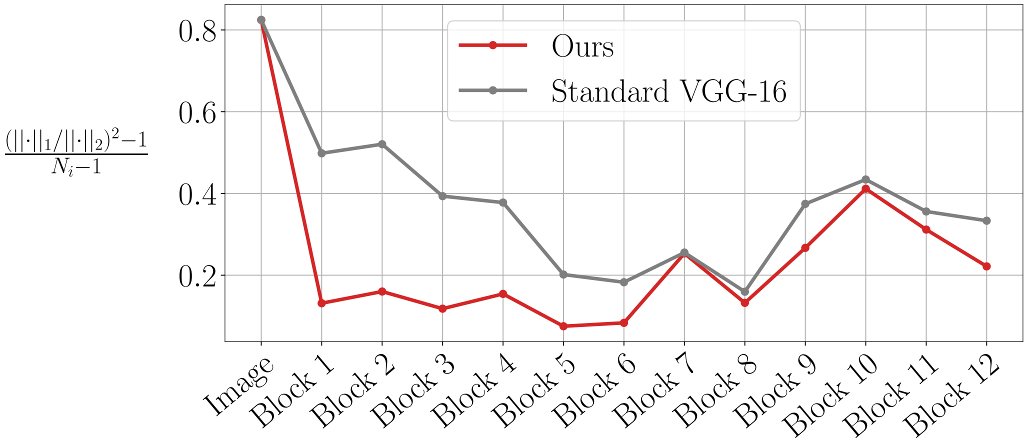

Figure 5.4: HaH blocks yield sparser activations than baseline, measured by the squared Hoyer term.

Sparser activations: To demonstrate that HaH blocks are operating as intended and attaining the sparse and strong activations, we compute the sparsity levels of intermediate representations and plot them in Figure 5.4. We compute the sparsity by the ratio of L^1 norm square to L^2 norm square (also known as squared Hoyer term [99]) of each spatial location's representation across the channel dimension. We then normalize the values so that the sparsity measure lies between 0 and 1. 0 corresponds to a single nonzero element whereas 1 corresponds to all elements being equal. We observe that the activations in the first 6 blocks are indeed sparser for our architecture than for the baseline VGG.

Enhanced robustness to noise: In order to obtain a block-wise measure of robustness, we borrow the concept of signal-to-noise-ratio (SNR) from wireless communication. Let $f_n(\boldsymbol{x})$ denote the input tensor of the n^{th} block in when clean image $\boldsymbol{x}$ is fed to the network, and $f_n(\boldsymbol{x}+\boldsymbol{w})$ denote the input tensor when the image corrupted by noise $\boldsymbol{w}$ is fed. We define SNR by

$$\text{SNR}_n = 10\log_{10}\left(\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{test}}\left[\frac{||f_n(\boldsymbol{x})||_2^2}{||f_n(\boldsymbol{x}+\boldsymbol{w}) - f_n(\boldsymbol{x})||_2^2}\right]\right) dB \tag{5.6}$$

as illustrated in Figure 5.5, converting to logarithmic decibel (dB) scale, as is common practice. Figure 5.6 shows that the SNR for our model well exceeds that of the standard model, especially in the first 6 HaH blocks.

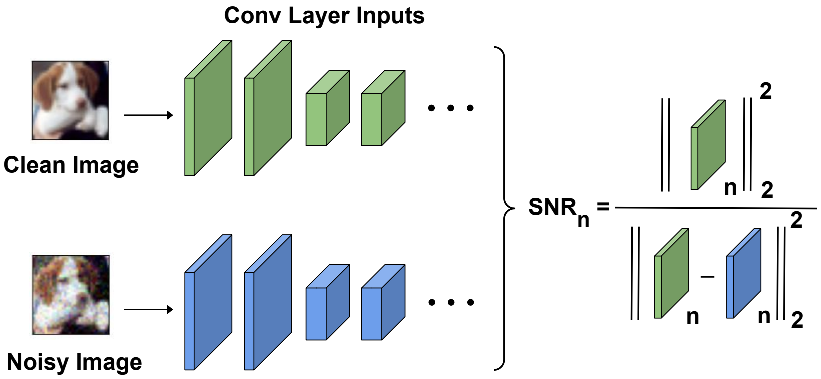

Figure 5.5: To compute the SNR at the n^{th} block inputs, we divide the L^2 norm of the block input corresponding to clean image by the L^2 norm of the difference of block corresponding to clean and noisy images.

These higher SNR values also correspond to gains in accuracy with noisy images. Figure 5.7 compares the accuracy of our model and the base model for different levels of Gaussian noise. There is substantial increase in accuracy at high noise levels: 64% vs. 26% at a noise standard deviation of 0.1, for example.

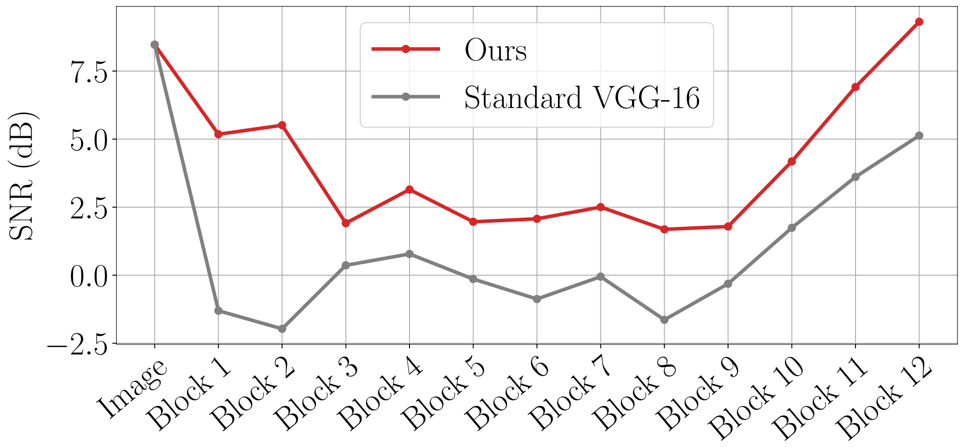

Figure 5.6: Comparison of SNR values of the block inputs for the standard base model (gray) and ours (red) under i.i.d. Gaussian noise with $\sigma = 0.1$.

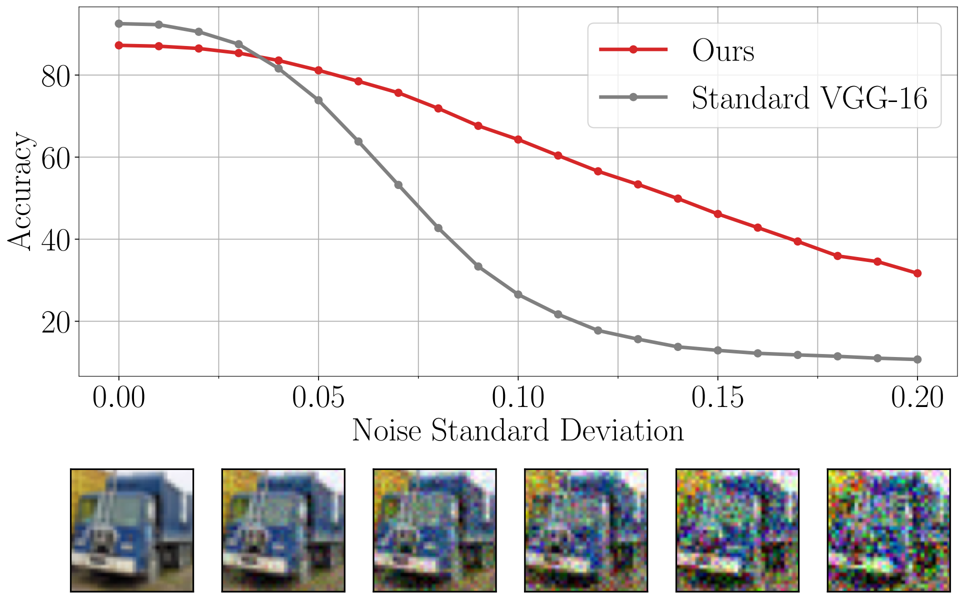

Figure 5.7: Comparison of classification accuracies as a function of noise standard deviation σ. To provide a concrete sense of the impact of noise, noisy images at increasing values of σ are shown below the graph.

Enhanced robustness to adversarial attacks: Even though we haven't trained with adversarial examples, we observe that the HaH blocks' ability to reject noise also translates into improvements in adversarial robustness compared to the baseline VGG model. This is true for state-of-the-art gradient-based attacks [9, 100], as well as AutoAttack, an ensemble of parameter-free attacks suggested by RobustBench [101]. We find no additional benefit of using gradient-free (black-box) attacks, and conclude that the robustness provided by our scheme is not due to gradient-masking or other flaws in evaluation.

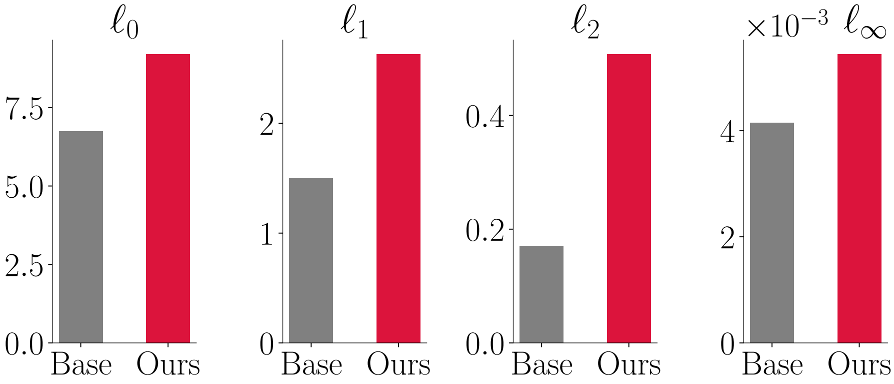

Figure 5.8: The average norm of minimum-norm adversarial attacks is higher for our model than the standard model for all L^p norms considered.

We can see in Figure 5.8 that the minimum distortion needed to flip the prediction of our model (computed using the recently proposed fast minimum norm computation method [100]) is higher for our model than the standard model for all the L^p attacks considered. We have also observed gains in adversarial accuracy against all four L^p norm attacks ($p = 0, 1, 2, \infty$) used as benchmarks in adversarial machine learning. A subset of results demonstrating the increased accuracy against noise and adversarial perturbations, at the expense of a slight decrease in clean accuracy, is displayed in Table 5.1.

Enhanced robustness to common corruptions: We also test our neuro-inspired framework under common corruptions suggested by [102]. These corruptions include digital manipulations, noise injection, and various weather conditions. Accuracies ob-

	Clean	Noisy ($\sigma = 0.1$)	Adv (L^∞) ($\epsilon = 2/255$)	Adv (L^2) ($\epsilon = 0.25$)
Standard	**92.5%**	26.6%	10.4%	13.9%
Ours	87.3%	**64.0%**	**21.5%**	**27.6%**

Table 5.1: Comparison of accuracies for the baseline model and our bottom-up defense under Gausian noise, L^∞ bounded attack and L^2 bounded attack.

tained by our model are compared with those for a standard model and an adversarially trained model in Table 5.2 . We observe that our neuro-inspired design successfully increases robustness against these common corruptions. Note that while adversarially trained models perform well against noise type corruptions, they perform significantly worse against more complex corruptions like contrast and fog [103, 104]. On the other hand, our HaH framework not only performs relatively well (performing much better than the standard model) for noise type corruptions but also substantially outperforms the adversarially trained model on more complex corruptions such as fog and contrast. Furthermore, the HaH-VGG16 performs better than both the standard model and the adversarially trained model in terms of the mean corruption accuracy (last column). Given that such corruptions barely affect human vision, these results demonstrate that neuro-inspiration presents a promising path towards general-purpose robustness against noise, adversarial perturbations, and common corruptions.

Corruptions →	Clean	Noise				Weather				Blur				Digital				Mean
Models ↓		Gauss.	Shot	Speckle	Impulse	Snow	Fog	Frost	Bright.	Defocus	Gauss.	Motion	Zoom	Contrast	Elastic	Pixelate	Spatter	of all
Standard	**92.5**	32.4	40.0	45.5	27.5	72.9	**64.5**	61.6	**87.4**	45.5	34.8	59.7	58.9	23.0	**74.8**	51.0	68.6	53.0
Adv(8/255)	78.7	**74.2**	**74.4**	**73.4**	**62.9**	62.3	29.7	59.0	60.4	**69.8**	**67.5**	**67.2**	**72.0**	18.0	72.5	**75.4**	71.5	63.1
HaH (Ours)	87.3	64.7	63.9	61.2	50.2	**74.4**	63.3	**73.3**	83.3	65.9	59.9	65.8	69.5	**76.3**	73.8	62.1	**76.3**	**67.7**

Table 5.2: Common corruption accuracies across different models. While standard and adversarially trained models are VGG16, HaH (ours) uses the aforementioned modified version of VGG16. Adversarially trained models perform poorly on fog and contrast corruptions while excelling on high-frequency corruptions like noise. On the other hand, the HaH framework consistently improves the robustness against all sorts of corruptions. Bright. = brightness, Gauss. = Gaussian, Elastic = elastic transformation.

Interpretability of features: The main aim of using a Hebbian cost function which directly influences the HaH layer outputs was to control the representations used by the neural network and have it use more interpretable features from the inputs. We can demonstrate this by visualizing what features the first layer of the network focuses on using a tool called Grad-CAM [105] for both standard and HaH trained networks. Grad-CAM uses heatmaps to demonstrate regions in a target layer's output that contributed most to the end decision of the network. The regions that contribute more to the end decision are colored red and regions that do not contribute as much are colored blue. We can see in Figure 5.9 that for most of the images our model focuses on features that humans would find more interpretable.

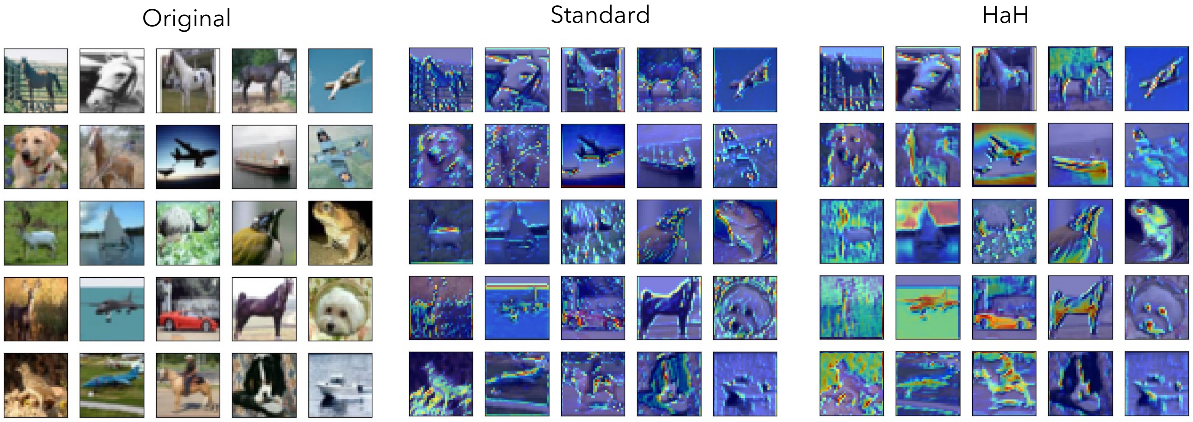

Figure 5.9: Visualization of what features the first layer of the network focuses on using Grad-CAM for both standard (middle) and HaH trained (right) networks.

Ablation: To test the effectiveness of different components in our HaH blocks, we do an ablation study by removing one component at a time and measuring its performance. The results of the ablation study are shown in Table 5.3, where it is evident that each and every one of the components (HaH loss, divisive normalization, per-channel thresholding) play an important role in the realized gain in robustness to noise and adversarial attacks.

We also test the effect of the number of HaH blocks in the network's robustness. Figure 5.10 shows the trade-off between clean accuracy and robust accuracy as the number

	Clean	Noisy ($\sigma = 0.1$)	Adv (L^∞) ($\epsilon = 2/255$)	Adv (L^2) ($\epsilon = 0.25$)
All included	87.3%	**64.0%**	**21.5%**	**27.6%**
No HaH loss	89.7%	50.4%	8.8%	11.7%
Batch norm instead of divisive norm	**90.4%**	46.7%	12.3%	17.4%
No thresholding	89.9%	37.5%	3.7%	2.5%

Table 5.3: Accuracies when we remove components from our defense one at a time (ablation study).

of HaH blocks increases. We can see that with each additional HaH block, robustness to noise improves, for both adversarial noise and otherwise. Note that after 7 HaH blocks the network fails to converge due to the layer inputs to divisive normalization being too small to be meaningful. As observed in other bio-inspired defenses, robustness through the use of HaH blocks also comes with a small decrease in clean accuracy.

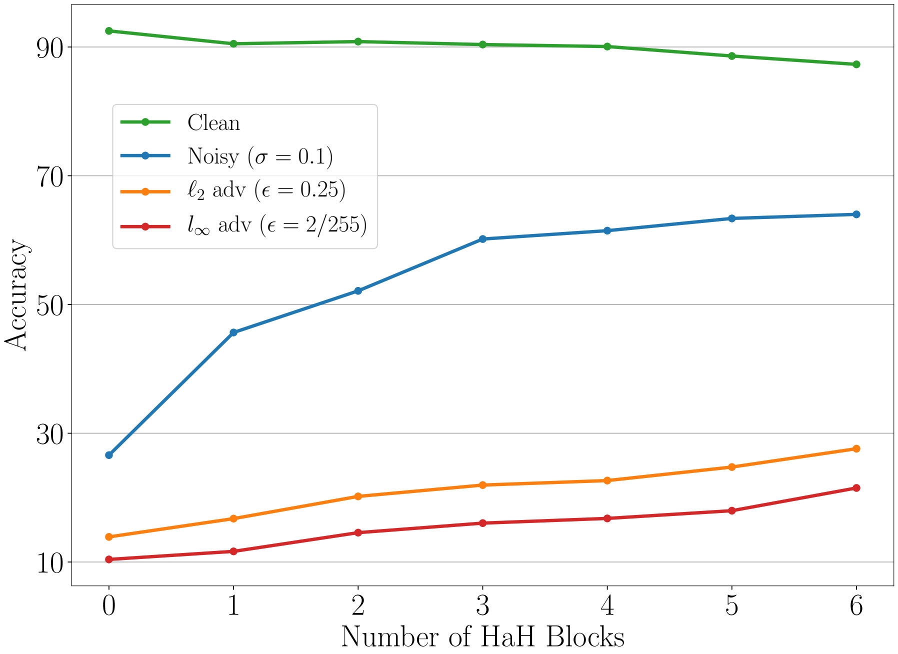

Figure 5.10: Ablation study for number of HaH blocks. Every additional HaH block contributes to the robustness of the model with a slight compromise on clean accuracy.

5.4 Conclusion

The preliminary results we obtained show the promise of layer-wise cost functions in controlling the features extracted by the intermediate layers and therefore improving the mainstream end-to-end paradigm in deep learning. The sparser and stronger activations achieved through our neuro-inspired approach to neuronal competition during training and inference clearly improve robustness against noise, common corruptions and adversarial perturbations than baseline models. Looking at the results using the CIFAR-10-C common corruptions dataset, we can say that the robustness attained by our approach, without any augmentation, is more general than that of adversarial training. We note that other recent bio-inspired adversarial defense approaches also came to similar conclusions [103].

Based on the promising results we obtained, we believe that a comprehensive analysis into enhancing end-to-end training using layer-wise cost functions is justified. Further research directions could include incorporating such techniques into a variety of architectures, training methods (including unsupervised and semi-supervised learning, and data augmentation) and applications. In particular, a logical next step is to combine data augmentation strategies (including adversarial training) with HaH architectures for robust machine learning.

Chapter 6

Conclusion

While the general trend in machine learning is in line with the observation that more data outperforms clever and hand-crafted solutions, over-reliance on data alone leads to problems of vulnerability and non-interpretability. Specifically, the susceptibility of neural networks to adversarial perturbations remains a key issue afflicting the area of machine learning and its deployment in safety critical systems. While the most common recourse to-date is to deploy adversarial training and its variants, this approach is still a long way off from providing the level of robustness that we expect from humans or from a vitally important automated system. Arguably, one of the key reasons behind this stagnation after many years of research is the fact that we still do not fully understand how adversarial training achieves the robustness it does. Although we can formulate the top-down minimax problem adversarial training operates on, the underlying properties it induces remain a black box. One of the ways in which we can alleviate this problem is by using structural elements whose principles can be explained and understood, which suppress adversarial perturbations at each step. We believe that it is worth pursuing this approach as an alternative and complement to a purely data-centric approach, in order to meet the intellectual and practical challenges left open by the current state of the art in deep learning.

Our work has been an attempt in this promising direction of using more principled building blocks to enhance the robustness and interpretability of deep neural networks. We have tried to pave the first steps towards a bottom-up approach, complementary to the mainstream approaches in machine learning today. Our defenses approach but do not exceed the level of robustness observed in adversarial training based methods. However, they show improvements in other important properties such as strength against common corruptions or interpretable feature usage. Thus, we hope that the techniques proposed in this dissertation motivate further research into this avenue of research, both for adversarial robustness and for other improvements in constructing DNNs.

Appendix A

Analysis of Adversarial Training

We depict below, our layer numbering schemes for the ResNet and VGG networks used in our experiments.

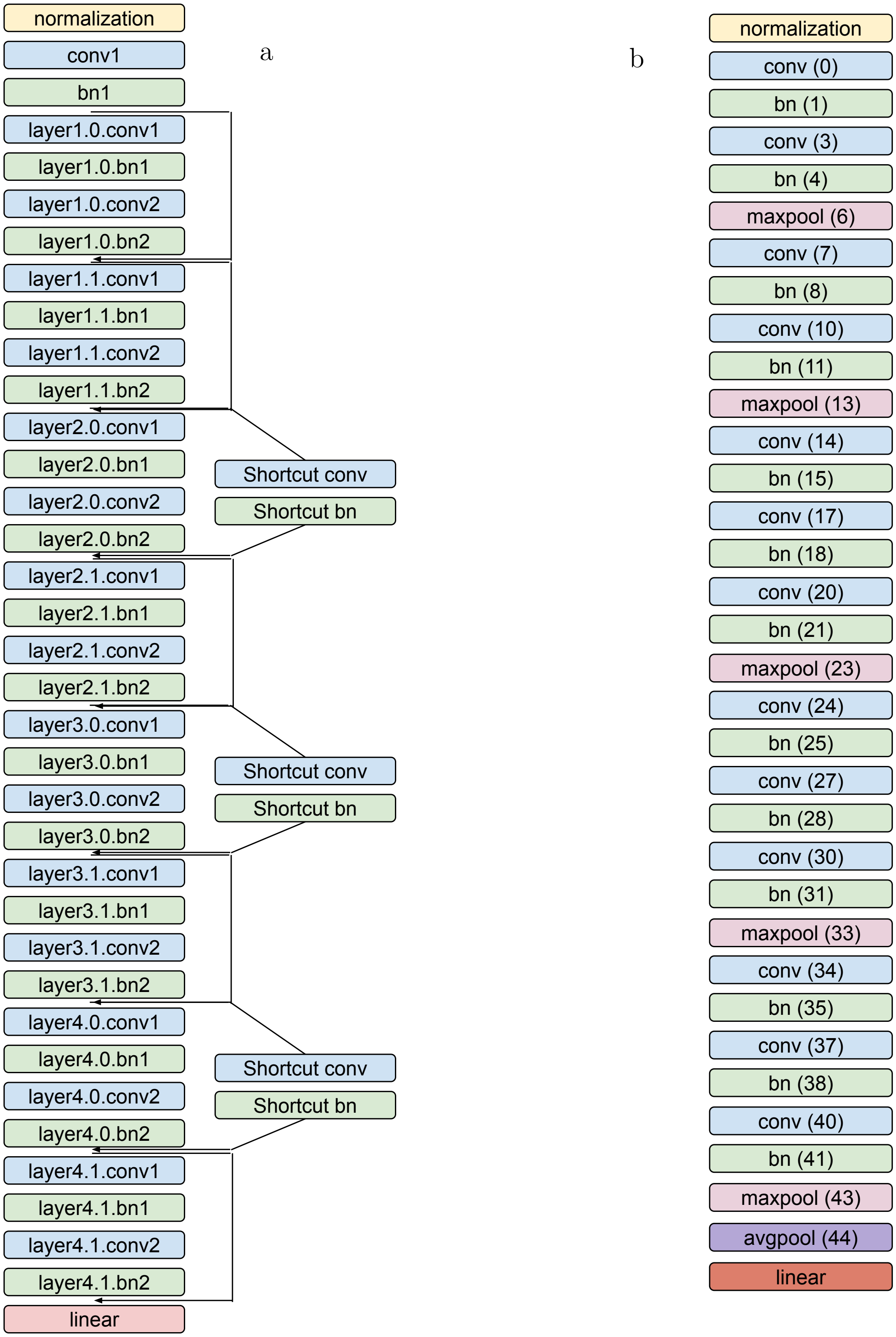

Figure A.1: Architectures and layer names for **(a)** ResNet **(b)** VGG

Appendix B

Sparse Encoding Defense Without Dropout

B.1 Block Diagram

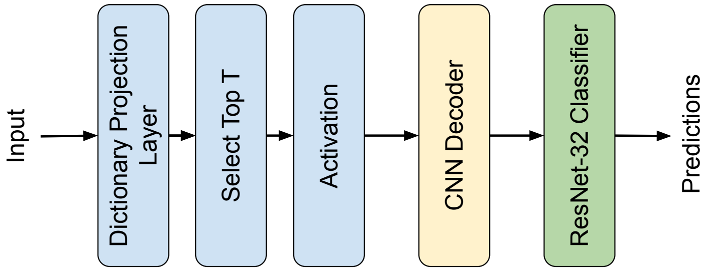

Figure B.1: A block diagram of our proposed defense (non-random)

Input images first go through the convolutional layer with the dictionary atoms being its filters. Then, for each patch representation, Top T coefficients in absolute value are kept while other coefficients are zeroed out. This is followed by the activation function described in Section 4.2.2.3. These quantized and highly sparse representations are then fed into a convolutional neural net based decoder which restores the image size. The outputs of the front end are the inputs of the ResNet-32 based classifier. Training of the decoder and the classifier are done simultaneously after the dictionary is learned and frozen.

B.2 Determining Attack Parameters

In order to determine the strongest attack settings against our defense, we test it with different attack parameters. In Table B.1, we report five white-box attack settings on our defense for different T values. For these attacks, every differentiable operation is differentiated. For the non-differentiable activation function, we take a smooth backward pass approximation. We also experiment with replacing it with identity in the backward pass but this results in weaker attacks (see Section B.4). For the baseline attack settings, we choose number of restarts $N_r = 10$, number of steps $N_s = 40$, and step size $\delta = 1/255$. We then change one or two settings at a time and report those in each column. In the third column, we report accuracies when gradients are propagated through the top U coefficients (rather than top T). For the fourth column, we increase N_s to 1000, reduce δ to 0.5 and N_r to 1 in order to keep computational complexity reasonable. For the fifth column, we increase N_r to 100 and keep $N_s = 40$. Finally, for sixth column, we propagate the gradients through the top U coefficients as well as increasing the number of restarts N_r to 100. For all values of T, last two columns' settings result in the strongest attacks. Increasing the number of restarts N_r to 100 especially, had the biggest impact on the accuracies.

By looking at the results in Table B.1, we determine to report on $T = 15$ in Table 4.5, since it represents a good trade off between clean and attacked accuracy. After the decision to use $T = 15$ for benchmark comparison in Table 4.5, we use the strongest attack for this defense setting, which is taking $N_r = 100$ and propagating gradients

	Clean	White-box (Baseline)	White-box ($U = 2T$)	White-box ($N_s = 1000$)	White-box ($N_r = 100$)	White-box ($U = 2T$ $N_r = 100$)
$T = 1$	81.81	44.79	48.26	59.91	34.89	37.83
$T = 2$	83.03	47.89	47.81	61.35	37.48	37.92
$T = 5$	83.24	47.11	42.21	60.37	36.66	32.83
$T = 10$	84.71	48.48	44.83	60.06	39.69	36.16
$T = 15$	85.45	46.89	44.55	57.13	38.43	37.33
$T = 20$	85.46	40.96	45.27	51.39	32.63	37.25

Table B.1: Accuracies for our defense method under different settings (CIFAR-10, L^∞ $\epsilon = 8/255$)

through the top $U = 2T = 30$ coefficients.

B.3 Choice of Hyperparameters and Training Settings

Our defense: We evaluate our defense on the CIFAR-10 dataset ($N = 32$), for which there are well-established benchmarks in adversarial ML. In our defense, we use 4×4 patches ($n = 4$) and an overcomplete dictionary with $L = 500$ atoms. The stride $S = 2$, so the encoder output is a $15 \times 15 \times 500$ tensor ($m = 15$, $L = 500$). The regularization parameter in Equation 4.4 is set to $\lambda = 1$, in the upper range of values resulting in convergence. The number of iterations in dictionary learning is chosen as 1000 to ensure convergence. The number of dictionary atoms L is chosen to be 10 times the ambient dimension of patches.

We test our defense for $T = 1, 2, 5, 10, 15$, and 20 with hyperparameter $\beta = 3$ for the threshold in Equation 4.6. We train the CNN-based decoder in supervised fashion in tandem with the classifier, using the standard cross-entropy loss. We use a cyclic learning rate scheduler [91] with a maximum learning rate of $\eta_{max} = 0.05$ for $T = 1, 2$

and $\eta_{max} = 0.02$ for $T = 5, 10, 15, 20$. In order to provide a consistent evaluation, we employ the ResNet-32 classifier used in [9] and train it for 70 epochs.

Benchmarks: For a fair comparison, we use the same ResNet-32 classifier architecture for the benchmarks. We train the PGD adversarially trained model from [9] with the same cyclic learning rate with $\eta_{max} = 0.05$ for 100 epochs. We train the model for TRADES defense with learning rate $\eta = 0.01$ for the first 50 epochs and then with $\eta = 10^{-3}$ for the next 50 epochs. For both PGD adversarially trained model and TRADES, training hyperparameters are $\epsilon = 8/255$, $\delta = 1/255$, $N_S = 10$, $N_R = 1$. Additionally for TRADES $\lambda_{\text{TRADES}} = 1/6$. We also report on naturally trained network (i.e., no defense). This network is also trained for 70 epochs with the same cyclic learning rate with $\eta_{max} = 0.05$.

B.4 Backward Pass Approximation to Activation

We try replacing the activation function with two different functions in the backward pass: identity and a smooth approximation (Figure B.2). We observe that the approximation with steepness= 4.0 results in the strongest attacks and use it in the backward pass of all reported attacks.

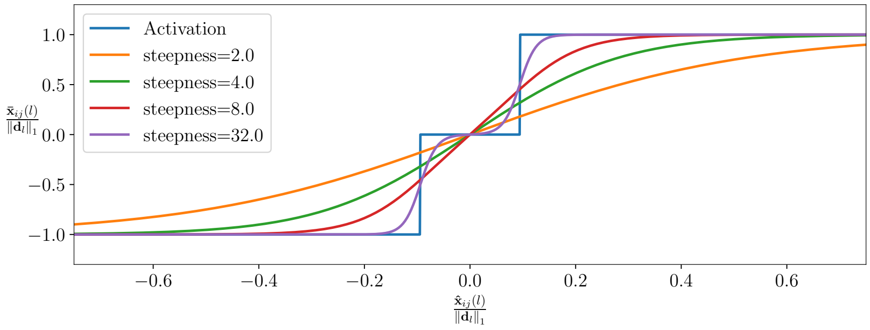

Figure B.2: Activation function and its backward pass smooth approximation with varying degrees of smoothness

B.5 Gradient Propagation Through Top U Coefficients

We test different settings for the hyperparameter U such as $U = 2T$, $U = 5T$, $U = 10T$ and find that, in general, $U = 2T$ results in the strongest attacks.

B.6 Gradient Smoothing

We test smoothing the gradients in each step by averaging them over 40 different points in the δ L^∞ neighborhood in each step of the attack. For computational feasibility, we reduce N_s to 20 and N_r to 1. In general, for the models we tried this on, this smoothing operation did not result in significantly stronger attacks. It decreased accuracies less than 0.5% while increasing the computational cost significantly. For this reason, we choose not to use this operation in the attacks reported.

B.7 Results for HopSkipJumpAttack

HopSkipJumpAttack [38] is a recently proposed decision-based attack. The results for this attack are reported in Table B.2. Similar to the last column of Table 4.5 the reported results are for the average L^2 norm of the perturbations. We observe that, like the decision boundary attack, our defense requires the highest average L^2 norm perturbations.

	HopSkipJumpAttack
	Average L^2 norm of successful attack images
Natural	$\overline{\|\|\mathbf{e}\|\|_2} = 0.31$
Adv. Training	$\overline{\|\|\mathbf{e}\|\|_2} = 2.08$
TRADES	$\overline{\|\|\mathbf{e}\|\|_2} = 2.00$
$T = 15$ (Ours)	$\overline{\|\|\mathbf{e}\|\|_2} = \mathbf{3.43}$

Table B.2: Results for HopSkipJumpAttack attack (CIFAR-10)

B.8 Results for Zeroth Order Optimization (ZOO) Based Attack

ZOO attack [36] is a query-based black-box attack where the gradients of the model with respect to each input pixel are approximated using numerical differentiation and then the attack is computed through stochastic coordinate descent. For computational complexity reasons we evaluate this attack on 100 randomly selected images. These do not include images that are wrongly classified by the corresponding model. The results for this attack are reported in Table B.3. Similar to the last column of Table 4.5 the reported results are for the average L^2 norm of the perturbations. We observe that the ZOO attack is unable to find adversarial examples for our defense. This is due to the attack computing gradients by changing one pixel at a time by $\pm\Delta x$. Since our front end is very insensitive to single pixel changes, the gradients cannot be calculated using such numerical differentiation techniques. We acknowledge that while this observation alone does not show that our defense is secure, it does mean that this particular type of attack is not applicable to our defense.

B.9 Effect of Attack Step Size

Nominally, we report results for when the step size $\delta = \epsilon/8$. For the L^∞ attack, when we use $N_s = 1000$, we decrease δ to $0.5/255$. We also test for $\delta = 2/255$ but this results in higher accuracies therefore is omitted in the tables. For the L^1 and L^2 bounded attacks, we try $\delta = \epsilon/20$ but this too results in weaker attacks.

B.10 Validation of Attack Code

To generate all attacks we use our own attack library and validate our results with the `foolbox` [106] and `torchattacks` [107] Python packages.

ZOO Attack		
	Attack success rate (%)	Average L^2 norm of successful attack images
Natural	100	$\overline{\|\|\mathbf{e}\|\|_2} = 0.13$
Adv. Training	100	$\overline{\|\|\mathbf{e}\|\|_2} = 0.89$
TRADES	100	$\overline{\|\|\mathbf{e}\|\|_2} = 0.93$
$T = 15$ (Ours)	7	$\overline{\|\|\mathbf{e}\|\|_2} = 0.04$

Table B.3: Results for ZOO attack (CIFAR-10)

Bibliography

[1] Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S. K. S., Ayan, B. K., Mahdavi, S. S., Lopes, R. G., et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022.

[2] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1625–1634, 2018.

[3] Gao, J., Lanchantin, J., Soffa, M. L., and Qi, Y. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 50–56. IEEE, 2018.

[4] Carlini, N. and Wagner, D. Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE security and privacy workshops (SPW)*, pp. 1–7. IEEE, 2018.

[5] Nicolae, M.-I., Sinn, M., Tran, M. N., Buesser, B., Rawat, A., Wistuba, M., Zantedeschi, V., Baracaldo, N., Chen, B., Ludwig, H., Molloy, I., and Edwards, B. Adversarial robustness toolbox v1.2.0. *CoRR*, 1807.01069, 2018. URL `https://arxiv.org/pdf/1807.01069`.

[6] Wu, T., Tong, L., and Vorobeychik, Y. Defending against physically realizable attacks on image classification. *arXiv preprint arXiv:1909.09552*, 2019.

[7] Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[8] Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial machine learning at scale. In *International Conference on Learning Representations (ICLR)*, 2017.

[9] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[10] Croce, F. and Hein, M. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2206–2216. PMLR, 13–18 Jul 2020.

[11] Andriushchenko, M., Croce, F., Flammarion, N., and Hein, M. Square attack: a query-efficient black-box adversarial attack via random search. In *European Conference on Computer Vision*, pp. 484–501. Springer, 2020.

[12] Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Olshausen, B. A. and Field, D. J. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision research*, 37(23):3311–3325, 1997.

[14] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[15] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419): 1140–1144, 2018.

[16] Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[17] Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[18] Berner, C., Brockman, G., Chan, B., Cheung, V., Dkebiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[19] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[20] Karras, T., Laine, S., and Aila, T. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4401–4410, 2019.

[21] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[22] Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[23] Allyn, B. The google engineer who sees company's ai as 'sentient' thinks a chatbot has a soul. *NPR Article*, 2022. URL `https://www.npr.org/2022/06/16/1105552435/google-ai-sentient`.

[24] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pp. 387–402. Springer, 2013.

[25] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

[26] Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[27] Burkart, N. and Huber, M. F. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research*, 70:245–317, 2021.

[28] Sutton, R. The bitter lesson.
`http://www.incompleteideas.net/IncIdeas/BitterLesson.html`, 2019.

[29] Guo, C., Rana, M., Cisse, M., and Van Der Maaten, L. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

[30] Yang, Y., Zhang, G., Katabi, D., and Xu, Z. ME-Net: Towards effective adversarial robustness with matrix estimation. In *International Conference on Machine Learning*, 2019.

[31] Bakiskan, C., Gopalakrishnan, S., Cekic, M., Madhow, U., and Pedarsani, R. Polarizing front ends for robust CNNs. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4257–4261. IEEE, 2020.

[32] Laidlaw, C., Singla, S., and Feizi, S. Perceptual adversarial robustness: Defense against unseen threat models. *arXiv preprint arXiv:2006.12655*, 2020.

[33] Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pp. 39–57. Ieee, 2017.

[34] Athalye, A., Engstrom, L., Ilyas, A., and Kwok, K. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

[35] Tramer, F., Carlini, N., Brendel, W., and Madry, A. On adaptive attacks to adversarial example defenses. *arXiv preprint arXiv:2002.08347*, 2020.

[36] Chen, P.-Y., Zhang, H., Sharma, Y., Yi, J., and Hsieh, C.-J. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pp. 15–26, 2017.

[37] Brendel, W., Rauber, J., and Bethge, M. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017.

[38] Chen, J., Jordan, M. I., and Wainwright, M. J. Hopskipjumpattack: A query-efficient decision-based attack. In *2020 ieee symposium on security and privacy (sp)*, pp. 1277–1294. IEEE, 2020.

[39] Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pp. 506–519, 2017.

[40] Tramèr, F., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[41] Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, pp. 39–57, 2017.

[42] Athalye, A., Carlini, N., and Wagner, D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pp. 274–283. PMLR, 2018.

[43] Carmon, Y., Raghunathan, A., Schmidt, L., Duchi, J. C., and Liang, P. S. Unlabeled data improves adversarial robustness. In *Advances in Neural Information Processing Systems*, pp. 11192–11203, 2019.

[44] Gowal, S., Rebuffi, S.-A., Wiles, O., Stimberg, F., Calian, D. A., and Mann, T. A. Improving robustness using generated data. *Advances in Neural Information Processing Systems*, 34:4218–4233, 2021.

[45] Rebuffi, S.-A., Gowal, S., Calian, D. A., Stimberg, F., Wiles, O., and Mann, T. Fixing data augmentation to improve adversarial robustness. *arXiv preprint arXiv:2103.01946*, 2021.

[46] Dai, S., Mahloujifar, S., and Mittal, P. Parameterizing activation functions for adversarial robustness. In *2022 IEEE Security and Privacy Workshops (SPW)*, pp. 80–87. IEEE, 2022.

[47] Zhang, H., Yu, Y., Jiao, J., Xing, E., El Ghaoui, L., and Jordan, M. Theoretically principled trade-off between robustness and accuracy. In *International conference on machine learning*, pp. 7472–7482. PMLR, 2019.

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

[49] Croce, F., Andriushchenko, M., and Hein, M. Provable robustness of ReLU networks via maximization of linear regions. *arXiv preprint arXiv:1810.07481*, 2018.

[50] Raghunathan, A., Steinhardt, J., and Liang, P. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

[51] Lecuyer, M., Atlidakis, V., Geambasu, R., Hsu, D., and Jana, S. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 656–672. IEEE, 2019.

[52] Cohen, J. M., Rosenfeld, E., and Kolter, J. Z. Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918*, 2019.

[53] Salman, H., Li, J., Razenshteyn, I., Zhang, P., Zhang, H., Bubeck, S., and Yang, G. Provably robust deep learning via adversarially trained smoothed classifiers. In *Advances in Neural Information Processing Systems*, pp. 11289–11300, 2019.

[54] Dapello, J., Marques, T., Schrimpf, M., Geiger, F., Cox, D., and DiCarlo, J. J. Simulating a primary visual cortex at the front of cnns improves robustness to image perturbations. *Advances in Neural Information Processing Systems*, 33:13073–13087, 2020.

[55] Feinman, R., Curtin, R. R., Shintre, S., and Gardner, A. B. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[56] Xu, W., Evans, D., and Qi, Y. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[57] Grosse, K., Manoharan, P., Papernot, N., Backes, M., and McDaniel, P. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 2017.

[58] Pang, T., Du, C., Dong, Y., and Zhu, J. Towards robust detection of adversarial examples. *Advances in Neural Information Processing Systems*, 31, 2018.

[59] Bhagoji, A. N., Cullina, D., Sitawarin, C., and Mittal, P. Enhancing robustness of machine learning systems via data transformations. In *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–5. IEEE, 2018.

[60] Hein, M. and Andriushchenko, M. Formal guarantees on the robustness of a classifier against adversarial manipulation. *Advances in neural information processing systems*, 30, 2017.

[61] Moosavi-Dezfooli, S.-M., Fawzi, A., Uesato, J., and Frossard, P. Robustness via curvature regularization, and vice versa. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9078–9086, 2019.

[62] Marblestone, A. H., Wayne, G., and Kording, K. P. Toward an integration of deep learning and neuroscience. *Frontiers in computational neuroscience*, pp. 94, 2016.

[63] Nayebi, A. and Ganguli, S. Biologically inspired protection of deep networks from adversarial attacks. *arXiv preprint arXiv:1703.09202*, 2017.

[64] Cekic, M., Bakiskan, C., and Madhow, U. Neuro-inspired deep neural networks with sparse, strong activations. *arXiv preprint arXiv:2202.13074*, 2022.

[65] Carlini, N. and Wagner, D. Adversarial examples are not easily detected: Bypassing ten detection methods. In *ACM Workshop on Artificial Intelligence and Security*, pp. 3–14, 2017.

[66] Liu, C., Salzmann, M., Lin, T., Tomioka, R., and Süsstrunk, S. On the loss landscape of adversarial training: Identifying challenges and how to overcome them. *Advances in Neural Information Processing Systems*, 33:21476–21487, 2020.

[67] He, W., Li, B., and Song, D. Decision boundary analysis of adversarial examples. In *International Conference on Learning Representations*, 2018.

[68] Tian, Q., Kuang, K., Jiang, K., Wu, F., and Wang, Y. Analysis and applications of class-wise robustness in adversarial training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1561–1570, 2021.

[69] Kanai, S., Yamada, M., Takahashi, H., Yamanaka, Y., and Ida, Y. Smoothness analysis of adversarial training. *arXiv preprint arXiv:2103.01400*, 2021.

[70] Xing, Y., Song, Q., and Cheng, G. On the algorithmic stability of adversarial training. *Advances in Neural Information Processing Systems*, 34, 2021.

[71] He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[72] Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[73] Rice, L., Wong, E., and Kolter, Z. Overfitting in adversarially robust deep learning. In *International Conference on Machine Learning*, pp. 8093–8104. PMLR, 2020.

[74] Frankle, J., Schwab, D. J., and Morcos, A. S. Training batchnorm and only batchnorm: On the expressive power of random features in cnns. *arXiv preprint arXiv:2003.00152*, 2020.

[75] Yang, H., Wen, W., and Li, H. Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. *arXiv preprint arXiv:1908.09979*, 2019.

[76] Lin, J., Gan, C., and Han, S. Defensive quantization: When efficiency meets robustness. *arXiv preprint arXiv:1904.08444*, 2019.

[77] Rakin, A. S., Yi, J., Gong, B., and Fan, D. Defend deep neural networks against adversarial examples via fixed and dynamic quantized activation functions. *arXiv preprint arXiv:1807.06714*, 2018.

[78] Khalid, F., Ali, H., Tariq, H., Hanif, M. A., Rehman, S., Ahmed, R., and Shafique, M. Qusecnets: Quantization-based defense mechanism for securing deep neural network against adversarial attacks. In *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 182–187. IEEE, 2019.

[79] Panda, P., Chakraborty, I., and Roy, K. Discretization based solutions for secure machine learning against adversarial attacks. *IEEE Access*, 7:70157–70168, 2019.

[80] Chen, J., Wu, X., Rastogi, V., Liang, Y., and Jha, S. Towards understanding limitations of pixel discretization against adversarial attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 480–495. IEEE, 2019.

[81] Zhang, Y. and Liang, P. Defending against whitebox adversarial attacks via randomized discretization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 684–693. PMLR, 2019.

[82] Marzi, Z., Gopalakrishnan, S., Madhow, U., and Pedarsani, R. Sparsity-based defense against adversarial attacks on linear classifiers. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 31–35. IEEE, 2018.

[83] Gopalakrishnan, S., Marzi, Z., Madhow, U., and Pedarsani, R. Combating adversarial attacks using sparse representations. *arXiv preprint arXiv:1803.03880*, 2018.

[84] Gopalakrishnan, S., Marzi, Z., Cekic, M., Madhow, U., and Pedarsani, R. Robust adversarial learning via sparsifying front ends. *arXiv preprint arXiv:1810.10625*, 2018.

[85] Blakemore, C., Carpenter, R. H., and Georgeson, M. A. Lateral inhibition between orientation detectors in the human visual system. *Nature*, 228(5266):37–39, 1970.

[86] Prescott, S. A. and De Koninck, Y. Gain control of firing rate by shunting inhibition: Roles of synaptic noise and dendritic saturation. *Proceedings of the National Academy of Sciences*, 100(4):2076–2081, 2003.

[87] Prenger, R., Wu, M. C.-K., David, S. V., and Gallant, J. L. Nonlinear V1 responses to natural scenes revealed by neural network analysis. *Neural Networks*, 17(5-6): 663–679, 2004.

[88] Mairal, J., Bach, F., Ponce, J., and Sapiro, G. Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pp. 689–696, 2009.

[89] Elad, M. and Aharon, M. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image processing*, 15(12):3736–3745, 2006.

[90] Carlini, N., Athalye, A., Papernot, N., Brendel, W., Rauber, J., Tsipras, D., Goodfellow, I., Madry, A., and Kurakin, A. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*, 2019.

[91] Smith, L. N. Cyclical learning rates for training neural networks. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464–472. IEEE, 2017.

[92] Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv:1905.11946*, 2019.

[93] Wong, E., Rice, L., and Kolter, J. Z. Fast is better than free: Revisiting adversarial training. *arXiv preprint arXiv:2001.03994*, 2020.

[94] Fukushima, K., Miyake, S., and Ito, T. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):826–834, 1983. doi: 10.1109/TSMC.1983.6313076.

[95] Amato, G., Carrara, F., Falchi, F., Gennaro, C., and Lagani, G. Hebbian learning meets deep convolutional neural networks. In Ricci, E., Rota Bulò, S., Snoek, C., Lanz, O., Messelodi, S., and Sebe, N. (eds.), *Image Analysis and Processing – ICIAP 2019*, pp. 324–334, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30642-7.

[96] Carandini, M. and Heeger, D. J. Normalization as a canonical neural computation. *Nature Reviews Neuroscience*, 13(1):51–62, 2012.

[97] Burg, M. F., Cadena, S. A., Denfield, G. H., Walker, E. Y., Tolias, A. S., Bethge, M., and Ecker, A. S. Learning divisive normalization in primary visual cortex. *PLOS Computational Biology*, 17(6):e1009028, 2021.

[98] Li, Z., Brendel, W., Walker, E., Cobos, E., Muhammad, T., Reimer, J., Bethge, M., Sinz, F., Pitkow, Z., and Tolias, A. Learning from brains how to regularize machines. *Advances in neural information processing systems*, 32, 2019.

[99] Hoyer, P. O. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5(9), 2004.

[100] Pintor, M., Roli, F., Brendel, W., and Biggio, B. Fast minimum-norm adversarial attacks through adaptive norm constraints. *Advances in Neural Information Processing Systems*, 34, 2021.

[101] Croce, F., Andriushchenko, M., Sehwag, V., Debenedetti, E., Flammarion, N., Chiang, M., Mittal, P., and Hein, M. Robustbench: a standardized adversarial robustness benchmark. *arXiv preprint arXiv:2010.09670*, 2020.

[102] Hendrycks, D. and Dietterich, T. G. Benchmarking neural network robustness to common corruptions and surface variations. *arXiv preprint arXiv:1807.01697*, 2018.

[103] Machiraju, H., Choung, O.-H., Herzog, M. H., and Frossard, P. Empirical advocacy of bio-inspired models for robust image recognition. *arXiv preprint arXiv:2205.09037*, 2022.

[104] Kireev, K., Andriushchenko, M., and Flammarion, N. On the effectiveness of adversarial training against common corruptions. *arXiv preprint arXiv:2103.02325*, 2021.

[105] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pp. 618–626, 2017.

[106] Rauber, J., Zimmermann, R., Bethge, M., and Brendel, W. Foolbox native: Fast adversarial attacks to benchmark the robustness of machine learning models in pytorch, tensorflow, and jax. *Journal of Open Source Software*, 5(53):2607, 2020. doi: 10.21105/joss.02607. URL `https://doi.org/10.21105/joss.02607`.

[107] Kim, H. Torchattacks: A pytorch repository for adversarial attacks. *arXiv preprint arXiv:2010.01950*, 2020.