Robust Convolutional Neural Networks with Resource Constraints

Charles Marshall

Supervised by Yiannis Georgiou & Denis Trystram

A thesis presented for the degree of Master of Informatics
Defended before a jury composed of:
Massih Reza-Amini
Martin Heusse
Denis Trystram
Yiannis Georgiou
Xavier Alameda-Pineda

Université Grenoble Alpes
ENSIMAG

15.06.2020

Acknowledgements

I would like to thank my supervisors Yiannis Georgiou and Denis Trystram for their guidance and support, as well as the entire team at Ryax technologies for providing me with a safe, productive environment to conduct my research during a period of quarantine.

Contents

List of Figures

Robust Convolutional Neural Networks with Resource Constraints

Charles Marshall

Abstract

State of the art research in artificial intelligence has shown the capability of Convolutional Neural Networks (CNNs) to perform exceptionally well in visual tasks. Furthermore, recent work in compression has proven that CNNs can be pruned to use only a fraction of the millions of parameters, with negligible loss in standard accuracy. However, as serverless and edge computing become the norm in industrial software solutions, additional considerations are required to deploy deep neural networks on embedded and Internet of Things (IoT) devices. Namely, the security of machine learning systems is vital especially in tasks where failures on the part of the model are potentially catastrophic. In this work we study the potential for compressed CNNs that are robust to *adversarial attacks* and can be deployed with confidence in environments with limited computing resources. We modify existing algorithms to develop a one-shot training procedure for robust Empirical Risk Minimization (ERM) with compression that runs on a single GPU. We then evaluate our models on a number of different adversarial attacks to show our models are robust to various threats, and finally we conduct a performance evaluation on several different hardware configurations. We show that robustness is still achievable in resource constrained scenarios despite being more complicated and seemingly at odds with the goals of model compression.

1. Introduction

In recent years the performance of convolutional neural networks (CNNs) has rivaled human level performance on vision tasks such as image classification and object recognition. Standard CNNs are computationally intensive, memory greedy, and consume lots of energy. However, modern algorithms are capable of *compressing* CNNs and other types of Deep Neural Networks (DNNs) to alleviate these three blockades. Ensuing demand from different fields and application types has called for the deployment of CNNs on a variety of embedded systems and hardwares [1]; moving industrial Machine Learning (ML) applications further away from the cloud towards resource-scarce devices at network edge.

Adjacent to this trend is the large and rapidly growing body of work on *adversarial robustness*: a term referring to a model's ability to defend against a variety of direct attacks that aim to deteriorate their performance. It is well known that CNNs and other Deep Learning (DL) models are extremely *brittle* [2, 3], and can be reduced to random classifiers by attacks that add small, humanly imperceptible perturbations to the input data. This is particularly worrisome in the context of security and time sensitive tasks, where mistakes on the part of the algorithm could be harmful to software systems, businesses, or individuals. Consider the case in which an autonomous vehicle fails to identify a single pedestrian, or when an attacker fools a biometric authentication system and gains access to private data.

As computational load continues to disperse to embedded systems at network edge, we are revealing additional points of attack. Taking measures to defend against such *failure modes* becomes increasingly difficult as attackers gain avenues for exploitation. This work aims to address this problem by studying the known and unknown trade-offs in adversarial robustness when paired with model compression. We raise the following key question:

How can we ensure robustness for convolutional neural networks in resource-scarce scenarios?

This question is confronted with a thorough examination of robust CNNs beginning with the data and training, and reinforced by a series of benchmarked evaluations. We begin with some background on the motivations for this research followed by our contributions, before discussing the state of the art in chapter 2, our implementation details in chapter 3, and our experiments in chapter 4.

1.1 Deep Learning at Network Edge

The actively shifting data processing paradigm towards IoT, fog, and edge challenges deep learning experts to design systems that are smaller, faster, and more energy

efficient. This has garnered large amounts of attention from the machine learning community in recent years [4], with some work dating back over a decade. Like other disciplines within Artificial Intelligence (AI), solutions to this family of problems originate from rule-based systems, and have evolved to use modern non-parametric DL models [5, 6, 7] and methods such as Neural Architecture Search (NAS) [8].

1.1.1 Model Compression

As optimization and machine learning make their way into more industries and application types, the engineering workflow to deploy models on target devices has become ubiquitous, albeit not standardized. This kind of production pipeline typically involves training a model in the cloud before conducting analyses to reduce its computational footprint, to speed up the forward pass, and to reduce bit-width to ease memory pressure. Further analyses regarding deployment, fine-tuning, and computation sharing can follow as well. All of these components are research fields in their own right [1], and highlight the difficulty to bring AI into industrial production.

This work focuses specifically on training, and evaluating models for robustness as well as runtime performance. The latter can be further improved by considering other previously mentioned techniques that do not fall within the scope of this project but are direct continuations. We argue that CNNs are overparameterized for many common vision tasks, and thus we can directly remove a large quantity of weights without grossly compromising performance. In particular, we will study how this claim is affected in the adversarial setting.

1.1.2 Adversarial Robustness

The term *robust* refers to the invariability of a model when presented with data that has been altered or *attacked* in some fashion. Such altered data is referred to as an *adversarial example*: a carefully crafted data input, practically (or fully) indistinguishable from the original data, yet that causes a classifier to misclassify that example while the original data was correctly classified. Adversarial examples are a well studied phenomenon with some very popular examples making their way into mainstream media[1]

Adversarial examples are puzzling from a human point of view, as they outline shortcomings of data and DL models that do not exist in humans (our perception of images does not waver when only 1 or several pixels have been altered). From a security point of view, deploying more models industrially means exposing more points of attack; where attackers can deliberately try to fool a model into failure or into giving some targeted response. These are called targeted and untargeted attacks, respectively, and are possible in some capacity no matter the amount of knowledge the attacker has about the model. An attack with full knowledge of the model is referred to as *white-box*, with its counterpart being *black-box* wherein nothing is known of the architecture or parameters of the model.

Integrating DL into real-time and real-world systems may call for a certain level of robustness depending on the use case. Furthermore, it is believed that training robust CNNs built to withstand attacks have the inherent trade-off of being less

[1]See: `https://www.youtube.com/watch?v=piYnd_wYlT8`

accurate [9]. To be confident in the ability of DL models deployed at the edge and in resource scarce environments, robustness and compression must be studied in unison.

1.2 Contributions

In this work we will study robust optimization of deep CNNs with resource constraints. We contextualize robustness with human level observations and draw from recent state of the art works to most effectively evaluate our models. We aim to quantify the effect of compression techniques on robust deep learning and to help inform the machine learning production life-cycle in resource constrained settings. We make the following contributions:

- We propose modifications to an existing algorithm to train compressed, robust models that withstand strong adversarial attacks with a fraction of the original parameters. Our algorithm is one-shot in that it requires no pre-training or fine tuning and runs on a single GPU.
- We train a variety of networks on the CIFAR10 and CIFAR100 datasets and find that we can largely take advantage of overparameterization for these datasets as we are able to find good solutions to the saddle point formulation at increasingly small compression ratios.
- We evaluate our models on a number of attacks and image classification benchmarks. Our models achieve accuracy comparable to the state of the art despite using fewer parameters and a less computationally demanding training procedure.
- We study the performance of our models on different hardware configurations ranging in computing power. We find that weight sparsity yields small performance boosts to our models and requires few programmatic alterations.

2. Related Work

In this section we review the necessary background information, provide comments on related works, and contextualize our contributions in the state of the art.

2.1 Adversarial Robustness

Several years ago as DL rose to become the norm in visual computing, the expressiveness of CNNs (and neural networks in general) was shown to be both a blessing and a curse. On one hand the learned representations are complex and allow CNNs to accurately describe some data distribution, yet on the other hand these representations are discontiguous and do not necessarily match up with what we perceive as humans.

More concretely, given some artificial neural network trained on benign examples from some dataset $(x, y) \sim \mathcal{D}$, there exists $\hat{x} = x + \delta$, with δ being a small pixel level perturbation, such that the network fails to correctly classify $\hat{x}$, and $\hat{x}$ is indistinguishable from x. This property, first dubbed a *blind spot* [10] (see appendix A.1) and later an *adversarial example*, opened the eyes of many in the field to explore the shortcomings of deep learning systems [11, 12]. Next we give the intuition behind adversarial examples as a phenomenon before categorizing different *attack* methods and reviewing common training techniques to mitigate risk that such examples exist at inference time.

2.1.1 Adversarial Examples

Some of the most influential and important research in adversarial robustness has come from Madry et al [13, 3]. Perhaps most fundamental is the perspective that neural networks have such high computational capacity that they are able to pick up on features that are either imperceptible or indiscernible to humans, who regard such features as *semantically insignificant* towards the classification of a given example.

This means for instance that in some dataset $\mathcal{D}$, all images of a certain class may have a single pixel off in the top corner of the image which happens to hold similar value across all training examples, thus making it an important representation of that class in the learned parameters of some classifier trained on $\mathcal{D}$. This notion characterizes a *non-robust feature*, while semantically human-meaningful features are called *robust features* (See appendix A for a more concrete definition).

Due to the highly predictive nature of non robust features we are able to achieve high levels of standard accuracy relying solely on them, yet such models are fragile, and can be fooled with adversarial examples crafted by making subtle manipulations to the data. This implies that robustness and standard accuracy are inherently at

odds and need careful consideration when preparing ML systems for real world scenarios.

Honing in on semantically significant, robust features makes models more *interpretable* as a by-product. In this work we define interpretability as the ability for a domain expert to interpret the feature representations of a CNN. This type of interpretability has been shown for robust models through visualization tools like *gradcam*, and is becoming popularized with new software packages like *captum* [14]. This type of model-domain expert interaction is forecasted to be normal in scenarios where DL systems are deployed industrially to further automate previously human tasks, and to enforce some level of accountability to the automated decision making process.

Ensuring robustness and interpretability of this sort involves finding ways to leverage both robust and non robust features in a way that verifiably fits the use case. Researchers have suggested encoding *priors* into our training algorithms, but there exists very little research to this end. This work does not experiment with any such priors, but operates under the theoretical framework of [3]; treating adversarial examples as a feature present in any static dataset. We will train models in ways that attempt to capture and leverage robust features in particular.

2.1.2 Training a Robust Classifier

In many DL tasks, training a classifier uses *Empirical Risk Minimization* (ERM): $\min \underset{(x,y)\sim\mathcal{D}}{\mathbb{E}} \mathcal{L}_\theta(x, y)$. This principle works in practice but enforces no distinction between robust and non-robust features. To train a robust classifier, we must account for the *known* presence of non-robust features in $\mathcal{D}$. This is done by introducing an adversary at training time [9] who can perturb inputs with some attack, or threat model which we will denote $\mathcal{A}$.

The adversary directly impedes ERM as they can easily perturb the highly representative non-robust features, with the goal of maximizing the loss of the classifier. Even an unsophisticated adversary can leverage non-robust features with ease and completely destroy the accuracy of the standard model. In this work, we use the following robust ERM formulation from [2] to train robust classifiers:

$$\min \underset{(x,y)\sim\mathcal{D}}{\mathbb{E}} \Big[\max_{\delta\in\Delta} \mathcal{L}_\theta\big(x+\delta, y\big)\Big] \tag{1}$$

In this formulation, labeled examples $x \in \mathbb{R}^d$, $y \in \{1...k\}$ are sampled from a distribution $\mathcal{D}$ and used to train a classifier with parameters $\theta \in \mathbb{R}^d$ with loss function $\mathcal{L}$; we will consider this to be the cross entropy loss of a deep neural network.

Additionally, $x + \delta$ denotes an input that has been perturbed by the adversary by some bounded quantity $\delta \in \Delta$. In this case, $\Delta = \{\delta \in \mathbb{R}^d \mid ||\delta||_p \leq \epsilon\}$ is a set of ℓ_p bounded perturbation quantities bounded by parameter ϵ which will later be discussed.

The formulation is a *two player game*, where on one hand a classifier wants to abide by ERM, while on the other the adversary is trying to maximize their effect on the input. Problems of this sort are referred to as *saddle point* problems and are structurally more complex to optimize, especially in the context of DNNs. We will

explore consequences of the saddle point formulation to a greater degree in section 4.1.2.

Unless otherwise stated we train our classifiers with a **first order adversary**, implying the adversary has full knowledge of the gradients and loss of the model at training time. This is the standard approach to the inner maximization problem in equation 1 but is known to be incomplete and does not encompass *verifiable robustness* which we will discuss later in this chapter. We first define our first order adversary:

Fast Gradient Sign Method (FGSM)

First proposed by [10], FGSM is an ℓ_p bounded adversary that can compute an adversarial example as follows:

$$x + \epsilon \times sgn\big(\nabla_x \mathcal{L}(\theta, x, y)\big) \tag{2}$$

Where ϵ is the maximum step size in the direction of the loss for a given example. FGSM is thus a *one-step* attack that uses the gradient feedback from the model to inform its perturbation, hence "first-order". It has been shown that FGSM is effective in training networks robust to small, first-order adversarial threats at test time.

It is difficult to train models robust to different threat models and furthermore the set of all possible threat models is unbounded. These issues complicate the field of adversarial robustness, as many methods exist for different threat models and attacks. FGSM is the baseline adversary but will not be further discussed as we will adopt the more effective, iterative version:

Projected Gradient Descent (PGD)

PGD is the predominant method for robust constrained optimization and is essentially an iterative FGSM attack where for t iterations, we solve:

$$x^{i+1} = \Pi_{x+\Delta}\left[x^i + \epsilon \times sgn\big(\nabla_x \mathcal{L}(\theta, x, y)\big)\right] \tag{3}$$

Although there exist stronger adversaries[1] [15], they are more computationally expensive and less understood. PGD is a sophisticated first-order adversary shown to be good algorithm to employ at training time and will be used thoroughly in this work both for training models and evaluating empirical robustness.

2.2 Types of Robustness

We previously introduced the predominant attack model used both for training and evaluating robust models. We will continue to call on Eq. 3 later in this work but it is important to acknowledge the broad landscape of robustness. The first distinction is that our first order adversary and its variants lend themselves to *empirical robustness*, but give no certifiable guarantee of robustness to a variety

[1]The second-order adversary in particular which uses second-order derivatives of the target model to perturb inputs. This type of adversary is extremely uncommon in the literature and is not directly addressed by this work.

of attacks. More recent works propose *verifiable* methods to ensure robustness including linear relaxations of the non-linear activations in a neural network, into convex sets. Other methods include randomized smoothing and Gaussian noise augmentation [16, 17]. Using verifiably robust training techniques has the benefit of making security guarantees for a model, but are far more computationally intensive than non-verifiable methods and thus are not considered in the training of our models.

Empirical and verifiable methods for robust machine learning are used at both train and test time. While there is now some cohesion in the literature, choosing and evaluating the proper threat model is not straightforward, is use case dependent, and is a vital developmental step. Most modern adversarial attacks use the ℓ_p norm to measure data perturbation, where $p \in \{1, 2, \infty\}$. This is common in the academic literature but is not a requirement for a real world adversary who can use other types of perturbations to craft adversarial examples [18].

Our analysis will include a variety of threat models, most of which are based on ℓ_p bounded perturbations. We do not use any verifiably robust methods for training models, but even without this our evaluations show that we can ensure robustness to an extent against many attacks that were not seen at training time.

2.3 Model Compression

The storage and computation needed to operate deep neural networks makes them unsuitable for resource scarce devices. This problem can be solved from a number of different angles all of which focus on finding compact representations for dense networks with many layers and parameters. We momentarily take a step back from robustness to review the state of the art in model compression and how it fits in to this research.

2.3.1 The Lottery Ticket Hypothesis

We first review a principle known as the lottery ticket hypothesis [19] which proposes the following[2]:

A randomly-initialized, dense neural network contains a sub-network that is initialized such that when trained in isolation it can match the test accuracy of the original network after training for at most the same number of iterations.

This phenomenon could not be true without the fact that many standard neural networks are *overparameterized*; meaning they are operating over more information than need be for the task being asked of them. Many parameters in neural networks are redundant and thus the network could perform identically with less parameters. It then follows that the training time of such a *sub-network* could be less than that of the full network. In fact, often a large quantity of weight parameters in neural networks tend to zero, meaning they have little to no influence on the algorithm (See appendix B). Works such as [20, 6] have studied these phenomena extensively in the standard test setting, yet little is known of how these properties change in the adversarial setting.

[2]Cited directly from [19] (See appendix B for additional information)

2.3.2 Compression Methods

There exist three main approaches to condensing the representations and data structures in deep neural networks:

Pruning

The most common method in literature and practice is known as *pruning*, and involves directly removing channels, filters, or weights from the network according to some *policy*. Some methods use a probabilistic policy [5], others directly search for the best parameters to remove [20], and some use a meta-learning policy akin to *Neural Architecture Search*(NAS) [21, 22].

As discussed previously and further demonstrated by our experiments in appendix B.1, it is possible in the benign setting to find sub-networks that perform equally to their dense counterparts while using a fraction of the parameters. Finding such a sub-network is beneficial in that it reduces the complexity of both a forward and backward pass through the network, which can lead to speed improvements at training and inference time depending on the pruning policy chosen. Additionally, pruning can easily be combined with other compression methods to reduce the storage capacity needed for a DNN. This work will focus primarily on pruning weights of CNNs but could be extended without any overhaul to include other compression strategies.

Quantization

More central to reducing the storage capacity needed for neural is quantization; a compression method which aims to reduce the bit-width of the data structures and weights used by the model. Similar to pruning, some methods are straightforward, and try to represent every numerical value using less bits, or even binary representations, while more recent methods use meta-learning strategies to search for optimal quantization strategies [23, 24, 25]. Reducing the bit-width of tensors in a CNN allows the algorithm to simplify the floating point operations needed to do a forward pass, for example binary matrix multiplication can speed up computation by a factor of 7 compared to 32 bit floating point numbers [4].

Tensor Factorization

Decomposing the matrices that store the parameters of CNNs can have some noticeable benefits. First, decompositions can be directly built into the architecture of the network to forego some computation. This is common in networks designed for resource scarce devices [26, 27, 28], and is not a direct compression method but should be mentioned in the same breath. These types of separable convolutions and tensor factorizations are unrelated, yet compatible with other hybrid architectural methods such as [29].

Similar to the other compression techniques, the way in which tensors get decomposed can either be enforced by the optimization process [30], or learned separately from the training of the model.

2.4 Robust Model Compression

Most in line with this work are the few existing works that directly combine a compression policy with adversarial optimization, in an effort to preserve robustness under relevant real world conditions [31, 32]. We draw directly from the state of the art in this intersectional field to inform and motivate our methodology and evaluation techniques.

In [33], a systematic evaluation of different pruning schemes was conducted due to the finding that weight pruning is essential when compressing CNNs in an adversarial setting. With similar results in [16], the key insights are that network capacity fundamentally increases in the adversarial setting, and that training a highly performant compressed adversarial network from scratch may not be feasible. This complicates the lottery ticket hypothesis in the adversarial setting. We will draw from the methodology of these works from a compression perspective by using an *irregular* pruning scheme which was found to be most useful. Additionally, we conduct some similar evaluations in a both a black box and white box setting, and will tie our results back in with the lottery ticket hypothesis and related state of the art work.

In [34], a *unified optimization framework* is proposed to combine pruning, quantization, and factorization into the optimization procedure in addition to adversarial training with a first order adversary. Their results along with the results from [31] provide good reason to believe that compression and adversarial machine learning can in fact be carefully combined to yield good performance. This goes against the fact that both of these features on their own can deteriorate a models performance and suggests more work in model architectures could be needed to find increasingly robust feature representations.

Some of the previously cited works in this section employ ADMM [35] to solve both optimization problems at training time. ADMM splits the optimization objective and solves for compression and robustness separately and iteratively. ADMM has shown to be effective in similar settings to ours and thus will be used to solve our optimization goal. As we will see, we alter the learning objective to suit our particular problem and we experiment with three different compression techniques.

Lastly, the large body of work on adversarial machine learning contains many evaluation techniques both empirical and verifiable. This makes the evaluation process incredibly important which is why the community has popularized a few benchmarks and tests to properly evaluate robust models. On the other hand there do not exist any standard evaluations for works also dealing with compression, and many of the evaluations in past works are incomplete. We will take the advice from the security community [36, 37] to extensively evaluate our work using various attacks in the black box, white box, and transfer setting.

3. Methodology

In this section we will discuss the formulation to train our CNNs: pruning weights while preserving robustness through the saddle point formulation. Similar to some works discussed in the previous chapter, we decompose our optimization objective and solve with *Alternating Direction Method of Multipliers* (ADMM).

Throughout the rest of this paper we will be considering a CNN with N layers. We first have the global optimization schema which draws from what we have seen in Eq 1:

$$\min \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \Big[\max_{\delta\in\Delta} \mathcal{L}_\theta\big(x+\delta, y\big)\Big] + \sum_{i=1}^{N} g(\theta_i') \tag{4}$$

Typical in ADMM, we can impose to the above that $\theta = \theta'$ and will use projected gradient descent from Eq 3 to solve the inner maximization which we discuss following. Furthermore, $g(\theta)$ is an indicator function to assess the sparsity of the network through each layer:

$$g(\theta) = \begin{cases} 0, & \text{if } \theta \in S \\ \infty, & \text{otherwise} \end{cases} \tag{5}$$

Drawing from [33] we can then define the standard pruning scheme used in this work[1], in which we limit the total amount of parameters in the CNN to some integer value k. That is, we define the set S as follows:

$$\theta \in S := \{\, \theta \mid ||\theta||_0 \leq k\} \tag{6}$$

Where $||\theta||_0$ denotes the amount of weights having value 0 at any layer of the network, and k can be treated either as a hyperparameter or as a part of the optimization scheme in the training algorithm which we will elaborate on in the following section. To solve this problem with ADMM, we formulate the augmented Lagrangian [35]:

$$\mathcal{L}(\theta, \theta', u) = \min \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \Big[\max_{\delta\in\Delta} \mathcal{L}_\theta\big(x+\delta, y\big)\Big] + \langle u, \theta - \theta'\rangle + \frac{\rho}{2}||\theta - \theta'||_2^2 + g(\theta') \tag{6}$$

3.1 Robust Training with Compression

In this section we describe our training algorithm using Eq. 6, in which u is a vector of Lagrangian multipliers to help enforce the equality of $\{\theta, \theta'\}$, $\rho > 0$ is

[1]Known in [33] as *irregular*, this pruning scheme yielded the best results in a concurrent adversarial training scheme. Other pruning schemes will be discussed later.

a predefined penalty parameter, k is a compression ratio in the pruning scenario described previously, and s, ϵ are robustness parameters denoting attack iterations and step size, respectively.

Solving Eq. 6 consists of a non-concave maximization in the form of the adversary, and a non-convex outer optimization problem. The solution will occur iteratively as follows:

Solve for $\mathbf{x^{adv}}$

The first step is to generate the maximum allowable perturbation to the sampled data (x, y). All experiments in this work use PGD with s steps of size $\leq \epsilon$ on the ℓ_∞ norm of the given example x. This step is given in Eq. 3.

Update θ'

In this step, at iteration t, we fix all other variables in order to update the parameters in θ' with the associated sparsity constraint. We consider the sparsity constraint as above in which $k := \{k \in \mathbb{Z} \mid 0 < k \leq |\theta|\}$. We iterate over each layer of the network to solve:

$$\theta'_t := \arg\min_{\theta'} \mathcal{L}(\theta_{t-1}, \theta', u_{t-1})$$

This is essentially solving:

$$\arg\min_{\theta'} \sum_{(x,y)\in\mathcal{D}} \langle u_{t-1}, \theta - \theta' \rangle + \frac{\rho}{2}||\theta_{t-1} - \theta'||_2^2 + g(\theta') \tag{7}$$

Update θ

This step solves Eq. 6 by fixing all variables but θ and optionally k or ϵ, the latter 2 of which are typically treated as hyperparameters. We can enforce that k and/or ϵ abide by a monotonically decreasing or increasing sequence respectively. Overall we are interested in:

$$\theta_t := \arg\min_{\theta} \mathcal{L}(\theta, \theta'_t, u_{t-1})$$

That is, we use Stochastic Gradient Descent (SGD) with learning rate η to update the parameters θ as follows:

$$\theta_t := \theta_{t-1} - \eta\nabla_\theta \Big[\mathcal{L}_\theta(x_t^{adv}, y) + \langle u_{t-1}, \theta - \theta'_t \rangle + \frac{\rho}{2}||\theta - \theta'_t||_2^2\Big] \tag{8}$$

At this step, solving for parameters $\{\epsilon, s, k\}$ may be done independently using SGD as well, while enforcing some maximal deterioration of the loss after each iteration t. However, this requires repeating the previous two operations several times which can substantially increase the runtime of the entire training algorithm and has not shown any better results than the algorithm as presented. Thus, unless otherwise stated, k, ϵ are updated manually my pre defining a set sequence of length t. Similarly, an update rule may be used for s in which it increases as long as ϵ stays relatively stable, and then resets when ϵ is incremented. We find the best results

when ϵ is updated every $t/3$ iterations, and s is updated every iteration, resetting to some minimal value whenever ϵ is incremented.

Update u

The augmented parameters follow the standard rule:

$$u_t := u_{t-1} + \rho(\theta_t - \theta'_t) \tag{9}$$

Algorithm 1: Robust Training with Compression

Input: Dataset $\mathcal{D}$, iteration number T, maximum values of ϵ, s and minimum value for k (or stepwise sequences), stepwise sequence $\{\eta_1 ... \eta_T\}$, sparsity set S, batch size b

Output: Model parameters θ

```
for t = 1, 2, ... T do
    Sample (x, y)^b from D
    for j = 1, 2, ... s do
        Iteratively apply PGD to obtain x^adv
    end for
    Solve Eq. 7 to obtain θ'_t
    Use SGD optimizer on Eq. 8 to obtain θ_t
    Update Lagrangian parameters with Eq. 9
end for
```

3.2 Training Settings

Our training algorithm requires the user to specify several hyper-parameters which together steer the output of the algorithm. This section motivates our choice of parameters in our experiments and comments on some of the intrinsic trade-offs of algorithm 1.

3.2.1 Weight Pruning Schemes

The ADMM algorithm lends itself to testing different compression schemes since the compression objective is independent in the algorithm. The following are two alternative pruning schemes to that proposed previously.

Funnel Pruning

This scheme arose from examining the gradient saliency maps of adversarially trained networks to track the importance of features in each layer of the network. Due to both the size of the features and thanks to some state of the art visualization techniques [14], it can be shown that deeper layers in a neural network are more granular than shallow layers. For this reason we try to condition the network to the coarser, shallower features in the network. It is also likely that the highly generalize-able, non robust learned features are more abundant deeper in the network.

Figure 3.1: Result of PGD attack on a model trained with $\epsilon = 0.016$, on an image of the CIFAR10 dataset. Shown is the effect of increasing the perturbation radius on the ℓ_∞ norm of the image.

Funnel pruning alters $\mathcal{S}$ by defining a per-layer rule to the network. Specifically, for a network of N layers:

$$\theta \in S := \{\, \forall \theta_n \in \theta \mid \frac{||\theta_n||_0}{|\theta_n|} - \frac{||\theta_{n+1}||_0}{|\theta_{n+1}|} > 0 \,\}$$

We impose that each consecutive layer of the network must have proportionally more weights pruned than the previous layer.

Redundant Pruning

This method takes inspiration from recent works that aim to suppress redundant weights in CNNs rather than weights with value close to zero. Like previous schemes, we must define some threshold ϕ which indicates the minimal tolerable difference between weight matrices:

$$\theta \in S := \{\, \forall \theta_n \in \theta \mid \nexists\, \theta_j \; s.t \; ||\theta_n - \theta_j|| < \phi \,\}$$

Where θ_j denotes the weights at any layer of the network other than layer n.

Similar methods to both of these have proved beneficial in different use cases than our own. We trained models using all three pruning schemes described but have not observed consistent performance improvements from one to the other. Unless otherwise specified, models shown in our results use the irregular pruning scheme as defined earlier on in the chapter.

Quantization

Although often necessary in industry to deploy deep learning models in resource constrained environments, this work does not include any network quantization. Quantization is a typical step to take for anyone deploying a model in resource scarce environments and could be be worked into our optimization framework with little hassle, or could be done following training using one of the many existing out of the box deep learning software packages. Studying the quantization of our models is a direct continuation of this work.

3.2.2 Adversarial Strength

Most works focusing on robustness train and test their models with constant adversarial parameters, such as keeping ϵ constant for an entire training. The choice of ϵ in most works is either 2, 4, or 8 on the CIFAR datasets, the last of which equates to $\epsilon = 0.03$ as stated in this work. However, the perturbation of such an adversary on the test data is still at times imperceptible to the human gaze.

Figure 3.1 is an example of the potential of a white-box PGD adversary for rising levels of ϵ and access to 7 gradient evaluations on an image in the CIFAR10 dataset. The attack was performed on a model trained with a constant value of $\epsilon = 0.016$. See appendix A.2 for more examples.

For small values of ϵ the image still appears unchanged to the naked eye, while for larger values the image has been altered beyond human interpretability. It can be shown that using higher adversarial hyperparameters can yield more robust models, with the added drawback of further reducing standard accuracy.

We would like our models to perform consistently against an adversary with step size up to $\epsilon \approx 0.1$, and with many gradient evaluations [12]. We believe this invariance to be an important and easily check-able trait for robust models from an empirical standpoint that we will address prior to evaluating using any of the other members from the large family of adversarial attacks.

4. Experiments

Training with limited resources and deploying deep learning models in constrained settings at network edge requires extra considerations; from security guarantees to efficiency. We will use this section to outline our experimental approach, starting with a description of the data and the models.

We go on to discuss training models with algorithm 1, and the trade-offs incurred from hyperparameter selection. We then detail our tests one by one with results in line on several robustness evaluations, followed by a brief analysis of computational performance.

Data

We will evaluate using the CIFAR10 and CIFAR100 datasets [38], both of which are made up of $60,000$ RGB images of size 32×32 with 10, and 100 class labels, respectively.

Models

We train several configurations of two commonly used CNN architectures; ResNet34 and MobilenetV2. We chose this pairing in particular to contrast two CNNs of vastly different size, geared towards deployment in different environments. The former has more layers and is known to be overparameterized, especially for the CIFAR10 dataset, while the latter has one tenth of the parameters and is optimized to run in resource constrained settings; specifically on CPU boards. See table 4.1 for more model details.

Experimental Training Parameters

We augment the CIFAR training data by flipping and rotating images randomly in addition to any adversarial perturbations we use on top of that. Our total loss function is a combination of the cross entropy loss on the natural and perturbed examples. We experimented with many weightings of the two components in the total loss and found that using 90% adversarial loss to 10% natural loss generalized best to sufficiently strong adversaries.

Model	**Parameters**	**CIFAR10**	**CIFAR100**
ResNet34	21M	94.0%	72.8%
MobilenetV2	2M	93.2%	66.0%

Table 4.1: Models used our experiments, with standard accuracy reports on the benign test sets.

We train for 200 epochs and carry on for longer if a model is experiencing noticeable decrease in adversarial loss. We use step learning rate decay every 10 epochs starting at $\eta = 5 \times 10^{-2}$. Tests were conducted for many combinations of pruning and adversarial parameters, but will be shown for $k = \{100\%, 50\%, 10\%\}$[1], maximum perturbation $\epsilon = \{0, 0.03, 0.1\}$[2], step count $s = \{7, 14, 21\}$ in the training setting, and step count $s = \{21, 100, 2000\}$ when evaluating.

All of our models were trained on a single Intel x86-64bit machine with 8 multithreaded CPU cores and 64GB memory with a GV100GL Nvidia GPU clocked at 33MHz. Our training algorithm requires no pretraining or fine tuning and terminated in a maximum of 28 hours in our experiments.

4.1 Analysis of the Training Algorithm

At this point we recall the two optimization objectives we are enforcing onto our models: robustness and compression. These two objectives are both known to deteriorate the standard accuracy of the model, and complicate training convergence when used independently. Using both at once is a difficult task to optimize. To better observe the effect of our training algorithm on the model, we will analyse two distinct features: the distribution of weights post-training, and the evolution of the adversarial loss at training time.

4.1.1 Distribution of Weights

Compressing CNNs in the benign setting can still achieve high levels of accuracy, and exploits to the fact that deep neural networks are usually overparameterized. Training in the adversarial setting complicates this by forcing the model to learn more difficult and potentially more subtle features. These features may require a larger amount of parameters to capture, and do not generalize as strongly.

In figure 4.1 we can see that with no pruning scheme in place, adversarially trained models exhibit more variance in their weight values, indicating more weights are required to represent data in the adversarial setting. This phenomenon is to be expected, as it was elegantly shown that robust training inflates variance in the case of a binary classifier [3].

Furthermore, pruning the weights of a neural network may result in useful features being foregone entirely. This could be further exacerbated in the adversarial setting, where finding sparse representations is more difficult. Things like choice of pruning scheme and training hyper-parameters can drastically effect the result of compressing models in this setting.

Figure 4.2 shows the effect of pruning the dense ResNet34 model. We observe that as more weights with values ≈ 0 are pruned, the model will attempt to compensate by accumulating larger weight values for those still active weight parameters. We observe that similar accuracy is still achievable for smaller, more compressed models, but only in settings with weak adversarial threat levels. We will further dis-

[1]Here we express k as a percentage of the total parameters shown in table 4.1 for simplicity. Typically, k is an integer value.

[2]Common in literature is to express ϵ as a fraction on the RBG scale 1-255. We will sometimes use a fraction but typically will stick to decimal notation.

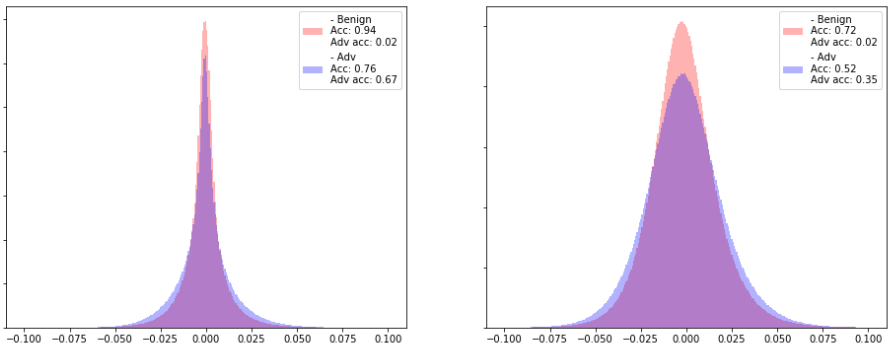

Figure 4.1: Histogram of weight values in dense models trained in the benign and adversarial settings with $\epsilon = 0.03$. Shown are results for the CIFAR10 (left) and CIFAR100 (right) datasets. Accuracy scores are reported in the legend in a benign test setting as well as with a PGD adversary and $\epsilon = 0.03$ (adv_acc).

cuss these phenomena in our empirical robustness evaluation of compressed models. See appendix B.2 for the same figures as seen here, for the MobilenetV2 models.

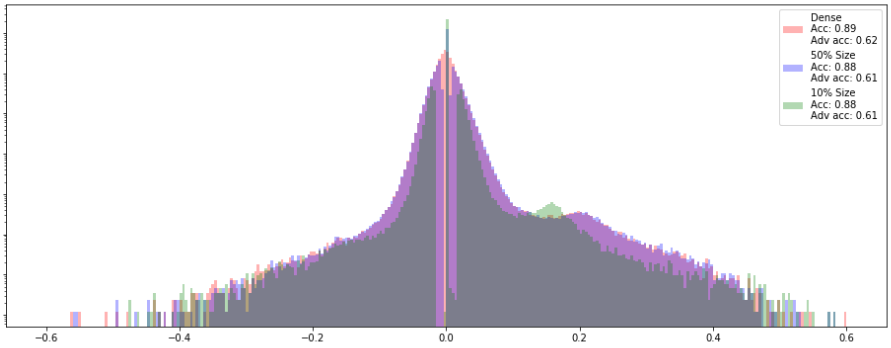

Figure 4.2: Histogram of weights for the dense ResNet, and compressed models of two different compression ratios, on the CIFAR10 dataset. All models were trained in the adversarial setting. Note this plot now has log-scale on the y axis and is normalized to 1 on the x axis for better viewing of the compressed models.

4.1.2 Evolution of the Adversarial Loss

From a different perspective, one way to guarantee robustness of a model to a degree is by observing a constantly decreasing adversarial loss across iterations at training time. This fact has been pointed out in previous works [2], and is empirically supported by subsequent results. However, the adversarial loss has been little studied with compression algorithms in play.

Figure 4.3 shows the evolution of the adversarial loss over our training algorithm without any compression policy in use. As we can see, the monotonically decreasing loss implies we can train increasingly robust models. The sudden drops in loss function value are due to hyperparameter changes over the training algorithm.

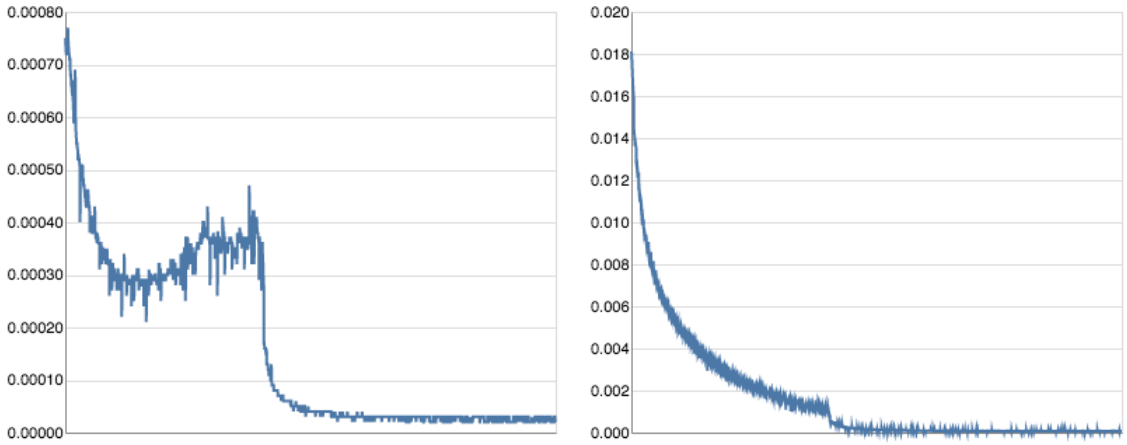

Figure 4.3: Evolution of the adversarial loss (y) over 200 training iterations (x) in the dense setting with no compression. Shown is ResNet (left) and MobileNet (right) using the CIFAR100 dataset.

The loss landscape becomes more complicated as we directly remove weights from the model and appears as if we cannot ensure an increase in robustness as we compress past a certain, unknown limit that depends on both the model architecture and the training data. Figure 4.4 shows two examples in which we employ pruning in the adversarial setting and portrays that in some cases we are able to find good solutions to the robust saddle point and compression problems simultaneously. However, is it not uncommon to see explosions in the adversarial loss, indicating no good solution can be found past a certain compression ratio.

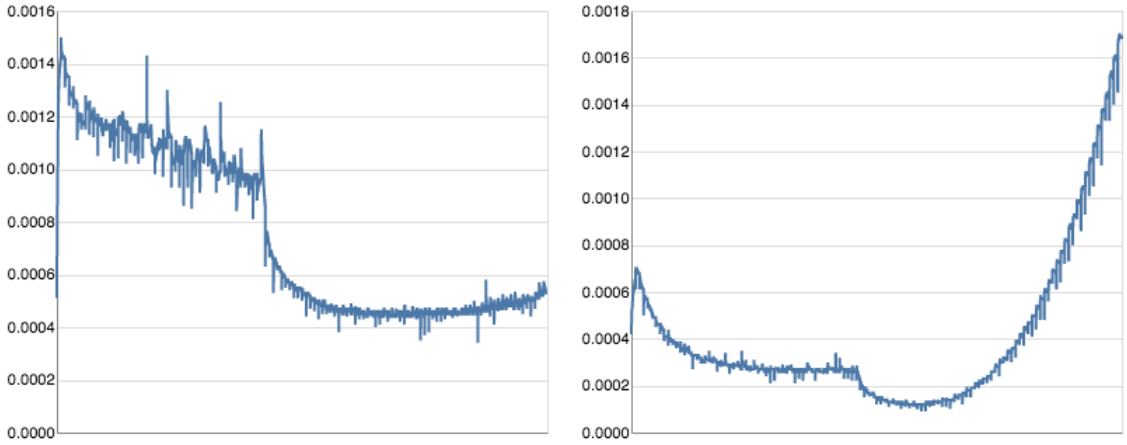

Figure 4.4: Evolution of the adversarial loss (y) over training iterations (x) while attempting to compress ResNet34 to 50% size (left), and MobilenetV2 to 10% size (right). The plot shows in both cases that robustness begins to diminish past a certain compression ratio.

It is also interesting that in the optimization process, good simultaneous solutions to the saddle point and the compression problems may appear somewhat randomly, and the criteria for a good solution is ill-defined. For this reason compression ratio and adversarial strength are preferred as step-wise parameters in our training algorithm. Nevertheless our results will be shown for a few selected thresholds of

compression and for models conditioned to two different adversaries. See appendix B.4 for more information on the evolution of the adversarial loss and supplementary figures.

Although our training algorithm does not allow us to *verify* robustness for compressed models, we observe that good sub-optimal solutions are consistently found without requiring further training. We will further explore this in the following section.

4.2 Robustness Evaluation

We previously spoke of the set of *allowable perturbations*: Δ. In mainstream research this set is typically defined by perturbations on the ℓ_p norm of the data where $p \in \{1, 2, \infty\}$. However, potential adversaries have no requirement to abide by this unit of measurement, and are free to attack classifiers using a number of methods including black box, white box, and other less commonly studied threat models.

To cope with this enormous difficulty, recent discourse in the field of robust DL has elevated the usage of standardized benchmarks and evaluation techniques. These typically involve several different threat models and measurements aside from just an ℓ_p norm.

4.2.1 Optimal Adversarial Examples

Stepping back momentarily to consider the greater scheme of this field, we are attempting to avoid cases in which carefully crafted data inputs are misclassified by a CNN, while to a human, these inputs are similar or indistinguishable from the benign data. Ideally, it should be *computationally difficult* to find an adversarial example, and those examples should appear to be distorted when evaluated by a human.

Regarding the optimal adversarial example x^*_{adv} on a deep neural network with parameters θ, and for some underlying distribution $\mathcal{D}$, we state that:

$$\forall (x, y) \in \mathcal{D}, \quad \exists\, x^*_{adv} = x + \delta^* : \quad \delta^* = \arg\min_{\delta \in \Delta} \max \mathcal{L}(x + \delta, y\,;\, \theta)$$

That is, for some benign sample x, the optimal adversarial example is that which minimizes $||\delta||_p$ or any other attack metric being used, such that the log loss is maximized over that example.

Recent works have suggested various ways to measure perturbations and have suggested that both the image capturing device as well as the digital format play a role in the landscape of possible adversarial examples. We will focus mainly on ℓ_p bounded perturbations and will present results to many commonly known attacks.

Attacks

We will conduct our analyses on the following threat models, in order (with brief explanations inline):

1. **PGD**: as our models are trained on the PGD adversary it is important to first evaluate robustness empirically against this threat model. Importantly,

we emphasize evaluating our models on the PGD adversary with increasing strength.

2. **APGD-ce & APGD-dlr**: two variants of the typical PGD attack, these adversaries forego the fixed step size, use momentum, and leverage exploration vs exploitation to find good starting points from which it can restart, and maximize the loss. The latter uses the *difference in logits-ratio* loss, as opposed to the cross-entropy (ce) loss function. These two attacks were proposed by [39] and are part of a now popular benchmark called *autoattack*.

3. **FAB**: another white box attack that aims to minimize the norm of the perturbation using gradient feedback.

4. **Square**: a score based black box query attack that randomly searches for adversarial examples without gradient information or even gradient approximations.

5. **Transfer PGD**: we will perform an attack using the PGD adversary on several different *source* models, and using the examples generated by those adversaries we will evaluate other *target* models and report the accuracy.

6. **Carlini-Wagner**: a relaxed, regularization based attack that uses a rectifier function to handle non-boundedness and the computational load.

Algorithm 2: Evaluation vs Increasingly Strong Adversary

Input: Dataset $\mathcal{D}$ with k class labels, adversarial threat model (PGD adversary) $\mathcal{A}$ with fixed step count and stepwise sequence $\mathcal{E} = \{\epsilon_0, \epsilon_1, \epsilon_2 \dots \epsilon_n\}$ where ϵ_0 denotes no perturbation, model θ

Output: Robustness Scores S

$S \leftarrow \emptyset$
for $i = 0, 1 \dots n$ **do**
  Sample (x, y) from $\mathcal{D}$ exhaustively, and attack θ using $\mathcal{A}, \epsilon_i$ to yield x^{adv}
  Evaluate exhaustively and accumulate correct classifications on (x^{adv}, y): $c_i = \frac{\#correct}{|\mathcal{D}|}$
  Add (ϵ_i, c_i) to S
  if $c_i < k^{-1}$ **then**
    break
end for

4.2.2 Evaluation on the PGD Adversary

Testing for empirical robustness on the vanilla PGD adversary involves doing a typical accuracy test over the training set with the added adversarial parameters. To eliminate manually deciding those adversarial parameters, we set only the amount of attack iterations $s = 100$, and iteratively conduct accuracy tests in the adversarial setting as we increase the strength of the adversary. The algorithm is depicted in Algorithm 2.

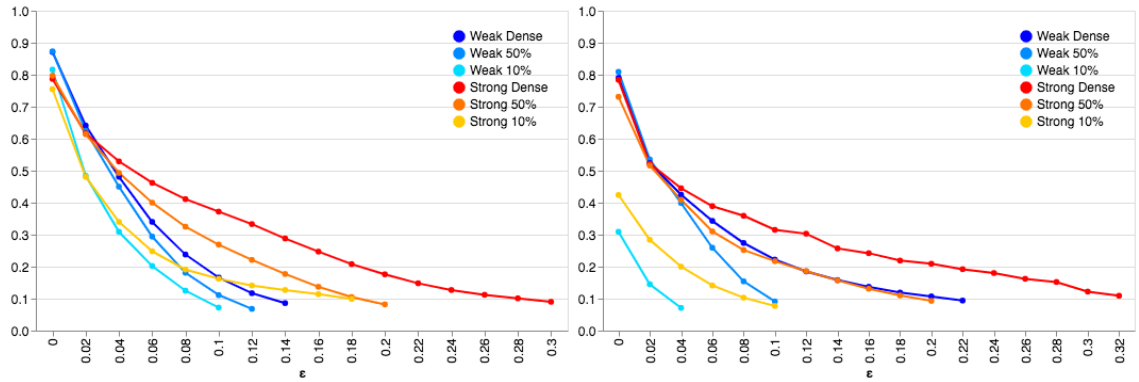

Figure 4.5: Evolution of empirical robust accuracy (y) for increasing values of ϵ (x). Shown are results for **ResNets (left)** and **MobileNets (right)** both trained with PGD. We show the robust models conditioned to $\epsilon = 0.03$ (weak) and $\epsilon = 0.1$ (strong).

Figure 4.5 depicts the result of algorithm 2 on both the ResNet and MobileNet models with varying compression ratios for the CIFAR10 dataset. It is clear that conditioning our models to an increasingly strong adversary at training time does deteriorate standard accuracy on the benign set and continues to do so as ϵ increases. However, this loss is compensated for by the clear gains in robustness against a stronger adversary. This is clear when contrasting the *strong* models in warm colors with the *weak* models in cool colors, conditioned to $\epsilon = 0.03$ and $\epsilon = 0.1$, respectively.

It seems that against adversaries up to $\epsilon = 0.04$ that all models perform similarly no matter the compression ratio. This is reassuring in that it shows empirical robustness is not off-limits for highly compressed models. As the threat level rises, we see the gap between dense and compressed models begin to grow, suggesting the additional parameters play a part in maintaining robustness agianst an increasingly strong adversary.

We also note that the MobileNet model suffers dramatically when pruned to have only 10% of the initial parameters. Such a small amount of parameters simply does not seem to be capable of modeling the CIFAR10 dataset, as benign accuracy is close to 40%. For this reason we will forego evaluating pruned MobileNets on the CIFAR100 dataset. See appendix C.1 for our evaluations using the CIFAR100 dataset.

4.2.3 Empirical Robustness

In this section we evaluate on attacks 2-4 from the list above. These attacks make up a benchmark known as *auto attack*[3], which has gained academic recognition over the past few months. Auto attack evaluates the two variants of PGD as well as FAB using 5 random restarts and 100 iterations after each restart. The square attack is given a limit of 5000 queries. Using all the same models as those from figure 4.5, our results are shown in table 4.2.

Considering the entire landscape of robust deep learning models, these results do

[3]There exists a public leaderboard for the auto attack available online: `https://github.com/fra31/auto-attack`

Model	Compression	Benign	APGD-ce	APGD-dlr	FAB	Square
ResNet $\epsilon = 0.1$	Dense	77.25	53.9	44.2	43.9	43.9
	50%	77.2	**53.4**	43.3	42.4	42.4
	10%	66.7	42.9	38.8	38.0	38.0
ResNet $\epsilon = 0.03$	Dense	90.9	36.9	35.4	35.5	35.5
	50%	90.0	36.6	35.5	35.5	35.5
	10%	86.0	29.9	29.3	29.2	29.2
MobileNet $\epsilon = 0.1$	Dense	79.9	47.2	41.2	40.8	40.8
	50%	79.7	40.8	38.8	37.7	37.7
	10%	40.4	28.8	24.5	24.0	24.0
MobileNet $\epsilon = 0.03$	Dense	89.8	36.9	35.8	35.6	32.6
	50%	87.4	36.5	35.5	35.5	35.5
	10%	36.8	8.0	3.0	2.5	2.5

Table 4.2: Accuracy reports on benign data and on four different adversarial attacks on the ℓ_∞ norm with $\epsilon = 8/255 = 0.03$ for ResNet and MobileNet models of different compression ratios and conditioned to different PGD adversaries, on the CIFAR10 dataset.

not entirely stand out, however most other methods either train using much more computational power, use pre or post-training tuning schemes, or use dense models, some of which are larger than our test models. We believe our robustness scores for compressed models are novel taking into consideration the previously mentioned criterion.

We conduct one additional test in which we combine the output of a benign (non robust) model with that of a robust model. That is, we take the average of the probabilistic class predictions from the two networks and consider this prediction. This method requires more computational overhead as the parameters of multiple networks must be in memory at once. That being said, we find that this method yields higher robust accuracy against two different adversaries, namely PGD and Carlini-Wagner, than the robust compressed models do on their own. These results are available in appendix C.1.

4.2.4 Transfer Attacks

We conduct a transfer attack wherein we consider a white-box vanilla PGD attack on a source model, and save the data generated from that attack to evaluate on many other models with no further attack. We would like to measure the vulnerability of these models to eachother, and how the accuracy of a model deteriorates when a similar model trained for the same task has been exposed. Deploying many similar models is becoming common in serverless and distributed environments where many models may be deployed. This experiment could contribute to the decision process for model deployment. The transfer test is depicted in algorithm 3.

Algorithm 3: Transfer Attack

Input: Dataset $\mathcal{D}$, batch size b, adversarial threat model $\mathcal{A}$, source model θ^s, set of n target models $\{\theta_1, \theta_2 ... \theta_n\} \in \mathcal{T}$

Output: Accuracy scores S

$S \leftarrow \{0\}^n$
for $i = 1, 2 \ldots, |\mathcal{D}|/b$ **do**
  Sample $(x, y)^b$ from $\mathcal{D}$
  Generate $(x^{adv}, y)^b$ by attacking θ^s with $\mathcal{A}$
  for $t = 1, 2 \ldots, n$ **do**
    Evaluate θ_t on x^{adv}
    Update S_t
  end for
end for

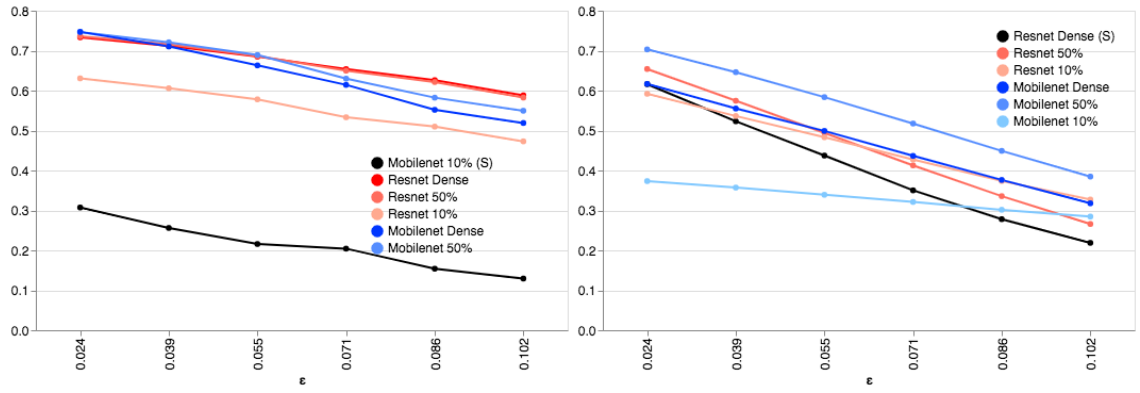

Figure 4.6: Results for the transfer experiment at rising values of ϵ (x axis), and the corresponding accuracy (y). The source model is denoted in black, with target ResNets in red and target MobileNets in blue. All models in this figure were conditioned to a maximum of $\epsilon = 0.1$.

Due to resource and time constraints we elected to evaluate using only our highest performing and lowest performing models as source models. Namely we use the dense ResNet conditioned to $\epsilon = 0.1$ and the MobileNet conditioned to $\epsilon = 0.03$. Figure 4.6 shows the results of the transfer attack in both directions. See appendix C.2 for transfer attack results for models conditioned to a weaker adversary.

To be expected, and supported by results in figure 4.6, attacks on a less performant or highly compressed model do not transfer well to the dense models. That is, attacks on less performant models do not generalize well to stronger models and different architectures. However, using dense models as the source of an attack we are able to find strong adversarial examples for all other models as the perturbation bound increases.

Interestingly, models across the board are more resistant to this variety of transfer attack when they have a different model architecture than the source model. We see this in figure 4.6 (right) where the MobileNets are able to outperform ResNets for almost all values of ϵ, when the source model itself is a ResNet [4]

[4]With the exception of MobileNet compressed to 10% which has not exhibited empirical ro-

4.3 Performance Evaluation

Finally, we conduct a performance evaluation of our dense and compressed models on a number of different hardwares ranging in computational power. We note that supplementary steps will be needed to obtain significant performance improvements on embedded and IoT hardwares. These steps include, but are not limited to compilation and quantization. Nevertheless, there are already small programmatic modifications we can make within our framework to speed up the computation of sparse matrices in our networks.

We conduct two experiments to measure the runtime of a forward pass through our networks. In particular we would like to measure:

- Time to compute a forward pass on a single image that is already in memory.
- Time to compute a forward pass on N images, regardless of if they persist in memory.

We will test our models on the following hardware configurations: 2013 Macbook Pro CPU, Intel NUC CPU, Nvidia Jetson Nano ARM CPU, Nvidia Jetson Nano GPU, Nvidia Tesla v100-PCIE GPU. We will separate plots for the Jetson ARM CPU from the others due to conflict of scale as the ARM CPU takes by far the most time to compute a forward pass for all models.

Algorithm 4: Protocol to Process 1 Image

Input: Dataset $\mathcal{D}$, model θ
Output: Average Performance t^{avg}

$\mathcal{T} \leftarrow \emptyset$
$i = 0$
while $i < |\mathcal{D}|$ **do**
  Draw a single sample (x, y) from $\mathcal{D}$
  Record time t_1
  Evaluate (x, y) using θ
  Record time t_2
  $t_i \leftarrow t_2 - t_1$
  Add t_i to $\mathcal{T}$
  $i \leftarrow i + 1$
end while
Compute the average of all elements in $\mathcal{T}$ and **return** t^{avg}

Results on different hardwares is shown in figure 4.7, and for the Jetson Nano ARM CPU in figure 4.8. First, and as expected, we observe that the Mobilenet is more efficient to run on CPU hardwares. On the other hand, devices with GPUs can parallellize more of the operations in the ResNet to further speed up computation.

Across the board, we see marginal performance gains from compressed models; on all hardwares the forward pass is faster for more highly compressed models. However, the gains are negligible in some cases as the computation may already be very fast.

bustness to much degree across all our experiments and to some degree can be disregarded. This could be due to underparameterization, as this model has parameters on the order of 200k. Refer back to figure 4.5.

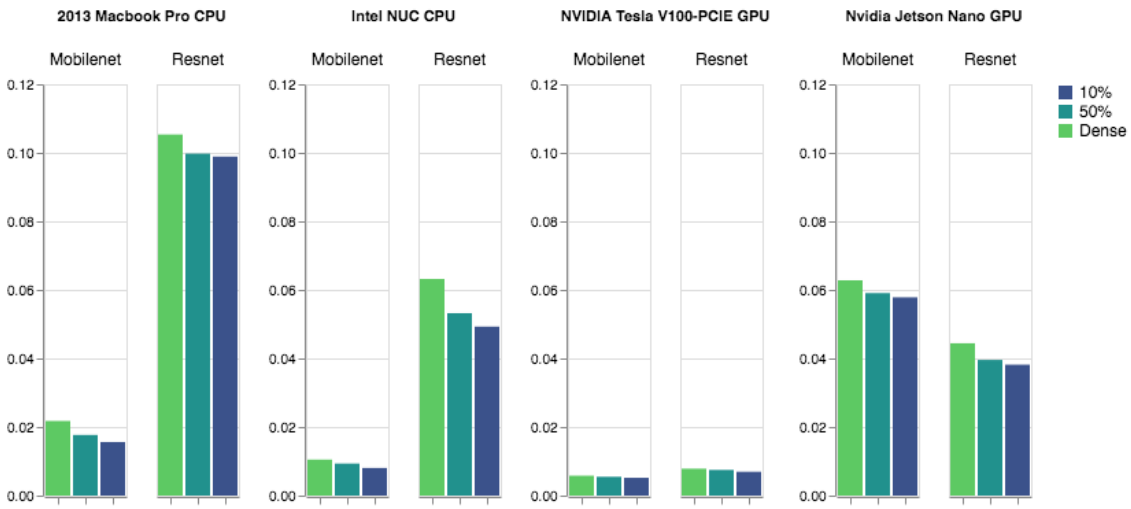

Figure 4.7: Average latency (in seconds) to compute a forward pass for a single image by our different CNNs, on different hardwares.

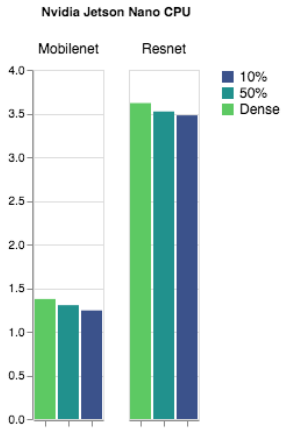

Figure 4.8: Same results as figure 4.7, for only the Jetson nano ARM CPU (note the change in scale on the y axis)

For example the Nvidia Tesla GPU has no problem processing even our largest networks and thus compression does not yield large gains. In many cases, the speed of a forward pass is a fraction of a second. We conduct our second analysis with 100 images at a time, and show similar figures to those here, in appendix D.1. These results confirm that pruning the weights of models contributes to computational efficiency at runtime, yet they also highlight the need for further steps to be taken programmatically in order to bring models into deployment.

5. Conclusion

In this work we have explored the state of the art in adversarial machine learning and in model compression, in the context of deep convolutional neural networks. We proposed a one-shot training algorithm using ADMM to address both of these problems at once: to train robust, compressed models in a tractable way, on a single GPU. Those models were evaluated under many different attack contexts and a performance evaluation was conducted on hardware with ranging computational power.

Our work is unique in that we conduct a wide variety of tests and begin to approach the steps necessary for deploying our models on embedded and IoT devices. Although, to best situate our work within the state of the art we will discuss our work within the context of software solutions like that of Ryax Technologies. We further discuss aspects of this work that need improvement, and the current research landscape within the topic.

This research helps to inform the workflow of Data Scientists and Machine Learning engineers working in industry to deploy models in the wild. At a company like Ryax Technologies, the methods an insights gathered throughout this project can manifest in several ways. First, it is common for algorithms to be standardized; our robust ERM, compression, and evaluation algorithms could all be offered as out of the box tools, along with extensions of them which we plan to develop. These algorithms have the same functionality on any CNN and need not be changed, making them readily usable. Moving forward, this research could be augmented by the ongoing projects and research pursuits at Ryax, particularly with regard to testing in a serverless production environment. We will further touch on this after first commenting more specifically on aspects of this work.

From a model training perspective, most works do not consider a shortage of computing power, and excellent results have recently been produced from methods using pre-training and fine-tuning in addition to a computationally heavy training procedure. Such strategies do not lend themselves to scenarios with resource limitations, or in which retraining is frequently required. Training robust models will continue to be a costly task, and mitigating the overall cost of maintaining deep learning systems in a production environment will call for clear delineation of when, and where to use certain training algorithms that could (or need to) be run in a tractable and cost-efficient manner.

On the other hand, the options for evaluating robust deep neural networks will continue to disperse as research interest and industrial applications continue to rise. Our experiments cover only a fraction of the existing techniques and do not necessarily carry over to other robustness criterion. In this way evaluation is impossible to quantify holistically and must be conducted according to the use case. In continuations of our work we will continue to evaluate empirical robustness using the

white-box first order adversary and will seek to include more black-box and query attacks to our methodology.

In the future we see two main continuations of this work. First and foremost it is necessary to bridge the gap between the research and industrial application. To this end, we hope to address the quantization and containerization of our models within the system level toolbox at Ryax Technologies. This is a subsequent step that will tie our work together into a pipeline to train, test, and deploy robust models on resource scarce devices. This pipeline could then be offered to Data Scientists who need only conduct data analysis and model tuning to deploy their own models, fit to whatever their use case may be. Further, we believe that a more thorough investigation of our training algorithm will provide insights into other phases of our work such as evaluation and model selection. We would like to investigate ulterior pruning schemes with a wider range of neural architectures to better understand what characteristics of the models we are using contribute most to empirical and verifiable robustness.

A.

Robust Features and Adversarial Examples

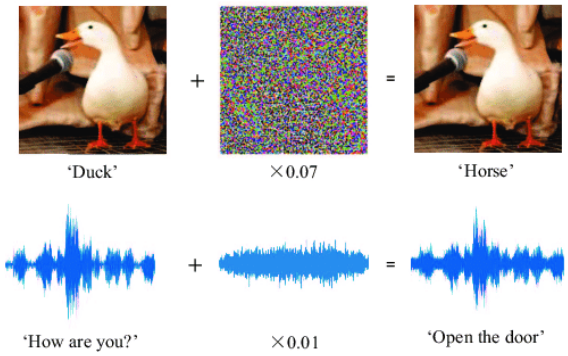

Figure A.1: An illustration of deep learning *blind spots*, where an imperceptible amount of added noise can cause a correctly classified example to be misclassified [10].

Robust and Non-Robust Features The following framework from [3] characterizes the robustness of data features.

Consider the binary classification setting where examples $(x, y) \in \mathcal{X}$; $x \in \mathbb{R}^d$, $y \in \{\pm 1\}$ are used to train a classifier $C : \mathcal{X} \to \{\pm 1\}$. Let $\mathcal{F} = \{f : \mathcal{X} \to \mathbb{R}\}$ be the set of all features f, with an individual feature being one such mapping described previously. For a given data distribution $\mathcal{D}$, such features can be characterized as follows:

- ρ**-useful**: A feature f is ρ-useful, with $\rho > 0$, if it correlates with the true data label.

$$\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[y \times f(x) \right] \geq \rho$$

- γ**-useful**: given a ρ-useful feature, this feature is γ-useful if when subject to perturbation from the valid set $\delta \in \Delta$, this feature is still ρ-useful.

$$\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\inf_{\delta\in\Delta} y \times f(x+\delta) \right] \geq \gamma$$

Training a Robust Classifier

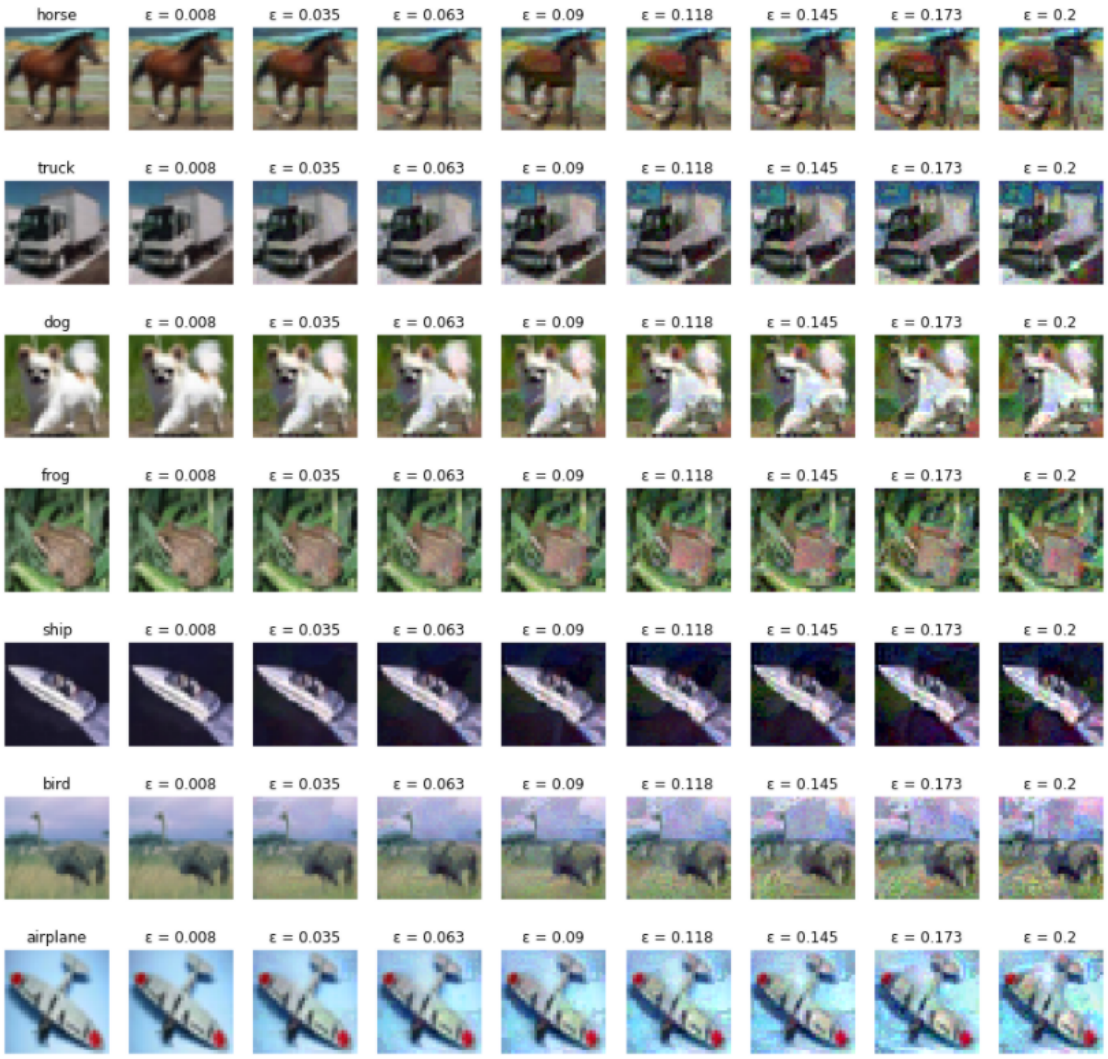

Figure A.2: Supplementary CIFAR10 attacked images. Shown is the effect of increasing the perturbation radius on the ℓ_∞ norm. Some images exhibit more intense distortion; a factor dependent on the parameters of the model being attacked.

B.

Supplementary Analysis of the Training Algorithm

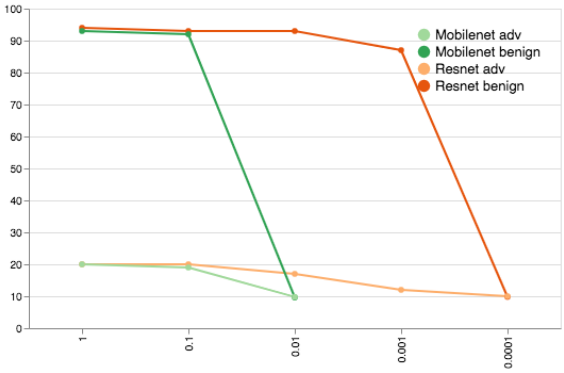

Figure B.1: Compression ratio (x) vs accuracy (y) of both ResNet34 and MobileNetv2 trained in a benign setting on the CIFAR10 dataset, evaluated in both a benign setting (benign) and with FGSM adversary with $\epsilon = 0.03$ (adv). Recall compression ratio denotes the ratio of the amount of nonzero weights in the network to the total amount of weights in the dense network.

Model Compression in the Adversarial Setting

Below are supplementary figures to those in Chapter 4, showing the distribution of weights in different configurations of the MobilenetV2.

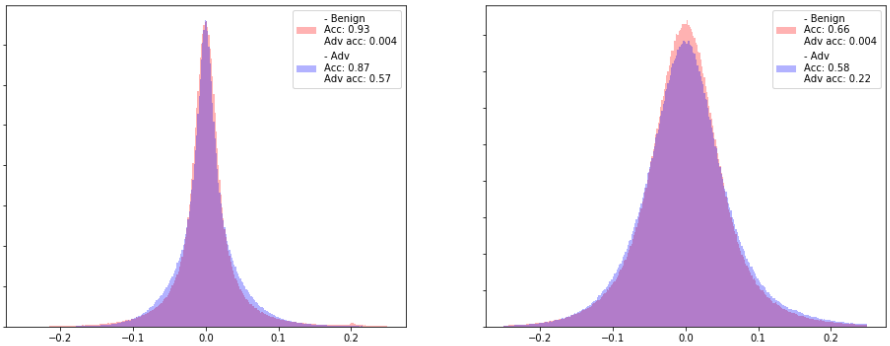

Figure B.2: Dense MobilenetV2 weight distribution in the benign and adversarial setting (Labeled in the legend with corresponding accuracies) for the CIFAR10 (left) and CIFAR100 (right) datasets.

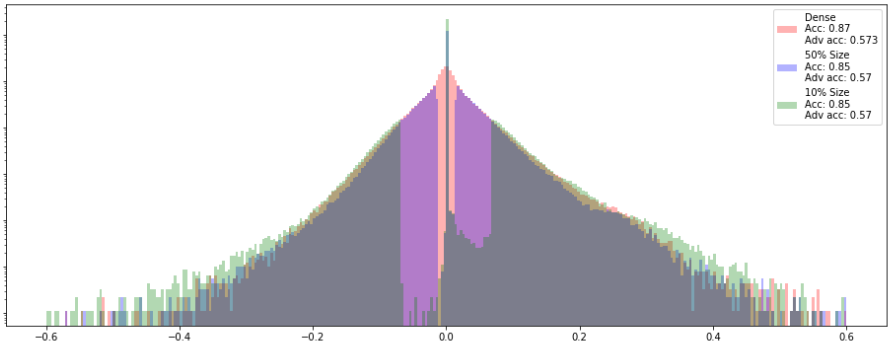

Figure B.3: Histogram of weights for MobilenetV2 trained in the adversarial setting, for different compression ratios. Against a weak adversary (PGD, $\epsilon = 5/255 = 0.019$), a compressed model can achieve similar accuracy to the dense model. Note the log scale on the y axis and normalized x axis to $\{-1 \dots 1\}$

Evolution of the Adversarial Loss

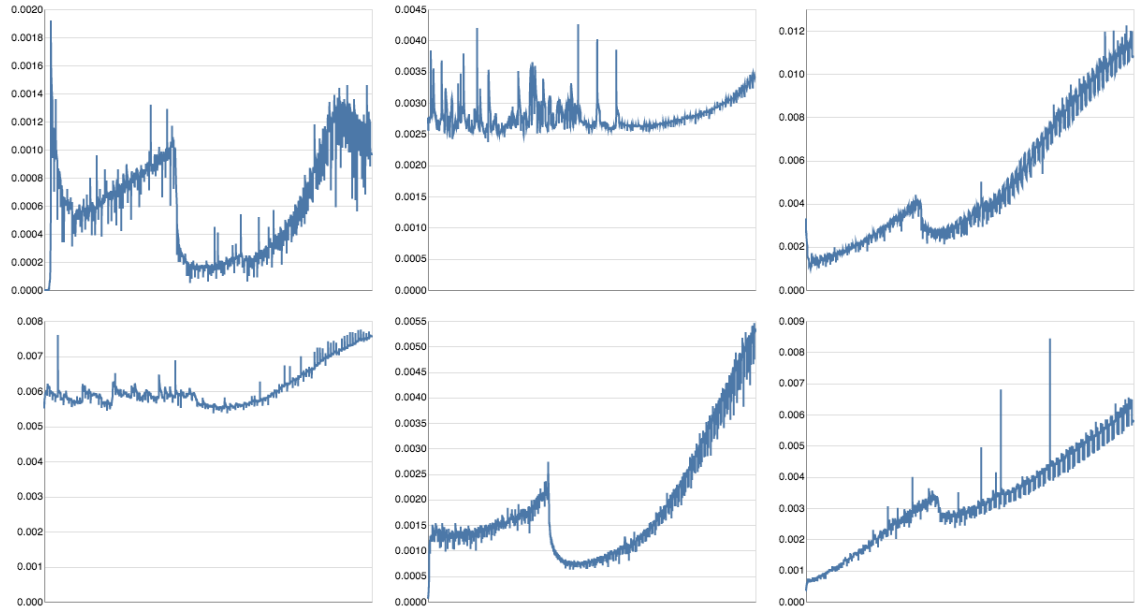

Figure B.4: Evolution of the adversarial loss (y) over training epochs (x) for:

Top row left to right: ResNets on CIFAR10 compressed to 50%, 10%, and 10% on CIFAR100. Bottom row left to right: Mobilenets on CIFAR10 compressed to 50%, 10%, and again 10% with a weaker adversary

It is clear that as models are further compressed, the adversarial loss is likely to explode either due to the lack of parameters or facility for the adversary to produce an example that fools the classifier. In addition, it is interesting to see that the loss at times settles to a minimum at some point in the training process while the model is still being compressed, before again increasing. This suggests that there may be nice solutions hidden at different combinations of robustness and compression ratios.

C.

Supplementary Robustness Evaluations

Evaluation on the PGD Adversary

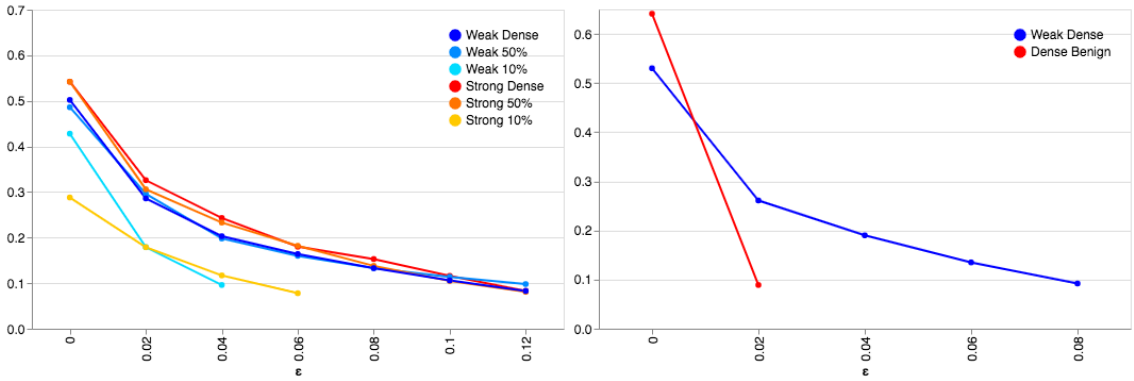

Figure C.1: Evolution of empirical robust accuracy (y) for increasing values of ϵ (x). Shown are results for **ResNets (left)** and **MobileNets (right)** both trained with PGD. We show the robust models conditioned to $\epsilon = 0.03$ (weak) and $\epsilon = 0.1$ (strong). Shown for mobilenets is the non-adversarial, fully parameterized model as well as the robust dense model conditioned to $\epsilon = 0.03$

Combining robust and non-robust networks

Model	Taining Method	Benign	PGD	CW
ResNet	Normal	94.1	1.3	0.9
	Robust	76.8	58.3	54.0
	Combined	85.6	**82.2**	**78.0**
MobileNet	Normal	93.2	0.8	0.7
	Robust	70.1	51.1	46.4
	Combined	82.3	**82.0**	**77.1**

Table C.1: Accuracy scores from averaging the class predictions from non robust and robust models at test time. Doing so deteriorates the standard accuracy against benign data but significantly improves accuracy against both the PGD and Carlini-Wagner attacks. This test was conducted with $\epsilon = 0.1$, and 32 steps. We observe completely deteriorated accuracy against any attack for non-robust models.

Transfer Attacks

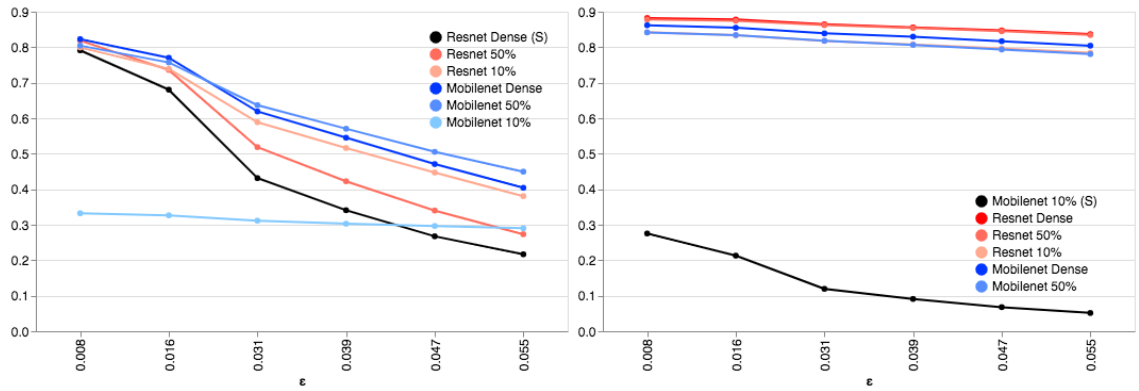

Figure C.2: Results for the transfer test algorithm at rising values of ϵ (x axis), and the corresponding accuracy (y). The source model is denoted in black, with target ResNets in red and target MobileNets in blue. All models in this figure were conditioned to a maximum of $\epsilon = 0.03$. As we can see the same trend is seen here as that in figure 4.6. Attacks on weaker or highly compressed models are incapable of generating adversarial examples for stronger, denser models. Similarly, attacks are more successful on models that have the same architecture as the source model, while models of different architectures are more difficult to attack even when compressed to be very small.

D.

Supplementary Performance Details

Algorithm 5: Protocol to Process n Images

Input: Dataset $\mathcal{D}$ clipped such that n divides $|\mathcal{D}|$, amount of samples to process at once n, model θ
Output: Average Performance t^{avg}

$\mathcal{T} \leftarrow \emptyset$
$i = 0$
while $i < |\mathcal{D}|/n$ **do**
    Record time t_1
    Load n samples from $\mathcal{D}$ into memory
    Evaluate those n samples with θ
    Record time t_2
    $t_i \leftarrow t_2 - t_1$
    Add t_i to $\mathcal{T}$
    $i \leftarrow i + 1$
end while
Compute the average of all elements in $\mathcal{T}$ and **return** t^{avg}

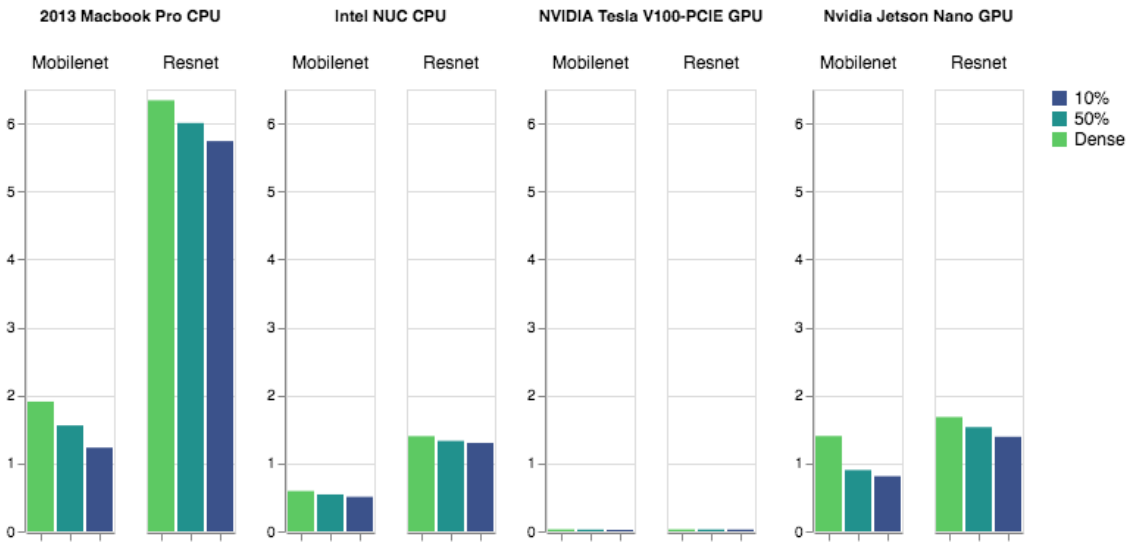

Figure D.1: Average latency (in seconds) to compute a forward pass for $N = 100$ images by our different CNNs, on different hardwares.

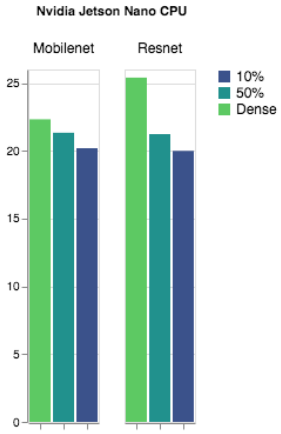

Figure D.2: Same results as figure D.1, for only the Jetson nano ARM CPU (note the change in scale on the y axis)

References

[1] Y. Han, X. Wang, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *ArXiv*, vol. abs/1907.08349, 2019.

[2] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *ArXiv*, vol. abs/1706.06083, 2017.

[3] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, "Adversarial examples are not bugs, they are features," in *NeurIPS*, 2019.

[4] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, pp. 187:1–187:30, 2016.

[5] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2775–2784, 2019.

[6] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4335–4344, 2018.

[7] S. Ye, X. Feng, T. Zhang, X. Ma, S. Lin, Z. Li, K. Xu, W. Wen, S. Liu, J. Tang, *et al.*, "Progressive dnn compression: A key to achieve ultra-high weight pruning and quantization rates using admm," *arXiv preprint arXiv:1903.09769*, 2019.

[8] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *ArXiv*, vol. abs/1812.00332, 2018.

[9] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "Robustness may be at odds with accuracy," in *ICLR*, 2018.

[10] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2014.

[11] C. Qin, J. Martens, S. Gowal, D. Krishnan, K. Dvijotham, A. Fawzi, S. De, R. Stanforth, and P. Kohli, "Adversarial robustness through local linearization," in *NeurIPS*, 2019.

[12] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2574–2582, 2016.

[13] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, B. Tran, and A. Madry, "Adversarial robustness as a prior for learned representations," in *ICLR*, 2019.

[14] "CAPTUM model interpretability for pytorch." `https://captum.ai/`. Accessed: 2020-05-10.

[15] B. Li, C. Chen, W. Wang, and L. Carin, "Second-order adversarial attack and certifiable robustness," *ArXiv*, vol. abs/1809.03113, 2018.

[16] V. Sehwag, S. Wang, P. Mittal, and S. Jana, "On pruning adversarially robust neural networks," *ArXiv*, vol. abs/2002.10509, 2020.

[17] H. Zhang, H. Chen, C. Xiao, B. Li, D. S. Boning, and C.-J. Hsieh, "Towards stable and efficient training of verifiably robust neural networks," *ArXiv*, vol. abs/1906.06316, 2020.

[18] D. Kang, Y. Sun, D. Hendrycks, T. Brown, and J. Steinhardt, "Testing robustness against unforeseen adversaries," *ArXiv*, vol. abs/1908.08016, 2019.

[19] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *ICLR*, 2018.

[20] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11256–11264, 2019.

[21] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, and J. Sun, "Metapruning: Meta learning for automatic neural network channel pruning," *ArXiv*, vol. abs/1903.10258, 2019.

[22] X. Li, Y. Zhou, Z. Pan, and J. Feng, "Partial order pruning: For best speed/accuracy trade-off in neural architecture search," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9137–9145, 2019.

[23] S. Jung, C. Son, S. Lee, J. Son, J.-J. Han, Y. Kwak, S. J. Hwang, and C. Choi, "Learning to quantize deep networks by optimizing quantization intervals with task loss," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4345–4354, 2018.

[24] H. Pouransari and O. Tuzel, "Least squares binary quantization of neural networks," *ArXiv*, vol. abs/2001.02786, 2020.

[25] S. Chen, W. Wang, and S. J. Pan, "Metaquant: Learning to quantize by learning to penetrate non-differentiable quantization," in *NeurIPS*, 2019.

[26] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv*, vol. abs/1704.04861, 2017.

[27] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[28] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," *ArXiv*, vol. abs/1905.02244, 2019.

[29] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, 2016.

[30] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Stanford, California: Foundations and Trends in Machine Learning. Vol. 3, No. 1, 2011.

[31] V. Sehwag, S. Wang, P. Mittal, and S. Jana, "Towards compact and robust deep neural networks," *ArXiv*, vol. abs/1906.06110, 2019.

[32] B. Li, S. Wang, Y. Jia, Y. Lu, Z. Zhong, L. Carin, and S. Jana, "Towards practical lottery ticket hypothesis for adversarial training," *ArXiv*, vol. abs/2003.05733, 2020.

[33] S. Ye, K. Xu, S. Liu, H. Cheng, J.-H. Lambrechts, H. Zhang, A. Zhou, K. Ma, Y. Wang, and X. Lin, "Adversarial robustness vs. model compression, or both?," in *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[34] S. Gui, H. Wang, H. Yang, C. Yu, Z. Wang, and J. Liu, "Model compression with adversarial robustness: A unified optimization framework," in *Proceedings of the 33rd Conference on Neural Information Processing Systems*, 2019.

[35] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, pp. 1–122, 01 2011.

[36] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57, 2016.

[37] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. J. Goodfellow, A. Madry, and A. Kurakin, "On evaluating adversarial robustness," *ArXiv*, vol. abs/1902.06705, 2019.

[38] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[39] F. Croce and M. Hein, "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks," *ArXiv*, vol. abs/2003.01690, 2020.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.