THE UNIVERSITY OF CHICAGO

INPUT AND MODEL COMPRESSION FOR ADAPTIVE AND ROBUST NEURAL NETWORKS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY

ADAM DZIEDZIC

CHICAGO, ILLINOIS
AUGUST 2020

Dedicated to my parents & sisters

All models are wrong, but some are useful.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express gratitude to my advisor Professor Sanjay Krishnan for his support, guidance, and motivation. Thank you for honing my skills as a researcher and instructing me in deep work as well as rigor that are key ingredients to good science. I am also greateful to all people whom I met, worked, and collaborated with while studying topics described in this thesis.

ABSTRACT

This dissertation explores compression techniques for neural networks to enable control of resource usage, accelerate training, and increase robustness against adversarial as well as out-of-distribution examples.

The convolutional layers are core building blocks of neural network architectures. In general, a convolutional filter applies to the entire frequency spectrum of the input data. Our new band-limiting method artificially constrains the frequency spectra of these filters and data during training and inference. The frequency-domain constraints apply to both the feed-forward and back-propagation steps. Experimental results confirm that band-limited models can effectively control resource usage (GPU and memory). The band-limited method with 50% compression in the frequency domain results in only a 1.5% drop in accuracy for the ReNet-18 model trained on CIFAR-10 data while reducing the GPU memory usage by 40% and the computation time by 30% in comparison to their full-spectra counterparts. The models trained with band-limited layers retain high prediction accuracy and require no modification to existing training algorithms or neural network architectures, unlike other compression schemes.

Since band-limited models naturally reject high-frequency noise, they are useful for studying the adversarial robustness of neural networks. I show that band-limited models are actually one instance of a broader family of adversarial defenses called perturbation-based defenses. These defenses, whether random or deterministic, are essentially equivalent in their efficacy, and all share similar weaknesses to adaptive attacks. I also illustrate applications to out-of-distribution robustness of models in terms of their generalization and detection of anomalous inputs in the NLP (Natural Language Processing) domain.

The band-limited models have practical applications in a number of time-series and signal processing domains. I present a new solution for a fair wireless co-existence between Wi-Fi Access Points (APs) and LTE-U (Long Term Evolution-Unlicensed) Base Stations (BS) using band-limited neural networks and demonstrate that such networks can accurately learn distinct patterns for the energy distributions in Wi-Fi AP transmissions. Our method results in higher accuracy (close to

99% in all cases) as compared to the existing auto-correlation (AC) and energy detection (ED) approaches.

CHAPTER 1

INTRODUCTION

1.1 Band-limited models

Convolutions are fundamental signal processing operations that amplify certain frequencies of the input and attenuate others. Convolutional layers are an integral part of neural network architectures for computer vision, natural language processing, and time-series analysis [90, 88, 94]. Recent results suggest that neural networks exhibit a *spectral bias* [127, 178]. They ultimately learn filters with a strong bias towards lower frequencies. Most input data, such as time-series and images, are also naturally biased towards lower frequencies [3, 54, 162]. This begs the question—does a convolutional neural network (CNN) need to explicitly represent the high-frequency components of its convolutional layers? We show that the answer to the question leads to some surprising new perspectives on resource management in terms of the training time and model compression as well as on the robustness of models to noisy inputs [48, 92].

Consider a frequency-domain implementation of the convolution function executed in the following steps: (1) transform the filter and the input to the frequency domain; (2) element-wise multiply both frequency spectra; and (3) transform the outcome product to the original domain. Let us assume that the final model is biased towards lower Fourier frequencies [127, 178]. Then, it follows that discarding a significant number of the Fourier coefficients from high frequencies after step (1) should have a minimal effect. A smaller intermediate array size after step (1) reduces the number of multiplications in step (2) as well as memory usage. This gives us a knob to tune the resource utilization, namely, memory and computation, as a function of how much of the high-frequency spectrum we choose to represent. Our primary research question in this thesis is whether we can train CNNs using such *band-limited* convolutional layers, which only exploit a subset of the frequency spectra of the filter and input data.

While there are several competing compression techniques, such as reduced precision arithmetic [82, 117, 136, 171], weight pruning [69, 181], or sparsification [101], these techniques can be

hard to operationalize. CNN optimization algorithms can be sensitive to the noise introduced during the training process, and training-time compression can require specialized libraries to avoid instability [136, 171]. Furthermore, pruning and sparsification techniques only reduce resource utilization during inference. In our experiments, surprisingly, band-limited training does not seem to suffer the same problems and gracefully degrades predictive performance as a function of the compression rate. Band-limited CNNs can be trained with any gradient-based algorithm, where the layer's gradient is projected onto the set of allowed frequencies. The band-limited training stimulates the network to learn low-frequency and robust features instead of high frequency and transient features.

We implement an FFT-based convolutional layer that selectively constrains the Fourier spectrum utilized during both forward and backward passes. Besides, we apply standard techniques to improve the efficiency of FFT-based convolution [112], as well as new insights about exploiting the conjugate symmetry of 2D FFTs, as suggested in [133]. With this FFT-based implementation, we find competitive reductions in memory usage and floating-point operations to reduced precision arithmetic (RPA) but with the added advantage of training stability and a continuum of compression rates.

In addition to the improvement in performance, the band-limited training provides a new perspective on adversarial robustness [122]. Adversarial attacks on neural networks tend to involve high-frequency perturbations of input data [81, 109, 122]. Our experiments show that band-limited training produces models that can better reject noise than their full spectra counterparts. This is a direct consequence of our compression methods, where we discard the high-frequency coefficients that usually correspond to noise. This observation leads us to the exploration of compression techniques for adversarial robustness.

1.2 Robust models

The robustness of neural networks to adversarial examples is one of the crucial requirements for safe deployments of neural networks in natural environments. Surprisingly, the adversarial inputs

themselves are not robust and FFT-based compression of the attacking input often recovers the desired prediction. The recovery techniques expand beyond the regime of compression techniques and are also successful when using random perturbations, such as Gaussian, Uniform, or Laplace noise.

The attacks on Convolutional Neural Networks, such as Carlini & Wanger [22] or PGD [109], generate strategically placed modifications that can be easily dominated by different types of perturbations resulting in correct predictions [51, 79, 135]. This suggests that the standard adversarial examples are not robust. Many defense techniques explicitly leverage this property and can be retrospectively interpreted as perturbations of the input images. However, a detailed understanding of this phenomenon is lacking from the research literature including: (1) what types of perturbations work and what is their underlying mechanism, (2) whether all attacks exhibit this property, and (3) possible counter-measures attackers can employ to defeat perturbation defenses. We address these questions in this thesis.

We can interpret a large number of recent defenses as a type of input perturbations, for example, feature squeezing [177], frequency, or JPEG compression [51], randomized smoothing [33], and perturbation of network structure or the inputs randomly [67, 86, 184]. The defense techniques exhibit very similar gains in robustness. To show it, we start with a simple model where every example is passed through a lossy channel (stochastic or deterministic) before model inference. This channel induces a small perturbation to the input. We optimize the perturbation to be small enough as not to affect the prediction accuracy on clean examples but large enough to dominate any adversarial attack [46]. We find that this trade-off is surprisingly consistent across very different families of input perturbations, where the relationship between channel distortion (the L_2 distance between channel input and output) and robustness is very similar.

Why are some state-of-the-art attacks are sensitive to perturbation-based defenses? We find that many attacks execute an optimization procedure that finds an adversarial image that is very close to the original image in terms of L_1, L_2, or L_∞ norm. The resultant optimum, i.e., the adversarial image, tends to exhibit a higher level of instability than natural examples, which we demonstrate

from the perspective of a first-order and second-order analyses. By instability, we mean that small perturbations of the example can affect the prediction confidences. The unification of perturbation-based defense also gives us insight into how an attacker might avoid them.

Our experiments suggest that all the perturbation based defenses are vulnerable to the same types of attack strategies. We argue that the optimization procedure in the attacker should find the smallest distance from the original image that closes the recovery window. We can devise a generic attacker that attacks a particularly strong lossy channel, based on the additive Laplace noise, and adaptive attacks designed on this channel are often successful against other defenses. This result implies that for many input perturbation defenses the attacker needs not be fully adaptive, i.e., they do not need to know exactly what kind of transformation is used to defend the network. The input perturbation-based defenses are not robust to adaptive attacks that circumvent easily the first defense layer. To make the networks more robust using the randomization technique, all the layers across the whole network should be perturbed and the model should be trained with these perturbations enabled to achieve a high level of robustness.

We also illustrate applications to out-of-distribution robustness of models in terms of their generalization and detection of anomalous inputs in the NLP (Natural Language Processing) domain. We find that pre-trained transformers (BERT and RoBERTa) generalize very well to out-of-distribution examples. They are also more effective at detecting anomalous examples than many previous NLP models, for example, bag-of-words, word embeddings, CNNs, or LSTMs that are frequently worse than chance. On the other hand, we find that compression of pre-trained transformers using distillation decreases their robustness.

1.3 Applications

In the final part of the thesis, we present applications of the aforementioned techniques [49, 166]. To increases network capacity and coverage, especially in crowded public indoor and outdoor areas, LTE (Long-Term Evolution) cellular carriers extend their service to an unlicensed spectrum (LTE-U). However, the unlicensed bands are already highly occupied by Wi-Fi Access Points (AP)

and the LTE-U Base Stations (BS) were not designed for shared spectrum access. We propose how to address the issue by casting the problem to a time-series classification task and using specialized neural networks. Our main approach is to replace heuristics with more robust and general models. We enable LTE-U Base Stations to reliably estimate the number of Wi-Fi Access Point devices. One of the challenges of the project is to collect and simulate an extensive amount of data samples. This allows us to train and test neural network models, such as deep convolutional neural networks, to achieve high levels of accuracy. We find that the Wi-Fi data are highly compressible and the band-limited method can be applied successfully in this setting.

CHAPTER 2

CONTRIBUTIONS

This dissertation contributes:

I. Band-limited Neural Networks:

1. Contributions:

   (a) A novel methodology for training and inference of neural networks using FFT-based convolutions with compression that shows that despite constraining the frequency band, the neural networks can retain high accuracy. This approach requires no modification of the existing training algorithms or neural network architecture, unlike other compression schemes.

   (b) The compression in the frequency domain that accelerates training and inference as well as decreases the memory usage in comparison to the full spectra models.

   (c) The band-limited models that provide robustness to noisy inputs perturbed with standard forms of noise, such as Gauss, Uniform, or Laplace noise.

   (d) The frequency perspective analysis that presents how neural networks start learning from low frequencies and as the training progresses higher frequency coefficients are learned.

   (e) An efficient FFT-based implementation of the band-limited convolutional layer for 1D and 2D data that exploits conjugate symmetry, fast complex multiplication, and frequency map reuse.

   (f) An extensive experimental evaluation across 1D and 2D CNN training tasks that illustrates the aforementioned points: (1) models trained with band-limited layers retain high prediction accuracy and (2) band-limited training can effectively control the resource usage (GPU and memory).

2. Publications:

(a) **Adam Dziedzic**, Ioannis Paparrizos, Sanjay Krishnan, Aaron J. Elmore, Michael Franklin, *Band-limited Training and Inference for Convolutional Neural Networks* [48], ICML 2019.

(b) Sanjay Krishnan, Aaron J. Elmore, Michael Franklin, Ioannis Paparrizos, Zechao Shang, **Adam Dziedzic**, Rui Liu, *Artificial Intelligence in Resource-Constrained and Shared Environments* [92], SIGOPS 2019.

3. Code: `https://github.com/adam-dziedzic/bandlimited-cnns`

II. Perturbation-based Defenses for Neural Networks and Out-Of-Distribution Robustness:

1. Contributions:

(a) A unification of perturbation based defenses that shows that all input perturbation defenses, whether random or deterministic, are equivalent in their efficacy and their underlying mechanisms are very similar. The adversarial attacks transfer between perturbation defenses so the attackers need not know the specific type of defense – only that it involves perturbations.

(b) Analysis of RobustNet that shows that the robustness of the model is independent of the type of injected noise (Gauss, Uniform, or Laplace), it is robust against adaptive attacks (such as PGD with EOT), and its ensemble technique is not useful against strong adversaries.

(c) The first and second-order analyses that show that adversarial examples in a close neighborhood of original inputs show an elevated sensitivity to perturbations.

(d) A combination of RobustNet and adversarial training with loss function for adversarial and standard examples that shows better performance in comparison to pure adversarial training or parameter perturbation with adversarial training.

(e) An extensive experimental evaluation across many NLP models that shows that modern

pre-trained transformers, such as BERT, exhibit better robustness in terms of generalization to out-of-distribution examples and higher detection of anomalous examples.

2. Publications:

   (a) **Adam Dziedzic**, Sanjay Krishnan, *Analysis of Random Perturbations for Robust Convolutional Neural Networks* [46], 2020.

   (b) Dan Hendrycks, Xiaoyuan Liu, Eric Wallace, **Adam Dziedzic**, Dawn Song, *Pretrained Transformers Improve Out-of-Distribution Robustness* [76], ACL 2020.

3. Code:

   (a) Perturbation-based defenses: `https://bit.ly/3gzOGpZ`

   (b) NLP robustness: `https://github.com/camelop/NLP-Robustness`

III. Applications to LTE-U and Wi-Fi Fair Coexistence:

1. Contributions:

   (a) Design, implementation, and validation of a deep learning approach that uses band-limited models to the problem of fair coexistence of LTE-U BS and Wi-Fi APs in the unlicensed spectrum. We demonstrate that there exist distinct patterns between the energy distributions for one and many Wi-Fi AP transmissions. We cast the problem as the time-series classification task. The proposed deep learning approach results in higher accuracy (close to 99% in all cases) as compared to the existing auto-correlation (AC) and energy detection (ED) approaches.

   (b) Application of band-limiting to the coexistence problem. We show that the Wi-Fi data is highly compressible and resilient to compression schemes applied in the band-limited models. Moreover, we find that the data is noisy and the removal of the high-frequency coefficients can increase the accuracy of the model.

(c) A thorough analysis of header, energy, auto-correlation, and machine learning-based approaches to detect and determine the number of Wi-Fi APs. We extend the previous analysis carried out for header, energy, and auto-correlation methods beyond two Wi-Fi APs. We compare several models from the machine learning, deep learning, and time-series fields to finally select the FCN model as the most performant one for the fair coexistence task.

2. Publications:

   (a) **Adam Dziedzic**, Vanlin Sathya, Monisha Ghosh, Sanjay Krishnan *Machine Learning for Fair Spectrum Sharing in Dense LTE Wi-Fi Coexistence* [49], IEEE Open Journal of Vehicular Technology, 2020

   (b) Vanlin Sathya, **Adam Dziedzic**, Monisha Ghosh, Sanjay Krishnan *Machine learning-based detection of multiple Wi-Fi BSSs for LTE-U CSAT* [166], ICNC 2020.

3. Code: `http://bit.ly/2Ob5kAr`

CHAPTER 3

BACKGROUND

3.1 Machine Learning

Machine learning is a field of computer science that uses statistical methods to enable computers to learn and to extract knowledge from data without being explicitly programmed. Machine learning combines applied statistics and computer science. In increases emphasis on the use of computers to statistically estimate complicated functions and decreases emphasis on providing confidence intervals around these functions. The fundamental components of a machine learning algorithm are an optimization algorithm (e.g., a Stochastic Gradient Descent), a cost function (e.g., a Cross-Entropy Loss), a model (e.g., a decision tree), and a dataset.

Machine learning algorithms contain learnable parameters that are adjusted during the training process and fixed during the testing process. There are also non-learnable parameters that are set before the training process and called hyperparameters.

A crucial aspect of machine learning is the ability to learn from data. The notion of learning was defined in [119] as *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves experience E*. The experience E is defined as a dataset on which we train the machine learning algorithms. A dataset is a collection of many examples that are also called data points. The tasks T specify what examples are provided as input and what is expected as output. An example x is a collection of features, where each element of x represents a separate feature. The performance measure P is a quantitative measure of the machine learning algorithm.

In this thesis, we are focused on the following tasks:

1. **Classification** - a machine learning algorithm is required to specify which of k classes a given input belongs to. The machine learning algorithm learns a function $f : \mathbb{R}^n \rightarrow \{1, ..., k\}$, where $y = f(x)$ and the model is taught to assign a class identified by a numeric code y to an input described by tensor x.

2. **Denoising** - a machine learning algorithm is given as input a corrupted example $\tilde{x} \in \mathbb{R}^n$ that was generated by adding some form of noise to a clean example $x \in \mathbb{R}^n$. The learner must predict the clean example x from the corrupted version $\tilde{x}$.

3. **Anomaly detection** - a machine learning algorithm is given a set of inputs in the form of events, objects, or text excerpts, and flags some of them as being atypical or out-of-distribution. An example of an anomaly detection task is detecting examples that are not similar to the examples on which the model was trained. One can train an NLP model for sentiment analysis on movie reviews to classify a given review as positive or negative. During the test time, examples that contain no sentiment (e.g., a description or news articles) should be flagged as anomalous.

The dataset for machine learning algorithms is usually divided into at least train and test sets (in some cases we also create a validation set to tune the hyperparameters). The learnable parameters of machine learning algorithms are tuned using the train set. The test set is used during the testing process (also called inference), where learnable parameters are fixed (and no longer modified), and we check how well the trained model performs on previously unseen data. Generalization is the ability of a learning algorithm to perform accurately on a new, unseen example or task after having experienced a train set. If a model is too simple with respect to the data, it does not fit the training data well and performs poorly both on the train and test sets. In that case, the model is underfitting. If a machine learning model performs well on a train set, but poorly on a test set, then it is overfitting. Overfitting is the situation where a model is too complex with respect to the data. For example, it can perfectly fit the train set, however, it is adapted too much to the train set and performs poorly on the test set. It simply does not generalize well.

Machine learning algorithms can be classified into supervised and unsupervised algorithms. In case of the supervised learning, the algorithm is provided input pairs (x_i, y_i), where x_i is the input to the algorithm and y_i is its output. The task is to find a function that correctly maps from x_i to y_i. Unsupervised learning algorithms try to find structure in the data without explicitly being provided with labels. For example, the k-means unsupervised learning algorithm tries to find optimal k

clusters in the data.

Machine learning algorithms can be also divided into traditional learning algorithms and deep learning algorithms. Traditional learning algorithms usually have fewer learnable parameters than deep learning algorithms and smaller learning capacity. The traditional learning algorithms are not designed to do feature extraction and users have to design a data representation that is then fed to the learning algorithms. The traditional machine learning algorithms (e.g. Decision Tree, Random Forest, AdaBoost, and SVM) work well on various important problems, however, they have not succeeded in solving the central problems in AI (Artificial Intelligence), such as recognizing speech or recognizing objects. The development of deep learning was motivated by the failure of the traditional machine learning algorithms to generalize on such AI tasks. Other important aspects that accelerated the development of deep learning were increased amount of collected data as well as access to the novel hardware (GPUs or TPUs) that enabled us to efficiently process the large amount of data using deep learning algorithms (described in Section 3.2).

In the following sections, we describe traditional supervised machine learning (ML) algorithms used for a classification task, which are applied in Section 6.7.3 to learn distinguished patterns of signals for a different number of Wi-Fi devices.

3.1.1 Decision Tree

Decision Tree is a simple non-parametric model that learns decision rules inferred from the data features. The deeper the tree, the more complex the decision rules, and the fitter the model. There are many techniques to regularize the model, for instance, by specifying the maximum depth of the tree or the minimum number of samples required to be at a leaf node.

A simple decision tree created on a medical dataset is presented in Figure 3.1. The first line in each box represents a name of the feature and a rule that leads to either the left or the right branch (also called a split). Given training vectors $x_i \in \mathbb{R}^n$, $i = 1,\ldots,N$ and a label vector $y_i \in \{0,1,\ldots,K-1\}$, a decision tree recursively partitions the space such that the samples with the same labels are grouped together in leaf nodes of the decision tree. Let as denote training points in

a node m by X_m and the proportion of class k training points in the node m by:

$$p_{mk} = \frac{1}{|X_m|} \sum_{x_i \in X_m} \mathbb{1}_{\{y_i = k\}} \tag{3.1}$$

Then, the measure of impurity of a node in the decision tree can be expressed a a Gini index:

$$H_G(X_m) = \sum_k p_{mk}(1 - p_{mk}) = \mathbb{E}_{x \sim p_m}(1 - p_m(x)) = \mathbb{E}_{x \sim p_m} I_G(x) \tag{3.2}$$

or by the standard measure from the information theory, being Entropy:

$$H_E(X_m) = \sum_k p_{mk} - \log(p_{mk}) = \mathbb{E}_{x \sim p_m} - \log(p_m(x)) = \mathbb{E}_{x \sim p_m} I_E(x) \tag{3.3}$$

Note that the internal expressions used in the definition of the Gini index:

$$I_G(x) = (1 - p_m(x)) \tag{3.4}$$

$$I_G(x) \in [0, 1] \tag{3.5}$$

and in the definition of Entropy:

$$I_E(x) = -\log p_m(x) \tag{3.6}$$

$$I_E(x) \in [0, +\infty) \tag{3.7}$$

are two slightly different definitions of a self-information. The differences are in their ranges and the exact mapping from a given event to its information gain. The self-information metrics, for less likely events, have higher values, hence more information content. Analogously, for more likely events, they have smaller values, hence less information content. Thus, the self-information can quantify our intuition about information. Intuitively, when there is a dominant value/class in a dataset (X_m), then the value of the Gini index or Entropy is low. On the other hand, when all the values/classes in a dataset (X_m) appear in an equal probability, the corresponding value of the Gini

index or Entropy is the highest. The Gini index and Entropy quantify the amount of uncertainty in a probability distribution. Usually, the Gini index works better in practice for Decision Trees.

The Decision Tree presented in Figure 3.1 uses the Gini index to express its impurity in nodes and this is expressed as the second line in each node in the diagram. The third row shows how many data samples are contained in a given node. The tree was created for a binary classification task with labels *neg* and *pos*. The fourth line presents the number of examples that belong to the *neg* class (the first value) and to the *pos* class (the second value). The last line is the majority label in a node.

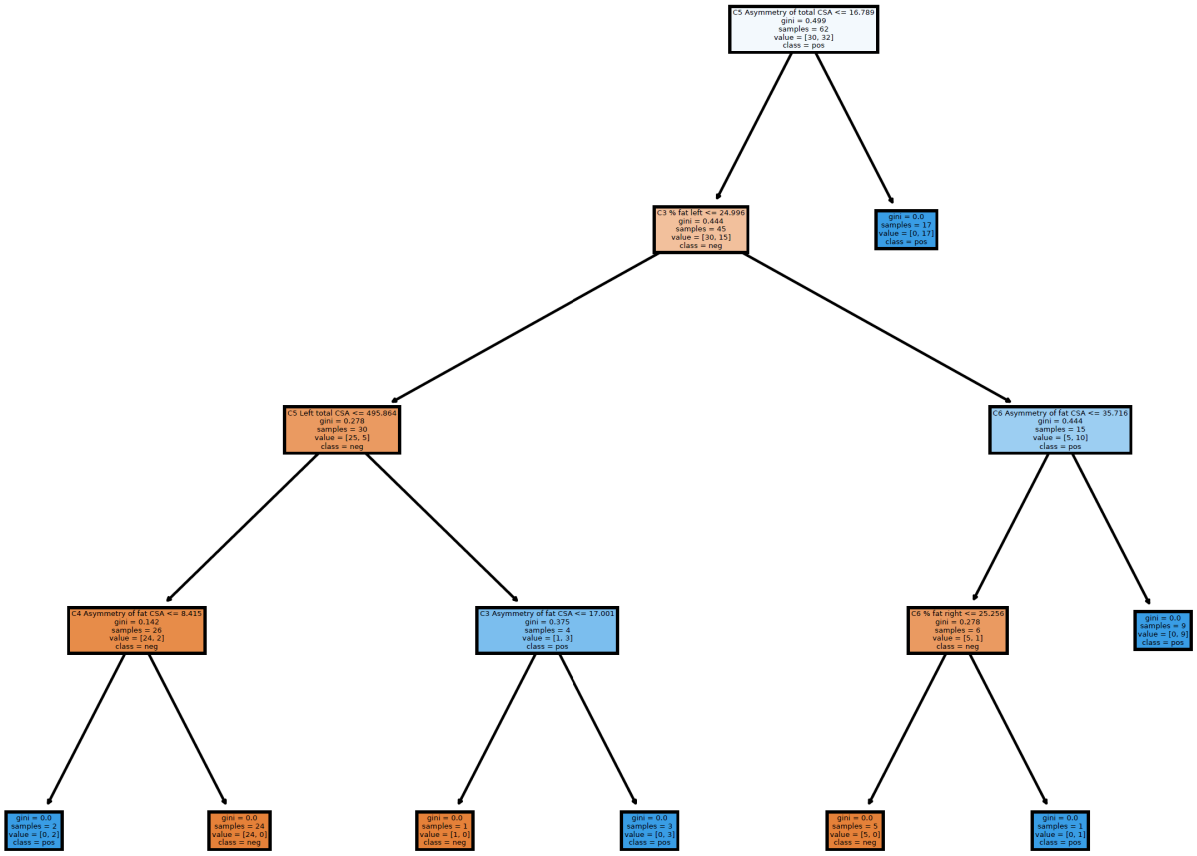

Figure 3.1: *An example of a simple Decision Tree classifier.*

3.1.2 Random Forest

Random Forest is a bagging algorithm (type of model averaging) based on Decision Trees [16, 59, 130]. Bagging is a boostrap aggregation method that learns a final *ensemble* predictor h by aggregating predictors h_b learned over multiple random draws (bootstrap samples) from a training data. In case of random forest, the constituent predictors h_b are Decision Trees. With bagging, we compute B different bootstrap samples, learn a predictor h_b for each bootstrap sample, and then aggregate h_b's to form the final predictor h. There are two standard forms of aggregation: majority vote (consensus) and probability aggregation. In case of the binary classifiction, where $\mathcal{Y} \in \{0,1\}$, we define the aggregation techniques in the following way:

1. **Majority vote (consensus)** – is provided with $h_b(x) \in \{0,1\}$ and the final output is computed as:

$$h(x) = \begin{cases} 0 \text{ if } \sum_{b=1}^{B} \mathbb{1}_{\{h_b(x)=0\}} > \sum_{b=1}^{B} \mathbb{1}_{\{h_b(x)=1\}} \\ 1, \text{ otherwise} \end{cases} \tag{3.8}$$

2. **Probability aggregation** – each aggregating predictor h_b provides $p_b(x)$, which is an estimate of:

$$\mathbb{E}[Y|X=x] = \mathbb{P}[Y=1|X=x] \tag{3.9}$$

and we compute the final output as:

$$p(x) = \frac{1}{B}\sum_{b=1}^{B} p_b(x) \tag{3.10}$$

$$h(x) = \begin{cases} 0 \text{ if } p(x) \leq \frac{1}{2} \\ 1, \text{ otherwise} \end{cases} \tag{3.11}$$

We use the probability aggregation method (the reasoning behind this choice is provided in 5.4.3). The standard Decision Trees are interpretable models and computationally efficient, however, they

introduce high variance and small perturbations in training data can yield very different splits (classification rules). On the other hand, Random Forests are characterized by less variance and good predictive accuracy. They are not interpretable and computationally complex but, in our experiments, we do provide enough computational power to train them.

3.1.3 AdaBoost

AdaBoost [70, 130] classifier is one of the best out-of-the-box models in the *sklearn* library that creates an ensemble of classifiers. Similarly to bagging, boosting is a method to combine multiple classifiers trained on different versions of the training data. The different versions of the data correspond to different weights applied to different samples and we aggregate the constituent classifiers by computing a weighted sum of them. The key idea behind the weights is that if individual classifiers $h_1, h_2, \ldots, h_t$ are regularly misclassifying a sample x_i, then we want to assign to x_i a large weight when we train h_{t+1} classifier, to help ensure it is correctly classified. One of the main differences between bagging (used in Random Forests) and boosting (used in AdaBoost) is that in the case of bagging we treat each classifier equally. On the other hand, boosting assigns weights to classifiers – they are not treated equally.

3.2 Deep Learning

Deep learning is a specific type of machine learning [64]. The central algorithms in deep learning are based on deep neural networks. The input to a deep neural network can be raw data (e.g., raw images), and an artificial intelligence specialist does not have to design a data representation (do feature extraction) for the neural networks.

Feed-forward (also called multi-layer perceptrons (MLP) or fully-connected) neural networks consist of multiple layers. Figure 3.2 depicts a simple neural network with four layers. The first layer is the input layer, and the last layer is the output layer. The two layers between the input and output layers are called hidden layers.

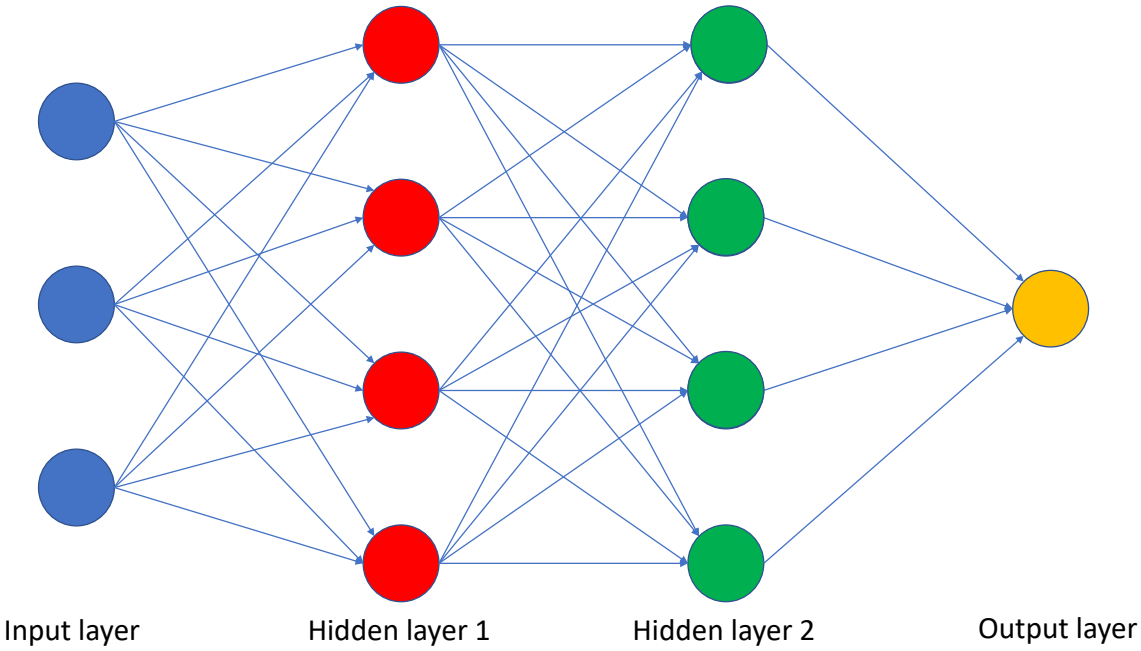

Figure 3.2: *A feed-forward neural network with input layer, two hidden layers, and output layer.*

The input examples x are fed to the neural network as input layer and the output layers return the results y, which can be, for instance, a numerical representation of a predicted class. The learning procedure of neural networks uses a backpropagation algorithm, which combines a loss function and an optimizer. The backpropagation consists of a forward pass and a backward pass. In the forward pass, the input data x is provided as the input layer of the neural network, next the data is passed through the hidden layers, and output y is obtained at the end. The loss between the ground truth y and the prediction $\hat{y}$ is calculated, and then in the backward pass, neural networks' parameters are tuned with respect to the loss.

We assume some set $\mathcal{X}$ of possible inputs, set $\mathcal{Y}$ of possible outputs, and a parameter vector θ. For $\theta \in \mathbb{R}^d$, $x \in \mathcal{X}$, and $y \in \mathcal{Y}$, a deep network computes a probability $P_\theta(y|x)$, which is a probability parameterized by θ of output (e.g., a class) y, given input x.

The fundamental equation of deep learning [114] is:

$$\theta^* = \arg\min_\theta E_{(x,y)\sim \text{Pop}} - \ln P_\theta(y|x) \tag{3.12}$$

where we assume a *population* distribution Pop on pairs (x, y), and the loss function $\mathcal{L}(x, y, \theta) = -\ln P_\theta(y|x)$, which is called the cross-entropy loss.

The second fundamental equation of deep learning is a softmax function that converts scores (also called logits) to probabilities:

$$\underset{y}{\text{softmax}}\, s_\theta(y|x) = P_\theta(y|x) = \frac{1}{Z} e^{s_\theta(y|x)} \tag{3.13}$$

where Z is defined as:

$$Z = \sum_y e^{s_\theta(y|x)} \tag{3.14}$$

3.3 Convolutional Neural Networks

Convolutional Neural Networks (abbreviated as CNNs or ConvNets) are specialized types of neural networks that are applied primarily to computer vision tasks [94]. However, they also find usage in other domains, such as NLP [106], speech [88], and time-series [12].

CNNs usually consists of many blocks, where each block contains many layers (see Figure 3.3). Traditionally, CNNs include convolution, batch normalization, ReLU, and max-pooling layers. The most important layer in CNNs is the convolutional layer. Its parameters are made of learnable kernels that are also called filters. Convolutional layers apply a convolution operation to the input. The convolution operation reduces the number of learnable parameters, in comparison to the fully-connected layers, which makes the neural network easier to train (this is also called parameter sharing). A filter is applied/convolved with input (e.g., an image) and a new feature map is returned.

The batch normalization layer was introduced in [85] and its main task is to normalize its input by re-centering and re-scaling it. This layer improves the speed, performance, and stability of neural networks.

The ReLU layer (ReLU $= max(0, x)$) is a non-linearity function that enables a neural network to be a universal function approximator. The sigmoid and tanh functions are other commonly used non-linearities. The ReLU function is by far the most popular activation function because

usually it makes neural networks train more efficiently compared to other activation functions. For example, ReLU helps to avoid the vanishing gradient problem that is frequently observed when using sigmoid activations.

The pooling layer transforms its input to an output that contains summary statistics of the nearby pixels. The max pooling [185] operation returns the maximum output within a rectangular neighborhood. Other popular pooling operations include averaging, L_2 norm of the rectangular neighborhood, or a weighted average (based on the distance from the central pixel) [64].

The last layer of CNNs is usually a fully connected layer that for classification tasks maps outputs from convolutional layers (that serve as feature extractors) to the final class. Fully connected layers connect every neuron in one layer to every neuron in another layer, as seen with the two hidden layers in Figure 3.2.

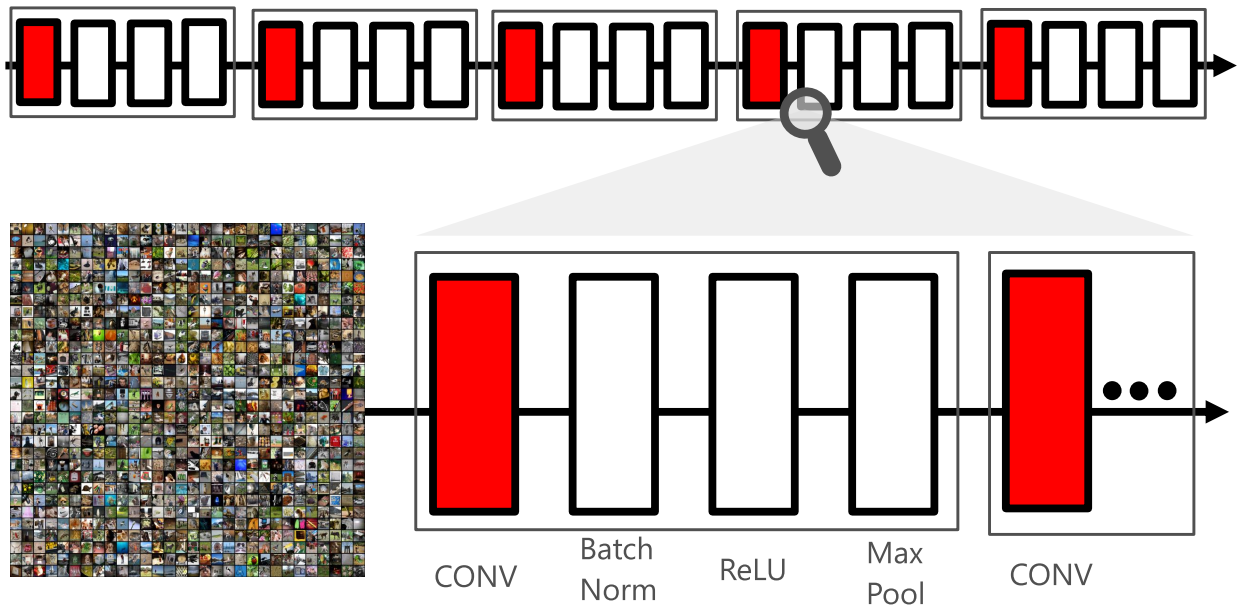

Figure 3.3: **Convolutional Neural Network**. *A schema of a CNN with many blocks and convolutional layers shown in red.*

The need for fast convolutional neural networks is clear. The training and inference are both limited by the convolution speed. Faster convolution means we can train larger models (since it takes GPU-days of computation to train on large data sets) and we want to execute the inference with lower latency, for example, for pedestrian detection in self-driving cars. There are different

types of convolutional algorithms. Their speed depends on many factors, for instance, the size of filters (kernels), memory budget, or the underlying processing unit (CPU or GPU). In the next sections, we present a comparison between convolutional algorithms and summarize the computational complexity of the algorithms in Table 3.1.

Table 3.1: Comparison of computational complexity between convolutional algorithms. For the 1D case, n denotes an input size and k is the filter size. (*) – the Winograd convolution is not practical for big filters (see Section 3.3.3)

Algorithm	**Small filter**	**Big filter**
Direct	$O(nk)$	$O(n^2)$
FFT	$O(n \log n)$	$O(n \log n)$
Winograd	$O(n)$	$O(n^*)$

3.3.1 Direct Convolution

The impulse decomposition allows signals to be examined one point at a time, leading to the powerful technique of convolution. Convolution is a mathematical operation that combines two inputs (e.g., signals) to form a single output (signal). It is the single most important technique in DSP (Digital Signal Processing) [152]. The simplest version of 1D convolution (formally cross-correlation) between input data D and filter G of size R can be defined as:

$$Y_x = \sum_{v=1}^{R} D_{x+v} G_v \tag{3.15}$$

The convolution operation is represented as $*$:

$$Y = D * G \tag{3.16}$$

There are a few ways to compute the convolution operation. The standard approach is to *slide* a filter across an input. This operation is also known as a sliding dot product or sliding inner-product. This is a direct implementation of the definition of the convolution operation. From the performance perspective, this method is computationally expensive. In 1D case, its computational

complexity is $O(nk)$, where n is the input size and k is the filter size, however, this method requires no additional memory buffers. In practice, many machine learning frameworks implement cross-correlation operation but call it convolution [64]. [1] The cross-correlation is very similar to convolution but without flipping the kernel. A 2D convolution operation (formally called cross-correlation) to compute a single output layer Y for filter G of size $R \times S$ and input D with C channels can be expressed as:

$$Y_{i,k,x,y} = \sum_{c=1}^{C} \sum_{v=1}^{R} \sum_{u=1}^{S} D_{i,c,x+v,y+u} G_{k,c,v,u} \tag{3.17}$$

where i denotes a mini-batch index. If we use the $*$ representation, the equation 3.17 can be simplified to:

$$Y_{i,k} = \sum_{c=1}^{C} D_{i,c} * G_{k,c} \tag{3.18}$$

3.3.2 FFT-based Convolution

Fourier analysis is a method for representing a function as a sum of periodic components, and for recovering the function from those components. When both the function and its Fourier transform are replaced with discretized counterparts, it is called the Discrete Fourier Transform (DFT).

FFT (Fast Fourier Transform) is a fast way to compute DFT [35]. FFT has revolutionized entire industries (especially signal processing [2]), by accelerating the Fourier transform [155]. The main idea behind the (discrete) Fourier transformation is to represent a function f as a sum of harmonics $c_k e^{ikx}$, where k is the index of Fourier coefficient, c_k is a complex coefficient, $i = \sqrt{-1}$ is an imaginary number, and x represents an angle $\frac{2\pi j}{n}$. For $n = 4$ coefficients and $j = \{0,1,2,3\}$, we consider the following equally spaced angles $\{0, \frac{2\pi}{4}, \frac{4\pi}{4}, \frac{6\pi}{4}\}$. We also call this representation a polar form, since we work primarily with angles. The expression $e^{2\pi i/n}$ is also called the principal n-th root of unity. The function f is seen in the frequency domain (space) through the coefficients

1. We adhere to this nomenclature and use the name convolution instead of cross-correlation.

2. In the context of DSP, *signal* is customarily defined as a discretized input in the physical domain, while the output of the Fourier transform is called a *spectrum* that exists in the frequency domain.

c_k, instead of in the physical domain (space, also known as the time in 1D or spatial in 2D), where it is seen through its values $f(x)$. We apply the Fourier transform to go between the coefficient c_k's and function values $f(x)$'s. The fast transformation between the frequency and physical spaces is via FFT. The standard transformation for input of size $K = n$ uses n^2 multiplications, however, FFT uses only $\frac{1}{2} n \log_2 n$ multiplications (in 1D case).

There are many ways to define DFT and they vary in the sign of the exponent or normalization [3]. In this thesis, we define 1D DFT (forward: from the physical to frequency domain) as:

$$c_k = \sum_{j=0}^{n-1} y_j \exp\{-2\pi i \frac{jk}{n}\} \quad k = 0, 1, \ldots, n-1 \tag{3.19}$$

The 1D inverse DFT (backward: from the frequency to physical domain) is defined as:

$$y_j = \frac{1}{n} \sum_{k=0}^{n-1} c_k \exp\{2\pi i \frac{jk}{n}\} \quad j = 0, 1, \ldots, n-1 \tag{3.20}$$

The $y_j = f(x)_j$ is the j-th value of function f at $x = \frac{2\pi j}{n}$. The difference between the forward and backward transforms is in the sign in the exponent and the normalization factor $\frac{1}{n}$.

If the input signal is represented only by real values, the Fourier transform is Hermitian and a component at frequency F_k is the complex conjugate of the component at frequency $-F_k$. Thus, the information in the positive frequency components is sufficient to fully re-create the real signal and there is no information in the negative frequency components. We exploit this symmetry by computing only the positive frequency components, up to and including the Nyquist frequency F_N, which is half of the sampling rate of a discrete signal. Thus, for n input real values we obtain $\left\lceil \frac{n}{2} + 1 \right\rceil$ complex output values.

3. We follow the conventions used in Numpy: `https://numpy.org/doc/stable/reference/routines.fft.html#module-numpy.fft` and PyTorch: `https://pytorch.org/docs/master/generated/torch.ifft.html#torch.ifft`

DFT can be extended to 2D as:

$$c_{kl} = \sum_{j=0}^{n-1} \sum_{p=0}^{r-1} y_{jp} \exp\{-2\pi i(\frac{jk}{n} + \frac{pl}{r})\} \quad k = 0,1,\ldots,n-1 \quad l = 0,1,\ldots,r-1 \tag{3.21}$$

In 2D, the Fourier transform is used, for example, for image analysis.

The FFT-based convolution for CNNs was first proposed by [112] and then refined by [168]. This convolution operation starts from transforming the data input x and filter y from the spatial or time domain to the frequency domain via FFT (see Figure 3.4). Next, it executes the convolution operation in the frequency domain by element-wise multiplying (also known as Hadamard product) the transformed input x and filter y. Finally, it goes back to the spatial or time domain via inverse FFT (IFFT) and returns the output which is data x convolved with filter y. The complexity of this operation in 1D is $O(n \log n)$, where n is the input size and we assume that $n >> k$, where n is the filter size. The FFT transformation dominates the computation and the complexity of the FFT-based convolution becomes independent of the filter size. Thus, the FFT-based convolution is usually most efficient for filters of size $k > 5$ (or larger than 5×5 in 2D) and uses additional memory workspace for intermediate results, one of the issues that we address in this thesis.

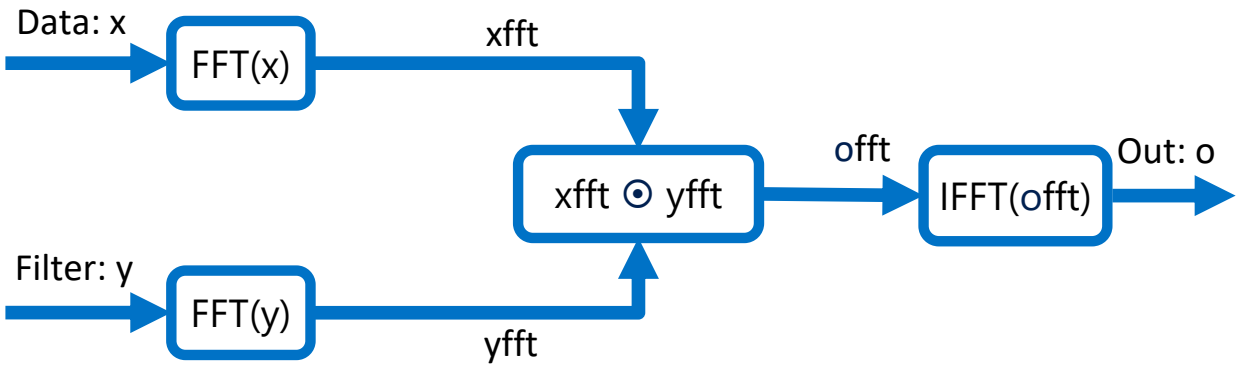

Figure 3.4: **Schema of the FFT-based convolution.**

3.3.3 *Winograd Convolution*

For completeness, we also discuss the Winograd convolution for CNNs, which was presented in [97].[4] This work also presents an algebraic framework for studying the arithmetic complexity of convolution operations. The authors leverage the framework to present practical algorithms, which give the fastest convolutional implementation available on GPUs for small filters of size $3x3$. The fundamental result is that **the number of multiplications required by the Winograd algorithm is equal to the input size**. Thus, we reduce the complexity to a single multiplication per single input value.

We directly compare the computation needed for direct convolution and the Winograd convolution when convolving input data D with filter G to obtain output Y.

$$D = [d_0, d_1, d_2, d_3] \tag{3.22}$$

$$G = [g_0, g_1, g_2] \tag{3.23}$$

$$Y = D * G \tag{3.24}$$

First, in the equations below (starting from 3.25), we show the direct convolution, where to compute the output Y we need 6 multiplications and 4 additions.

4. I provide the implementation of the Winograd algorithm here: `https://github.com/adam-dziedzic/winograd`

$$Y_0 = [d_0, d_1, d_2] \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} \tag{3.25}$$

$$Y_1 = [d_1, d_2, d_3] \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} \tag{3.26}$$

$$Y = \begin{bmatrix} d_0, d_1, d_2 \\ d_1, d_2, d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} \tag{3.27}$$

$$Y = \begin{bmatrix} d_0 \times g_0 + d_1 \times g_1 + d_2 \times g_2 \\ d_1 \times g_0 + d_2 \times g_1 + d_3 \times g_2 \end{bmatrix} \tag{3.28}$$

Winograd ([175] page 43) documented the following minimal algorithm:

$$Y = \begin{bmatrix} d_0, d_1, d_2 \\ d_1, d_2, d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} \tag{3.29}$$

$$Y = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix} \tag{3.30}$$

$$m_1 = (d_0 - d_2) \times g_0 \tag{3.31}$$

$$m_2 = (d_1 + d_2) \times \frac{g_0 + g_1 + g_2}{2} \tag{3.32}$$

$$m_3 = (d_2 - d_1) \times \frac{g_0 - g_1 + g_2}{2} \tag{3.33}$$

$$m_4 = (d_1 - d_3) \times g_2 \tag{3.34}$$

This algorithm requires only 4 multiplications but also 11 additions/subtractions in total, and 2 bit shifts. There are 3 additions/subtractions within elements of the filter G (the addition $g_1 + g_2$ can be computed once), there are 4 additions/subtractions involving the data D, and 4 additions/-subtractions to reduce the m_i's to the final results. Thus, in comparison to the direct convolution, the Winograd algorithm reduces the number of multiplications from 6 to 4. However, the number of additions/subtractions increases from 4 to 11 and we have additional relatively inexpensive 2 bit shifts. [5] It has been known since at least 1980 that the minimal convolution algorithm for computing m outputs with a filter of size k, which is called $F(m,k)$, requires:

$$\mu(F(m,k)) = m + k - 1 \tag{3.35}$$

multiplications. Thus, the Winograd algorithm is minimal. The number of additions and constant multiplications required by the minimal Winograd transforms increase quadratically with the kernel size ([108] page 211). For large kernels, usually larger than 5×5 for the 2D case, the complexity of the transforms will overwhelm any savings in the number of multiplications.

The Winograd algorithm can be written in the matrix form in the following way:

$$Y = A^T[(Gg) \odot (B^T d)] \tag{3.36}$$

The operation $\odot$ is the Hadamard product (the element-wise multiplication), and the additional matrices are defined as:

5. We can pre-compute and cache the additions/subtractions within the filter G that reduces the total additions/subtractions required to 8.

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \tag{3.37}$$

$$G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{3.38}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix} \tag{3.39}$$

$$g = \begin{bmatrix} g_0 & g_1 & g_2 \end{bmatrix}^T \tag{3.40}$$

$$d = \begin{bmatrix} d_0 & d_1 & d_2 & d_3 \end{bmatrix}^T \tag{3.41}$$

A minimal 1D algorithm defined in equation 3.36 can be nested with itself to obtain a minimal 2D algorithm:

$$Y = A^T[(GgG^T) \odot (B^T dB)]A \tag{3.42}$$

Using equation 3.42 to compute a convolution in 2D between an input of size 4×4 and a filter of size 3×3, and to return an output of size 2×2, requires 16 multiplications (which is the number of elements in the input: 4×4). On the other hand, computing the convolution using the direct algorithm requires $2 \times 2 \times 3 \times 3 = 36$ multiplications.

3.4 Adversarial Machine Learning

If we train neural network models and test them on IID (Identically and Independently Distributed) examples (they are also commonly referred to as the in-distribution examples), then the neural

networks achieve human-level performance on many machine learning tasks. Thus, we should test if neural networks extract the same features from the examples as human beings do. It occurs that when we perturb an input x by adding an imperceptible (for humans) vector noise ε that can be added to x to produce $x' = x + \varepsilon$, then we are able to fool the neural network model such that $f(x) \neq f(x')$, where $f(\cdot)$ denotes a class returned by a neural network for a given input, e.g., $f(x)$ is the original class returned by the neural network for the unperturbed example x. Interestingly enough, there are also adversarial examples that can fool human-beings. For example, Figure 3.5 presents concentric circles and not intertwined spirals [10] (this is Figure 12c in the original work).

There are different explanations for why we can easily find adversarial examples for neural networks. One of the hypotheses is that the primary reason for the adversarial examples is excessive linearity [65]. Neural networks consist mostly of linear layers and the final functions that they express can be highly linear. If the input examples are high dimensional, then the simple linear function $y = w(x + \varepsilon)$ can change substantially despite the small change ε in the input. The adversarial examples can also be attributed to the presence of non-robust features, i.e., features derived from patterns in the data distribution that are highly predictive, yet brittle and incomprehensible to humans [84]. There are other explanations of this phenomena and the community has not reached a consensus on what is the main reason for these adversarial examples.

There have been many unsuccessful defenses against adversarial examples and the general agreement is that it is easy to attack neural networks. Up to date, the most used technique to provide some level of empirical robustness has been adversarial training. The method trains neural networks on adversarial examples and encourages a model to be robust to such inputs. Other commonly used techniques against adversarial examples harness randomization, for example, by perturbing all internal layers in neural networks (which we elaborate on in Chapter 5 in this thesis) or pre-training (by providing a much larger dataset on which the model is trained first before tuning it on the target dataset).

The other machine learning models are also susceptible to adversarial noise. Adversarial training of linear models, such as linear regression or SVM (with a linear kernel), is less useful and

very similar to weight decay since these models cannot learn a step function, which in case of adversarial training on neural networks is possible and encourages the networks to be locally constant in the neighborhood of the training data. Regarding other models, for instance, the k-NN (k Nearest Neighbors) model is prone to overfitting when trained adversarially. The main upshot is that neural networks can actually become more secure than other models. Adversarially trained neural networks exhibit the best empirical success rate on adversarial examples of any machine learning models [83].

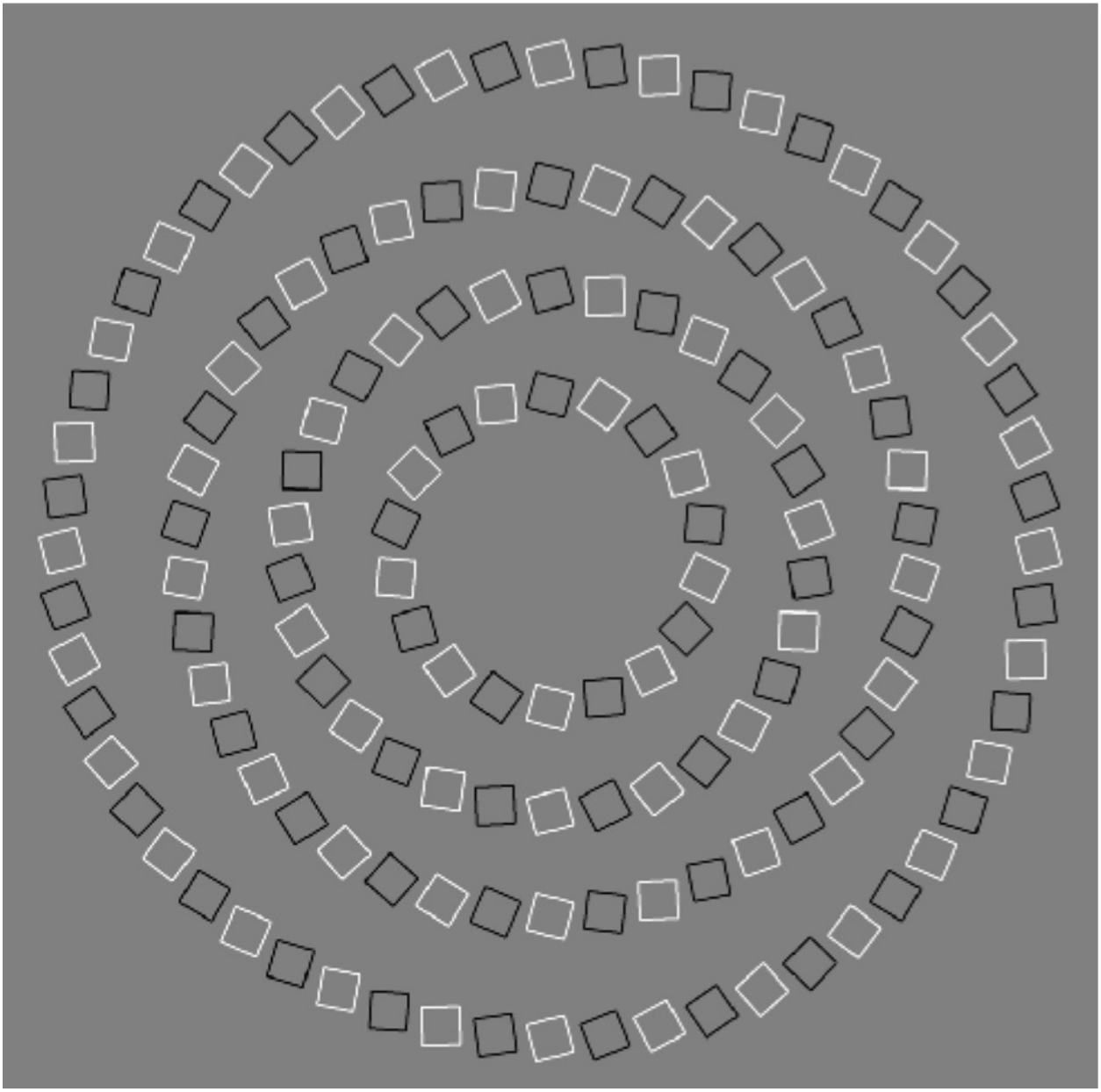

Figure 3.5: An example of an illusion (adversarial example) for human beings. These are not intertwined spirals but concentric circles.

CHAPTER 4

BAND-LIMITED TRAINING AND INFERENCE

4.1 Introduction

We propose a novel methodology for band-limited training and inference of CNNs that constrains the Fourier spectrum utilized during both forward and backward passes. Our approach requires no modifications of the existing training algorithms or neural network architectures, unlike other compression schemes. An efficient FFT-based implementation of the band-limited convolutional layer for 1D and 2D data that exploits conjugate symmetry, fast complex multiplication, and frequency map reuse. An extensive experimental evaluation across 1D and 2D CNN training tasks illustrates: (1) band-limited training can effectively control the resource usage (GPU and memory) and (2) models trained with band-limited layers retain high prediction accuracy.

4.2 Related Work

Model Compression: The idea of model compression to reduce the memory footprint or feed-forward (inference) latency has been extensively studied (also related to distillation) [27, 74, 77, 149]. The ancillary benefits of compression and distillation, such as adversarial robustness, have also been noted in prior work [81, 109, 122]. One of the first approaches was called weight pruning [69], but recently, the community is moving towards convolution-approximation methods [28, 102]. We see an opportunity for a detailed study of the training dynamics with both filter and signal compression in convolutional networks. We carefully control this approximation by tracking the spectral energy level preserved.

Reduced Precision Training: We see band-limited neural network training as a form of reduced-precision training [5, 39, 82, 144]. Our focus is to understand how a spectral-domain approximation affects model training and we hypothesize that such compression is more stable and gracefully degrades compared to harsher alternatives.

Spectral Properties of CNNs: There is substantial recent interest in studying the spectral properties of CNNs [127, 133, 178], with applications to better initialization techniques, theoretical understanding of CNN capacity, and eventually, better training methodologies. More practically, FFT-based convolution implementations have been long supported in popular deep learning frameworks (especially in cases where filters are large in size). Recent work further suggests that FFT-based convolutions might be useful on smaller filters as well on CPU architectures [186].

Fully Spectral CNNs: The whole CNN can be learned in the frequency domain with the FFT transformations needed only on the boundaries of the network [89]. Kernels (for convolution) are kept small in the spatial domain and converted to the frequency domain only when needed for the forward or backward passes. The sinc operation (also known as the cardinal sinusoid function) is used to interpolate kernels to the size of an image (or the input map to convolution). Experiments show the same accuracy of CNNs in both the frequency and the spatial domains. The spectral (FFT based) pooling is used for compression in the frequency domain. Instead of ReLU non-linearity (which requires convolution in the frequency domain), tanh and sigmoid non-linearities are approximated by linear functions.

Data transformations: Input data and filters can be represented in Winograd, FFT, DCT, Wavelet, or other domains. In our work, we investigate the most popular FFT-based frequency representation that is natively supported in many deep learning frameworks (e.g., PyTorch) and highly optimized [167]. Winograd domain was first explored in [97] for faster convolution but this domain does not expose the notion of frequencies. An alternative DCT representation is commonly used for image compression. It can be extracted from JPEG images and provided as an input to a model. However, for the method proposed in [66], the JPEG quality used during encoding is 100%. The convolution via DCT [131] is also more expensive than via FFT.

Small vs Large Filters: FFT-based convolution is a standard algorithm included in popular libraries, such as cuDNN [1]. While alternative convolutional algorithms [97] are more efficient for small filter sizes (e.g., 3x3), the larger filters are also significant. (1) During the backward pass, the

1. https://developer.nvidia.com/cudnn

gradient acts as a large convolutional filter. (2) The trade-offs are chipset-dependent and [186] suggest using FFTs on CPUs. (3) For ImageNet, both ResNet and DenseNet use 7x7 filters in their 1st layers (improvement via FFT noted by [167]), which can be combined with spectral pooling [134]. (4) The theoretical properties of the Fourier domain are well-understood, and this study elicits the frequency domain properties of CNNs.

Winograd convolution: The convolution operation can be also accelerated by Winograd's minimal filtering algorithm [97]. Lavin and Gray present an efficient convolution in the Winograd domain and an algebraic framework for studying the arithmetic complexity of the operation. The authors show experimentally that their approach is the fastest CNN implementation available for GPUs for filter sizes 3x3. In this case of small 3x3 filters, the algorithm requires only a single multiplication per an input element whereas the FFT based approach requires about 1.5 real multiplications. Moreover, the memory requirements are much less for the Winograd convolution, for example, for $F(2x2, 3x3)$ transformations, the Winograd convolution either does not use any global memory workspace or up to 16 MB when a filter transform kernel is run first and stored in a workspace buffer. In contrast, for this setting, the FFT-based convolution uses up to 2.6 GB of additional memory in their experiments. This further motivates the need to reduce the memory requirements in the frequency domain.

Reinforcement learning: Our band-limited technique was used in reinforcement learning to reduce the interdependency between the low and high-frequency components of the state-action value approximation, allowing the critic, in the actor-critic algorithm, to learn faster [19]. During the policy improvement step (actor-network), Monte Carlo sampling in the action space averages out rapid variations of the state-action value function; that is, filters out high-frequency components. During the policy evaluation step, the critic network prioritizes the learning of slow variations (low-frequency components) of the state-action value function. This is an example of the spectral bias of deep neural networks. The band-limiting method has a positive impact on convergence because it encourages the critic to learn the most relevant low-frequency components.

Robust Learning with Frequency Domain Regularization Another extension of our band-limited

convolutions proposes a different approach to discarding the frequencies [68]. In our work, we compared methods based on lead (retain the highest frequency coefficients, the leading ones, and remove the smallest frequency coefficients) and tail (remove the highest frequency coefficients) approaches with static, dynamic, and hybrid levels of compression ratios used. The new work proposes to selectively discard frequency coefficients by designing a mask that in the forward pass is binary and in the backward pass maintains real values in the range $[0, 1]$. The mask is binarized in the forward pass by rounding its real values. The mask is element-wise multiplied with the output of the convolution in the frequency domain and trained with back-propagation. The authors show that such an approach to the removal of frequency coefficients can discard a significant portion of the frequencies, especially within the rear layers of neural networks, and provide higher accuracy as well as the robustness of models trained with the masks. For a given convolution operation, the size of the mask is the same as the output of the convolution in the frequency domain, thus there is a separate mask for each channel, and the elements in the mask control selectively all the frequency coefficients to find the most relevant ones.

4.3 Band-limited Convolution

4.3.1 Design

We define band-limiting as masking out high-frequency coefficients. In Figure 4.1, we present a high-level overview of the band-limited convolution operation, which can also be called an FFT convolution with compression. The input is data x and filter y, which we transform via FFT to the frequency domain. In the red boxes, we band-limit the input x and filter y in the frequency domain by discarding high-frequency coefficients. Thus, the red boxes indicate where we use less memory. Next, we use fewer frequency coefficients for the element-wise multiplications, which speeds up the computation. In other words, the data x and filter y are compressed so both have fewer coefficients. When we multiply them element-wise, we use fewer multiplications and speed up the computation. This is indicated by the green box. Finally, we go back to the spatial domain

via inverse FFT and return output o, which is input x convolved with filter y. We compress to the point that enough spectrum is preserved in the frequency domain so that we retain the high accuracy of our models.

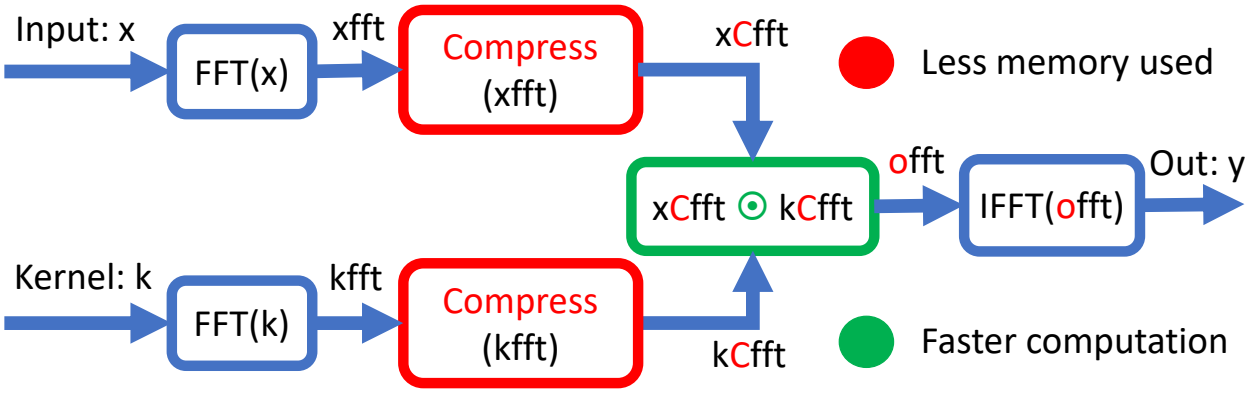

Figure 4.1: **Band-limited convolution.** *Compression applied to filters and activation input maps for FFT-based convolution with annotation of steps showing where we save memory and reduce computation.*

4.3.2 Definition

Let x be an input tensor (e.g., a signal) and y be another tensor representing the filter. We denote the convolution operation as $x * y$. Both x and y can be thought of as discrete functions (mapping tensor index positions $\mathbf{n}$ to values $x[\mathbf{n}]$). Accordingly, they have a corresponding Fourier representation, which re-indexes each tensor in the spectral (or frequency) domain:

$$F_x[\omega] = F(x[\mathbf{n}]) \qquad F_y[\omega] = F(y[\mathbf{n}])$$

This mapping is invertible $x = F^{-1}(F(x))$. Convolutions in the spectral domain correspond to element-wise multiplications:

$$x * y = F^{-1}(F_x[\omega] \cdot F_y[\omega])$$

The intermediate quantity $S[\omega] = F_x[\omega] \cdot F_y[\omega]$ is called the *spectrum* of the convolution. We start with the modeling assumption that for a substantial portion of the high-frequency domain, $|S[\omega]|$

is close to 0. This assumption is substantiated by the recent work by Rahman et al. studying the inductive biases of CNNs [127], with experimental results suggesting that CNNs are biased towards learning low-frequency filters (i.e., smooth functions). We take this a step further and consider the joint spectra of both the filter and the signal to understand the memory and computation implications of this insight.

4.3.3 Compression

Let $M_c[\omega]$ be a discrete indicator function defined as follows:

$$M_c[\omega] = \begin{cases} 1, \omega \leq c \\ 0, \omega > c \end{cases}$$

$M_c[\omega]$ is a mask that limits the $S[\omega]$ to a certain *band* of frequencies. The *band-limited* spectrum is defined as, $S[\omega] \cdot M_c[\omega]$, and the band-limited convolution operation is defined as:

$$x *_c y = F^{-1}\{(F_x[\omega] \cdot M_c[\omega]) \cdot (F_y[\omega] \cdot M_c[\omega])\} \quad (1)$$

$$= F^{-1}(S[\omega] \cdot M_c[\omega]) \quad (2)$$

The operation $*_c$ is compatible with automatic differentiation as implemented in popular deep learning frameworks such as PyTorch and TensorFlow. The mask $M_c[\omega]$ is applied to both the signal $F_x[\omega]$ and filter $F_y[\omega]$ (in equation 1) to indicate the compression of both arguments and fewer number of element-wise multiplications in the frequency domain.

4.3.4 FFT Implementation

We implement band-limited convolution with the Fast Fourier Transform. FFT-based convolution is supported by many Deep Learning libraries (e.g., cuDNN). It is most effective for larger filter-sizes where it significantly reduces the amount of floating-point operations. While convolutions

can be implemented by many algorithms, including matrix multiplication and the Winograd minimal filtering algorithm, the use of an FFT is important (as explained above in section 4.2 on Small vs Large Filters). The compression mask $M_c[\omega]$ is sparse in the Fourier domain. $F^{-1}(M_c)$ is, however, dense in the spatial or temporal domains. If the algorithm does not operate in the Fourier domain, it cannot take advantage of the sparsity in the frequency domain.

The Expense of FFT-based Convolution

It is worth noting that pre-processing steps are crucial for the correct implementation of convolution via FFT. The filter is usually much smaller (than the input) and has to be padded with zeros to the final length of the input signal. The input signal has to be padded on one end with as many zeros as the size of the filter to prevent the effects of wrapped-around filter data (for example, the last values of convolution should be calculated only from the final overlap of the filter with the input signal and should not be polluted with values from the beginning of the input signal).

Due to this padding and expansion, FFT-based convolution implementations are often expensive in terms of memory usage. Such an approach is typically avoided on GPU architecture, but recent results suggest improvements on CPU architecture [186]. The compression mask $M_c[\omega]$ reduces the size of the expanded spectra; we need not compute the product for those values that are masked out. Therefore, a band-limiting approach has the potential to make FFT-based convolution more practical for smaller filter sizes.

Band-limiting technique

We present the transformations from a natural image to a band-limited FFT map in Figure 4.2.

The FFT domain cannot be arbitrarily manipulated as we must preserve *conjugate symmetry*. For a 1D signal, this is straight-forward. $F[-\omega] = F^*[\omega]$, where the sign of the imaginary part is opposite when $\omega < 0$. The compression is applied by discarding the high frequencies in the first half of the signal. We have to do the same to the filter, and then, the element-wise multiplication in the frequency domain is performed between the compressed signal (input map) and the compressed

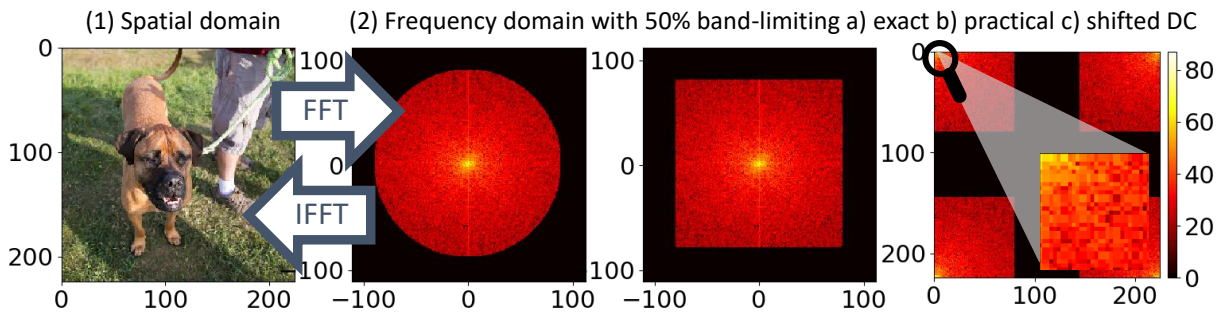

Figure 4.2: *Transformations from input image to compressed FFT map. (1) Natural image in the spatial domain. (2) FFT transformation to frequency domain and a) exact band-limiting to 50%, b) practical band-limiting to 50%, c) lowest frequencies shifted to corners. The heat maps of magnitudes of Fourier coefficients are plotted for a single channel (0-th) in a logarithmic scale (dB) with linear interpolation and the max value is colored with white while the min value is colored with black.*

filter. We use zero padding to align the sizes of the signal and filter. We execute the inverse FFT (IFFT) of the output of this multiplication to return to the original spatial or time domain.

In addition to the conjugate symmetry, certain values are constrained to be real. For example, the first coefficient is real (the average value) for the odd and even length signals and the middle element ($\left\lfloor \frac{N}{2} \right\rfloor + 1$) is also real for the even-length signals. We do not violate these constraints and keep the coefficients real, for instance, by replacing the middle value with zero during compression or padding the output with zeros.

The conjugate symmetry for a 2D signal $F[-\omega, -\theta] = F^*[\omega, \theta]$ is more complicated. If the real input map is of size $M \times N$, then its complex representation in the frequency domain is of size $M \times (\left\lfloor \frac{N}{2} \right\rfloor + 1)$. The real constraints for 2D inputs were explained in detail in Figure 4.3, similarly to [133]. For the most interesting and most common case of even height and width of the input, there are always four real coefficients in the spectral representation (depicted as **Orange** cells: top-left corner, the middle value in the top row, the middle value in the most-left column and the value in the center). The DC component is located in the top-left corner. The largest values are placed in the corners and decrease towards the center. This trend is our guideline in the design of the compression pattern, in which for the *left half* of the input, we discard coefficients from the center in *L-like* shapes towards the top-left and bottom-left corners.

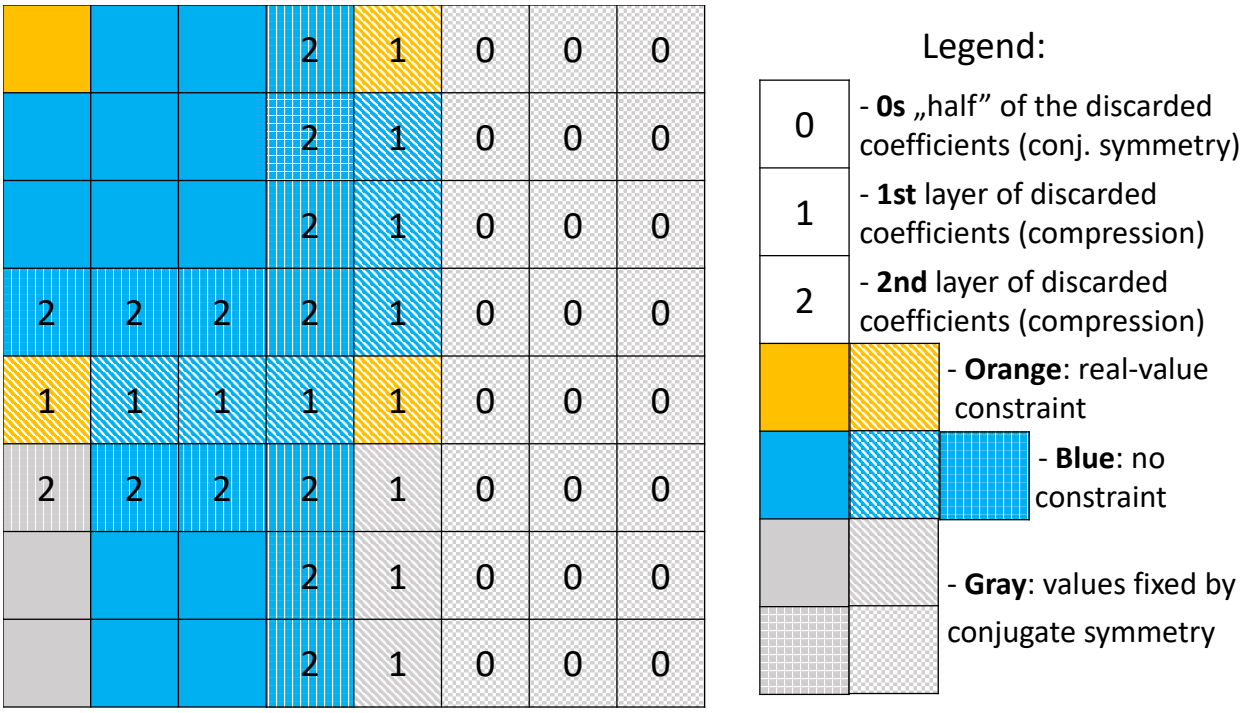

Figure 4.3: *An example of a square input map with marked conjugate symmetry (**Gray** cells). Almost half of the input cells marked with **0**s (zeros) are discarded first due to the conjugate symmetry. The remaining map is compressed layer by layer (we present how the first two layers: **1** and **2** are selected). **Blue** and **Orange** cells represent a minimal number of coefficients that have to be preserved in the frequency domain to fully reconstruct the initial spatial input. Additionally, the **Orange** cells represent real-valued coefficients.*

Map Reuse

The FFT computations of the tensors: input map, filter, and the gradient of the output as well as the IFFT of the final output tensors are one of the most expensive operations in the FFT-based convolution. We avoid re-computation of the FFT for the input map and the filter by saving their frequency representations at the end of the forward pass and reusing them in the corresponding backward pass. The memory footprint for the input map in the spatial and frequency domains is almost the same. We retain only half of the frequency coefficients but they are represented as complex numbers. Further on, we assume square input maps and filters (the most common case). For an $N \times N$ real input map, the initial *complex-size* is $N \times (\left\lfloor \frac{N}{2} \right\rfloor + 1)$. The filter (also called the kernel) is of size $K \times K$. The FFT-ed input map has to be convolved with the gradient of size $G \times G$ in the backward pass and usually $G > K$. Thus, to reuse the FFT-ed input map and avoid

wrapped-around values, the required padding is of size: $P = \max(K-1, G-1)$. This gives us the final full spatial size of tensors used in FFT operations $(N+P) \times (N+P)$ and the corresponding full *complex-size* $(N+P) \times (\left\lfloor \frac{(N+P)}{2} \right\rfloor + 1)$ that is finally compressed.

To give more details on the implementation part of the *map reuse*, we provide a code-level overview of how the map is handled. We divide the input map M (with half of the map already removed due to the conjugate symmetry) into two parts: upper D1 and lower D2. We crop out the top-left (S1) corner from D1 and bottom-left (S2) corner from D2. The two compressed representations S1 and S2 can be maintained separately (small saving in computation time) or concatenated (more convenient) for the backward pass. In the backward pass, we pad the two corners S1 and S2 to their initial sizes D1 and D2, respectively. Finally, we concatenate D1 and D2 to get the FFT map M', where the high-frequency coefficients are replaced with zeros.

If the memory usage should be decreased as much as possible and the filter is small, we can trade the lower memory usage for the longer computation time and save the filter in the spatial domain at the end of the forward pass, followed by the FFT re-computation of the filter in the backward pass. The full frequency representation of the input map (after padding) is bigger than its spatial representation, thus the profitability of re-computing the input to save the GPU memory depends on the applied compression rate.

We also leverage a fast shift of the DC coefficients (the lowest frequency coefficients) either to the center or to the top-left corner of the frequency map. The code for the element-wise solution uses two for loops and copies each element separately. For the full FFT map, we divide it into quadrants (I - top-right, II - top-left, III - bottom-left, IV - bottom-right). Then, we permute the quadrants in the following way: I $\rightarrow$ III, II $\rightarrow$ IV, III $\rightarrow$ I, IV $\rightarrow$ II.

4.3.5 Implementation in PyTorch and CUDA

Our compression in the frequency domain is implemented as a module in PyTorch that can be plugged into any architecture as a convolutional layer. The code is written in Python with extensions in C++ and CUDA for the main bottleneck of the algorithm. The most expensive compu-

tationally and memory-wise component is the Hadamard product in the frequency domain. The complexity analysis of the FFT-based convolution is described in [112] (section 2.3, page 3). The complex multiplications for the convolution in the frequency domain require $3Sf'fn^2$ real multiplications and $5Sf'fn^2$ real additions, where S is the mini-batch size, f' is the number of filter banks (i.e., kernels or output channels), f is the number of input channels, and n is the height and width of the inputs. In comparison, the cost of the FFT of the input map is $Sfn^2 2logn$, and usually, $f' >> 2logn$. We implemented in CUDA the fast algorithm to multiply complex numbers with 3 real multiplications instead of 4 as described in [97].

Our approach to convolution in the frequency domain aims at saving memory and utilizing as many GPU threads as possible. In our CUDA code, we fuse the element-wise complex multiplication (which in a standalone version is an injective one-to-one map operator) with the summation along an input channel (a reduction operator) in a thread execution path to limit the memory size from $2Sff'n^2$, where 2 represents the real and imaginary parts, to the size of the output $2Sf'n^2$, and avoid any additional synchronization by focusing on the computation of a single output cell: (x,y) coordinates in the output map. We also implemented another variant of convolution in the frequency domain by using tensor transpositions and replacing the complex tensor multiplication (CGEMM) with three real tensor multiplications (SGEMM).

We use $min(max\ threads\ in\ block, n^2)$ threads per block and the total number of GPU blocks is Sf'. Each block of threads is used to compute a single output plane. Intuitively, each thread in a block of threads incrementally executes a complex multiplication and adds the result to aggregate for all f input channels to obtain a single output cell (x,y).

Additional optimizations can be applied, for example, maintaining the filters only in the frequency domain or tiling. The tiling technique divides input channels into tiles. For filters of size k, the neighboring tiles overlap by $k-1$ elements. In Figure 4.4, the input of size 6×6 is divided into 4 tiles, each of size 4×4 and the filter is of size 3×3. The neighboring tiles overlap by 2 elements. The output is of size 4×4. This is a standard technique used in the Winograd convolution and is

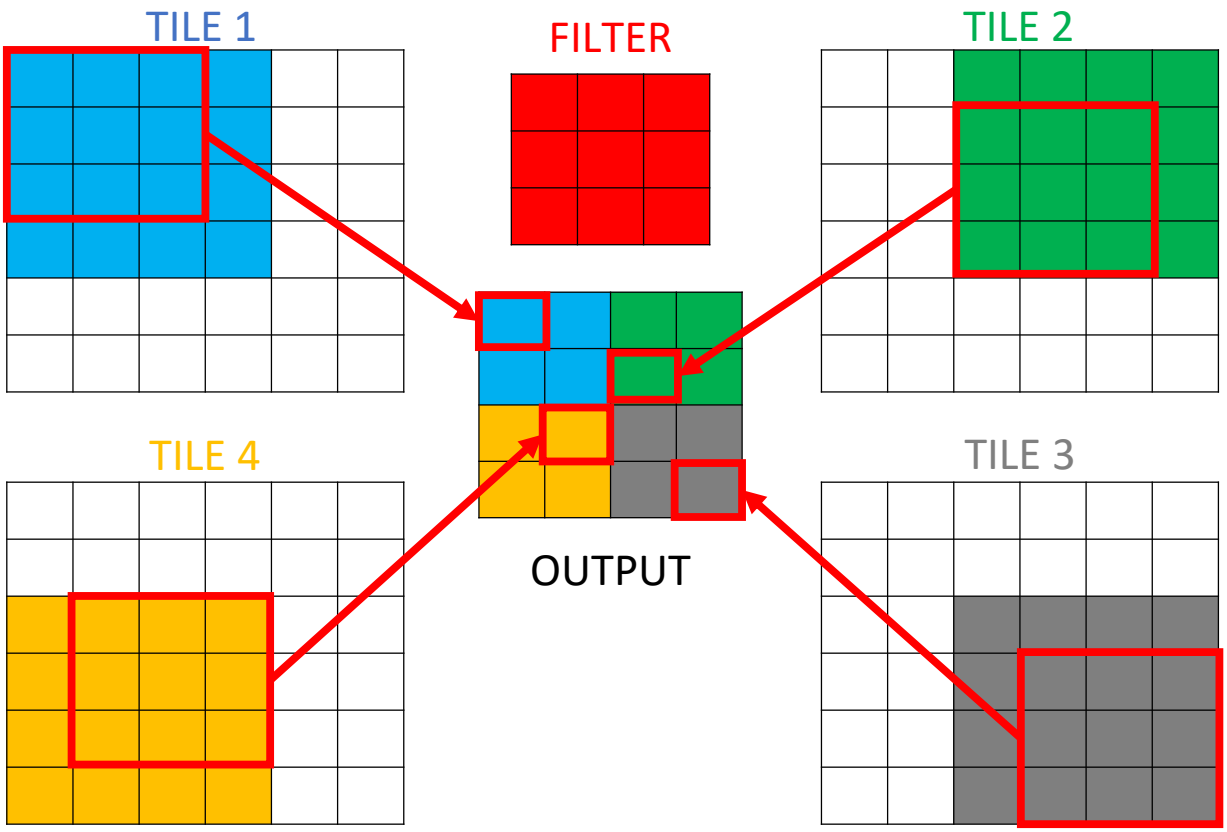

Figure 4.4: An example of the tiling technique

.

also applied to the FFT-based convolution [2]. It is used to find sizes for sub-inputs that can provide the best performance.

2. https://developer.nvidia.com/cudnn

Table 4.1: Test accuracies for ResNet-18 on CIFAR-10 and DenseNet-121 on CIFAR-100 with the same compression rate across all layers. We vary compression from 0% (full-spectra model) to 50% (band-limited model).

CIFAR	0%	10%	20%	30%	40%	50%
10	93.69	93.42	93.24	92.89	92.61	92.32
100	75.30	75.28	74.25	73.66	72.26	71.18

4.4 Experimental Setup

We run our experiments on single GPU deployments with NVidia P-100 GPUs and 16 GBs of total memory. We use the ResNet-18 architecture [72] trained on the CIFAR-10 dataset [93] and the DenseNet-121 architecture [80] trained on the CIFAR-100 dataset [72]. The objective of our experiments is to demonstrate the robustness and explore the properties of band-limited training and inference for CNNs. [3]

We also use data from the UCR archive [29], with a representative dataset being 50 words time-series dataset with 270 values per data point, 50 classes, 450 train data points, 455 test data points, 2 MB in size. One of the best performing CNN models for the data is a 3 layer Fully Convolutional Neural Network (FCN) with filter sizes: 8, 5, 3. The number of filter banks is: 128, 256, 128. [4]

4.5 Impact on Accuracy

4.5.1 Effects of Band-limited Training on Inference

First, we study how band-limiting training effects the final test accuracy of two popular deep neural networks, namely, ResNet-18 and DenseNet-121, on CIFAR-10 and CIFAR-100 datasets, respectively. Specifically, we vary the compression rate between 0% and 50% for each convolutional layer (i.e., the percentage of coefficients discarded) and we train the two models for 350 epochs. Then, we measure the final test accuracy using the same compression rate as the one used during

3. We publish our source code at: `https://github.com/adam-dziedzic/bandlimited-cnns`

4. Deep Learning for Time Series Classification on GitHub: `http://bit.ly/2FbdQNV`

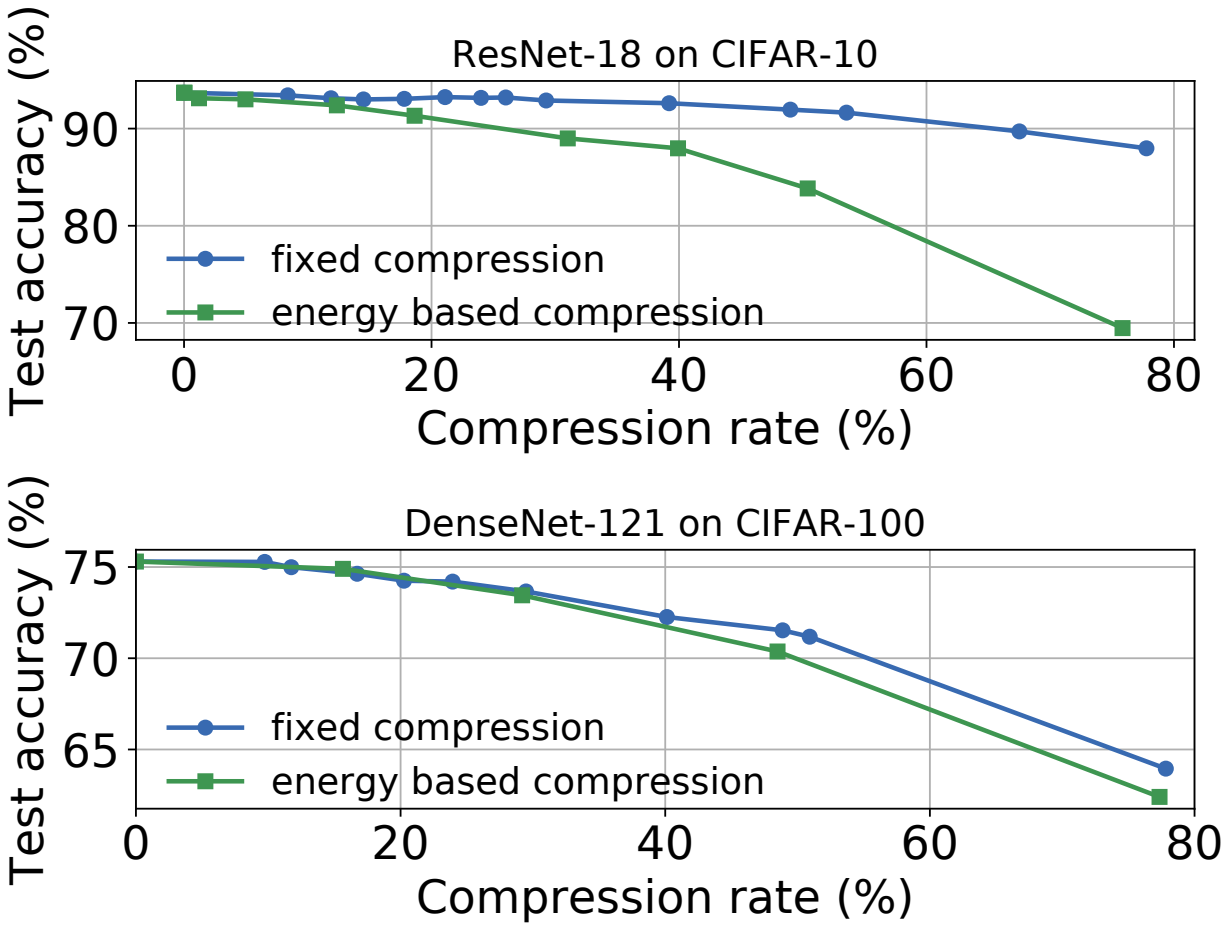

Figure 4.5: *Test accuracy as a function of the compression rate for ResNet-18 on CIFAR-10 and DenseNet-121 on CIFAR-100. The fixed compression scheme that uses the same compression rate for each layer gives the highest test accuracy.*

training. Our results in Table 4.1 show a smooth degradation in accuracy despite the aggressive compression applied during band-limiting training.

To better understand the effects of band-limiting training, in Figure 4.5, we explore two different compression schemes: (1) fixed compression, which discards the same percentage of spectral coefficients in each layer and (2) energy compression, which discards coefficients in an adaptive manner based on the specified energy retention in the frequency spectrum. By Parseval's theorem, the energy of an input tensor x is preserved in the Fourier domain and defined for normalized FFT transformation as:

$$E(x) = \sum_{n=0}^{N-1} |x[n]|^2 = \sum_{\omega=0}^{2\pi} |F_x[\omega]| \tag{4.1}$$

For example, for two convolutional layers of the same size, a fixed compression of 50% discards 50% of coefficients in each layer. On the other hand, the energy approach may find that 90% of the energy is preserved in the 40% of the low-frequency coefficients in the first convolutional layer while for the second convolutional layer, 90% of energy is preserved in 60% of the low-frequency coefficients.

For more than 50% of compression rate for both techniques (the precise compression ratios are 53.53% and 53.74%, respectively), the fixed compression method achieves the max test accuracy of 92.32% (only about 1% worse than the best test accuracy for the full model) whereas the preserved energy method results in significant losses (e.g., ResNet-18 reaches 83.37% on CIFAR-10). Our findings suggest that altering the compression rate during model training may affect the dynamics of SGD. The worse accuracy of the models trained with any form of dynamic compression is a result of the higher noise incurred by frequent changes to the number of coefficients that are considered during training.

The adaptive energy-based scheme changes the compression ratio frequently and drastically. On the other hand, the static method is rather brittle and non-adaptive. We run another experiment using the ResNet-18 architecture on the CIFAR-10 dataset, where we calculate the compression ratio in the first epoch using the energy metric and then use it statically for the remaining epochs. We present the result separately in Figure 4.6. The test accuracy of the *mixed* method falls between that of static and dynamic compression. Thus, we select the static method as our main compression method.

The test accuracy for energy-based compression follows the coefficient one for DenseNet-121 while they markedly diverge for ResNet-18. One of the reasons for this effect is the use of different operations in the residual connections. ResNet combines outputs from L and $L+1$ layers

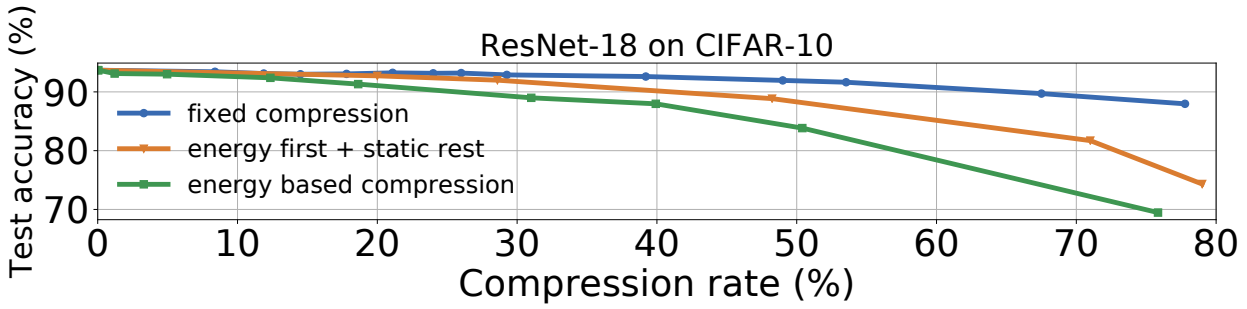

Figure 4.6: *Test accuracy as a function of the compression rate for ResNet-18 on CIFAR-10. The middle orange line represents the mixed method, which is a combination of static and energy-based methods.*

by summation. In the adaptive scheme, this means adding maps produced from different spectral bands. In contrast, DenseNet concatenates the layers.

To dive deeper into the effects on SGD, we perform an experiment where we keep the same energy preserved in each layer and for every epoch. Every epoch we record what is the physical compression (number of discarded coefficients) for each layer. The dynamic compression based on the energy preserved shows that at the beginning of the training the network is focused on the low-frequency coefficients and as the training unfolds, more and more coefficients are taken into account, which is shown in Figure 4.7. The compression based on preserved energy does not steadily converge to a certain compression rate but can decrease significantly and abruptly even at the end of the training process (especially, for the initial layers). The main observation is that the distribution of the energy in the frequency spectrum gets naturally broader as the training progresses.

4.5.2 Training Compression vs Inference Compression

Having shown a smooth degradation in performance for various compression rates, we now study the effect of changing the compression rates during training and inference phases. This scenario is useful during dynamic resource allocation in model serving systems.

Figure 4.8 illustrates the test accuracy of ResNet-18 and DenseNet-121 models trained with specific coefficient compression rates (e.g., 0%, 50%, and 85%) while the compression rates are

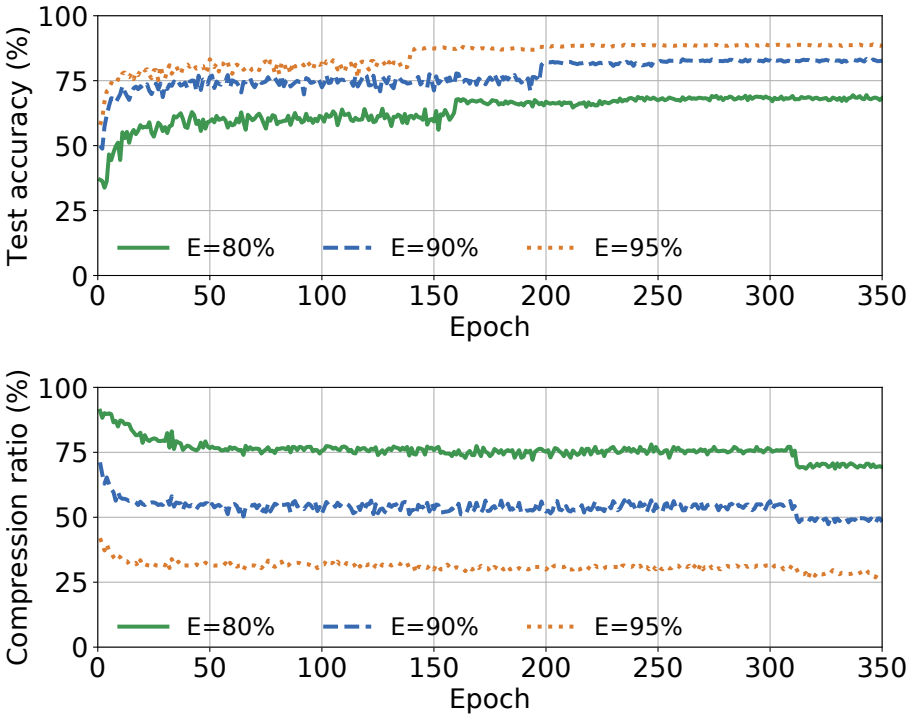

Figure 4.7: *Compression changes during training with constant energy preserved: the longer we train the models with the same energy preserved, the smaller compression is applied. The compression rate (%) is calculated based on the size of the intermediate results for the FFT based convolution. E - is the amount of energy (in %) preserved in the spectral representation: 80, 90, and 95. We trained ResNet-18 models on CIFAR-10 for 350 epochs. The best test accuracy levels achieved by the models are 69.47%, 83.37% and 88.99%, respectively.*

changed systematically during inference. We observe that the models achieve their best test accuracy when the same level of compression is used during training and inference. Besides, we performed the same experiment across 25 randomly chosen time-series datasets from the UCR archive [29] using a 3-layer Fully Convolutional Network (FCN), which has achieved state-of-the-art results in prior work [172]. We used the Friedman statistical test [60] followed by the post-hoc Nemenyi test [120] to assess the performance of multiple compression rates during inference over multiple datasets (see supplementary material for details). Our results suggest that the best test accuracy is achieved when the same compression rate is used during training and inference and, importantly, the difference in accuracy is statistically significantly better in comparison to the test accuracy achieved with different compression rates during inference.

Overall, our experiments show that band-limited CNNs learn the constrained spectrum and

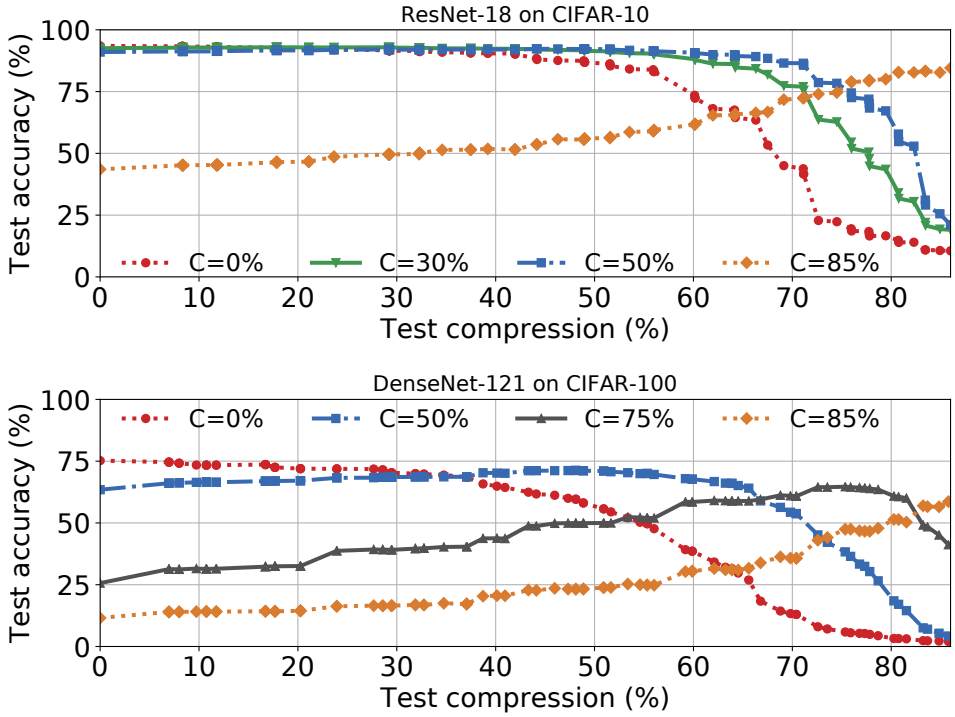

Figure 4.8: *The highest accuracy during testing is for the same compression level as used for training and the test accuracy degrades smoothly for higher or lower levels of compression. First, we train models with different compression levels (e.g. DenseNet-121 on CIFAR-100 with compression rates: 0%, 50%, 75%, and 85%). Second, we test each model with compression levels ranging from 0% to 85%.*

perform the best for similar constraining during inference. In addition, the smooth degradation in performance is a valuable property of band-limited training as it permits outer optimizations to tune the compression rate parameter without unexpected instabilities or performance cliffs.

4.6 Resource Usage

4.6.1 Comparison Against Reduced Precision Method

Until now, we have demonstrated the performance of band-limited CNNs in comparison to the full spectra counterparts. It remains to show how the compression mechanism compares against a strong baseline. Specifically, we evaluate band-limited CNNs against CNNs using reduced pre-

Table 4.2: Resource utilization (RES. in %) for a given precision and compression rate (SETUP). MEM. ALLOC. - the memory size allocated on the GPU device, MEM. UTIL. - percent of time when memory was read or written, GPU UTIL. - percent of time when one or more kernels was executing on the GPU. C - denotes the compression rate (%) applied, e.g., FP32-C=50% is model trained with 32 bit precision for floating point numbers and 50% compression applied.

RES(%) \ SETUP	FP32-C=0%	FP16-C=0%	FP32-C=50%	FP32-C=85%
AVG. MEM. ALLOC.	6.69	4.79	6.45	4.92
MAX. MEM. ALLOC.	16.36	11.69	14.98	10.75
AVG. MEM. UTIL.	9.97	5.46	5.54	3.50
MAX. MEM. UTIL.	41	22	24	20
AVG. GPU UTIL.	24.38	22.53	21.70	16.87
MAX. GPU UTIL.	89	81	74	70
TEST ACC.	93.69	91.53	92.32	85.4

cision arithmetic (RPA). RPA-based methods require specialized libraries [5] and are notoriously unstable. They require significant architectural modifications for precision levels under 16-bits–if not new training chipsets [171, 136]. [6] From a resource perspective, band-limited CNNs are competitive with RPA-based CNNs–without requiring specialized libraries. To record the memory allocation, we run ResNet-18 on CIFAR-10 with batch size 32 and we query the VBIOS (via nvidia-smi [7]) every millisecond in the window of 7 seconds). Table 4.2 shows a set of basic statistics for resource utilization for RPA-based (fp16) and band-limited models. The more compression is applied or the lower the precision set (fp16), the lower the utilization of resources. We also show that it is possible to combine the two methods.

5. Mixed-precision training for PyTorch: `https://devblogs.nvidia.com/apex-pytorch-easy-mixed-precision-training/`

6. PyTorch delegates the selection of the convolution algorithm to the NVIDIA's cuDNN library: `https://bit.ly/3eSATus`

7. nvidia-smi: `https://bit.ly/2SZryG8`

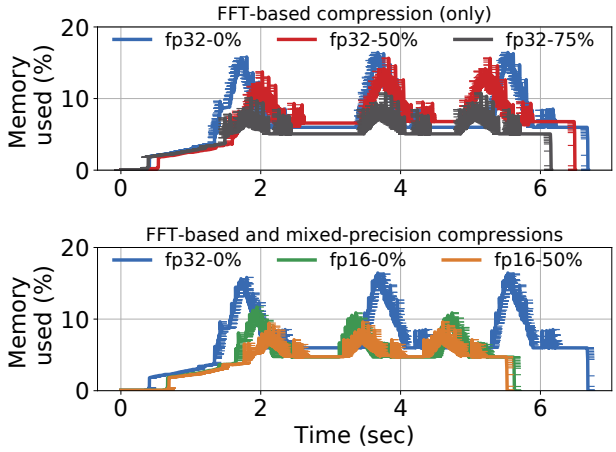

Figure 4.9: *Memory used (%) for the first 3 iterations (train and test) with mixed-precision and FFT-based compression techniques. Mixed precision allows only a certain level of compression whereas with the FFT based compression we can adjust the required compression and accuracy. Finally, the two methods are combined and denoted as fp16-50%, where fp16 indicates mixed-precision arithmetic, and 50% is the FFT compression rate.*

4.6.2 Reduced Precision and Band-limited Training

In Figure 4.9 we plot the maximum allocation of the GPU memory during 3 first iterations. Each iteration consists of training (forward and backward passes) followed by testing (a single forward pass). We use CIFAR-10 data on ResNet-18 architecture. We show the memory profiles of RPA (Reduced Precision Arithmetic), band-limited training, and applying both. A detailed convergence graph is shown in Figure 4.10.

4.6.3 Control of the GPU and Memory Usage

In Figure 4.11, we compare two metrics: maximum GPU memory allocated (during training) and time per epoch, as we increase the compression rate. The points in the graph with 100% performance for 0% of compression rate correspond to the values of the metrics for the full spectra (uncompressed) model. We normalize the values for the compressed models as:

$$\frac{\text{metric value for a compressed model}}{\text{metric value for the full spectra model}} \cdot 100\% \tag{4.2}$$

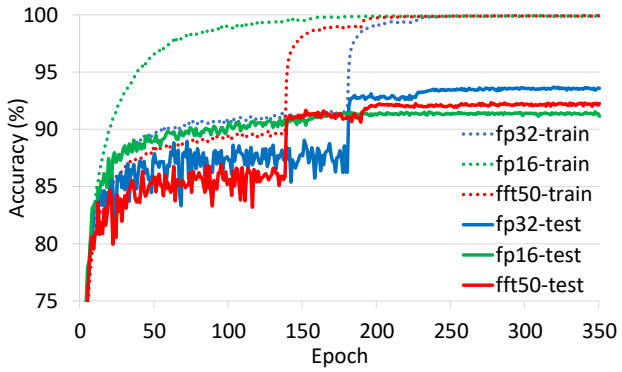

Figure 4.10: *Train and test accuracy during training for CIFAR-10 dataset trained on ResNet-18 architecture using convolution from PyTorch (fp32), mixed-precision (fp16) and FFT-based convolutions with 50% of compression for intermediate results and filters (fft50). The highest test accuracy observed are: 93.69 (fp32), 91.53 (fp16), 92.32 (fft50).*

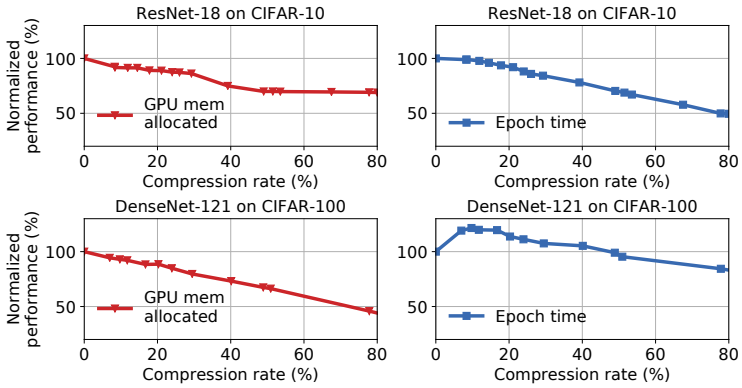

Figure 4.11: *Normalized performance (%) between models trained with different FFT-compression rates.*

For ResNet-18 architecture, a small drop in accuracy can save a significant amount of computation time. However, for more convolutional layers in DenseNet-121, the overhead of compression (for small compression rate) is no longer amortized by fewer multiplications (between compressed tensors). The overhead is due to the modifications of tensors to compress them in the frequency domain and their decompression (restoration to the initial size) before going back to the spatial domain (to preserve the same frequencies for the inverse FFT). FFT-ed tensors in PyTorch place the lowest frequency coefficients in the corners. For compression, we extract parts of a tensor from its top-left and bottom-left corners. For the decompression part, we pad the discarded parts with zeros and concatenate the top and bottom slices.

DenseNet-121 shows a significant drop in GPU memory usage with a relatively low decrease in accuracy. On the other hand, ResNet-18 is a smaller network and after about 50% of the compression rate, other than convolutional layers start to dominate the memory usage. The convolution operation remains the major computation bottleneck and we decrease it with higher compression.

4.7 Band-limited Exploration

We further explore the band-limiting technique by applying it to other domains (e.g., time-series data), comparing band-limited convolutions with standard convolution operations, visualizing its behavior, and checking how robust it is to noisy inputs.

4.7.1 Generality of Band-limiting Technique

To show the applicability of band-limited training to different domains, we apply our technique using the FCN architecture discussed previously on the time-series datasets from the UCR archive. Figure 4.12 compares the test accuracy between full-spectra (no compression) and band-limited (with 50% compression) models with FFT-based 1D convolutions. As with the results for 2D convolutions, we find that not only is accuracy preserved but there are very significant savings in terms of GPU memory usage (Table 4.3).

Table 4.3: Resource utilization and accuracy for the FCN network on a representative time-series dataset (see supplement for details).

ENERGY PRESERVED (%)	AVG. GPU MEM USAGE (MB)	MAX. TEST ACCURACY (%)
100	118	64.40
90	**25**	**63.52**
50	21	59.34
10	17	40.00

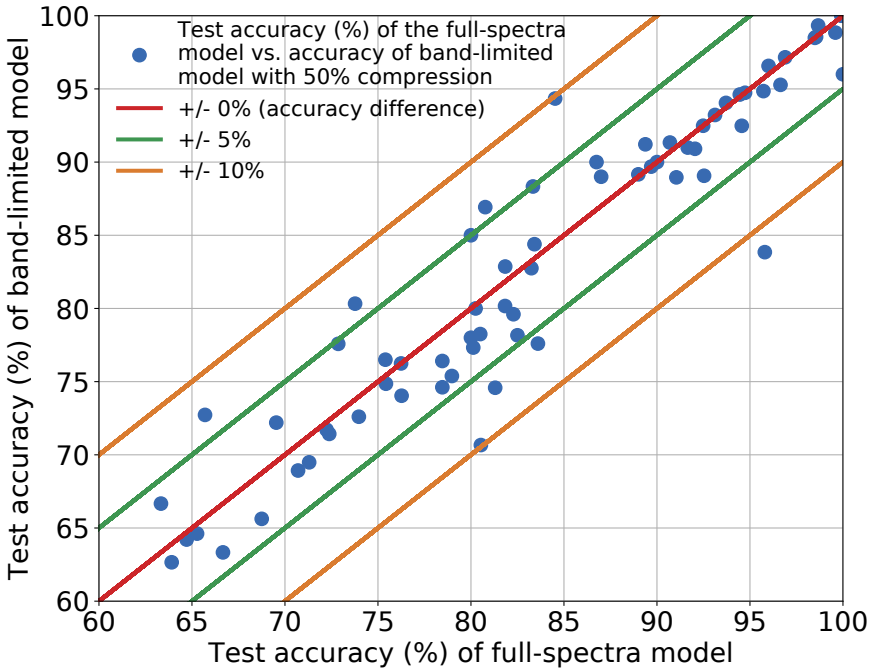

Figure 4.12: *Comparing test accuracy (%) on a 3-layer FCN architecture between full-spectra models (100% energy preserved, no compression) and a band-limited models with 50% compression rate for time-series datasets from the UCR archive. The red line indicates no difference in accuracy between the models while green and orange margin lines show +/- 5% and +/- 10% differences.*

4.7.2 Per-Operation Error

Despite the accurate predictive performance of the trained models, we find that the approximation error in the individual convolution operations can be relatively high. We execute our convolution operations with compression for different compression rates on a single CIFAR-10 image. The compression is measured relative to the execution of our bandlimited convolution without any compression (100% of the energy of the input image is preserved). The results show that the relative error is high (about 22.07%) for a single index discarded (the smallest possible compression of about 6%) in Figure 4.13.

This result is a mathematically curious point. Why are models trained with band-limited convolutions so accurate when the individual convolution operations are not? All of our experimental results suggest that band-limiting is not simply an operator-level approximation, but rather, the *network learns to use the low-frequency coefficients to make predictions and changes its intermediate*

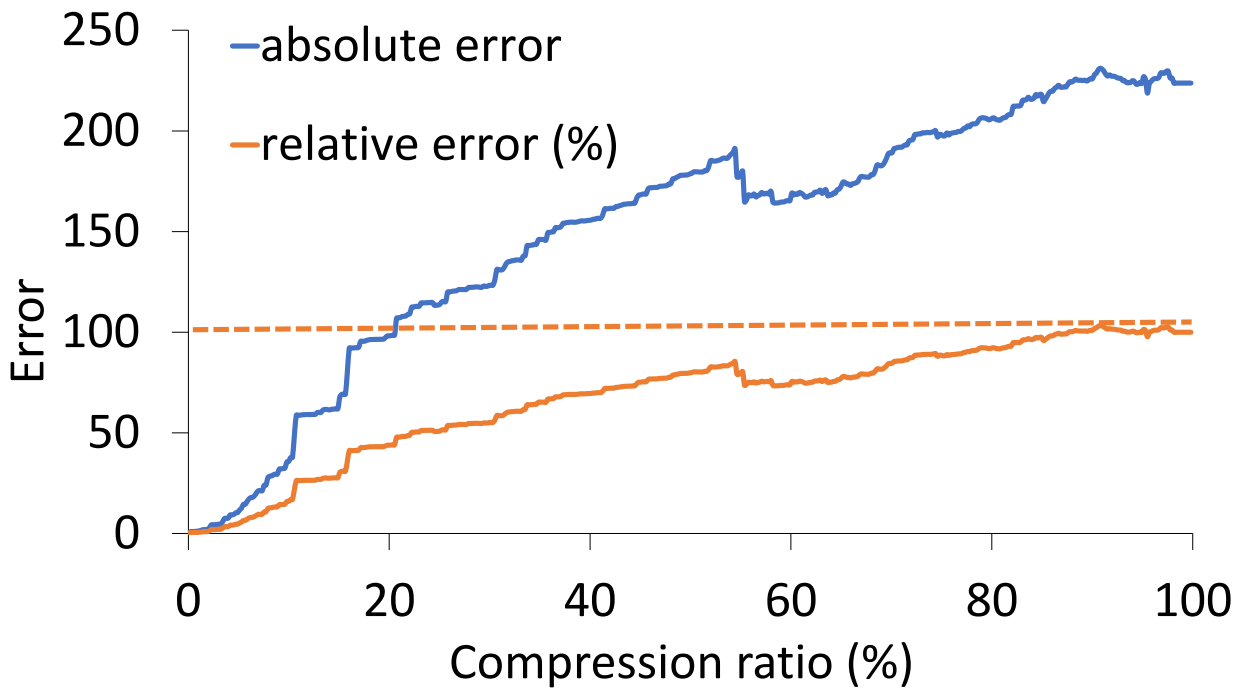

Figure 4.13: *A comparison of the relative (in %) and absolute errors between 2D convolution from PyTorch (which is our gold standard with high numeric accuracy) and a fine-grained top compression method for a CIFAR-10 image and a 5x5 filter (with 3 channels).*

representations.

4.7.3 Visualisations

FFT-based compression in 1D

We present the visualization of our FFT-based compression method in Figure 4.14. We use a time series (signal) from the UCR archive, the 50 words dataset in this case, and plot it in the top-left quadrant. Its frequency representation (as power spectrum) after normalized FFT transformation is shown in the top-right quadrant. The signal is compressed by 95% (we zero out the *middle* Fourier coefficients) and presented in the bottom-right quadrant. We compare the initial signal and its compressed version in the bottom-left quadrant. The magnitudes of Fourier coefficients are presented in the logarithmic (dB) scale.

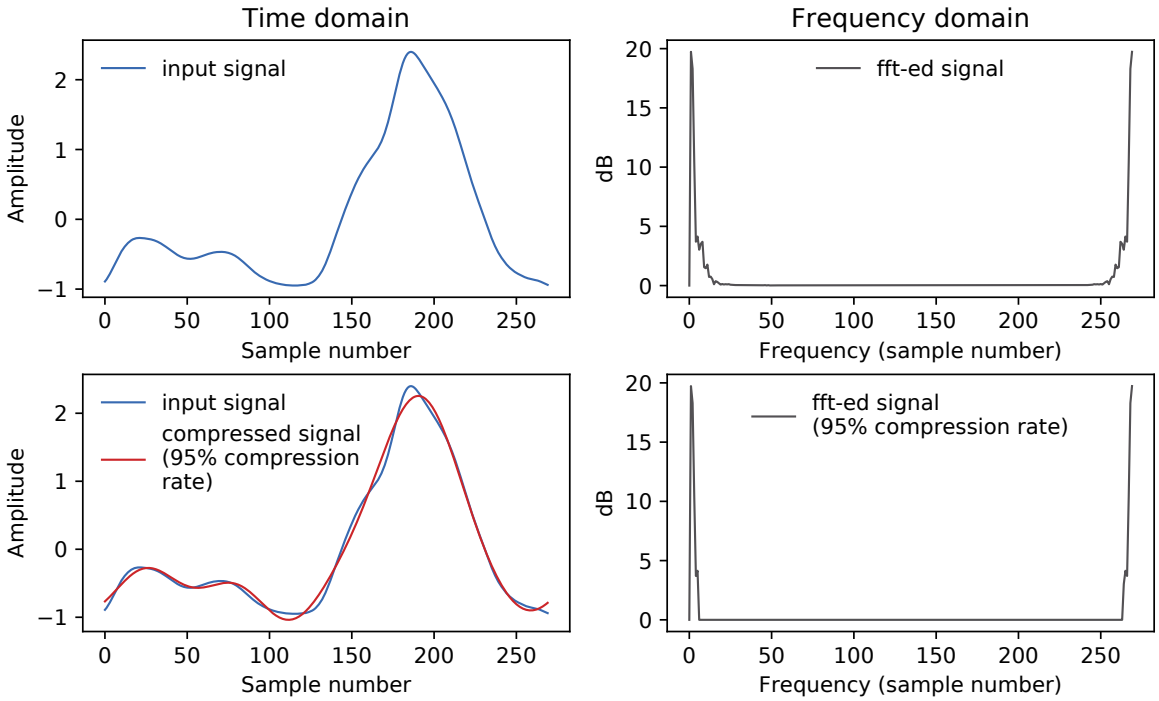

Figure 4.14: *Comparison of original and FFT compressed 1D signal in time and frequency domains.*

2D FFT Map

The heat map of an FFT representation that contains the absolute values of the frequency coefficients is shown in Figure 4.15. We use the linear interpolation and the max value is colored with white, the min value is colored with black. The FFT-ed input is a single (0-th) channel of a randomly selected image from the CIFAR-10 dataset. The DC component (the lowest frequency coefficient) is placed in the top-left corner.

4.7.4 Robustness to Noise

Finally, we evaluate the robustness of band-limited CNNs. Specifically, models trained with more compression discard part of the noise by removing the high-frequency Fourier coefficients. In Figure 4.16, we show the test accuracy for input images perturbed with different levels of Gaussian noise, which is controlled systematically by the sigma parameter, fed into models trained with

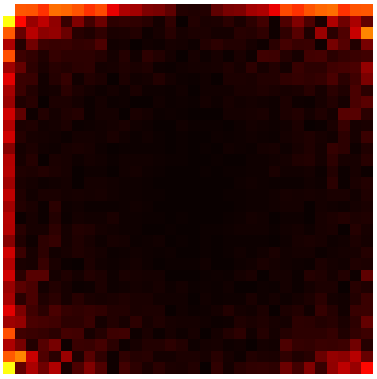

Figure 4.15: *A 2D heat map of absolute values (magnitudes) of FFT coefficients.*

different compression levels (i.e., 0%, 50%, and 85%) and methods (i.e., band-limited vs RPA-based). Our results demonstrate that models trained with higher compression are more robust to the inserted noise. Interestingly, band-limited CNNs also outperform the RPA-based method and under-fitted models (e.g., via early stopping), which do not exhibit the robustness to noise.

We additionally run experiments using Foolbox [129]. Our method is robust to decision-based and score-based (black-box) attacks (e.g., the band-limited model is better than the full-spectra model in 70% of cases for the additive uniform noise, and in about 99% cases for the pixel perturbations attacks) but not to the gradient-based (white-box and adaptive) attacks, e.g., Carlini-Wagner [21] (band-limited convolutions return proper gradients). Fourier properties suggest further investigation of invariances under adversarial rotations and translations [53].

4.8 Conclusions

Our main finding is that compressing a model in the frequency domain, called band-limiting, gracefully degrades the predictive accuracy as a function of the compression rate. We also develop principled schemes to reduce the resource consumption of neural network training. Neural networks are heavily over-parametrized and modern compression techniques exploit this redundancy. Reducing this footprint during training is more challenging than during inference due to the sensitivity of gradient-based optimization to noise.

While implementing an efficient band-limited convolutional layer is not trivial, one has to ex-

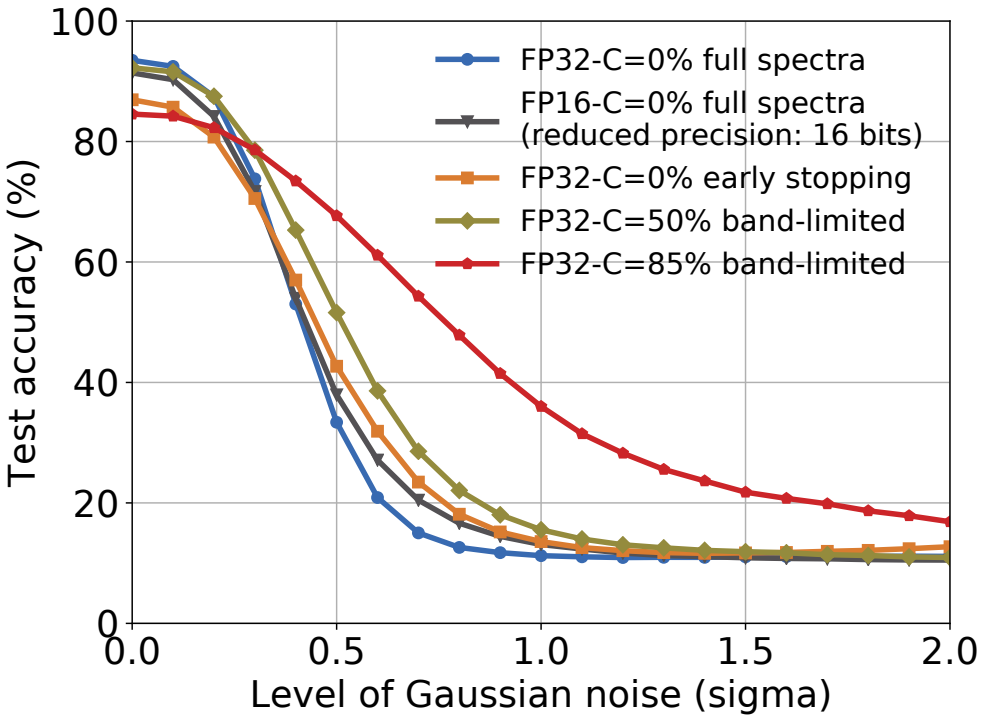

Figure 4.16: *Input test images are perturbed with Gaussian noise, where the sigma parameter is changed from 0 to 2. The more band-limited model, the more robust it is to the introduced noise. We use ResNet-18 models trained on CIFAR-10.*

ploit conjugate symmetry, cache locality, and fast complex arithmetic, no additional modifications to the architecture or training procedure is needed. Band-limited training provides a continuous knob to trade-off resource utilization vs predictive performance. Beyond the computational performance, frequency restriction serves as a strong prior. If we know that our data has biased frequency spectra or that the functions learned by the model should be smooth, band-limited training provides an efficient way to enforce those constraints.

CHAPTER 5

ROBUST MODELS FOR ADVERSARIAL AND OOD EXAMPLES

Recent work has extensively shown that randomized perturbations of neural networks can improve robustness to adversarial attacks. The literature is, however, lacking a detailed compare-and-contrast of the latest proposals to understand what classes of perturbations work, when they work, and why they work. We contribute a detailed evaluation that elucidates these questions and benchmarks perturbation based defenses consistently. In particular, we show five main results: (1) all input perturbation defenses, whether random or deterministic, are equivalent in their efficacy, (2) attacks transfer between perturbation defenses so the attackers need not know the specific type of defense – only that it involves perturbations, (3) a tuned sequence of noise layers across a network provides the best empirical robustness, (4) perturbation based defenses offer almost no robustness to adaptive attacks unless these perturbations are observed during training, and (5) adversarial examples in a close neighborhood of original inputs show an elevated sensitivity to perturbations in first and second-order analyses.

We also illustrate applications to out-of-distribution (OOD) robustness of models in terms of their generalization and detection of anomalous inputs in the NLP (Natural Language Processing) domain.

5.1 Introduction

The attacks on Convolutional Neural Networks, such as Carlini & Wagner [22] or PGD [109], generate strategically placed modifications to induce a deliberate misprediction. Recently, there have been many defenses against such attacks that use perturbation during inference time and/or training to improve robustness [51, 135, 179]. The success of perturbation-based defenses suggests that adversarial examples are not robust themselves, where a small amount of noise can dominate the strategically placed perturbations rendering them ineffective. However, a detailed understanding of this phenomenon is lacking from the research literature including: (1) what types of perturbations

work and what is their underlying mechanism, (2) is this property specific to certain attacks and architectures, (3) whether such defenses are effective and how to make them effective, (4) what the magnitude of perturbations should be, and (5) where to place them in a network.

We first analyze the inference-time input perturbations, where the input to the network is manipulated prior to inference. The main trade-off is a selection of the perturbation strength to carefully mitigate prediction errors over true examples but maximize recovery of the adversarial examples. Interestingly enough, we find that this trade-off is consistent across very different families of perturbations, where the relationship between channel distortion (effective perturbation of the input) and robustness is very similar. One might think that a frequency-based or JPEG based defense should be more tuned to a natural image classification setting—but we do not find that this is the case. The particular distribution of noise added to the inference process is not as important as its magnitude. Figure 5.1 presents a sample image from the ImageNet dataset attacked by the Carlini-Wagner L_2 algorithm and recovery via different inference-time input perturbations: color-depth reduction (also known as feature-squeezing), FFT-based compression, Gaussian, and Uniform noise.

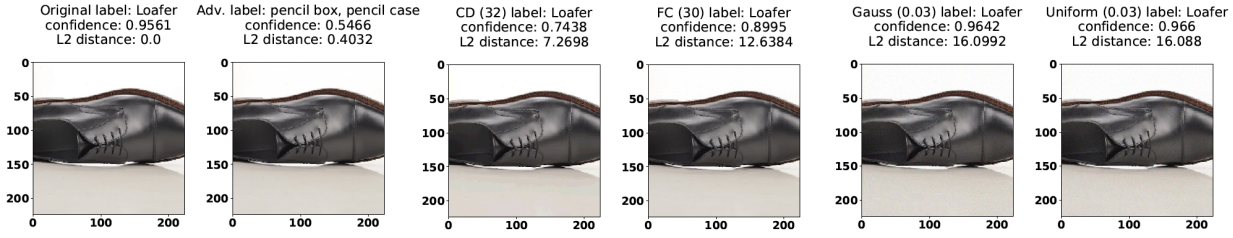

Figure 5.1: ***Recovery via input-based perturbations.*** We plot a sample image from the ImageNet dataset in its original state, after adversarial (white-box, non-adaptive C&W L_2) attack, and then after recovery via imprecise channels: CD (color depth reduction with 5 bits), FC (30% compression in the frequency domain), Gaussian, and Uniform noise ($\varepsilon = 0.03$).

Figure 5.2 shows the effect of the FC (a frequency-based imprecise channel with FFT compression) in both spatial and Fourier domains. The heat maps of magnitudes of Fourier coefficients are presented in a logarithmic scale (dB) with linear interpolation and the max value is colored with white while the min value is colored with black. The black part of the (bottom-right) image represents the removed high-frequency coefficients. The Fourier-ed representation is plotted for a

single (0-th) channel. The lowest frequency coefficients are placed in the corners of the FFT maps (with the DC component in the top-left corner).

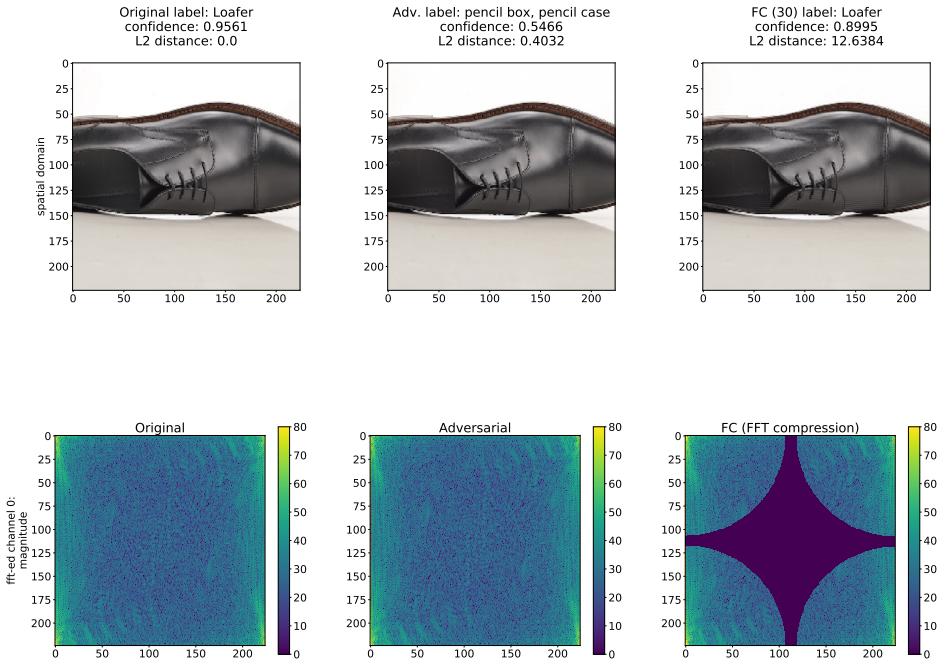

Figure 5.2: ***Spatial and Frequency domains for original, adversarial, and recovered images.*** The original image from the ImageNet dataset, its adversarial example, and the state of the image after recovery from the attack via the 30% compressed Fourier Channel (FC) showed in spatial and frequency domains.

In Figure 5.3, we further zoom into a single attack and recovery instance using FFT. In the ATTACK box, an adversary adds an imperceptible adversarial noise that we enlarge 100X so that it is visible. The network is fooled by the adversary and classifies the adversarial image incorrectly as a tiger cat instead of a bull mastiff. In the DEFENSE box, we perturb the image even more than the adversary by applying the FFT-based compression to the attacked image. After applying the perturbation via FFT, the network recovers the correct label and returns bull mastiff.

The unification of input perturbation defenses gives us an insight into how an attacker might avoid them even if they did not know the particular defense. Our experiments suggest that all the input perturbation defenses are vulnerable to the same types of attack strategy where the attacker

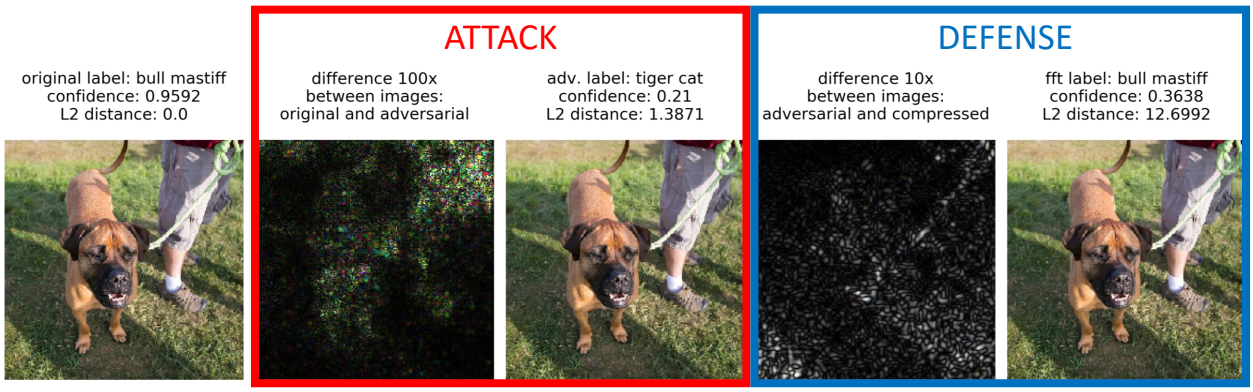

Figure 5.3: ***Attack and Recovery noise.*** We present a single instance of attack and recovery along with adversarial and perturbation noise.

simply finds an adversarial example further away from the original image. This optimization procedure can be tuned to find the smallest distance from the original image that closes a "recovery window" (demonstrated in our experiments). We can further optimize this distance with a generic attacker that assumes a particularly strong perturbation, based on the additive Laplace noise. Adaptive attacks designed on this channel are often successful against other defenses. This result implies that input perturbation defenses are simply not effective and attackers can easily circumvent them without much knowledge about the particular defense.

Our experiments compare different placements of noise layers across the network. In general, we find that the most effective current defenses leverage a strategy used in RobustNet [103], which adds noise throughout the network. On the other hand, we show that simply adding noise in a single layer, as in [100], is ineffective in practice. We present a schematic diagram in Figure 5.4 that presents a difference in the noise placement between the input perturbations and RobustNet. The input perturbation defenses place a noise layer only in front of the network. The more effective RobustNet defense injects noise before every convolutional operation.

We also find that training the network with this noise is crucial for high accuracy, and there is a threshold of perturbation after which a network must be trained with the noise to achieve any reasonable performance (approximately the standard deviation of a perturbed tensor). Our empirical findings indicate that the injection of noise into internal layers is an important and effective defense that can be combined with adversarial training [109] to further improve robustness.

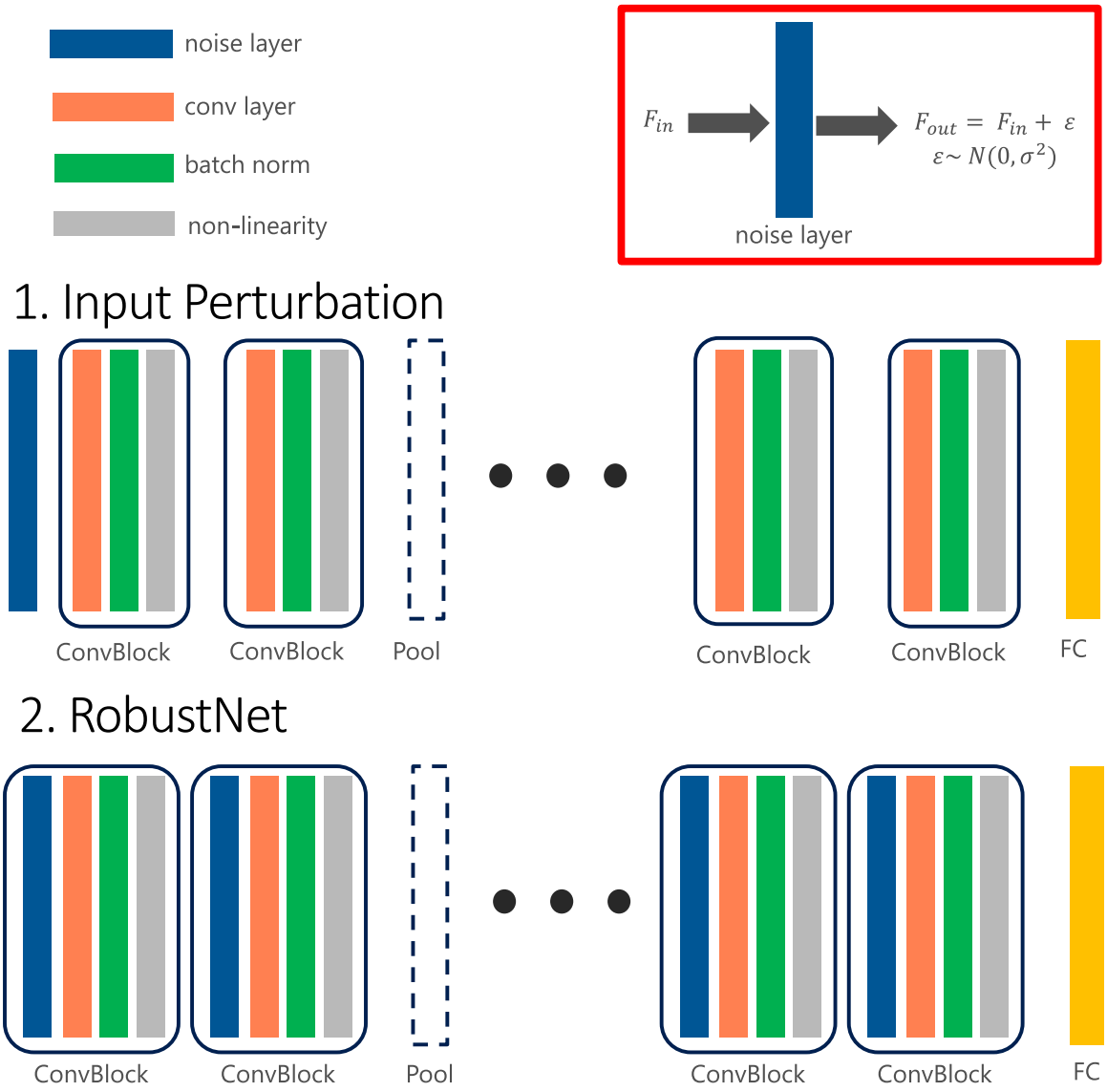

Figure 5.4: ***Input Perturbations vs RobustNet.*** We present a difference in the noise placemenet between the input perturbations and RobustNet.

We believe that our analyses are valuable to the community as they highlight important properties of perturbation-based defenses and the principles of constructing new strong defenses.

5.2 Related Work

Much of the community's current understanding of adversarial sensitivity in neural networks is based on the seminal work by Szegedy et al. [158]. Multiple contemporaneous works also stud-

ied different aspects of this problem, postulating linearity and over-parametrization as possible explanations [13, 65]. Since the beginning of this line of work, the connection between compression and adversarial robustness has been recognized. Researchers noticed this property a few years ago with a number of inference-time "input perturbation" defenses, for example, feature squeezing [177], JPEG compression [51], randomized smoothing [33], and types of structured perturbations [86, 184, 67]. Other main defense strategies include: the idea of defensive network distillation[1] [122], quantizing inputs using thermometer encoding [18], frequency-based compression harnessed by [8, 9, 37, 38, 48, 67, 105].

Another highly related line of research leverages randomization for adversarial robustness: pixel deflection [124], random resizing and padding of the input image [176], total variance minimization [67], dropout randomization [56], random pixel elimination followed by matrix estimation [179]. More effective perturbation-based defenses are possible by injecting noise to the internal layers [100, 103], model parameters [75], or via a separately trained auto-encoder added in front of a network [100]. There are also defenses based on random perturbations that give proven guarantees of robustness [33, 100, 184]. More recent work focuses on a combination of randomized smoothing with adversarial training and achieves state of the art in terms of the provable robustness [137].

Despite all of this research and several theoretical results, the empirical success of these approaches has not completely been established. Many aforementioned defenses were later broken [6, 22, 165], and many recent defenses have not been compared against each other in a standard-setting. This motivates our analysis.

1. The distillation is a form of compression, however, the defensive distillation does not result in smaller models.

5.3 Notation and Metrics

We consider convolutional neural networks that take $w \times h$ (width times height) RGB digital images as input, giving an example space of $\mathcal{X} \in (255)^{w \times h \times 3}$, where (z) denotes the integer numbers from 0 to z. We consider a discrete label space of k classes represented as a confidence value $\mathcal{Y} \in [0,1]^k$. Neural networks are parametrized functions (by a weight vector θ) between the example and label spaces $f(x;\theta) : \mathcal{X} \mapsto \mathcal{Y}$.

An adversarial input x_{adv} is a perturbation of a correctly predicted example x that is incorrectly predicted by f:

$$f(x) \neq f(x_{adv})$$

The measure the *distortion*, for example, as the ℓ_2 error between an original input x and its adversarial example x_{adv}:

$$\delta_{adv} = \|x - x_{adv}\|_2$$

5.4 Lossy-Channel Model

5.4.1 Unified View on Perturbations

While there is extensive work on using randomization or compression as a defense, we find that all of the approaches essentially follow the same format. They approximate $f(\cdot)$ with a less precise version $\bar{f}(\cdot)$ that counter-intuitively makes it more robust [51] and an adversarial example reverts back to the original class:

$$f(x) = \bar{f}(x_{adv})$$

Intuitively, a lossy version of f introduces noise into a prediction that dominates the strategic perturbations found by an adversarial attack procedure. It turns out that we can characterize a number of popular defense methodologies with this basic framework.

Let x be an example and f be a trained neural network. Precise evaluation means running $f(x)$ and observing the predicted label. Imprecise evaluation involves first transforming x through a

deterministic or stochastic noise process $C(x) = C[x' \mid x]$, and then evaluating the neural network

$$y = f(x') \quad x' \sim C(x)$$

We can think of $C(x)$ as a noisy channel (as in signal processing). The *distortion* of a $C(x)$ is the expected ℓ_2 reconstruction error:

$$\delta_c = \mathbf{E}[\|C(x) - x\|_2],$$

which is a measure of how much information is lost passing the example through a channel.

This paper shows that there is a subtle trade-off between δ_c and δ_{adv}. In particular, we can find δ_c such that $\delta_c >> \delta_{adv}$ and $f(x) = f(C(x_{adv}))$. We show that compression and randomization based techniques exhibit this property.

We have discussed the input perturbations that first transform x through a noisy channel $C(x) = C[x' \mid x]$, and then evaluate the neural network $\hat{f}(x) = f(x')$, where $x' \sim C(x)$. One could also perturb the network itself by adding ε noise to the parameters: $\hat{f}(x) = f(x; \hat{\theta})$, where $\hat{\theta} = \theta + \varepsilon$ or by injecting noise to any or all of the intermediate layers of a network $f(x) = g(h(x))$, where $\hat{f}(x) = g(h(x + \varepsilon_1) + \varepsilon_2)$.

5.4.2 Deterministic Channels

Color-depth compression

When $C(x)$ is deterministic it can be thought of as a lossy compression technique. Essentially, we run the following operation on each input example:

$$x' = \texttt{compress}(x)$$

One form of compression for CNNs is color-depth compression. Most common image classification neural network architectures convert the integer-valued inputs into floating-point numbers. We abstract this process with the `norm` function that for each pixel $n \in (255)$ maps it to a real

number $v \in [0,1]$ by normalizing the value and the corresponding `denorm` function that retrieves the original integer value (where $\lfloor \rceil$ denotes the nearest integer function) [2]:

$$\texttt{norm}(n) := \frac{n}{255} \qquad \texttt{denorm}(v) := \lfloor 255 * v \rceil$$

This process is normally reversible $v = \texttt{norm}(\texttt{denorm}(v))$, but we can artificially make this process lossy. Consider a parametrized $C(\cdot)$ version of the color-depth compression function:

$$C(v,b) := \frac{1}{2^b - 1} \cdot \lfloor (2^b - 1) * v \rceil$$

By decreasing b by Δb we reduce the fidelity of representing v by a factor of $2^{\Delta b}$ (for the b bits of precision).

FFT-based compression

We apply compression in the frequency domain to reduce the precision of the input images. Let x be an input image, which has corresponding Fourier representation that re-indexes each tensor in the frequency domain:

$$F[\omega] = F(x[\mathbf{n}])$$

This Fourier representation can be efficiently computed with an FFT. The mapping is invertible $x = F^{-1}(F(x))$. Let $M_f[\omega]$ be a discrete indicator function defined as follows:

$$M_f[\omega] = \begin{cases} 1, \omega \leq f \\ 0, \omega > f \end{cases}$$

$M_f[\omega]$ is a mask that limits the $F[\omega]$ to a certain *band* of frequencies. f represents *how much of the frequency domain* is considered. The *band-limited* spectrum is defined as, $F[\omega] \cdot M_f[\omega]$, and

2. More complex normalization schemes exist but for ease of exposition we focus on this simple process.

the band-limited filtering is defined as:

$$x' = F^{-1}(F[\omega] \cdot M_f[\omega])$$

SVD-based compression

Analogously to the FFT-based method, we decompose an image with SVD transformation and reconstruct its compressed version with dominant singular values. The bases used in SVD are adaptive and determined by an image, as opposed to the pre-selected basis used in FFT. This can result in a higher quality for the same compression rate in the case of SVD, however, it is more computationally intensive than FFT-based compression.

5.4.3 Stochastic Channels

The channel model is particularly interesting when $C(x)$ is stochastic. Randomization has also been noted to play a big role in strong defenses in prior work [33, 109, 184]. For example, we could add independent random noise to each input pixel:

$$x' = x + \varepsilon$$

We consider the following forms of noise:

1. Gaussian noise: $\varepsilon \sim N(\mu = 0, \sigma)$ (with zero mean μ and σ standard deviation);
2. Uniform noise: $\varepsilon \sim U(-B, B)$ (with bound B);
3. Laplace noise: $\varepsilon \sim L(\mu = 0, b)$ (with zero mean and scale b).

One of the advantages of randomization is that an adversary cannot anticipate how the particular channel C will transform an input before prediction. There is another subtle advantage to randomization. Randomized approaches can partially recover their loss in accuracy due to imprecision by averaging over multiple instances of the perturbation. For multi-label classification

problems with K labels, we can take the most frequent label seen after T perturbation trials (also called a majority vote or consensus):

$$\bar{f}(x) = \arg\max_{1...K} \sum_{i=1}^{T} f_k(x+\varepsilon) \tag{5.1}$$

where f_k is the vote 0 or 1 for label $k \in [1,...,K]$ in trial $t \in [1,...,T]$. We can also select the label based on the highest average probability (as in ensemble models), where similarly f_k is the probability $P \in [0,1]$ for label $k \in [1,...,K]$ in trial $t \in [1,...,T]$. Usually, the latter method provides improved estimates of the class probabilities and also tends to produce ensemble classifiers with lower variance, especially for ensembles with small number of models [59] (chapter 8.10).

However, this method can be applied only to weak adversarial perturbations and the ensemble methods deteriorate the final accuracy for a strong adversary. In general, the ensemble method performs better than a single model if each of the ensemble models is a good predictor. On the other hand, when a single predictor does not perform well then the ensemble of the weak predictors magnifies the bad effect and makes the ensemble model worse than a single predictor.

RobustNet [103] is an example of a model that can be extended to an ensemble during test time by performing several forward propagations, each time with different prediction scores due to the noise layers. For weak attacks, RobustNet performs well and its good performance is amplified by the ensemble. For strong attacks, the RobustNet predictor performs poorly and ensemble magnifies the errors. Additionally, from the system perspective, the ensemble for RobustNet decelerates the inference process.

5.5 Experimental Setup

We run our experiments using ResNet-18, ResNet-20, or VGG-16 on CIFAR-10, ResNet-20 on SVHN, and ResNet-50 on ImageNet using P-100 GPUs (16GB memory). We explore a number of different attacks that are implemented in the foolbox library [129]. In each experiment, we measure the test accuracy (%), the confidence of predictions, and distances between the original images and either their adversarial counterparts or the recovered images after applying one of the defenses. We present our results for *non-targeted attacks*; if the adversary is successful it induces any misclassification.

5.6 White-box Attacks

The white-box attacks (also called gradient-based) assume that an adversary has access to a parametric description of the model he/she is attacking. White-box adversarial examples can be synthesized in three main settings. In the *adaptive* setting, the attacker knows the model and what defense is used. In the *non-adaptive* setting, the attacker does not know about the defense technique and assumes no defense is used. In our experiments, we consider a novel *partially adaptive setting*, where the attacker knows that a perturbation-based defense is being used but does not know precisely which one.

We apply the following main white-box attacks:

- **Carlini-Wagner** L_2 (C&W L_2) attack is a generalization of the LBFGS optimization algorithm and devised after exhaustive search over possible space of: norms, loss functions, box optimization procedures, etc. [21].

- **PGD** L_∞ the Projected Gradient Descent Attack that is an iterative version of the FGSM attack; we use the version that minimizes the L_∞ distance [109].

We also experiment with other gradient-based attacks provided in the foolbox library [129]: LBFGS (introduced by [158] and further extended in [159]), FGSM (Fast Gradient Sign Method [65]),

L1 BIM (Basic Iterative Method [95])).

Score-based attacks (e.g., Single Pixel Attack) do not require gradients of the model but they expect meaningful scores such as probabilities or logits that are used to approximate gradients.

For the white-box adaptive case, we extend attacks to reduce the effects of gradient obfuscation. We approximate the gradients for the backward pass on the compression layers (usually as an identity function), similarly to [6, 73]. For the RobustNet [103] and PNI [75], the C&W attack is aware of the randomization procedure and we use its adaptive version presented in [103]. We also use the PGD attack from [75] and add EOT [6].

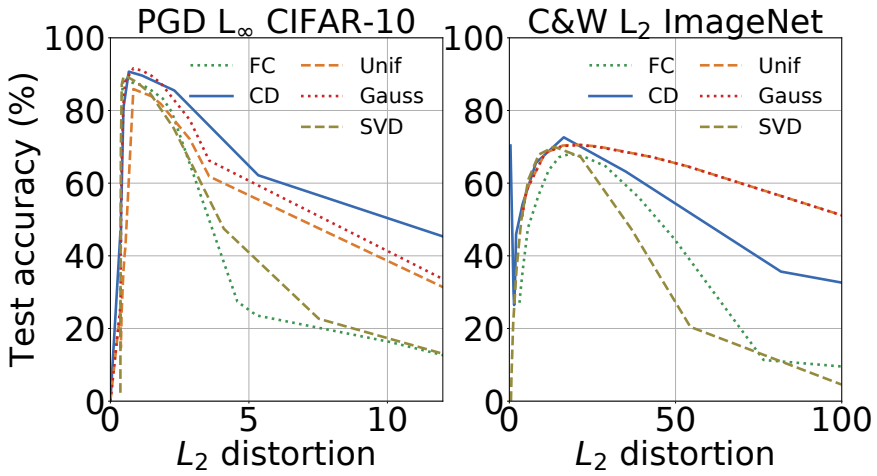

Figure 5.5: ***Input perturbations provide similar gains in robustness.*** Distortion is due to defense. Test accuracy on clean data is 93.56% and 76.13% for CIFAR-10 (ResNet-18) and ImageNet (ResNet-50), respectively. Defenses: FC - Frequency-based Compression, CD is Color-Depth reduction (feature squeezing), Uniform & Gaussian noise, SVD compression.

5.6.1 Input Perturbation Defenses Lead to Similar Gains in Robustness

We first study the simplest class of perturbation defenses that perturb only inputs. In this experiment, we consider the non-adaptive setting where the attacker does not know about the defense. The attack is untargeted, any misclassification is considered a success. An interesting way to compare very different classes of input perturbation defenses is to look at the "input distortion" (L_2 distance between the input and its perturbation, not to be confused with the distortion of the attack). We compare this metric against the test accuracy of a defended model, which indicates the

ability of the perturbation to recover the original label.

For each image from the CIFAR-10 test and ImageNet dev sets, we generate an adversarial attack using PGD (40 iterations with a random start) and C&W (100 iterations). The attacks are tuned to minimize the distance between adversarial inputs and original examples. We present the results in Figure 5.5. L_2 distortion of 0.0 indicates no defense and the attacks reduce the test accuracy to 0%. The defenses are applied only during inference. Each of the tested perturbation techniques can recover a substantial portion of correct labels from the adversarial examples. They recover the accuracy to about 85% for CIFAR-10 and 70% for ImageNet. The curves are very similar in terms of where they achieve their peak robustness—the particular distribution of perturbations is not as relevant as their magnitude. The "optimally-tuned" defense essentially finds the same distortion parameter across very different stochastic and deterministic techniques.

5.6.2 How to Attack Input Perturbation Defenses Non-adaptively?

Input perturbation defenses might seem a simple and effective trick to make a model adversarially robust, however, their similarity is a major pitfall. Even if the attacker does not know which particular defense is being used he/she can still attack the model with a "less-precise" attack—one that creates a larger distortion but higher confidence adversarial image. The experiment in Section 5.6.1 shows that the defense philosophy is to generate a perturbation big enough to dominate the adversarial noise but small enough to generate valid predictions. This creates an easy to recognize attack vector, where the attacker simply makes the adversarial perturbations large enough that the defender significantly hurts the accuracy of the model when trying to dominate the adversarially placed strategic perturbations. Such attacks remain imperceptible (see Figure B.4).

5.6.3 Partially Adaptive Setting

The perturbation-based defenses fail if the attacker makes the adversarial distortion large enough. Can an attacker still break such defenses with a low-distortion attack? One approach is to consider the adaptive setting. If the attacker has full knowledge of the defense, it is possible to construct an

A \ D	FC	CD	SVD	Gauss	Uniform	Laplace
Empty	93.32	93.01	93.12	92.53	91.6	91.35
FC	**0.20**	80.75	83.05	81.15	79.65	78.70
CD	3.85	**0.70**	43.60	47.30	60.45	62.35
SVD	1.99	47.96	**0.77**	46.52	62.87	65.75
Gauss	4.45	48.70	44.80	51.50	61.75	60.15
Uniform	3.45	30.30	30.60	30.15	48.05	51.55
Laplace	3.05	23.35	24.60	**23.80**	**39.15**	**46.70**

Table 5.1: ***Transferability of adversarial images for the CIFAR-10 dataset*** created using partially adaptive attack (A) and tested against defense (D). Each cell represents a recovered test accuracy (%). The clean test accuracy is 93.56% for CIFAR-10 dataset trained on ResNet-18 architecture.

(adaptive) attack that is impervious to the defense. It has been known that many perturbation-based defenses are easily broken in the adaptive setting.

We go one step further. Since our experiments show that input transformations are so similar in their mechanisms, we find that an attacker *does not need to be fully adaptive*. To the best of our knowledge, such a problem setting has not been studied in the previous literature. The attacker simply assumes a particular strong perturbation-based defense and that same adversarial input often transfers to other defenses. Our attacker assumes that the defender is using a Laplace noise perturbation to defend the model, and generates an attack. In the adaptive attack setting, the deterministic channels are fully broken, and the randomized channels are mostly broken (with a maximum accuracy of about 23.8%). We can strengthen the attack by giving to an adversary a larger *compute budget*, for example, with an unlimited number of adaptive steps (Figure B.5 in Appendix) and thereby driving the accuracy to 0%.

We present result in Table 5.1. We use 30% FC compression, 50% SVD compression, 4-bit values in CD, 0.03 noise level for Gauss and Laplace, and 0.04 noise level for the Uniform channel. We use the ResNet-18 model, 2000 images from the CIFAR-10 test set, and 100 attack iterations with 5 binary steps to find the c value (with initial c value set to 0.01) for the adaptive C&W L_2 attack.

A \ D	FC	CD	SVD	Gauss	Uniform	Laplace
FC	**0.10**	75.50	75.83	77.04	77.49	76.29
CD	0.17	**1.16**	6.77	62.60	62.04	65.46
SVD	12.02	72.33	**0.46**	72.79	72.52	73.09
Gauss	0.57	26.67	6.67	58.68	58.62	64.95
Uniform	0.50	26.71	6.99	58.48	59.06	64.59
Laplace	0.33	18.59	4.16	**29.76**	**29.84**	**50.00**

Table 5.2: ***Transferability of adversarial images for the ImageNet dataset*** (this is an extension of the results presented in Table 5.1).

Laplace attacked images transfer the best to other defenses. They decrease the accuracy of the defense models by at least 44.3% (for the Laplace-based defense itself). FC attacked images do not transfer well to other defenses and the maximum drop in accuracy of the model protected by other defenses is 12.26%. Most adversarial images (against a given defense) transfer very well to the FC defense. An adversarial image against any defense (e.g. CD, SVD, Gauss, Uniform, or Laplace) is also adversarial against the FC defense. The adversarial images generated against the Uniform defense show better transfer to other defenses in comparison to the adversarial images generated against the Gaussian defense. This is because the higher noise level is applied in the Uniform defense.

We observe analogous trends for the ImageNet dataset and present results in Table 5.2. We use 30% FC compression, 4-bit values in CD, 50% SVD compression, 0.04 noise level for the Gauss and Uniform channels, and 0.03 noise level for Laplace channel. We use 3000 images from the ImageNet validation set and 100 iterations with 5 binary steps to find the c value (with an initial c value set to 0.01) for the adaptive C&W L_2 attack.

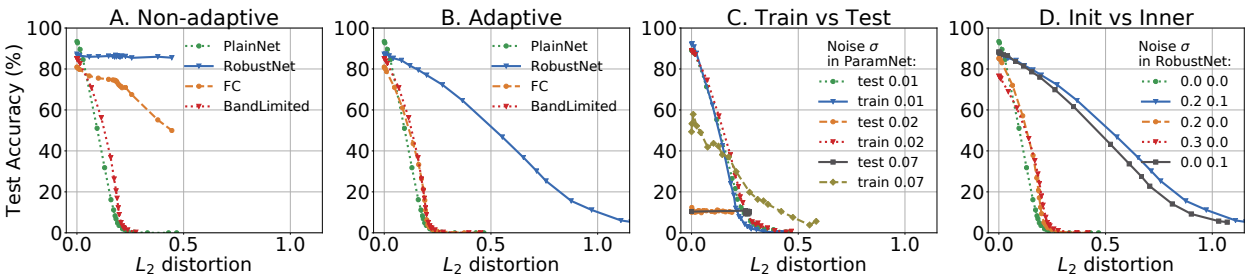

Figure 5.6: *Comparison between attacks.* ***A. Non-adaptive attack.*** Adversarial images generated on PlainNet and tested against RobustNet, FC, and BandLimited defenses. ***B. Adaptive.*** Generate the attack with knowledge about the defenses. ***C. Train vs Test.*** The robustness of ParamNet with noise added either during both train and test phases or only during test. ***D. Init vs Inner.*** Higher robustness when noise layers placed before every convolutional layer (0.2 0.1) than only before the first layer (0.2 0.0 and 0.3 0.0) or only in internal layers (0.0 0.1). We use C&W L2 attack and VGG-16 trained on CIFAR-10.

5.6.4 Robustness to Adaptive Attacks

The previous set of experiments shows that simply adding post-hoc perturbations to a model is not an effective defense. A crucial way to boost the robustness of perturbation based defenses is to *train the model* observing training data perturbed by the future defense. This approach seems similar to adversarial training but turns out to be much more computationally efficient as the perturbations are usually easy to calculate. While it has been previously noted that adversarial examples can fool multiple models [163], we find that perturbation defenses significantly affect transferability.

We test FC – the frequency-based channel with a 50% compression rate, which is one of the better performing input perturbations in this setting. RobustNet [103] adds a random noise layer in front of each convolutional layer (we set the noise in the initial layer (init) to 0.2, and in all the internal layers (inner) to 0.1 as in [103]). The BandLimited model (described in Chapter 4) modifies each convolutional layer by using FFT-based convolutions with compression. In this experiment, we set the compression rate to 80%.

For the non-adaptive attack, adversarial images generated on PlainNet do not transfer to RobustNet (see Figure 5.6 A). On the other hand, the BandLimited model, which is robust to Gaussian noise, is not robust to gradient-based attacks. We observe that gradients estimated for the *band-limited* convolution operations are approximate enough and thus useful to generate the first-order

attacks. The FC defense provides even up to 50% robust accuracy in the nonadaptive setting (for L_2 distortion of 0.4), however, once we compute its gradients (similarly to the BandLimited layers) in the adaptive setting, the accuracy of the defense drops to 0% (Figure 5.6 B).

In general, many defenses rely on gradient obfuscation. The latest work [164] indicates that adversarial training suffers from gradient masking as well. Thus, we extend the adaptive case and test RobustNet against PGD + EOT (Expectation Over Transformation) [6]. We present a schematic diagram of the EOT method in Figure 5.7. Attacking RobustNet with 10 iterations of EOT decreases its accuracy by about 10% in comparison to the standard PGD attack. However, there are diminishing returns for more EOT iterations (see Figure 5.8). Thus, EOT helps partially but does not allow us to find fully useful gradients to defeat the defense. This result implies that RobustNet obfuscates the gradients by randomizing them. Another way to identify gradient obfuscation is to check if a defense performs better against white-box than black-box attacks. It occurs that, for example, the Boundary black-box attack [17] is unsuccessful since it executes a random walk along (in this case) a *random decision boundary* (See Section 5.7.1).

5.6.5 Noise Injection during Train vs Test Time

RobustNet randomly perturbs inputs and feature maps. An alternative approach is to randomly perturb parameters and we call such network a ParamNet. We investigate how much noise can be added to ParamNet so that the clean accuracy does not decrease significantly and the robustness of the network is improved (Figure 5.6 C). For a small amount of noise ($\sigma = 0.01$), the drop in clean accuracy is negligible, however, there is no gain in robustness. Interestingly enough, a slight increase in the amount of injected noise (to $\sigma = 0.02$) during the test phase turns the network into a random classifier. This problem arises when the values of the parameters are dominated by random noise. To restore clean accuracy, it is necessary to train the network with noise. We can improve robustness by further increasing the amount of noise ($\sigma = 0.07$) but at the cost of lower accuracy on clean data.

Figure 5.6 D demonstrates where and how to add noise to RobustNet. Adding noise to the first

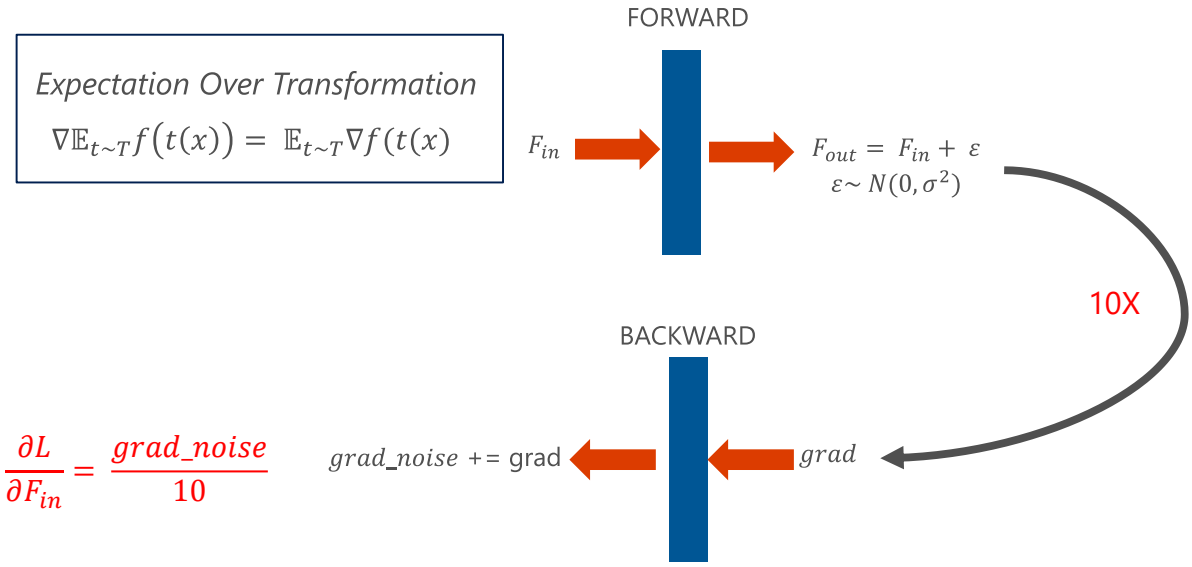

Figure 5.7: ***EOT schema.*** EOT (Expectation Over Transformation) is a method used against randomized defenses. Many attacks rely on gradients and if the gradients are noisy then they are not useful. The EOT method computes the gradients in many iterations and averages them to obtain a more accurate gradient. In practice, it is sufficient to aggregate the gradients after running the forward and backward pass 10 times.

layer is essentially the same as input perturbation. We could also add noise only to the internal layers [100], however, to achieve a high level of robustness, we have to add noise to all the layers.

It occurs that RobustNet is significantly more robust than ParamNet (Figures 5.6 C and D). The reason is two-fold. First, the number of parameters is an order of magnitude lower than the number of elements in the input and intermediate feature maps [132]. Second, the standard deviations of tensors for features are an order of magnitude higher than for parameters. Thus, a more effective amount of variation can be added to features than to parameters.

5.6.6 Noise Injection and Adversarial Training

Adversarial training (AdvTrain) [109] is one of the most successful practical defenses. RobustNet is another strong defense that injects noise into inputs and intermediate feature maps [103]. PNI-W-Adv [75] combines noise injection into network parameters with adversarial training and can give higher robustness than the pure AdvTrain defense. Previous Section 5.6.5 shows that injecting

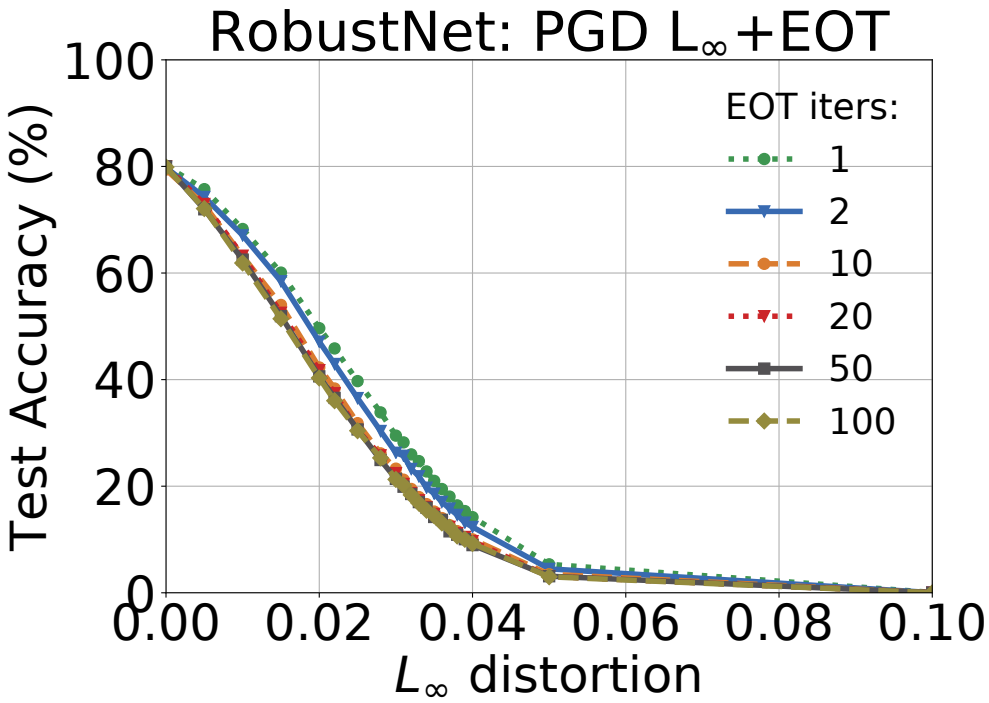

Figure 5.8: ***PGD + EOT.*** We use the PGD attack with 40 iterations and EOT with different number of iterations tested against RobustNet. We train ResNet-20 model on CIFAR-10 dataset. Clean accuracy is 88%.

noise into network layers performs better than noise injection into parameters. Thus, we propose to combine RobustNet with AdvTrain (denoted as RobustNetAdv) and compare it to AdvTrain, PNI-W-Adv, and RobustNet defense methods.

Attack Setup

We run the C&W attack with 200 iterations for the gradient descent and vary the c parameter from 0 to 100. For all compared defenses, adversarial training uses PGD with 7 iterations, similarly to [75, 109], the perturbation scale of 8/255 (0.031), and step size 2.55/255 (0.01). We only vary either the number of iterations for the attack or perturbation scale ε (which influences the L_∞ distortion of adversarial examples). Besides using 40 and 100 total attack iterations, we also increase the iterations to 1000 to further strengthen the adversary, similarly to [179]. The current state of the art attack is white-box and adaptive. RobustNet and PNI-W-Adv employ the randomization techniques so we use a modified version of attacks with EOT (Expectation Over Transformation) [6].

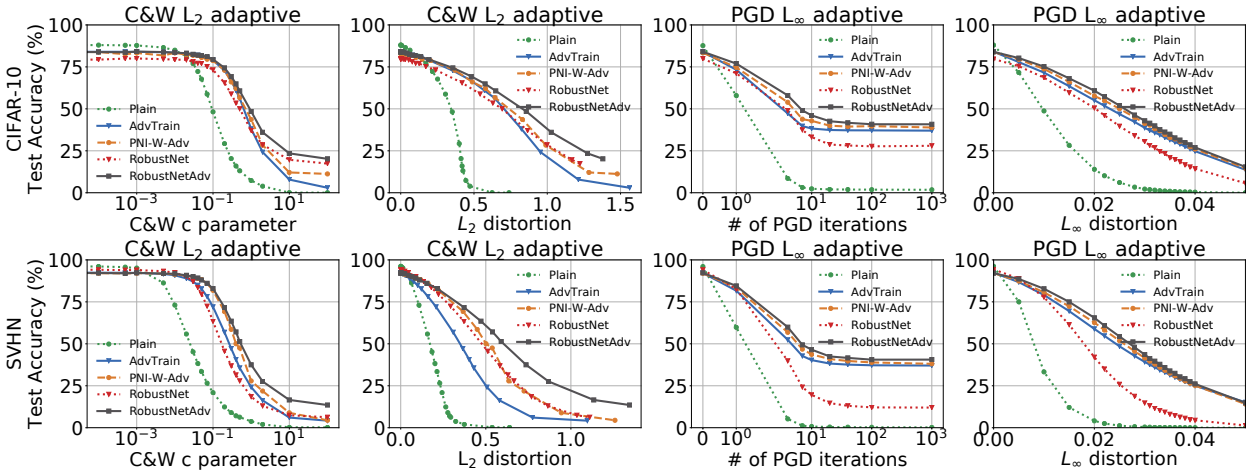

Figure 5.9: White-box adaptive attacks C&W and PGD used to generate adversarial examples and tested against the standard model (Plain) as well as the following defenses: adversarial training (AdvTrain), parameter noise injection to the weights that uses adversarial training (PNI-W-Adv), feature noise injection (RobustNet), and RobustNet combined with adversarial training (RobustNetAdv). We train ResNet-20 on CIFAR-10 and SVHN datasets.

Defense Setup

We tune noise layers in RobustNet. The standard deviations of inputs to convolutional layers are guiding values for the injected noise level. To achieve good performance, we adjust the noise separately for at least the initial and internal layers. PNI-W-Adv goes a step further and injects noise scaled by a *trained factor*. The initial noise magnitude is based on the standard deviation of network parameters. However, it requires us to adjust weights in ensemble loss, which consists of two terms for clean data loss and adversarial data loss. Training RobustNet in a similar manner to PNI-W-Adv usually lowers the injected noise scale too much and makes the network much less robust.

For RobustNet, we set the standard deviation σ of the injected noise to 0.2 (and 0.08) in the input layer and 0.1 (and 0.07) in all the remaining internal layers for CIFAR-10 (and SVHN). For RobustNetAdv, we keep the same $\sigma = 0.1$ for all the noisy layers for CIFAR-10 and use the same 0.08 for initial and 0.07 for internal layers for SVHN. We train PNI-W-Adv in the same setup as in [75], with the equally weighted (by 0.5) sum of losses for clean and adversarial data.

We present the results in Fig. 5.9 as the test accuracy being a function of either strength of an

attack or its incurred distortion. The former approach allows us to adjust attack parameters while the latter allows direct comparison between defenses.

Best Performance: RobustNet + Adversarial Training

AdvTrain and PNI-W-Adv are trained against PGD and for this attack perform better than RobustNet. However, for CW, RobustNet outperforms AdvTrain. Then, RobustNetAdv provides an improvement also in comparison to PNI-W-Adv for CIFAR-10 and SVHN datasets when using CW or PGD attacks. PNI-W-Adv is on par with AdvTrain for CW for most distortion levels and slightly outperforms it for highly distorted examples, similarly to RobustNet. The placement of the noise injections is important and RobustNetAdv outperforms PNI-W-Adv in most cases.

RobustNet Noise Types and Adversarial Training

We also test different types of noise: Gauss, Uniform, and Laplace injected into RobustNet. Following the general rules established in the preceding Section 5.6.1, the robustness of the network is independent of the type of injected noise. We test these different types of noise injected into RobustNet and present results in Figure 5.10. RobustNet performs much better than the Plain network and we can boost its robustness by combining it with adversarial training (RobustNet Gauss Adv. Train).

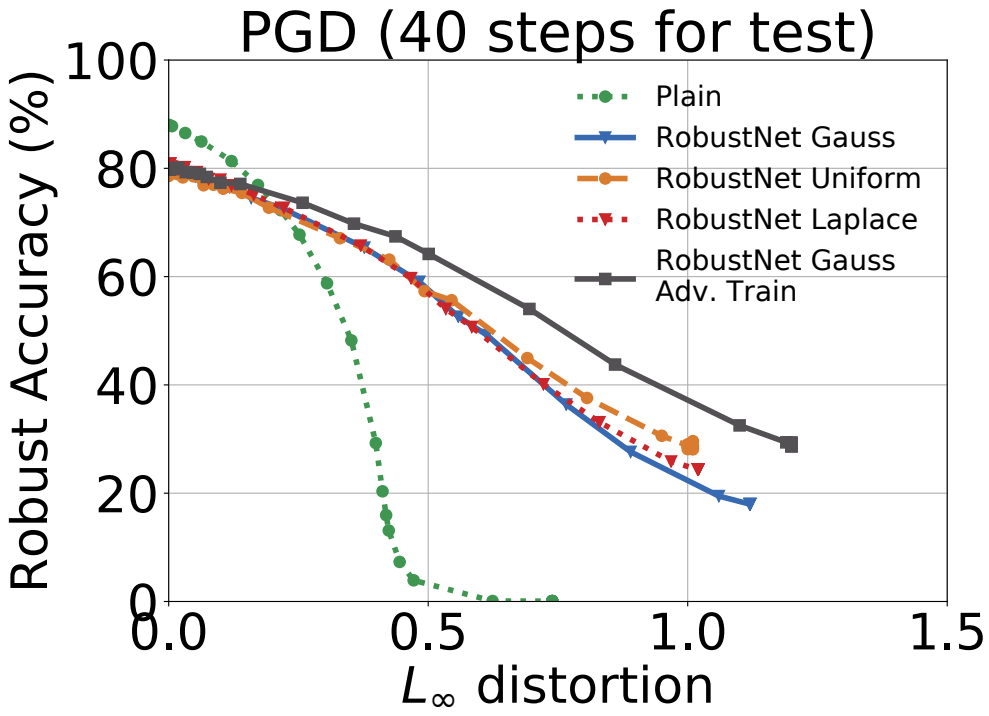

Figure 5.10: ***Different types of noise injected into RobustNet and Adversarial Training*** We use the PGD attack with 40 iterations. We train ResNet-20 model on CIFAR-10 dataset. Clean accuracy is 88%. The Gauss, Uniform, and Laplace noise provide very similar levels of robustness.

5.7 Black-box Attacks

In the previous sections, we test models against white-box attacks that are based on gradients. Next, we investigate the performance of models against black-box attacks. As a black-box attack, we define an attack that does not need knowledge about the gradient or the model. These attacks are also called zero-order methods. We usually use them to attack models that we do not have direct access to, for example, when models are exposed only via an external API. [3] The black-box attacks also help to assess if a given model obfuscates gradients.

5.7.1 Decision-based Attacks

The decision-based attacks require neither gradients nor probabilities. They operate directly on the images and rely only on the class decision of the model. An example of a strong decision-based attack is the Boundary Attack [17].

Boundary Attack

We test our models against a state-of-the-art black-box Boundary Attack [17], that intuitively is based on a random walk along a decision boundary.

It is not trivial to find an initial random adversarial example when attacking RobustNet (or PNI-W-Adv). The Boundary Attack is initialized by random adversarial examples drawn from a uniform distribution. The uniform noise is applied many times (subject to a hyper-parameter) in different directions. After an initial adversarial example is found, the iterative interpolation is used between the found random adversarial example and its corresponding original image by gradually moving from the original image towards the adversarial example until the interpolated example is misclassified. However, RobustNet applies random perturbations during inference, which becomes a non-deterministic process that can classify differently the same image during consecutive tests. Thus, initially found random adversarial examples might not remain adversarial for the next step

3.

of the attack. This hinders the proper initialization of the Boundary Attack and causes that only about 20% of the initially found adversarial examples remain adversarial for the main part of the algorithm.

To ensure a high rate of initial adversarial examples, we run the initialization until all examples in a given batch are adversarial and apply 25K iterations for the main part of the algorithm, in which random directions are explored and a similar issue to the initialization repeats. This could be partially ameliorated by testing the same random direction many times but it makes the already expensive algorithm a few times more costly. The Boundary Attack optimizes for the L_2 distance between the adversarial and original examples, so we plot the test accuracy (%) against a wide range of possible L_2 distortions (from 0.0 to 5.0). We present the results in Figure 5.11.

The parameter noise injection to the weights (PNI-W-Adv) performs similarly to the RobustNetAdv defense. Thus, an appropriate perturbation (either in parameter or feature map space), combined with adversarial training, gives a more robust defense than either of the methods alone. The RobustNet and PNI models are randomized models and their decision boundary is also stochastic. Thus, the Boundary Attack executes the random walk on the random decision boundary and this approach fails to evade RobustNet.

Robustness to Uniform and Gaussian Noise

We evaluate the robustness of band-limited CNNs. Specifically, models trained with more compression discard part of the noise by removing the high-frequency Fourier coefficients (FC channel). In Figure 5.12, we show the test accuracy for input images perturbed with different levels of uniform and Gaussian noise, which is controlled systematically by the sigma parameter, fed into models trained with different compression levels (i.e., 0%, 50%, or 85%) and methods (i.e., band-limited vs. RPA-based[4]). Our results demonstrate that models trained with higher compression are more robust to the inserted noise. Interestingly, band-limited CNNs also outperform the

4. The Reduced Precision Arithmetic, where operations on 16-bit floats are used instead of on 32 or 64-bit float numbers.

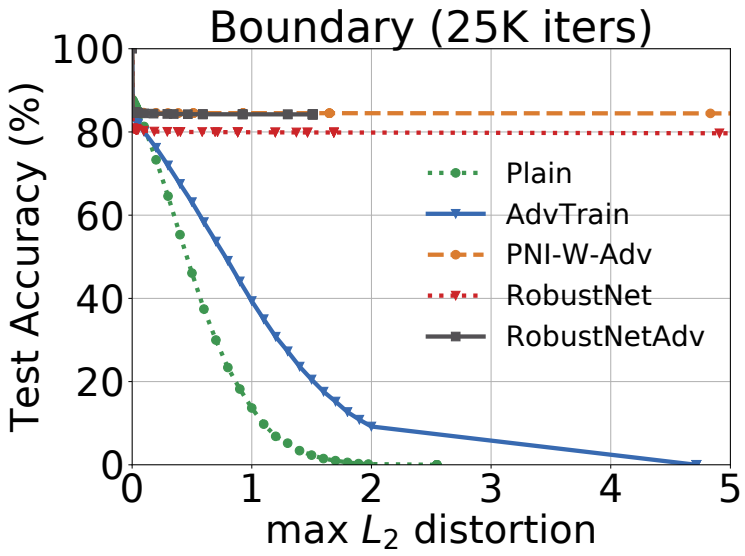

Figure 5.11: ***Robustness to the Boundary Attack.*** We use the Boundary Attack with 25K iterations tested against the standard model (Plain) as well as the following defenses: adversarial training (AdvTrain), parameter noise injection to the weights that uses adversarial training (PNI-W-Adv), feature noise injection (RobustNet), and RobustNet combined with adversarial training (RobustNetAdv). We train ResNet-20 model on CIFAR-10 dataset.

RPA-based method and under-fitted models (e.g., via early stopping), which do not exhibit the robustness to noise.

Input test images are perturbed with uniform or Gaussian noise, where the sigma parameter is changed from 0 to 1 or 0 to 2, respectively. The more band-limited model, the more robust it is to the introduced noise.

Contrast Reduction Attack

This black-box attack gradually distorts all the pixels:

$$\text{target} = \frac{\max + \min}{2}$$

$$\text{perturbed} = (1-\varepsilon) * \text{image} + \varepsilon * \text{target}$$

where min and max values are computed across all pixels of images in the dataset.

We can defend the attack with CD (Color Depth reduction) until a certain value of epsilon, but

then every pixel is perturbed smoothly so there are no high-frequency coefficients increased in the FFT domain of the image. The contrast reduction attack becomes a low-frequency based attack when considered in the frequency domain. Another way to defend the attack is to run a high-pass filter in the frequency domain instead of the low-pass filter.

We run the experiments for different models with CD and two band-limited models (the model with full spectra and no compression as well as a model with 85% of compression - with FC layers). The CD does defend the attack to some extent and the fewer pixels per channel (the *stronger* the CD in a model), the more robust the model is against the contrast reduction attack.

Test accuracy as a function of the contrast reduction attack for ResNet-18 on CIFAR-10 (after 350 epochs) is plotted in Figure 5.12. We control the strength of the attack with parameter epsilon that is changed systematically from 0.0 to 1.0. We use the whole test set for CIFAR-10. R denotes the number of values used per channel (e.g., R=32 means that we use 32 values instead of standard 256).

Multiple-pixels Attack

The foolbox library supports a single-pixel attack, where a given pixel is set to white or black. A certain number of pixels (e.g., 1000) is chosen and each of them is checked separately if it can lead to the misclassification of the image. The natural extension is to increase the number of pixels to be perturbed, in a case where the single-pixel attack does not succeed. We present results for the multiple pixel attack in Figure 5.12.

5.7.2 Spatial-based Attacks

Spatial attacks apply adversarial rotations and translations that can be easily added to the data augmentation during training. However, these attacks are defended neither by removing the high-frequency coefficients nor by quantization (CD) . We separately apply rotation by changing its angle from 0 to 20 degrees and do the translations within a horizontal and vertical limit of shifted pixels (Figure 5.12).

5.7.3 Score-based Attacks

The score based attack requires access to the model predictions and its probabilities (the inputs to the softmax) or the logits to estimate the gradients.

Local Search Attack

The local search attack estimates the sensitivity of individual pixels by applying extreme perturbations and observing the effect on the probability of the correct class. Next, it perturbs the pixels to which the model is most sensitive. The procedure is repeated until the image is misclassified, searching for additional critical pixels in the neighborhood of previously found ones. We run the experiments for the attack on 100 test images from CIFAR-10 since the attack is relatively slow (Figure 5.12).

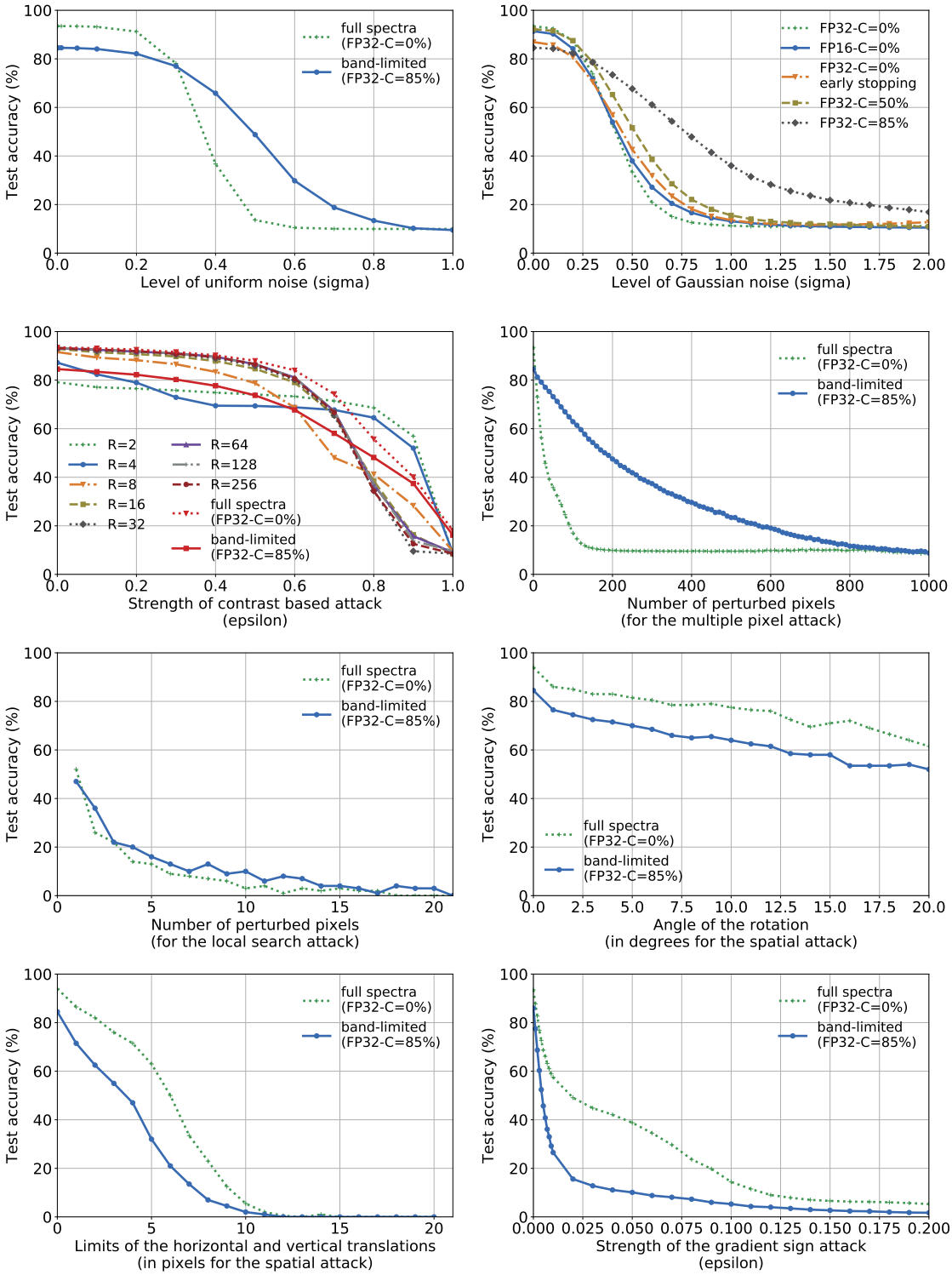

Figure 5.12: Test accuracy as a function of the strenghts of the attacks for ResNet-18 on CIFAR-10.

5.8 Perturbation Analysis

While the intuition is that the channel's perturbations dominate strategically placed distortions in an adversarial example, the underlying mathematical mechanism of why recovery is possible is not clear. We start with the hypothesis that synthesized adversarial examples have *unstable* predictions–meaning that small perturbations to the input space can change confidence values drastically. How do we quantify instability?

Let $f(x)$ be a function that maps an image to a single class confidence value (i.e., a scalar output). We want to understand how $f(x)$ changes if x is perturbed by ε. We can apply a Taylor expansion of f around the given example x:

$$f(x+\varepsilon) \approx f(x) + \varepsilon^T \nabla_x f(x) + \frac{1}{2}\varepsilon^T \nabla_x^2 f(x)\varepsilon + ...$$

where $\nabla_x f(x)$ denotes the gradient of the function f with respect to x and $\nabla_x^2 f(x)$ denotes the Hessian of the function f with respect to x. The magnitude of the change in confidence is governed by the Taylor series terms in factorially decreasing importance. $\|\varepsilon\|_2$ is the distortion measure δ_c (described at the beginning of Section 5.4.1). Thus, the expression is bounded in terms of the *operator* norm, or the maximal change in norm that could be induced, of each of the terms:

$$\varepsilon^T \nabla_x f(x) + \frac{1}{2}\varepsilon^T \nabla_x^2 f(x)\varepsilon + ... \leq \delta_c \, M_1(x) + \frac{1}{2}\delta_c^2 \, M_2(x) + ... \tag{5.2}$$

As $\nabla_x f(x)$ is a vector, this is simply the familiar ℓ_2 norm, and for the second order term this is the maximal eigenvalue:

$$M_1(x) = \|\nabla_x f(x)\|_2 \quad M_2(x) = \lambda_{max}(\nabla_x^2 f(x))$$

First, we compare the first expression on the left-hand side of the inequality 5.2 with the first expression on the right-hand side of the inequality 5.2:

$$(1) \quad \varepsilon^T \nabla_x f(x) \leq \delta_c \, M_1(x)$$

From the Cauchy-Schwarz inequality:

$$\varepsilon^T \nabla_x f(x) \leq ||\varepsilon||_2 \, M_1(x)$$

$$||\varepsilon||_2 \, M_1(x) = \delta_{adv} \, M_1(x) \leq \delta_c \, M_1(x) \;\; (\text{ since } \delta_{adv} << \delta_c)$$

Second, we compare the second expression on the left-hand side of the inequality 5.2 with the second expression on the right-hand side of the inequality 5.2:

$$(2) \quad \varepsilon^T \nabla_x^2 f(x) \varepsilon \leq \delta_c^2 \, M_2(x)$$

From the definition of maximum eigenvalue :

$$\lambda_{max} \geq \frac{\varepsilon^T \nabla_x^2 f(x) \varepsilon}{\varepsilon^T \varepsilon}$$

$$\varepsilon^T \nabla_x^2 f(x) \varepsilon \leq ||\varepsilon||_2^2 \lambda_{max} = \delta_{adv}^2 \lambda_{max} \leq \delta_c^2 \lambda_{max}$$

When M_1 and M_2 are larger this means there is a greater propensity to change the prediction for small perturbations. We will show experimentally that for certain types of attacks the M_1 and M_2 values around adversarial examples exhibit signs of instability compared to those around natural examples–suggesting a mathematical mechanism of why recovery is possible.

5.9 Adversarial Examples Are Unstable

The key question is why adversarial examples are more sensitive to perturbations than natural inputs when there is evidence that from an input perspective they are statistically indistinguishable. Our experiments suggest that this sensitivity arises from the optimization process that generates adversarial examples. In the next sections, we present gradient and Hessian based analyses.

5.9.1 Gradient-based Analysis

Based on the operator-norm analysis presented in Section 5.8, we measure L_2 norm of the input gradients w.r.t. the original x_{org} and adversarial x_{adv} images for original (correct) c_{org} and adversarial classes c_{adv}. For natural images, the class with the lowest input gradient is always the class with the highest confidence. Figure 5.16 shows that the lower the attack distortion, the more likely a "gradient anomaly" for the adversarial example, where the lowest gradient does not correspond to the highest confidence class. This is not surprising in retrospect—at first-order approximation, an ε-sized L_2 *untargeted* adversarial attack increases the loss $\mathcal{L}$ at point x by $\varepsilon\|\partial_x \mathcal{L}(x, c_{org})\|_2$. Analogously, at first-order approximation, an ε-sized L_2 *targeted* adversarial attack decreases the loss $\mathcal{L}$ at point x by $\varepsilon\|\partial_x \mathcal{L}(x, c_{adv})\|_2$ [147].

5.9.2 Controlling Gradient with Gaussian Noise

We can systematically measure the phenomena (described in the previous Section 5.9.1) by adding more Gaussian noise to the original or adversarial images (Figures 5.13, 5.14).

We start the analysis from clean images that are classified correctly. We present how the gradient of the loss w.r.t. the input image changes for the correct class as we add the Gaussian noise to the original image in Figure 5.13. The norm of the gradient smoothly increases. In Figure 5.14, we start from an adversarial image found with the default C&W attack from the foolbox library. Then, we systematically add Gaussian noise to the adversarial image and collect data on the norm of gradients for the original and adversarial classes. The norm of the gradients for the adversarial class increases while the norm of the gradients for the original class decreases. We cross the decision boundary to the correct class very early and recover the correct labels for images. Then, as we add substantially more Gaussian noise, the predictions of the classifier become random and the norms of the gradients converge to a single value. In Figure 5.15, we plot the gradients also for a random class. We observe that for an untargeted attack, the gradients for the original and adversarial classes are larger than for the other classes. The targeted attack decreases the loss for the target class and the gradients for the adversarial classes are lower when compared with gradients from

the untargeted attacks, so fewer images can be recovered in the former case. The targeted attack causes a smaller increase in the norms of gradients for the original class than the untargeted attack. However, it is still higher than for a random class.

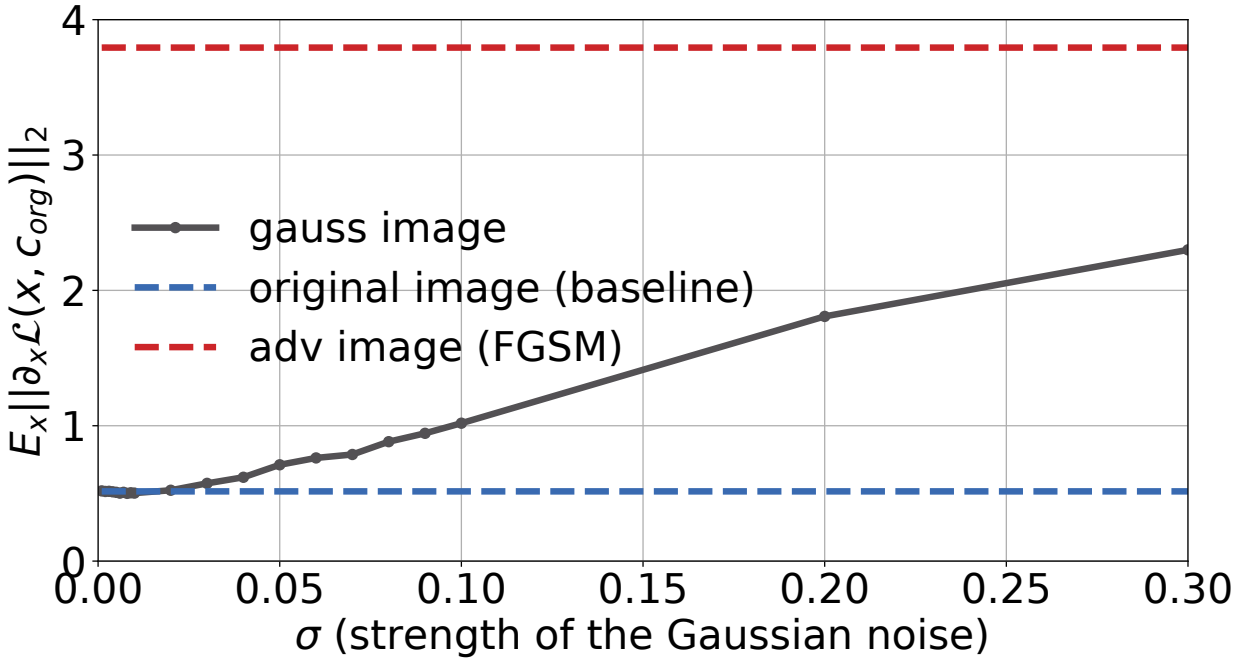

Figure 5.13: The changes in the L_2 norm of the gradient of the loss w.r.t. the input image x for the correct class c_{org} as we add Gaussian noise to the original image. The experiment is run on 1000 images from the ImageNet dataset.

5.9.3 Controlling Gradient with Attack Confidence

The gradient magnitude can be controlled with the confidence parameter in the C&W attack. There is a high dependence between what confidence level is set for the attack and what the gradient norm is. If we set the confidence to 0, then we only intend to misclassify an example, and the gradient norms are very low (also the prediction confidence of the misclassified examples is low). The gradient norm is w.r.t. an adversarial image for its adversarial class. If we set the confidence of the attack to a higher level, then also the gradient is smaller for the adversarial examples and the prediction confidence (for adversarial classes) is increased. The results are not consistent as we would expect the highest prediction confidence for the highest attack confidence. We find

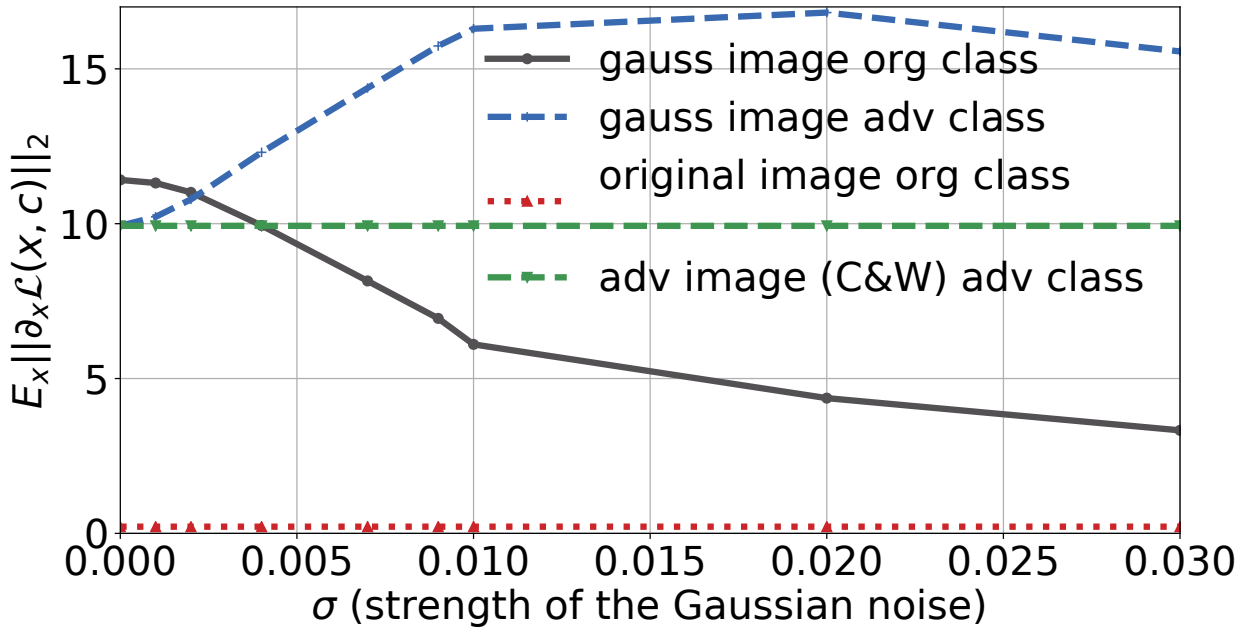

Figure 5.14: The changes in the L_2 norm of the gradient for the correct class c_{org} and the adversarial class c_{adv} as we add Gaussian noise to the adversarial image generated with C&W L_2 attack. The experiment is run on 1000 images from the CIFAR-10 dataset.

that overall the attack confidence for the C&W attack of about 1000 gives us the highest average confidence of predictions (from the model that the given adversarial example is of the adversarial class). However, for the attack confidence level of about 10, we can reach almost 100% prediction confidence for very high strength of the attack. For the consecutive plots in Figure 5.16 we increase the confidence level for the C&W attack. In each plot, we average confidence in predictions across images as we systematically increase the attack strength. For the attack confidence levels 0, 10, 100, we run the experiments on 1000 images from the CIFAR-10 test set. For the attack confidence level 1000, we run the experiment on the whole CIFAR-10 test set.

The softmax probabilities and norms of gradients are correlated for original images but not necessarily for adversarial examples. We run the experiment for 1000 images from the CIFAR-10 dataset. The classification accuracy on the clean data is 93.9%. Next, we take only the correctly classified images and for each image, we generate an adversarial example using the default C&W attack from the foolbox library (where the confidence parameter is set to zero). We record the softmax probabilities and norms of the gradients for each of the 10 classes. For 99% of the original

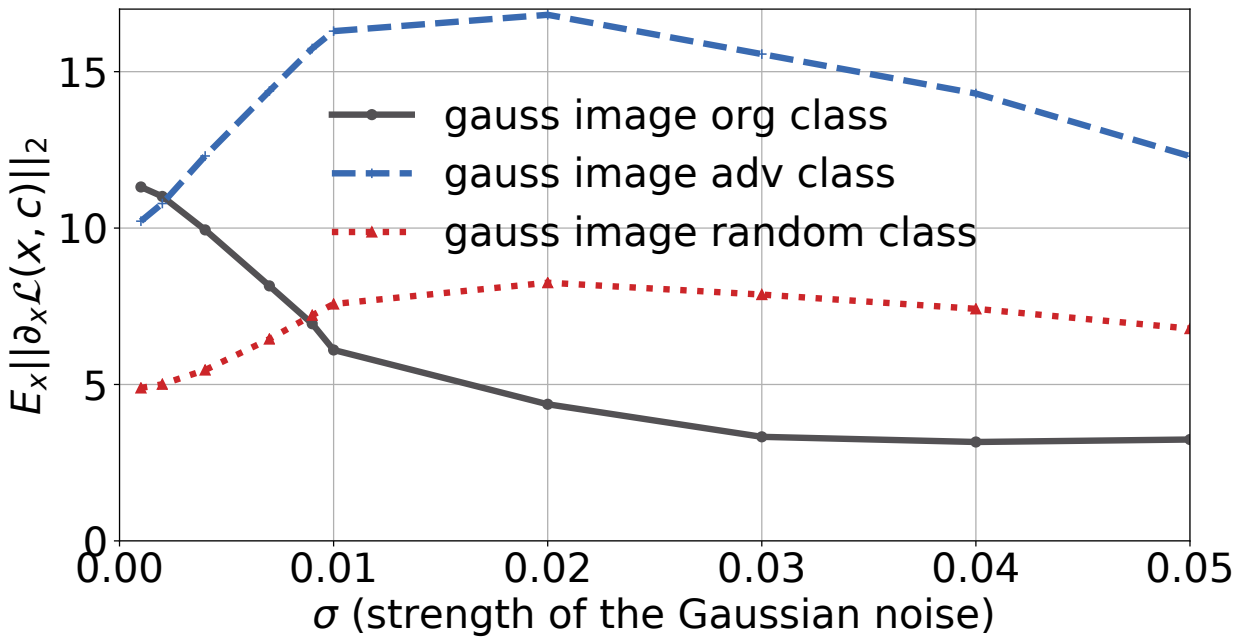

Figure 5.15: The changes in the L_2 norm of the gradient of the loss for the correct class c_{org}, the adversarial class c_{adv}, and a random class c_{ran} as we add Gaussian noise to the adversarial image generated with C&W L_2 attack. The experiment is run on 1000 images from the CIFAR-10 dataset.

images, the lowest gradients are for the original class. Only for 1% of the adversarial examples, the lowest gradients are for the adversarial class.

5.9.4 Hessian-based Analysis

We extend the second-order analysis [180] and compute Hessians with respect to inputs instead of parameters. We present the Hessian spectrum in Figure 5.17. Our experiments show that adversarial examples lead to an order of magnitude higher eigenvalues of the Hessians (on average) than original inputs. This suggests that the model predictions for the adversarial inputs are less stable and random perturbations of the adversarial images with some form of noise can easily change the classification outcome. On the other hand, predictions for the original images are more stable and retain their correct labels when perturbed with a small amount of random noise.

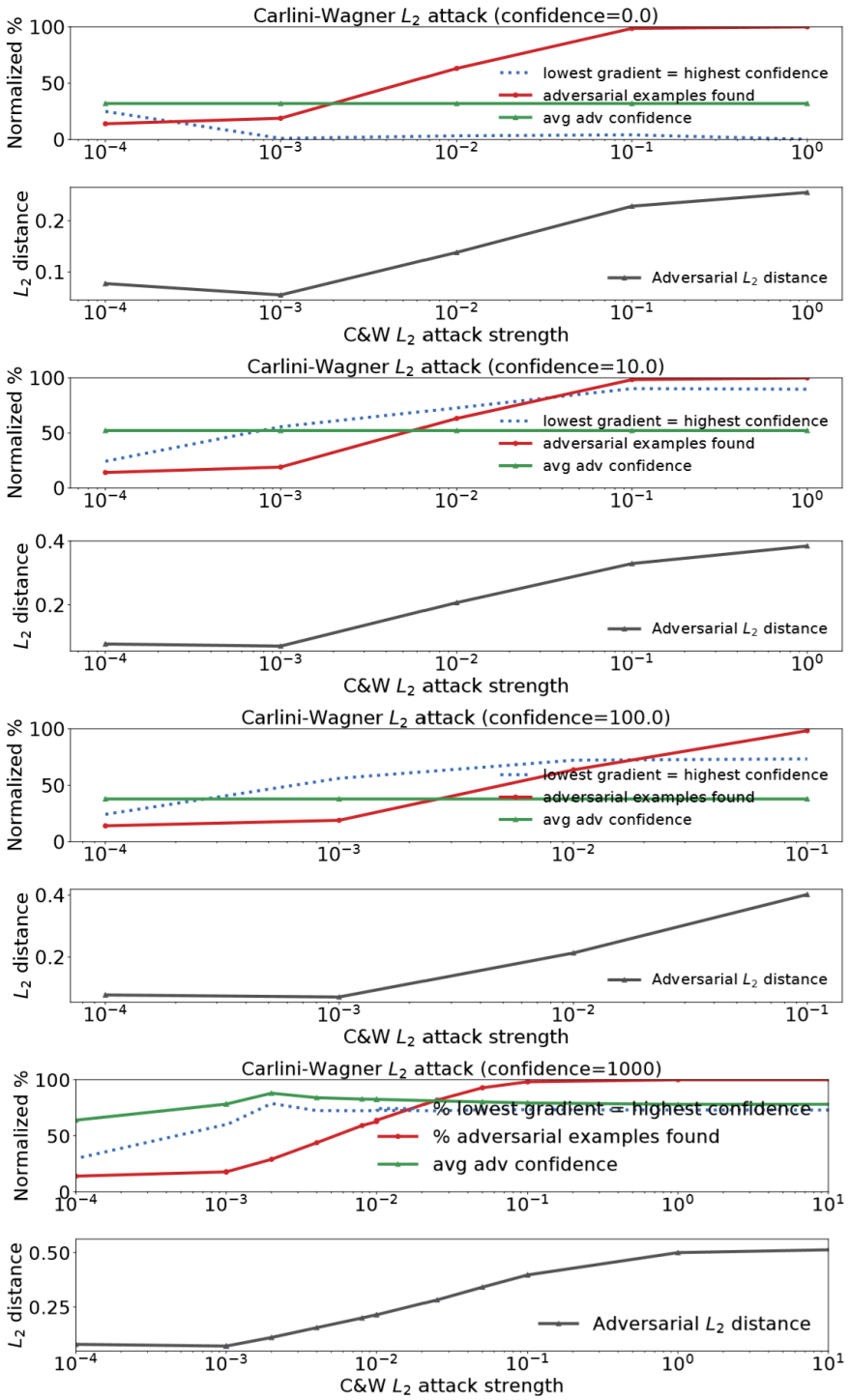

Figure 5.16: Dependence between confidence levels of the CW attack and gradients of the generated adversarial examples.

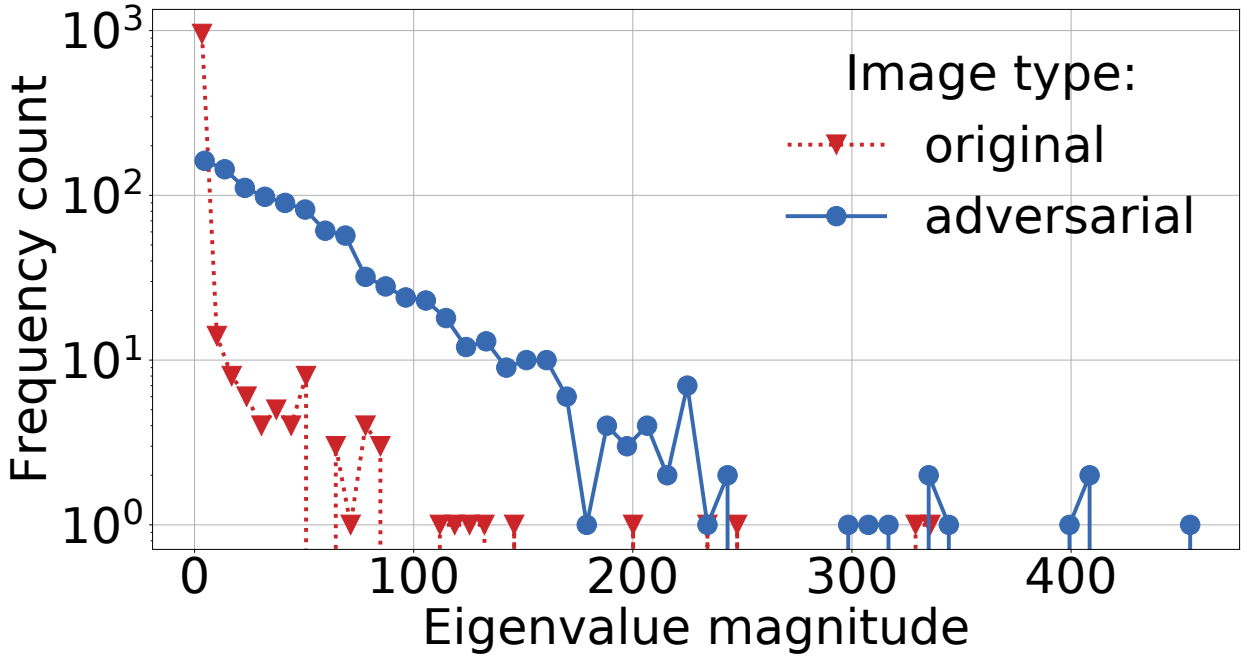

Figure 5.17: Histogram of top eigenvalues of the Hessians w.r.t. the input 1024 images from the CIFAR-10 dataset trained on the ResNet-18 architecture.

5.10 Accuracy of Perturbation Defenses

5.10.1 Accuracy on Clean Data

One pitfall of the input perturbation defenses is that they introduce errors whether or not there are any adversarial examples. The errors act as an upper-bound for the best possible test accuracy we can get under adversarial perturbations.

Table 5.3 shows the results for all test images from CIFAR-10 on the ResNet-18 architecture, for three of the imprecise channels, and for different noise settings.

Table 5.3: Accuracy of perturbation-based defenses on clean data. For the ResNet-18 architecture trained on the CIFAR-10 dataset, we measure the max test accuracy without any adversarial perturbation. This signifies the amount of accuracy we sacrifice with respect to the baseline clean test accuracy 93.56% of the model (without any perturbations of the images).

FC (%)	Acc. (%)
1	93.5
10	93.42
50	91.6
75	79.53

CD (bits)	Acc. (%)
8	93.4
6	93.3
4	91.9
2	87.4

Uniform (ε)	Acc. (%)
0.009	93.52
0.03	92.59
0.07	85.2
0.1	70.67

We present the results in Figure 5.18 for six different noisy channels; three of them are compression based: FC, CD, SVD, and other three add different types of noise: Gauss, Uniform, and Laplace. For each of the compression based approaches, we increase the compression rate systematically from 0 to about 90% (in case of the CD, the compression rate is computed based on how many bits are used per value). For the noise-based approaches, we increase the strength of the noise by controlling the epsilon parameter ε (in case of the Gaussian noise, it corresponds to the standard deviation parameter σ).

The test accuracy of the models can be increased by training with compression, e.g., by using FFT based convolutions with 50% compression in the frequency domain increases the accuracy to 92.32%.

5.10.2 Tunining Channels on Clean and Adversarial Examples

We compare the accuracy of the perturbation channels on clean and adversarial examples in Figure 5.19. We plot the results for the whole test set from CIFAR-10 and the whole validation set from ImageNet. We use two deterministic channels (FFT compression denoted by FC and SVD compression) and two stochastic channels (Gaussian and Uniform noise). We use C&W and PGD attacks. The *FC Clean* label denotes that we pass clean images through the channel that applies FFT compression. The *SVD C&W* label denotes that we pass adversarial images found with the C&W attack through the channel that applies SVD compression. We tune the channels for a given dataset across the attacks and present the results in Table 5.4. The accuracy of the channels on clean data gives us the upper bound for the accuracy on the adversarial examples. Thus, we choose channel parameters based on the highest accuracy on the adversarial images.

Table 5.4: **Channel tuning**. The best parameters for the perturbation channels when tuned on the PGD and C&W attacks.

Channel \ *Dataset*	CIFAR-10	ImageNet
FC (%)	20	60
SVD (%)	40	70
Gauss (ε)	0.015	0.04
Uniform (ε)	0.025	0.04

5.11 Regularization

Perturbing the inputs and internal feature maps during training is equivalent to a form of Lipschitz regularization. However, defenses studied in this thesis add noise also during inference and are not standard forms of regularization (e.g. Tikhonov or Lasso). We do show that training the models with the anticipated noise (that is added as a defense at inference time) is a valuable optimization but we do not believe that this is fully a regularization effect. The techniques are also akin to data augmentation. Furthermore, defenses such as feature squeezing are non-differentiable and do not

fit into a standard regularization framework. Finally, regularization methods have to be designed and tuned specifically to create a strong defense.

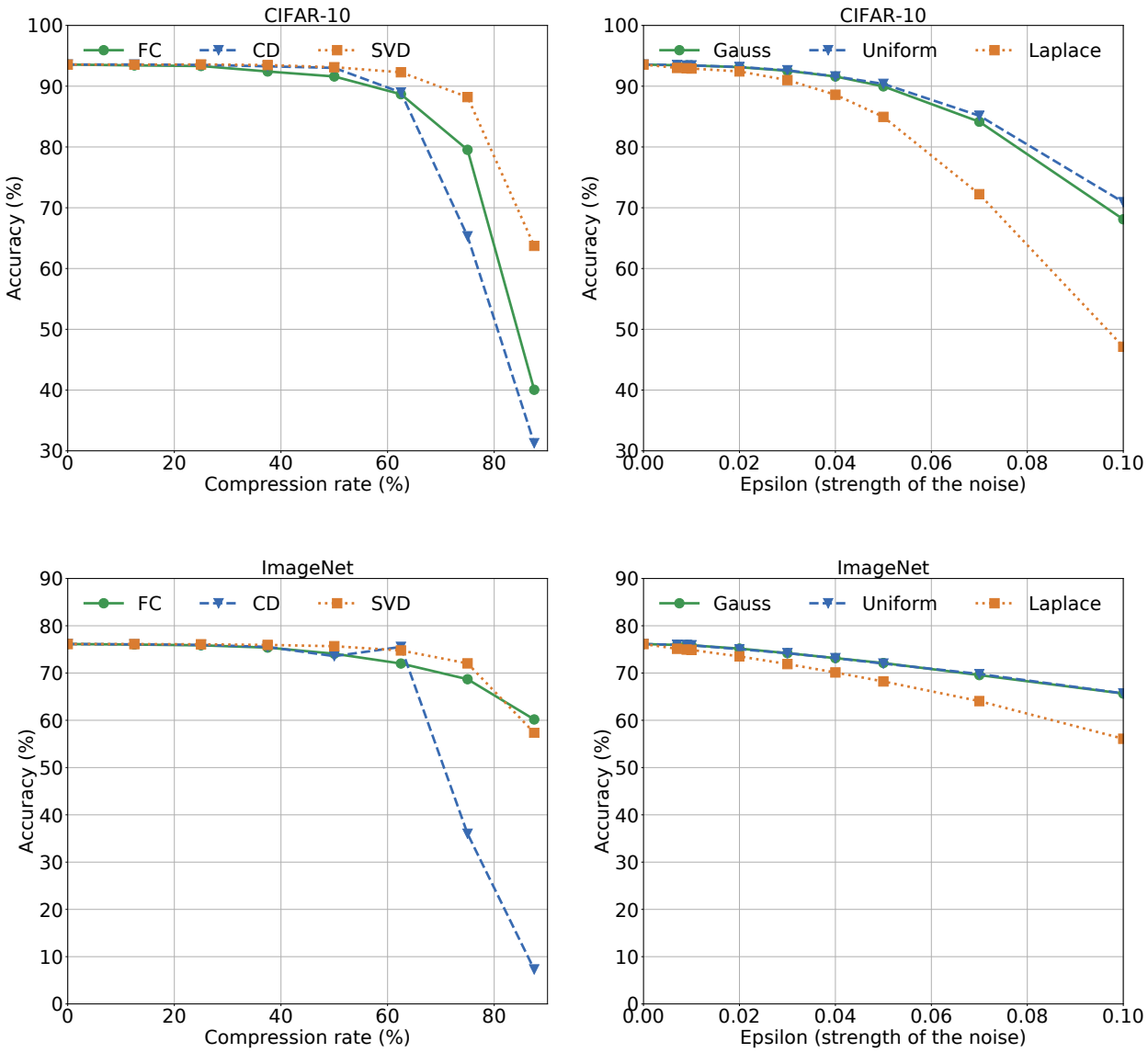

Figure 5.18: The test accuracy after passing clean images through six different input perturbation defenses, where the added noise is controlled by the compression rate and epsilon parameters. We use full CIFAR-10 test set for ResNet-18, and full ImageNet validation set for ResNet-50.

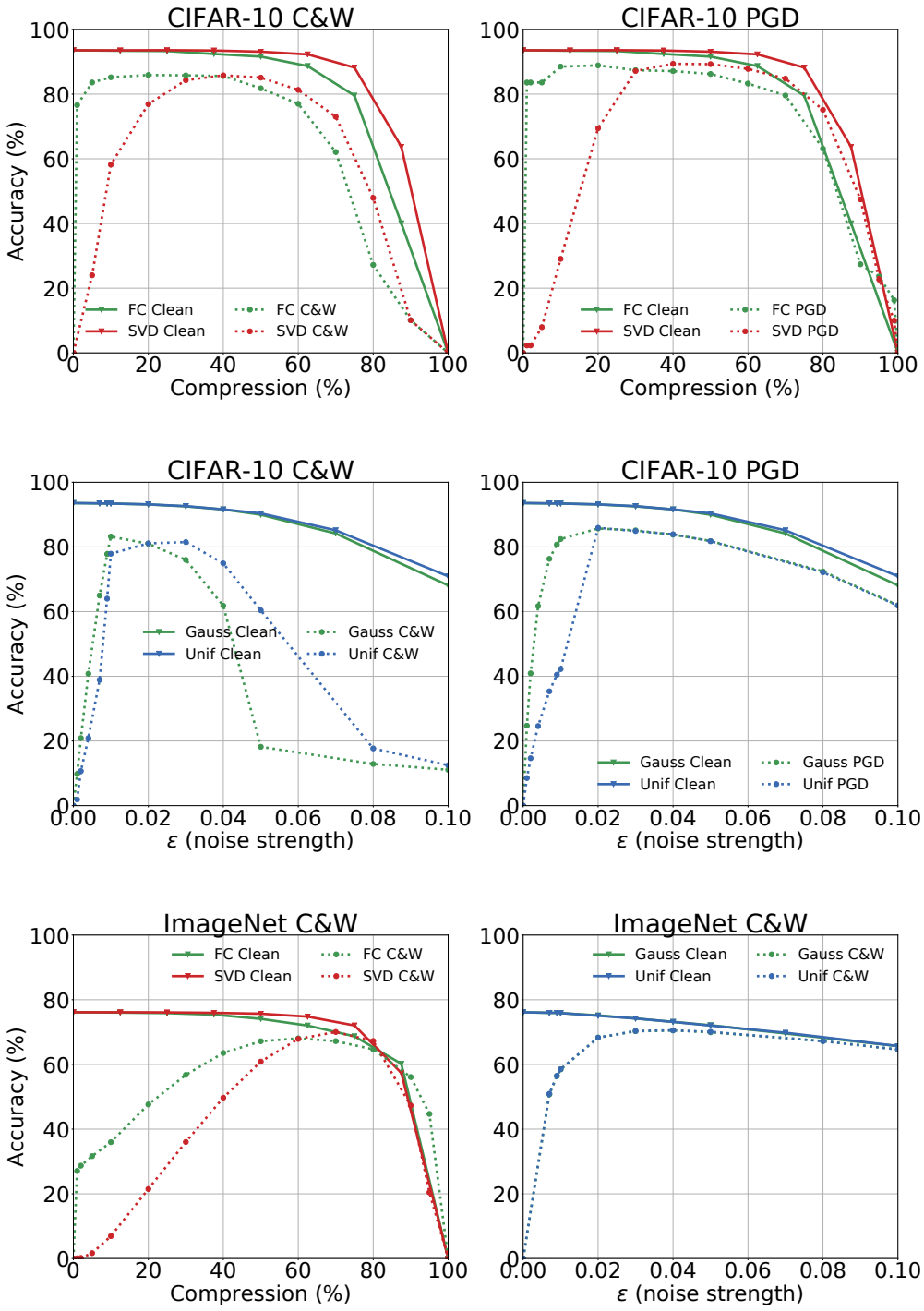

Figure 5.19: Comparison of accuracy of the perturbation channels on clean and adversarial images. We pass either clean or adversarial images through the channels and measure their accuracy. The accuracy on clean inputs gives us an upper bound for the accuracy on the adversarial examples. The channels can be tuned based on their accuracy after different attacks.

5.12 Out-Of-Distribution Robustness

We illustrate applications to out-of-distribution (OOD) robustness of models in terms of their generalization and detection of anomalous inputs in the NLP (Natural Language Processing) domain.

Models should generalize to OOD examples when it is possible, for example, there is a small shift in the data distribution. Moreover, when OOD examples do not belong to any known label, then models should mark these examples as anomalous. For generalization part (shown in Figure 5.20 in the green box), we train models on IMDb lay movie reviews [107] and test the models on SST-2 expert movie reviews [153]. Our goal is to mimic a realistic data distribution shift from lay to expert movie reviews. The sentiment of expert reviews should be classified correctly as either positive or negative by models trained on lay reviews.

For the detection part (shown in Figure 5.20 in the red box), we train the models on the IMDb lay movie reviews and test them on data that contains no reviews or any expression of sentiment. Models should detect that these examples do not belong to any known class. For example, we train the models on IMDb lay movie reviews but test them on datasets that contain some inference task, which is unrelated to movie reviews. We use textual entailment datasets SNLI [15] and RTE [36] as the representative out-of-distribution data. The SNLI dataset is a collection of 570k human-written English sentence pairs manually labeled for balanced classification with the labels entailment, contradiction, and neutral. For the RTE (Recognising Textual Entailment) dataset [5], we use its GLUE version [170]. It contains 5770 sentence pairs that are labeled as either *entailment* or *not_entailment*.

We carry out our experiments on many NLP models. We use word2vec [118] word embeddings which are encoded with the following models: word averages [174], LSTMs [78], and Convolutional Neural Networks (here denoted as ConvNet). We pass the representation from the encoders to MLP models. We also test pretrained bidirectional Transformers [169], in particular, the BERT-based models [42] in their Base and Large versions. Additionally, we experiment with

5. RTE dataset: `https://bit.ly/2YULSOq`

RoBERTa [104], which is a pretrained transformed trained on a larger dataset than BERT, and a distilled version of BERT, DistilBERT [138].

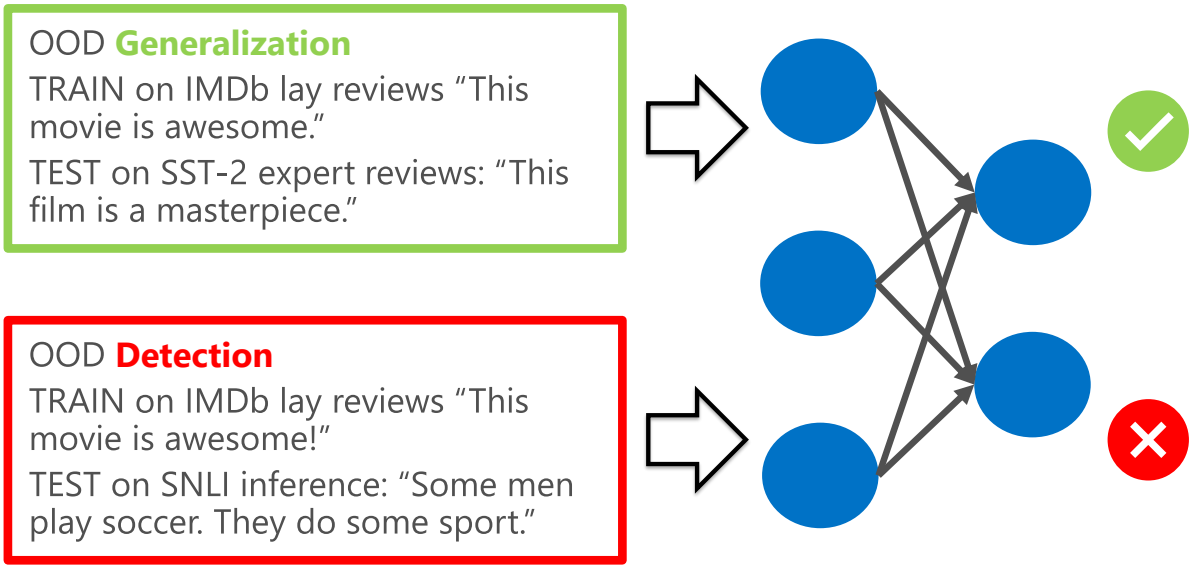

Figure 5.20: Out-Of-Distribution robustness consists of robust generalization and anomalous detection.

5.12.1 Robust Generalization

First, we present results for robust generalization in Figure 5.21. In this experiment, we train models on the IMDb lay movie reviews and test the models on SST-2 expert movie reviews. The x axis represents different NLP models, and the y axis is test accuracy. The IID (in-distribution) accuracy shown in blue does not vary much, however, the out-of-distribution (OOD) accuracy (shown in red) varies significantly. The drop in accuracy for previous common NLP models such as LSTMs is substantial. The generalization gap for LSTMs is even 27%. On the other hand, we find that the pre-trained transformers generalize much better to out-of-distribution data and achieve much higher accuracy. The generalization gap is only about 5% for RoBERTa.

BERT Base contains 108 million parameters and its pre-training takes 4 days on 4 cloud TPUs, while BERT Large contains 334 million parameters and is trained for 4 days as well but on 16 cloud TPUs. There are many proposals on how to compress BERT. We compare DistilBERT,

ALBERT, and MobileBERT models. DistilBERT [138] uses the standard distillation technique, where a student model is trained on the full output distribution of a teacher model. ALBERT [96] uses two-parameter reduction techniques. First, it repeats layers, which is a form of parameter sharing. Second, it decouples the size of the word embedding and hidden vectors, which makes the vocabulary embedding matrix much smaller. MobileBERT [157] is as deep as BERT but makes each of its layers much narrower. Compressing the BERT models does not hurt the in-distribution accuracy shown as blue columns in Figure 5.22. However, compression can substantially increase the generalization gap even up to 16% for MobileBERT. This is an example where compression does not help. In general, we find that smaller or compressed pre-trained transformers perform worse than the biggest and uncompressed transformers, such as RoBERTa Large.

We present detailed experimental results for the comparison between standard transformers and their compressed counterparts in Table 5.5. The values are accuracies for different models and datasets. The I-IMDb denotes that IMDb is the in-distribution data, O-SST-2 means that SST-2 is the out-of-distribution data, Diff-IS is the difference between in-distribution IMDb accuracy and the out-of-distribution accuracy for SST-2. Similarly, I-SST-2 denotes that SST-2 is the in-distribution data, O-IMDb means that IMDb is the out-of-distribution data, Diff-SI is the difference between in-distribution SST-2 accuracy and the out-of-distribution accuracy for IMDb. The compressed models trained on IMDb and tested on SST-2 show up to an 18% generalization gap. The compressed models do not substantially increase the generalization gap when trained on the SST-2 dataset and tested on the IMDb dataset.

5.12.2 Anomalous Detection

The second part of out-of-distribution robustness is detection. In this experiment presented in Figure 5.23, we measure how good the models are for out-of-distribution detection when we train the models on SST-2 dataset, which contains lay movie reviews, and test the models on data with no reviews that contains pairs of sentences for inference tasks from SNLI and RTE datasets.

We tune the models to detect 95% of out-of-distribution examples and check the False Alarm

Table 5.5: Comparison between the standard uncompressed pretrained transformers (BERT, RoBERTa, XLNet) and their compressed counterparts (MobileBERT, DistilBERT, ALBERT, DistilRoBERTa).

Model	**I-IMDb**	**O-SST-2**	**Diff-IS**	**I-SST-2**	**O-IMDb**	**Diff-SI**
bert-large-uncased	0.94796	0.886468	0.061492	0.933486	0.90268	0.030806
bert-base-uncased	0.93848	0.875	0.06348	0.913991	0.89464	0.019351
mobile-bert	0.93284	0.771789	0.161051	0.912844	0.89988	0.012964
distil-bert-base	0.93084	0.868119	0.062721	0.90367	0.89076	0.01291
albert-base-v1	0.92024	0.832569	0.087671	0.885321	0.8758	0.009521
albert-large-v1	0.90164	0.813073	0.088567	0.509174	0.5	0.009174
albert-xlarge-v1	0.93808	0.852064	0.086016	0.5	0.5	0
albert-base-v2	0.9428	0.895642	0.047158	0.908257	0.88468	0.023577
albert-large-v2	0.83584	0.720183	0.115657	0.517202	0.53824	-0.02104
albert-xxlarge-v2	0.9004	0.720183	0.180217	0.509174	0.5	0.009174
distil-roberta-base	0.93752	0.876147	0.061373	0.923165	0.9092	0.013965
roberta-base	0.95244	0.881881	0.070559	0.940367	0.91252	0.027847
roberta-large	0.96096	0.90711	0.05385	0.509174	0.5	0.009174
xlnet-base-cased	0.95064	0.883028	0.067612	0.940367	0.89384	0.046527
xlnet-large-cased	0.9594	0.889908	0.069492	0.509174	0.5	0.009174
min	0.83584	0.720183	0.047158	0.5	0.5	0
max	0.96096	0.90711	0.180217	0.940367	0.91252	0.046527
average	0.930005	0.844878	0.085128	0.754358	0.740149	0.014208
median	0.93808	0.875	0.069492	0.90367	0.88468	0.01291

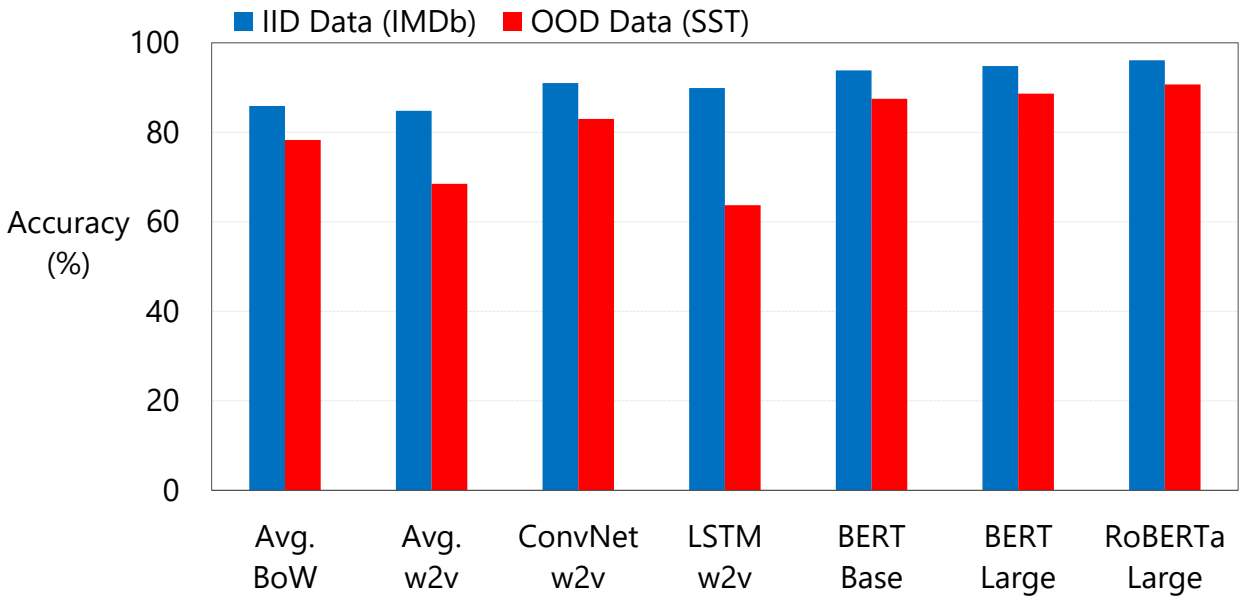

Figure 5.21: ***Robust generalization. Comparison between preceding standard NLP models (BoW, word2vec, ConvNets, LSTMs) and pretrained transformers (BERT Base, BERT Large, RoBERTa Large).*** Sentiment binary classification for movie reviews. Models trained on IMDb and tested on SST-2.

Rate (FAR) – the percentage of in-distribution examples that are detected as out-of-distribution. The x axis represents different NLP models, and the y axis is FAR, where lower values denote better anomalous detection. Previous models, such as LSTMs, struggle at out-of-distribution detection and they often assign a higher probability to out-of-distribution examples than to in-distribution examples. In contrast, pretrained transformers exhibit much better performance in out-of-distribution detection.

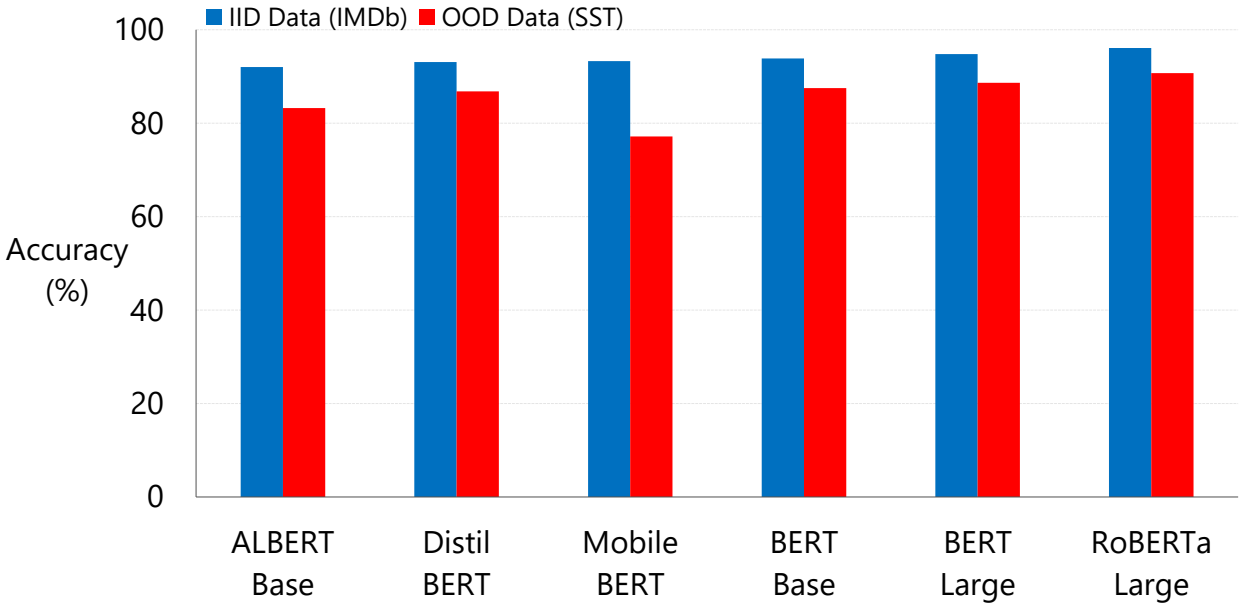

Figure 5.22: ***Robust generalization. Comparison between compressed pretrained transformers (ALBERT, DistilBERT, and MobileBERT) and the standard uncompressed pretrained transformers (BERT Base, BERT Large, RoBERTa Large).*** Sentiment binary classification for movie reviews. Models trained on IMDb and tested on SST-2.

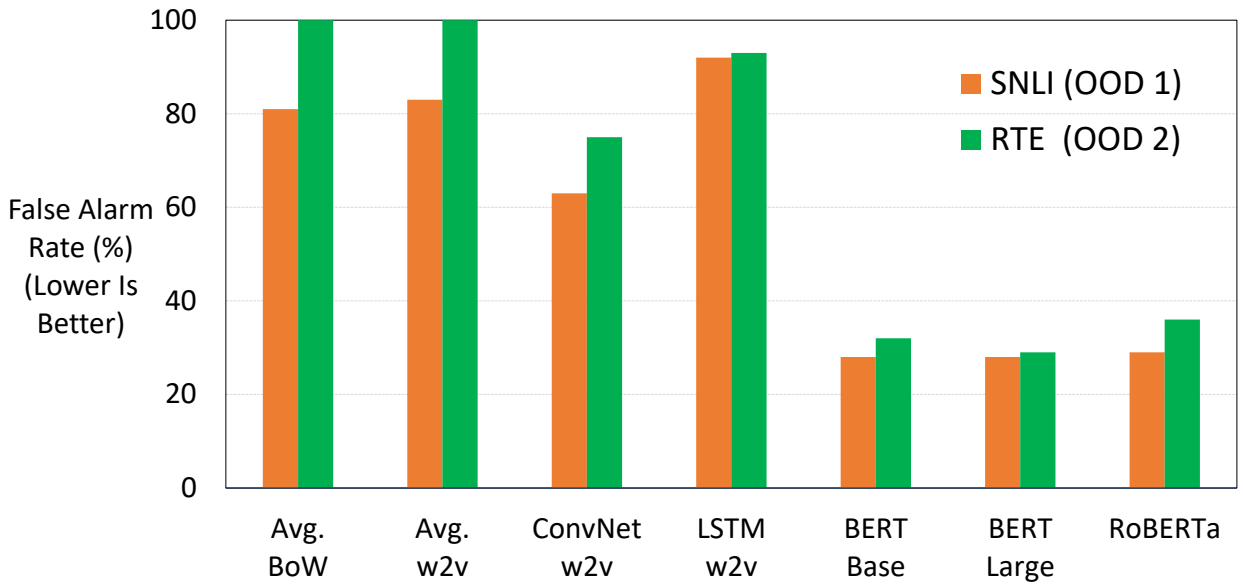

Figure 5.23: ***Anomalous detection.*** Sentiment binary classifier trained on SST-2 *movie reviews*, detect 95% OOD vs IID False Alarm Rate (%) on SNLI and RTE *entailment* datasets.

5.13 Conclusions

The non-adaptive attacks are not robust since small changes to adversarial inputs often recover the correct labels. This is an obvious corollary to the very existence of adversarial examples that by definition are relatively close to correctly predicted examples in the input space. Random perturbations of inputs can dominate the strategically placed perturbations synthesized by an attack. The results are consistent across both deterministic and stochastic channels that degrade the fidelity of input examples. From the perspective of the attacker, the recovery window can be closed to make the perturbation based techniques ineffective. Moreover, a strong input perturbation defense can be assumed a priori to achieve high transferability of the attacks. The most effective defenses perturb not only inputs but also the internal layers. The scale of the noise can be set as a parameter. We find that the combination of layer perturbations and adversarial training improves robustness against adaptive CW and PGD attacks in comparison to pure adversarial training or parameter noise injection with adversarial training. The training of networks with strong perturbation defenses (high level of added noise) is essential to recover the test accuracy on clean data. Our first and second-order analyses show that the perturbation based defenses are possible since the adversarial examples are not robust themselves.

CHAPTER 6

APPLICATIONS TO LTE-U AND WI-FI FAIR COEXISTENCE

The application of Machine Learning (ML) and Deep Learning (DL) techniques [1] to complex problems has proved to be an attractive and efficient solution [14, 94]. The ML techniques effectively replace heuristic-based approaches. The promise of ML techniques in solving non-linear problems influenced this part of the thesis that aims to apply the ML techniques, especially the band-limited models (described in Chapter 4), and develop new solutions for the unlicensed wireless spectrum sharing between Wi-Fi APs (Access Points) and LTE-U BSs (Base Stations).

LTE-U is part of the LTE evolution track on the route to 5G (the 5th generation mobile network). It augments a carrier's LTE service by utilizing the unlicensed spectrum in the 5 GHz range. The unlicensed spectrum significantly increases network capacity and coverage, especially in crowded public indoor or outdoor areas. However, LTE-U was not designed for a shared environment and the unlicensed spectrum is already occupied by many Wi-Fi devices. On the one hand, LTE-U can leverage Wi-Fi signals to estimate the number of Wi-Fi devices. This method is fast and resource-efficient but usually based on manually set thresholds and far from the desired accuracy. On the other hand, the LTE-U base stations could try to decode Wi-Fi packets, however, this approach is more expensive, slower, and requires establishing another communication standard that every party should adhere to. In this part of the thesis, we develop an ML technique to tackle the problem.

We focus on the LTE-Unlicensed (LTE-U) specification developed by the LTE-U Forum, which uses the duty-cycle approach for **fair coexistence**. The specification suggests reducing the duty cycle at the LTE-U BS when the number of co-channel Wi-Fi APs increases from one to two or more. However, without decoding the Wi-Fi packets, detecting the number of Wi-Fi APs operating on the channel in real-time is a challenging problem. We demonstrate a novel ML-based and band-limited approaches that solve this problem by using energy values observed during the LTE-U OFF duration. It is relatively straightforward to observe only the energy values during the LTE-U

1. They are commonly referred to as ML techniques.

BS OFF time compared to decoding the entire Wi-Fi packet, which would require a full Wi-Fi receiver at the LTE-U BS. We implement and validate the proposed ML-based and band-limited approaches by real-time experiments and demonstrate that there exist distinct patterns between the energy distributions between one and many Wi-Fi AP transmissions. The proposed approaches result in higher accuracy (close to 99% in all cases) as compared to the existing auto-correlation (AC) and energy detection (ED) approaches.

6.1 Introduction

The growing penetration of high-end consumer devices like smartphones and tablets running bandwidth-hungry applications (e.g., mobile multimedia streaming) has led to a commensurate surge in demand for mobile data (pegged to soar up to 77 exabytes by 2022 [32]). An anticipated second wave will result from the emerging Augmented/Virtual Reality (AR/VR) industry [4] and more broadly, the Internet-of-Things that will connect an unprecedented number of intelligent devices to next-generation (5G and beyond) mobile networks as shown in Figure 6.1. Existing wireless networks, both cellular and Wi-Fi, must therefore greatly expand their aggregate *network* capacity to meet this challenge. This is being achieved by a combination of approaches including the use of multi-input, multi-output (MIMO) techniques [63], network densification (i.e. deploying small cells [143]) and more efficient traffic management and radio resource allocation.

Since the licensed spectrum is a limited and expensive resource, its optimal utilization may require spectrum sharing between multiple network operators/providers of different types. Licensed-unlicensed sharing is being contemplated to enhance network spectral efficiency. As the most common unlicensed incumbent, Wi-Fi is now broadly deployed in the unlicensed 5 GHz band in North America where approximately 500 MHz of bandwidth is available. However, these 5 GHz unlicensed bands are also seeing the increasing deployment of cellular services such as Long Term Evolution (LTE) Licensed Assisted Access (LTE-LAA). Recently, the Federal Communications Commission (FCC) sought to open up 1.2 GHz of additional spectrum for unlicensed operation in the 6 GHz band through a Notice of Proposed Rule Making (NPRM) [34]. This allocation of

spectrum for the unlicensed operation will thus only accelerate the need for further coexistence solutions among heterogeneous systems.

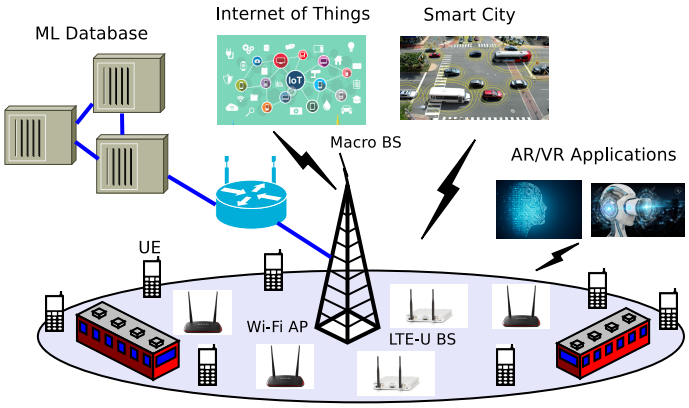

Figure 6.1: Future Applications on Unlicensed Spectrum Band.

However, the benefits of spectrum sharing are not devoid of challenges, the foremost being the search for effective coexistence solutions between cellular (LTE and 5G) and Wi-Fi networks whose medium access control (MAC) protocols are very different. While cellular systems employ a Time Division Multiple Access (TDMA) [2] or Frequency Division Multiple Access (FDMA) [3] scheduling mechanism, Wi-Fi depends on the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) mechanism. The Carrier Sense (CS) means that the devices can distinguish between an idle and a busy link, while the Multiple Access (MA) means that a set of devices sends and receives frames over a shared link (the data link layer in the OSI model [123]). The Ethernet (802.3) and Multiple Access Networks (MANs) use CD (Collision Detection) instead of CA (Collision Avoidance). Collision Detection (CD) means that a device transmits and listens at the same time, hence it can detect when a frame it transmits interferes with a frame transmitted by another device. Thus, in CD, a device waits until the transmission link becomes idle to start its transmission and exponentially back-offs when a collision occurs. The CA (Collision Avoidance) was designed for wireless communication where, as opposed to Ethernet (which is full-duplex),

2. TDMA provides different time slots to different transmitters.

3. In FDMA, different frequency bands are allocated to different devices in a cyclically repetitive frame structure. For example, node 1 may use time slot 1, node 2 may use time slot 2, etc. until the last transmitter when it starts over.

Wi-Fi devices cannot transmit and listen at the same time (half-duplex) nor any device receives every other device's transmission. The Wi-Fi devices cannot transmit and listen simultaneously because the power of the transmitted signal is much higher than the power of any received signal [123]. A device might not receive signals from all other devices since they might be too far away from each other or blocked by an obstacle. CA (Collision Avoidance) adds two additional mechanisms to CD (Collision Detection). First, if a packet is successfully decoded at the receiver, then the receiver sends an acknowledgment back to the sender. Second, an optional mechanism called Ready to Send-Clean to Send (RTS-CTS), in which a sender sends an RTS short packet to its intended receiver, and if the packet is received successfully by the receiver, then the receiver sends the short CTS packet to the sender (we elaborate more on the CSMA/CA in Section 6.3.1).

The 5 GHz band being unlicensed and offering 500 MHz of available bandwidth has prompted several key players in the cellular industry to develop the LTE-LAA specification within the Third Generation Partnership Project (3GPP). Specification differences between LTE and the incumbent Wi-Fi will lead to many issues due to the incompatibility between the two standards. Therefore, to ensure fair coexistence, certain medium access protocols have been developed as an addition to the licensed LTE standard. In addition to LTE-LAA, there also exists LTE-U that was developed by an industry consortium called the LTE-U Forum [58].

LTE-LAA was proposed by 3GPP [116, 126] and its working mechanism is similar to the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol used by Wi-Fi. In LTE-LAA, an LAA base station (BS) acts essentially similar to a Wi-Fi access point (AP) in terms of channel access, *i.e.*, a BS needs to ensure that the channel is free before transmitting any data, otherwise, it will perform an exponential back-off [4] procedure similar to CSMA/CA in Wi-Fi. Therefore, there is no need to precisely determine the number of coexisting Wi-Fi APs, due to the channel sensing and back-off mechanism which is adaptable to varying channel occupancy. However, LTE-U which was developed by the LTE-U Forum [58], uses a simple duty-cycling technique where the LTE-U BS periodically switches between ON and OFF states in an interval

4. The exponential back-off is a strategy of doubling the delay interval between each transmission attempt.

set according to the number of Wi-Fi APs present in the channel. In the ON state, the BS transmits data as a normal LTE transmission while in the OFF state, the BS does not transmit any data but passively senses the channel for the presence of Wi-Fi. The number of sensed Wi-Fi APs is then used to properly adjust the duty cycle interval, and this process is known as Carrier Sense Adaptive Transmission (CSAT). Therefore, accurately determining the number of coexisting Wi-Fi APs is important for optimum operation of the CSAT procedure.

Existing literature addresses the LTE-U and Wi-Fi coexistence in terms of optimizing the ON and OFF duty cycle [150], power control [24], hidden node problem [7], etc. On the other hand, the LTE-U specification does not specify, and there has been relatively less work on, how an LTE-U operator should detect the number of Wi-Fi APs on the channel to adjust the duty cycle appropriately. There are several candidate techniques to determine the number of Wi-Fi APs as follows:

- **Header-Based CSAT (HD):** Wi-Fi APs transmit beacon packets every 102.4 ms, containing important information about the AP, such as the Basic Service Set Identification (BSSID) which is unique to each AP. This is a straightforward way to identify the Wi-Fi AP, but it adds additional complexity since the LTE-U BS would require a full Wi-Fi decoder to obtain this information from the packet.

- **Energy-Based CSAT (ED):** Rather than a full decoding process, it is hypothesized that sensing the energy level of the channel is enough to detect the number of Wi-Fi APs on the channel. However, it is still a challenging problem since the energy level may not correctly correlate to the number of APs under varying conditions (*e.g.*, a different category of traffic, a large number of Wi-Fi APs, variations in transmission powers, multipath, etc).

- **Autocorrelation-Based CSAT (AC):** To detect the Wi-Fi signal at the LTE-U BS, one can develop an auto-correlation (AC) based detector where the LTE-U BS performs auto-correlation on the Wi-Fi preamble, without fully decoding the packet. This is possible since

all Wi-Fi preambles [5] contain the legacy short training field (L-STF) and legacy long training field (L-LTF) symbols which contain multiple repeats of a known sequence. However, the AC function can only determine whether a signal is a Wi-Fi signal and cannot derive any distinct information pertaining to each Wi-Fi AP.

Table 6.1 lists the different types of CSAT approaches with their pros and cons. The energy detection (ED) and auto-correlation (AC) based detection methods of Wi-Fi APs were studied in [142] and [141]. [6] Of late, Machine Learning (ML) approaches are beginning to be used in wireless networks to solve problems such as agile management of network resources using real-time analytics based on data. The advantage of ML is that it can learn useful information from input data, which can help improve network performance. ML models enable us to replace heuristics with more robust and general alternatives. **We observe the Wi-Fi AP energy values during LTE-U OFF duration and use the data to train different ML models [183], especially neural networks with band-limited convolution.** We also apply the models in an online experiment to detect the number of Wi-Fi APs. Finally, we demonstrate significant improvement in the performance of our new approach as compared to the ED and AC detectors.

Figure 6.2 illustrates an example of a dense LTE-U/Wi-Fi coexistence, where several Wi-Fi APs and one LTE-U BS are operating on the same channel, with multiple clients associated with each AP and BS. In such a situation, LTE-U must reduce its duty-cycle proportional to the number of Wi-Fi APs, else with a duty-cycle of 50% the Wi-Fi APs will be starved of air-time. As the number of Wi-Fi APs increase on the channel, it becomes increasingly important to detect the number of APs accurately at the LTE-U BS without any co-ordination *i.e.,* in a distributed manner. According to the LTE-U forum, it is expected that the LTE-U BS will adjust its duty cycle when one or more Wi-Fi APs are turned off, and vice versa. **With a large number of Wi-Fi APs, it is very difficult to detect the number accurately using either energy-based or correlation-based approaches. We infer the presence of one or more Wi-Fi APs accurately from the collected**

5. All Wi-Fi frames, even those in newer specifications like 802.11ax, begin with the legacy short training field (L-STF) symbol.

6. The latest version can be found here: http://bit.ly/2LDVWWo

Table 6.1: Different Types of LTE-U CSAT.

CSAT Types	Method	Pros	Cons
Header Decoding (HD)	Decodes the Wi-Fi MAC header at the LTE-U BS	100% accurate	Additional Complexity [23], high cost
Energy Detection (ED)	Based on the change in the *energy level* of the air medium	Low-cost, low-complexity	Low-accuracy [142]
Auto-correlation (AC)	LTE-U BS performs correlation on the Wi-Fi L-STF symbol in the preamble	Low-cost, low-complexity	Medium accuracy (more accurate than ED) [141]
Machine Learning (ML)	Train the model based on energy values on the channel	Much more accurate than ED and AC methods	Requires gathering data and training models

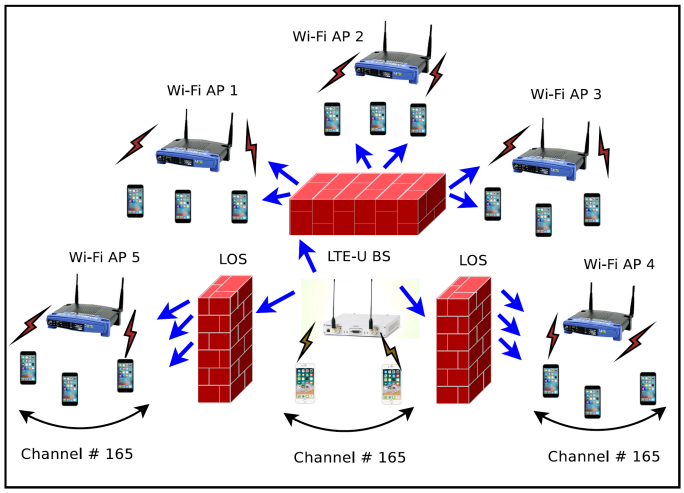

Figure 6.2: Dense LTE Wi-Fi Co-existence Deployment Setup.

energy level data using ML algorithms that have been trained on real data. We accomplish this by creating a realistic open lab experimental scenarios using a National Instruments (NI) USRP RIO board with an LTE-U module, five Netgear Wi-Fi APs, and five Wi-Fi clients.

6.2 Related Work

In this section, we briefly discuss (1) the existing work on LTE Wi-Fi coexistence without ML, (2) the use of ML in general wireless networks, and (3) the application of ML to LTE Wi-Fi coexistence.

6.2.1 Existing Work On LTE and Wi-Fi Coexistence

There has been a significant amount of research, from both academia and industry, on the co-existence of LTE and Wi-Fi that discuss several key challenges such as Wi-Fi client association, interference management, fair coexistence, resource allocation, carrier sensing, etc. Coexistence scenarios are well studied in simulations for both LAA/Wi-Fi and LTE-U/Wi-Fi deployments [20, 23, 26], where coexistence fairness in varying combinations of detection threshold and duty-cycle is examined.

The energy-based CSAT for duty cycle adaptation in LTE-U was proposed in [139, 140, 142], and studied via rigorous theoretical and experimental analyses. The energy-based CSAT algorithm can infer the number of coexisting Wi-Fi APs by detecting the energy level in the channel, which is then used to adjust the duty cycle accordingly. Using a threshold of -42 dBm, the algorithm can differentiate between one or two Wi-Fi APs, with a successful detection probability P_D of greater than 80% and false-positive probability P_{FA} of less than 5%. Hence, this initial work proved the feasibility of stand-alone energy-based detection, without the need for packet decoding.

The algorithm that utilizes auto-correlation function (AC) was proposed in [141] to infer the number of active Wi-Fi APs operating in a channel. The AC function is performed on the preamble of a signal to determine if the signal is a Wi-Fi signal. This work further improved the performance of the energy-based approach, with P_D of 90% and P_{FA} of less than 2%, when using an AC threshold N_E of 0.8. In both [141, 142], the maximum number of Wi-Fi APs considered on the channel was two. In realistic dense deployment scenarios, we can expect more than 2 APs on the same channel. We study the performance of ED and AC for more realistic dense deployment scenarios.

6.2.2 ML Applied to Wireless Networks

In [156], several state-of-the-art applications of ML in wireless communication and unresolved problems have been described. Resource management in the MAC layer, networking, mobility management in the network layer and localization in the application layer are some topics that have been identified as being suitable for ML approaches. Within each of these topics, the authors provide a survey of the diverse ML-based approaches that have been proposed. In [25, 182], a comprehensive tutorial has been provided on the use of artificial neural networks-based machine learning for enabling a variety of applications in wireless networks. In particular, the authors presented an overview of several key types of neural networks such as recurrent and deep neural networks. For each type, the basic architecture, as well as the associated challenges and opportunities, have been presented, followed by an overview of the variety of wireless communication problems that can be addressed using artificial neural networks (ANNs). This work further investigated many emerging applications including unmanned aerial vehicles, wireless virtual reality, mobile edge caching and computing, Internet of Things, and multi-Random Access Technology (RAT) wireless networks. For each application, the authors provide the main motivation for using ANNs along with their associated challenges while also providing a detailed example for a use case scenario.

6.2.3 ML Applied to LTE Wi-Fi Coexistence

A learning-based coexistence mechanism for LTE unlicensed based heterogeneous networks (HetNets) was presented in [160]. The motivation was to maximize the normalized throughput of the unlicensed band while guaranteeing the Quality of Service (QoS) for users: the authors thus considered the joint resource allocation and network access problem. A two-level framework was developed to decompose the problem into two subproblems, which were then solved using learning-based approaches. The outcome of the proposed solution achieves near-optimal performance and is more efficient and adaptive due to the distributed and learning-based approach. Authors in [11] provide an overview of learning schemes that enable efficient spectrum sharing using a generic

cognitive radio setting as well as LTE and Wi-Fi coexistence scenarios. Most LTE-U duty cycle solutions rely on static coexistence parameter configurations, which may not be applicable in real-life scenarios which are dynamic. Hence in [40], the author uses the Markov decision process modeling along with a solution based on an ML CSAT algorithm that adapts the LTE duty-cycle ratio to the transmitted data rate, intending to maximize the Wi-Fi and LTE-U aggregated throughput. An ML-based approach was proposed in [128] for a model-free decision-making implementation of the opportunistic coexistence of LTE-U with Wi-Fi, which enabled the LTE-U BS to dynamically identify and further exploit white spaces in the Wi-Fi channel, without requiring detailed knowledge of the Wi-Fi system. By adaptively adjusting the LTE-U duty cycle to Wi-Fi activity, the proposed algorithm enabled maximal utilization of idle resources for LTE-U transmissions, while decreasing the latency imposed on Wi-Fi traffic. The proposed approach also provided a means to control the trade-off between LTE-U utilization and Wi-Fi latency in the coexisting networks.

In [110], the authors analyze the LTE-U scheme when it coexists with Wi-Fi and introduces an ML technique that can be used by an LTE-U network to learn the wireless environment and autonomously select the transmission opportunity (TXOP) and muting period configurations that can provide fair coexistence with other co-located technologies. Simulation results show how ML can assist LTE-U in finding optimal configurations and adapt to changes in the wireless environment thus providing the desired fair coexistence. Authors in [111] propose a convolutional neural network (CNN) that is trained to perform identification of LTE and Wi-Fi transmissions which can also identify the hidden terminal effect caused by multiple LTE transmissions, multiple Wi-Fi transmissions, or concurrent LTE and Wi-Fi transmissions. The designed CNN has been trained and validated using commercial off-the-shelf LTE and Wi-Fi hardware equipment. The experimentation results show that the data representation affects the accuracy of CNN. The obtained information from CNN can be exploited by the LTE-U scheme to provide fair coexistence between the two wireless technologies.

The related work on ML in the wireless and unlicensed spectrum does not address the problem of accurately identifying the number of Wi-Fi APs which is a crucial first step in ad-

dressing fair coexistence for LTE-U/Wi-Fi coexistence. We modify the classical ML approaches to develop algorithms that can identify the number of Wi-Fi APs on air faster and more reliably than existing methods. Our approach is based on collecting data in realistic coexistence environments for both training and testing. We also compare the performance of the ML-based approaches with the more conventional ED and AC methods described above.

Table 6.2: Experimental Set-up Parameters

Parameter	**Value**
Available Spectrum and Frequency	20 MHz and 5.825 GHz
Maximum Tx power for both LTE and Wi-Fi	23 dBm
Wi-Fi sensing protocol	CSMA/CA
Traffic	Full Buffer (Saturation Case)
Wi-Fi & LTE-U Antenna Type	MIMO & SISO
LTE-U data & control channel	PDSCH & PDCCH
Type of Wi-Fi Clients	2 Google Pixel, 1 Samsung, 1 Redmi, and 1 Apple Laptop

6.3 Channel Access Procedure for Wi-Fi and LTE-U

In this section, we discuss the differences in the channel access procedures for Wi-Fi that uses CSMA/CA and LTE-U that uses the duty cycle mechanism.

6.3.1 Wi-Fi CSMA/CA

The Wi-Fi MAC distributed coordination function (DCF) employs CSMA/CA as illustrated in Figure 6.3. Each node attempting transmission must first ensure that the medium has been idle for a duration of DCF Interframe Spacing (DIFS) using the Energy Detection (ED) and Carrier Sensing (CS) mechanism. If either ED or CS is true, the Clear Channel Assessment (CCA) is set to be busy. If the channel is idle and the station has not just completed a successful transmission, the station transmits. Otherwise, if the channel is sensed busy during the DIFS sensing period or the station is contending after successful transmission, the station persists with monitoring the channel

until it is measured idle for a DIFS period, then selects a random back-off duration (counted in units of slot time) and counts down. Specifically, a station selects a back-off counter uniformly at random in the range of $[0; 2^i W_0 - 1]$ where the value of i (the back-off stage) is initialized to 0 and W_0 is the minimum contention window chosen initially. Each failed transmission due to packet collision results in incrementing the back-off stage by 1 (binary exponential back-off or BEB) and the node counts down from the selected back-off value; *i.e.,* the node decrements the counter every $\sigma(\mu s)$ corresponding to a back-off slot as long as no other transmissions are detected. If during the countdown a transmission is detected, the counting is paused (freeze the back-off counter), and nodes continue to monitor the busy channel until it goes idle; thereafter the medium must remain idle for a further DIFS period before the back-off countdown is resumed for accessing the channel. Once the counter hits zero, the node transmits a packet. When a transmission has been completed successfully, the value of i is reset to 0. The maximum value of the back-off stage i is m with the maximum contention window size of W_m and it stays in m-th stage for one more unsuccessful transmission with the same contention window size W_m, i.e. the retry limit is 1. The value of W_0 and m is determined in the standard. If the last transmission was unsuccessful, the node drops the packet and resets the backoff stage to $i = 0$. If a unicast transmission is successful the intended receiver will transmit an Acknowledgment frame (ACK) after a Short Interframe Spacing (SIFS) duration post successful reception; the ACK frame structure that consists of preamble and MAC header. The ACK frame chooses the highest basic data rate (6 Mbps, 12 Mbps, or 24 Mbps) for transmitting the MAC header which is smaller than the data rate used for data transmission.

6.3.2 LTE-U Duty Cycle

LTE-U uses a duty-cycling approach (i.e. alternating the ON and OFF period, where the LTE-U BS is allowed to transmit only during the ON duration) where the duty cycle (ratio of ON duration to one cycle period) is determined by perceived Wi-Fi usage at the LTE-U BS, using carrier sensing. During the ON period, the LTE-U BS schedules downlink transmissions to a user equipment (UE), unlike Wi-Fi in which transmissions are governed by the CSMA/CA process. Figure 6.4 shows

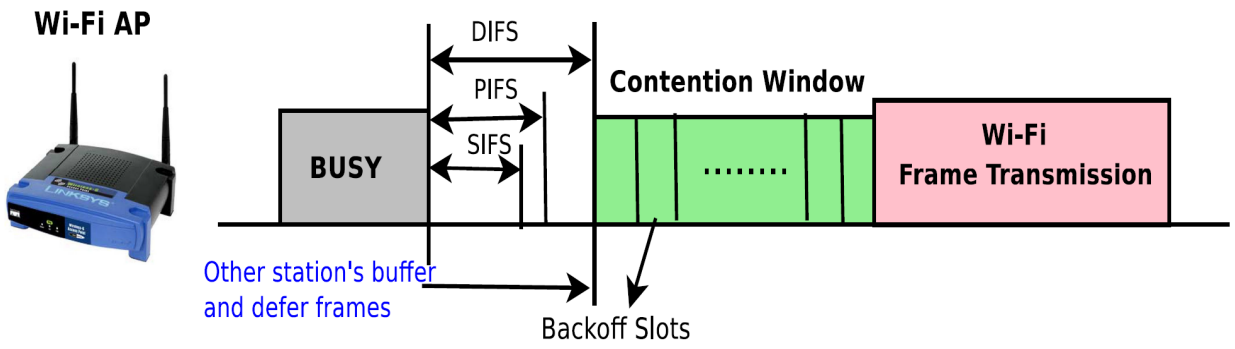

Figure 6.3: Wi-Fi CSMA/CA Transmission

the LTE-U transmission for the duty cycle of 0.5. LTE-U uses the basic LTE subframe structure,

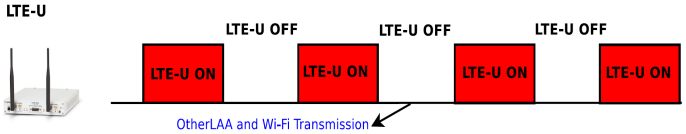

Figure 6.4: LTE-U Duty Cycle Transmission

i.e., the subframe length of 1 ms; each sub-frame consists of two 0.5 ms slots. Each subframe consists of 14 OFDM symbols of which 1 to 3 are Physical Downlink Control Channel (PDCCH) symbols and the rest are Physical Downlink Shared Channel (PDSCH) data. LTE-U BSs start downlink transmissions synchronized with slot boundaries, for (at least) one subframe (2 LTE slots) duration. After transmission, the intended receiver (or receivers) transmits the ACK on the uplink via the licensed band if the decoding is successful.

In LTE, a Resource Block (RB) is the smallest unit of radio resource which can be allocated to a piece of user equipment (UE), equal to 180 kHz bandwidth over a Transmission Time Interval (TTI) of one subframe (1 ms). Each RB of 180 kHz bandwidth contains 12 subcarriers, each with 14 OFDM symbols, equaling 168 Resource Elements (REs). Depending upon the modulation and coding schemes (QPSK, 16-QAM, 64-QAM), each symbol or resource element in the RB carries 2, 4, or 6 bits per symbol, respectively. In the LTE system with 20 MHz bandwidth, there are 100 RBs available.

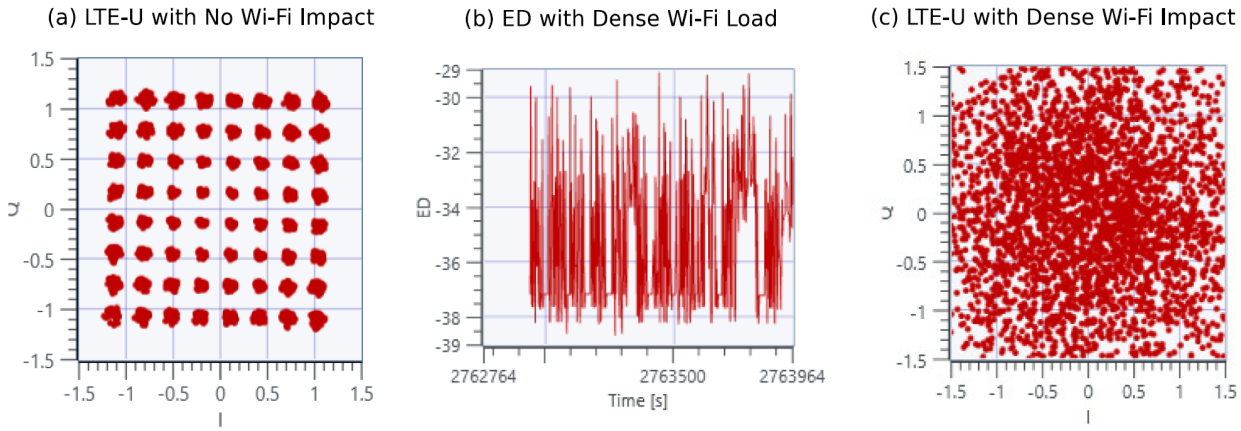

Figure 6.5: Wi-Fi Impact on LTE-U ON Transmission.

6.4 System Model and Mutal Impact of LTE-U and Wi-Fi

6.4.1 Coexistence System Model

We assume a deployment where LTE-U and Wi-Fi are operating on the same unlicensed 20 MHz channel in the 5 GHz band. The LTE-U BS transmits only downlink packets on the unlicensed spectrum, while all uplink transmissions are on the licensed spectrum. [7] Control and data packets are transmitted using PDCCH and PDSCH respectively. The LTE-U BS operates at maximum transmit power using all possible resource blocks and the highest modulation coding scheme (*i.e.*, 64-QAM). We assume that the Wi-Fi APs also operate at maximum transmission power, transmitting a full buffer video traffic. CSMA/CA and duty-cycle adaptation mechanisms are used for channel access for Wi-Fi and LTE-U, respectively. Both Wi-Fi and LTE-U follow their respective retransmission schemes such that when a packet transmission is unsuccessful (packet or acknowledgment lost), the packet will be retransmitted. Finally, we assume that the Wi-Fi APs support both active and passive scanning mode, *i.e.,* both the beacon and probe response packets are transmitted by the AP during the association process.

7. LTE works on two different types of air interfaces (radio links), one is downlink (from tower/BS to device/user equipment), and the other is uplink (from device/user equipment to tower/BS).

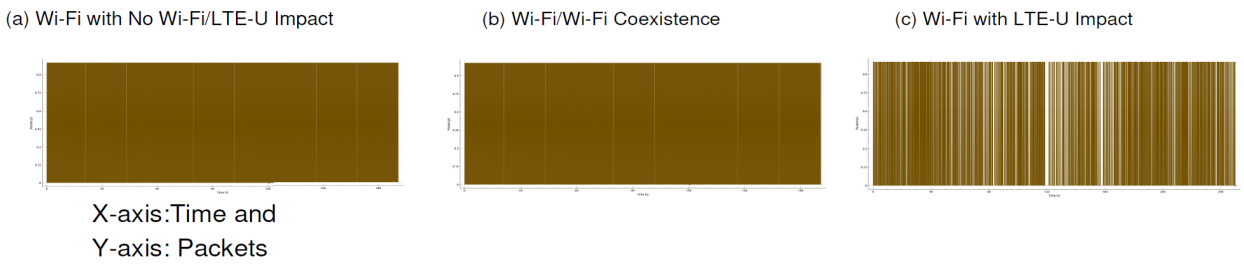

Figure 6.6: LTE-U Impact on Wi-Fi Transmission.

6.4.2 *Impact of Wi-Fi on LTE-U During the ON Period*

To observe the impact of Wi-Fi on LTE-U during the ON period (*i.e.,* LTE-U is ON without appropriate sensing of a Wi-Fi transmission), we deploy a NI based LTE-U BS (Section 6.5 describes the experiment set-up in detail) on channel 165, which is a 20 MHz channel, and five Wi-Fi APs on the same channel. Each client is associated with one Wi-Fi AP with full buffer video transmission. Figure 6.5 (a) shows the constellation of received signals when there is no Wi-Fi AP on the channel, that is, LTE-U BS can transmit the data with the high modulation coding scheme of 64-QAM. Similarly, Figure 6.5 (b) shows the energy value observed when there are 5 Wi-Fi APs on the same channel, where X-axis represents time and Y-axis represents energy values. Figure 6.5 (c) shows the effect of Wi-Fi transmissions on LTE-U during the ON period when Wi-Fi APs are unaware of the sudden LTE-U ON cycle starting in the middle of an ongoing Wi-Fi transmission: the constellation is distorted. This points to the inefficient use of the spectrum and the need for the LTE-U BS to sense or learn the medium to identify the number of Wi-Fi APs on the air and scale back its duty cycle accordingly.

6.4.3 *Impact of LTE-U on Transmission of Wi-Fi Data*

In case of Wi-Fi/Wi-Fi coexistence where 5 Wi-Fi APs are deployed at the distance of 6F, we observe successful transmission of packets as shown in Figure 6.6 (a) and (b). We see that the CSMA/CA mechanism works well for Wi-Fi/Wi-Fi coexistence since the number of packets in error with no LTE-U is similar to that when Wi-Fi coexists with Wi-Fi. Fig 6.6 (c) shows the

packet transmission errors when Wi-Fi coexists with a fixed, LTE-U duty cycle: the number of Wi-Fi packets in error increases. To solve the above problem, the LTE-U forum proposed the dynamic CSAT approach [58, 141, 142] based on the number of Wi-Fi APs on the same channel. Figure 6.7 shows the LTE-U duty cycle adaptation process when detecting a varying number of Wi-Fi APs. When no AP is detected on the channel, the LTE-U BS will operate at the maximum 95% duty cycle [58] (*i.e.,* minimum of 1 ms OFF duration). When one AP is detected (assumed using a predetermined sensing technique), the BS will scale back to a 50% duty cycle (*i.e.*, 20 ms ON time, and 20 ms OFF time). If a new Wi-Fi AP starts transmitting, it will contend with the existing AP only during the OFF time which is 50% of the available medium. Since this is unfair to the Wi-Fi APs, the LTE-U specification recommends scaling the duty cycle back to 33% when more than one Wi-Fi AP is using the channel. However, there is no specific mechanism proposed to detect the number of coexisting Wi-Fi APs in both sparse and dense deployment scenarios.

6.5 Experimental Setup for Machine Learning-Based Detection

The experimental set-up consists of one LTE-U BS and a maximum of five Wi-Fi APs. To emulate the LTE-U BS, we use the National Instruments USRP 2953-R software-defined radio (SDR) which is equipped with the LTE-U radio framework. There are five Netgear Wi-Fi APs and five Wi-Fi clients deployed in a static configuration. The Wi-Fi clients are a combination of laptops and smartphones capable of Wi-Fi 802.11 connection. As soon as the client connects to the Wi-Fi AP, it starts a live video streaming application to simulate a full-buffer transmission. The experimental setup is shown in Figure 6.9 and the complete experimental parameters are described in Table 6.2.

We set the BS and APs to be active in the same 20 MHz channel in the 5 GHz band (*i.e.*, Wi-Fi channel 165, and LTE band 46 EARFCN 53540). We separate the APs and BS into six cells, with five cells (Cell A, C, D, E, and F) as Wi-Fi cells and one cell (Cell B) as the LTE-U cell. Each Wi-Fi cell consists of one AP and one client, while the LTE-U BS and UE are contained within the same USRP board.

The BS transmits full buffer data at maximum power by enabling all of its resource blocks with

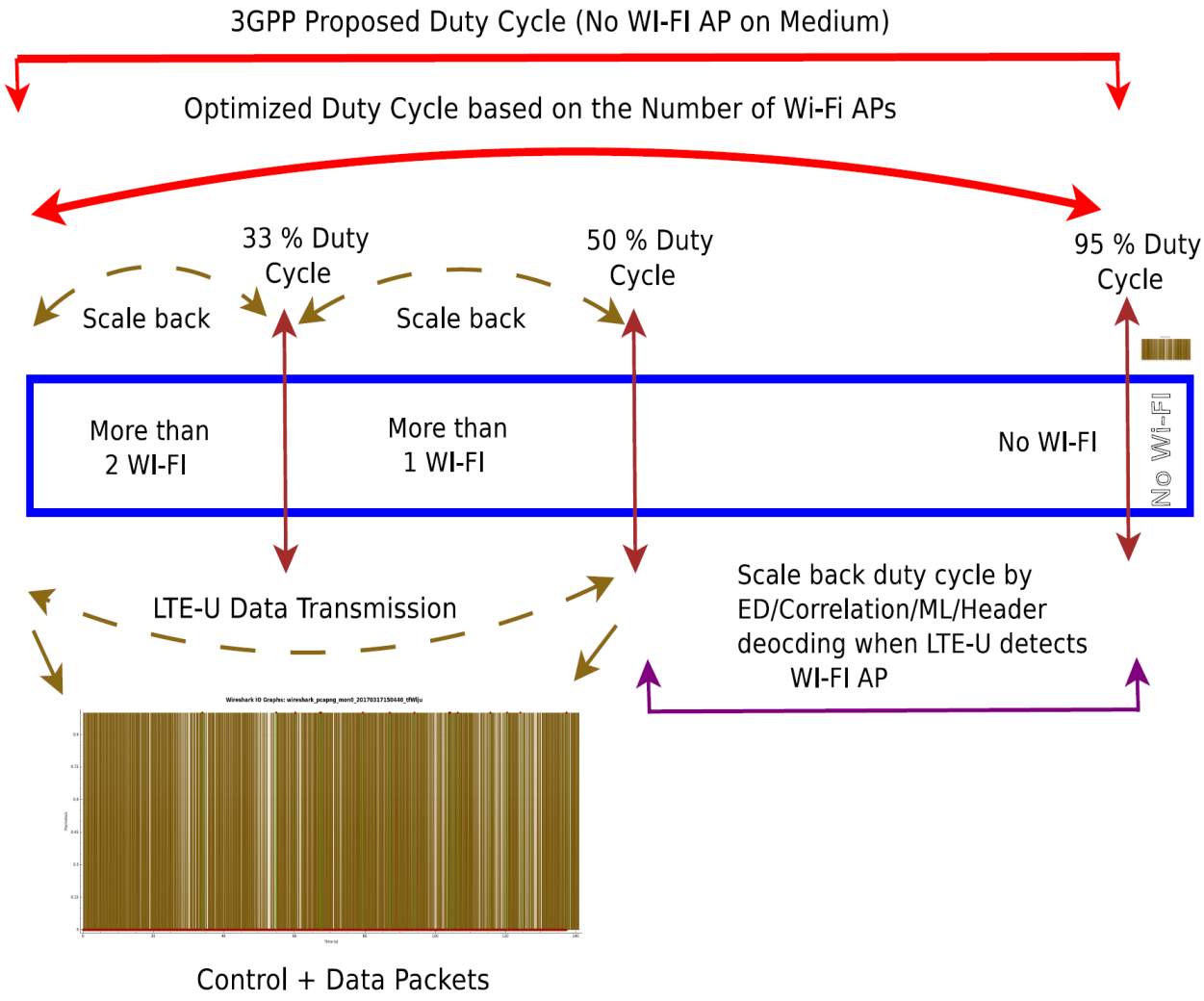

Figure 6.7: LTE-U Duty Cycle Mechanism.

the highest modulation coding scheme (*i.e.*, 64-QAM). It operates at a 50% duty cycle during the experiment, and listens to the configured unlicensed channel during the OFF period for *RF power* and AC measurement. The *RF power* measurement is configured in the LTE block control module of the NI LTE application framework, which outputs energy value as defined in 6.6.2. The AC function is also configured in the LTE block control module of the same framework and outputs the AC events as defined in 6.6.3. The energy values observed from Algorithm 2 are given as input to the ML algorithm (explained in detail in Section 6.7) to classify the number of Wi-Fi APs on the channel. Each Wi-Fi AP transmits full buffer downlink data and beacon frames, with occasional

probe responses if it receives probe requests for clients in the vicinity. We also ensure that there is no extra interference in the channel from other Wi-Fi APs.

We measure the energy (Energy Detection (ED)), AC (Auto-Correlation) value and ML (same energy value as input to ED) at the LTE-U BS for the following scenarios:

- **Scenario 0:** No Wi-Fi APs are deployed and only one LTE-U cell (*i.e.*, Cell B) is deployed.
- **Scenario 1:** One Wi-Fi AP (*i.e.,* Cell A) and one LTE-U (*i.e.*, Cell B) is deployed.
- **Scenario 2:** Two Wi-Fi APs (*i.e.*, Cell A & C) and one LTE-U (*i.e.*, Cell B) are deployed.
- **Scenario 3:** Three Wi-Fi APs (*i.e.*, Cell D, E, & F) and one LTE-U (*i.e.*, Cell B) are deployed.
- **Scenario 4:** Four Wi-Fi APs (*i.e.*, Scenario 1: Cell A, Scenario 3: Cell D, E, & F) and one LTE-U (*i.e.*, Cell B) are deployed.
- **Scenario 5:** Five Wi-Fi APs (*i.e.*, Cell A, C, D, E, & F) and LTE-U (*i.e.*, Cell B) are deployed.

In all scenarios, Cell B measures the energy and AC values during the LTE-U OFF period, while the rest of the Wi-Fi cells are transmitting full buffer downlink transmission. We also vary the distances and the LOS and NLOS environment of each cell. In the NLOS setup, the wall act as an obstruction between the LTE-U and Wi-Fi APs. We measure the received Wi-Fi AP signals at the LTE-U BS for different 6 feet (for example in Scenario 5, where all the 5 Wi-Fi APs placed at 6 feet from the LTE-U BS), 10 feet and 15 feet distances. The previous work focused only on detecting Scenarios 1 and 2 (*i.e.*, 1, and 2 Wi-Fi APs coexisting with LTE-U) [141, 142]. Additionally, we demonstrate that Scenario 0 can be easily distinguished from other scenarios [139].

6.6 LTE-U Duty Cycle Adaptation Algorithms

To solve the problems identified in the previous section, we propose header (HD), energy (ED) and auto-correlation (AC) based detection algorithms for a dense deployment scenario to identify the

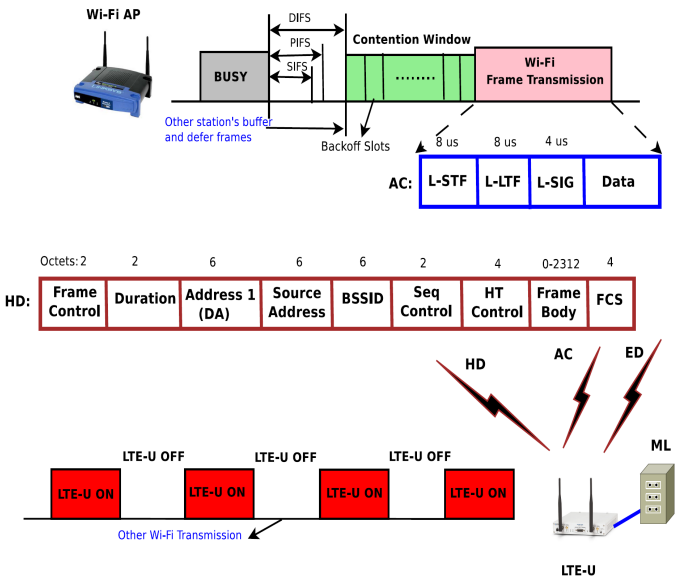

Figure 6.8: LTE-U Duty Cycle Adaptation Algorithm.

number of Wi-Fi APs on the channel. Figure 6.8 explains how different sensing algorithms work based on the known Wi-Fi packet structure.

6.6.1 Header-Decoding based LTE-U Duty Cycle Adaptation Algorithm

We assume that there is either a common preamble [1, 2] between the LTE-U and Wi-Fi systems or the LTE-U BS has a full Wi-Fi decoder that will allow it to decode the Wi-Fi MAC header and obtain the BSSID. Doing so, one can accurately detect the number of Wi-Fi APs on the channel and hence header-based decoding is the most accurate method compared to energy, auto-correlation, and ML. However, the decision algorithm to adapt the duty cycle needs to be designed carefully to avoid misclassification. We define a simple algorithm shown in Algorithm 1, to classify the number of active Wi-Fi APs at each time slot. In brief, the algorithm counts the number of beacons of each uniquely identifiable BSSID, for a defined time slot. Since we can expect that an AP in a real deployment may hop between channels frequently, it is important to collect beacons for a longer time rather than deciding based on just one beacon. We initially set a time slot of 10 beacons (*i.e.*, 1.024 s) and count the number of beacons for each BSSID in the time slot. We set a threshold

of 9 beacons for an AP to be considered as active, this means that there is 90% confidence that the AP is active. The length of the time slot determines the inference delay, thus one would like this delay to be as small as possible. When we reduce the time slot to 5 beacons (0.512 s), to get the same accuracy as for 10 beacons, we need to set the threshold to 4 beacons, which means that the confidence rate is at a lower 80%. Thus, with a slightly lower confidence rate, we can reduce the inference time to half without compromising the detection accuracy.

Algorithm 1 Header-decoding based LTE-U Scale Back.

Initialization: (i) $Beacon_i = 0$

(ii) $Count.detect_i = 0,\ Count.falsealarm_i = 0$

(iii) $LastTime = 0, TimeSlot = 0.512s, Threshold = 4$

```
while true do
    if Wi-Fi beacon with BSSID i is detected then
        ++Beacon_i;
    end
    if CurrentTime − LastTime ≥ TimeSlot then
        NumberOfAp = 0;
        for i in Beacon do
            if Beacon_i ≥ Threshold then
                ++NumberOfAp;
            end
            Beacon_i = 0;
        end
        LastTime = CurrentTime;
        for i = 1 to 5 do
            if i Wi-Fi is ON (ground truth) then
                if i == NumberOfAp then
                    ++Count.detect_i;
                end
                else
                    ++Count.falsealarm_i;
                end
            end
        end
    end
end
```

6.6.2 *Energy based LTE-U Duty Cycle Adaptation Algorithm*

The experiment setup is shown in Figure 6.9. We measure the received energy at the LTE-U BS for different distances between the LTE-U BS and Wi-Fi APs and obtain histograms of the measured signal when one or more Wi-Fi APs are transmitting at 6, 10 and 15 feet from the LTE-U BS. We then fit the measured histograms to probability distribution functions as described in [142] to develop a classification algorithm. In Algorithm 2, an energy-based detection listens to the energy level in the channel and according to a set threshold [142], decides whether to scale back the duty cycle or not. Since the measured energy threshold depends on the number of detected Wi-Fi APs, the choice of threshold is important to the algorithm. Finally, we implement the algorithm in the LTE-U BS NI hardware and validate it experimentally.

First, we modify the NI LTE application framework to measure *RF power* during the LTE-U OFF period. The collected energy values are then averaged over one second time duration and used for algorithm input. If the averaged energy value is greater than the specified threshold α_1, *i.e.*, if energy value $\geq \alpha_1$ then there is a possibility of Wi-Fi packets (beacon, probe request, probe response, data, or ACK) transmitted in the channel. The BS then can declare whether one, two, three, four, or five APs are present, based on the thresholds: α_2, α_3, α_4, α_5 (*e.g.*, if $\alpha_3 \leq$ energy value $\leq \alpha_4$ then there are 4 APs in the channel). By keeping the count of correct and incorrect decisions made by the algorithm, we calculate the probability of correct detection and false positive on predicting the number of Wi-Fi APs in the unlicensed spectrum. These probability values are used as a metric to determine the performance of the threshold, such that we pick a set of thresholds with a high probability of correct detection and low probability of false positive.

Algorithm 2 Energy Based LTE-U Scale Back.

Input: $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$

Initialization: (i) $\alpha_6 = \infty$

(ii) $Count.detect_i = 0, Count.falsealarm_i = 0$

while *true* **do**

  Received $Avg(EnergyLevel)$ over one second.

  for *i = 1 to 5* **do**

    if *i Wi-Fi is ON (ground truth)* **then**

      if $\alpha_i \leq$ *Avg(Energy Level)* $\leq \alpha_{i+1}$ **then**

        $++Count.detect_i$;

      else

        $++Count.false.alarm_i$;

      end

    end

  end

end

6.6.3 Auto-Correlation based LTE-U Duty Cycle Adaptation algorithm

In the same experimental setup as shown in Figure 6.9, we count the total number of AC events that are above a threshold for every one second over the duration of 90 seconds. We measure the total number of events above the AC threshold at the LTE-U BS for 6, 10, and 15 feet distances. Then, we observe the PDF distribution of the number of AC events above the threshold [141] for Scenario 0 to 5 described above. We make use of this key observation to develop a classification algorithm (*i.e.,* Algorithm 3) for both LOS and NLOS scenarios. The algorithm uses AC functions and optimal thresholds to determine the number of Wi-Fi APs in the channel, therefore the selection of thresholds is also important. We implement the algorithm in the LTE-U BS hardware and validate it experimentally. The AC function is performed at LTE-U BS to sense the spectrum for Wi-Fi preamble signals (*i.e.,* L-STF). The output of the function is an AC value that determines

the likelihood that the signal is a Wi-Fi preamble. We observed on many experiments, that the threshold th_ρ of 0.25 is sufficient to determine that the captured signal is a Wi-Fi signal (beacon, probe request, probe response, data, or ACK). Using the threshold, we predicted the number of Wi-Fi signals in every one second period. Next, we calculate the ratio [141] and then compare to R_i which is a threshold determined during a preliminary experiment with i Wi-Fi AP and no LTE-U on the channel. The R_i is determined such that the true positive rate is as high as the possible and false positive rate is as low as possible during the preliminary experiment. Since it is not possible for the observed ratio to be higher than R_i, we set a correct prediction that i Wi-Fi AP is present in the channel if the ratio is less than or equal to the threshold R_i, and false prediction otherwise.

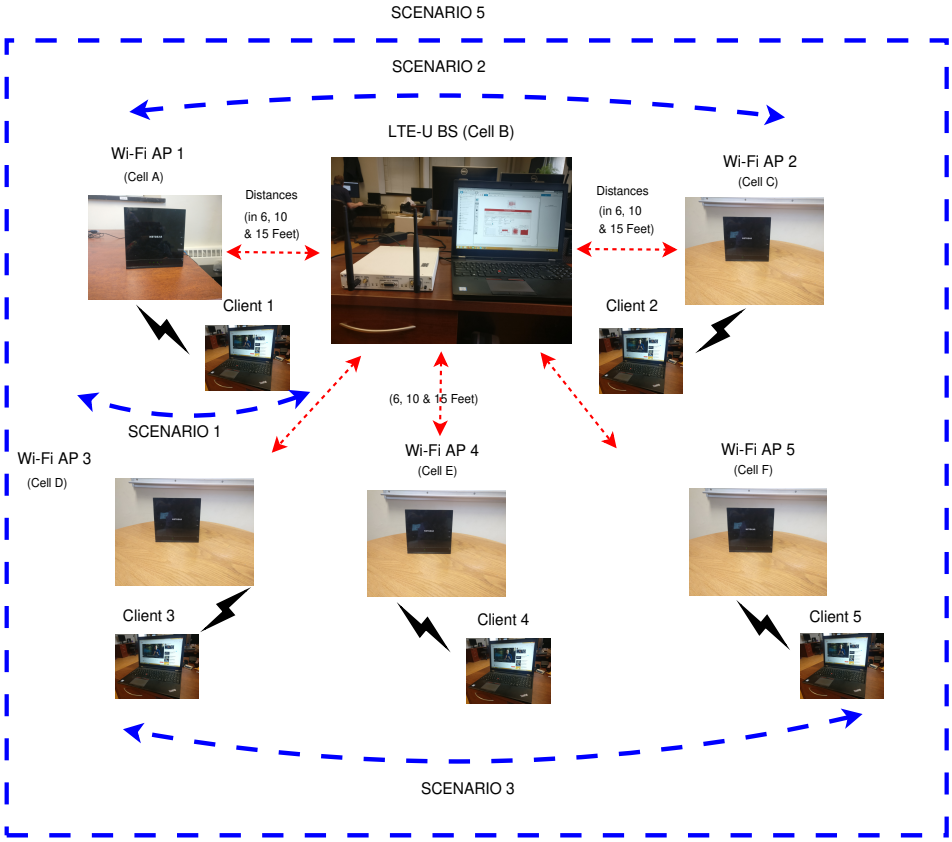

Figure 6.9: LTE Wi-Fi Co-existence Experimental Setup.

Algorithm 3 Auto-correlation Based LTE-U Scale Back.

Input: th_ρ, R

Initialization: $Count.detect_i = 0$, $Count.falsealarm_i = 0$

while *true* **do**

Received T number of AC values over one second.

for i = *1 to 5* **do**

if i *Wi-Fi is ON (ground truth)* **then**

$Signal = 0$;

for t = *1 to* T **do**

if $AC_t \geq th_\rho$ **then**

$++Signal$;

end

end

$ratio = \frac{Signal}{T}$;

if $ratio \leq R_i$ **then**

$++Count.detect_i$;

else

$++Count.falsealarm_i$;

end

end

end

end

6.7 ML Algorithms for LTE-U Duty Cycle Adaption

ML models enable us to replace heuristics with more robust and general alternatives. For the problem of distinguishing between different numbers of Wi-Fi APs, we train a model to detect a pattern in the signals instead of finding specific energy thresholds in a heuristic manner. The

state-of-the-art ML models leverage the unprecedented performance of neural network models that can surpass human performance on many tasks, for example, image recognition [71], and help us answer complex queries on videos [91]. This efficiency is a result of large amounts of data that can be collected and labeled as well as usage of highly parallel hardware such as GPUs or TPUs [31, 87]. We train our neural network models on NVidia GPUs and collect enough data samples that enable our models to achieve high accuracy. The major task is a classification problem to distinguish between zero, one, two, three, four, or five Wi-Fi APs.

We consider machine learning models that take time-series data of width w as input, giving an example space of $\mathcal{X} \in \mathcal{R}^w$, where $\mathcal{R}$ denotes the real numbers. The discrete label space of k classes is represented as $\mathcal{Y} \in \{0, k-1\}$. For example, $k = 3$ classes, enables us to distinguish between 0, 1, and 2 Wi-Fi APs. Machine learning models represent parametrized functions (by a weight vector θ) between the example and label spaces $f(x;\theta) : \mathcal{X} \mapsto \mathcal{Y}$. The weight vector θ is iteratively updated during the training process until the convergence of the training accuracy or training loss (usually determined by very small changes to the values despite further training), and then the final state of θ is used for testing and real-time inference.

6.7.1 Data Preparation

The training and testing data is collected over an extended period with a single scenario taking about 8 hours. For ease of exposition, we consider the case with one and two Wi-Fi APs in this section. We collect data for each Wi-Fi AP independently and store the two datasets in separate files. Each file contains more than 2.5 million values and the total raw data size in CSV format is about 60 MB. Each file is treated as time-series data with a sequence of values that are first divided into chunks. We overlap the time-series chunks arbitrarily by three-fourths of their widths w. For example, for chunks width $w = 128$, the first chunk starts at index 0, the second chunk is formed starting from index 32, the third chunk starts at index 64, and so on. This is part of our data augmentation and a soft guarantee that much fewer patterns are broken on the boundary of chunks. The width w of the (time-series data) chunk acts as a parameter for our ML model. It denotes the

number of samples that have to be provided to the model to perform the classification. The longer the time-series width w, the more data samples have to be collected during inference. The result is higher latency of the system, however, the more samples are gathered, the more accurate the predictions of the model. On the other hand, with a smaller number of samples per chunk, the time to collect the samples is shorter, the inference is faster but of lower accuracy. We elaborate more on this topic in Section 6.8.

The collection of chunks are shuffled randomly. We divide the input data into training and test sets, each 50% of the overall data size. The aforementioned shuffling ensures that we evenly distribute different types of patterns through the training and test sets so that the classification accuracy of both sets is comparable. Training and test sets contain roughly the same number of chunks that represent one or two Wi-Fi APs. We enumerate classes from 0. For the case of 2 classes (either one or two Wi-Fis), we denote by *0* the class that represents a single Wi-Fi AP and by *1* the class that represents 2 Wi-Fi APs. Next, we compute the mean μ and standard deviation σ only on the training set. We check for outliers and replace the values that are larger than 4σ with the μ value (e.g., there are only 4 such values in class *1*).

The data for the two classes have different ranges (from about -45.46 to -26.93 dBm for class *0*, and from about -52.02 to about -22.28 dBm for class *1*). Thus, we normalize the data D in the standard way: $ND = \frac{(D-\mu)}{\sigma}$, where ND are the normalized data output, μ and σ are the mean and standard deviation computed on the training data. We attach the appropriate label to each chunk of the data. The overall size of the data after the preparation to detect one or two Wi-Fi APs is about 382 MB, where the Wi-Fi APs are on opposite sides of the LTE-U BSS and placed at 6 feet distance from the LTE-U BSS). We collect data for many more scenarios and present them in Section 6.8. The final size of the collected data is 3.4 GB.

For training, we do not insert values from different numbers of Wi-Fi APs into a single chunk. The received signal in the LTE-U BSS has higher energy on average for more Wi-Fi APs, thus there are differences in the mean values for each dataset. Our data preparation script handles many possible numbers of Wi-Fi APs and generates the data in the format that can be used for model

training and inference (we follow the format for datasets from the UCR archive [29]).

6.7.2 Neural Network Models: FC, VGG and FCN

The data is treated as a uni-variate time-series for each chunk. There are many different models proposed for the standard time-series benchmark [30].

First, we test *fully connected (FC)* neural networks. For simple architectures with two linear layers followed by the ReLU non-linearity the maximum accuracy achieved is about 90%. More linear layers, or using other non-linearities (e.g. sigmoid) and weight decays do not help to increase the accuracy of the model significantly. Thus, next, we extract more patterns from the data using the convolutional layers.

Second, we adapt the *VGG* network [148] to the one dimensional classification task. We changed the number of weight layers to 6 (we also tested 7, 5, and 4 layers, but found that 6 gives the highest test accuracy of about 99.52%). However, the drawback is that with fewer convolutional layers, the fully connected layers at the end of *VGG* net become bigger to the point that it hurts the performance (for 4 weight layers it drops to about 95.75%). This architecture gives us higher accuracy but is rather difficult to adjust to small data.[8]

Finally, one of the strongest and flexible models called *FCN* is based on convolutional neural networks that find general patterns in time-series sequences [173]. The advantages of the model are simplicity (no data-specific hyper-parameters), no additional data pre-processing required, no feature crafting required, and significant academic and industrial effort into improving the accuracy of convolutional neural networks [48, 98].

The architecture of the FCN network contains three blocks, where each of them consists of a convolutional layer, followed by batch normalization $f(x) = \frac{x-\mu}{\sqrt{\sigma^2+\varepsilon}}$ (where ε is a small constant added for numerical stability) and ReLU activation function $y(x) = \max(0,x)$. There are 128, 256, and 128 filter banks in each of the consecutive 3 layer blocks, where the sizes of the filters are: 8, 5, and 3, respectively. We follow the standard convention for Convolutional Neural Networks

8. The dimensionality of the data is reduced slowly because of the small filter of size 3.

(CNNs) and refer to the discrete cross-correlation operation as convolution. The input x to the first convolution is the time-series data chunk with a single channel c. After its convolution with f filters, with filters denoted as y, the output feature map o has f channels. For training, we insert $s = 32$ time-series data chunks into a mini-batch. We have $j \in f$ and the discrete convolution [168] that can be expressed as:

$$o = x * y \tag{6.1}$$

and in the Einstein notation:

$$o_{(s,j)} = \sum_{i \in c} x_{(s,i)} \cdot y_{(j,i)} \tag{6.2}$$

6.7.3 ML Models from the scikit-learn Library

To diversify the machine learning models used in our comparison, we select the most popular models (described in Section 3.1) from the scikit-learn (also denoted as *sklearn*) library [9]. The library exposes classical machine learning algorithms implemented in Python. This is a common tool used for science and engineering. We run our experiments using *sklearn* version 0.19.1 with Python 3.6. We analyze how the selected models perform on our WiFi data and report their test accuracy.

The Decision Tree classifier achieves an accuracy of 79.46% for the task of distinguishing between one or two Wi-Fi APs. The test accuracy of Random Forest on the same task with Wi-Fi APs is 79.87%. In our experiments, the AdaBoost classifier (described in Section 3.1.3) begins by fitting a Decision Tree classifier on the original dataset and then fits additional Decision Tree classifiers on the same dataset but where the weights of incorrectly classified instances are modified such that subsequent classifiers focus more on difficult cases. It is tuned by adjusting the maximum number of Decision Tree classifiers used. AdaBoost achieves an accuracy of 94.57% on the 2 Wi-Fi

9. https://scikit-learn.org/stable/index.html

dataset.

We find that the best-tested model from the *sklearn* library is AdaBoost. The highest test accuracy achieved for AdaBoost for the standard case with two Wi-Fi APs is worse by about 5% when compared to the overall best FCN model (described in section 6.7.2), which achieves an accuracy of 99.38% for the same configuration (with 2 Wi-Fi APs, 512 chunk size, NLOS, and 6 feet distance). For the case with two classes, Random Forest performs similarly to Decision Tree for two Wi-Fi data. However, for more than 2 classes, the Random Forest model achieves significantly higher accuracy than the Decision Tree classifier (see Figure 6.15).

6.7.4 Time-series Specific Models

The BOSS (Bag-of-SFA-Symbols) model is a structure-based similarity measure. It extracts patterns (substructures) from raw time series and applies low pass filtering and quantization to the substructures to reduce noise as well as to allow for the application of string matching algorithms. This Symbolic Fourier Approximation (SFA) results in SFA words. The BOSS model describes the structure of a time series by a histogram built on the SFA words. The time series are compared pair-wise based on the differences in the set of patterns.

BOSS in Vector Space (BOSS VS) model [146] is a time-series classification algorithm, whose properties make it suitable for our task since it extends the BOSS model to be scalable (about an order of magnitude speed-up) but also preserves the properties of high classification accuracy and resilience to noise. BOSS VS compactly represents classes instead of the time series by using the TF-IDF representation for each class. This algorithm is characterized by fast inference and tolerance to noise that enables us to achieve high test accuracy, and moderate training time, which allows for periodic model updates. Moreover, BOSS VS achieves the best test accuracy for repetitive and long time-series data. Within the time-series specific models, we also test the WEASEL [145] model, which yielded lower test accuracy despite much longer training time.

We run the BOSS VS time-series specific model for the NLOS 6 feet case. Other time-series models train much longer (in the order of days) on our large (a few GBs) time-series data or do

not fit even into 128 GB of RAM provided. We observe that from 2 to up to 4 WiFi APs, the performance of the BOSS VS model is on-par with the performance of the FCN model. However, for the scenario where we have to distinguish between 0 to 5 WiFi APs, the accuracy of the FCN model is higher by about 7%. One concern with the BOSS VS model is that we have to use a machine with 128 GB of RAM to train the model and for larger data sizes, the out of memory exception is thrown as well (the model is implemented in Java). For the FCN, we can scale to an arbitrary amount of data. Based on the thorough experimental analysis, we see the FCN model and other neural network-based models as the most accurate and scalable models that can be used to predict the number of Wi-Fi APs.

6.8 Experimental Results

In this section, we discuss the model training, inference, and transition between different classes. The code for our project can be found on GitHub: `http://bit.ly/2Ob5kAr`.

6.8.1 Training and Inference

Each model is trained for at least 100 epochs. We experiment with different gradient descent optimization algorithms, e.g. Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam) [10]. For the SGD algorithm, we grid search for the best initial learning rate and primarily use 0.0001. The learning rate is reduced on a plateau by 2X after 50 consecutive iterations (scheduled patience). SGD is used with a momentum value of 0.9. We use standard parameters for the Adam optimization algorithm. The batch size is set to $s = 32$ to provide high statistical efficiency. The weight decay is set to 0.0001. For our neural network models, the dataset is relatively simple. The Wi-Fi data can be compared in its size and complexity to the MNIST dataset [41] or the GunPoint series from the UCR archive [30].

10. A very good explanation can be found here: http://bit.ly/2Y9XaQ8

6.8.2 *Time-series Width*

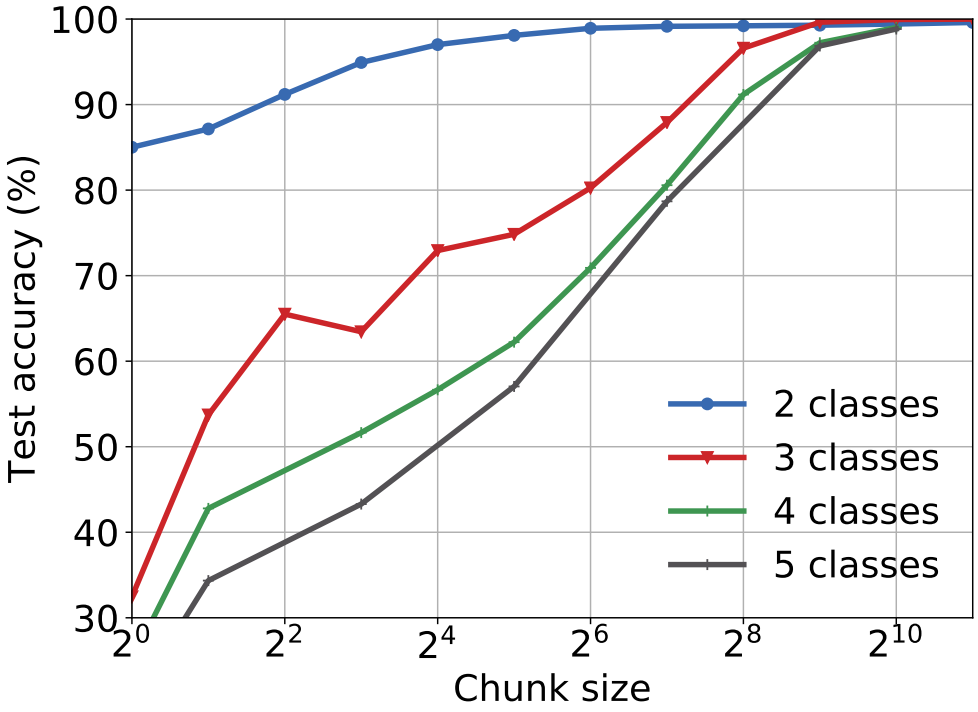

Figure 6.10: The test accuracy (%) for a model trained and tested for a given chunk size (ranging from 1 to 2048) to distinguish between 2 classes (either 1 or 2 Wi-Fi APs), 3 classes (distinguish between 0, 1, or 2 Wi-Fi APs), 4 classes (distinguish between 0, 1, 2, or 3 Wi-Fi APs), and 5 classes (distinguish between 0, 1, 2, 3 or 4 Wi-Fi APs)

The number of samples collected per second by the LTE-U BS is about 192. The inference of a neural network is executed in milliseconds and can be further optimized by compressing the network. The final width of the time-series chunk imposes a major bottleneck in terms of the system latency. The smaller the time-series chunk width w, the lower the latency of the system. However, the neural network has to remain highly accurate despite the small amount of data provided for its inference. Thus, we train many models and systematically vary the chunk width w from 1 to 2048 (see Figure 6.10). In this case, each model is trained only for the single scenario (placement of the Wi-Fi APs) and with zero, one, two, or three active Wi-Fi APs. When we decrease the chunk sizes to the smaller chunk consisting of a single sample, the test accuracy deteriorates steadily down to the random choice out of the 3 or 4 classes (accuracy of about 33% and 25%, respectively) and for

the 2 classes, its performance is very close to the ED (Energy-based Detection) method.

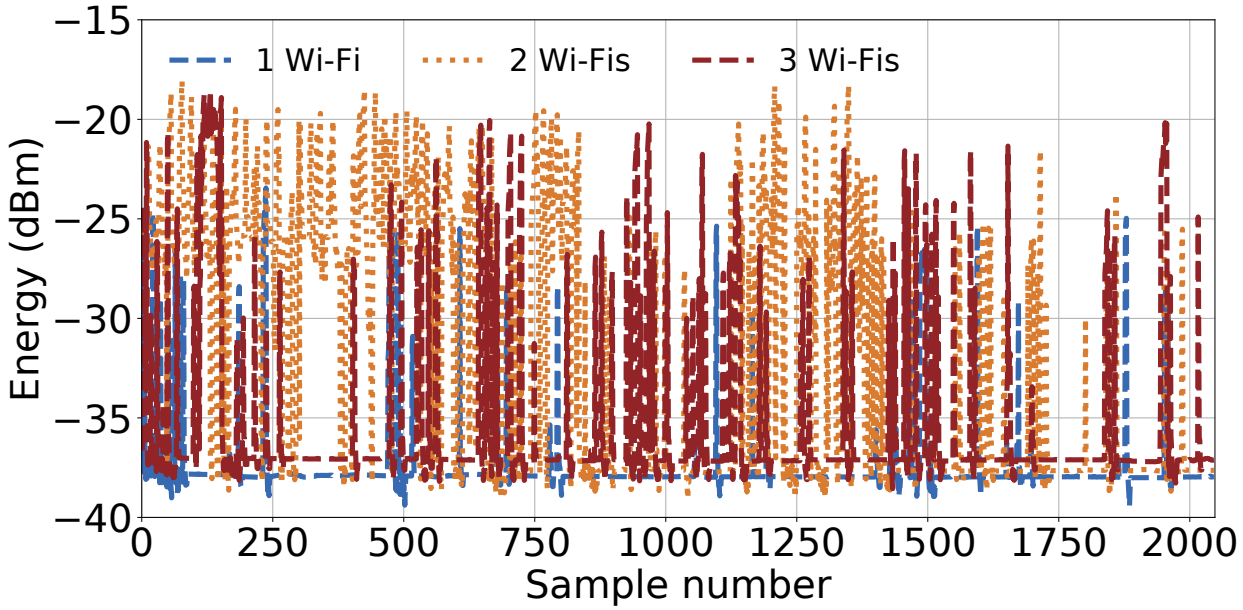

Figure 6.11: **Number of Wi-Fi APs.** The values of the energy (in dBm) captured for 2048 samples in LTE-U BS while there are 1 Wi-Fi, 2, and 3 Wi-Fis scenarios at 6 Feet, NLOS. The more Wi-Fi APs active, the more energy picks we observe.

We present the energy of the signals captured in different configurations: (1) Figure 6.11 shows the values of energy captured for different number of Wi-Fi APs (one, two and three), (2) Figure 6.12 demonstrates the scenario with different distances of Wi-Fi APs from the LTE-U, and (3) Figure 6.13 gives insight into energy of the signal in NLOS and LOS scenarios.

We consider in detail the signal from about 1500th sample to 2000th sample in Figure 6.11. It is challenging to distinguish between two or three Wi-Fis [11]. The visual inspection of the signals suggests that the width of the time-series chunk should be longer than 500 samples. Signals with a width of 384 achieve test accuracy below 99% and signals with width 512 can be trained to obtain 99.68% of test accuracy. Based on the experiments in Figs. 6.10 and 6.11, we find that the best trade-off between accuracy and inference time is achieved for chunk of size 512.

11. The Energy values for 4 and 5 Wi-Fi APs are more dense and challenging. To better visualize we plotted only 1, 2, and 3 Wi-Fi APs

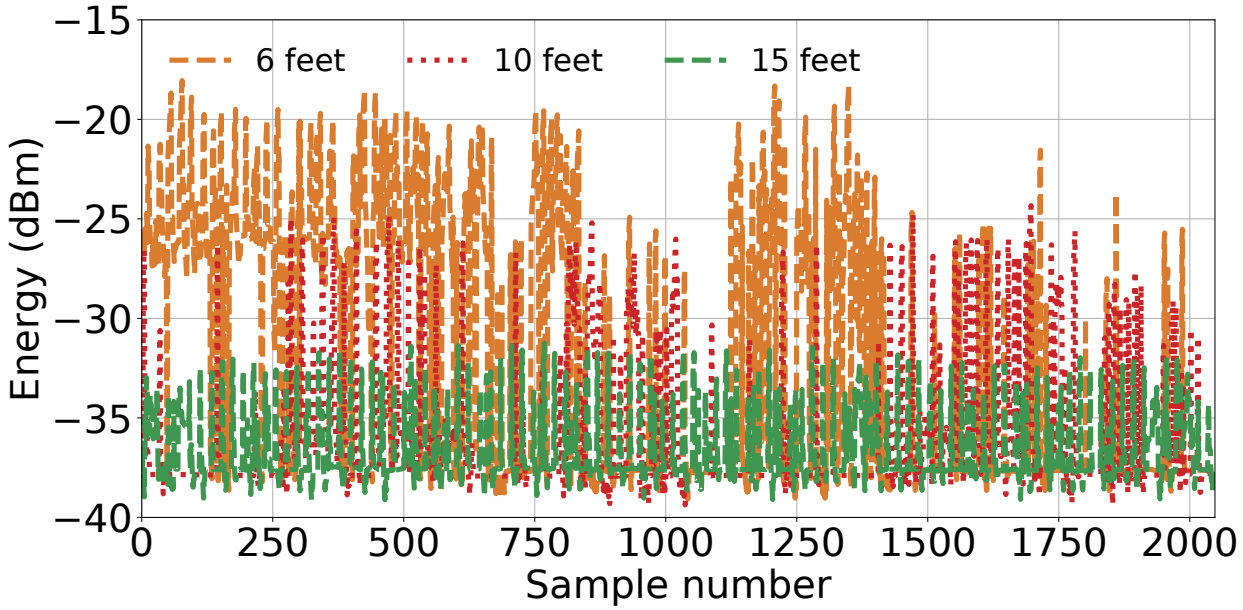

Figure 6.12: **Distances from LTE-U.** The values of the energy (in dBm) captured for 2048 samples in LTE-U BS while there are 2 Wi-Fi APs at 6, 10, and 15 Feet, NLOS. The closer the Wi-Fi APs are to the LTE-U, the higher energy is captured.

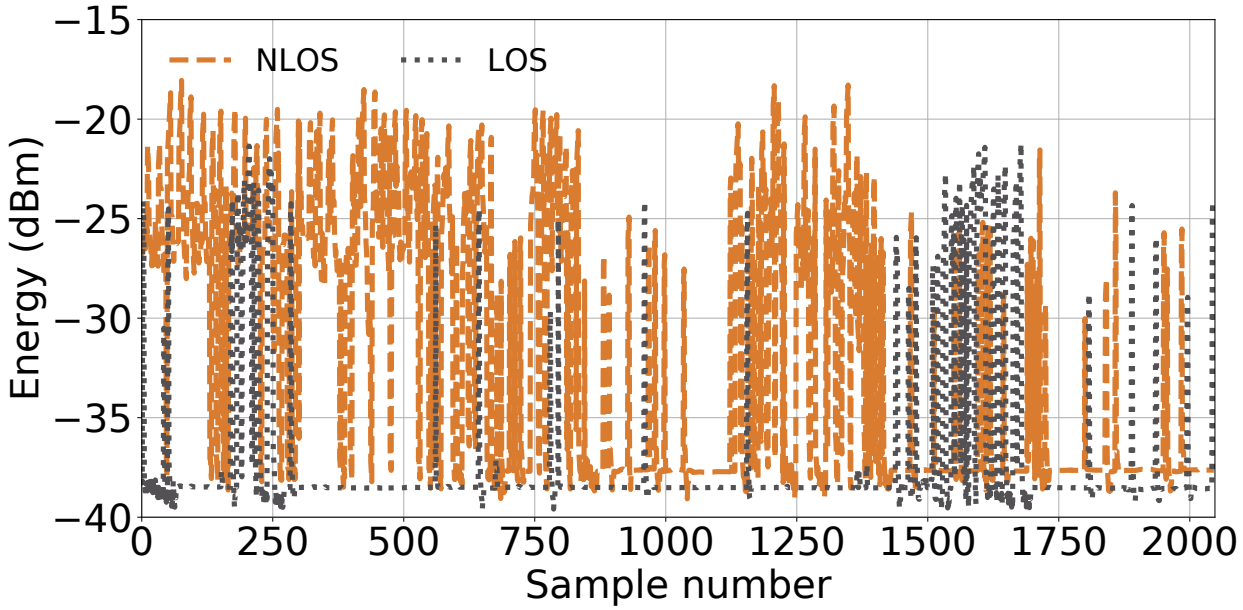

Figure 6.13: **NLOS vs LOS.** The values of the energy (in dBm) captured for 2048 samples in LTE-U BS while there are 2 Wi-Fi APs at 6 Feet, in NLOS and LOS scenarios. The fewer obstructions, the higher energy is captured.

6.8.3 *Transitions Between Classes*

When we switch to another class (change the state of the system in terms of the number of Wi-Fis), we account for the transition period. If in a given window of 1 second a new Wi-Fi is added, the samples from this first second with new Wi-Fi (or without one of the existing Wi-Fis - when it is removed), the chunk is containing values from n and $n+1$ (or $n-1$) number of Wi-Fis. An easy workaround for the *contaminated* chunk is to change the state of the system to a new number of Wi-Fis only after the same class is returned by the model in two consecutive inferences (classifications).

6.8.4 *Real-time Inference*

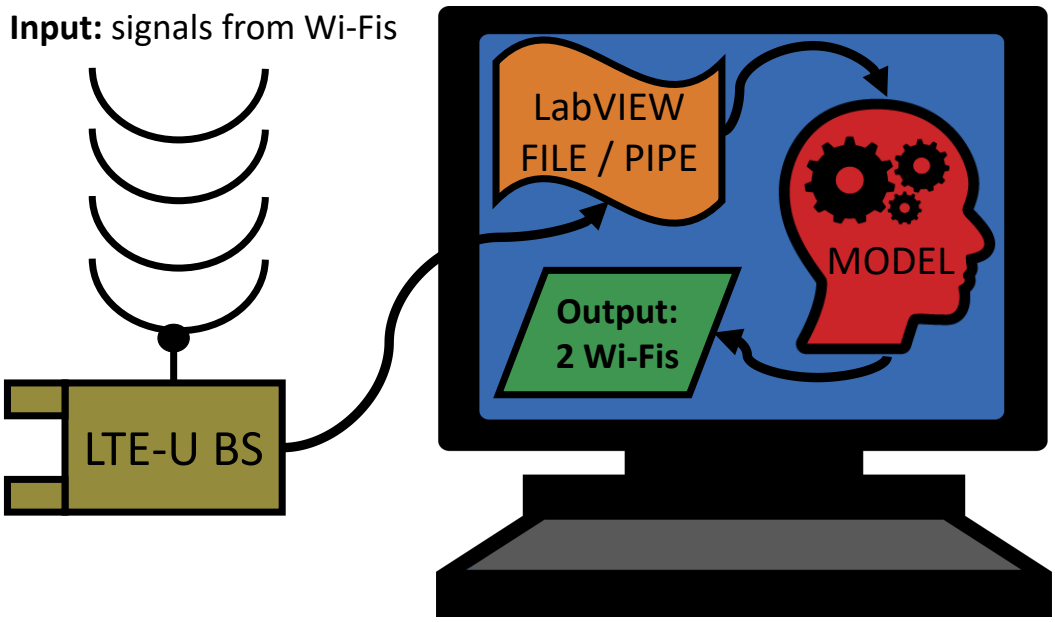

Figure 6.14: The schema of the inference process, where the input received by the LTE-U BS is signals from Wi-Fi APs and the output is the predicted number of Wi-Fi APs.

We deploy the model in real-time, which is similar to the energy data collection experiment setup, and is shown in Figure 6.14. We prepare the model only for the inference task in the following steps. Python scripts load and deploy the trained PyTorch model. We set up the Wi-Fi devices and generate some network load for each device. The LTE-U BS is connected to a computer with

the hardware requirements of at least 8 GB RAM (Installed Memory), 64-bit operating system, x64-based processor, Intel(R) Core i7, CPU clock 2.60GHz. The energy of the Wi-Fi transmission signal in a given moment in time is a capture using NI LabVIEW. From the program, we generate an output file or write the data to a pipe. The ML model reads the new values from the file until it reaches the time-series chunk length. Next, the chunk is normalized and passed through the model that gives a categorical output that indicates the predicted number of Wi-Fis in the real-time environment.

6.9 Performance Comparison Between HD, ED, AC and ML Methods

We analyze and study the performance differences between HD, ED, AC, and ML methods for different configuration setups and discus the inference delay. In the ML method, we validate the performance of ML real-time inference data. For the final evaluation, we train a single Machine Learning model that is based on the FCN network and used for all the following experiments. The model is trained on the whole dataset of size 3.4 GB, where the train and test sets are of the same size of about 1.7 GB.

6.9.1 Comparison Between ML Methods

We present comparison between ML methods in Figure 6.15. The time-series specific neural network models, such as FCN (6.7.2) as well as BOSS VS (6.7.4), perform much better than the general purpose models from scikit-learn library (described in section 6.7.3). The middle-ground between the two options is a simple two-layer convolutional network called *LeNet*. The main benefit of using FCN (MEDIUM) or BOSS VS is greater model learning capacity than *LeNet* or scikit-learn models. There is a negligible difference in terms of test accuracy between the FCN and BOSS VS models. However, the FCN models can scale to much bigger data sizes and we observe that the BOSS VS model often goes out of memory for more than a few GBs of input data. Thus, we select FCN as our main Machine Learning (ML) model for all the remaining experiments.

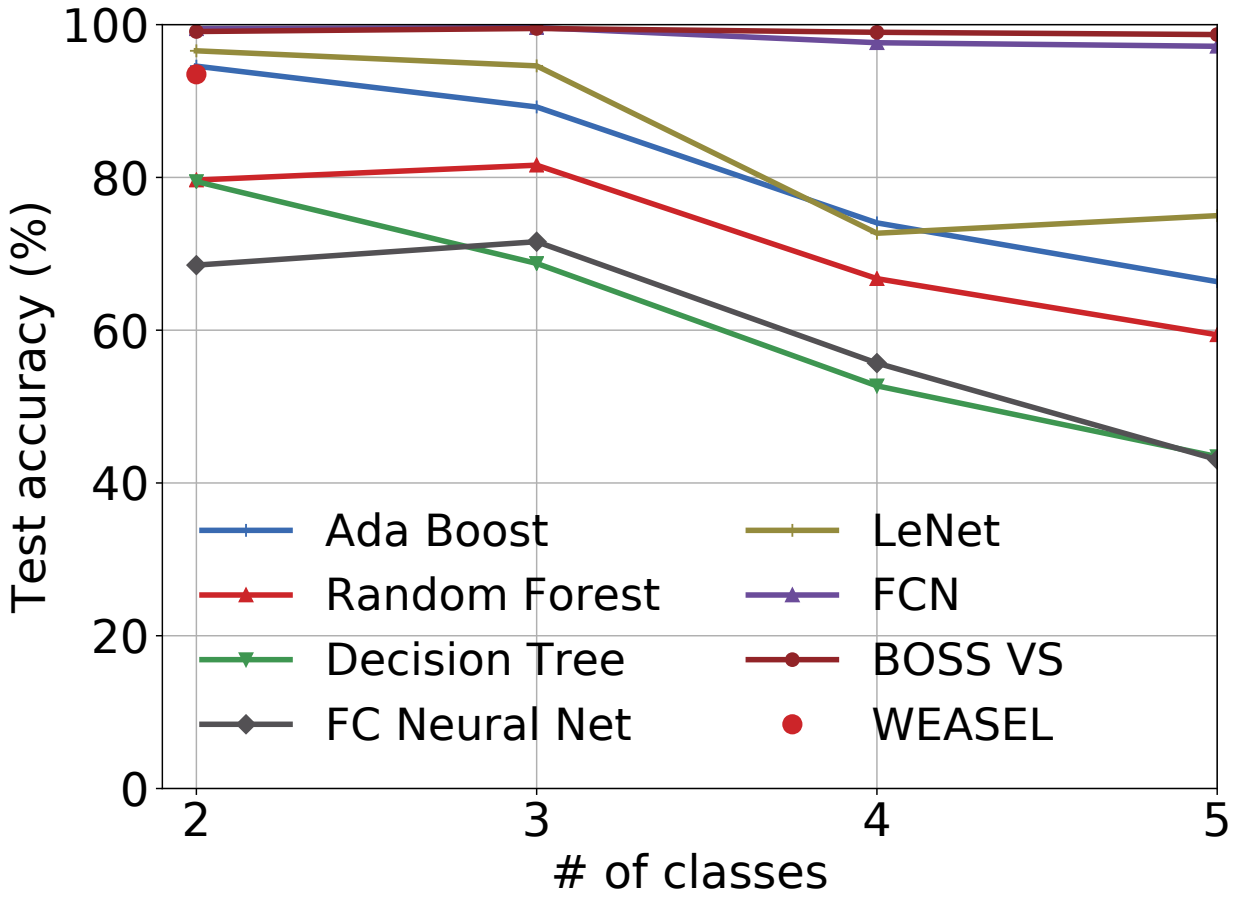

Figure 6.15: Comparison of test accuracy for different ML methods. Number of Wi-Fi APs equals to 2 denotes the Case D configuration (NLOS, 6 feet). Thus, 2 on the x axis corresponds to distinguishing between 1 and 2 Wi-Fi APs, whereas 3 denotes distinguishing between 0, 1, or 2 Wi-Fi APs. Similarly, the values on the x axis (4,5) denote distinguishing from 0 to (x-1) WiFi APs.

6.9.2 Successful Detection at Fixed Distance

We compare the ML performance with HD[12], ED and AC approach using the NI USRP platform as shown in Figure 6.9. Similarly, we compare the performance of HD by analyzing the Wi-Fi BSSID through Wireshark capture. In the experiment, Wi-Fi APs are transmitting full buffer data, along with beacon and probe response frames following the 802.11 CSMA specification. We performed different experiments with 6ft, 10ft, and 15ft for LOS and NLOS scenarios. Fig 6.16 shows the

12. The successful Wi-Fi detection in HD for LOS and NLOS scenarios is 100%. Hence we have not included in Figure 6.16.

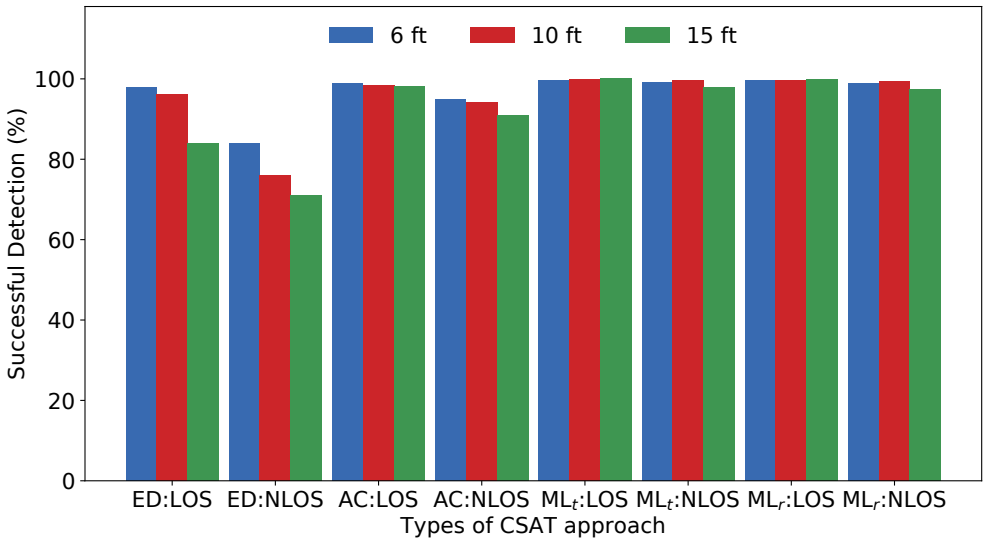

Figure 6.16: Comparison of results for successful detection between ED, AC and ML methods. ML results are presented for the test data (denoted as ML_t:) and for the real time inference (denoted as ML_r:).

performance of detection for LOS and NLOS scenarios. In ED and AC based approaches, the proposed detection algorithm achieves the successful detection on average at 93% and 95% for the LOS scenario. Similarly, the algorithm achieves 80% and 90% for the NLOS scenario. In this work, we show that the ML approach can achieve close to 100% successful detection rate for both LOS and NLOS, and different distance scenarios (6ft, 10ft & 15ft). We observe the ML approach works close to the performance of HD.

Table 6.3 shows the performance of detection for fixed distance configuration setup. From, this table the number of Wi-Fis columns represents the number of Wi-Fi APs deployed in the coexistence setup. The number of Wi-Fi AP 2 corresponds to distinguishing between 1 and 2 Wi-Fi APs, whereas 3 denotes distinguishing between 0, 1, or 2 Wi-Fi APs and so on. In all cases, the performance of ML is close to 100%.

6.9.3 Successful Detection at Different Configurations

We verify how the detection works in different configurations. We placed more than two Wi-Fi APs on the same side of the LTE-U BS, unlike the above configuration (i.e., 6ft, 10ft and 15ft) where

Table 6.3: Performance of detection for fixed distance configuration setup.

Config	Classes	HD (%)		ED (%)		AC (%)		ML (%)	
Distance	Wi-Fis	LOS	NLOS	LOS	NLOS	LOS	NLOS	LOS	NLOS
6F	2	100	100	96	91	98	96	98.60	99.10
	3	100	100	88	85	95	90	99.10	99.50
	4	100	100	80	74	87	81	99.40	99.00
	5	100	100	74	62	76	65	99.20	98.70
	6	100	100	62	51	70	59	99.30	99.0
10F	2	100	100	94	89	97	94	99.80	99.98
	3	100	100	86	82	91	88	99.80	99.98
	4	100	100	78	72	85	79	99.80	99.90
	5	100	100	72	60	75	63	99.50	99.85
	6	100	100	64	54	68	57	99.80	99.84
15F	2	100	100	92	87	95	90	99.80	99.80
	3	100	100	84	80	85	81	99.90	99.60
	4	100	100	75	70	79	71	99.90	99.60
	5	100	100	70	58	71	64	99.60	99.50
	6	100	100	63	53	66	55	99.50	99.40

they were on opposite sides. Wi-Fi AP 1, Wi-Fi AP 2, Wi-Fi AP 3, Wi-Fi AP 4 and Wi-Fi AP 5 are placed at distances of 6 feet, 10 feet and 15 feet from the LTE-U BS respectively. We measured the performance of detection with LOS and NLOS configurations. The goal in this section is to observe the performance of detection in the ML compared with HD, ED, and AC. Some of the possible cases are listed below.

- **Case A:** Only the Wi-Fi AP 1 at 6 feet is ON.
- **Case B:** Only the Wi-Fi AP 2 at 10 feet is ON.
- **Case C:** Only the Wi-Fi AP 3 at 15 feet is ON.
- **Case D:** Wi-Fi AP 1 at 6 feet is ON and Wi-Fi AP 2 at 6 feet is ON.
- **Case E:** The Wi-Fi AP 1 at 6 feet and Wi-Fi AP 3 at 15 feet is ON.
- **Case F:** The Wi-Fi AP 1 at 10 feet and Wi-Fi AP 3 at 15 feet is ON.
- **Case G:** Wi-Fi AP 1 and Wi-Fi AP 2 at 6 feet is ON and Wi-Fi AP 3 at 15 feet is ON.

- **Case H:** Wi-Fi AP 1 at 6 feet is ON, Wi-Fi AP 2 at 10 feet is ON, and Wi-Fi AP 3 at 15 feet is ON.
- **Case I:** Wi-Fi AP 1 and Wi-Fi AP 2 at 6 feet is ON, Wi-Fi AP 3 at 10 feet is ON and Wi-Fi AP 4 at 15 feet is ON.
- **Case J:** Wi-Fi AP 1 at 6 feet is ON, Wi-Fi AP 2 and Wi-Fi AP 3 at 10 feet is ON and Wi-Fi AP 4 at 15 feet is ON.
- **Case K:** Wi-Fi AP 1 and Wi-Fi AP 2 at 6 feet is ON, Wi-Fi AP 3 and Wi-Fi AP 4 at 10 feet is ON and Wi-Fi AP 5 at 15 feet is ON.
- **Case L:** Wi-Fi AP 1 at 6 feet is ON, Wi-Fi AP 2 , Wi-Fi AP 3 and Wi-Fi AP 4 at 10 feet is ON and Wi-Fi AP 5 at 15 feet is ON.
- **Case M:** Wi-Fi AP 1 at 6 feet is ON, Wi-Fi AP 2 at 10 feet is ON, Wi-Fi AP 3, Wi-Fi AP 4 and Wi-Fi AP 5 at 15 feet is ON.
- **Case N:** Wi-Fi AP 1 and Wi-Fi AP 2 at 6 feet is ON, Wi-Fi AP 3 at 10 feet is ON, Wi-Fi AP 4 and Wi-Fi AP 5 at 15 feet is ON.

The different configurations are for LTE-U when it coexists with a different number of Wi-Fi APs (from 1 to 5). Tables 6.4 and 6.5 show better performance for ED and AC compared to the tables 6.6 and 6.7. This is due to a fewer number of Wi-Fi AP deployments from Case A to G compared to Case H to N. Hence, the ED and AC methods can detect the number of Wi-Fi APs close to 80% for ED and up to 90% for AC. As the number of Wi-Fi APs increases from 3 to 5 Wi-Fi APs (*i.e.,* Case H to N), we observe substantial degradation in ED performance (to 56%) and AC performance (to 63%). Tables 6.4, 6.5, 6.6, and 6.7 show that there is no such degradation in the performance of ML as compared to ED and AC. Hence, we believe that the ML approach is the preferred method for a LTE-U BS in a dense environment to detect the number of Wi-Fi APs and scale back the duty cycle efficiently.

Table 6.4: Performance of detection for different configuration setup (from case A to D).

CSAT Types	CASE A (%)		CASE B (%)		CASE C (%)		CASE D (%)	
	LOS	NLOS	LOS	NLOS	LOS	NLOS	LOS	NLOS
HD	100	100	100	100	100	100	100	100
ED	91	82	90	79	85	78	82	77
AC	95	91	94	91	92	90	91	90
ML	98.80	97.96	99.94	99.37	99.96	97.74	99.46	97.80

Table 6.5: Performance of detection for different configuration setup (from case E to G).

CSAT Types	CASE E (%)		CASE F (%)		CASE G (%)	
	LOS	NLOS	LOS	NLOS	LOS	NLOS
HD	100	100	100	100	100	100
ED	80	74	81	72	80	69
AC	88	85	88	83	86	77
ML	99.21	99.14	99.32	99.10	99.56	98.44

Table 6.6: Performance of detection for different configuration setup (from case H to K).

CSAT Types	CASE H (%)		CASE I (%)		CASE J (%)		CASE K (%)	
	LOS	NLOS	LOS	NLOS	LOS	NLOS	LOS	NLOS
HD	100	100	100	100	100	100	100	100
ED	80	69	77	67	76	65	68	61
AC	84	74	79	68	77	67	75	65
ML	98.70	97.36	99.24	98.26	99.76	98.24	98.83	98.06

Table 6.7: Performance of detection for different configuration setup (from case L to N).

CSAT Types	CASE L (%)		CASE M (%)		CASE N (%)	
	LOS	NLOS	LOS	NLOS	LOS	NLOS
HD	100	100	100	100	100	100
ED	67	56	68	57	66	52
AC	74	64	72	63	71	59
ML	99.11	99.04	99.02	98.05	99.96	97.74

Table 6.8: The delay to detect the Wi-Fi AP due to the NI hardware.

CSAT Types	NI HW Delay (sec)
Header Decoding (HD)	1.4
Energy Detection (ED)	5.9
Auto-correlation (AC)	4.8
Machine Learning (ML)	3.1

6.9.4 Delay to Detect Wi-Fi APs

To study the additional delay to detect a Wi-FI AP, we consider a 5 Wi-Fi AP deployment scenario, where, Wi-Fi AP 1 and Wi-Fi AP 2 at 6 feet are ON, Wi-Fi AP 3 and Wi-Fi AP 4 at 10 feet are ON and Wi-Fi AP 5 at 15 feet is ON. We observe a large number of Wi-Fi packets on the air and the LTE-U ON cycle interference impacts the delay in Wi-Fi transmissions. In HD, the total time for the LTE-U BS to decode the BSSID is 1.4 seconds (i.e., Wi-Fi 1st BSSID beacon packet + LTE-U detects K beacon + Additional layer complexity + NI USRP RIO hardware processing time). In ED, the total time for the energy-based CSAT algorithm to adopt or change the duty cycle from 50% to 33% is 5.9 seconds (i.e., Wi-Fi 1st beacon transmission time + LTE-U detects K beacon (or) data packets time + NI USRP RIO hardware processing time) as shown in Table 6.8. In AC, the total time for the AC based CSAT algorithm to change the duty cycle from 50% to 33% is 4.8 seconds (i.e., Wi-Fi 1st L-STF packet frame + LTE-U detects L-STF frame time + NI USRP RIO hardware processing time). In ML, the total time for the CSAT algorithm to adopt the duty cycle from 50% to 33% is about 3.1 seconds. This approach is dependent on the chunk size (in this case set to 512).

6.10 FFT Compression

We test the FCN model using the *band-limited* convolutional layers (FFT based convolutional layers with compression) as described in Chapter 4. The results are presented in Figure 6.17. We observe that for 2 and 3 classes the data is highly compressible and we can allow up to even 60% compression with the test accuracy preserved on the level of above 99%. As we increase the

number of classes, the accuracy of the model gracefully degrades and the 60% compression rate allows us to retain the test accuracy of about 90% for 5 classes.

We do not observe a significant difference between the cases with 2 and 3 classes. For 2 classes, we have 1 or 2 Wi-Fi APs and for 3 classes, we distinguish between 0, 1, or 2 Wi-Fi APs. The signal for no Wi-Fi APs is very different and hence easier to classify than for the remaining signals with active Wi-Fi APs.

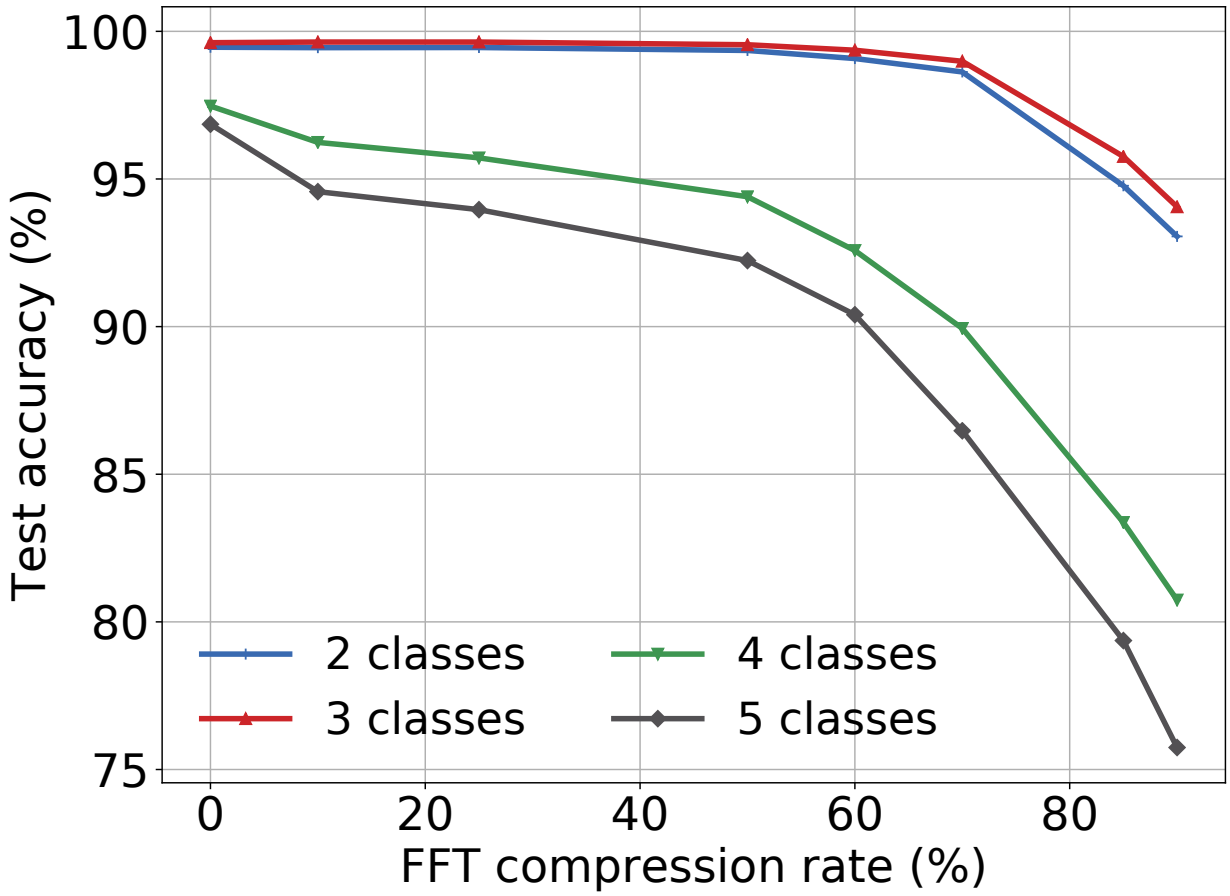

Figure 6.17: Effect of FFT compression embedded into the convolutional layers of the FCN model on test accuracy. We use the Case D configuration for 2 classes and the same configuration with NLOS and 6 feet for the remaining classes.

6.11 Conclusions

We have presented a comprehensive experimental study of different kinds of ML algorithms that could be used to address the problem of identifying the number of active Wi-Fi APs on the air to aid in setting the LTE-U duty cycle appropriately. Additionally, we have compared the performance of the optimum ML algorithm based on the band-limited models to conventional methods using energy detection and auto-correlation detection and demonstrated superior performance in multiple configurations. We believe that this is the first result that demonstrates the feasibility of using ML on energy values in real-time, instead of packet decoding [23], to reliably distinguish between the presence of a different number of Wi-Fi APs. Such a result can have applications beyond LTE-U duty-cycle adaptation, for example in better Wi-Fi frequency management.

CHAPTER 7

SUMMARY AND FUTURE WORK

7.1 Conclusions

This thesis presents neural networks that are more adaptive, performant, and robust against adversarial as well as out-of-distribution examples. Our primary method relies on different transformation techniques within neural networks with the main focus on the FFT-based transformation. We show that the frequency domain allows us for tight control of the resource usage, boosts performance for big filters, and its properties make it very attractive for the adversarial robustness or interpretability of the internal mechanisms of neural networks.

We show that different compression techniques, for example, FFT, SVD, or bit-level compression, as well as random perturbations (Gauss, Uniform, or Laplace noise) lead to similar gains in robustness. Their underlying mechanism is very similar and can be summarized as inducing enough distortion to cover specific adversarial noise but also retain high accuracy of models on clean data. Most of these perturbation-based defenses are not differentiable and the gradient is hard to approximate, especially if the perturbation techniques are applied in many internal layers of neural networks. The perturbation techniques can be applied during training and inference or exclusively during inference time. The accuracy of a model with perturbation-based defenses is a few percentage points lower on clean data. The input perturbation-based defenses are not robust against adaptive attacks, however, the attacks are much less successful (including their adaptive versions) when we include the perturbation layers across a network in internal layers and enable the defense during model training. We find that pre-trained transformers are more robust to out-of-distribution examples than previous NLP models, such as LSTMs.

The band-limited models can be successfully applied in many domains. We present one of the applications that allow us to design more fair co-existence between Wi-Fi Access Points and LTE-U Base Stations. Our method achieves much higher accuracy than previous energy-based and auto-correlation approaches.

7.2 Future Work and my Insights

Neural networks have a large number of parameters with the latest pre-trained transformers reaching the limits of memory size of a single GPU. For example, BERT Large has 334 millions of parameters and requires a 4-day training on 16 cloud TPUs. There are pros and cons of the big sizes of neural networks. From the perspective of robust training, we observe that injecting different forms of noise into networks and training such perturb models is still possible. The noise is adjusted so that we do not hurt the final model accuracy and make the neural networks more robust. Injecting the same amount of noise only during inference causes substantial degradation of the model's performance (in most cases the model becomes a random classifier). The disadvantage of large networks is that they require long and costly training. Moreover, these models can be run on server-class machines and require significant compression efforts so that they can fit on a mobile device while ensuring that their accuracy drop is acceptable. As a side effect of the compression process, the robustness of these models usually decreases.

The compression efforts of neural networks are carried out on many different granularity levels. The most coarse-grained level usually removes the whole layers or filter channels in CNNs. We can also classify the distillation techniques in this category, which train smaller specialized models on full probability distributions of bigger teacher models. For instance, big transformers are pre-trained on a large corpus of text (RoBERTa is trained on 160 GB of text data and the model contains 355 million parameters). Subsequently, they are fine-tuned on specialized down-stream tasks. Then, a much smaller LSTM model is trained based on probability outputs from the pre-trained and fine-tuned transformer that targets the same down-stream task, such as movie review classification. The task-specific BiLSTM trained on output distribution (logits) from BERT Large for the SST-2 dataset, has about 1 million parameters (about 300X fewer parameters than BERT Large) and its inference time is about 2.5 sec compared to about 1000 sec for BERT Large, thus it is about 400X faster [161].

The more fine-grained models provide weights for each filter that are also trained. The compression follows the training process. It starts removing the filter layers from the ones with

the smallest values and proceeds toward large weight values until the target compression rate is achieved. Some very fine-grained methods propose to train a binary or a weight mask that for each element in a feature map assigns a binary or float weight. Again, based on the target compression rate, we zero-out the elements starting from the ones with the smallest float weights. All of these methods point to the conclusion that there is a non-negligible amount of redundancy in neural networks. Depending on the final required accuracy, dataset, target hardware, and performance level, different methods offer a wide space of trade-offs that have to be carefully navigated to reach the desired goal.

The compression in the frequency domain has also benefits in analyzing neural networks in the Fourier domain. We find that we can track the training progress and identify which frequency coefficients are leveraged by a network when we constrain it to use a specific limit of power, which is expressed as the squared sum of all coefficients. The power is accumulated only in the lowest frequency levels and as the training unfolds, the power is spread out to more frequency coefficients.

Robustness is currently one of the most interesting subfields of deep learning. It is concerned not only with attacks and defenses but also provides many interesting insights into the internal mechanisms of neural networks. It is rather surprising to realize how fragile the high accuracy is. The problem lies in learning non-robust or spurious features that provide high accuracy on the train and test data but perform poorly on slightly modified data, for example, due to a natural distributional shift. The current big NLP models, the pre-trained transformers, seem to be paving a new way into more robust models, where the trend is that the bigger models are pre-trained on bigger and more diverse datasets and usually provide higher levels of robustness. If this trend is giving sufficiently large gains without too low diminishing returns, the already big neural networks might become even larger and will require even more effort in their compression as well as distributed training.

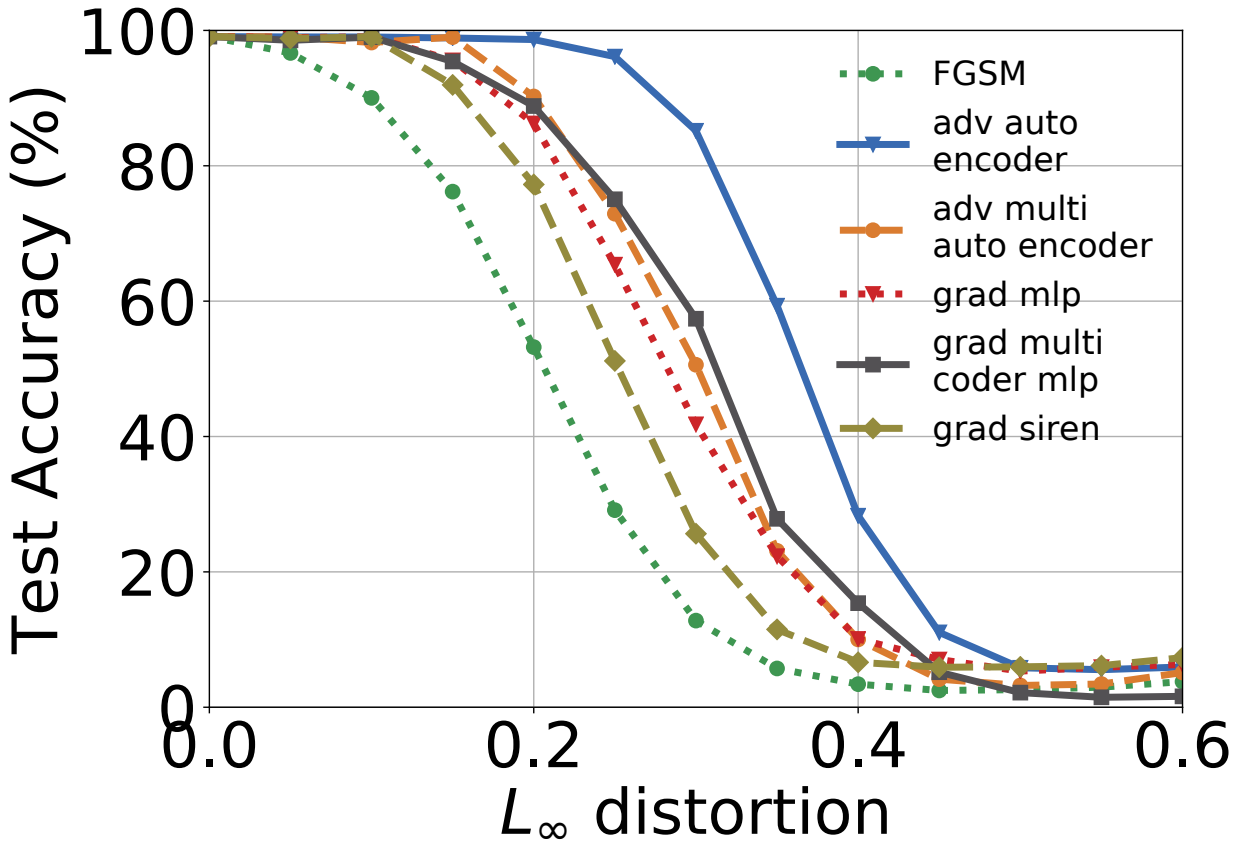

Figure 7.1: *Imitation of the FGSM attack.*

7.2.1 Learning Adversarial Examples

In our future work, we will explore the problem of learning adversarial examples. We present our preliminary results in this thesis.

We investigate if it is possible to teach a neural network to generate adversarial examples. The problem reduces to finding an approximation of the gradient of the attacked model. We teach the adversarial neural network what the gradients are on the train data. Then, we want the network to predict the gradients on the test data. The adversarial example x' found using the FGSM attack is based on the gradient of the loss with respect to the input image x:

$$x' = x + \varepsilon \nabla_x L(x, y, \theta) \tag{7.1}$$

where ε is the maximum allowed distortion, L is the loss function, y is the correct label, and

θ are parameters of the attacked model. The gradient $\nabla_x L$ is required to be provided by the adversarial network. We find that this task is non-trivial since it requires a precise output. Standard CNNs, auto-encoder, or even MLP (Multi-Layer Perceptron) networks are not suitable for this task. Their performance can be slightly improved when provided separate models for each class, hence generating a gradient for a given image within a given class. We find that a specific type of MLP with sinusoidal activation functions (SIREN) [151] provides more precise gradients than the other architectures.

We present our experimental results in Figure 7.1. The FGSM label in the graph denotes the standard FGSM attack. The labels with prefix *adv* generate full adversarial examples while the labels with prefix *grad* generate only the approximations of gradients that are substituted for the gradients in the FGSM attack as presented in equation 7.1.

What is the advantage of generating adversarial examples via neural networks? The final goal is to devise a black-box attack, which leverages the approximate gradients and uses a sequence of decisions based on RL algorithms. The main foundation of this attack is the appropriate architecture to approximate the gradients as precisely as possible.

7.2.2 Machine Learning for the Wireless Domain

Different types of Machine Learning algorithms can be used to address problems in the wireless domain. In this thesis, we compared the performance of many Machine Learning algorithms to conventional methods used in the wireless community and showed superior performance of the ML approaches in multiple configurations. We believe that our results will find applications beyond LTE-U duty-cycle adaptation, for example, in better Wi-Fi frequency management.

We aim to extend our approach by distinguishing between LTE-LAA BS and Wi-Fi APs for the coexistence scenario between Wi-Fi AP, LTE-U BS, and LTE-LAA BS, thus enabling even finer duty cycle adjustments of an LTE-U BS and improved coexistence with Wi-Fi APs. Additionally, we are interested in developing a Machine Learning framework that predicts the type of Wi-Fi traffic *i.,e.,* voice, video, or data which in turn can further ensure fair access to the unlicensed

spectrum since each traffic-type requires different transmission opportunity times (TXOPs) and per-traffic fairness is vital. Similar concepts can also be applied to LTE-LAA BS and Wi-Fi APs coexistence deployments and future NR-U BS and Wi-Fi APs coexistence in the 6 GHz band.

7.2.3 Data for Machine Learning

Machine Learning (ML) is adopted on a massive scale nowadays. Users want to analyze large amounts of data and make informed decisions. These trends challenged well-entrenched database systems and spurred a significant research as well as industrial efforts to create new systems for machine learning. ML can already serve us in a diverse array of applications. It has potential to cater to more people, thus we need to further explore new areas where it can be applied. However, for ML to continue its growth and be widely adopted, it has to address performance issues and decrease the vulnerabilities inherent in ML platforms — particularly in terms of altering, corrupting or deceiving these systems. The data for machine learning and analysis has to be efficiently managed, indexed [50], transformed, and migrated between diverse set of systems [43, 44, 45, 47, 61, 62, 113, 115, 121].

REFERENCES

[1] Imran Latif (Quantenna). Efficient and fair medium sharing enabled by a common preamble. http://grouper.ieee.org/groups/802/11/Workshops/2019-July-Coex/Quantenna_Contribution.pdf, 2019.

[2] R. Bendlin (AT&T). Common preamble design in the 6 GHz band – merits and challenges. http://grouper.ieee.org/groups/802/11/Workshops/2019-July-Coex/att_coex_ws_final.pdf, 2019.

[3] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In *International conference on foundations of data organization and algorithms*, pages 69–84. Springer, 1993.

[4] Ali Al-Shuwaili and Osvaldo Simeone. Energy-efficient resource allocation for mobile edge computing-based augmented reality applications. *IEEE Wireless Communications Letters*, pages 398–401, 2017.

[5] Dan Alistarh, Christopher De Sa, and Nikola Konstantinov. The convergence of stochastic gradient descent in asynchronous shared memory. *arXiv preprint arXiv:1803.08841*, 2018.

[6] Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*, 2018.

[7] Touheed Anwar Atif, Anand M Baswade, Bheemarjuna Reddy Tamma, and A Antony Franklin. A complete solution to LTE-U and Wi-Fi hidden terminal problem. *IEEE Transactions on Cognitive Communications and Networking*, pages 920–934, 2019.

[8] Ayse Elvan Aydemir, Alptekin Temizel, and Tugba Taskaya - Temizel. The effects of JPEG and JPEG2000 compression on attacks using adversarial examples. *CoRR*, abs/1803.10418, 2018.

[9] Mitali Bafna, Jack Murtagh, and Nikhil Vyas. Thwarting adversarial examples: An l_0-robust sparse fourier transform. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10075–10085. Curran Associates, Inc., 2018.

[10] Baingio Pinna. *Pinna illusion*, 2009 (accessed June 1, 2020). `http://www.scholarpedia.org/article/Pinna_illusion`.

[11] Suzan Bayhan and Gürkan Gür. IEEE Tutorial on Machine Learning for Spectrum Sharing in Wireless Networks. 2019.

[12] Mikolaj Bińkowski, Gautier Marti, and Philippe Donnat. Autoregressive convolutional neural networks for asynchronous time series. *ICML (under review)*, 2018.

[13] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[14] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *ArXiv*, abs/1604.07316, 2016.

[15] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015.

[16] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.

[17] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017.

[18] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*, 2018.

[19] Miguel Campo, Zhengxing Chen, Luke Kung, Kittipat Virochsiri, and Jianyu Wang. Band-limited soft actor critic model, 2020.

[20] Cristina Cano and Douglas J Leith. Unlicensed LTE/WiFi coexistence: Is LBT inherently fairer than CSAT ? In *Proc. of IEEE International Conference on Communications*, pages 1–6, 2016.

[21] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, May 2017.

[22] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM, 2017.

[23] Eugene Chai, Karthik Sundaresan, Mohammad A Khojastepour, and Sampath Rangarajan. LTE in unlicensed spectrum: Are we there yet? In *Proc. of ACM 22nd Annual International Conference on Mobile Computing and Networking*, pages 135–148, 2016.

[24] Fabiano S Chaves, Erika PL Almeida, Robson D Vieira, Andre M Cavalcante, Fuad M Abinader, Sayantan Choudhury, and Klaus Doppler. LTE UL power control for the improvement of LTE/Wi-Fi coexistence. In *Proc. of IEEE 78th vehicular technology conference (VTC Fall)*, pages 1–6, 2013.

[25] Mingzhe Chen, Ursula Challita, Walid Saad, Changchuan Yin, and M é rouane Debbah. Artificial neural networks-based machine learning for wireless networks: A tutorial. *IEEE Communications Surveys & Tutorials*, 2019.

[26] Qimei Chen, Guanding Yu, and Zhi Ding. Optimizing unlicensed spectrum sharing for LTE-U and Wi-Fi network coexistence. *IEEE Journal on Selected Areas in Communications*, pages 2562–2574, 2016.

[27] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[28] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Compressing convolutional neural networks in the frequency domain. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1475–1484. ACM, 2016.

[29] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. The ucr time series classification archive, July 2015. `www.cs.ucr.edu/~eamonn/time\_series\_data/`.

[30] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. The ucr time series classification archive. 2015.

[31] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[32] VNI CISCO. CISCO visual networking index: Forecast and trends, 2017–2022. *White Paper*, 1, 2018.

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

[34] Federal Communications Commission. Notice of proposed rulemaking on unlicensed use of the 6 GHz band. `https://docs.fcc.gov/public/attachments/FCC-18-147A1.pdf`, 2018.

[35] James Cooley and John Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[36] Ido Dagan, Oren Glickman, and Bernardo Magnini. The PASCAL recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, 2005.

[37] Nilaksh Das, Madhuri Shanbhogue, Shang -Tse Chen, Fred Hohman, Li Chen, Michael E. Kounavis, and Duen Horng Chau. Keeping the bad guys out: Protecting and vaccinating deep learning with JPEG compression. *CoRR*, abs/1705.02900, 2017.

[38] Nilaksh Das, Madhuri Shanbhogue, Shang -Tse Chen, Fred Hohman, Siwei Li, Li Chen, Michael E. Kounavis, and Duen Horng Chau. Shield: Fast, practical defense and vaccination for deep learning using JPEG compression. *CoRR*, abs/1802.06816, 2018.

[39] Christopher De Sa, Megan Leszczynski, Jian Zhang, Alana Marzoev, Christopher R Aberger, Kunle Olukotun, and Christopher R é. High-accuracy low-precision training. *arXiv preprint arXiv:1803.03383*, 2018.

[40] Pedro M de Santana, Vicente A de Sousa, Fuad M Abinader, and José M de C Neto. Dmcsat: a LTE-U/Wi-Fi coexistence solution based on reinforcement learning. *Telecommunication Systems*, pages 1–12, 2019.

[41] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 2012.

[42] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[43] A. Dziedzic, A. J. Elmore, and M. Stonebraker. Data transformation and migration in polystores. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, 2016.

[44] Adam Dziedzic. Data Loading, Transformation and Migration for Database Management Systems. 2017.

[45] Adam Dziedzic, Manos Karpathiotakis, Ioannis Alagiannis, Raja Appuswamy, and Anastasia Ailamaki. DBMS Data Loading: An Analysis on Modern Hardware. In *Data Management on New Hardware*, pages 95–117, Cham, 2017. Springer International Publishing.

[46] Adam Dziedzic and Sanjay Krishnan. Analysis of random perturbations for robust convolutional neural networks, 2020.

[47] Adam Dziedzic and Jan Mulawka. Analysis and comparison of NoSQL databases with an introduction to consistent references in big data storage systems. In Ryszard S. Romaniuk, editor, *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2014*, volume 9290, pages 855 – 863. International Society for Optics and Photonics, SPIE, 2014.

[48] Adam Dziedzic, John Paparrizos, Sanjay Krishnan, Aaron Elmore, and Michael Franklin. Band-limited training and inference for convolutional neural networks. In *Proc. of International Conference on Machine Learning*, pages 1745–1754, 2019.

[49] Adam Dziedzic, Vanlin Sathya, Muhammad Iqbal Cholilur Rochman, Monisha Ghosh, and Sanjay Krishnan. Machine Learning Enabled Spectrum Sharing in Dense LTE-U/Wi-Fi Coexistence Scenarios. *IEEE Open Journal of Vehicular Technology*, 1:173–189, 2020.

[50] Adam Dziedzic, Jingjing Wang, Sudipto Das, Bolin Ding, Vivek R. Narasayya, and Manoj Syamala. Columnstore and B+ Tree - Are Hybrid Physical Designs Important? In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 177–190, New York, NY, USA, 2018. Association for Computing Machinery.

[51] Gintare Karolina Dziugaite, Zoubin Ghahramani, and Daniel M Roy. A study of the effect of jpg compression on adversarial images. *arXiv preprint arXiv:1608.00853*, 2016.

[52] Micha ë l Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *CoRR*, abs/1312.5851, 2013.

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

[54] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, SIGMOD '94, pages 419–429, New York, NY, USA, 1994. ACM.

[55] fbcunn - Deep Learning CUDA Extensions from Facebook AI Research. `https://goo.gl/jwBJYt`, 2018.

[56] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[57] FFTW. `https://fftw.org`, 2018.

[58] LTE-U Forum. *LTE-U CSAT Procedure TS V1.0*, 2015.

[59] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2017.

[60] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937.

[61] V Gadepally, K O'Brien, A Dziedzic, A Elmore, J Kepner, S Madden, T Mattson, J Rogers, Z She, and M Stonebraker. September 2017. BigDAWG Version 0.1. *IEEE High Performance Extreme*.

[62] Vijay Gadepally, Kyle O'Brien, Adam Dziedzic, Aaron Elmore, Jeremy Kepner, Samuel Madden, Tim Mattson, Jennie Rogers, Zuohao She, and Michael Stonebraker. BigDAWG version 0.1. *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep 2017.

[63] Gopinath Gampala and CJ Reddy. Massive mimo beyond 4G and a basis for 5G. In *Proc. of IEEE International Applied Computational Electromagnetics Society Symposium (ACES)*, pages 1–2, 2018.

[64] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[65] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[66] Lionel Gueguen, Alex Sergeev, Ben Kadlec, Rosanne Liu, and Jason Yosinski. Faster neural networks straight from jpeg. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 3937–3948. Curran Associates, Inc., 2018.

[67] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering Adversarial Images using Input Transformations. *arXiv e-prints*, page arXiv:1711.00117, Oct 2017.

[68] Weiyu Guo and Yidong Ouyang. Robust Learning with Frequency Domain Regularization, 2020.

[69] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[70] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, pages 349–360, 2009.

[71] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. of IEEE international conference on computer vision*, pages 1026–1034, 2015.

[72] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[73] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. Adversarial example defense: Ensembles of weak defenses are not strong. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[74] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[75] Zhezhi He, Adnan Siraj Rakin, and Deliang Fan. Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[76] Dan Hendrycks, Xiaoyuan Liu, Eric Wallace, Adam Dziedzic, Rishabh Krishnan, and Dawn Song. Pretrained transformers improve out-of-distribution robustness. *ACL*, 2020.

[77] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[78] Sepp Hochreiter and J ü rgen Schmidhuber. Long short-term memory. In *Neural Computation*, 1997.

[79] Shengyuan Hu, Tao Yu, Chuan Guo, Wei-Lun Chao, and Kilian Q Weinberger. A new defense against adversarial images: Turning a weakness into a strength. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 1633–1644. Curran Associates, Inc., 2019.

[80] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[81] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[82] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[83] Ian Goodfellow. *Adversarial Examples and Adversarial Training*, 2016 (accessed March 12, 2020). `http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture16.pdf`.

[84] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems 32*, pages 125–136. Curran Associates, Inc., 2019.

[85] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceed-*

ings of the 32nd International Conference on Machine Learning, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[86] Mehdi Jafarnia-Jahromi, Tasmin Chowdhury, Hsin-Tai Wu, and Sayandev Mukherjee. Ppd: Permutation phase defense against adversarial examples in deep learning. *arXiv preprint arXiv:1812.10049*, 2018.

[87] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proc. of ACM/IEEE 44th Annual International Symposium on Computer Architecture*, pages 1–12, 2017.

[88] Herman Kamper, Weiran Wang, and Karen Livescu. Deep convolutional acoustic word embeddings using word-pair side information. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4950–4954, 2016.

[89] Jong Hwan Ko, Burhan Mudassar, Taesik Na, and Saibal Mukhopadhyay. Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation. In *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, pages 59:1–59:6, New York, NY, USA, 2017. ACM.

[90] Mikolaj Bi´ n kowski, Gautier Marti, and Philippe Donnat. Autoregressive convolutional neural networks for asynchronous time series. *CoRR*, abs/1703.04122, 2017.

[91] Sanjay Krishnan, Adam Dziedzic, and Aaron J Elmore. Deeplens: Towards a visual data management system. *arXiv preprint arXiv:1812.07607*, 2018.

[92] Sanjay Krishnan, Aaron J. Elmore, Michael Franklin, John Paparrizos, Zechao Shang, Adam Dziedzic, and Rui Liu. Artificial intelligence in resource-constrained and shared environments. *SIGOPS Oper. Syst. Rev.*, 53(1):1–6, July 2019.

[93] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, The University of Toronto, 2009.

[94] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[95] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016.

[96] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2019.

[97] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4013–4021, 2016.

[98] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4013–4021, 2016.

[99] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[100] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified Robustness to Adversarial Examples with Differential Privacy. *arXiv e-prints*, page arXiv:1802.03471, Feb 2018.

[101] Sheng R. Li, Jongsoo Park, and Ping Tak Peter Tang. Enabling sparse winograd convolution by native pruning. *CoRR*, abs/1702.08597, 2017.

[102] Xingyu Liu, Jeff Pool, Song Han, and William J Dally. Efficient sparse-winograd convolutional neural networks. *arXiv preprint arXiv:1802.06367*, 2018.

[103] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 381–397, Cham, 2018. Springer International Publishing.

[104] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa : A robustly optimized BERT pretraining approach. *ArXiv*, abs/1907.11692, 2019.

[105] Zihao Liu, Qi Liu, Tao Liu, Nuo Xu, Xue Lin, Yanzhi Wang, and Wujie Wen. Feature distillation: Dnn-oriented jpeg compression against adversarial examples. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[106] Xuezhe Ma and Eduard H. Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *CoRR*, abs/1603.01354, 2016.

[107] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *ACL*, 2011.

[108] Vijay Madisetti. *The Digital Signal Processing Handbook*. CRC Press, Inc., USA, 2nd edition, 2009.

[109] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[110] Vasilis Maglogiannis, Dries Naudts, Adnan Shahid, and Ingrid Moerman. A Q-learning scheme for fair coexistence between LTE and Wi-Fi in unlicensed spectrum. *IEEE Access*, pages 27278–27293, 2018.

[111] Vasilis Maglogiannis, Adnan Shahid, Dries Naudts, Eli De Poorter, and Ingrid Moerman. Enhancing the Coexistence of LTE and Wi-Fi in Unlicensed Spectrum Through Convolutional Neural Networks. *IEEE Access*, pages 28464–28477, 2019.

[112] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

[113] Tim Mattson, Vijay Gadepally, Zuohao She, Adam Dziedzic, and Jeff Parkhurst. Demonstrating the BigDAWG Polystore System for Ocean Metagenomics Analysis. 2017.

[114] David McAllester. Fundamentals of Deep Learning. `https://mcallester.github.io/ttic-31230/`, 2020. [Online; accessed 19-March-2020].

[115] J. Meehan, S. Zdonik, Shaobo Tian, Yulong Tian, N. Tatbul, A. Dziedzic, and A. Elmore. Integrating real-time and batch processing in a polystore. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2016.

[116] M. Mehrnoush, S. Roy, V. Sathya, and M. Ghosh. On the fairness of Wi-Fi and LTE-LAA coexistence. *IEEE Transactions on Cognitive Communications and Networking*, Dec 2018.

[117] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David Garc í a, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. *CoRR*, abs/1710.03740, 2017.

[118] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[119] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., USA, 1 edition, 1997.

[120] Peter Nemenyi. Distribution-free multiple comparisons. In *Biometrics*, volume 18, page 263. INTERNATIONAL BIOMETRIC SOC 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210, 1962.

[121] Kyle OBrien, Vijay Gadepally, Jennie Duggan, Adam Dziedzic, Aaron Elmore, Jeremy Kepner, Samuel Madden, Tim Mattson, Zuohao She, and Michael Stonebraker. Bigdawg polystore release and demonstration, 2017.

[122] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *arXiv preprint arXiv:1511.04508*, 2015.

[123] Larry L. Peterson and Bruce S. Davie. *Computer Networks, Fifth Edition: A Systems Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.

[124] Aaditya Prakash, Nick Moran, Solomon Garber, Antonella DiLillo, and James Storer. Deflecting adversarial attacks with pixel deflection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[125] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[126] 3G Partnership Project. 3GPP release 13 specification. `http://www.3gpp.org/release-13/`, 2015. Accessed: 2019-12-12.

[127] Nasim Rahaman, Devansh Arpit, Aristide Baratin, Felix Draxler, Min Lin, Fred A Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of deep neural networks. *arXiv preprint arXiv:1806.08734*, 2018.

[128] Nazanin Rastegardoost and Bijan Jabbari. A machine learning algorithm for unlicensed LTE and Wi-Fi spectrum sharing. In *Proc. of IEEE International Symposium on Dynamic Spectrum Access Networks*, pages 1–6, 2018.

[129] Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*, 2017.

[130] Rebecca Willet and Yuxin Chen. *Machine Learning*, 2020 (accessed June 3, 2020). https://voices.uchicago.edu/machinelearning/stats37710-cmsc35400-s20/.

[131] V. G. Reju, S. N. Koh, and I. Y. Soon. Convolution using discrete sine and cosine transforms. *IEEE Signal Processing Letters*, 14(7):445–448, July 2007.

[132] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen Keckler. vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design. pages 1–13, 10 2016.

[133] Oren Rippel, Jasper Snoek, and Ryan P Adams. Spectral representations for convolutional neural networks. In *Advances in neural information processing systems*, pages 2449–2457, 2015.

[134] Oren Rippel, Jasper Snoek, and Ryan P. Adams. Spectral representations for convolutional neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2449–2457, Cambridge, MA, USA, 2015. MIT Press.

[135] Kevin Roth, Yannic Kilcher, and Thomas Hofmann. The odds are odd: A statistical test for detecting adversarial examples. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5498–5507, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[136] Christopher De Sa, Megan Leszczynski, Jian Zhang, Alana Marzoev, Christopher R. Aberger, Kunle Olukotun, and Christopher R é. High-accuracy low-precision training. 2018.

[137] Hadi Salman, Greg Yang, Jerry Li, Pengchuan Zhang, Huan Zhang, Ilya P. Razenshteyn, and Sé bastien Bubeck. Provably robust deep learning via adversarially trained smoothed classifiers. *CoRR*, abs/1906.04584, 2019.

[138] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT , a distilled version of bert: smaller, faster, cheaper and lighter. In *NeurIPS EMC^2 Workshop*, 2019.

[139] V. Sathya, M. Mehrnoush, M. Ghosh, and S. Roy. Analysis of CSAT performance in Wi-Fi and LTE-U coexistence. *Proc. of IEEE ICC Workshops 2018*, May 2018.

[140] Vanlin Sathya, Morteza Mehrnoush, Monisha Ghosh, and Sumit Roy. Association fairness in Wi-Fi and LTE-U coexistence. In *Proc. of IEEE Wireless Communications and Networking Conference*, pages 1–6, 2018.

[141] Vanlin Sathya, Morteza Mehrnoush, Monisha Ghosh, and Sumit Roy. Auto-correlation based sensing of multiple Wi-Fi BSSs for LTE-U CSAT. In *Proc. of IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pages 1–7, 2019.

[142] Vanlin Sathya, Morteza Merhnoush, Monisha Ghosh, and Sumit Roy. Energy detection based sensing of multiple Wi-Fi BSSs for LTE-U CSAT. In *Proc. of IEEE Global Communications Conference*, pages 1–7, 2018.

[143] Vanlin Sathya, Arun Ramamurthy, and Bheemarjuna Reddy Tamma. On placement and dynamic power control of femtocells in LTE HetNets. In *Proc. of IEEE Global Communications Conference*, pages 4394–4399, 2014.

[144] Kaz Sato, Cliff Young, and David Patterson. An in-depth look at google's first tensor processing unit (tpu). *Google Cloud Big Data and Machine Learning Blog*, 12, 2017.

[145] Patrick Schäfer and Ulf Leser. Fast and Accurate Time Series Classification with WEASEL. In *Proc. of ACM on Conference on Information and Knowledge Management*, CIKM '17, USA, 2017.

[146] Patrick Schäfer. Scalable time series classification. *Data Mining and Knowledge Discovery*, 30, 11 2015.

[147] Carl-Johann Simon-Gabriel, Yann Ollivier, Leon Bottou, Bernhard Schö lkopf, and David Lopez-Paz. First-order adversarial vulnerability of neural networks and input dimension. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, Long Beach, California, USA, 09–15 Jun 2019.

[148] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[149] Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In *Advances in Neural Information Processing Systems*, pages 3088–3096, 2015.

[150] Mohit Kumar Singh, Anand M Baswade, and Bheemarjuna Reddy Tamma. Wi-Fi user's video QoE in the presence of duty cycled LTE-U. In *Proc. of the 24th Annual International Conference on Mobile Computing and Networking*, pages 720–722, 2018.

[151] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit Neural Representations with Periodic Activation Functions, 2020.

[152] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, USA, 1997.

[153] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

[154] SFFT: Sparse Fast Fourier Transform. `https://goo.gl/ycSNvJ`, 2018.

[155] Gilbert Strang. *Introduction to linear algebra*, volume 3. Wellesley-Cambridge Press Wellesley, MA, 2005.

[156] Yaohua Sun, Mugen Peng, Yangcheng Zhou, Yuzhe Huang, and Shiwen Mao. Application

of machine learning in wireless networks: Key techniques and open issues. *IEEE Communications Surveys & Tutorials*, 2019.

[157] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices, 2020.

[158] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2014.

[159] Pedro Tabacof and Eduardo Valle. Exploring the space of adversarial images. *CoRR*, abs/1510.05328, 2015.

[160] Junjie Tan, Sa Xiao, Shiying Han, and Ying-Chang Liang. A learning-based coexistence mechanism for LAA-LTE based HetNets. In *2018 IEEE International Conference on Communications*, pages 1–6, 2018.

[161] Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling Task-Specific Knowledge from BERT into Simple Neural Networks, 2019.

[162] Antonio Torralba and Aude Oliva. Statistics of natural image categories. *Network: Computation in Neural Systems*, 14(3):391–412, 2003.

[163] Florian Tram è r, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[164] Florian Tramer and Dan Boneh. Adversarial training and robustness for multiple perturbations. In *Advances in Neural Information Processing Systems 32*, pages 5866–5876. Curran Associates, Inc., 2019.

[165] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses, 2020.

[166] Sathya Vanlin, Adam Dziedzic, Monisha Ghosh, and Sanjay Krishnan. Machine learning based detection of multiple wi-fi bsss for lte-u csat. *International Conference on Computing, Networking and Communications (ICNC)*, 2019.

[167] Nicolas Vasilache, Jeff Johnson, Micha ë l Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A GPU performance evaluation. *ICLR*, abs/1412.7580, 2015.

[168] Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A gpu performance evaluation. *arXiv preprint arXiv:1412.7580*, 2014.

[169] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

[170] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE : A multitask benchmark and analysis platform for natural language understanding. In *ICLR*, 2019.

[171] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Advances in Neural Information Processing Systems*, pages 7686–7695, 2018.

[172] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017.

[173] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *Proc. of IEEE International joint conference on neural networks*, pages 1578–1585, 2017.

[174] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Towards universal paraphrastic sentence embeddings. In *ICLR*, 2016.

[175] S. Winograd. *Arithmetic complexity of computations*. Society for Industrial and Applied Mathematics Philadelphia, Pa, 1980.

[176] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.

[177] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[178] Zhi-Qin J Xu, Yaoyu Zhang, and Yanyang Xiao. Training behavior of deep neural network in frequency domain. *arXiv preprint arXiv:1807.01251*, 2018.

[179] Yuzhe Yang, Guo Zhang, Dina Katabi, and Zhi Xu. ME -net: Towards effective adversarial robustness with matrix estimation. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7025–7034, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[180] Zhewei Yao, Amir Gholami, Qi Lei, Kurt Keutzer, and Michael W Mahoney. Hessian-based analysis of large batch training and robustness to adversaries. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 4949–4959. Curran Associates, Inc., 2018.

[181] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 2130–2141. Curran Associates, Inc., 2019.

[182] Alessio Zappone, Marco Di Renzo, and M é rouane Debbah. Wireless networks design in the era of deep learning: Model-based, AI-based, or both? *arXiv preprint arXiv:1902.02647*, 2019.

[183] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys & Tutorials*, 2019.

[184] Yuchen Zhang and Percy Liang. Defending against whitebox adversarial attacks via randomized discretization. *AISTATS*, 2019.

[185] Yi-Tong Zhou and Rama Chellappa. Computation of optical flow using a neural network. In *ICNN*, pages 71–78, 1988.

[186] Aleksandar Zlateski, Zhen Jia, Kai Li, and Fredo Durand. Fft convolutions are faster than winograd on modern cpus, here is why, 2018.

APPENDIX A

BAND-LIMITED NEURAL NETWORKS

A.1 Details on DenseNet-121 Architecture Trained on CIFAR-100

We train DenseNet-121 (with growth rate 12) on the CIFAR-100 dataset.

In Figure A.1 we show small differences in test accuracy during training between models with different levels of energy preserved for the FFT-based convolution.

In Figure A.2 we show small differences in accuracy and loss between models with different convolution implementations. The results were normalized with respect to the values obtained for the standard convolution used in PyTorch.

A.2 Application of Fast Fourier Transform to Convolution

We implement convolutional layers using different methods and compare their performance. The first implementation of a naive version of convolution (with many for loops - for each data point in the mini-batch, channel, and each dimension in the input) serves as our baseline and the ground truth in terms of the returned value. Next, we implement convolution via FFT. The algorithm is written according to the schema presented in [125] (chapter 13). It is worth noting that the pre-processing steps are crucial for the correct implementation of convolution via FFT. The filter is usually much smaller (than the input) and has to be padded with zeros to the final length of the input signal. The input signal has to be padded on both ends with as many zeros as the size of the filter to prevent the effects of wrapped-around filter data (for example, the last values of convolution should be calculated only from the final overlap of the filter and input signal and should not be polluted with values from the beginning of the input signal).

Another implementation of FFT-based convolution is in Python using *numpy.fft* library. Subsequently, the convolution operation is used to implement the forward and backward pass of the convolutional layer (with a focus on the 1D case). We also test the performance of FFTW [57].

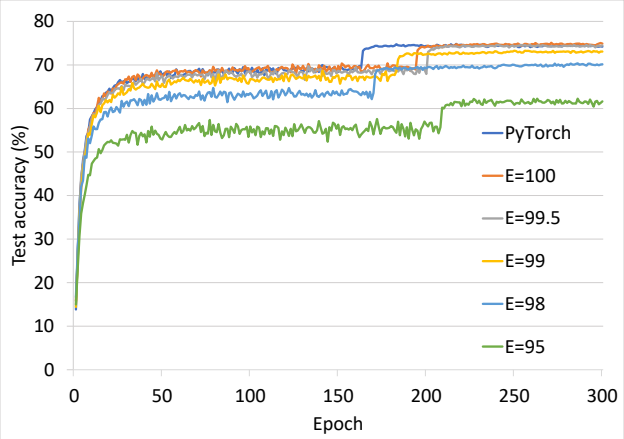

Figure A.1: *Comparing test accuracy during training for CIFAR-100 dataset trained on DenseNet-121 (growth rate 12) architecture using convolution from PyTorch and FFT-based convolutions with different energy rates preserved.*

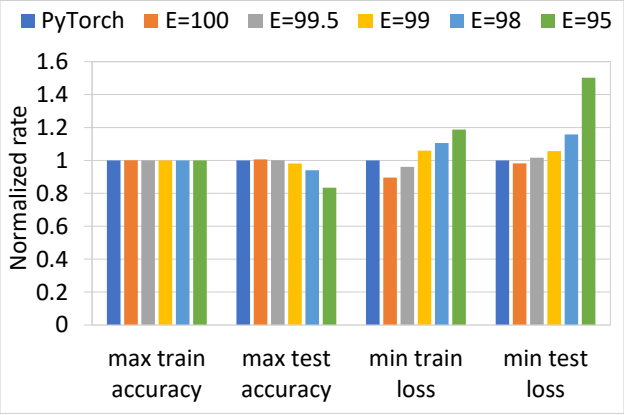

Figure A.2: *Comparing accuracy and loss for test and train sets from CIFAR-100 dataset trained on DenseNet-121 (growth rate 12) architecture using convolution from PyTorch and FFT-based convolutions with different energy rates preserved.*

Interestingly enough, it is significantly slower (even about 10X) for 1D convolution than *numpy.fft*. On the other hand, FFTW showed better performance than *numpy.fft* for 2D convolution. We also examine cross-correlation (which is used in convolutional networks) from scipy. It has an interesting feature exposed to the 1D case. We can choose 3 modes of execution: fft (computing cross-correlation via fft), direct (computation), auto (choose direct or fft method depending on which one is predicted to run faster). Thus, scipy provides a simplified optimizer to select the calculation of convolution (either via FFT or direct method) depending on the cost estimation of each method. Internally, scipy calls fft from *numpy.fft*. There is a crossover point between convolution

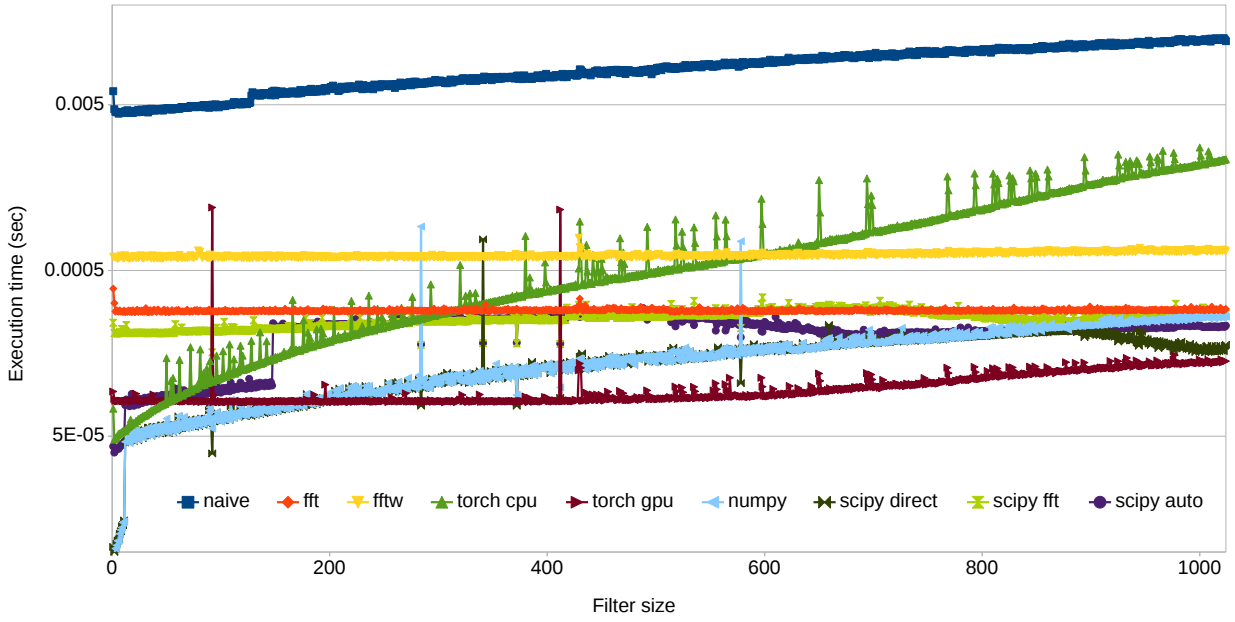

Figure A.3: *A comparison of convolution operations implemented in different frameworks and using various FFT versions for one-dimensional signals.*

via fft and direct one. For example, for a signal of input size 4096, the convolution via fft is faster for filters of size greater than about 300. For very small sizes of filter and input, the overhead of the auto mode (the optimization version in scipy) is large (probably scipy could be changed to always select the direct convolution for the number of required multiplications below 2^{20}).

PyTorch standard CPU implementation of convolution is surprisingly slower than scipy or numpy and even there is a crossover point between our FFT-based convolution and the one from PyTorch. We also test GPU-based convolution from PyTorch and for an input signal of size 4096, it is superior to all other implementations.

The fft based convolution from papers [52, 167] is not incorporated into the standard version of PyTorch but has to be installed separately [55].

A.3 Compression in the Frequency Domain

Our focus is on how to compress a signal and filter in the frequency domain so that the relative error[1] between the output of the convolution and the gold standard [2] is small. To be more precise, we want to compute convolution using FFT. The input to the convolution (e.g. a speech signal, time-series data, or an image) and a filter (also called the kernel) are transformed from the time-domain to the frequency domain via FFT. Then we compress both the signal and the filter by discarding high-frequency coefficients (the intuition is that we want to discard noise that is usually of high frequency). The number of discarded coefficients is chosen based on how much energy we want to preserve from the input signal (usually the filters are smaller than the input so easier to compress, thus we focus on the compression of the input signal). Finally, we apply the inverse FFT to go back to the time domain (see Figure 4.1).

We observe that the relative errors: $\delta = 100\% \frac{|V - V_{\text{approx}}|}{|V| + |V_{\text{approx}}|}$ (where V is the input signal, V_{approx} is the approximation of the input signal after applying compression) between gold standard and output of the convolution via FFT is relatively high (for example, after preserving 99.9% energy of a signal, we discard about 100 coefficients out of 270 and get the relative errors of about 0.03). We check if similar errors are obtained when we compare the input signal with the same signal after its compression in the frequency domain. It occurs that this is the case, hence the compression in the frequency domain causes high numerical (relative) errors. However, the shape of the input signal and the output signal (after compression) are very similar visually (the errors can be discerned only when we zoom-in our graphs).

The sparse FFT (SFFT [154]) shows better performance than FFTW [57] only for very long input signals. The results presented in [154] indicate that SFFT outperforms FFTW for signals of size at least 2^{13}, however, the maximum length of the signal that we experiment within one dimension is 2^{12}. For this reason, we decide not to use the sparse FFT.

We test an end-to-end performance of a simple 3 layer neural network with our FFT-based

1. Relative error definition: `https://en.wikipedia.org/wiki/Approximation\_error`

2. The standard version of convolution computed directly in the time domain.

convolution. The main goal is to check if the relative errors introduced by the compression in the frequency domain have an impact on the training of a model. Our network has the following architecture: *conv - relu - max pool - affine (fully connected) - relu - affine (fully connected) - softmax*. We use 32 one dimensional filters of size 49 each, a single data point is an array of 4096 values with 3 channels, the max pool is with pool width 2 and stride 2. This small network is sufficient to train a model for CIFAR-10 data (with 10 classes) and obtain accuracy above 90%. To simulate 1-dimensional data, we reshape each image from shape 3x32x32 to 3x1024 (the 3 channels are retained and the 32x32 image is translated to an array with 1024 values). The sanity check is for 10 training images. The loss is decreasing in the same way for uncompressed and compressed versions of convolution. Next, we test our version of the convolution on bigger data sizes. We run 15 epochs for different variants of our convolutional layer. For example, when no compression is used (the output of the convolution had the relative error to the gold standard in the ballpark of about 10^{-14}) the model achieves the training accuracy of about 70% and the validation accuracy of about 45%. When we use our compressed version of convolution, the model is able to achieve the training accuracy of about 55% and the validation accuracy of about 40% (with 98% of the energy of the signal preserved). Moreover, the results for the compressed convolution are fluctuating, and even with more energy preserved we are obtaining worse results. This indicates that the relative errors introduced by the compressed convolution propagate to the training of the models and might demonstrate in a slower learning process.

Finally, we train the 3 layer network for many days to see if the loss of information inhibits the network from learning. In Figure A.4, we present results of many day training, where most models are able to reach 500 epochs. We observe that even the highly compressed signal with only 95% of retained energy is able to achieve high accuracy values of about 90%, however, it takes more epochs than for other models that do not apply compression. The most interesting result is that the pure calculation of convolution in the frequency domain without any compression is achieving 90% accuracy faster than the models which compute convolution directly. It is worth noting that our convolution without any compression in the frequency domain has very small relative errors

(from 10^{-10} to 10^{-12}) in comparison to the results of direct convolution. Our conjecture why it happens is that the FFT based convolution exhibits some smoothing effect. When we go from the time domain to the frequency domain, the Fourier coefficients exhibit symmetry in the frequency domain. We take advantage of it and cut off half of the coefficients to save memory and the number of multiplications. We check if the two halves of the signal in the frequency domain are the same and it occurs that there are only some minor numerical differences of about 10^{-12}. After computing convolution in the frequency domain, we restore the signal by mirroring it and taking its conjugate. This perfectly symmetrical output signal might be a better output than the result of direct convolution with small numerical errors.

The outcome of the non-compressed version of convolution suggests that the compression might have a detrimental effect, on both training time and the number of epochs to reach a certain accuracy value. We obtain an interesting result. In Figure A.4, the models that use compression on both the input signal and the filter, with 95% energy preserved or after removing a single Fourier coefficient, indeed, are learning slower than the models based on direct convolution (lines labeled: train_acc naive and train_acc numpy). On the other hand, the model with 99% of compression applied is learning faster than the models based on direct convolution. The initialization of the models is random (even though we initialize the seed, the slight change in the environment or change in the filter size) does affect the learning process. In another experiment, where we use only a different pool width (5 instead of 3), the model with 98% compressed energy learns faster than a model with 99% compressed energy (at least for the first 15 epochs).

The take-away from this analysis is that for the carefully chosen compression ratio we can learn as fast as standard models (with direct convolution) in terms of the number of epochs, but faster in terms of total execution time (in this case for 99% of preserved energy).

A.3.1 Execution time and errors with compression for convolution

Figure A.5 shows that a very high value of the preserved energy in the input signal corresponds to relatively small absolute error and a non-negligible speed-up of the convolution operation. The

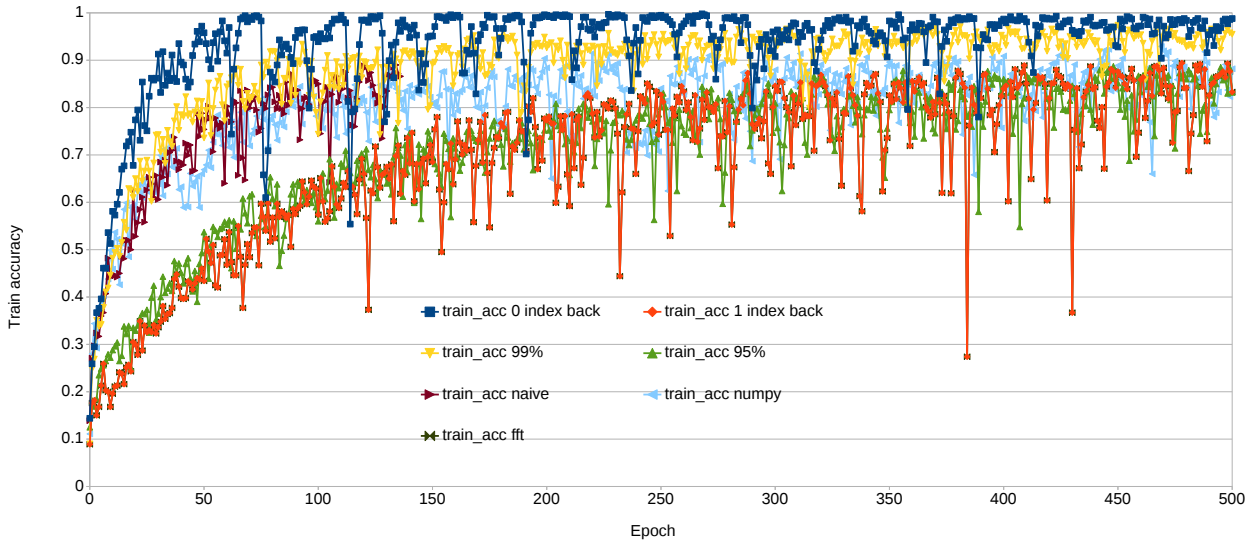

Figure A.4: *Training accuracy with different types of convolutional layers applying various compression ratios (multi-day training).*

experiment confirms our expectation that the higher the compression ratio, the larger the error (between the direct result of convolution and the convolution computed in the frequency domain after compression of the input signal and filter). For very high compression ratios, the error surges and also the performance of the convolution plateaus. The 95% of the preserved energy in the input is the lowest acceptable value for the 50words dataset from the UCR archive.

A.4 Tests for Speech Data

We also train the same network as in Section A.3 for the speech data from the Switchboard corpus of English conversational telephone speech. Data is parameterized as Mel-frequency cepstral coefficients with the first and second-order derivatives. For the training set, we use the set of about 4k word tokens. We leverage the code published in [88] to process the data. We choose tokens that occur at least 14 times in the training set so that the size of the training set allows us to test convolutions with many different compression ratios. The input data is divided into batches of size 50, we treat each data point as a one-dimensional signal with 39 channels and 200 values in each channel. Similarly to CIFAR-10 data we use 32 filters with size 49 each. The number of classes is

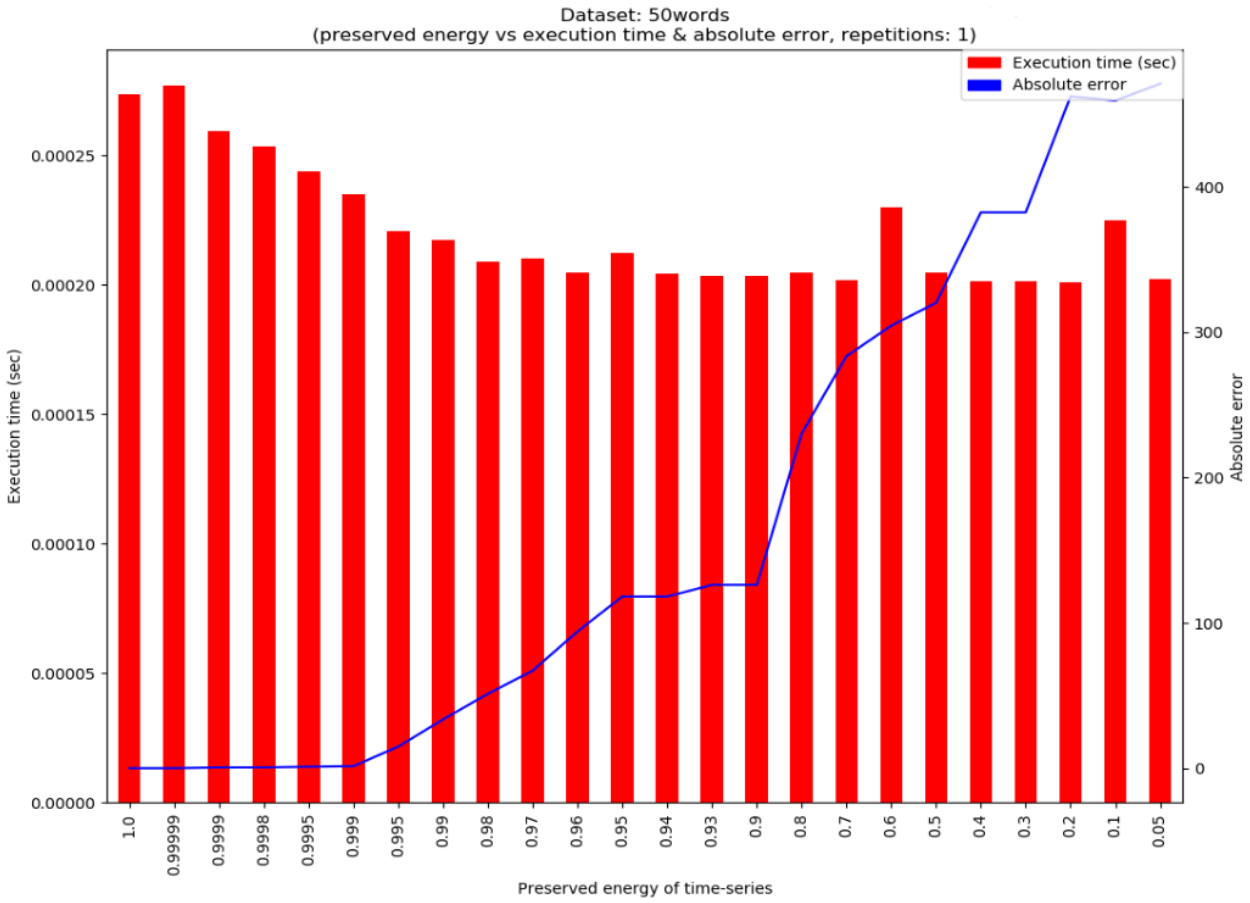

Figure A.5: *Assessing how the preserved energy rate influences the execution time and the final absolute error of the convolution.*

129. The training accuracy above 90% is achieved after 45 epochs (99% after 83 epochs) for the network where convolutions were computed using FFT and without any compression (we do not even smooth the signals via mirroring two halves in the frequency domain). We present the results in Figure A.6.

The results for Switchboard data are better than the ones for CIFAR-10. We observe consistency, the lower level of preserved energy, the longer training. However, we can train as fast as a standard model with 90% preserved energy. This result is probably because of more information per a single data point in the Switchboard dataset. It shows that we can efficiently compress the speech signals and learn much faster with a compressed version of convolution in the frequency domain.

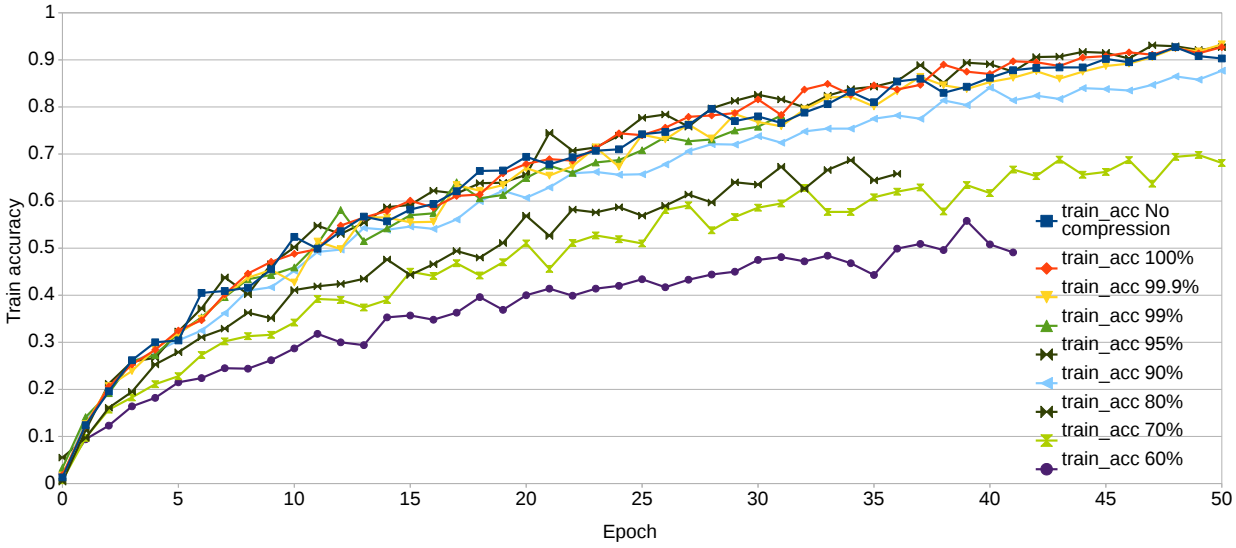

Figure A.6: *Training accuracy with different types of convolutional layers applying various compression ratios for the Switchboard dataset.*

APPENDIX B

PERTURBATION-BASED DEFENSES

B.1 Distributions of Input Perturbations

We emphasize in this thesis that the distribution of noise does not matter as much as the magnitude. To illustrate this point, we plot the distribution of *deltas* for six imprecise channels in Figure B.1. We compute the *deltas* by subtracting an original image from the perturbed adversarial image and plot the histograms of differences. We use an image from the ImageNet dataset. For all the examples, the correct labels were recovered. We use the C&W attack with 1000 iterations and the initial value $c = 0.01$. All of the distributions are varied, yet they achieve similar robustness in the non-adaptive setting.

B.2 Hybrid Input Perturbations

A natural thought is whether input perturbation defenses can be made more effective by combining them. This is an approach that has been applied, for example, in random discretization [184]. Our experiments contrast with the previous work and suggest that there is little benefit to hybrid approaches. We present an illustrative example for pair-wise combinations of Frequency Compression (FC), Color Depth (CD) compression, and Uniform (Unif) noise. Other combinations are also possible but for brevity. We found no to very small benefit to combining for properly tuned channels. We show the recovery rate, i.e. a fraction of original labels recovered. Table B.1 presents results for the Carlini-Wagner attack on the MNIST test set [99], the whole test set of CIFAR-10, and the whole development set of ImageNet.

We believe these results suggest that the noisy channels do not exploit anything inherent to the images. Simply the addition of noise (through reconstruction error) is the mechanism for robustness. Composing two different schemes usually increases this noise.

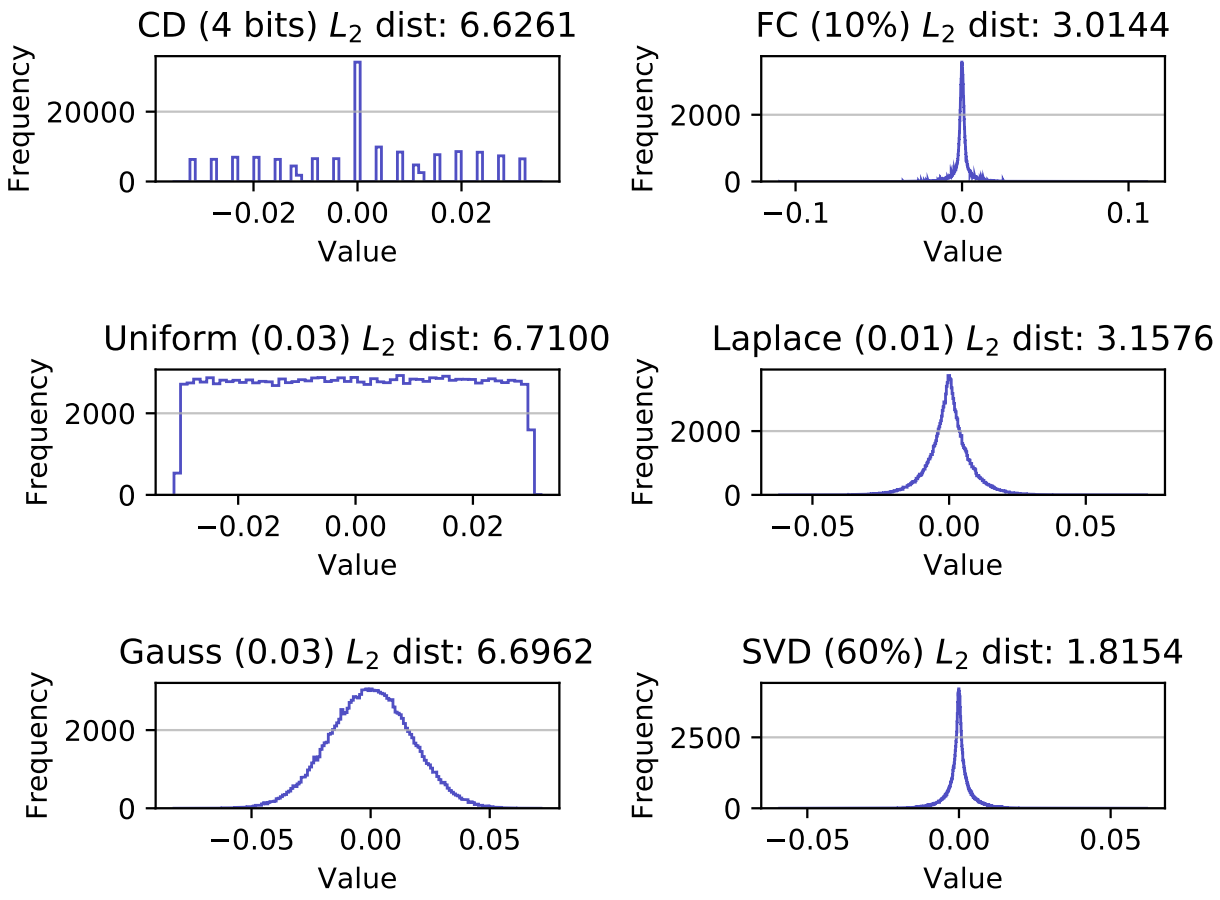

Figure B.1: Distribution of *deltas* for input perturbation defences.

B.3 More Attacks Against Input Perturbations

We further show the performance of the input perturbations against different attacks. In addition to the attacks described in Section 5.5, we also run experiments for the following attacks:

- **LBFGS** minimizes the distance between the input image and the adversarial example as well as the cross-entropy between the predictions for the adversarial and the input image; introduced by [158] and further extended in [159].

- **BIM** L_1 is a modified version of the Basic Iterative Method that minimizes the L_1 distance [95].

- **FGSM** adds the sign of the gradient to the image, gradually increasing the magnitude until

Table B.1: Comparison of hybrid perturbation-based defenses.

	CD (bits)	FC (%)	Unif (ε)	CD+FC	FC+CD	CD+Unif	FC+Unif
MNIST	100	95	100	100	100	100	100
CIFAR-10	84.4(4)	85.2(20)	83.4(0.03)	86.0	83.6	85.18	86.45
ImageNet	80.9(4)	78.7(50)	80.3(0.03)	78.6	77.5	73.15	76.13

the image is misclassified [65].

We show results in Table B.2 and Figure B.2. Given an adversarial input, we pass the input through a channel before prediction. We evaluate the accuracy (%) of the classifier over 1000 images, the accuracy on clean data is 93.5% for CIFAR-10 (the first half of the table) and 83.5% for ImageNet (the other half of the table). For the channels, we report in parentheses: compression used (%), number of bits per value, and the strength of the attack (ε).

Table B.2: White-box attacks against input perturbations.

Attack	FC (%)	CD (bits)	Uniform (ε)	Gauss (ε)
BIM L_1	86.2 (20)	85.1 (4)	82.8 (0.03)	82.1 (0.03)
LBFGS	82.8 (50)	80.2 (4)	79.2 (0.04)	79.3 (0.05)
C&W L_2	85.2 (20)	84.4 (4)	84.3 (0.01)	84.8 (0.02)
FGSM	79.0 (50)	49.2 (4)	49.4 (0.03)	49.9 (0.03)
PGD L_∞	88.6 (10)	84.3 (5)	85.7 (0.01)	84.9 (0.02)
BIM L_1	81.5 (10)	82.0 (4)	81.2 (0.009)	81.2 (0.009)
LBFGS	71.7 (70)	77.6 (4)	76.4 (0.07)	76.5 (0.07)
C&W L_2	78.7 (50)	80.9 (4)	81.4 (0.03)	80.4 (0.03)
FGSM	73.8 (50)	76.0 (4)	75.4 (0.03)	75.4 (0.02)
PGD L_∞	82.1 (5)	82.9 (4)	82.0 (0.007)	80.9 (0.01)

Both frequency (FC) and color depth compression (CD) methods achieve a comparable performance for similar distortion rates. The SVD compression behaves similarly to FC compression up to the L2 distortion levels of 4 and 20, on CIFAR-10 and ImageNet datasets. Then, SVD is even slightly better than FC but worse than CD on CIFAR-10, however, FC is better than SVD and CD on ImageNet data. The Gaussian and Uniform channels exhibit very similar trends.

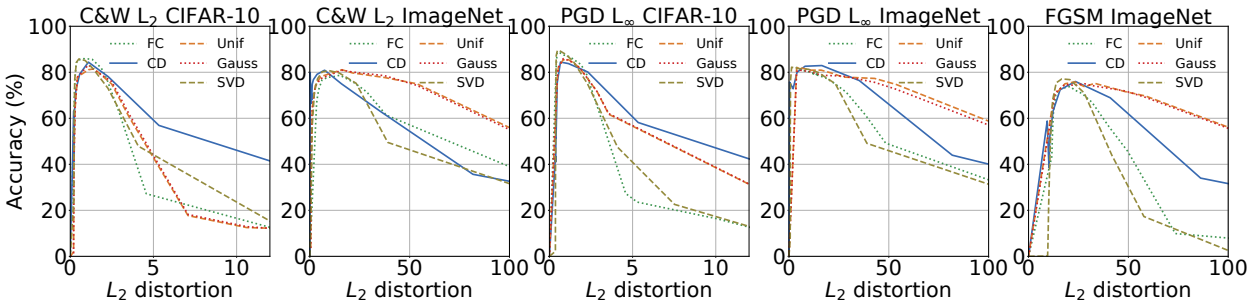

Figure B.2: *Channel distortion* against the test accuracy (%). The distortion of the imprecise channels has to be large enough to recover the correct label but not so large that it degrades model performance. The base test accuracy is about 93.5% for CIFAR-10 and 83.5% for ImageNet on 1000 randomly chosen images. The experiment is run for the C&W L_2 attack with 100 iterations and the PGD attack with 40 iterations (random start).

B.4 Attack Input Perturbation Defenses Non-adaptively

We present experiments for the analysis described in Section 5.6.2.

The attack vector against input perturbation defenses simply makes the adversarial perturbations large enough that the defender significantly hurts the accuracy of the model when trying to dominate the adversarially placed strategic perturbations.

This interplay is visualized in Figure B.3. We use an example from the ImageNet dataset, the ResNet-50 architecture, and set the stochastic channel to the Gaussian noise. We start from an adversarial example generated with the Carlini & Wagner (non-adaptive) L_2 attack (left-hand side of each subplot) and for consecutive subplots (in the top to bottom sequence), we increase the attack strength and incur higher distortion of the adversarial image from the original image. For a single plot, we increase the L_2 distance of the output from the stochastic channel to the adversarial example by increasing the Gaussian noise (controlled by its standard deviation σ and with mean $\mu = 0$). For L_2 distances incurred by different noise levels, we execute 100 predictions. We use the frequency count and report how many times the model predicts the original, adversarial, or other class.

The adversarial class is induced by a non-targeted attack. The **other class** is neither original (correct) nor adversarial and caused by (too much) Gaussian perturbation used in the defense. In this experiment, we begin with an adversarial example and systematically add more Gaussian

noise. The model's predictions change from the adversarial class, through original, and to a random other class.

The plot shows what range of distances from the adversarial image reveals the correct class. For the adversarial examples that are very close to the original image (e.g. adversarial distance of 0.006 for the top figure), the window of recovery (which indicates which strengths of the random noise can recover the correct label) is relatively wide, however, as we increase the distance of the adversarial image from the original image (by increasing the strength of the attack in consecutive plots), the window shrinks and finally we are unable to recover the correct label. Figure 5.1 shows that these large distortion attacks are still imperceptible to the human eye. To avoid any sensitivity to input perturbation defenses, an attacker can eliminate this recovery window by generating a larger distortion attack. Besides, statistical indications of adversarial examples should also be eschewed [135].

B.5 Multiple Trials for Stochastic Perturbations

Another compelling reason to use stochastic perturbations (like Uniform noise) as a defense is that it can be run repeatedly in many random trials. We show that doing so slightly improves the efficacy of the defense. We randomly choose 1000 images from the CIFAR-10 test set. For each of these images, we generate an adversarial attack. We then pass each image through the same stochastic perturbation multiple times. We take the most frequent prediction. Figure B.4 illustrates the results.

Only 16 trials are needed to get a relatively strong defense. We argue that this result is significant. Randomized defenses are difficult to attack. The attacker cannot anticipate which particular perturbation to the model will happen. The downside is the potential of erratic predictions. We show that a relatively small number of trials can greatly reduce this noise. Furthermore, the expense of running multiple trials of a randomized defense is small relative to the expense of synthesizing an attack in the first place.

B.6 More Details on White-Box Adaptive Attacks

The stochastic perturbations are harder to attack with gradient-based adaptive methods. Our strategy is to send output from the adversarial algorithm through the imprecise network and the network at least as many times as set in the defense. We mark the attack as successful if the most frequent output label is different from the ground truth. The more passes through the noisy channel we optimize for, the stronger the attack. Furthermore, we run many iterations of the attack to decrease the L_2 distortion. An attack that always evades the noise injection defense is more difficult to generate because of randomization. The other randomized approach was introduced in dropout [56]. The attacks against randomized defenses require optimization of complex loss functions, incur higher distortion, and the attacks are not fully successful [22]. In the Figure B.5, we present the result of running attacks and defenses on CIFAR-10 data with single and many iterations. The defense with many trials can be drawn to 0% accuracy, however, the defense not fully optimized by the adversary (single noise injection) can result in about 40% or higher accuracy.

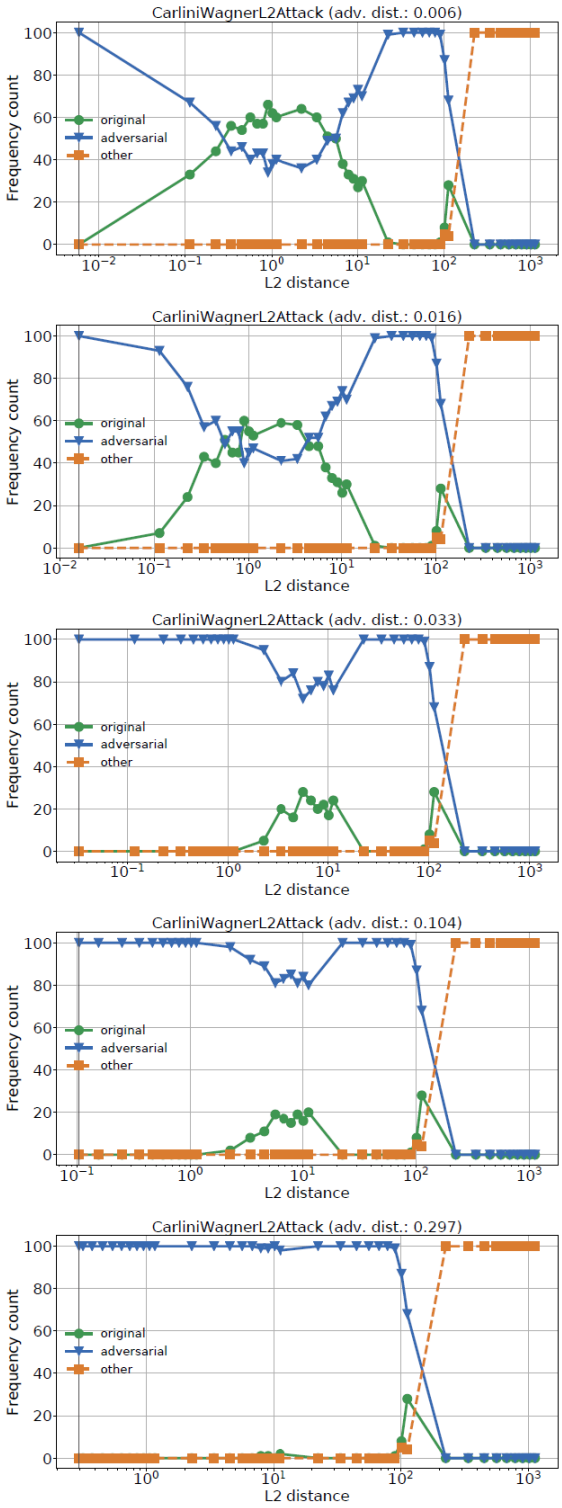

Figure B.3: Frequency of model predictions for original, adversarial, and other classes as we increase the attack strength.

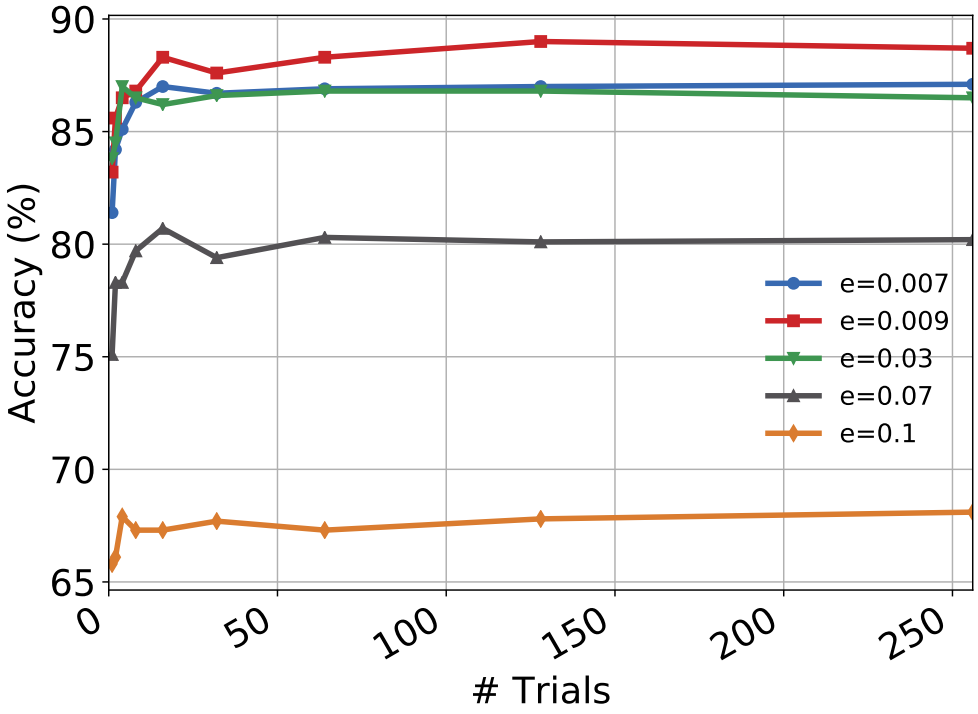

Figure B.4: For the CIFAR-10 dataset, we run multiple trials of the uniform noise perturbation and take the most frequent prediction. We further test multiple noise levels. The multiple trials improve overall accuracy for different noise levels significantly. After 128 trials for the best setting we are within 3% of the overall model accuracy (of about 93.5%).

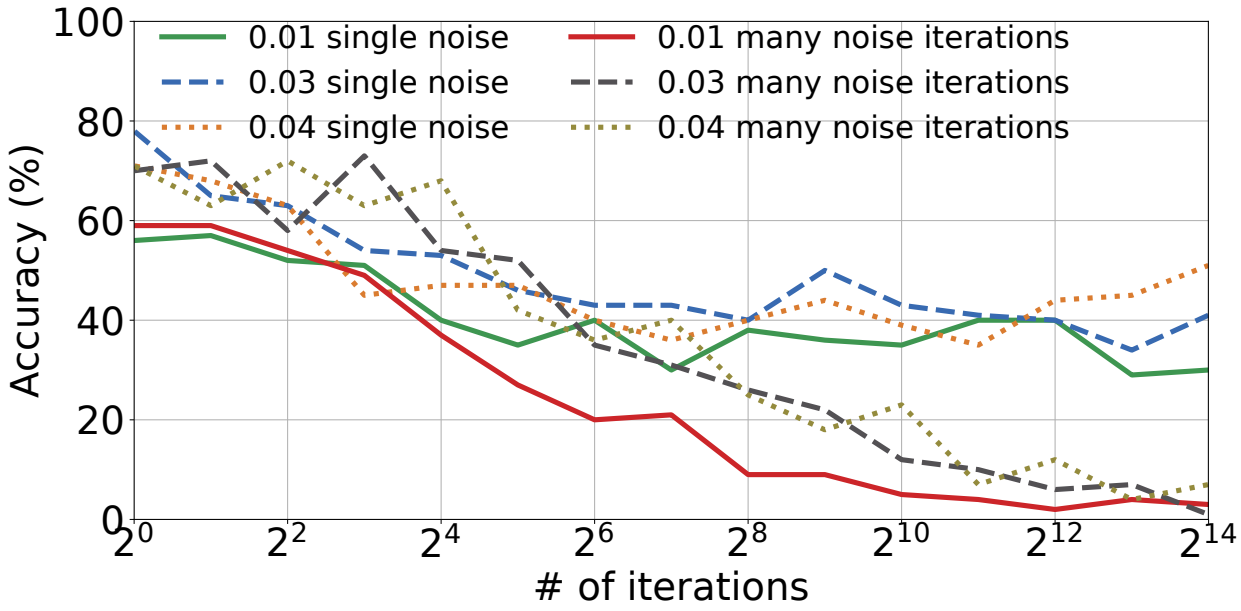

Figure B.5: For the CIFAR-10 dataset, we run multiple trials of the uniform noise perturbation and take the most frequent prediction in the defense (many noise iterations). We also run just a single noise injection and return the predicated label. The attacker runs the same number of many uniform trials as the defender. The experiment is run on 100 images, with 100 C&W L_2 attack iterations.