Improving Convolutional Neural Network Robustness to Adversarial Images Through Image Filtering

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Natalie Elizabeth Bogda

August 2020

Copyright © by Natalie Elizabeth Bogda, 2020
All Rights Reserved.

To my parents – without their sacrifices I would not be the person I am today.

Acknowledgments

I would like to thank my parents for raising me to have a scientific mind. Because of them, I grew up with a sense of curiosity and a desire to learn.

I would like to thank Dr. Amir Sadovnik for being an excellent, involved advisor who cares about my success. His guidance was invaluable towards completing this endeavor. I would like to thank my fellow lab members in the UVU (Understanding Visual Understanding) lab for sharing information about their work with me and giving me direction on where to go with my own. I would like to thank everyone who I worked for during my time as a undergraduate researcher, with special thanks to Dr. Jian Huang, for guiding me on how to perform research. This experience greatly contributed to my graduate degree.

I would like to thank my committee members, Dr. Jian Liu and Dr. Jens Gregor, with special note to Dr. Jens Gregor for his instruction in image processing. I would like to thank the computer science program at the University of Tennessee as a whole. Every course contributed to my knowledge and understanding of research.

I would like to thank my boyfriend Cameron for being a constant source of support and understanding for me as I completed this thesis. Finally, I would like to thank my friends for their patience and encouragement as I completed this work.

Abstract

The field of computer vision and deep learning is known for its ability to recognize images with extremely high accuracy. Convolutional neural networks exist that can correctly classify 96% of 1.2 million images of complex scenes. However, with just a few carefully positioned imperceptible changes to the pixels of an input image, an otherwise accurate network will misclassify this almost identical image with high confidence. These perturbed images are known as *adversarial examples* and expose that convolutional neural networks do not necessarily "see" the world in the way that humans do. This work focuses on increasing the robustness of classifiers to these adversarial examples using image filtering. The goal of this work is to find a middle ground between state-of-the-art accuracy and robustness in defense even as the adversary strength increases. This work focuses on images from three data sets: The MNIST Dataset of Handwritten Digits, CIFAR-10, and a custom data set collected using a cross-reference of ImageNet and CINIC-10. Two types of classifiers are made robust in this work: support vector machines and convolutional neural networks. Two types of adversarial attacks are compared and contrasted as multiple image processing techniques are applied to the affected data sets. For smaller images, it is found that resizing the image or blurring the image using the Gaussian filter are the most effective techniques in increasing robustness. The Gaussian blur improved classification accuracy from 8% to 65% on the strongest attack in the CIFAR-10 data set. For larger images, it was found that a sequence of the median and Gaussian filter masks the adversarial noise most effectively, and is the most robust defense when used in combination with a model trained on images filtered with the same technique. The largest improvement using this method increased image classification accuracy from 7% to 57% when applied to the strongest adversarial attack.

Table of Contents

1 Introduction **1**

1.1 Motivation . . . 2

1.2 Goals . . . 3

2 Background and Literature Review **4**

2.1 Adversarial Attack Methods . . . 4

2.1.1 Fast Gradient Sign Method . . . 5

2.1.2 Projected Gradient Descent . . . 6

2.1.3 Blackbox Attacks . . . 9

2.1.4 Other Attack Methods . . . 11

2.2 Defense Methods . . . 15

2.2.1 Filtering Defenses Used . . . 15

2.2.2 Comparison to Other Filtering Defense Papers . . . 20

2.2.3 Other Defense Methods . . . 22

3 Support Vector Machine Adversarial Attacks **25**

3.1 Methodology . . . 26

3.1.1 Creating Adversarial Images . . . 26

3.1.2 Defense Building . . . 28

3.2 Experiments . . . 29

3.2.1 Unaltered Results . . . 29

3.2.2 Thresholding . . . 31

3.2.3 Gaussian Filtering . . . 31

3.2.4 Median Filtering . . . 34
3.2.5 Bilateral Filtering . . . 34
3.2.6 Resizing . . . 37
3.3 Discussion . . . 37

4 Data Sets and Networks **40**
4.1 Data Collection . . . 40
4.1.1 The MNIST Dataset of Handwritten Digits . . . 41
4.1.2 CIFAR-10 . . . 42
4.1.3 CINIC-10 Large . . . 44
4.2 Convolutional Neural Network Structure . . . 48
4.2.1 MNIST Convolutional Neural Network . . . 48
4.2.2 CIFAR-10 Convolutional Neural Network . . . 52
4.2.3 CINIC-10 Large Convolutional Neural Network . . . 59

5 Defending Against Adversarial Examples **70**
5.1 Defending MNIST . . . 71
5.1.1 Attacks . . . 71
5.1.2 Defenses . . . 76
5.2 Defending CIFAR-10 . . . 85
5.2.1 Attacks . . . 85
5.2.2 Defenses . . . 92
5.3 Defending CINIC-10 Large . . . 98
5.3.1 Attacks . . . 99
5.3.2 Defenses . . . 105

6 Conclusions **118**

Bibliography **122**

Vita **128**

List of Figures

2.1 An example of the FGSM attack. Difference image multiplied by 10. 7
2.2 An example of the PGD attack. Difference image multiplied by 10. 10

3.1 Figure from *Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples* [38] 26
3.2 Example of an adversarial image created from an example in class '2'. The left shows the original image and its prediction from the SVM. The right shows the adversarial image. Although similar to the human eye, the SVM classifies the image on the right as belonging to class '3' with 99% confidence. 28
3.3 A set of unaltered adversarial images and their original image counterparts. . 30
3.4 The results from thresholding both the adversarial and original images. . . . 32
3.5 The results from convolving both the adversarial and original images with the Gaussian kernel. 33
3.6 The results from filtering both the adversarial and original images with the median filter. 35
3.7 The results from filtering both the adversarial and original images with the bilateral filter. 36
3.8 The results from resizing both the adversarial and original images. 38

4.1 The distribution of the MNIST data set of handwritten digits in both the training and testing sets. 41
4.2 MNIST data set example by Lim et al. [33] 42
4.3 The distribution of the CIFAR-10 data set in both the training and testing sets. 43

4.4 An example from the authors' website of the contents of the CIFAR-10 data set [27]. . . . 45
4.5 The CINIC-10 Large data set distribution after merging the testing and validation sets. . . . 47
4.6 One random example from each class of the CINIC-10 Large data set. . . . 49
4.7 The structure of the convolutional neural network used for the MNIST experiments. . . . 50
4.8 The two models used in the experiments. . . . 51
4.9 The architecture of the first convolutional neural network attempt for training a model on CIFAR-10. . . . 53
4.10 Training and validation accuracy over 100 epochs with both strategies attempted on the first CIFAR-10 model. . . . 54
4.11 The architecture of the second convolutional neural network attempt for training a model on CIFAR-10. . . . 56
4.12 Training and validation accuracy over 200 epochs on the second convolutional neural network architecture. . . . 57
4.13 An illustration of a residual layer from He et al. [22]. . . . 58
4.14 The structure of the ResNet used for CIFAR-10. . . . 60
4.15 Training and validation accuracy over 100 epochs on the ResNet. . . . 61
4.16 Structure of VGG16 CNN. . . . 63
4.17 Training and validation accuracy over 10 epochs on VGG16. . . . 64
4.18 Final structure of R-VGG16. . . . 66
4.19 Distribution of reduced data set. . . . 68
4.20 Training and testing accuracy of R-VGG16 on the reduced data set of 30,000 train and 10,000 test images. . . . 69

5.1 Example of FGSM applied to MNIST images with increasing epsilon. . . . 72
5.2 Example of FGSM applied to an image from the 1 class. Epsilon = 36 73
5.3 Example of PGD applied to MNIST images with increasing epsilon. . . . 74
5.4 Example of PGD applied to an image from the 1 class. Epsilon = 36. . . . 75

5.5 Comparison of all attacks on the MNIST data set. 77
5.6 Example of PGD applied to an image from the 1 class. Epsilon = 36. 78
5.7 Comparison of filters applied to the original, medium epsilon, and highest strength epsilon from the FGSM attack. 80
5.8 Comparison of filters applied to the original, medium epsilon, and highest strength epsilon from the PGD attack. 81
5.9 Performance of filters on increasing epsilons in the FGSM attack on the MNIST test set. 82
5.10 Performance of filters on increasing epsilons in the PGD attack on the MNIST test set. 82
5.11 Performance of filters on increasing epsilons in the FGSM attack on the MNIST test set in a black-box setting. 84
5.12 Performance of filters on increasing epsilons in the PGD attack on the MNIST test set in a black-box setting. 84
5.13 Example of FGSM with increasing epsilon applied to three random CIFAR-10 images. Lightening added: 0.19. 87
5.14 Example image from the airplane class misclassified as a cat as a result of the FGSM attack. Epsilon = 5. No image lightening applied. 88
5.15 Example of PGD with increasing epsilon applied to three random CIFAR-10 images. Lightening added: 0.19. 89
5.16 Example image from the airplane class misclassified as a cat as a result of the PGD attack. Epsilon = 5. No image lightening applied. 90
5.17 Comparison of white and black box attacks produced from the FGSM and PGD attacks on the CIFAR-10 testing set. 91
5.18 Comparison of filters applied to the original, medium epsilon, and highest strength epsilon from the FGSM attack on a random image from the CIFAR-10 testing set. 93
5.19 Comparison of filters applied to the original, medium epsilon, and highest strength epsilon from the PGD attack on a random image from the CIFAR-10 testing set. 94

5.20 Performance of filters on increasing epsilons in the FGSM attack on the CIFAR-10 test set. 96
5.21 Performance of filters on increasing epsilons in the PGD attack on the CIFAR-10 test set. 96
5.22 Performance of filters on increasing epsilons in the FGSM attack on the CIFAR-10 test set in a black-box setting. 97
5.23 Performance of filters on increasing epsilons in the FGSM attack on the CIFAR-10 test set in a black-box setting. 97
5.24 Example of FGSM with increasing epsilon applied to three random CINIC-10 Large images. 100
5.25 Example of FGSM applied to a random image from the dog class. Epsilon = 5. Difference image multiplied by 10. 101
5.26 Example of PGD with increasing epsilon applied to three random CINIC-10 Large images. 102
5.27 Example of PGD applied to a random image from the dog class. Epsilon = 5. Difference image multiplied by 10. 103
5.28 Performance comparison of the FGSM and PGD attacks on the white-box R-VGG16 and the black-box VGG16 networks. 104
5.29 Comparison of filters applied to the original, medium epsilon, and highest strength epsilon from the FGSM attack on a random image from the CINIC-10 Large testing set. 107
5.30 Comparison of filters applied to the original, medium epsilon, and highest strength epsilon from the PGD attack on a random image from the CINIC-10 Large testing set. 108
5.31 Performance of filtering defenses on increasing epsilons on the FGSM attack on the CINIC-10 Large test set. 109
5.32 Performance of filtering defenses on increasing epsilons on the PGD attack on the CINIC-10 Large test set. 110
5.33 Performance of filtering defenses on increasing epsilons on the FGSM attack on the CINIC-10 Large test set in a black-box setting. 111

5.34 Performance of filtering defenses on increasing epsilons on the PGD attack on the CINIC-10 Large test set in a black-box setting. 112
5.35 Performance of filtering defenses on increasing epsilons on the FGSM attack on the CINIC-10 Large test set after training models on the filtered images. . 114
5.36 Performance of filtering defenses on increasing epsilons on the PGD attack on the CINIC-10 Large test set after training models on the filtered images. . . 115
5.37 Comparison of trained filter defenses and filtered image FGSM attacks. . . . 116
5.38 Comparison of trained filter defenses and filtered image PGD attacks. 117

Chapter 1

Introduction

With the advent of the convolutional neural network, the field of computer vision broke through in its goal to perceive the world with human-like understanding. Computer vision has existed since the late 1950s, when David Hubel and Torsten Wiesel examined the primary visual cortex of a cat's brain as they showed it various images [24]. They made the discovery that there is a range of neurons performing visual processing in the brain. These neurons are arranged from simple to complex in order to process different parts of visual information, much like the convolutional layers of a convolutional neural network.

Fifty-three years later the world saw AlexNet, an eight-layer-deep convolutional neural network that won the ImageNet Large Scale Visual Recognition Challenge in 2012 [28]. The ImageNet Challenge is a worldwide benchmark in the field of computer vision that records the progress of visual recognition algorithms through time. It invites participants to classify 1.2 million images into 1000 different classes as accurately as possible [44]. AlexNet was able to classify the images in the data set with a top-5 test error rate of 15.3%, an error rate over 10% lower than the second runner-up [28]. This network has long since been outperformed by other convolutional neural networks, but it played an influential part in pioneering the highly accurate deep convolutional neural network.

Today, convolutional neural networks continue to improve in their accuracy and usefulness. Applications of convolutional neural networks range from face recognition to identifying malignant legions in medical patients [2, 49] . With the amount of information being created in image form everyday, convolutional neural networks play an important role

in big data analytics. In 2019, a convolutional neural network named *AmoebaNet-A* set the latest state-of-the-art with the ImageNet Challenge: a top-1 accuracy of 83.9% and a top-5 accuracy of 96.6% [42]. With all of this advancement and progress, it seems that convolutional neural networks will soon be on-par with humans in the way they "see" the world.

However, in 2014, Szegedy et al. [47] discovered a strange flaw in neural networks which exposed that they do not exactly "see" the world in the way that humans do. They discovered that by applying just a small, imperceptible perturbation to an image, they could cause a neural network to wrongly classify an image with high confidence. In addition, they found that the same small perturbation would cause this same behavior in a different neural network. In other words, the perturbation was shown to be portable across various convolutional neural networks. These perturbed images are referred to as *adversarial examples* [19], and are characterized by their imperceptibility to the human eye while simultaneously causing an otherwise accurate convolutional neural network to misclassify them.

The portability and imperceptibility of these adversarial examples makes them a very active field of research in both improving their effectiveness and building defenses against them. This thesis will discuss how to defend against such attacks while retaining accuracy on both affected and unaffected images. The overall goal of this thesis is to find a middle ground between robustness and accuracy for adversarial and regular images.

1.1 Motivation

Adversarial examples can pose a threat to systems that rely on convolutional neural networks to analyze and process data. For example, a system that scans handwritten digits could incorrectly classify these digits as the wrong number [20]. Additionally, adversarial examples could trick a content filter on a website. A user could post content that violates that website's guidelines, and by applying some adversarial noise to the image, the content filter would not be able to correctly flag that image as being in violation.

These attacks translate into the real world as well, but are much less practical in such a setting. Kukakin et al. [29] were able to show that printing images that had been adversarially perturbed showed the same adversarial behavior as their digital counterparts via the Tensorflow Camera Demo app. Eykholt et al. [16] were able to perturb stop signs via stickers and printed "decoy" stop signs. The signs were most often misclassified as "Speed Limit 45" signs, which was their target classification. Sharif et al. [45] were able to create a pair of adversarial glasses that would allow a user fool a facial recognition system into either misclassifying his or her face as someone else's, or to avoid his or her face being detected altogether.

Adversarial examples have become prevalent since Szegedy et al. [47] discovered the susceptibility of convolutional neural networks to such small perturbations in their inputs. This shortcoming exposes that neural networks do not "learn" about images in the same way as a human would. However, since it is not practical in the short term to alter the fundamental architecture of deep learning, it is instead desirable to focus on defenses against these attacks.

1.2 Goals

The goal of this thesis is to propose a way to defend against such adversarial attacks by transforming the input. This involves taking an image that has been perturbed adversarially and altering it in such a way that the neural network still classifies it accurately, despite the adversarial perturbations. In addition to correctly classifying an adversarial example, this work also aims to maintain high accuracy on images that have not been adversarially perturbed. Therefore, the classifier would be *robust* against such adversarial examples while not sacrificing too much accuracy on clean images. Due to the transferability of such attacks, this work will aim to defend against both *white-box* and *black-box* attacks. These are attacks that are made to target a specific network and attacks that can transfer to a different network entirely. The goal is to be robust against these attacks using one standard input transformation because the network will not have advance knowledge of whether it is classifying a clean image, a white-box example, or a black-box example.

Chapter 2

Background and Literature Review

2.1 Adversarial Attack Methods

Since the research done by Szegedy et al. [47], many techniques have been invented to create adversarial examples. Most techniques work by minimizing the distance between an image and its adversarial counterpart be under some norm while still causing the classifier to misclassify the example [35]. Many techniques use the model's gradients to compute the perturbations needed to create an adversarial example.

The first adversarial example, made by Szegedy et al., exploited the fact that neural networks are susceptible to small perturbations in their inputs. They theorized that the cause was due to the high non-linearity within the layers of a neural network, causing a sort of "blind spot" within training algorithms. They sought to expose this blind spot using box-constrained L-BFGS in order to minimize the below equation with respect to r [35, 47].

$$loss(\hat{f}(x+r), l) + c \cdot |r| \tag{2.1}$$

In the equation above, the variable x represents the original image, the variable r is the adversarial perturbation to be added to the image, which has a guarantee to remain within the range [0, 1], l is the target class, and c balances the distances between images and predictions. [35]. This formula allowed the authors to maximize loss in the model and create optimal perturbations for a convolutional neural network.

The technique above is the original technique proposed to create adversarial examples. Since its inception, many researchers have created a variety of different techniques to generate adversarial examples. They all share the same goal of fooling a classifier while beating proposed defenses. At the time of this writing, there are currently at least twenty variations of adversarial attack methods focused on image misclassification alone [1]. Some of the strongest attacks, such as Carlini & Wagner and ReColorAdv are very difficult to beat with existing defense techniques.[6, 30].

Despite the abundance of adversarial attack methods available, this thesis focused on quick, gradient-based attack methods. This is because of the large images used in testing defenses against these attacks in later parts of this thesis. Since some of the strongest attacks take a very long time to compute and apply to an image of size 224 by 224 pixels, it was not feasible to use a majority of these methods. Instead, this thesis focuses on the Fast Gradient Sign Method [19] and Projected Gradient Descent [34] attacks in order to test input transformation defenses. A background of these methods is given in the following subsections.

2.1.1 Fast Gradient Sign Method

The Fast Gradient Sign Method, or FGSM for short, builds upon the previous work of Szegedy et al. [47], in which they hypothesized that the effectiveness of adversarial examples is due to the high non-linearity of the convolutional neural network. Goodfellow et al. [19] instead explain that the linear behavior in the high-dimensional spaces of a neural network's layers is more than enough to create adversarial examples. They explain that the linear nature of these networks allows for a small input to grow linearly through the input dimensions resulting in a large change to the output. They call this "accidental steganography", and conclude that any network can have adversarial examples given high enough dimensionality in the input [19].

Linear models are susceptible to adversarial examples because they allow a perturbation to multiply with high dimensionality in the input. The equation that explains this behavior is shown in Equation 2.2. In their paper, Goodfellow et al. describe a scenario which an image with an 8-bit channel could easily produce an adversarial example. Any change in a pixel

value below the precision of the image, which here would be a pixel value of less than 1/255, should not affect how the classifier classifies an image. This means that as long as $||\eta||_\infty < \epsilon$, where ϵ is the maximum possible value below the precision, the outcome of the classification should not be affected. This change should theoretically be discarded. However, in high dimensional spaces the perturbation η will be multiplied by the weight vector ω, which has dimensionality n and average magnitude m. This will cause the activation to grow as much as ϵmn if the perturbation is chosen to be sign(ω). Essentially, the extremely small perturbations, if optimized to maximize the loss in the input x, will add up and cause a very large change to the output of the model.

$$\omega^{\mathrm{T}}\tilde{x} = \omega^{\mathrm{T}}x + \omega^{\mathrm{T}}\eta \tag{2.2}$$

Linear behavior allowed for Goodfellow et al. [19] to create a fast and cheap way to generate adversarial examples using a model's gradient. The formula below describes how to get an adversarial example x' using the FGSM. x is the original image, ϵ is the amount by which to shift the pixel intensity, $\nabla_x J$ is the gradient of the loss with respect to the input x, y is the vector of true labels, and θ is the vector of model parameters [19, 35].

$$x' = x + \epsilon \cdot sign(\nabla_x J(\theta, x, y)) \tag{2.3}$$

This equation facilitates a quick way to generate adversarial examples, and will be one of the two attack algorithms used in this thesis. The core idea behind the FGSM is that it maximizes the loss function with respect to the input in order to influence the activations within the model's layers by only altering the input. This way, the other parameters of the model are held constant while only the input is modified. An example of the FGSM attack in action on an image from one of the data sets used in this thesis is shown in Figure 2.1. The adversarial noise causes this automobile image to be misclassified as a truck.

2.1.2 Projected Gradient Descent

The second adversarial attack method used in this thesis is Projected Gradient Descent, or PGD [34]. It was created by Madry et al. in order to create a deep learning model that

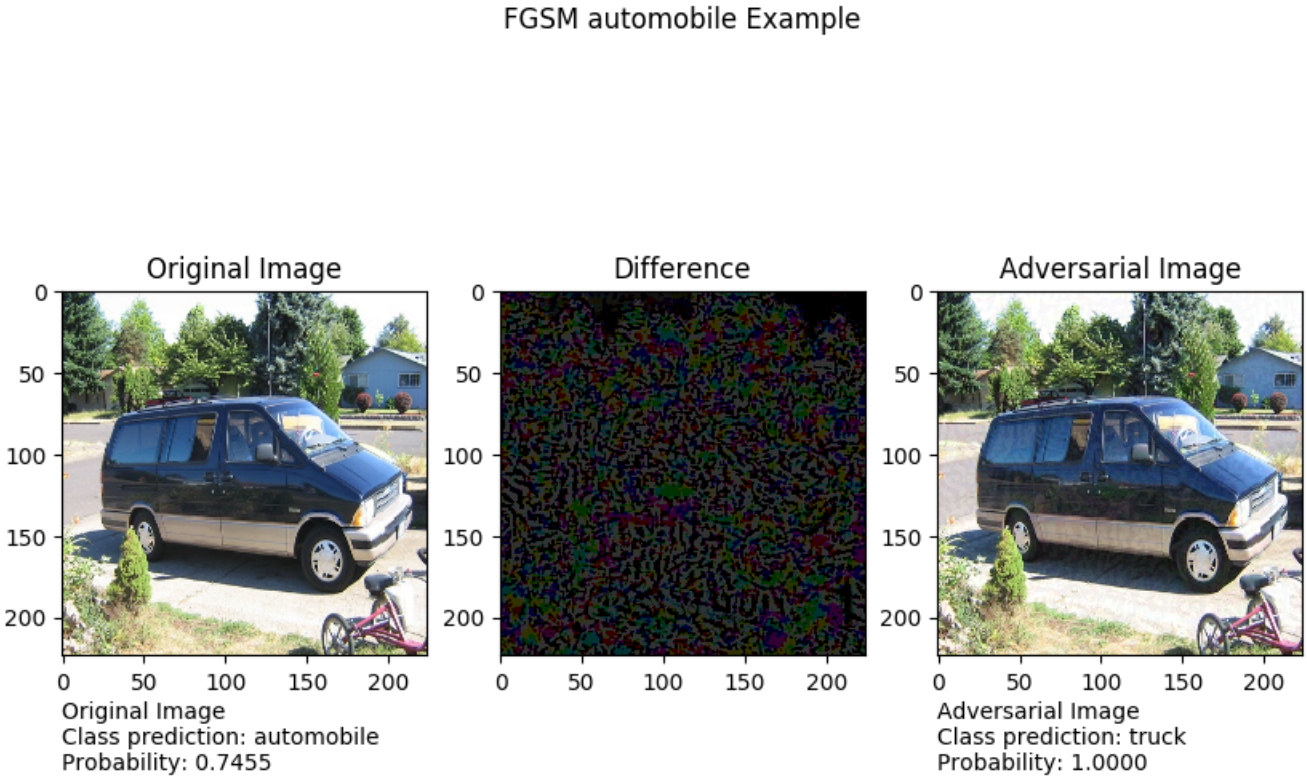

Figure 2.1: An example of the FGSM attack. Difference image multiplied by 10.

is as resistant to *first-order adversaries* as possible. A first-order adversary is one that has access to first-order information about the network that it is attacking. This means that it has access to the model's gradient and therefore can maximize the loss with respect to the input of that model as much as possible. The FGSM is also a first-order adversary. However, PGD is a much stronger attack than FGSM. This is because PGD is an iterative version of FGSM, also known as I-FGSM, where the only difference between it and I-FGSM is that PGD starts in a small "ball" area around the gradient with respect to the input [8].

This makes PGD a *constrained* optimization problem because as it takes steps it will verify that it does not move outside of the L-ball, usually the L_∞ ball or L_2 ball. The L-ball is defined as the difference in pixels between the original image and the adversarial image. The L_2 ball refers to the Euclidean distance between the original image and the adversarial image. The L_∞ distance refers to the difference between the largest pixels in the images. If the gradient steps taken during the process lead outside of the L-ball, then the result must be *projected* back into the constrained area. Staying within this constraint is to prevent the adversarial image from being too visually different from the original image. This constraint enforces the definition of the adversarial example – one that fools the neural network but looks like the original class to the viewer.

The equation for PGD is shown in Equation 2.4. PGD is different than FGSM because unlike FGSM, which computes the gradient of the cost function once and applies the transformation, PGD moves in small steps, calculating a new gradient for each step that it takes. At each step it adds a small perturbation α to each pixel in the input. Therefore, PGD can most effectively find the maximum of the loss function. This makes PGD a stronger attack than FGSM, and why it is claimed by Madry et al. to be a "universal" first order adversary, in which a neural network resistant to its attacks can be safely assumed to be resistant to all first-order attacks.

$$x^{t+1} = \Pi_{x+S}(x^t + \alpha \cdot sign(\nabla_x J(\theta, x, y)) \tag{2.4}$$

In 2.4, x^{t+1} is the image at the current step t, S is the allowed perturbation distance from x, Π_{x+S} is the constraint which must be projected back onto, and α is the small perturbation

constant. The rest of the terms are the same as in Equation 2.3 for the FGSM. An example of the PGD attack on the same image from the previous section is shown in Figure 2.2. Once again, the adversarial noise causes this automobile image to be misclassified as a truck.

2.1.3 Blackbox Attacks

Blackbox attacks, sometimes referred to as *transfer attacks*, are attacks on a previously unseen model. So far only attacks that have access to the model's gradient have been discussed. However, it is possible to create an attack on one model that can transfer to a different model. This will cause the new model to misclassify the adversarial example with high confidence, even though the example has had no access the model's parameters. An intriguing aspect of these transfer attacks is the agreement of the old model and the new model on what wrong class the adversarial example will be classified as. Transfer attacks will be used with both PGD and FGSM in this thesis.

Goodfellow et al. [19] explain both the phenomenon of portability and misclassification agreement across models in their paper. They state that, in the linear sense, the generalization of adversarial examples across models makes sense. Unlike their predecessor, Szegedy et al. [47], who theorized that adversarial examples exist in "pockets", they found that they actually exist in contiguous regions of the space defined by the attack. Therefore, it is only necessary to "push" the input far enough into this space and expect it to be adversarial across models. In regards to portability, they theorize that most modern neural networks resemble a similar linear model when trained on the same training set. This results in similar classification weights that produce the best generalization capabilities in a model. Since these weights are stable, it is expected that adversarial examples will follow suit. They proved this by training a shallow softmax network, a maxout network, and an RBF network, which is very non-linear, on the same data set and then applied transfer attacks. These transfer attacks were created on a deep maxout network. They found that the shallow softmax and the maxout network agreed on their misclassifications 84.6% of the time, while RBF only agreed with the softmax 53.6% of the time. This proves that adversarial examples do mostly generalize across networks.

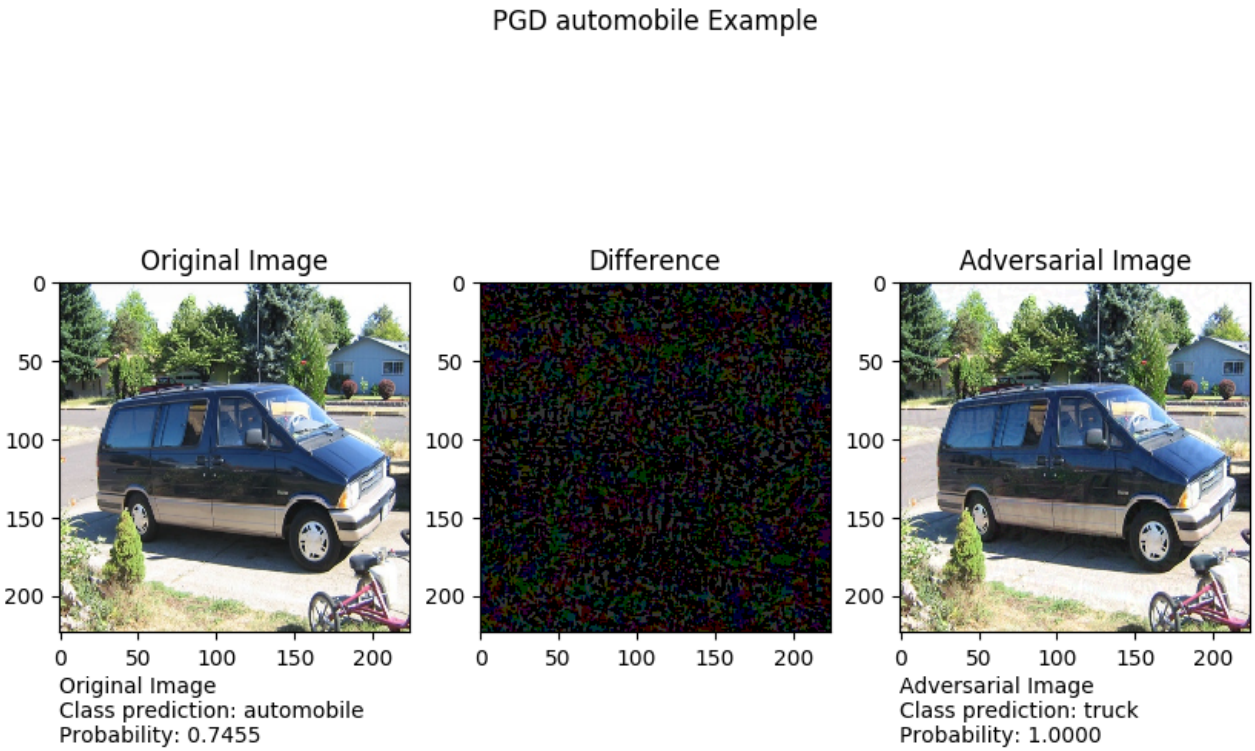

Figure 2.2: An example of the PGD attack. Difference image multiplied by 10.

2.1.4 Other Attack Methods

In addition to the two attacks discussed above, which will be used in this thesis, it is worth mentioning some other popular attacks in the literature. These attacks will be briefly discussed; for more detail, all original papers are referenced within their respective paragraphs for further reading. None of the attacks discussed below were used in this thesis because of their computational complexity, especially on the larger images that will be used in later experiments. It was important to have attacks that were quick to generate with the resources available, so while these attacks exist, it was not feasible to use them in heavy experimentation.

Gradient-Based Attacks

Gradient-based attacks rely on maximizing loss by having access to the gradient of the loss with respect to the input. FGSM and PGD are two such examples of these attacks, however, more attacks exist. In this section, the Carlini & Wagner attack, the DeepFool attack, and the Jacobian-based Saliency Map Attack will be discussed.

The Jacobian-based Saliency Map Attack, or JMSA, is a targeted attack, meaning that its goal is to have the model misclassify some input as a specific target class. It was created by Papernot et al. [39] to demonstrate a new type of gradient-based attack. Unlike other methods, this attack creates a direct mapping function from the input class to the output target class. It then iterates through the input image's pixels and computes which pixels to change on each iteration. They choose the pixels by evaluating the Jacobian of the gradient of the loss with respect to the input on every pixel. They refer to this as the *adversarial saliency map*, which computes which pixels to include in the perturbation in order to cause the neural network to misclassify the input image as the target class. The JMSA claims that it is able to create adversarial examples with the smallest fraction of pixels altered.

The DeepFool Attack is an untargeted attack, meaning that it does not push the input towards a specific class like JMSA does. DeepFool was created by Moosavi et al. [36] as a way to evaluate the robustness of neural networks against adversarial attacks. They motivated the paper in two ways: the desire to create a simpler and more accurate method for

robustness evaluation, and the desire to create an image with more optimal, less perceptible perturbations. The DeepFool Attack is simple in theory. It divides the class space with hyperplanes and projects the current input onto the nearest hyperplane, meaning the class space closest to the current class. This is referred to as the minimal perturbation, which they end up exceeding slightly in order to create an adversarial example. The result of this is an image that is nearly indistinguishable from the original image because the adversarial perturbations have been optimized to be as minimal as possible.

The Carlini & Wagner attack, or CW attack, is one of the strongest gradient-based attacks in the literature [6]. It was created as an attack to prove the weakness of *defensive distillation*, which will be discussed in the next section. It can be both targeted and untargeted. The attack comes in three forms: L_0, L_2, and L_∞, each of which constrains the allowed difference between the original image and the adversarial image. This is similar to PGD's constraints. In general, all three attacks work by inspecting every pixel of an image and finding the minimal perturbation that will change it to a different class. This can be either a target class or just a class other than the input class. The three metrics define the noise level of the perturbations which the attack uses to compute the smallest amount of noise to add to the image. The CW attack has the longest run time on a single image compared to the other two attacks discussed in this gradient-based section.

Score-Based Attacks

Score-based attacks do not need as much information about the model as gradient-based attacks. These attacks only need to know the class probabilities of the model in order to estimate the gradient. These are another type of blackbox attack, different from the one discussed in the subsection about blackbox attacks. The previous subsection discusses having a *substitute model* on which to create adversarial examples. These attacks involve only having access to the model's inputs and outputs, and repeatedly querying the model with different perturbations of images to examine how the confidence scores are affected. When such a perturbation has affected the confidence score and predictions to a point where the input image is misclassified, the attacker can build a set of adversarial examples with

which to attack the model. Two such attacks that will be discussed in this section are the following: a blackbox version of JSMA, and a blackbox version of CW.

The blackbox version of JMSA, created by Narodytska et al.[37], similar the white-box version discussed above, performs a greedy search to optimize pixels in an input image towards a target class. It uses a function the authors define as LocSearchAdv, which searches local neighborhoods of pixels in an image to find which pixels are critical to the image's classification. They decided to use local neighborhoods to keep computational costs realistic. They iteratively search for adversarial examples by randomly changing a neighborhood of pixels in an image and querying the output of the neural network to see if the probabilities or classifications changed. They found that this method is similar to the saliency maps generated by the JMSA attack, and that over 23% of the pixels chosen to perturb overlapped with those that would be chosen if a gradient-based saliency map was computed. They compared their method to reverse engineering gradient-based saliency maps.

The blackbox version of the CW attack is named Zeroth Order Optimization, or ZOO. It was created by Chen et al. [10] It is another technique that does not train a substitute model. Similar to the blackbox JMSA above, it only has access to the neural network's inputs and outputs. They use this technique to estimate the gradients of the neural network to generate adversarial examples. They do this through zeroth order stochastic coordinate descent combined with dimensionality reduction and importance sampling techniques. They create an objective function inspired by the formulation of the CW attack. They evaluate the objective function value at two close points in order to estimate the gradient vector from the probability distribution output by the model. After this, they perform coordinate descent on mini-batches of coordinates to optimize the most important pixels in an image to create an adversary. They call it zeroth order because it only relies on the objective function value computed from some input.

Decision-Based Attacks

Decision-based attacks are somewhat similar to score-based attacks in that they do not have access to the model's inner workings. However, they differ in that they do not have access to the model's confidence scores or logits, making them more realistic. Usually an attacker

will not have access to a model's confidence scores, but they will have access to the model's *decision*. They use this information to query the model and generate adversarial examples.

The Boundary Attack, created by Brendel et al. [4], was the first kind of decision-based attack to appear in the literature. It seeks to make an adversarial image that is as minimally perturbed as possible. It does this by iteratively modifying a starting image to look like some image that the attacker wishes to make adversarial. In the case of an untargeted attack, the starting image is a uniform distribution of adversarial noise. The boundary algorithm will then continue to iteratively modify the image by drawing from the adversarial noise to make it look like the desired image while retaining the image's adversariality. In the case of a targeted attack, the starting image is one from the desired misclassification class, which is then modified to look like an image from another class. This is done by walking along the decision boundary and repeatedly querying the model to see if the image is still adversarial after each modification. A parameter ϵ is used as the length of the step-size. The closer the walk gets to the original image, the smaller the step length becomes. This process ends whenever ϵ converges to zero. This process is very computationally expensive but results in adversarial images that look very clean.

HopSkipJump is a variant of the Boundary Attack, created by Chen et al. [9]. It improves on the Boundary Attack by reducing the amount of model queries. The Boundary Attack has to make a large number of queries in order to be a high level threat. Additionally, whenever the Boundary Attack finds a perturbation that does not match with the target class, it throws it out. However, the HopSkipJump attack uses this to its advantage to estimate the direction of the model's gradient. It uses the same principles as zeroth order optimization but is more limited in that it only has access to the model's decision. They use this gradient estimate to optimize the distance between an original image and its adversarially perturbed counterpart. This is a less expensive way to create adversarial images than the Boundary Attack but just as effective. This method is very similar to the Boundary Attack except that it incorporates gradient estimations, which is why it was called BoundaryAttack++ in older versions of the paper.

2.2 Defense Methods

One cannot have an attack without a defense. Whenever the FGSM attack was introduced, the authors naturally tested a defense against their own creation. The world of adversarial attacks and defenses is akin to a game of cat and mouse. Once a defense has been created against an attack, another stronger attack comes along to break it. Then another stronger defense is created against that attack, and an even stronger attack appears to break that defense. The world of defenses and attacks has come a long way since the FGSM, and covering all of them would be out of the scope of this thesis. Instead, this section will focus on defenses relevant to the ones used in the experiments performed here, as well as some other defenses that the author finds interesting.

2.2.1 Filtering Defenses Used

The defenses used in this thesis consist of input transformations. The nature of the adversarial example is to perturb the input in such a way that its alterations multiply in the high dimensional space. However, if the perturbations to the input are removed or masked by some means, such as image filtering, then the adversarially altered pixels should not have as large of an impact on the decision of the network. This is the motivation behind using input transformations to defend against adversarial examples. Because these transformations do not assume anything about the type of attack being leveraged against the network, they can theoretically apply to any attack. Several filtering defenses were experimented with in this thesis. Some data sets had different filters applied to them than others based on the nature of the images. The MNIST data set, for example, is a black and white data set of digits, so different filters are more suited to such a binary image. The CIFAR-10 and CINIC-10 Large data sets are small and large data sets with more detail and color, so more blurring filters were experimented with. There is some overlap on which filters were used on both data sets. All of the filters will be discussed in this section. The specific filters and parameters used for the data sets will be elaborated on Section 3 and Section 5.

Thresholding

Thresholding is the process of making an image binary by setting its pixel intensities to 255 if they are above a certain threshold intensity and setting pixels to 0 if they are equal to or less than the threshold intensity. The equation for thresholding is shown in Equation 2.5, where *f(x,y)* is the original image and *g(x,y)* is the altered image.

$$g(x,y) \begin{cases} 255 & f(x,y) > threshold \\ 0 & f(x,y) \leq threshold \end{cases} \tag{2.5}$$

Gaussian Filtering

The Gaussian filter is a smoothing or blurring filter that convolves an image with the Gaussian kernel. The formula for the Gaussian kernel is shown in Equation 2.6, where *g(x,y)* is the resulting blurred image. The kernel shown works in two dimensions, however, it is possible to convolve an image with two one dimensional Gaussians as well, making the Gaussian kernel a *separable* kernel. The Gaussian used in this thesis is the two-dimensional version. The resulting blur from this function can be controlled via the variable σ and the size of the kernel. A larger value for σ will result in a more blurred image, while a larger kernel size will result in more pixels being considered in the convolution.

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.6}$$

Median Filtering

The median filter works by locating spikes of intensity in an image and replaces them with the median of the pixels surrounding the spiked pixel. Therefore, it does not introduce any new values into the image, preserving image quality. Equation 2.7 shows how the median filter works. Here i and j represent pixel offsets from the central pixel at x, y while staying within the pixel neighborhood of size R. The median filter is best at removing salt and pepper noise.

$$g(x,y) = median(f(x+i, y+j)|(i,j) \in R) \tag{2.7}$$

Bilateral Filtering

The bilateral filter works by replacing the intensity of each pixel with a weighted average of intensity values from a pixel neighborhood. The formula is shown in Equation 2.8, where p is the coordinates of a pixel to filter, I_p is the value of said pixel, q is the relative position to p, I_q is the value at q, G_{σ_r} is the range kernel for smoothing intensities, and G_{σ_s} is the domain kernel for smoothing coordinates. The bilateral filter is best at smoothing images while preserving edges.

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(||p-q||) G_{\sigma_r}(|I_p - I_q|) I_q \tag{2.8}$$

where the normalization factor W_p , which keeps the pixel value within bounds, is:

$$W_p = \sum_{q \in S} G_{\sigma_s}(||p-q||) G_{\sigma_r}(|I_p - I_q|) \tag{2.9}$$

Resizing

Resizing is the process of either downsampling or upsampling an image. Downsampling involves averaging pixels together in order to create the fewer pixels needed for a smaller image. This is done using a weighted average to determine what value the smaller pixels will hold. The weight comes from the distance between the pixels, with closer pixels having higher weights. Upsampling involves interpolation to fill in the "missing" values. These missing pixel values are calculated using a weighted average as well. It uses bilinear interpolation in order to estimate the new pixel values between two known pixel values.

Total Variation Denoising

Two variants of the total variation denoising filter are used in this thesis. The first variant is the original 1992 filter created by Rudin et al. [43], and is commonly referred to as the Rudin-Osher-Fatemi, or ROF, filter. The second variant is the 2004 variant created by

Chambolle [7]. The goal of the total variation denoising filter is to smooth images while preserving boundaries. The algorithms performs on the assumption that a noisy image will have high amount of variation and removing this variation will remove noise while leaving edges unaffected. Both methods work the same way – they seek to minimize the total variation norm of the estimated noise in an image. However, they both have different solutions to this minimization problem, shown in Equation 2.10.

$$\frac{\gamma}{2}\int (u-v)^2 + \int |\nabla u| \rightarrow \min_{u \in BV} \tag{2.10}$$

Rudin et al. first proposed this constrained optimization problem. They seek to recover the original image u from the noisy image v while staying within a bounded variation BV. This allows for "spurious oscillations" to be removed while retaining sharp edges. Here, γ is a given Lagrange multiplier which controls the amount of smoothing in the image. The larger the γ, the more smooth and "cartoonish" the image will appear. Chambolle improved on the original solution to the algorithm using a dual formation to find the minimization. The math to both solutions is rather complicated, so it is recommended to refer to the original papers for more details.

Based on both Chambolle's paper and visual inspection, Chambolle's TV algorithm is somewhat better at denoising images while retaining object boundaries. Since the goal of this thesis to remove adversarial noise while retaining original accuracy, it was decided to look at both a filter that might overblur the boundaries while removing most of the adversarial noise, and one that might not remove as much noise but retain the image's original structure.

Non-local Means

Non-local means, created by Buades et al. [5] in 2004, is a filter for denoising images. It works by replacing a noisy pixel with the average of other pixel values. These pixels are chosen based on their similarity to the pixel. The averages are selected from patches of neighborhoods in the image; the most similar patches to the current noisy pixel's patch are averaged to replace the noisy pixel. This is why it is called "non-local"; it uses the entire image to compute denoising values instead of just a local neighborhood around the pixel.

However, this method is computationally expensive, so a faster version of the non-local means algorithm is used in this thesis. The faster version uses a uniform spatial weighting instead of a Gaussian one to compute patch similarity.

The equation for non-local means with uniform weighting B is shown in Equation 2.11, where u is the original noisy image, $C(x)$ is a regularizing parameter, $Omega$ is the set of all pixels in the image, x is the current pixel being evaluated, and h is a cut-off distance for patch size consideration. A larger h results in a blurrier image. There is an additional optional parameter $sigma$ that represents the expected noise variance in the image. This is a parameter used to subtract the noise variance out when computing the patches; it can be either explicitly given or estimated.

$$NLM[u](x) = \frac{1}{C(x)} \int_{\Omega} e^{-\frac{B * |u(x+.) - u(y+.)|^2(0)}{h^2}} u(y)d(y) \tag{2.11}$$

Anisotropic Diffusion

Anisotropic diffusion, sometimes referred to as Perona-Malik diffusion [41] is another filter with the goal of removing noise from an image. Similar to the bilateral filter, it seeks to eliminate noise while preserving edges. It works by only convolving the image orthogonal to an edge, instead of convolving in all directions. The equation for anisotropic diffusion is shown in Equation 2.12, where div is the divergence operator, ∇ is the gradient operator, Δ is the Laplacian, and $c(x, y, t)$ is the diffusion coefficient, which dictates how the diffusion spreads.

$$I_t = div(c(x, y, t)\nabla I) = c(x, y, t)\Delta I + \nabla c \cdot \nabla I \tag{2.12}$$

Perona and Malik derive two different functions for the coefficient, shown in Equations 2.13 and 2.14. The first equation is more receptive to high contrast edges, while the second prioritizes wide regions in an image. The variable K is a constant that controls conduction, meaning how sensitive the smoothing is to edges in the image. A smaller value for K means smaller edges will be preserved. There is an additional parameter γ that controls the speed of the diffusion, where $0 \leq \gamma \leq 1/4$ for stability.

$$g(\nabla I) = e^{-(\frac{||\nabla I||}{K})^2} \tag{2.13}$$

$$g(\nabla I) = \frac{1}{1 + (\frac{||\nabla I||}{K})^2} \tag{2.14}$$

2.2.2 Comparison to Other Filtering Defense Papers

Guo et al. [21] first examined filtering defenses in their 2018 paper *Countering Adversarial Images Using Input Transformations*. They apply a variety of input transformations to adversaries produced using the FGSM [19], the iterative FGSM [29], DeepFool [36], and the Carlini & Wagner attack [6]. They used the following input transformation methods: crop ensemble, total variation minimization, image quilting, bit depth reduction, and JPEG compression. Total variation minimization has already been covered in the previous section, so the remaining four filters will be briefly explained here.

Image cropping and rescaling is the process of randomly cropping and rescaling the image with the idea that the adversarial perturbations will be repositioned. Because the adversarial perturbations must be precise within their location in the input, the idea of averaging predictions over random crops and rescales of the same image should reduce the adversarial effect. This is because the perturbations will no longer be in the same position in the image. Bit-depth reduction is the process of reducing the variations in pixels, which should theoretically remove some potentially adversarial perturbations. The authors reduce the images to three bits. JPEG compression is lossy and should potentially remove adversarial information from the image. They perform compression at 75% quality level. Finally, image quilting, introduced by Efros et al. [15], is the process of texture synthesis and transfer to piece together a new, potentially "clean" image on top of an adversarial one. The motivation behind this method is that the new image will only be sewn out of clean patches.

They perform three variants of experiments: white-box, black-box, and gray-box attacks. White-box and black-box have been explained in the earlier sections of this paper. Gray-box attacks, in their definition, are attacks where the adversary has access to the inner components of the network but not to the filter being used. They perform these experiments on ImageNet, with the full 1.2 million training image, 50,000 testing image, 1,000 class data set. They use images of size 224 by 224. They found the greatest success with the total variation and image quilting defenses and attribute it to the filters' randomness. Success is defined as a filter that retains high accuracy on the data set in both the original and adversarial cases. Additionally, they trained the neural network on the filtered images to increase accuracy. They also found some mild improvement by using an ensemble of models trained on the best filters.

Another paper by Lee et al. [32] uses the ROF and non-local means (NLM) filters to defend against adversarial noise. They use the 10K ImageNet training and testing data with images of size 224 by 224. They used universal adversarial perturbations as the attack. In short, this attack is an iterative version of the DeepFool algorithm in that it finds perturbations that push the input data points to the nearest decision boundary and into a different class. They compare performance against clean images, adversarial images, denoised adversarial images, and denoised adversarial + random noise images. They found that the ROF and NLM filters increased accuracy in blackbox and whitebox attacks to roughly the same overall accuracy, which is about 50% accuracy. The maximum accuracy is 70% on all models.

This thesis covers some of the same experiments as the papers above, however, when selecting the filters, none of the papers performed filter parameter optimization to select filters with the best trade-off between clean and adversarial accuracy. It is unclear how they chose their parameters. Additionally, some filters are used in this thesis that are not used in those papers, such as anisotropic diffusion. Finally, this thesis uses multiple filters in sequence as one of the filtering defenses to add an extra layer of randomness and defense. Additionally, an attack that breaks the defenses is discovered, lending itself to the "cat" role in the perpetual cat and mouse game of defense. This new attack could be considered a white box attack because it is an extra step beyond the gray-box attacks the first paper

covers. This is because the attacker has knowledge of all model parameters *and* filtering techniques.

2.2.3 Other Defense Methods

In addition to image filtering, there is a wealth of other adversarial defense methods [1]. In addition to modifying the input, there are defenses that modify the network, such as changing its structure or loss function. Some use external models to add onto the original network at classification time. Some try to detect the adversarial examples. One of each type of these defenses will be discussed in this section, however, for more detailed coverage of defenses, it is recommended to refer to the survey paper cited here.

Adversarial Training

The first line of defense is usually adversarial training. Most papers that introduce a new attack propose adversarial training as the first defense strategy. This can be seen as early as the introduction of the FGSM [19], where they propose to train the network with adversarial examples included in the training set. This increases the network's robustness because the network has examples of the attack that it will encounter at classification time and will have learned the correct classification of these adversarial images. Goodfellow et al. were able to reduce the error from 89.4% to 17.9% with their adversarial training method.

Defensive Distillation

Defensive distillation was created by Papernot et al. [39]. This is the defense that the CW attack was designed to break, mentioned in an earlier section. It works by taking a smaller network that is trained on probabilities of a larger network. The intuition behind this approach is that information learned during training in the network is encoded in the probability vectors output from the network. This information contains class knowledge that can be used to train a smaller neural network without loss of accuracy. This allows for extra information to be learned about each class, as opposed to simple "hard labels". The authors use the example of a 1 and a 7 in the MNIST data set. These two images

will give high probability scores for both classes; therefore, the model learns that there is structural similarity between the digits. This method has the advantage of being robust to unseen attacks because it can hide information from the attacker by being an additional filter between the input network and the adversarial output.

Deep Cloak

Gao et al. [18] propose a solution that removes extra unnecessary features extracted from an image in a classifier. They hypothesize that the reason that adversarial examples are so effective is because they are able to take advantage of all of these unnecessary features. They do this by modifying the network structure to have an extra "mask" layer. They built the mask by comparing pairs of original and adversarial examples to discover which features are unnecessary. They determined that the features that change too quickly are adversarial and should be removed. Using this knowledge, they insert the mask layer at the end of the feature extracting layers in the model architecture and remove those unnecessary features. They found that masking even just 1% of features reduced adversarial effectiveness by 10% when tested against the FGSM.

Feature Squeezing

Feature squeezing is an adversarial detection method proposed by Xu et al. [48]. They explore two methods of this squeezing: reducing the color bit depth of each pixel in the image and image smoothing. This requires adding two external models to the main model; one that reduces the depth and one that smooths the image. They then take the results of all three predictions and compare them. If there is a drastic difference between the three images, they determine that the example is adversarial and discard it.

The rest of this thesis will cover experiments performed with the techniques described above. The next section will cover a full experiment performed with a support vector machine. Section 4 will give background on the experimental setup of the main experiments performed with three data sets and their corresponding neural networks. Section 5 will show

the results of these experiments. Section 6 will discuss the results and introduce potential avenues for future work.

Chapter 3

Support Vector Machine Adversarial Attacks

Before moving into attacks and defenses with the convolutional neural network, a section of this thesis will be dedicated to providing a proof of concept with a support vector machine (SVM). The SVM will be used for the supervised classification problem of classifying the MNIST data set of handwritten digits, which will be expanded on in the following subsection and in Section 4. The SVM learns how to generalize on new samples by finding the best *decision boundary* that separates class samples in the training set. An example of a decision boundary is shown in 3.1. The figure shows a dotted line on either side of the decision boundary. This is the margin, which should be as wide as possible for best generalization. The class samples that determine how wide this margin is are the *support vectors*. They lie on either side of the margin and designate the beginning of their respective class space. The SVM draws a linear decision boundary, but it uses a technique called the "kernel trick" to project the data into a higher dimensional space so that it can draw non-linear decision boundaries in the original space.

The SVM used in this proof of concept has a linear decision boundary and is trained using a *one-vs-the-rest* decision function, meaning that it will train as many models as there are classes [40]. This will result in n decision boundaries, with n being the number of classes. This means that every side of each decision boundary will have a positive class, and anything on the other side of that boundary belongs to the negative class. The linear nature of these

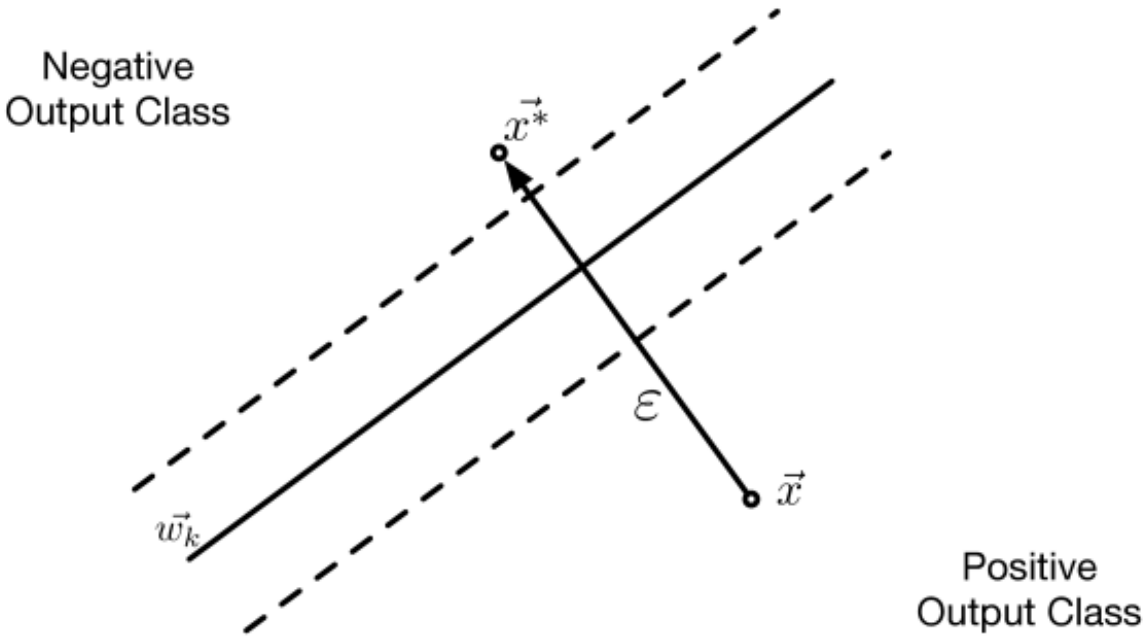

Figure 3.1: Figure from *Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples* [38]

boundaries can be exploited with perturbations that can essentially "push" an image by ϵ amount over the decision boundary and cause it to be misclassified. Figure 3.1 shows a visualization from Papernot et al. [38] demonstrating how this "pushing" occurs. It shows how an input image $\vec{x}$ belonging to class k can become its adversarial counterpart $\vec{x^*}$ when perturbed by an amount ϵ over the linear decision boundary $\vec{\omega_k}$ [38].

3.1 Methodology

3.1.1 Creating Adversarial Images

The MNIST data set of handwritten digits was used for this proof of concept [31] The data set has ten classes, consisting of 60,000 training examples and 10,000 test examples. In order to make this proof of concept simple while still being a multi-class classification experiment, the data set was reduced to three classes: 2, 3, and 7. There is no significance

to these numbers, any other classes from MNIST could have been chosen. After reduction, the training set consisted of ~6,500 examples per class, and the testing set consisted of ~500 examples per class.

The adversarial images were created using the methodology of Papernot et al. [38]. The images from the training set were trained on the `LinearSVC` model from scikit-learn[40]. This classifier was chosen for its speed. The the regular `SVC` with a linear kernel took over ten hours to train while the `LinearSVC` took three minutes. The documentation on scikit-learn's website states that the fit time scales quadratically with the number of samples, and that training with this classifier is impractical beyond tens of thousands of samples. They provide `LinearSVC` as an alternative to solve this issue. Both of these classifiers are provided from the LIBLINEAR library [17] and have comparable performance on large data sets, within a percentage point of each other.

No preprocessing was applied to the image vectors. After completing training, the weight vectors were extracted from the SVM and used to craft the adversarial examples using the following equation.

$$\vec{x^*} = \vec{x} - \epsilon \cdot \frac{\omega\vec{[k]}}{||\vec{\omega_k}||} \tag{3.1}$$

Due to the nature of the *one-against-the-rest* decision function, each class had a corresponding weight vector. This is because the SVM has trained multiple binary classifiers with samples from one class being labeled as positive and samples from another class as negative. When taking the dot product of the weight vector with an example from that class, the result will be either positive or negative, with positive meaning that the current class was labeled as the positive class for that specific binary classifier. Other examples will, in turn, be negative. This is important in order to know which example $\vec{x}$ corresponds to the positive side of the weight vector $\omega\vec{[k]}$, where k is the positive class.

After fetching the proper weight vectors and determining the appropriate class hyperplanes, the adversarial images were produced using a set value of $\epsilon = 200$. This number was chosen because it consistently produced adversarial examples that were not very visually

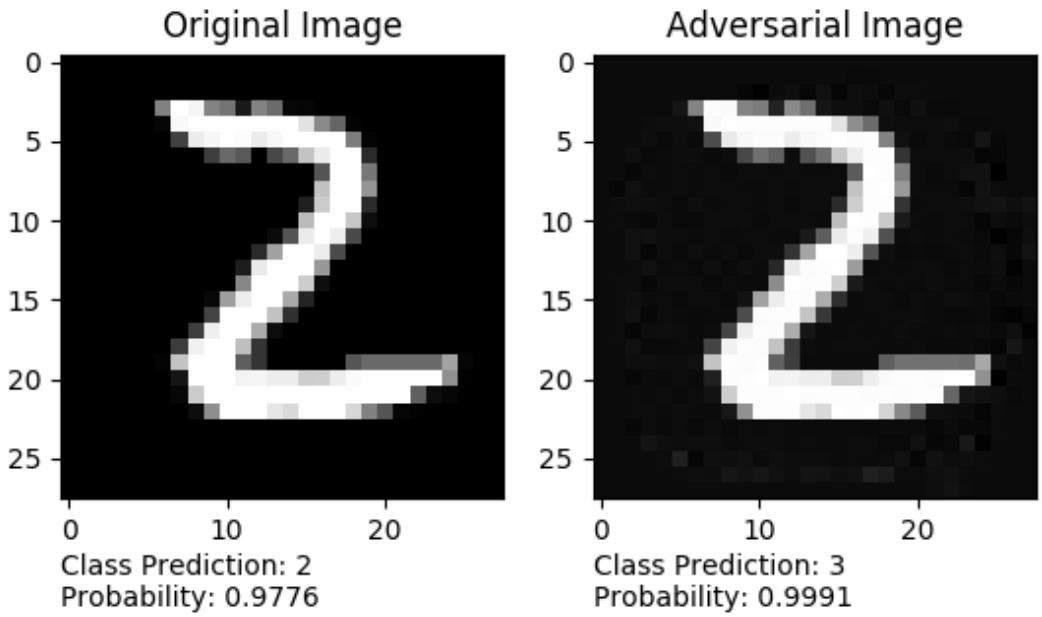

Figure 3.2: Example of an adversarial image created from an example in class '2'. The left shows the original image and its prediction from the SVM. The right shows the adversarial image. Although similar to the human eye, the SVM classifies the image on the right as belonging to class '3' with 99% confidence.

distorted while causing the classifier to return incorrect predictions with high confidence. An example of an adversarial image is shown in Figure 3.2.

3.1.2 Defense Building

After creating adversarial images on the testing set, several image processing techniques were used in an attempt to defend against the adversarial effects. Five methods were applied, all using the OpenCV library in python [3]. They are listed below. All of the parameters for these filters were chosen using manual inspection that produced a balance between visual integrity of the image and a successful defense with the best confidence score results. All filtering techniques were applied both original and adversarial images in order to compare the accuracy gain or loss when filtering both sets of images.

1. Thresholding

2. Gaussian Filtering

3. Median Filtering

4. Bilateral Filtering

5. Resizing

3.2 Experiments

This section will describe the experiments performed and show the results of the adversarial images with and without filters applied. All techniques described were applied to both the original and adversarial image in order to compare how filtering affects both accuracies. A large part of making the SVM more robust to adversarial images is using filters that retain high accuracy on the unaltered original images.

The same set of three images will be used throughout this section for comparison purposes. They were chosen for their high confidence on both the original and adversarial images. Additionally, an overall score on the test set will be shown for the original and adversarial images.

3.2.1 Unaltered Results

Before taking a deep dive into the defenses, the unaltered images and their unaltered adversarial counter parts must be examined. They are shown in Figure 3.3. One thing to notice about the adversarial images is that they retain their visual integrity. There is obviously some noise in the background, but the human eye can still clearly read the 2, 3, and 7. This is a large part of what makes adversarial images so intriguing. The human eye can see the image correctly but the classifier cannot. Based on the visual integrity of the images, and the results in Table 3.1, it can be concluded that the adversarial attack was successful on the SVM. The accuracy on the original images is high, while the accuracy on the adversarial images is very low, but the confidence remains high.

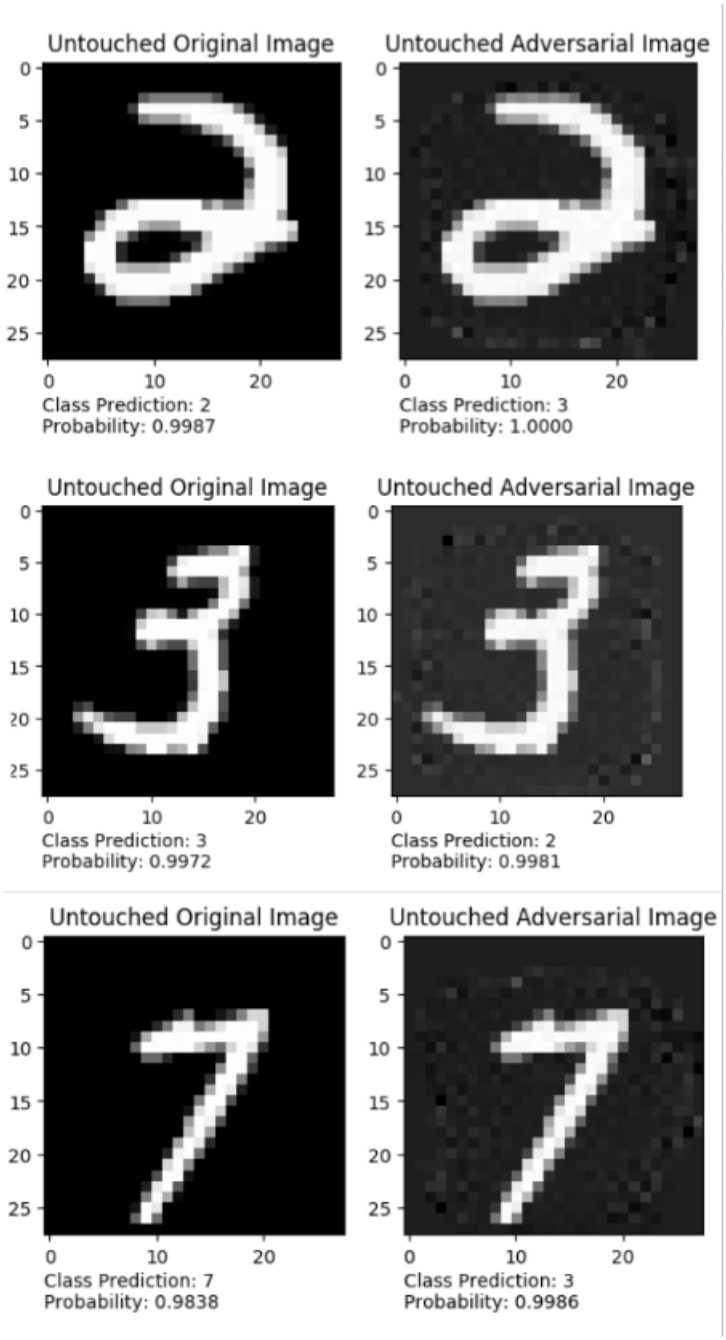

Figure 3.3: A set of unaltered adversarial images and their original image counterparts.

3.2.2 Thresholding

The first defense was to threshold the images. This is a defense unique to MNIST because of the black and white nature of the images. The pixel intensity of 100 was chosen for this thresholding after inspecting a histogram of the pixels and finding that they mostly diverge around the 100 mark.

The results of thresholding are shown in Figure 3.4. One thing to notice about the three images is that they all happen to have the same confidence score. However, looking at the metrics in Table 3.1, it is clear that the thresholding still performs slightly worse than the original images. Overall, the results from thresholding are very good. This method almost completely restored the accuracy. However, thresholding does not generalize to other types of images, and the MNIST data set is a special case in this regard. Other types of filtering methods should be used instead when trying to make a general defense.

3.2.3 Gaussian Filtering

The next defense method was Gaussian filtering. This method was chosen because it blurs the image, and therefore should blur the adversarial noise to a point where its effect on the classification of the image is minimized. The images were convolved with a 5 by 5 kernel and with $\sigma = 5$. These parameters were chosen after testing various values for the Gaussian filter and finding that those distorted the image the least while keeping the confidence score high.

The results of filtering with the Gaussian kernel are shown in Figure 3.5 . The filtering improved the accuracy of the SVM on the adversarial image significantly. It did not restore original accuracy, but the improvement from nearly 0% to 82% is significant, shown in Table 3.1. It appears that the Gaussian kernel effectively blurred out the adversarial noise while leaving the original image mostly intact, which is the desired balance when filtering these images.

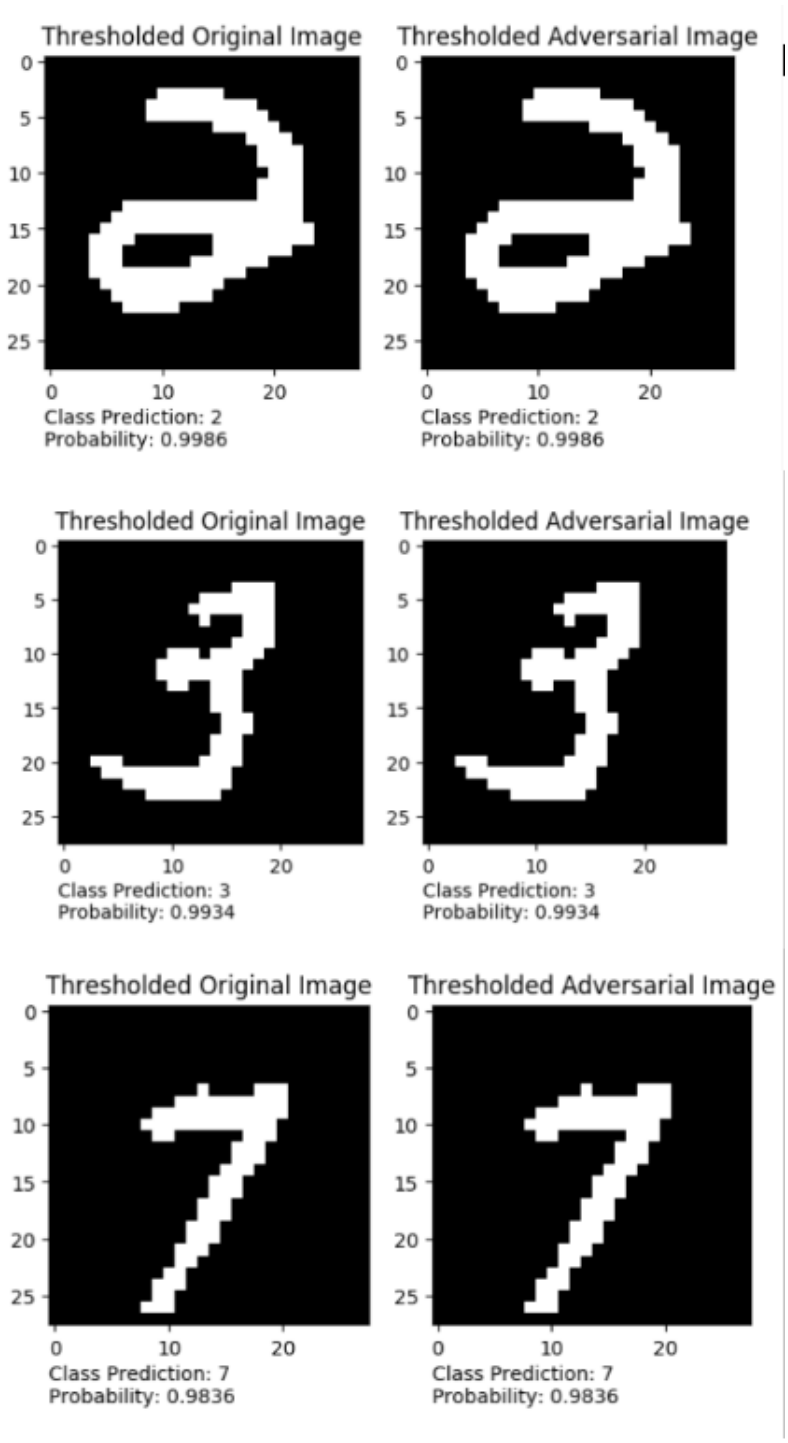

Figure 3.4: The results from thresholding both the adversarial and original images.

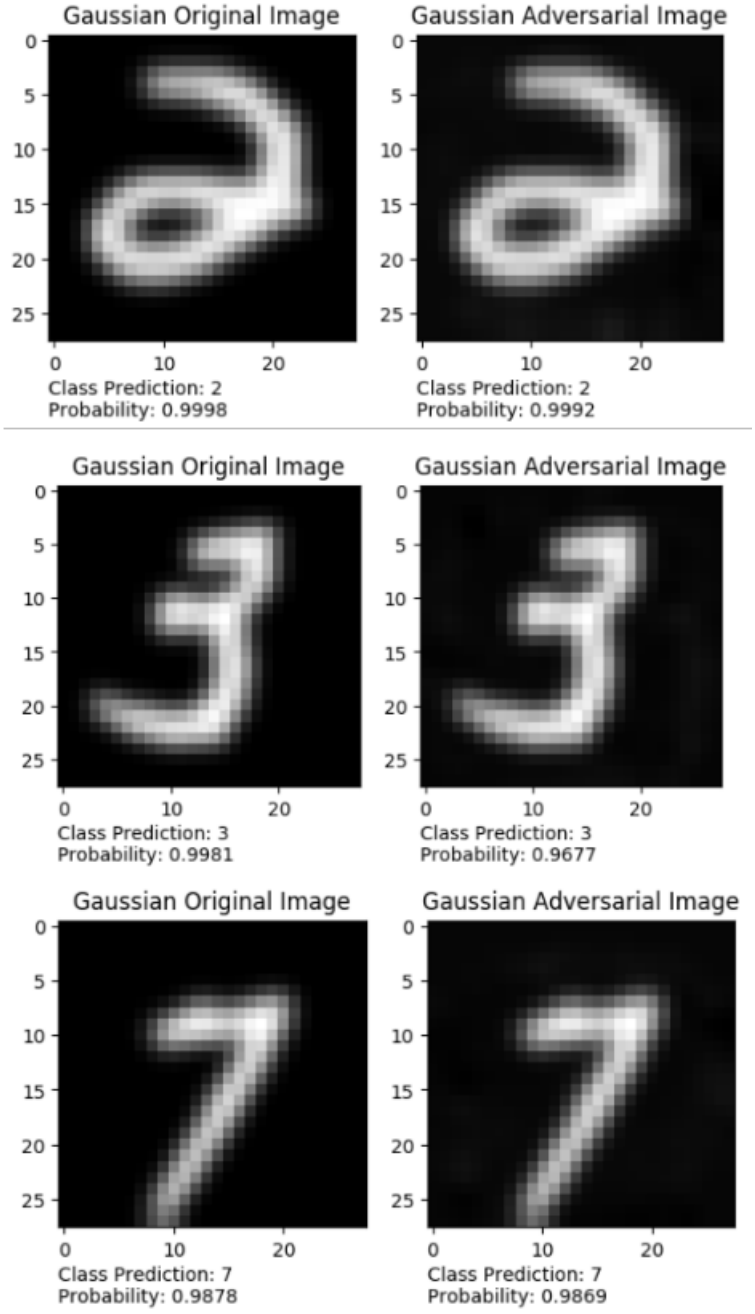

Figure 3.5: The results from convolving both the adversarial and original images with the Gaussian kernel.

3.2.4 Median Filtering

The median filter was chosen for this task due to its utility in removing salt-and-pepper noise. The adversarial noise in these images produces some sudden spikes and dips in intensity, so the median filter should be able to smooth these spikes away. The size of R was chosen to be 3 because it performed better than larger windows, which distorted the image.

The results of applying the median filter are shown in Figure 3.6. Overall, this filter worked well in removing the adversarial pixels and smoothing the noise. The filtered adversarial images almost identical to their original counter parts, but upon close inspection, it is possible to see some artifacts still remain in the images. Although the median filter appears to have nearly restored the images to their non-adversarial state, the accuracy score in Table 3.1 still reports a loss in accuracy from the original images. So while the median filter mostly defended against the adversarial noise, the resulting image was still marginally affected.

3.2.5 Bilateral Filtering

The bilateral filter was chosen for its ability to remove noise while preserving edges. Therefore, using this filter should result in removing adversarial noise removed while preserving the outlines of the digits.

The pixel neighborhood size chosen was 9, based on visual inspection of what size removed noise while preserving the integrity of the image. Two additional parameters corresponding to the range weight and space weight were set. In OpenCV, they are called sigmacolor and sigmafilter respectively. They were both set to 100 because this produced the best visual result.

The bilateral filter preserves edges and effectively blurs out the adversarial noise, seen in Figure 3.7. Upon close inspection, a slight glow can be seen in the images, which is more pronounced in the adversarial images. This defense comes the closest to restoring the image to its original state and performs well for classification. However, it is still outperformed by the median filter, likely because the noise has more of a salt-and-pepper quality, which the median filter is more suited for removing. The results can be seen in Table 3.1.

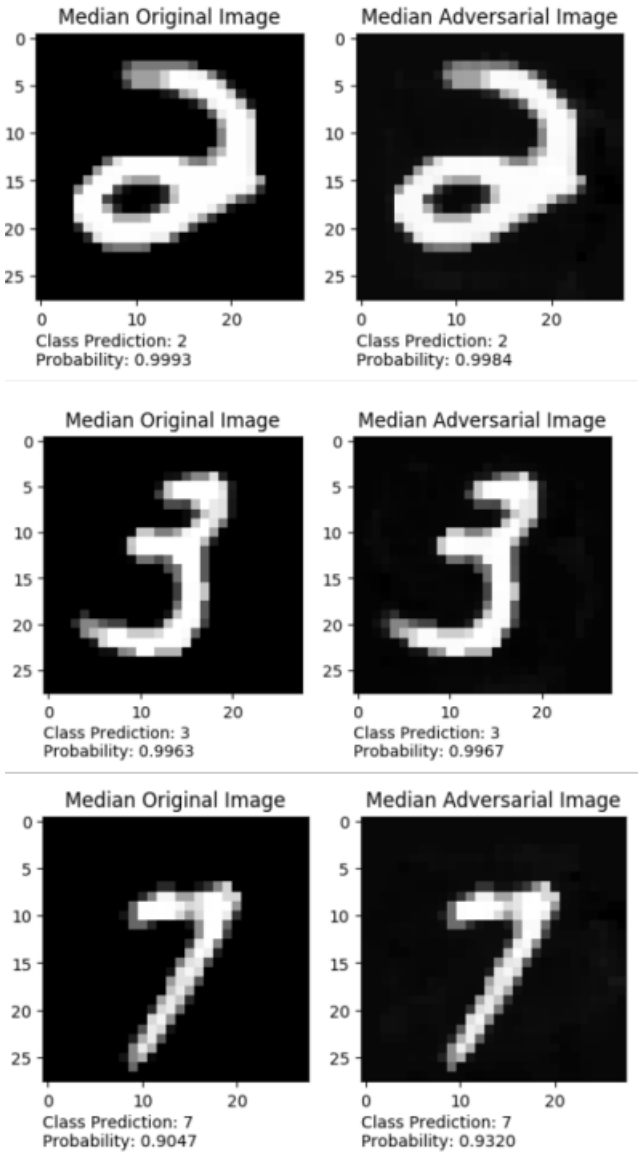

Figure 3.6: The results from filtering both the adversarial and original images with the median filter.

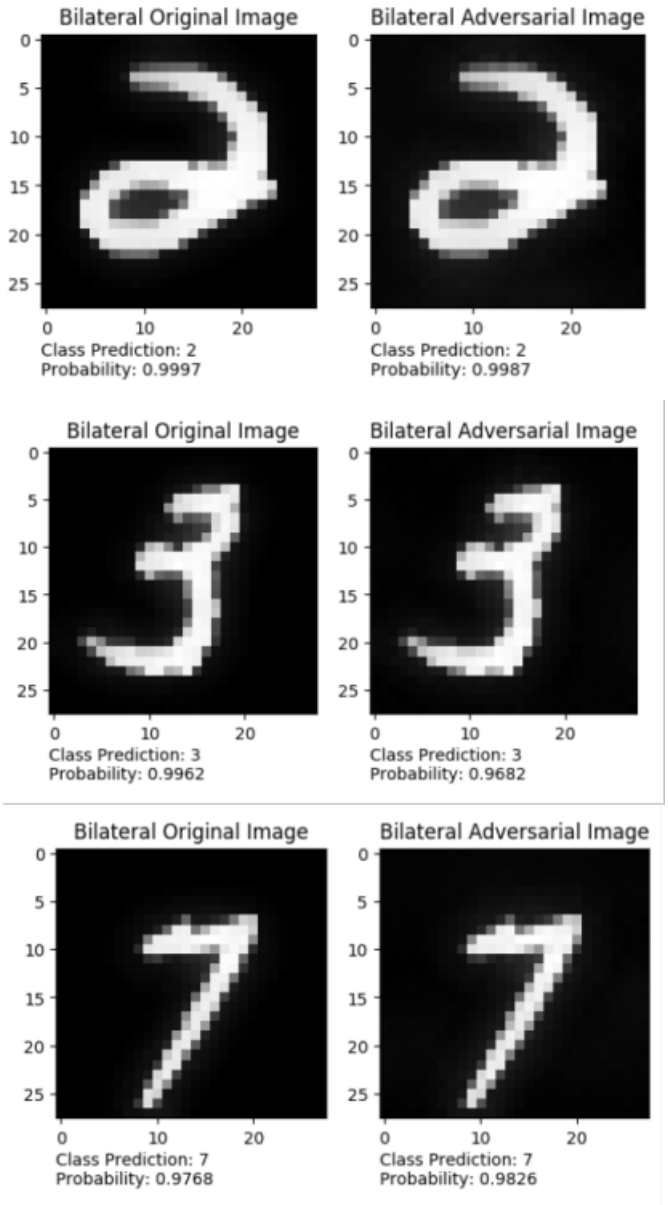

Figure 3.7: The results from filtering both the adversarial and original images with the bilateral filter.

3.2.6 Resizing

The last defense method experimented with was resizing the images to a smaller size and then resizing them back to their original shape. The motive for this experiment is due to the re-sampling that occurs when downsizing an image. This method causes some loss of information, and consequently, loss of the adversarial noise. The dimensions chosen for downsizing were 16 by 16. Other dimensions were tested, but they either retained too many adversarial artifacts or they caused too much pixelation in the image when upsizing back to size 28 by 28. The results of the resized image defense are shown in Figure 3.8.

It is clear that the adversarial effect has been visually minimized, but the images have become very pixelated. This defense performed the worst out of all of the defenses as seen in the metrics in Table 3.1. The adversarial effect is still visible in the images, and this technique does not perform as well as the others in terms of retaining visual integrity and accuracy.

3.3 Discussion

Overall, the best performing filters in this proof of concept were the bilateral and median filters. They did the best job at removing adversarial noise while preserving the image shape, and had the highest classification accuracies on adversarial images. They also did not significantly reduce the classification accuracy on the original images, which means that they provide a balance between original image classification and adversarial image classification.

This proof of concept showed that adversarial examples generated on a SVM could be defended against using image processing techniques. Although it was a data set of simple images, these experiments showed that it is possible to alter the input to the classifier and correct for adversarial noise. For the remainder of this thesis, some of these defenses and some new ones will be explored on more complex images on a convolutional neural network.

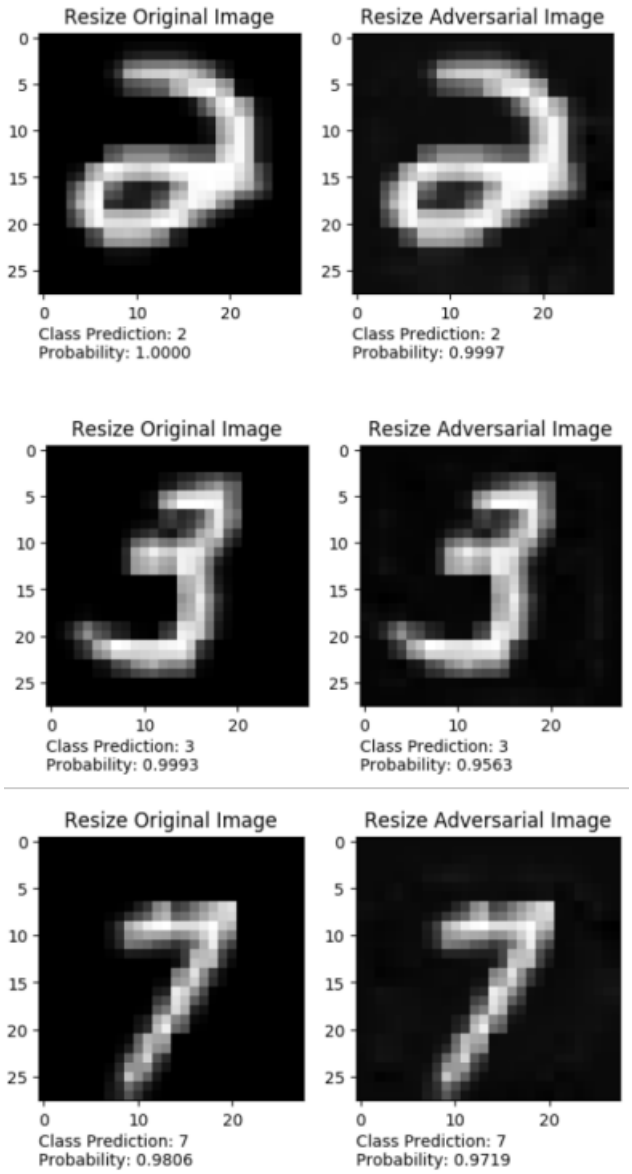

Figure 3.8: The results from resizing both the adversarial and original images.

Table 3.1: Accuracies of classifier on all original and adversarial images.

	Original Accuracy	**Adversarial Accuracy**
Unaltered	0.968	0.00065
Threshold	0.958	0.947
Gaussian	0.967	0.816
Median	0.966	0.896
Bilateral	0.967	0.868
Resize	0.956	0.738

Chapter 4

Data Sets and Networks

The remainder of this thesis will focus on defending against adversarial images created from a convolutional neural network. This chapter will describe the three data sets used: MNIST [31], CIFAR-10 [27], and a custom, higher resolution version of CINIC-10 [13] and their respective convolutional neural networks. Each data set was transformed with different filters because of the varying nature of the images contained within. The next chapter will show the results of the filtering defenses for each of the three data sets.

In addition to the FGSM and PGD attacks, this thesis will focus on white and black box attacks for both of the attack methods. Therefore, a minimum of two convolutional neural networks will be used to assess results. To reiterate, a white-box attack uses the gradient from the target model to create attacks. A black-box attack uses the gradient from a different model than the target model. This is examined because of the *portability* of adversarial images across networks.

4.1 Data Collection

This section will cover the three data sets used in this thesis. These data sets were chosen for their increasing complexity. The first data set, the MNIST Dataset of Handwritten Digits, contains the simplest images and provides a good foundation upon which to build increasingly more complex experiments. CIFAR-10 is a middle ground between the first and the last data set experimented with. It provides a more complex set of images to experiment

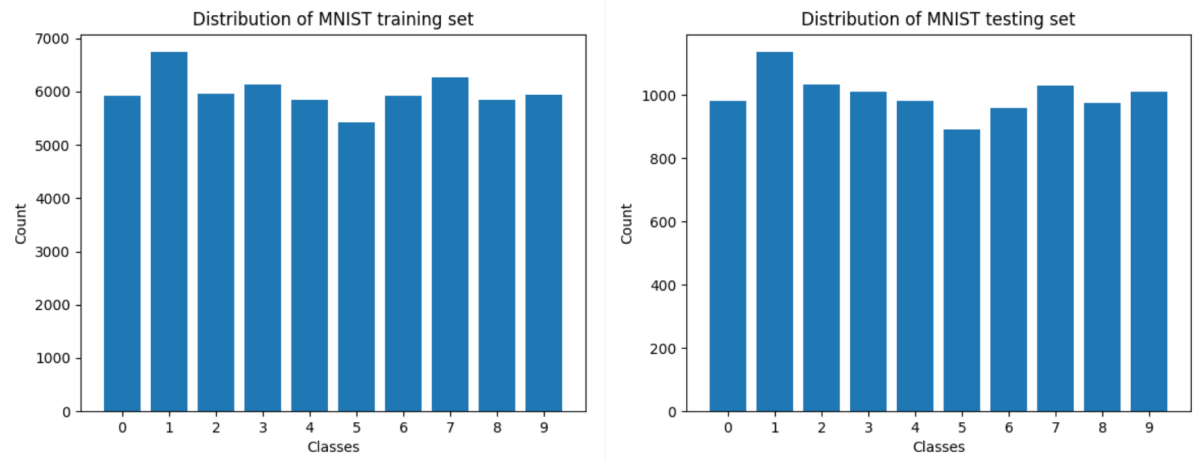

Figure 4.1: The distribution of the MNIST data set of handwritten digits in both the training and testing sets.

on, but still small and simple enough that results can be achieved quickly. The final data set, and the one most experimented with, is a custom data set collected by cross-referencing CINIC-10[13] and ImageNet[14]. It contains the largest and most complex images, which is important for allowing a wider range of experimentation without severely distorting the images.

4.1.1 The MNIST Dataset of Handwritten Digits

The proof of concept in the previous section briefly described the MNIST data set of handwritten digits, but this section will expand on the data set slightly more and why it was chosen to be included in this thesis. The full MNIST data set consists of 60,000 training examples and 10,000 test examples. The data set consists of 10 classes which are images of digits ranging from 0 to 9. The distribution of all classes in the data set, shown in Figure 4.1 is not completely balanced, but still balanced enough. In particular, class 1 is the most represented, while class 5 is the least represented.

The MNIST data set was chosen for these experiments because of its simplicity and availability. It is conveniently available from Keras [11] as part of the datasets library. All of the images consist of a single white digit centered in a black background, and each image

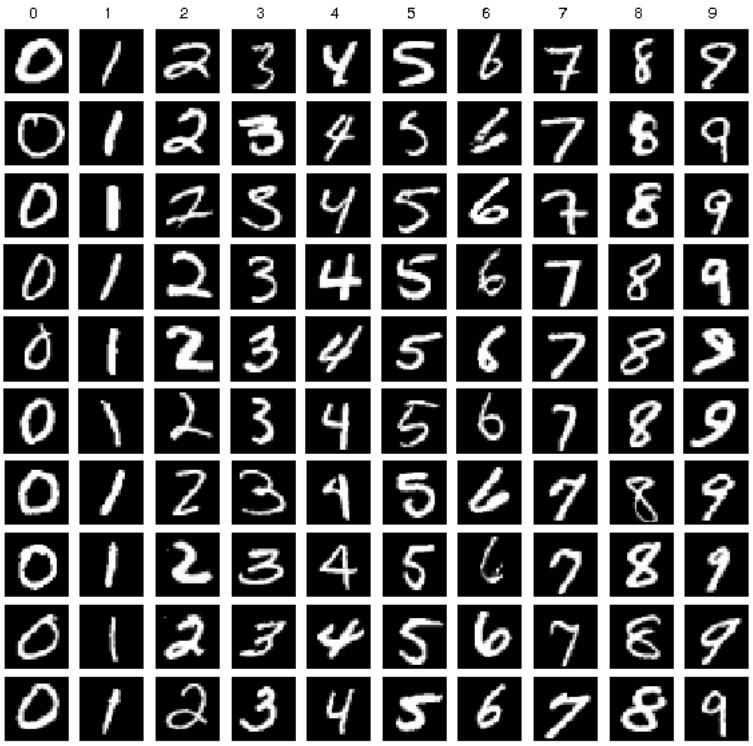

Figure 4.2: MNIST data set example by Lim et al. [33]

is a standard size of 28 by 28 pixels. The minimalism and size of these images make this data set easy to perform initial experiments on before moving onto more complex images. A visual example of the contents of the MNIST data set, made by Lim et al. [33], is shown in Figure 4.2.

4.1.2 CIFAR-10

In order to increase the complexity of the images being worked with, the next data set experimented with was CIFAR-10 [27]. This is a good intermediate data set to work with because the images are more complex than the MNIST data set but are still very small. The CIFAR-10 data set consists of 60,000 32 by 32 color images, split into ten classes. It is a labeled subset of the 80 million tiny images data set and was collected by Krizhevsky et al. [27] for use in studying how convolutional neural networks learn features of images. It

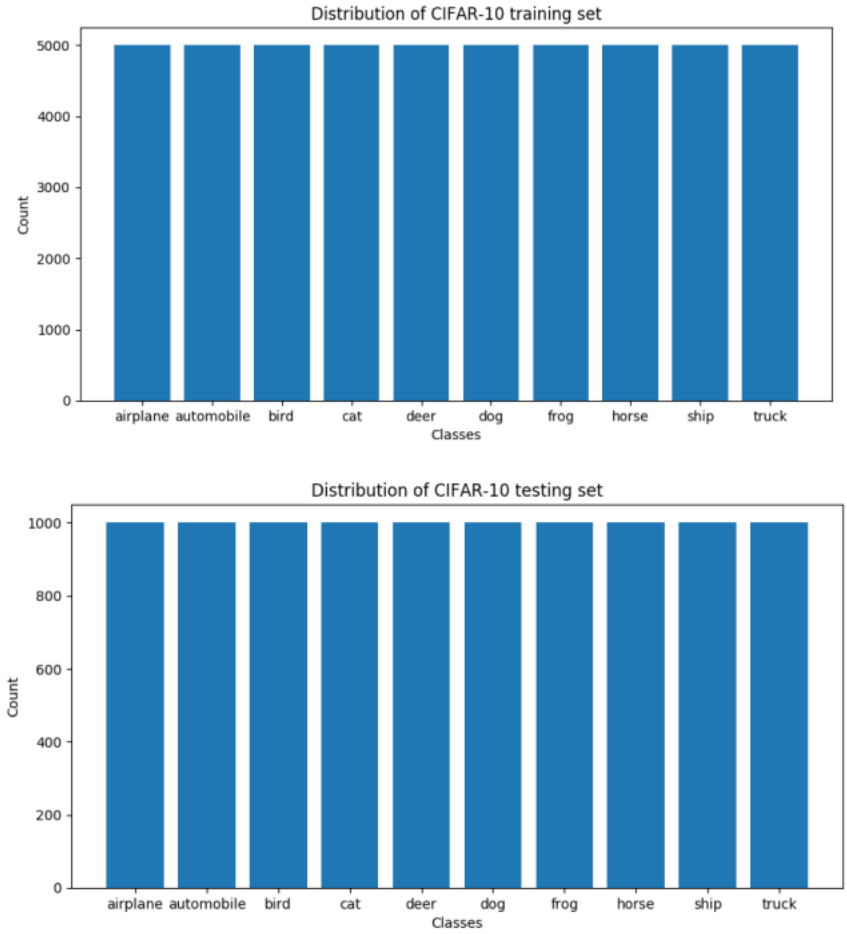

Figure 4.3: The distribution of the CIFAR-10 data set in both the training and testing sets.

contains a perfectly even distribution of 50,000 training images and 10,000 test images. A plot of the distribution of the data sets is shown in Figure 4.3.

As seen from the plot, there are ten classes of images that include animals and vehicles. The authors assure that there is no overlap between the classes – they are completely mutually exclusive. For instance, one may think that trucks and automobiles might overlap in the realm of pickup trucks, however, that is not the case. The truck class only includes big trucks, while automobile encompasses any vehicle except for pickup trucks. The authors provide a listing of all classes in the data set, including ten random images from each class. Their example is shown in Figure 4.4. This data set was chosen because of its small size

and convenience of availability, as well as providing a good stepping stone to further test experiments before moving onto larger images. The authors state that their training images are available in batches, however, for these experiments, all of the training images were used at once. The images are available on their website for download as a pickled object, which was very simple to unpack using Python's pickle library.

4.1.3 CINIC-10 Large

The last data set gathered for this thesis, and the most heavily experimented on, is a custom version of the CINIC-10 data set. The authors claim on their website that "CINIC-10 Is Not ImageNet or CIFAR-10". Their reasoning for developing this data set is because they believe that CIFAR-10 is too simple and easy, while ImageNet [14] is too large and difficult. At the time of writing this thesis, ImageNet contains 14,197,122 images, divided into over 100,000 synsets, with a minimum of 1,000 images per synset. A synset, or synonym set, is a group of words arranged in a hierarchical structure. For example, the toad/frog synset might have child synset images of tree frog, fire bellied toad, etc... Additionally, the images on ImageNet are not easily accessible to download because ImageNet requires that one makes an account and send a request to download the original images. This is because many of the images are copyrighted. However, the images' URLs are publicly available to use for downloading them.

With this in mind, the authors of CINIC-10 created a data set that contains the same ten classes as CIFAR-10, with the same image sizes as CIFAR-10, but with 4.5 times more images than CIFAR-10. The CINIC-10 data set contains 270,000 images and is available for download from their website as a tarball. However, for the purposes of this thesis, the 32 by 32 image size from CINIC-10 is too small and a larger version needed to be scraped. The authors conveniently put a file on their website that for each image lists: the synset, the image number, the set in the CINIC-10 data set (training, validation, or testing), and the class which it belongs to out of the ten stated previously. They mentioned that they used the Fall 2011 release of all ImageNet URLS to extract their images, which is available for download on the ImageNet website.

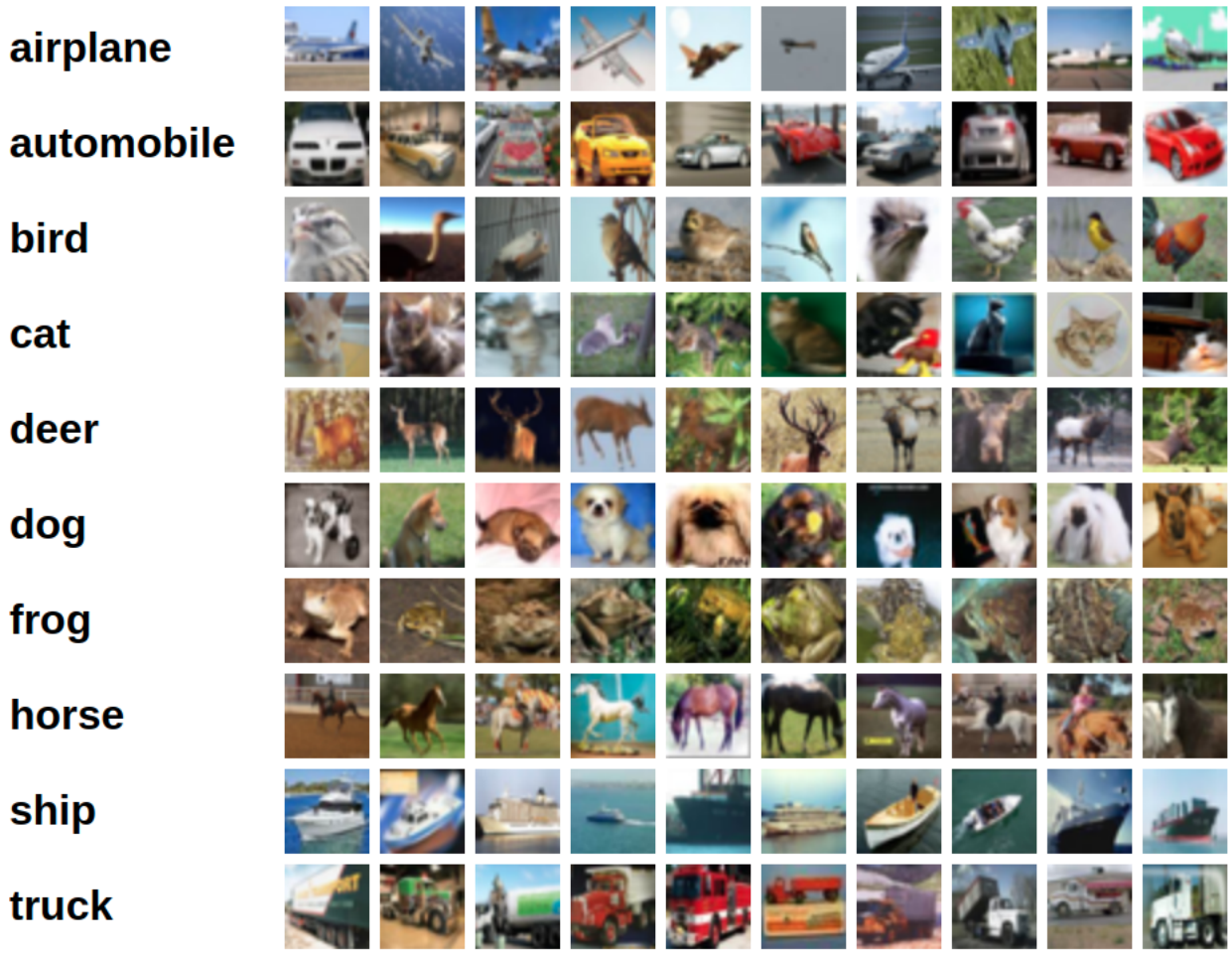

Figure 4.4: An example from the authors' website of the contents of the CIFAR-10 data set [27].

In order to extract a larger version of the CINIC-10 data set, a Python script was written that cross-references the file the CINIC-10 authors put on their website and the list of Fall 2011 URLs from ImageNet. Each synset and image number in the CINIC-10 file was used to find the corresponding URL in the Fall 2011 file, which was then fetched, resized to 224 by 224 pixels, sorted into the appropriate set and class, and downloaded locally. The script took a few days to run, and the resulting data set is shown in Figure 4.5. The authors of CINIC-10 state that their data set is completely evenly distributed in each subset, with 9,000 images per class per subset, split into train, validation, and test. However, scraping the links from the Fall 2011 URLs revealed that many of them were broken, so these results show the best effort scraping from each class. The final data set size resulted in 79,884 train images and 39,975 test images. The validation and testing sets were combined into one testing set. The images in the CINIC-10 Large data set are very similar to the ones in CIFAR-10, shown in Figure 4.4.

The motivation for gathering larger images comes from the nature of adversarial images. They are intended to look unsuspecting to the human observer while tricking a neural network into misclassifying them with high confidence. Such small images as the ones in CIFAR-10 and CINIC-10 do not have a lot of detail in them, and therefore would not be used in any real scenario where an attacker would perturb an image. The images are supposed to be easily recognizable to the human eye, however, with only 1024 pixels of detail, even an unaltered image from the CIFAR-10 or CINIC-10 data set might be to difficult to identify. Additionally, filtering methods on such small images will inevitably distort the small image beyond any sort of recognition, which might make a neural network misclassify an image simply because it is too blurry. Therefore, for better results, it was necessary to gather images with more detail than those available in the previously mentioned data sets. This allowed for a larger range of experimentation than would have possible within the limits of the smaller images.

One random example from each class is shown in Figure 4.6. While the images are significantly larger than the previous classes, they are still rather small compared to the 4k HD images that are the norm on most websites. There are two reasons for not using such large image sizes. The first is due to computational complexity. Such large images

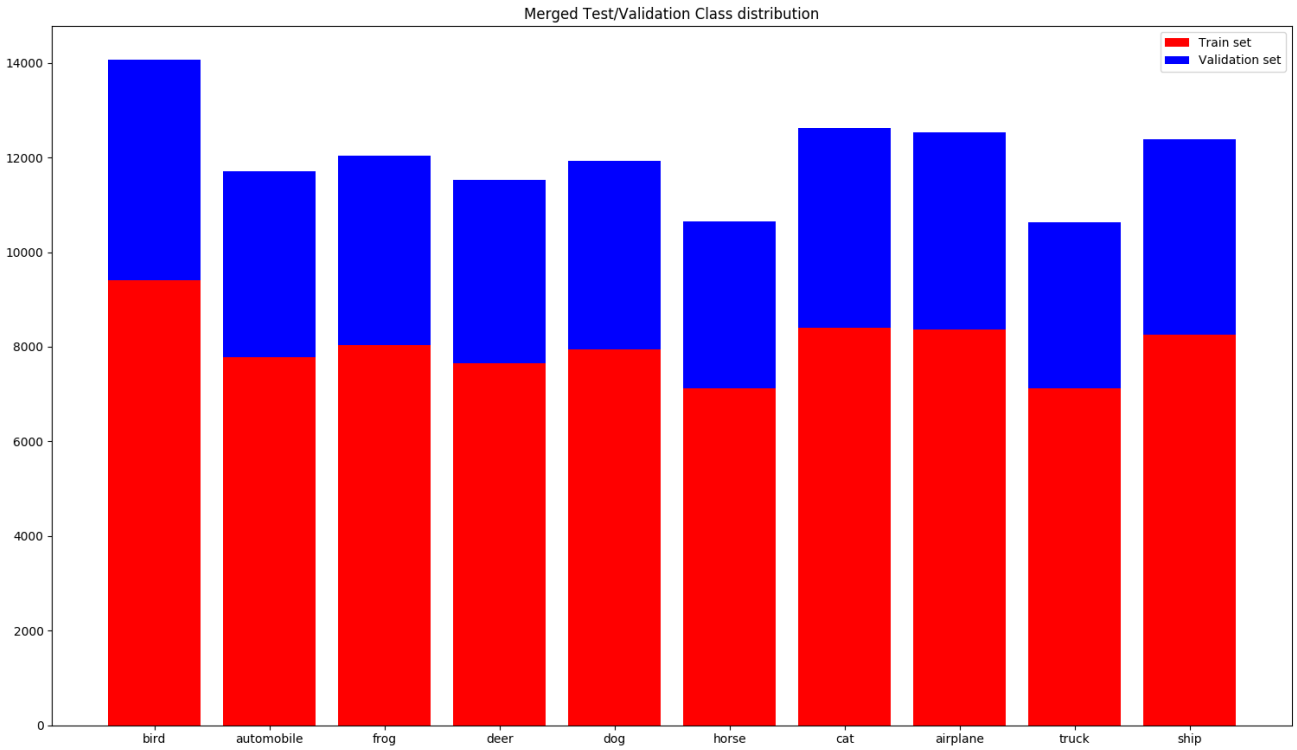

Figure 4.5: The CINIC-10 Large data set distribution after merging the testing and validation sets.

would require significant computing time. These 224 by 224 images already require much more computing power than the smaller 32 by 32 images, so increasing the size would only exponentially increase the run time of the experiments. The second is due to the input size of the pretrained network that was used in the CINIC-10 Large experiments. This pretrained network will be discussed later in this chapter.

4.2 Convolutional Neural Network Structure

This section will cover the convolutional neural network architectures used for each data set. Each data set had a different neural network created for it and each data set's neural network was attacked with a white-box and a black-box attack. In order to do this, two models were created in each section at minimum. However, two different models does not necessarily mean two different neural network architectures. A fully trained neural network, or model, is one that has learned weights to give the highest accuracy. Even by training the same neural network architecture on the same data set twice, the weights will not be identically calibrated in the two resulting models. Therefore, the two models can be considered different, and an attack on one using the gradient from a different model is considered a black-box attack. All neural networks were made using Keras [11].

4.2.1 MNIST Convolutional Neural Network

Since MNIST is such a simple data set, a simple convolutional neural network was created for experimentation. The structure of the convolutional network used for the MNIST experiments is shown in Figure 4.7. This model is broken in basic blocks which consist of convolutional layers, max pooling layers, and dense layers. These are the essential building blocks of a convolutional neural network. Other types of layers exist, but those will be discussed when they are encountered in the next subsections.

Briefly, a convolutional layer works by convolving the input with a filter to extract features. This results in a feature map that can be used to train the network. Features include simple shapes such as edges, or more complicated shapes such as structures. Normally convolutional layers deeper into the network will extract more complex features. Max pooling

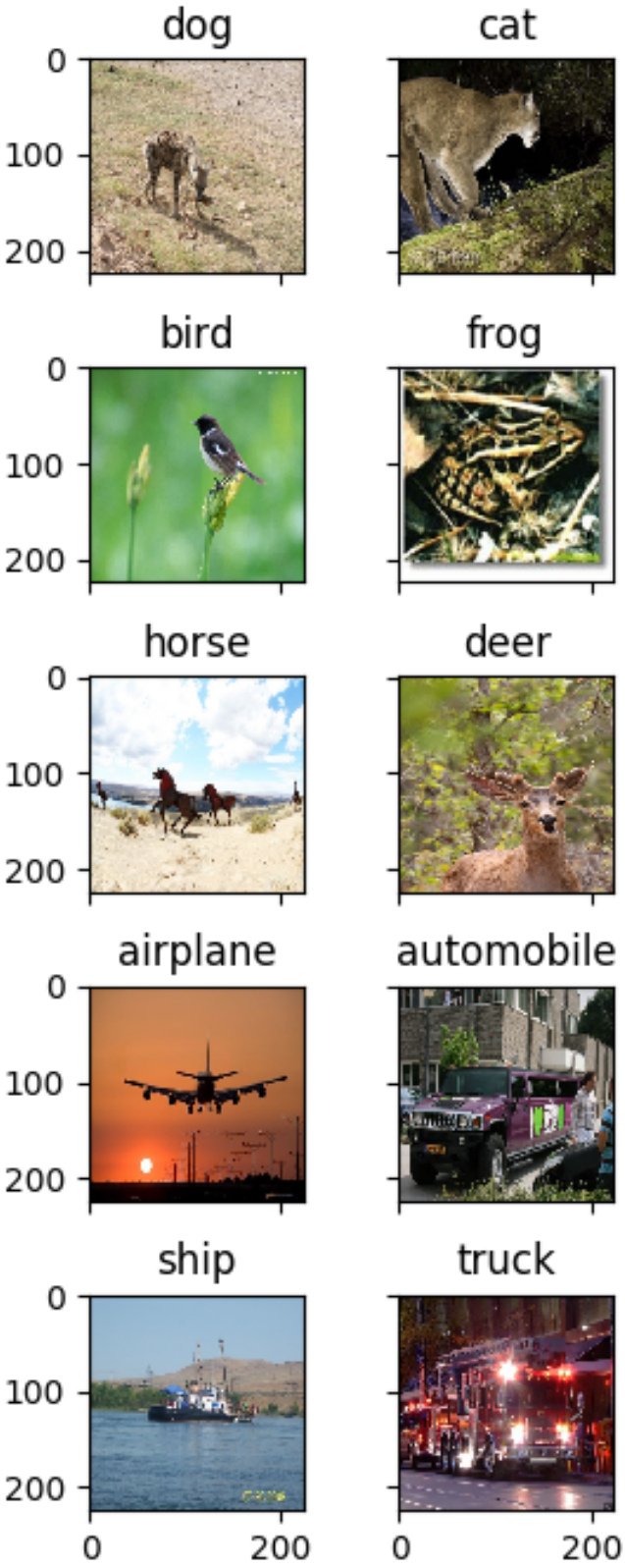

Figure 4.6: One random example from each class of the CINIC-10 Large data set.

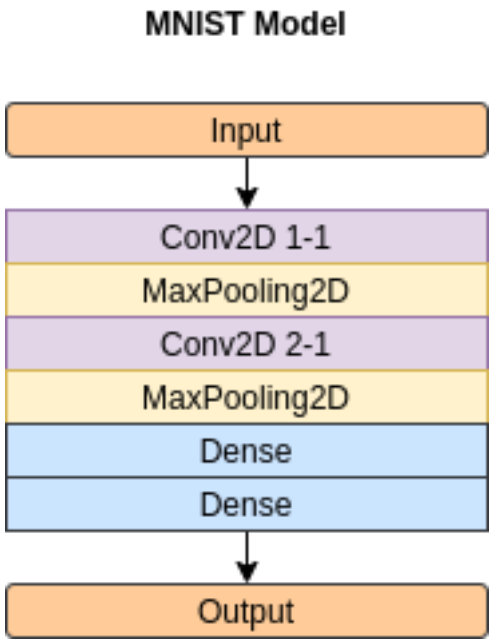

Figure 4.7: The structure of the convolutional neural network used for the MNIST experiments.

layers work with convolutional layers to provide a way to down-sample the feature maps extracted from the convolutions. Max pooling is important to make the model robust to the position of the image content. A typical size for pooling is 2 by 2 with a stride of 2, meaning that it will take the maximum value of each 2 by 2 patch in the feature map, moving 2 pixels at a time, and return a set of smaller feature maps. The dense layers are fully connected neural networks that update weights between neurons before making a final class output in the softmax layer.

In order to train this model, all images were scaled to be between 0 and 1 instead of 0 and 255. The neural network was trained for five epochs. Two runs of training this network are shown in Figure 4.8. Both runs achieve high accuracy on the training and validation set, at around 99% accuracy for both models.

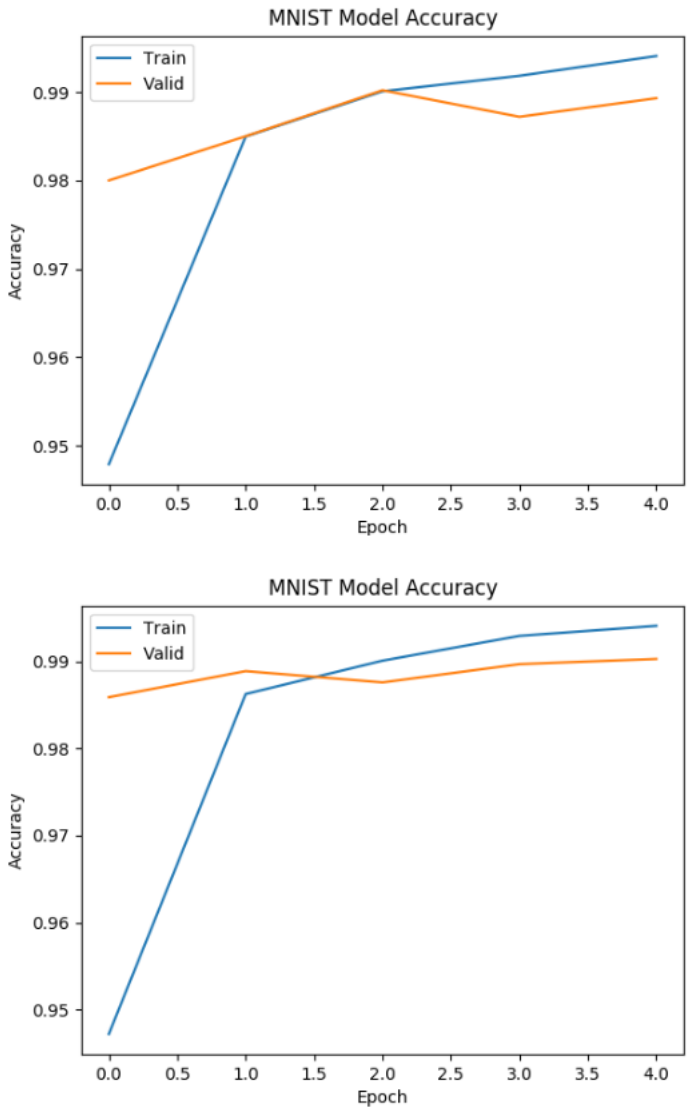

Figure 4.8: The two models used in the experiments.

4.2.2 CIFAR-10 Convolutional Neural Network

The CIFAR-10 convolutional neural network took some trial and error to get to a high accuracy. The first two attempts at creating a neural network architecture resulted in validation and test set accuracy of roughly 70% to 80% each time the model was trained. The third attempt resulted in a highly accurate residual neural network that was used for experimentation. Its accuracy on the validation and testing set was in the low 90% range. All three networks will be discussed in this section.

The first neural network's structure is shown in Figure 4.9. Because this data set is more complicated than MNIST, it needed a deeper neural network. This network has more convolutional layers and max pooling layers so that it can learn more features during training. Additionally, it has dropout layers to prevent overfitting and uses ReLU activation.

Two strategies were attempted to get at least 90% accuracy with this model. The first strategy was to train the model for a large number of epochs. There was no preprocessing on the images except to scale them to between 0 and 1. The optimizer used was Adam, included in Keras. The model was trained for 100 epochs, and it was apparent that the accuracy was plateauing at around 75%. In a second attempt to increase the accuracy with this same architecture, the images were augmented during training. The augmentations consisted of giving the neural network images that were randomly rotated within a range from 0 to 180 degrees, shifting the images horizontally and vertically, and randomly flipping the images both horizontally and vertically. Other augmentations were attempted as well, such as ZCA whitening, but these reduced accuracy. The progress of training the model with both of these strategies is shown in Figure 4.10.

It is clear from looking at the difference in the validation and training set that the model begins to overfit between the twentieth and thirtieth epoch. The training set accuracy continues to increase at a small but steady rate, while the validation set plateaus at around 75% accuracy for both of the model training strategies. This low performance is not ideal for testing the effect adversarial images have on a neural network's base accuracy. To remedy this, an alternative network structure was tested in an attempt to increase accuracy.

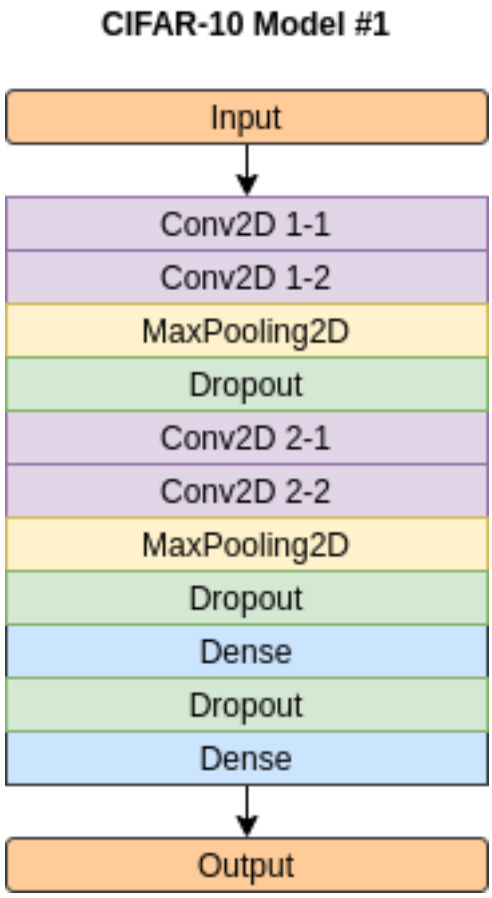

Figure 4.9: The architecture of the first convolutional neural network attempt for training a model on CIFAR-10.

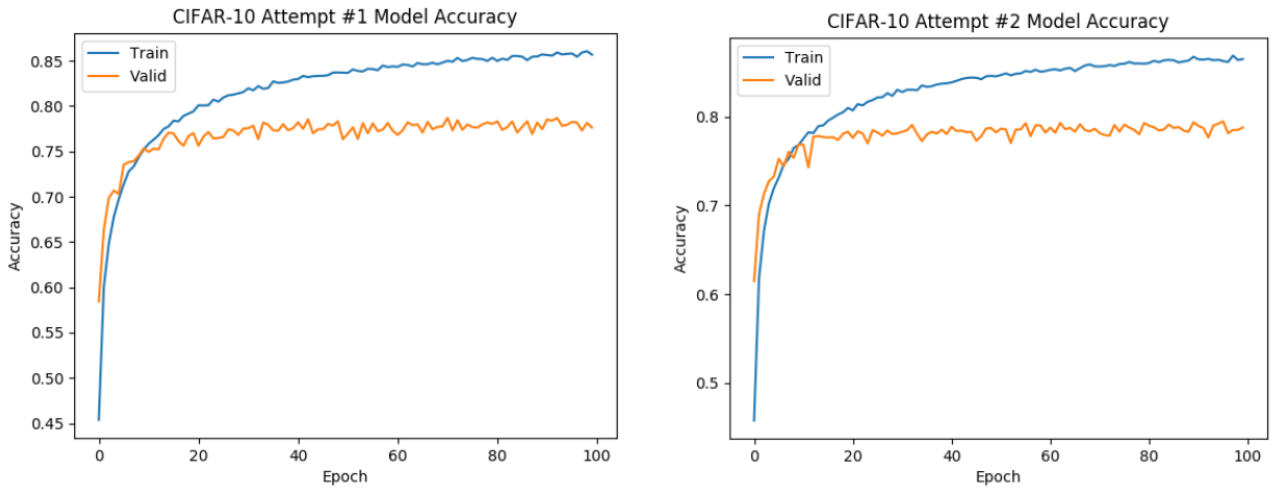

Figure 4.10: Training and validation accuracy over 100 epochs with both strategies attempted on the first CIFAR-10 model.

The next convolutional neural network created was deeper than the previous one and incorporates layers with batch normalization. Batch normalization was created by Ioffe et al. [26], and this process claims to reduce internal covariate shift, which should improve training speed. Internal covariate shift refers to the instability of the distributions of the internal nodes, which can cause the network to get stuck in a saturated mode if not addressed. A saturated mode is one where the nodes have extreme values in the positive or negative direction and applying a change in the weights during training will not significantly change the value of the node. Therefore, batch normalization whitens the input to each layer, meaning it normalizes the input to have a mean of 0 and a variance of 1. In order to make this work with stochastic gradient descent, which seeks to minimize the loss, this normalization is adjusted with two parameters: multiplication of gamma and addition of beta. This method claims that internal covariate shift will be reduced, and training will be accelerated.

The architecture of the second neural network is shown in Figure 4.11. In addition to using batch normalization between layers, the input was normalized as well. The exponential linear function, or ELU, is used as the activation function. This was chosen due to the claim that it can produce more accurate and quick results than the traditional RELU. This is

because unlike RELU, ELU allows for some some negative inputs which smoothly transition to positive inputs [12].

In addition to normalizing the inputs, the images were augmented with rotations within fifteen degrees, slight width and height shifts, and random horizontal flips. The model was trained for 100 epochs with a learning rate scheduler such that the learning rate reduces when beyond a certain number of epochs. Additionally, RMSProp was used as the optimizer for its ability to avoid exploding and vanishing gradients.

The results of training this second network for 100 epochs is shown in Figure 4.12. The learning rate was set very low for this network, at 0.00005, and three dropout layers were used, which explains why the validation accuracy is equal to or greater than the training accuracy. Looking at the model, it is clear that the accuracy does get high, plateauing at roughly 85% accuracy on the validation set at around the 120th epoch. Although 85% accuracy is fairly high, research showed that there was another model architecture available that claimed to reach roughly 90% accuracy on the CIFAR-10 test set.

This accurate neural network was created by by He et al. [23]. He et al. were the first people to introduce the concept of a *residual neural network*, which is a network that has *skip layers* [22]. The residual neural network, or ResNet, was created in order to address the problem of vanishing/exploding gradients in deep networks. As layers continue to be stacked in a deep neural network, the accuracy becomes saturated and begins to degrade [22]. This saturation problem has been mostly resolved via normalization, which is what the previous neural network structure used in both normalizing the inputs and intermediate layers. However, the batch normalization does not provide any guarantees that optimization will be reached because it is still an approximation and may still cause information loss.

The ResNet solves this degradation problem by using "skip layers", shown in Figure 4.13. The basic concept behind skip layers is to stack them to create a deep neural network, and optimize the *residual* mapping. The authors describe the desired underlying mapping $H(x)$ and the equivalency of this mapping in terms of $F(x) := H(x) - x$. They reformulate this to the residual mapping $F(x) + x$. They hypothesize that it is simpler to optimize the residual mapping, especially in the extreme case where the identity, x, is optimal, it would be simpler to push the residual to zero than to x. Essentially, the residual function is easier to optimize.

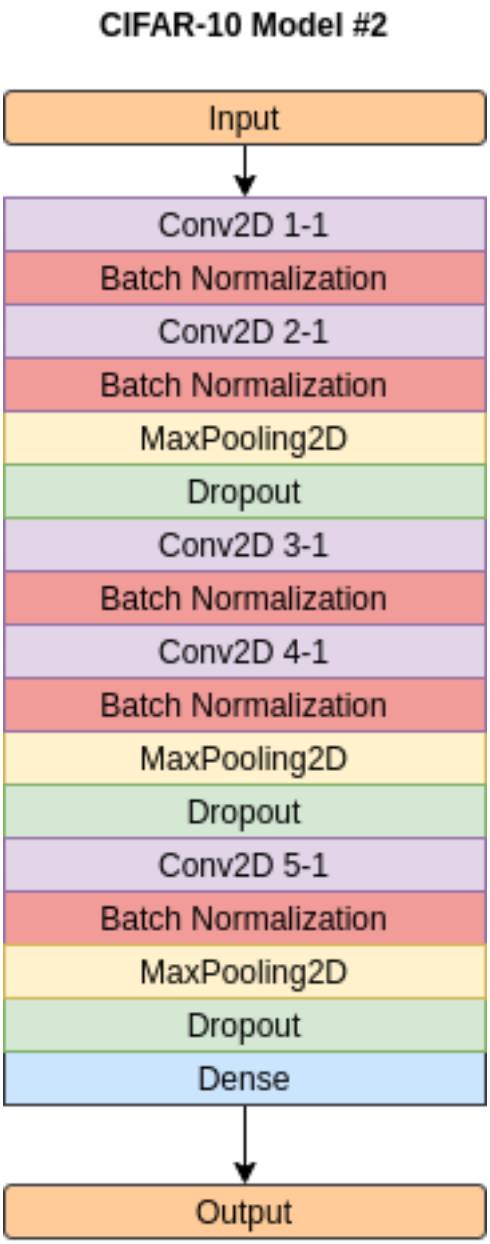

Figure 4.11: The architecture of the second convolutional neural network attempt for training a model on CIFAR-10.

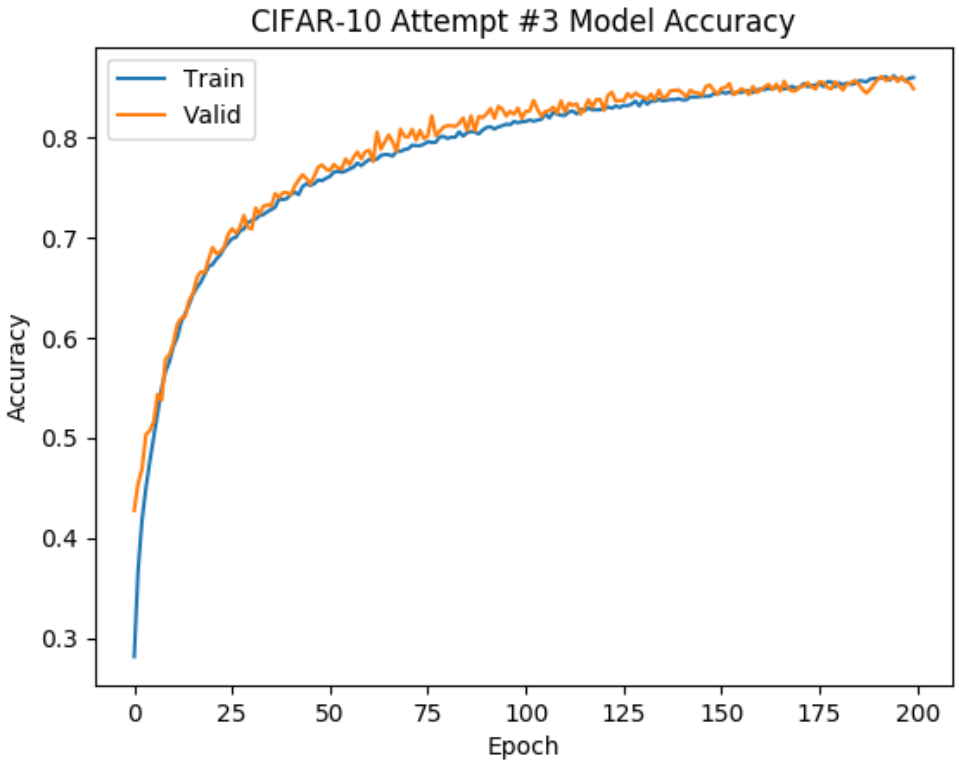

Figure 4.12: Training and validation accuracy over 200 epochs on the second convolutional neural network architecture.

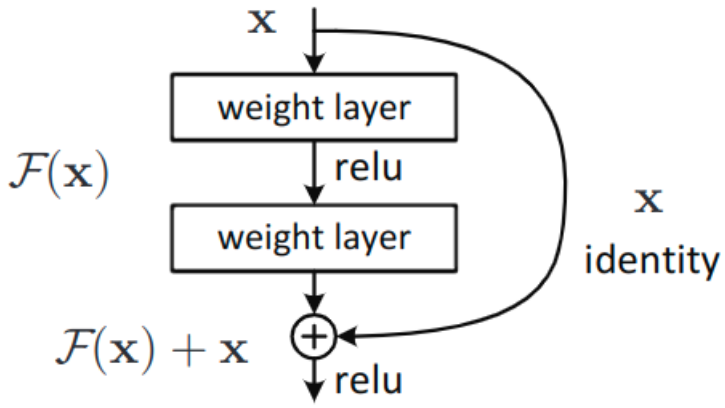

Figure 4.13: An illustration of a residual layer from He et al. [22].

Additionally, the authors note that the skip layers in the network, which they refer to as "identity shortcut connections", do not add extra parameters or complexity. If *F(x)* ends up not contributing any useful weight updates, meaning that is zero, then the identity will be fed forward through the network, and the result will be as if the layer being skipped over did not exist in the first place. Whereas if there was no skip connection, then there would be some information loss and the network would behave more poorly.

Given this brief background on the residual building block, the ResNet used in this thesis is the second one built by He et al. [23]. This network was slightly more accurate than the previous network because they modified the structure of their residual block. They experimented with various residual block architectures and found that removing the ReLU at the end of the skip layer, and having simple addition with full pre-activation reduced loss on the CIFAR-10 data set by 0.4%. All other architectures increased loss. This is the residual building block used in the ResNet in this thesis.

The structure of the ResNet for CIFAR-10 is shown in Figure 4.14. The second ResNet created by He et al. is conveniently available from Keras [11] on their website. In addition to the residual blocks described above, the network makes use of pixel mean subtraction, which is the process of subtracting the mean value of each channel in the image from each pixel. The authors use this method in their paper because it normalizes the inputs by having

each input centered around zero, which increases accuracy. The network uses a learning rate scheduler that decreases the learning rate with higher epochs. Additionally, it uses data augmentation with small random horizontal and vertical shifts and random horizontal flips.

The model trained for 100 epochs and produced an accuracy on the validation and testing set of 91%, shown in Figure 4.15. The sudden jump in the accuracy is due to the learning rate scheduler, which reduced the learning rate from 0.001 to 0.00001 at the eightieth epoch. The spikiness in the validation accuracy is because of the high learning rate, which is clearly addressed when the learning rate is reduced and the accuracy trend smooths. This network will be the one used in all CIFAR-10 experimentation and will be used to produce a minimum of two models to create and test white and black-box attacks.

4.2.3 CINIC-10 Large Convolutional Neural Network

The last data set, and the one that will be most heavily experimented on, is the CINIC-10 Large data set. Building a convolutional neural network for images of size 224 by 224 would likely have required a large effort based on the difficulty to achieve a highly accurate neural network for the CIFAR-10 data set, which has much smaller images. In order to get a very accurate neural network quickly, *transfer learning* was used. Transfer learning is the process of taking a pretrained convolutional neural network, which has already been trained on a very large data set and has high accuracy, and modifying it to fit a different task. In this case, the pretrained convolutional neural network used was VGG16 [46], a deep neural network that won the ImageNet ILSVRC-2014 competition, with a top-5 test accuracy of 92.7%. It beat AlexNet due to its smaller receptive field in which it is better able to filter out background noise in an image and extract important features instead. It achieves this because of its small, stacked convolutional layers. VGG16 is pretrained on ImageNet, which took the developers several weeks to train on NVIDIA Titan Black GPUs. It is conveniently available from Keras [11] in the applications library.

VGG16 was used a feature extractor, meaning that all of the convolutional and max pooling layers were frozen, and the last fully connected dense layers were kept unfrozen. Since VGG16 is trained on ImageNet, which has 1,000 classes, the softmax layer had 1,000 output nodes. This layer was popped off of the network and replaced with a softmax layer

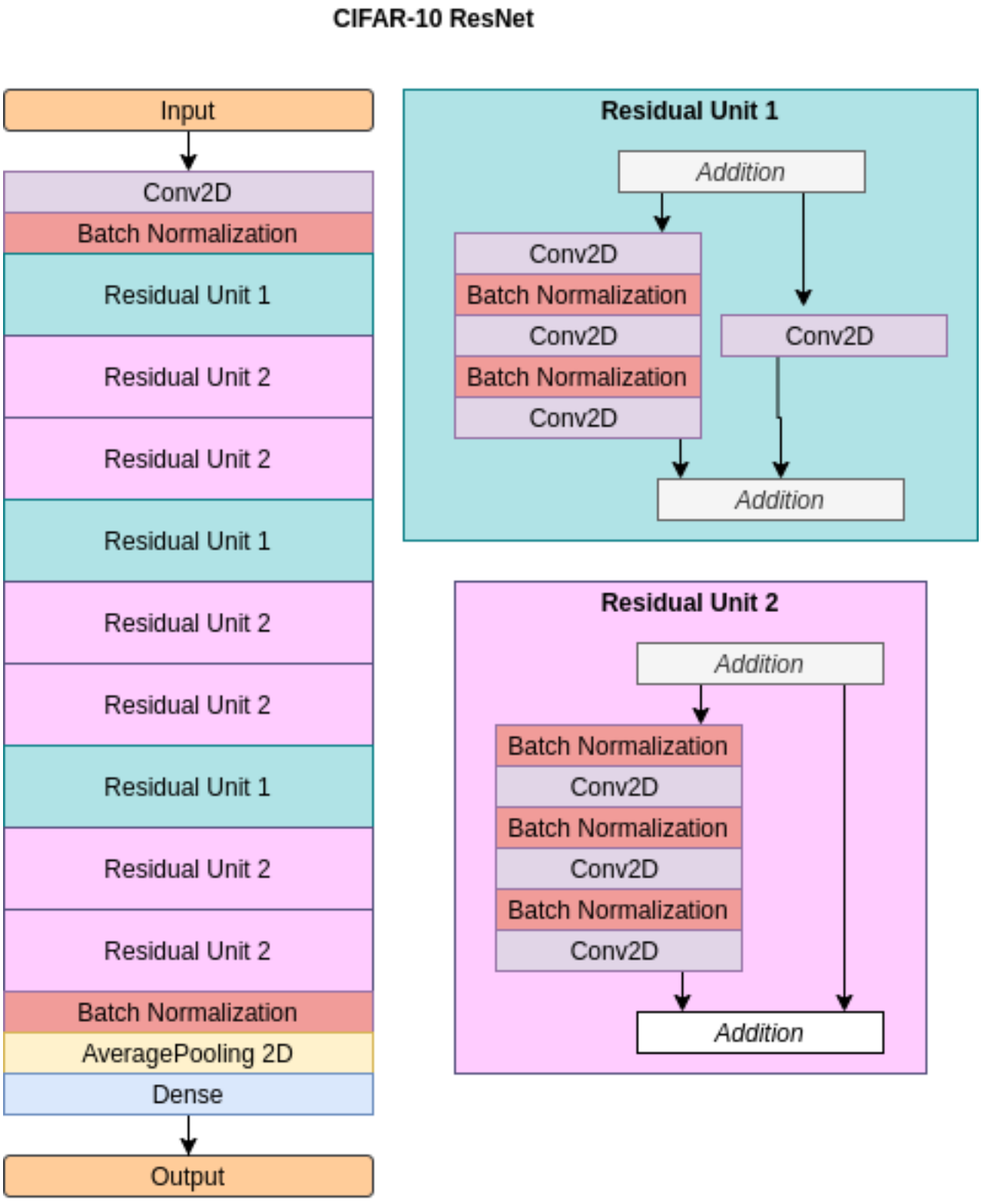

Figure 4.14: The structure of the ResNet used for CIFAR-10.

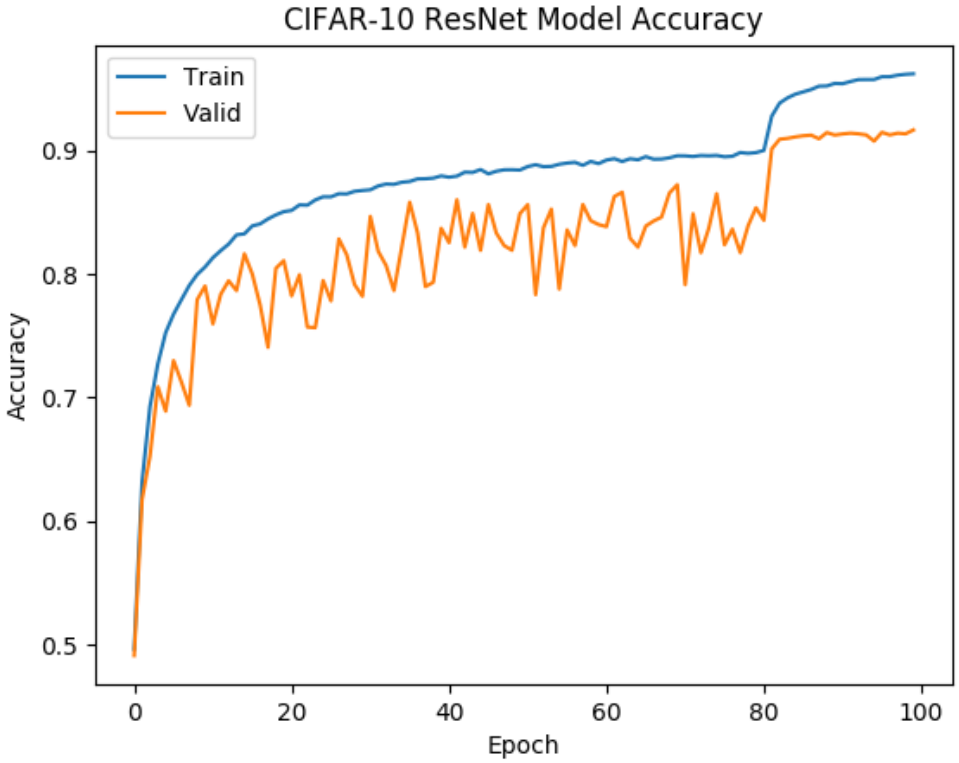

Figure 4.15: Training and validation accuracy over 100 epochs on the ResNet.

consisting of 10 nodes, one for each class. The default input size of the model, 224 by 224, was kept as default. To continue staying consistent with VGG16, all images were scaled to 0-1 and mean pixel subtraction applied, similar to the method used in the ResNet for CIFAR-10. The structure of the VGG16 network is shown in Figure 4.16.

The dense layers of the VGG16 network did not require many epochs of training to reach over 90% accuracy on the testing set. The training process is shown in Figure 4.17. These layers only needed to be trained for ten epochs to achieve 91% accuracy on the validation set. The accuracy likely started very high because the model was already trained on a super-set of these images. Instead of training on over a million images with one thousand classes, the model's task was now to simply train on 79,884 images with 10 classes.

This conclusion would usually not be a problem for a normal classification task, however, in this case, it was a problem. VGG16 is highly accurate and extremely confident in its predictions, at 100% confidence, meaning that many of its predictions did not produce any loss. In order to retrieve the gradient, there needed to be some loss to begin with. VGG16's predictions were confident to the point that there was almost no loss. Both FGSM and PGD need some loss to compute optimal perturbations because they require the sign of the gradient with respect to the input to calculate in which direction to push each pixel intensity. The most perplexing part of the model's overconfidence was that it not only displayed this behavior on correct predictions, it was 100% confident on incorrect predictions as well. On the entire testing set, the model was 100% confident on 35,620 correct classifications and 2,682 incorrect classifications. The remaining 827 correct and 858 incorrect classifications were the only ones eligible to create adversarial images from. This is, of course, unacceptable for the purposes of these experiments, so several methods were attempted in order to create some gradients on the VGG16 network.

Before starting these methods, however, a sanity check were performed to ensure that there was no human error involved. The first check involved ensuring no overlap in images between the test and train set. Having the same images in the training set as the testing set would be a plausible explanation for the model's overconfidence. A script was written that checks the difference between each image in the training and testing set. Each image in the testing set was subtracted from each image in the train set. If this subtraction equaled

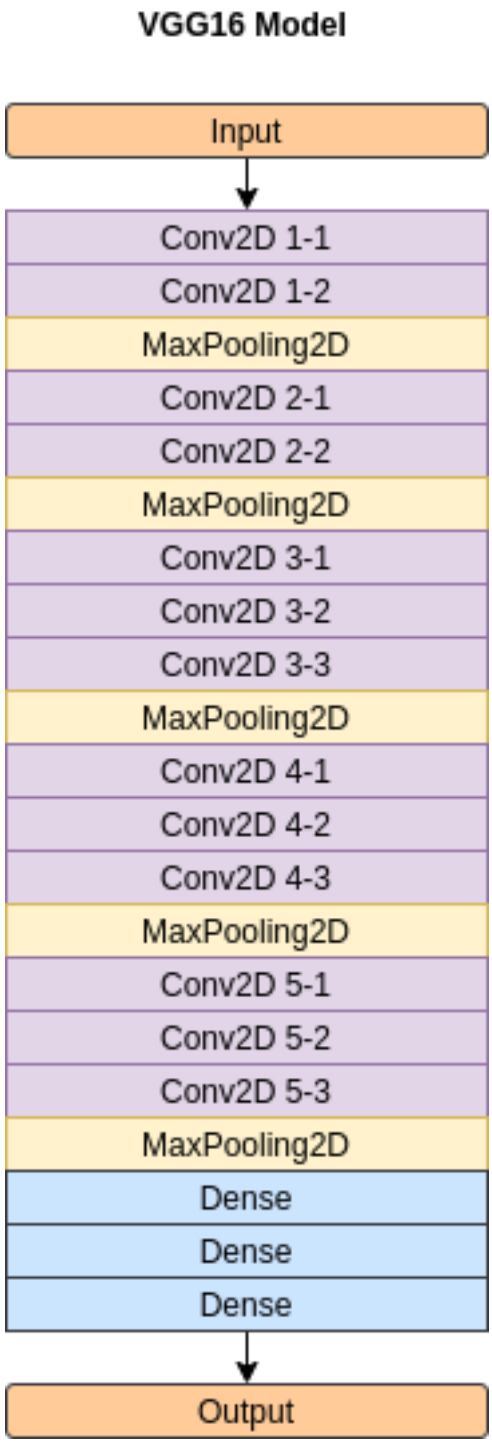

Figure 4.16: Structure of VGG16 CNN.

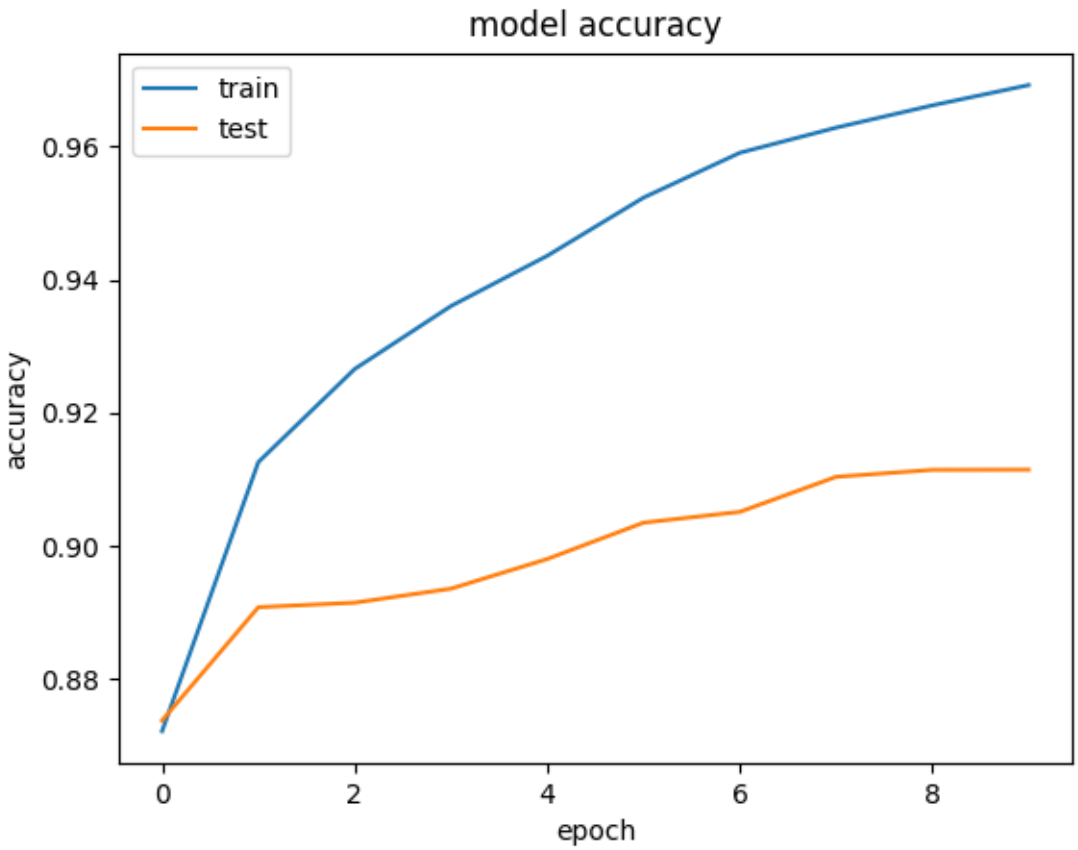

Figure 4.17: Training and validation accuracy over 10 epochs on VGG16.

zero, then the images would be identical, since each pixel is exactly the same. The result of running of this script found no such images.

With human error not being a factor, modifying the network was the next strategy. Some dropout layers were added to the model between the dense layers. Dropout layers are used as a means of preventing overfitting during training in a neural network by randomly setting some nodes to zero in the output layers of the model. This effectively makes the node disappear from the model and forces the model to only have a fraction of the weights to learn with in each epoch. Different dropout layer placements and dropout percentages were experimented with, but overall, no amount of dropout produced any substantial loss increase. The model continued to be overly confident, but only slightly less so with the addition of dropout layers. Reducing the data set and adding random noise to the images did not increase the model's loss either.

The last strategy was to add dropout layers and Leaky ReLU activations. The convolutional and max pooling layers were kept frozen in the architecture for their feature extraction capabilities, but the final dense layers were replaced. The model is simply a modified VGG16 network with a significant amount of regularization applied. To differentiate this regularized model from the original, it will be referred to as Regularized-VGG16, or R-VGG16 for short.

This strategy finally produced the much needed gradients to create adversarial images with. The model produced 89% non-zero gradients on the testing set. However, the accuracy was not as high as desired, so some more measures were taken in an attempt to increase accuracy. The mean pixel subtraction step was removed from the process. Instead, no preprocessing was applied to the images, including the scaling to between 0-1. This removal of scaling did not make a difference, but the full 0-255 range was kept in the images. This method marginally increased accuracy to 88%. The final model structure is shown in Figure 4.18. This figure might look like only a couple of dropout layers were added, but a significant amount of regularization was added to other places in the model that the structure doesn't show. The fact that this regularization made it so the model stopped being so overconfident lends itself to the hypothesis that the VGG16 model was overfitting.

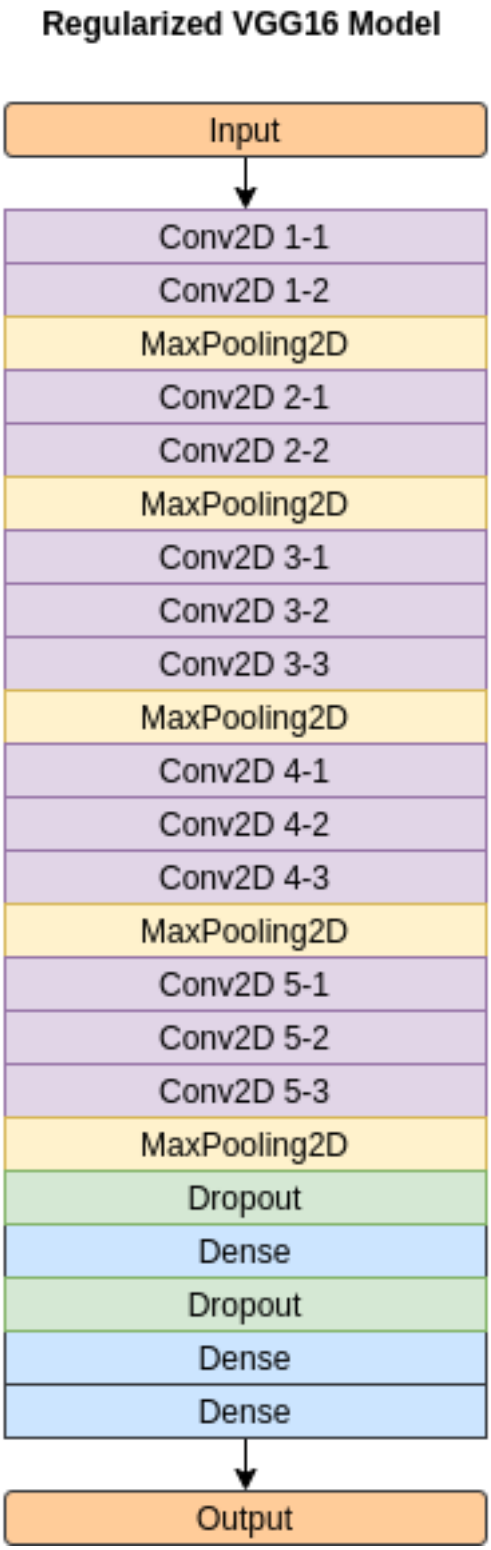

Figure 4.18: Final structure of R-VGG16.

Unfortunately, the full data set was not feasible to use. Some of the filters used in experimentation were very computationally expensive, which was discovered after attempting to work with the full data set for a while. The most expensive filters were the TVD filters. Additionally, the PGD attack is very expensive, and the nature of creating the gray-box attacks required these computations many times over. The makeup of the data set after reducing its size is shown in Figure 4.19. The training set was reduced to 3,000 images per class and the testing set was reduced to 1,000 images per class, for a total of 30,000 training images and 10,000 testing images. This was not ideal, but running all of the gray-box experiments would have taken several months if the full data set was to be used. The results of training and testing R-VGG16 on the reduced size data set is shown in Figure 4.20. The final accuracy on this reduced training set is 85%, which is not ideal but a necessary trade-off for the purposes of the experiments. This smaller data set size will be used for all CINIC-10 Large experiments.

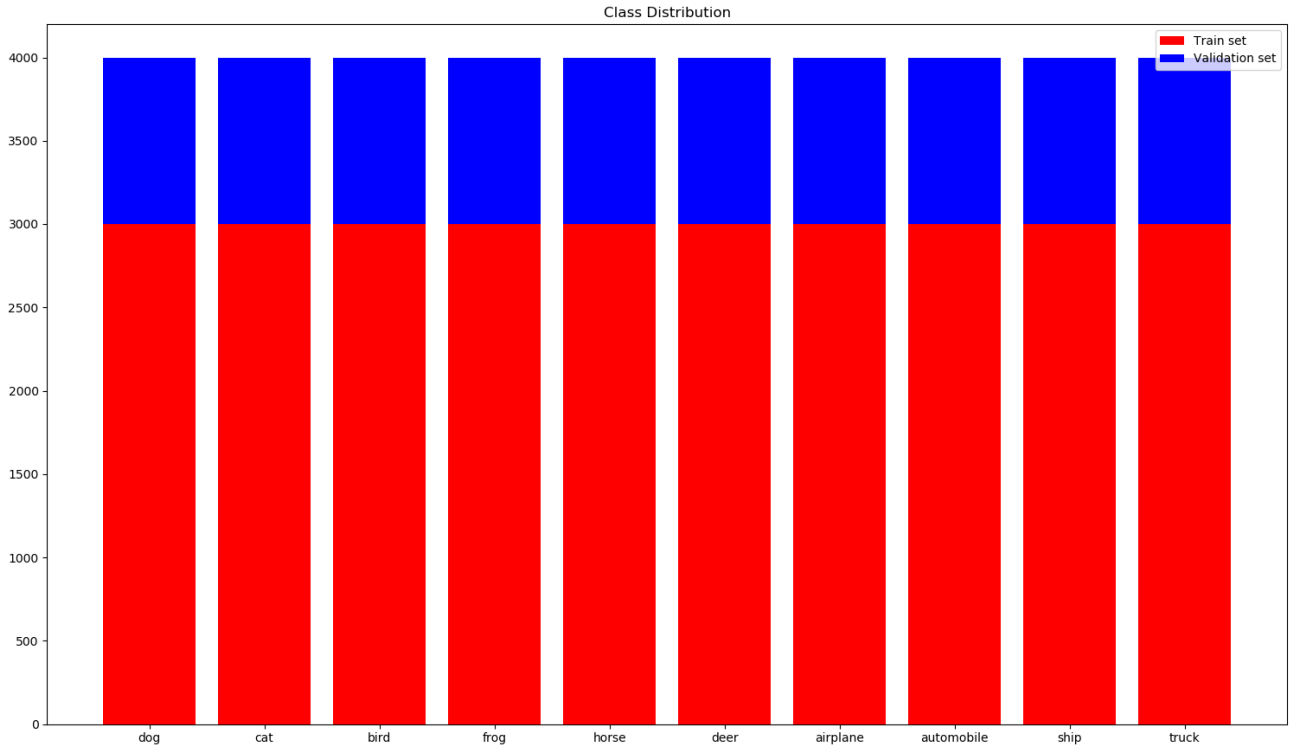

Figure 4.19: Distribution of reduced data set.

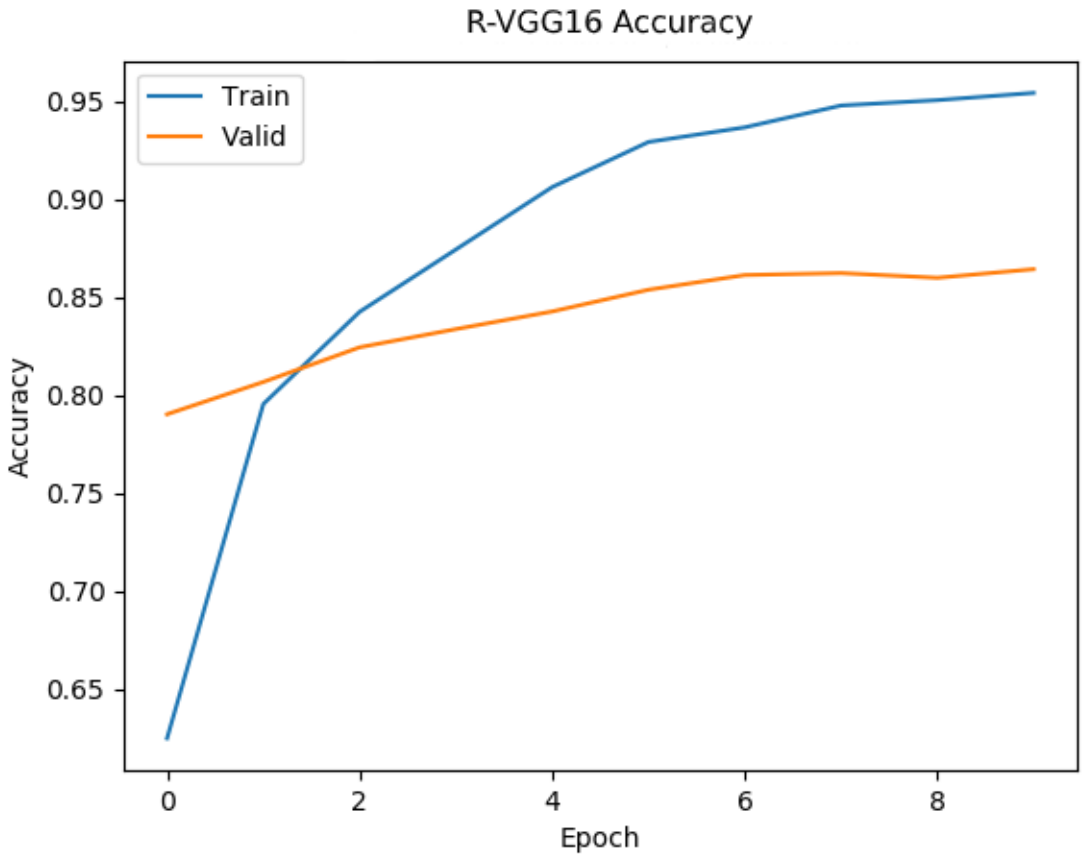

Figure 4.20: Training and testing accuracy of R-VGG16 on the reduced data set of 30,000 train and 10,000 test images.

Chapter 5

Defending Against Adversarial Examples

Adversarial examples can be easy to create, but difficult to defend against. Image filtering is a way to defend against adversarial examples in that it masks the effect of adversarial noise which manipulates the input to the convolutional neural network. If the filter also manipulates the input in such a way that the adversarial noise does not have any more effect on the correct classification on that input, then it can be considered a successful defense. However, it is important that original accuracy is not affected in the process. Some filters can remove the adversarial noise, but they might blur the image so much that it is not recognizable anymore to the viewer or the neural network. For example, one could simply make the image black, and then the adversarial effect would be lost, but so would the image. Therefore, it is important to have a balance in which the original image can be reconstructed while also removing the adversarial effect.

This section will be organized in the following way. Each of the three data sets discussed in the last section: MNIST, CIFAR-10, and CINIC-10 Large will have adversarial examples and defenses applied to them. Each data set will be attacked with PGD and FGSM, and then defended using image filters that will be elaborated on more in each subsection. Each data set will be attacked with a white and black-box attack. CINIC 10 Large will also be attacked with "gray-box" attacks. Gray-box means that the attacker has access to the internal mechanisms of the neural network, but does not have access to the filter being used.

There will also be an example of a white-box attack where the attacker does have knowledge of the filter being used, and it will be shown that this attack counters the defenses being proposed. However, this counter unacceptably distorts the image, and therefore it is not a practical attack in any setting except academic curiosity.

5.1 Defending MNIST

MNIST was the first data set experimented on. This is a unique data set, consisting of black and white images of numbers. Therefore, some of the filters used in this section of defense will not be used in other sections. For example, the thresholding filter, which converts the image to only contain binary pixel values, is practical for these images but not practical for the other two data sets.

The filters used in this section will be the same as the filters in Section 3, where the SVM is covered. The motivation behind reusing the same filters is to test if the filtering translates from the SVM attack to both adversarial attacks created on the neural network. The filters are relisted here as follows:

- *Resizing*
- *Thresholding*
- *Gaussian Blurring*
- *Median Filtering*
- *Bilateral Filtering*

5.1.1 Attacks

The white-box and black-box versions of the FGSM and PGD attacks were tested in this section. This section will cover all four attacks and compare how they influence the model's accuracy. The adversarial images will be shown as well to display how adversarial noise affects the image. All epsilons will be referred to as their true value multiplied by 255 because this was the scale used in the images after reducing them to the 0 - 1 range.

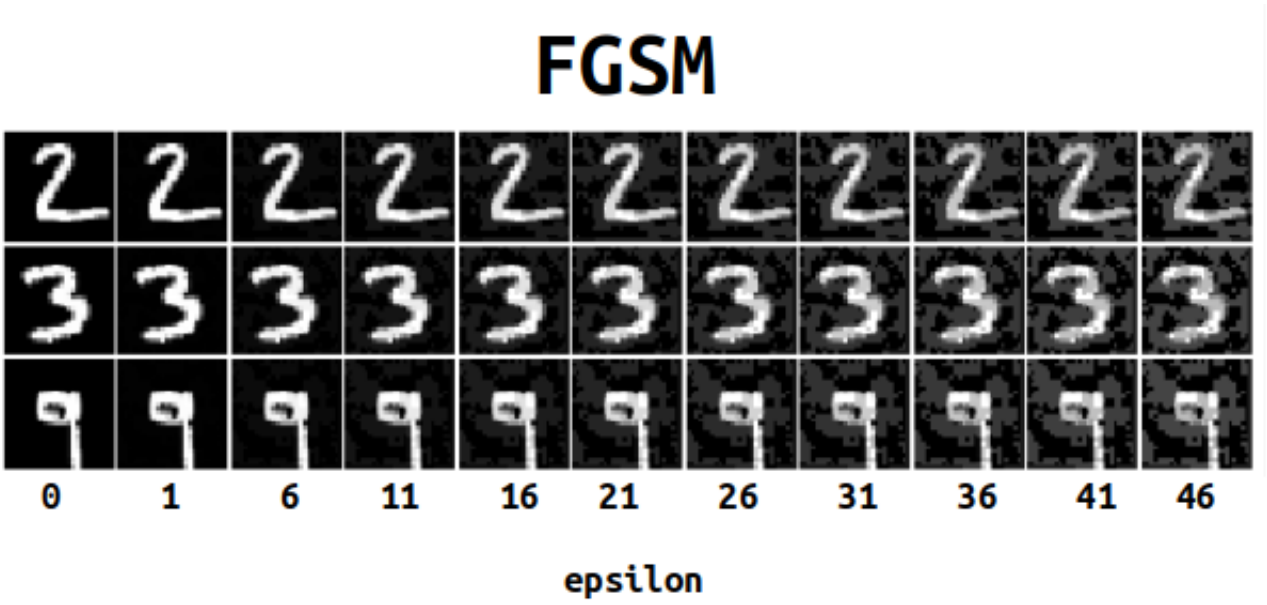

Figure 5.1: Example of FGSM applied to MNIST images with increasing epsilon.

FGSM

The FGSM attack was applied to the testing set of the MNIST data set. The value for epsilon had to be rather large in order to have an adversarial effect, on the order of 21 before any major change in accuracy. However, increasing epsilon above a certain point, in this case above 46, caused the image to become extremely distorted. Even examining Figure 5.6 shows some major distortion past 21. The figure at the end of this subsection, Figure 5.5, shows the accuracy on the unfiltered testing set for each epsilon. The base accuracy of the model, where epsilon is 0, is 99%, while the accuracy on the largest epsilon, 46, is 21%. As epsilon increases, the adversarial noise becomes more pronounced. It bears the same pattern, but becomes more white as its intensity increases. An example of an image affected by the FGSM attack is shown in Figure 5.2.

In addition to the white-box attack shown above, the same images generated from the FGSM were used in a black-box attack. The model used for the black-box attack is discussed in Section 4.2.2, where two models consisting of the same architecture were trained. The results of the black-box attack are shown in the upcoming subsection.

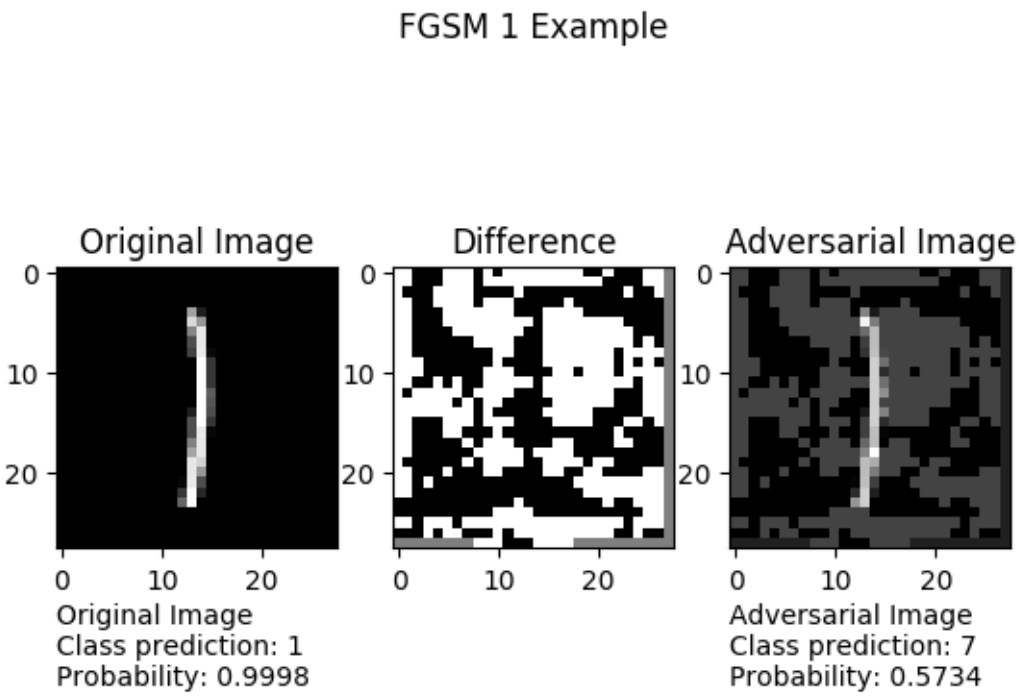

Figure 5.2: Example of FGSM applied to an image from the 1 class. Epsilon = 36

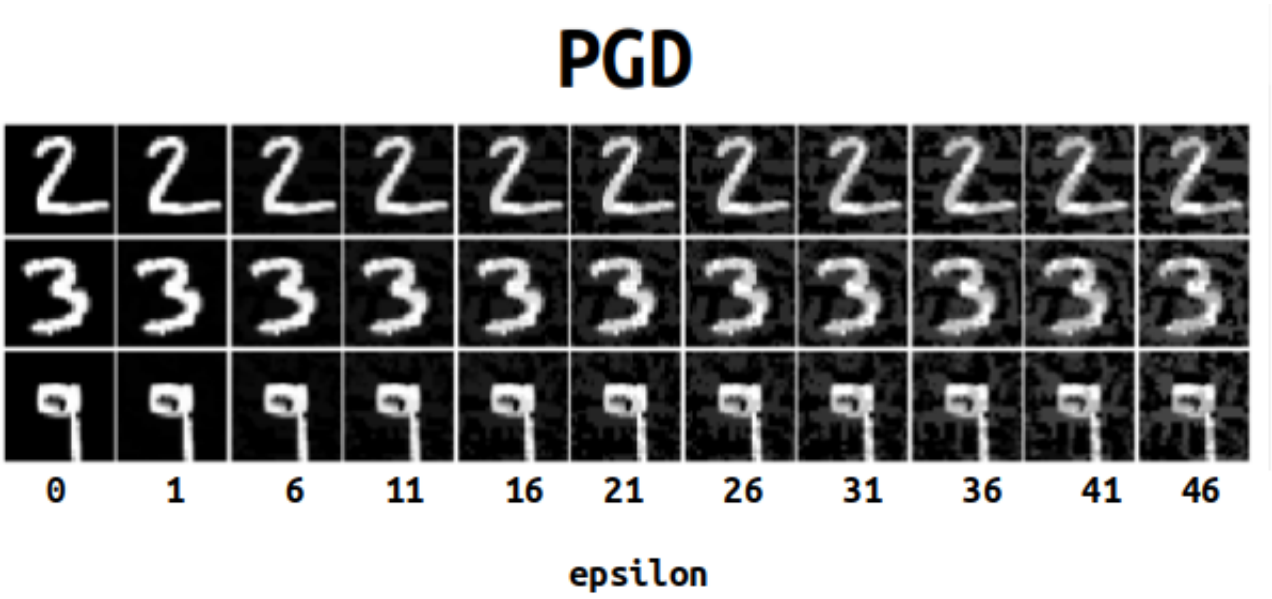

Figure 5.3: Example of PGD applied to MNIST images with increasing epsilon.

PGD

The PGD attack was also applied to the testing set of the MNIST data set. The same epsilons were used for PGD as FGSM. The results started similarly at first, but as the epsilon increased for PGD, the accuracy dropped much quicker than FGSM. This supports the claim that PGD is a stronger attack than FGSM. The step-size for PGD was set to be half of the epsilon value at each iteration of epsilon. The maximum iterations for PGD was held at 10. This allowed for PGD to compute faster while still giving the same results as higher numbers of iterations. Figure 5.3 shows the PGD attack applied to the same digits as the previous figure. The accuracy for the largest perturbation, epsilon at 46, is 0.7%, much lower than the accuracy for FGSM, which was 21%. An example of an image affected by the PGD attack is shown in Figure 5.4.

A black-box attack was created from the images generated here as well. It uses the same strategy as the black-box attack created for the FGSM.

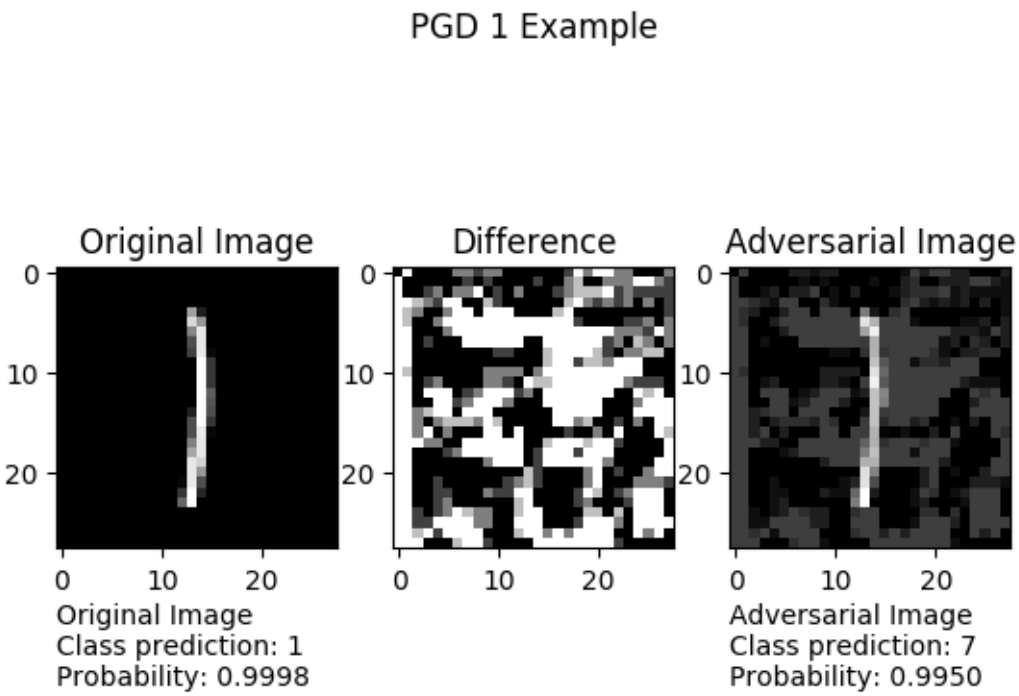

Figure 5.4: Example of PGD applied to an image from the 1 class. Epsilon = 36.

Attack Summary

This section compares all four attacks: white and black-box FGSM and PGD. The graph in Figure 5.5 compares all four attacks. The white-box PGD attack is the most damaging out of the four, which is expected. It was created to be the ultimate first-order adversary, meaning that it claims to be the strongest white-box attack in the current literature. Based on the sharp descent of the green line in the comparison graph, that claim seems to be true. In second place is the white-box version of FGSM, which is also expected. The white-box attack will always be more powerful than its equivalent black-box attack because it has access to the model's loss function and therefore can fine-tune the input to maximize the loss of that specific model. In third place is the black-box version of PGD, which is also expected because it performed much better than FGSM in the white-box setting. Finally, the weakest out of the four is the black-box FGSM, which is also expected. These results show that the white-box PGD is the strongest adversary out of the four. However, all four attacks still manage to trick the model into misclassification.

5.1.2 Defenses

This section will show the results of the defenses against the PGD and FGSM attacks. Each filter's parameters were hand-tuned based on inspection of the accuracy values of the model on the white-box adversarial testing set. This was a relatively fast process because of the size of the MNIST images. There was a window where the filter accuracy improved on the base adversarial accuracy while not too severely deteriorating the original accuracy. Unfortunately, a sacrifice in original accuracy was inevitable to create improved accuracy on adversarial images. This negative impact of filtering can be improved by training models on the filtered images, which will be discussed in Section 5.3.

The results of filtering the FGSM and PGD images are shown in Figures 5.7 and 5.8. This figure shows an image with epsilon at 0, which is equivalent to the original image, epsilon at 16, which is a medium strength attack, and epsilon at 46, which is the strongest attack tested in this section. Based on visual inspection, it is plausible that thresholding did the best job of restoring the image and that the median filter did the worst. This is subjective

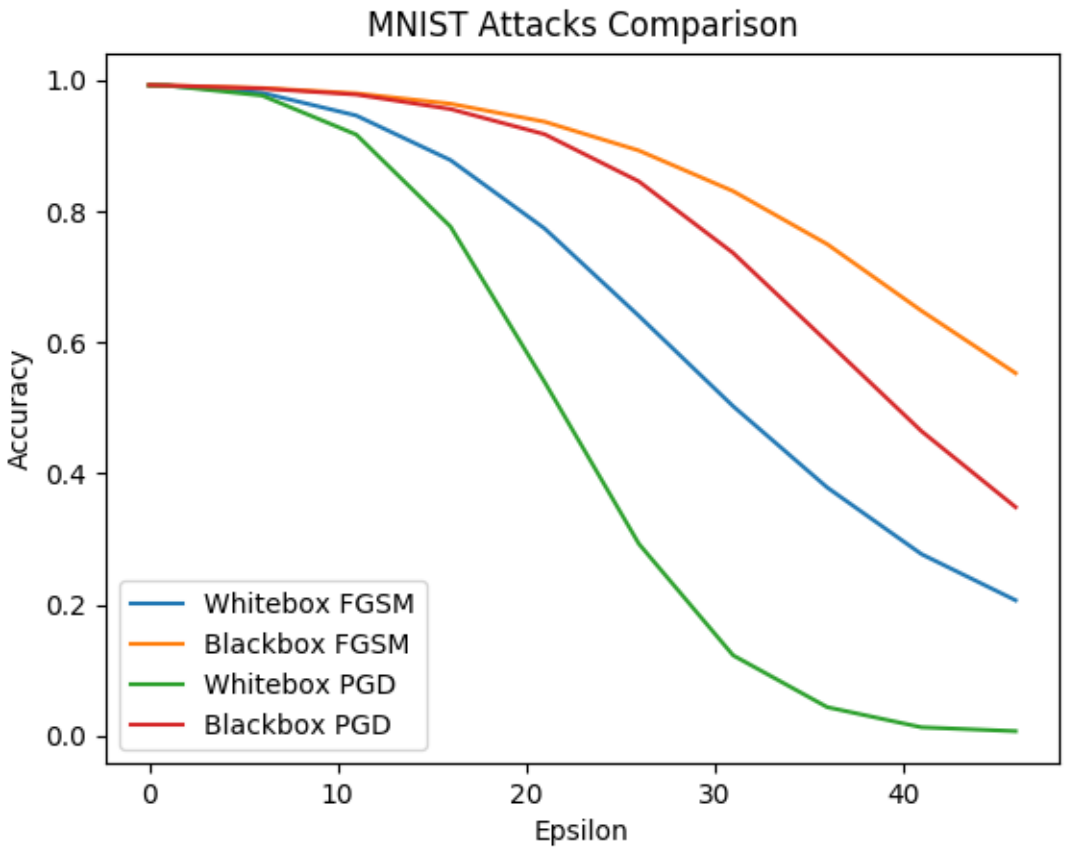

Figure 5.5: Comparison of all attacks on the MNIST data set.

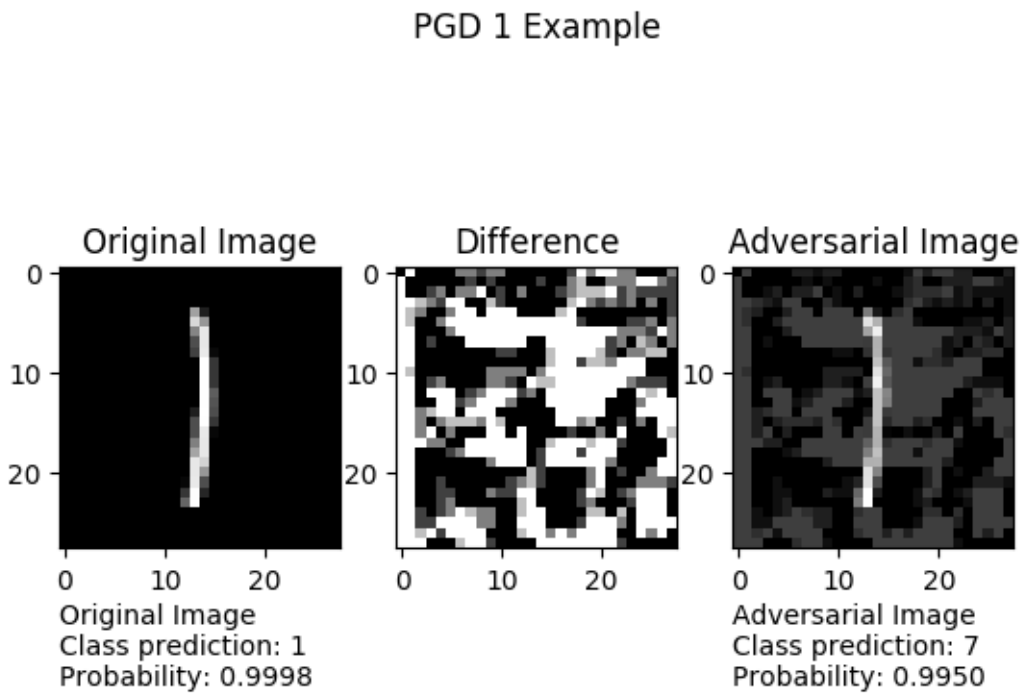

Figure 5.6: Example of PGD applied to an image from the 1 class. Epsilon = 36.

based on how much of the original structure was retained in the image while eliminating excess noise. The parameters used for each filter are listed as follows:

- *Resizing:* Down-sample to 7 by 7 using inter-area interpolation, up-sample back to 28 by 28 using inter-lanczos interpolation.
- *Thresholding:* Binarize image at pixel value of 100.
- *Gaussian Blurring:* Filter size of 7 by 7 with sigma of 10.
- *Median Filtering:* Filter size of 5 by 5.
- *Bilateral Filtering:* Filter size of 9 by 9, range and domain kernels of 100.

The next section will compare the defenses against the white and black-box attacks generated from the PGD and FGSM methods.

White-box attacks

As a reminder, white-box attacks are the most powerful because they have direct access to the model's loss function. Figure 5.5 shows the impact on accuracy that the white-box FGSM and PGD attacks have on the model. The PGD attack especially harms the model's performance. The results of the filtering defenses on both the PGD and FGSM attacks are shown in Figures 5.9 and 5.10.

The dotted blue line shows the accuracy of the adversarial attacks before any sort of filtering. The value at zero is equivalent to a non-adversarial image. Based on these graphs, it is clear that all of the filtering defenses succeeded in increasing the model's accuracy from the base adversarial accuracy. The median filter performed the worst, which follows from the earlier observations about the median filter damaging the structure of the image more than any other filter. Not surprisingly, it also had the highest accuracy on the original, non-adversarial image. This is most likely because it does not blur the image to the extreme that the other filters do. But because it does not blur, it is likely that a combination of the adversarial noise and the damage of the median filter decrease the accuracy of the classifier on the images. The best performing filter is the thresholding filter, which is rather unfair

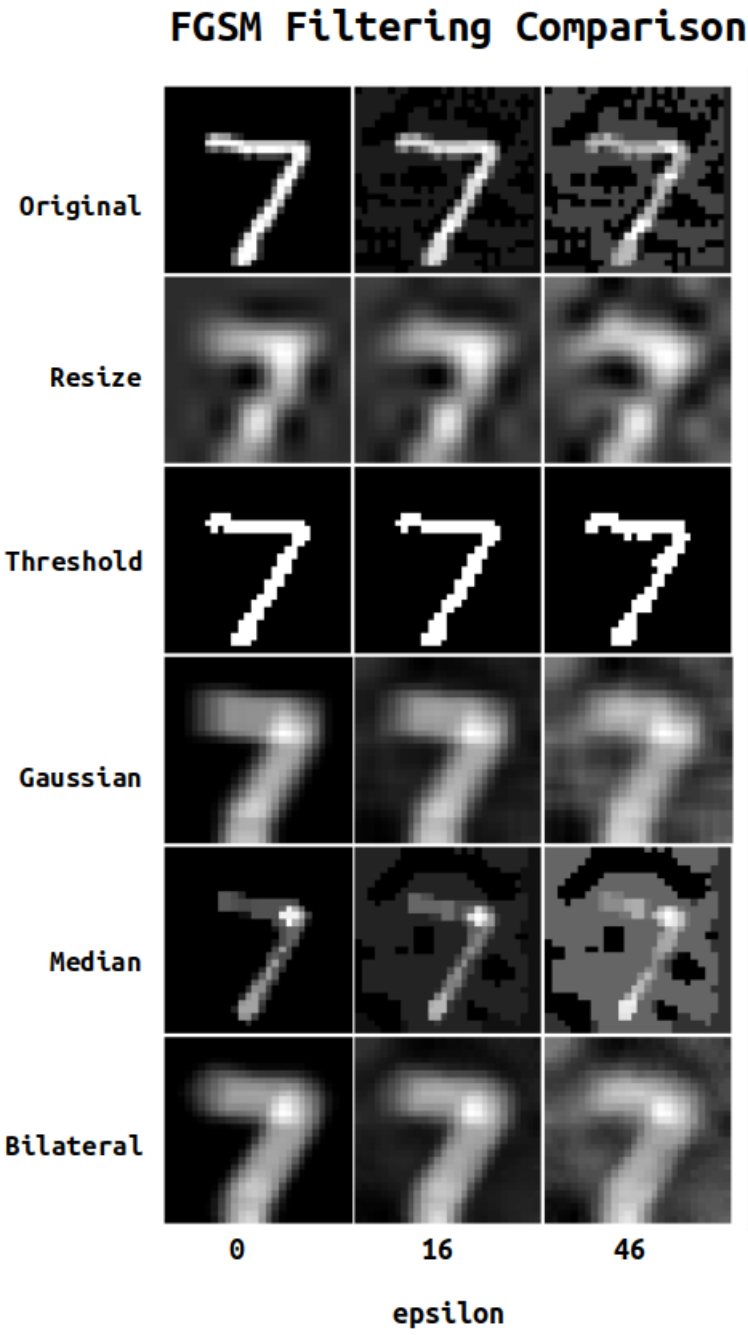

Figure 5.7: Comparison of filters applied to the original, medium epsilon, and highest strength epsilon from the FGSM attack.

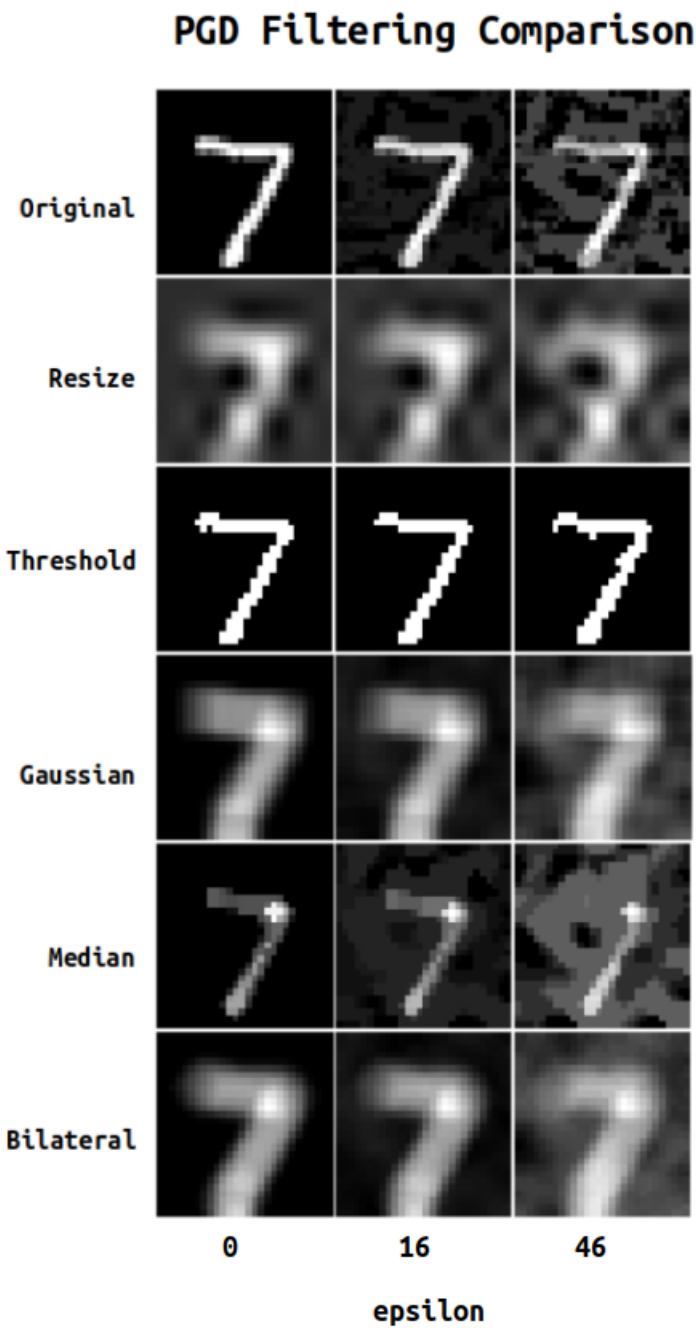

Figure 5.8: Comparison of filters applied to the original, medium epsilon, and highest strength epsilon from the PGD attack.

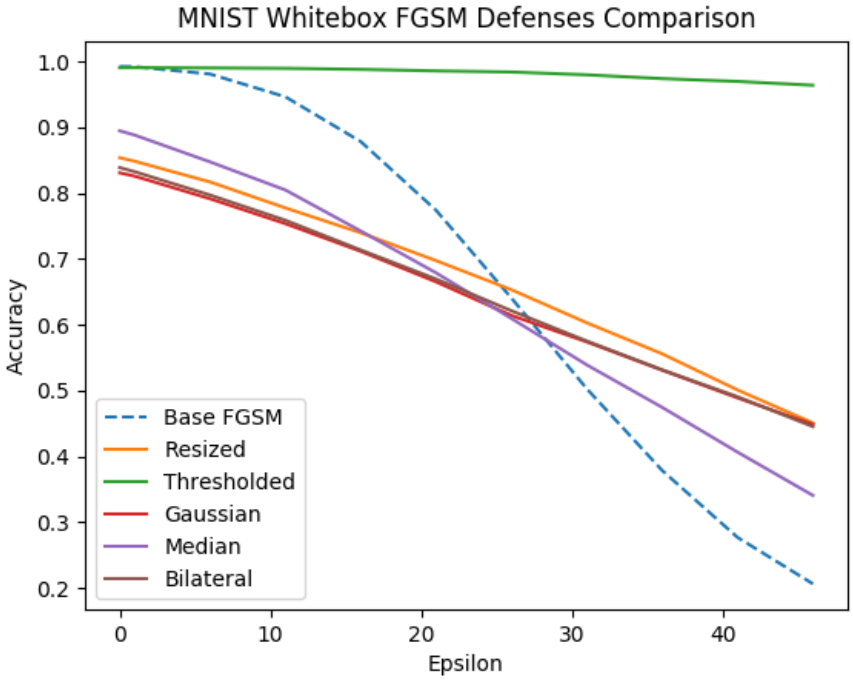

Figure 5.9: Performance of filters on increasing epsilons in the FGSM attack on the MNIST test set.

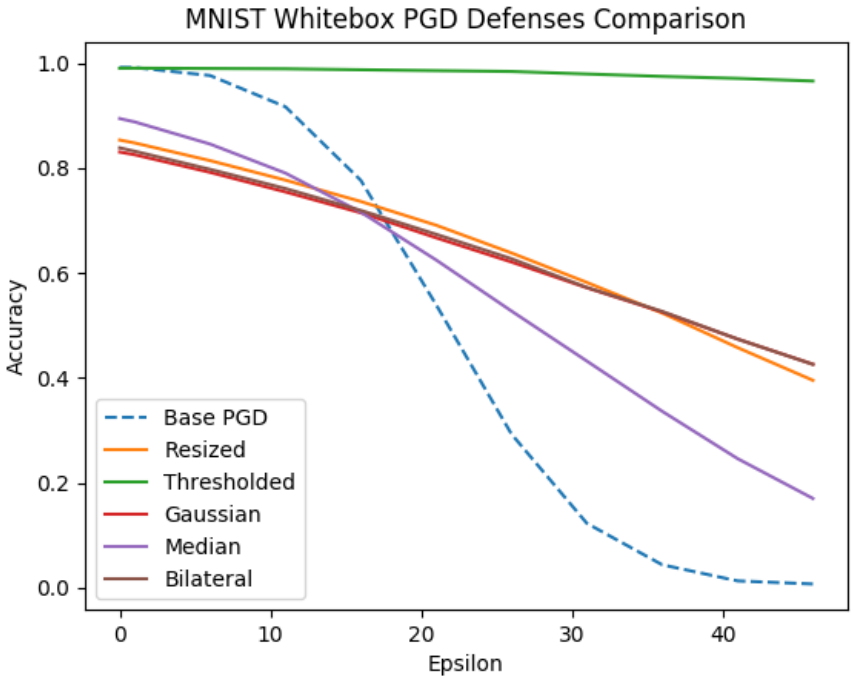

Figure 5.10: Performance of filters on increasing epsilons in the PGD attack on the MNIST test set.

because it only works on this specific type of image. If all images were just black and white like MNIST, there would be no need for any other filters. The other three filters had roughly the same performance. They decreased the original accuracy of the classifier while increasing the adversarial accuracy. This is the accuracy-defense trade-off which makes the accuracy over epsilon more horizontal at the cost of harming original accuracy. One thing to note is the severe nature of the higher epsilon images. These images have some very apparent adversarial noise and would likely not be considered good adversarial examples because of how visible the perturbations are in the image. However, the nature of this data set made it so that it required strong, visible perturbations before classifier accuracy was severely affected.

Black-box attacks

Figure 5.5 shows the accuracy of the black-box attacks. Recall that the black-box attacks performed here involve taking the adversarial images created on one model and attacking an entirely different model. The model used has the same architecture as the one used to create the black-box attacks, but because it has been trained separately, it will have different weights. The blurring defense parameters are the same as the ones used in the previous section. The reasoning for this is if the model defends against the white-box attacks, it will be more easily able to defend against black-box ones because they will be inherently weaker due to their lack of access to the model's loss function.

The results of filtering the images on the black-box model are shown in Figures 5.11 and 5.12. The very first thing to notice in these graphs is the harm that the filters do on the black-box FGSM attack. The model has higher accuracy on almost every epsilon on the adversarial images than the filtered images, which are meant to improve accuracy. This is most likely because of how much the blurring filters damage the structure of the images themselves. It is not until the highest epsilon that the blurring filters outperform the adversarial images. An important observation to note is the y-axis. The black-box version of the FGSM is a weak attack, with the strongest epsilon only reducing the model's accuracy to roughly 55%. All of the blurring defense filters here outperform the ones in the white-box of this attack. The black-box FGSM is a weak attack to begin with, so the intense blurring

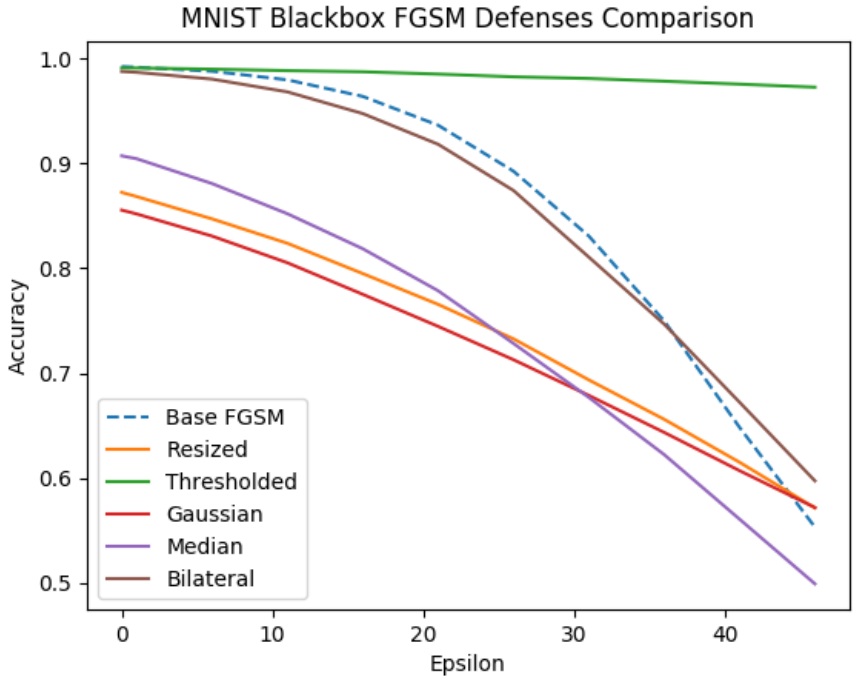

Figure 5.11: Performance of filters on increasing epsilons in the FGSM attack on the MNIST test set in a black-box setting.

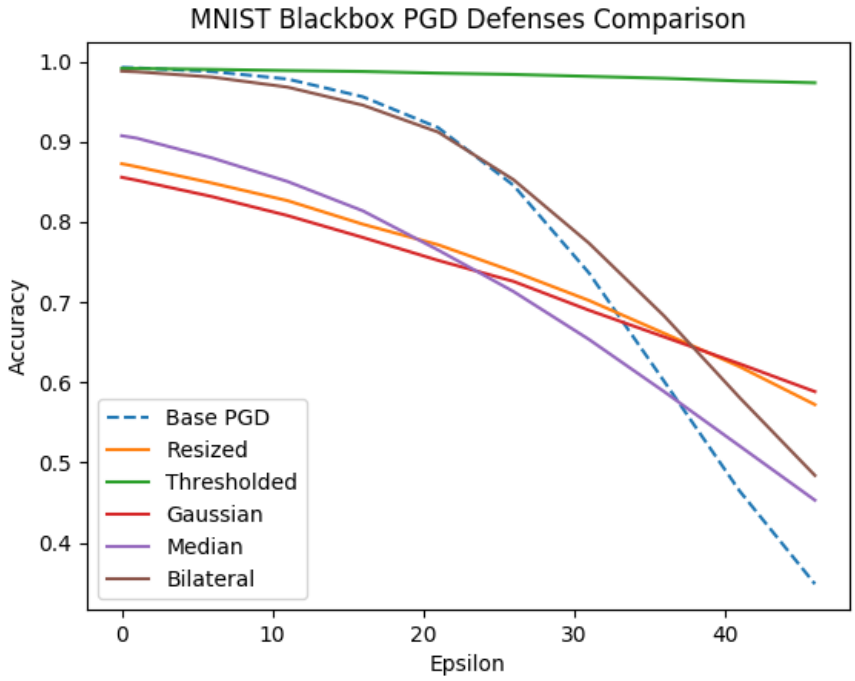

Figure 5.12: Performance of filters on increasing epsilons in the PGD attack on the MNIST test set in a black-box setting.

of these filters is too severe for this task. However, training the model on the filtered images will increase accuracy, discussed in Section 5.3.

Overall, the MNIST data set is difficult to quantify results on. This is likely because of the simple nature of the images. It takes significant distortion to the image before any adversarial effect is seen. Either the image will be clean and not affect accuracy, or distorted and affect accuracy significantly. It is only natural to move away from MNIST and begin experimenting with more complex images, like those found in the CIFAR-10 data set. The next section will discuss the PGD and FGSM attacks as used on the CIFAR-10 data set. Some of the same and some different filters will be used in the next section.

5.2 Defending CIFAR-10

This section will cover the white and black-box FGSM and PGD attacks as applied to the CIFAR-10 data set. As a reminder this data set is more complex than MNIST in that it contains images of size 32 by 32 with more detailed scenes. It has ten classes consisting of images of animals and vehicles. This data set is somewhat of an intermediate data set between MNIST and CINIC-10 Large in that it contains more detailed images than MNIST but contains images that are significantly smaller than CINIC-10 Large's 224 by 224 images.

All of the same filters used in MNIST will be used here, with the exception of the thresholding filter. They are listed below:

- *Resizing*
- *Gaussian Blurring*
- *Median Filtering*
- *Bilateral Filtering*

5.2.1 Attacks

Just like the last section, this section will examine the results of attacking CIFAR-10 with the white and black-box versions of the PGD and FGSM attacks. As a reminder, the images

here were augmented with pixel-mean subtraction when they were used to train the ResNet. This caused most of the images to become darker. Some of the results shown in this section will therefore look darker than the original images. The darker images will be shown with some lightening applied to them for viewing purposes only. The amount of lightening will be indicated in the description of the images. All epsilons will be referred to as their true value multiplied by 255 because this was the scale used in the images after reducing them to the 0 - 1 range.

FGSM

Unlike MNIST, the FGSM attack did not require such a large epsilon to obtain an adversarial effect on the classifier. An epsilon value of 1 was sufficient to decrease accuracy. This is likely because these images have higher dimensionality than the MNIST images. Epsilon values of 1 through 9 were tested on CIFAR-10. The image in Figure 5.13 shows examples of three images from the data set as increasing epsilons are applied to them, with epsilon at zero being the original image. Although somewhat difficult to see, the image becomes more distorted as epsilon increases. The easiest place to see distortion is in the sky behind the ship in the first row. Two more places to look are at the fur of the dog and the grass the horse is standing on. As epsilon increases, it is clear that these spots begin to get more noisy. Figure 5.14 shows an example of an original image and its incorrectly classified counterpart.

Black-box attacks were created in this section as well. The black-box attacks used here are also transfer attacks where the adversarial images were created on one network and then used to attack another network. To make it interesting, black-box attacks will be tested on two different networks: the same network used here but retrained, and another network entirely. The new network will not have the same architecture as the one that generated the adversarial images. It will instead be one of the other networks discussed in Section 4.2.2. More specifically, the second attempted network architecture. This network was not trained on images with mean pixel subtraction, so it needed to be retrained on those. The network achieved 82% accuracy on the testing set after training with the mean pixel subtracted images for 100 epochs.

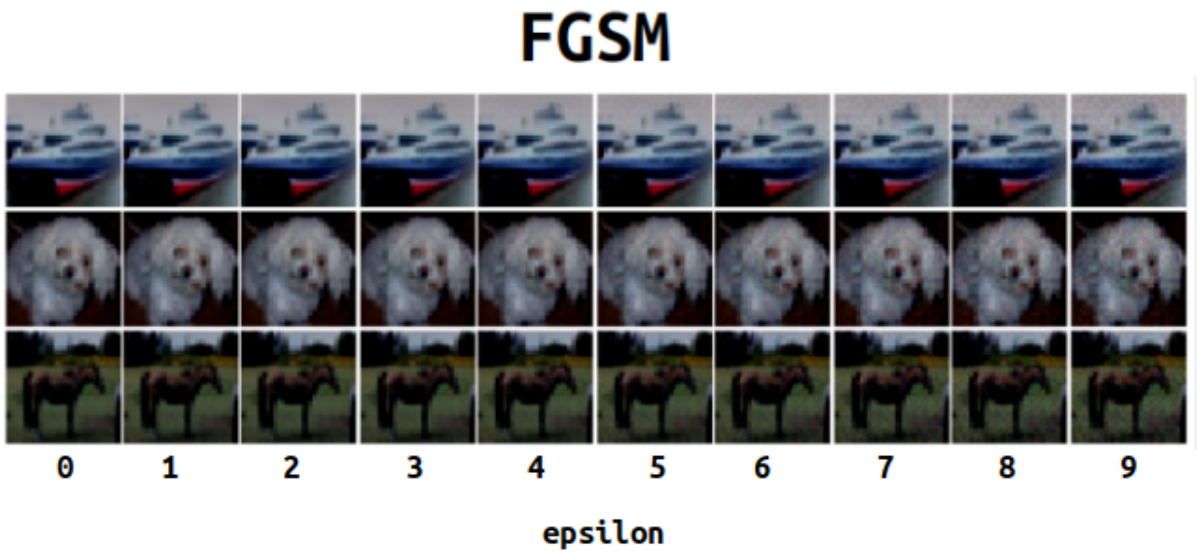

Figure 5.13: Example of FGSM with increasing epsilon applied to three random CIFAR-10 images. Lightening added: 0.19.

PGD

The PGD attack here was stronger as well. It did not require as high of an epsilon to start decreasing image accuracy. PGD severely reduced the classifier accuracy on the CIFAR-10 testing set, with an epsilon of just 1 driving accuracy down from 91% to 20%, and an epsilon of 2 decreasing it even more to 6.1%. The same three images as Figure 5.15 are shown. It might be somewhat difficult to differentiate the noise in the PGD images from the noise in the FGSM images because they are so small and complex. The most notable difference is that PGD is less noisy than FGSM. Looking at the same three spots in the images as before, it is apparent that there is less noise in the PGD images. Another example of a single image and its misclassification from PGD is shown in 5.16. Once again, it might be difficult to tell the difference between this image and the FGSM one, but upon closer inspection it is clear that this image is less noisy. The difference image here and in the FGSM are distinct in that the FGSM difference has significantly more noise. This is most likely due to PGD's iterative process, which looks to maximize loss iteratively, unlike FGSM's "one-shot" method.

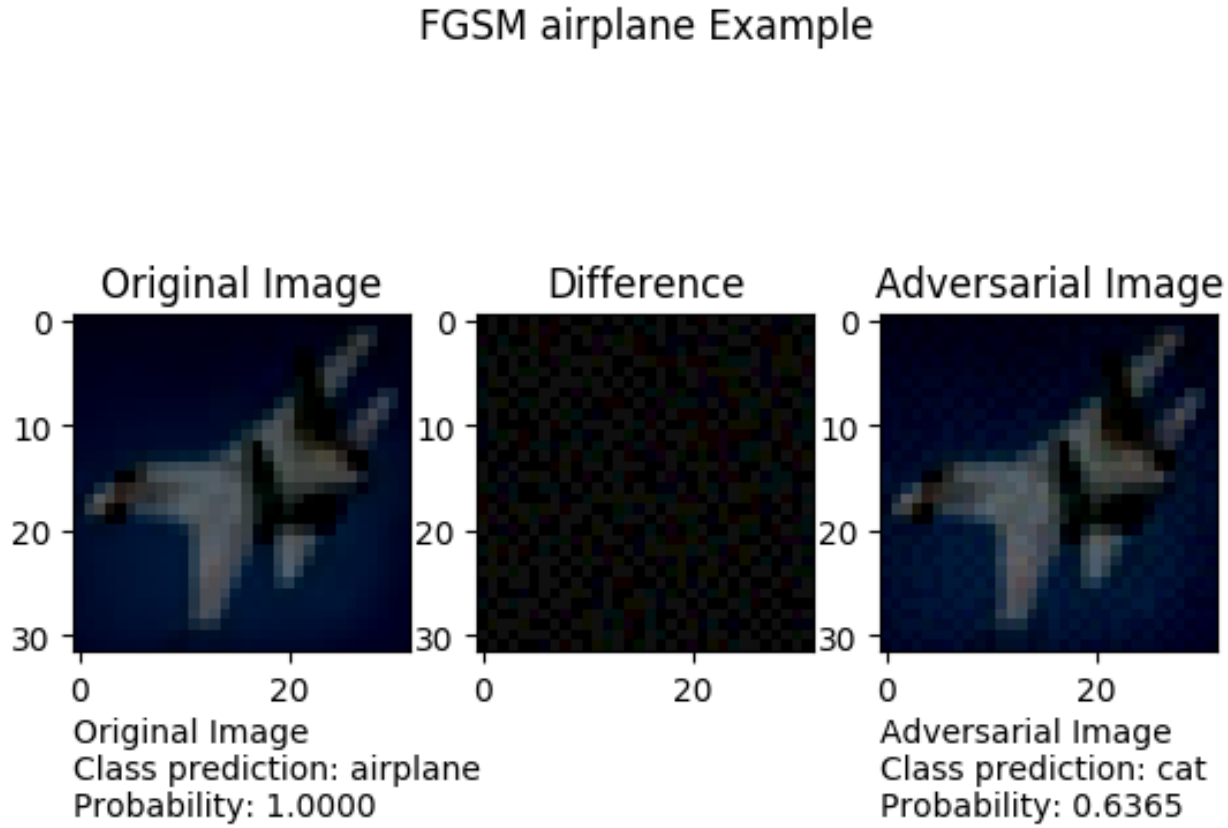

Figure 5.14: Example image from the airplane class misclassified as a cat as a result of the FGSM attack. Epsilon = 5. No image lightening applied.

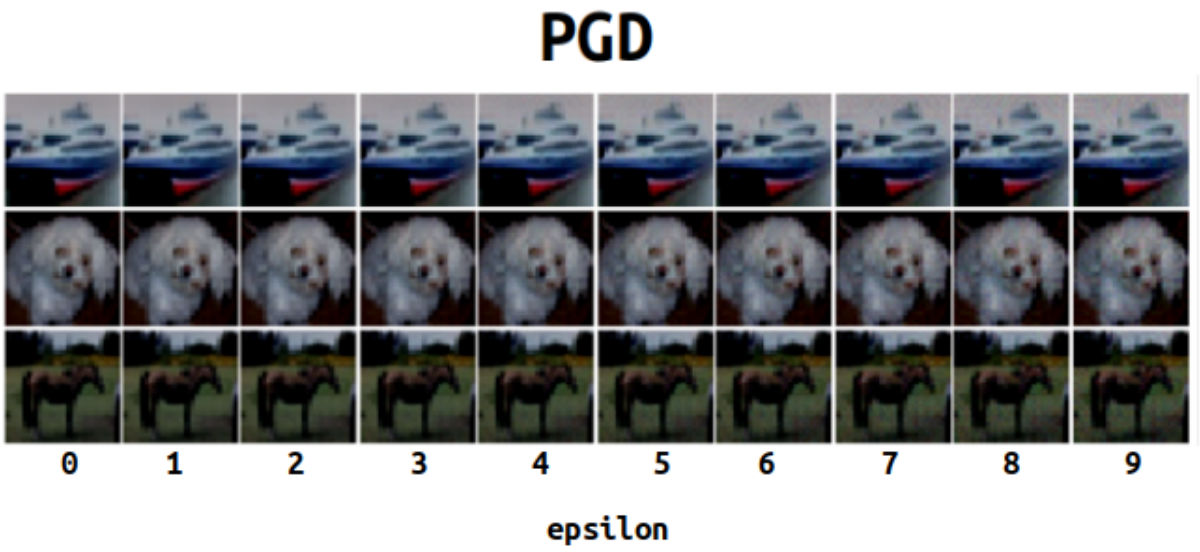

Figure 5.15: Example of PGD with increasing epsilon applied to three random CIFAR-10 images. Lightening added: 0.19.

To stay consistent with the FGSM attacks, the same methodology for black-box attacks via PGD will be used here. Two different architectures, just to make it interesting.

Attack Summary

Overall, the PGD greatly outperformed the FGSM attack in reducing classifier accuracy. It is significantly more powerful and proves its claim as the "universal first-order adversary". The graph in Figure 5.17 shows how much more powerful the PGD attack is than the FGSM attack. The white-box PGD attack brought accuracy down to a very low 5% in the last few epsilons. It also outperformed FGSM in the black-box setting on the ResNet network. Reminder, the ResNet network has the same network structure as the one used to create the adversarial images, but was re-trained so that it would produce different weights. The most curious result is the out-performance of PGD by FGSM on the new network. The results would suggest that FGSM is a weaker attack than PGD in every scenario, but it decreased classifier accuracy more than PGD on the network with different structure.

The attacks became weaker as the model moved away from the original model used to create the adversarial examples. The black-box attacks on the retrained ResNet are stronger

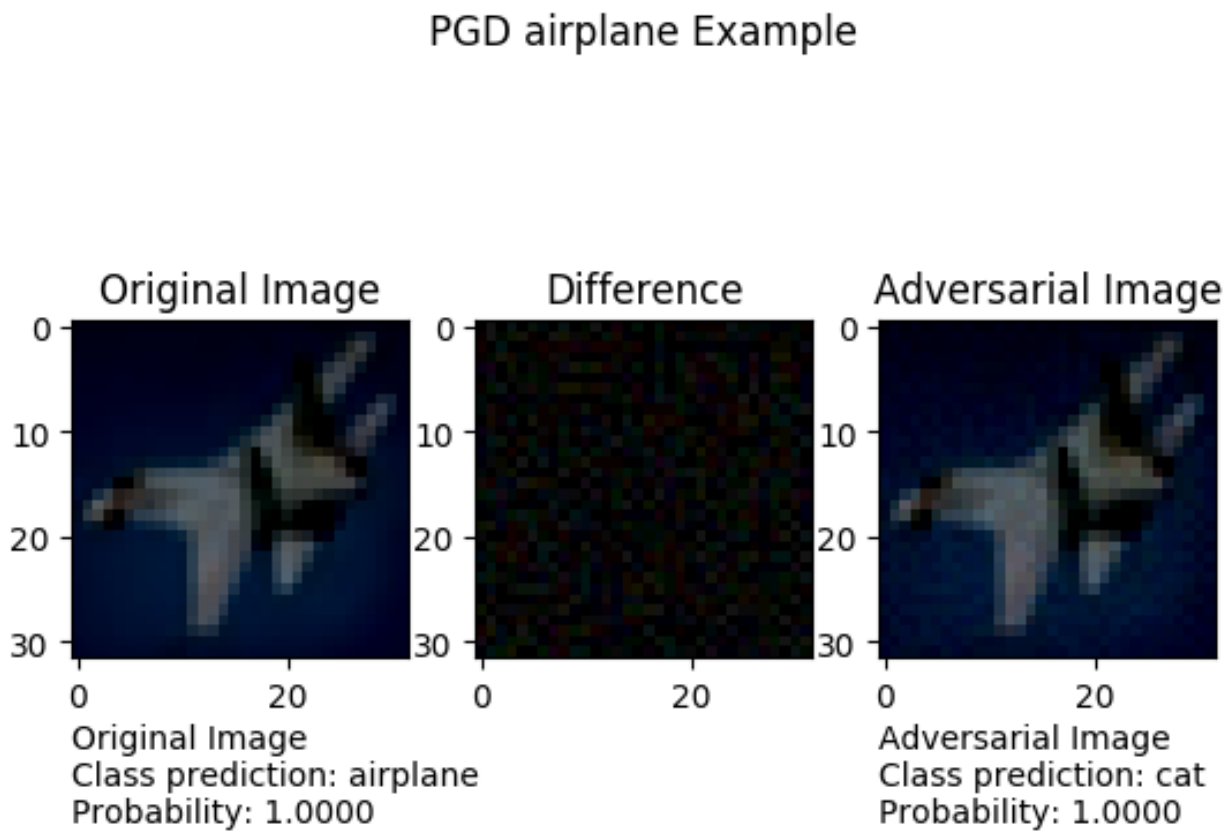

Figure 5.16: Example image from the airplane class misclassified as a cat as a result of the PGD attack. Epsilon = 5. No image lightening applied.

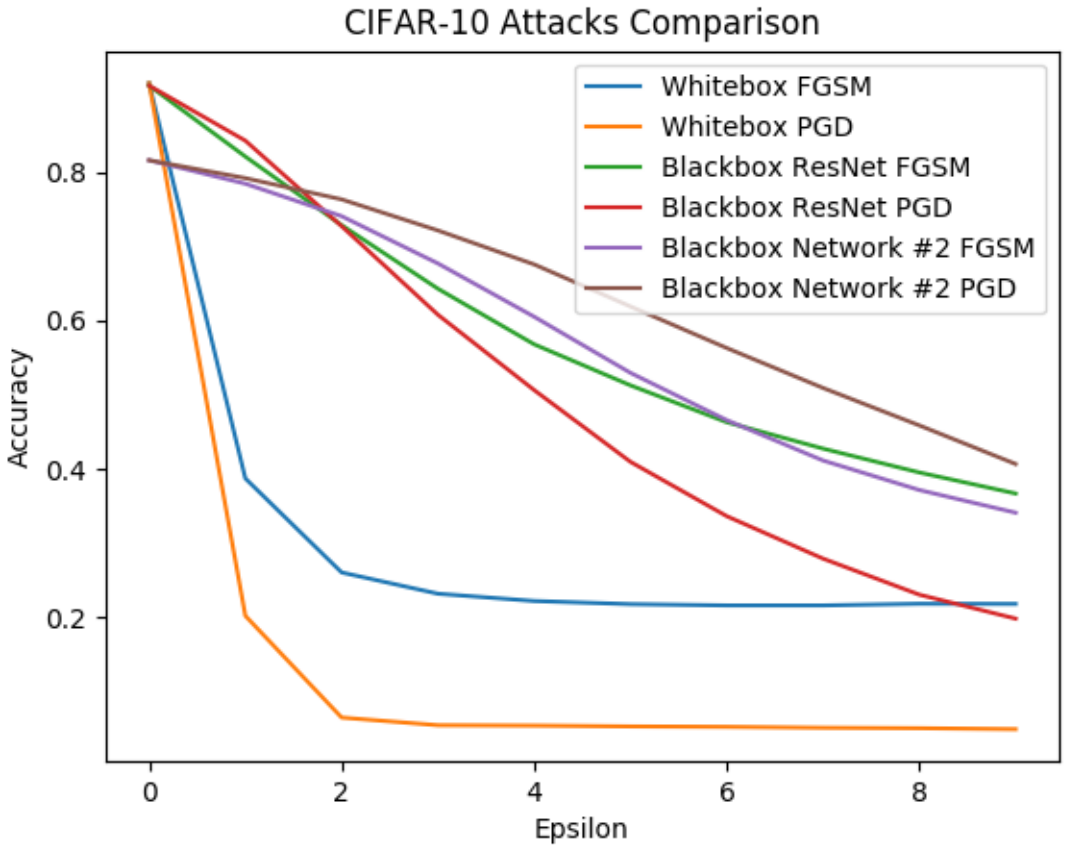

Figure 5.17: Comparison of white and black box attacks produced from the FGSM and PGD attacks on the CIFAR-10 testing set.

than the ones on the differently structured model. This likely goes with Goodfellow et al.'s observation that adversarial examples generalize because the models learn similar weights during training. The model with different structure likely had more different weights than the ResNet, and therefore the adversarial examples were not as powerful.

The next section will discuss the defenses created against these adversarial images. It will only show the defenses against the white and black-box ResNets because of the different model's low initial accuracy and how much better the second ResNet black-box attacks performed.

5.2.2 Defenses

This section will show the results of filtering the images produced by the FGSM and PGD attacks. Once again, the parameters were hand-tuned based on inspection of the accuracy values over the ten epsilons. Each filter's parameters were considered "good" if they successfully brought the adversarial accuracy rate up while not overly diminishing the original accuracy. There will inevitably be some image distortion to remove the adversarial effect, so some initial sacrifice in accuracy is expected. It is worth restating that this dip in accuracy can be solved with training on the filtered images, which will be discussed in Section 5.3. The parameters were hand-tuned by inspecting the accuracy values on the white-box PGD attack only because it reduced classification accuracy the most. The motivation for not tuning to each attack is because the network will not know which adversarial method it is being attacked with. Therefore, it makes sense to only tune on the strongest attack and make the defenses robust to "any" attack they encounter.

The final parameters used in defending the network are listed below.

- *Resizing:* Down-sample to 17 by 17 using inter-area interpolation, up-sample back to 32 by 32 using inter-lanczos interpolation.
- *Gaussian Blurring:* Filter size of 5 by 5 with sigma of 1.
- *Median Filtering:* Filter size of 5 by 5.
- *Bilateral Filtering:* Filter size of 5 by 5, range and domain kernels of 100.

Examples of an image at epsilon = 0, 5, and 10 after applying the above filtering to them are shown in Figures 5.18 and 5.19. This is what the images will look like when they are filtered in both the black and white-box attacks, discussed in the next subsections. It is difficult to tell what class this image belongs to but inspecting the labels reveals that this image belongs to the cat class. Similar to MNIST, it is clear that the best defense must strongly blur the image in order to mask the adversarial effect. The median filter obscures the image the most, to the point where it looks like a blob of colors. The other filters also remove a significant amount of detail, but there was not much detail to begin with in this small image.

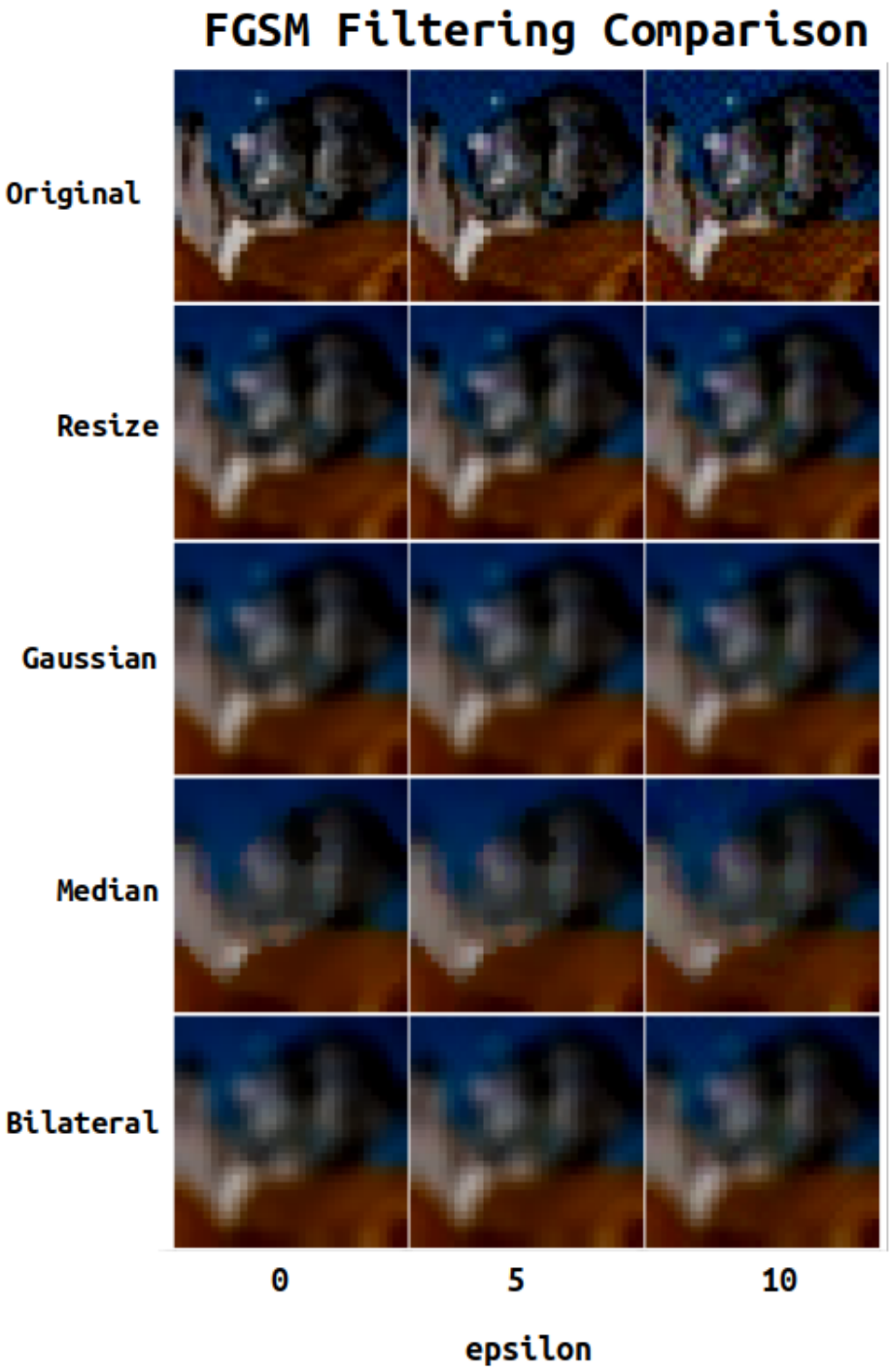

Figure 5.18: Comparison of filters applied to the original, medium epsilon, and highest strength epsilon from the FGSM attack on a random image from the CIFAR-10 testing set.

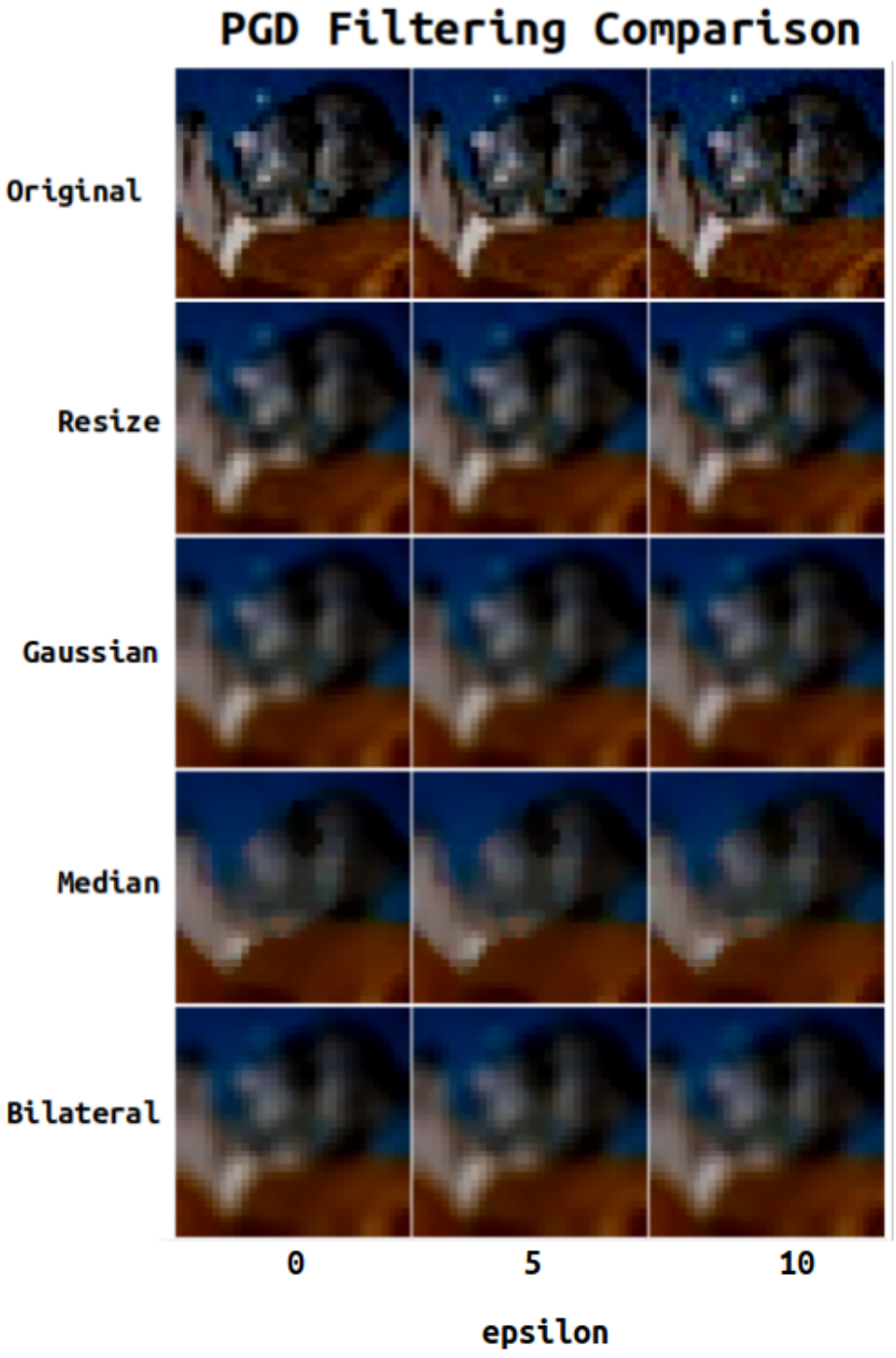

Figure 5.19: Comparison of filters applied to the original, medium epsilon, and highest strength epsilon from the PGD attack on a random image from the CIFAR-10 testing set.

White-box attacks

The white-box attack is more powerful than the black-box attack because the algorithm has access to the model's loss function. The comparison of filtering defenses against the white-box attacks from the FGSM and PGD attacks are shown in Figures 5.20 and 5.21. It appears that the various defenses against both attacks increased accuracy to roughly the same amount. The gap between the attack accuracy and the defense accuracy is wider on the PGD graph. The initial accuracy decrease and final accuracies for all filters on both graphs start and end at roughly the same point. This is likely because of the amount of detail lost with the level of filtering applied, so both attacks affect the images similarly. The FGSM attack only gains a small percentage more accuracy, which is difficult to see on the graph. The resizing method defends the best until the epsilon at 5 mark where the Gaussian filter surpasses it. The bilateral filter performed the worst. The median filter does a good job at keeping the accuracy in a mostly straight line, which is indicative of it blurring the image to the point that no matter how much noise is added, the post-blurred image will always look the same. Indeed, looking at the examples of the filtered images in the last subsection, one can see how much more this filter blurs than any of the others. This might be a valid defense strategy if used with a model trained on the filtered images such that initial accuracy is not affected as drastically.

Black-box attacks

The comparison of filtering defenses against the black-box attacks from the FGSM and PGD attacks are shown in Figures 5.22 and 5.23. As expected, the PGD attack manages to drive down accuracy more than the FGSM attack with final accuracies of roughly 0.15 and 0.35, respectively. Once again, the filters defend similarly, with only very minor differences in the actual percentages as epsilon increases. The initial accuracy takes a large hit. The filters do not increase performance until roughly the epsilon at 4 mark. At this point, the loss to accuracy is more severe from the adversarial noise than it is from the blurring. All of the filters exhibit similar levels of defense as they did in the white-box portion, with the exception of the resizing method, which outperformed all other filters on every single epsilon.

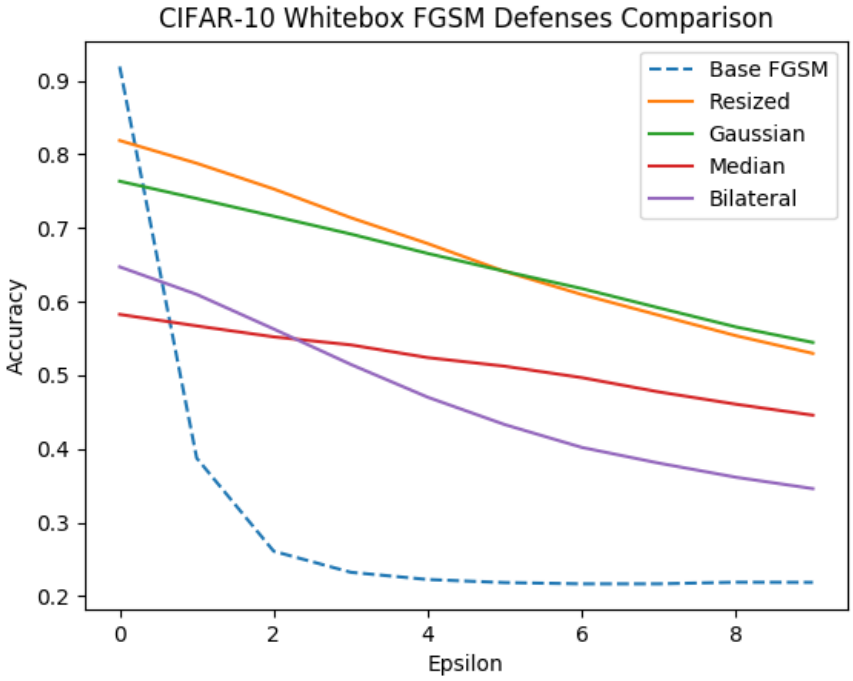

Figure 5.20: Performance of filters on increasing epsilons in the FGSM attack on the CIFAR-10 test set.

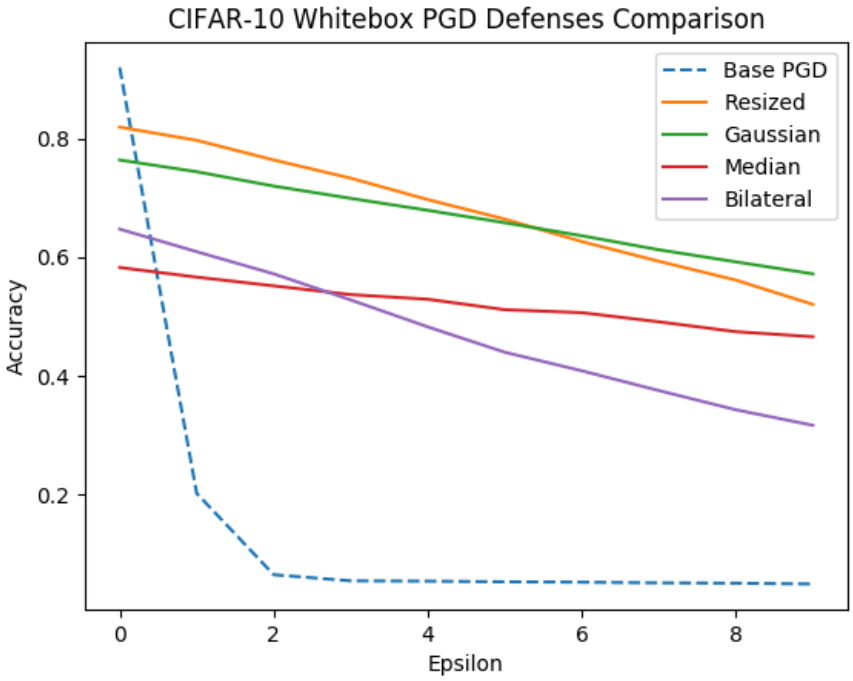

Figure 5.21: Performance of filters on increasing epsilons in the PGD attack on the CIFAR-10 test set.

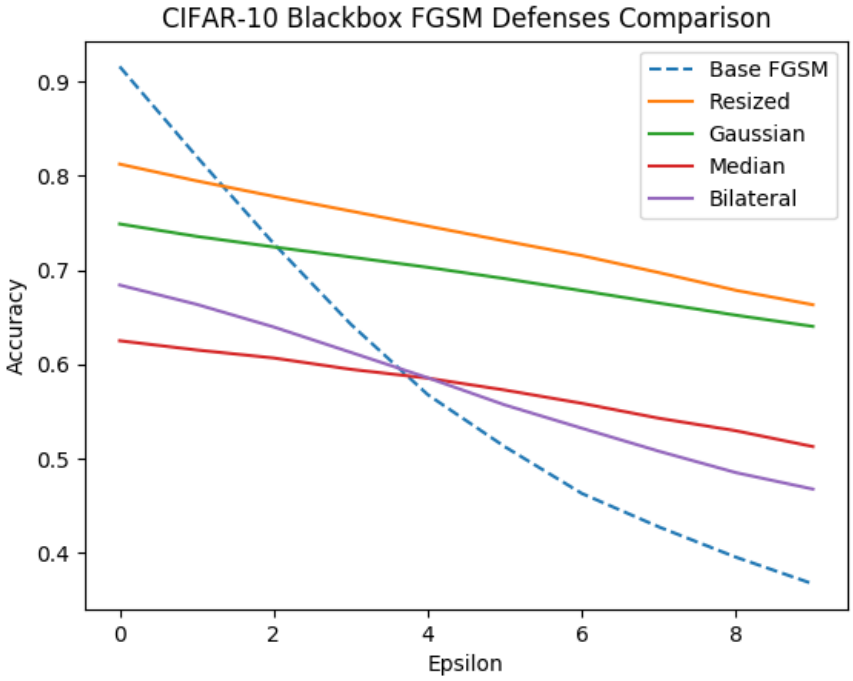

Figure 5.22: Performance of filters on increasing epsilons in the FGSM attack on the CIFAR-10 test set in a black-box setting.

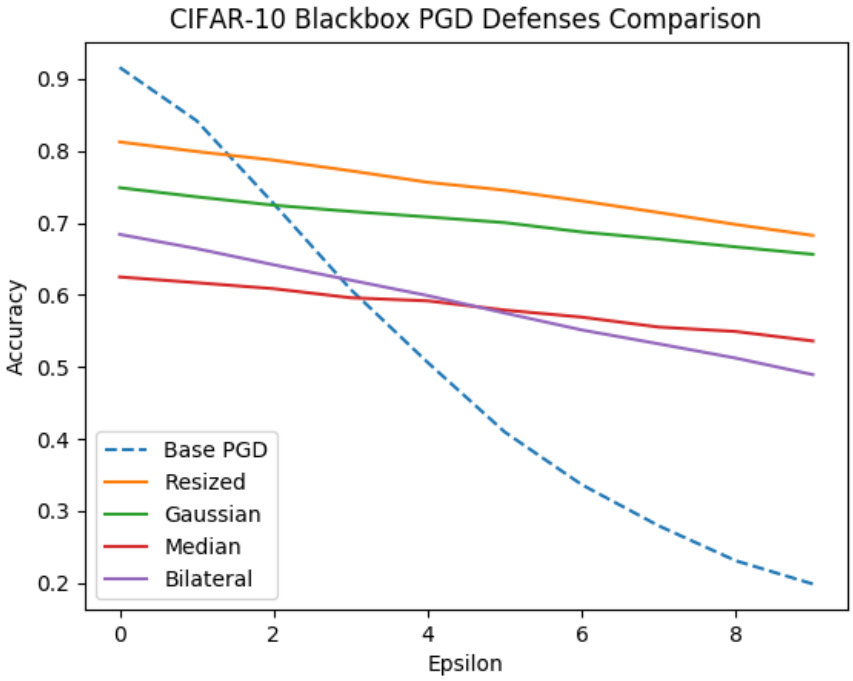

Figure 5.23: Performance of filters on increasing epsilons in the FGSM attack on the CIFAR-10 test set in a black-box setting.

Perhaps if more epsilons were to be tested, the resizing and Gaussian filtering methods would intersect. The black-box attacks are much weaker than the white-box attacks, which is to be expected because there is no direct access to the model's loss function.

Overall, the CIFAR-10 data set was a good intermediate data set to run experiments on. However, the challenge of having such small images was a barrier to effectively testing filtering defenses. There is not much detail in the images, so even a small amount of filtering causes a significant amount of detail loss. The possible range of experimentation was limited in this way. The next step was to perform testing on larger images and examine the results of applying filters to more realistically sized and complex images. Even examining Figures 5.18 and 5.19 shows how difficult it can be for even a human to recognize what some of the images are. Larger images with more detail provide a better basis for examining the results of filtering after adversarial noise has been applied. However, images that are too large would be too computationally complex to perform all of these operations on, so an intermediate size of 224 by 224 was used.

5.3 Defending CINIC-10 Large

The last data set, and the most experimented on, is the CINIC-10 Large data set, which needed to be reduced to 30,000 training images and 10,000 test images for experimentation purposes. The details of the images and network used in this section were covered in Section 4.2.3. Once again, this data set was chosen for its larger image sizes after finding that the small images were limiting. This section will cover several new filters, covered in the background, and gray-box defenses. As a reminder, gray-box involves training the network on the filtered images, and then attacking it with a clean image that has been adversarially altered using the gradients from the same network. This section will also include a more sophisticated method of optimizing the hyper-parameters in the filters and a way to beat the filtering defenses proposed.

This section will cover the following defenses, which were optimized using a parameter search on the white-box PGD images:

- *Total Variation Denoising - ROF*

- *Total Variation Denoising - Chambolle's Implementation*
- *Non-local Means*
- *Anisotropic Diffusion*
- *Gaussian + Median Filter*

Some other filters, like the ones used in the previous sections, were tested on the attacks on this data set. It was found that these filters did not perform very well, so they were discarded at this point in the experiment. Some other filters that were attempted include: resizing, bilateral filtering, minimum filtering, maximum filtering, and mean filtering.

5.3.1 Attacks

PGD and FGSM will be tested here as usual. In addition to the usual white and black-box attacks, there will be gray-box attacks, the results of which will be shown in the defense section.

The gray-box attacks will involve using the gradients created from a network trained on filtered images to create the FGSM and PGD attacks. These gradients will be applied to a clean image which is subsequently filtered with the respective filter corresponding to that network. This is not a white-box attack because it does not involve using the filtered image and applying adversarial noise to it. Therefore, the attack knows everything about the network it is attacking except for what filter of images it was trained on. As will be seen at the end of this section, knowing the filter the model was trained on can be used as part of an adversarial attack.

FGSM

The FGSM attack was applied on the images with epsilons 1 through 9. This is different than the epsilons in the previous sections because the images here were kept in the full 0-255 range. There was not a difference one way or the other with the image scaling. The results of the FGSM attack applied to the images is shown in Figure 5.24. The viewer will likely have to zoom in to see the noise in these images. The best place to see the change are in

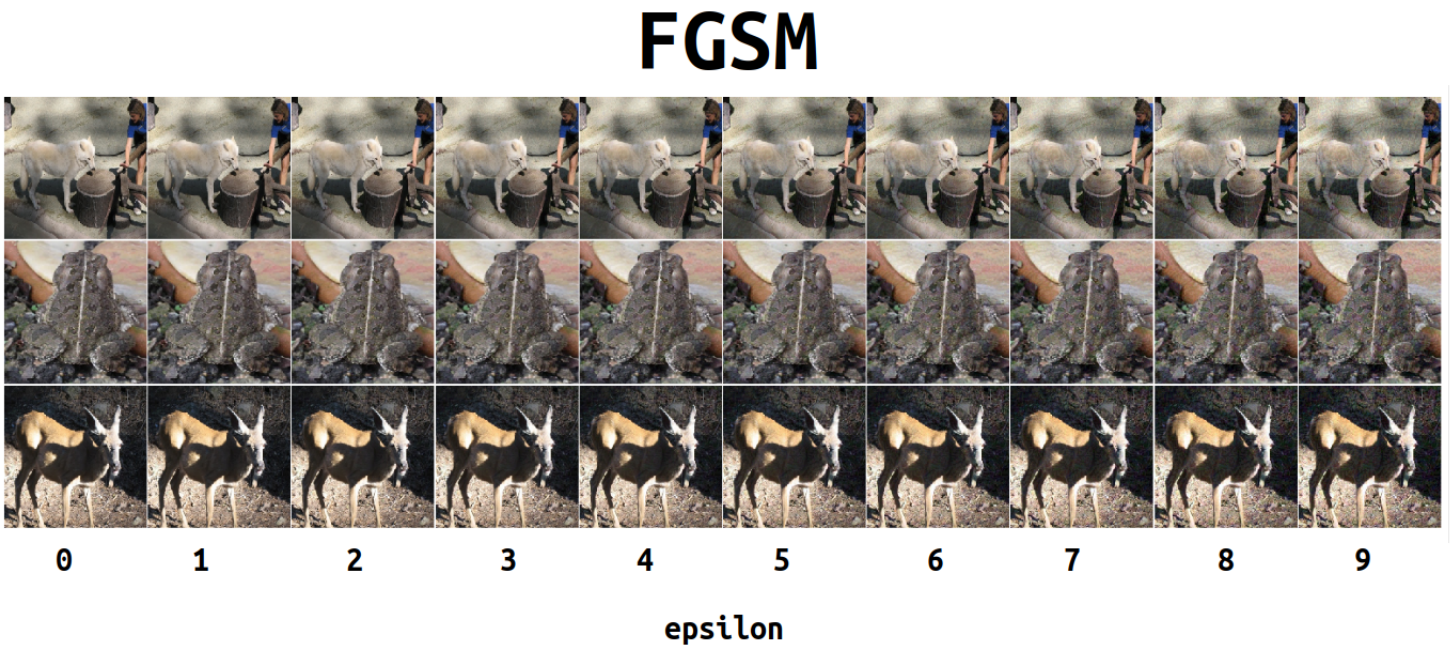

Figure 5.24: Example of FGSM with increasing epsilon applied to three random CINIC-10 Large images.

uniform areas, such as the speckles in the background of the dog image, the skin on the toad, and the background of the deer. An example of the FGSM attack applied to random image is shown in Figure 5.25.

The original VGG16 network that did not produce any gradients will be targeted for the black-box attack. The structure of R-VGG16 is based on the structure of VGG16, including using the same initial weights. The only difference are some Leaky ReLus and Dropout layers. Therefore, the adversarial examples should generalize to this almost identical network. As a reminder, this network was trained on the full data set and achieved a 91% accuracy on the test set.

PGD

The PGD attack was also applied with epsilons 1 through 9. The maximum iterations for PGD was set to two, as this attack is very expensive and allowing it to run for hundreds of iterations on the larger images would have taken a very long time. An example of images

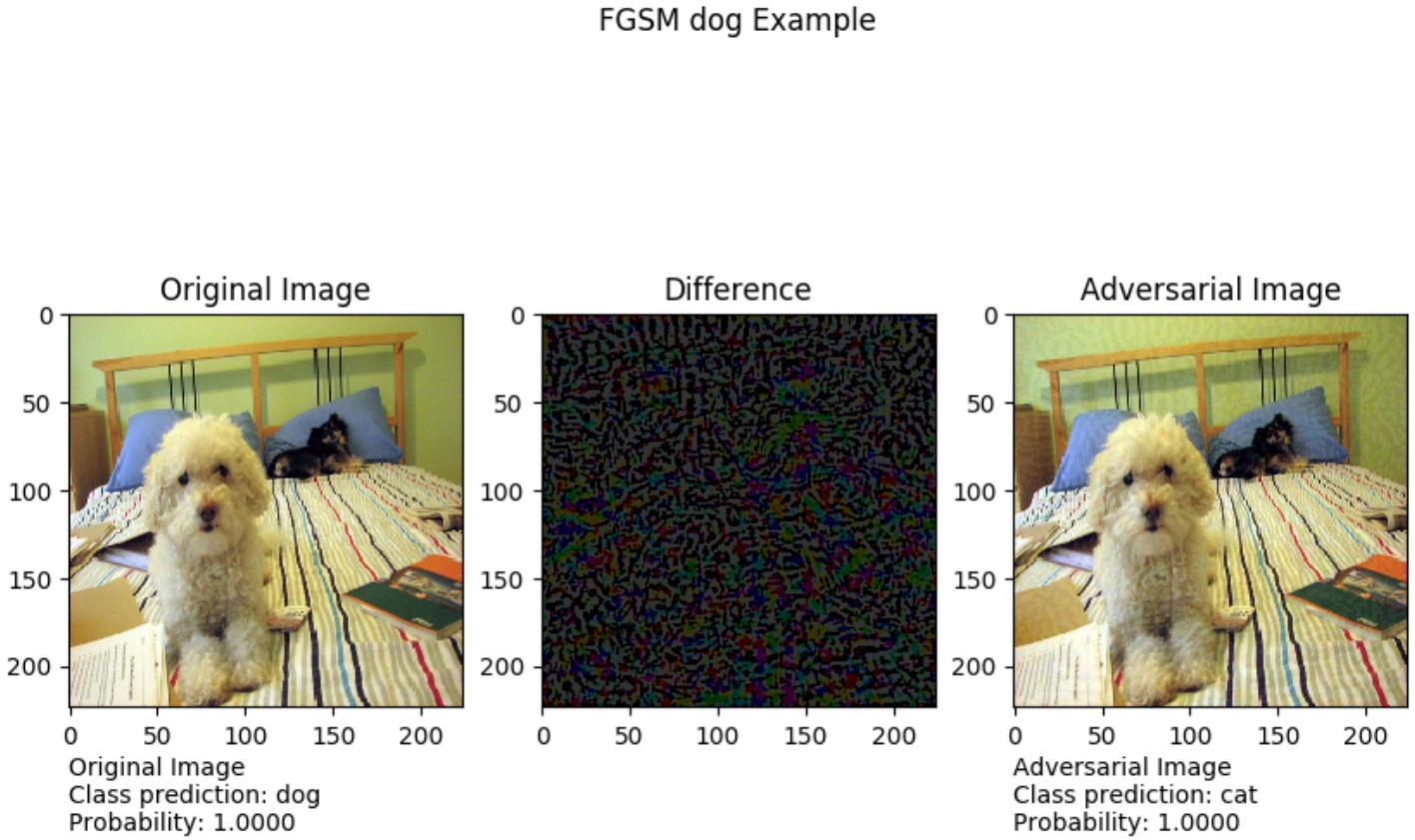

Figure 5.25: Example of FGSM applied to a random image from the dog class. Epsilon = 5. Difference image multiplied by 10.

after the attack is shown in Figure 5.26. Similar to what was seen in CIFAR-10, the PGD attack creates less noise in the image than FGSM does. It is difficult to see the difference in the large images, but a good spot to look is the white area to the upper left of the frog. The difference in noise is most apparent at the 9th epsilon in both regions. An example of the FGSM attack applied to random image is shown in Figure 5.27.

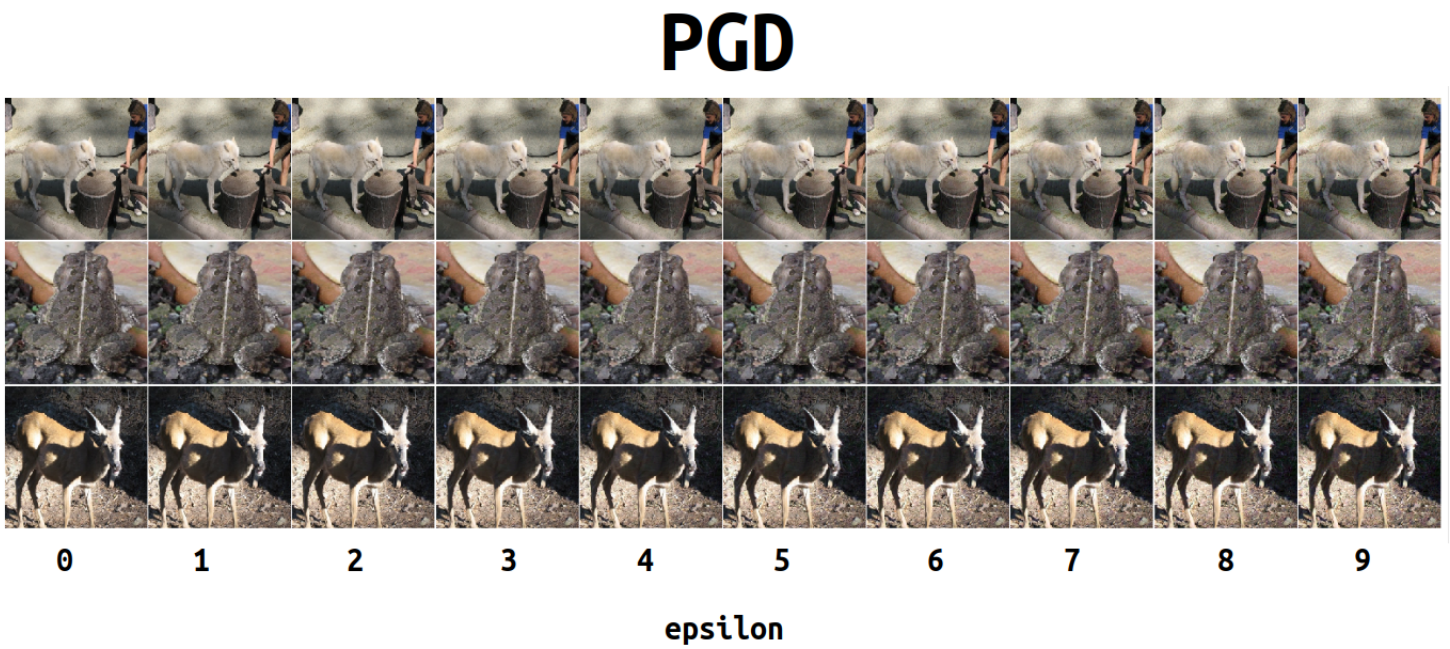

Figure 5.26: Example of PGD with increasing epsilon applied to three random CINIC-10 Large images.

The same method as the FGSM attack will be used for the black-box portion of this attack. It will target the original VGG16 network.

Attack Summary

The results of the white and black box attacks are shown in Figure 5.28. As expected, the white-box attacks on the original R-VGG16 more severely brought down accuracy than their black-box counterparts. An important observation to note is the similar performance between the white and black box attacks. R-VGG16 is simply the VGG16 network with a large amount of regularization applied. Therefore, it is expected that the networks will

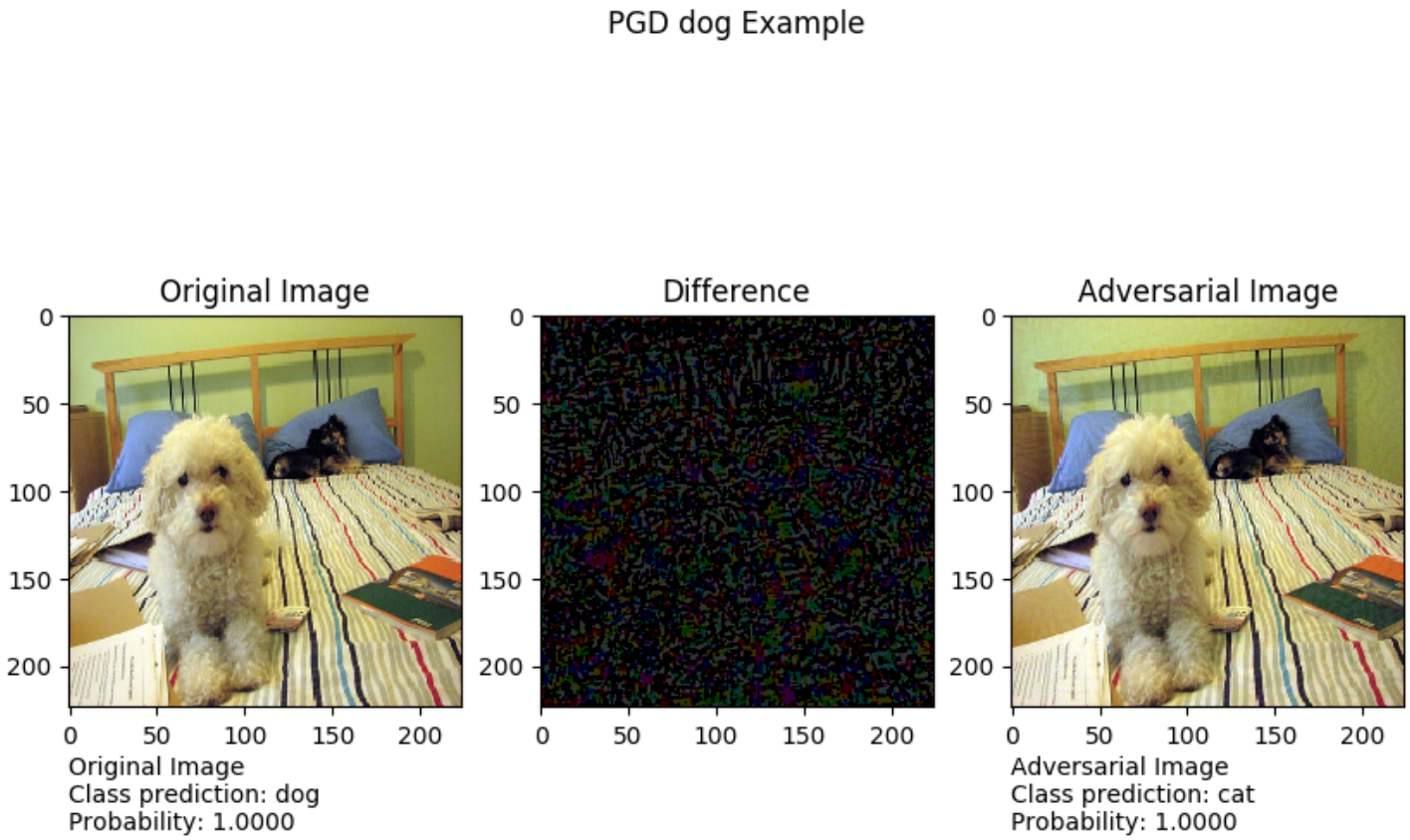

Figure 5.27: Example of PGD applied to a random image from the dog class. Epsilon = 5. Difference image multiplied by 10.

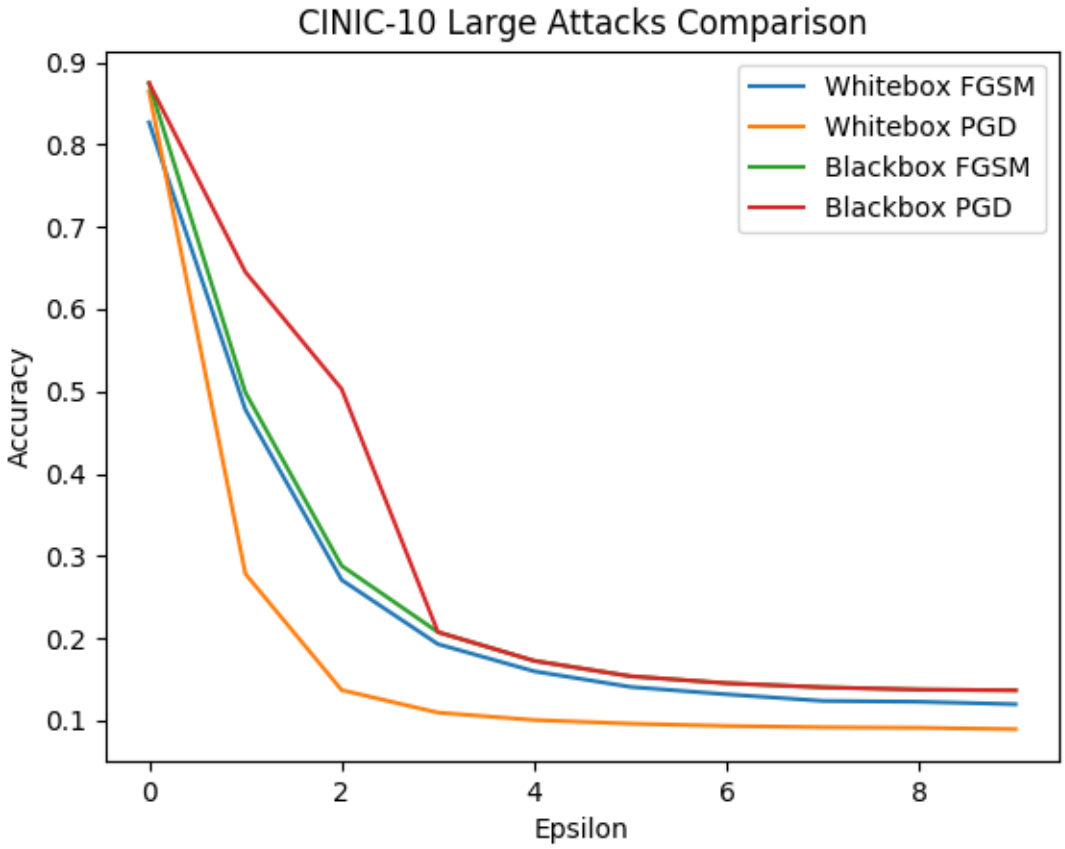

Figure 5.28: Performance comparison of the FGSM and PGD attacks on the white-box R-VGG16 and the black-box VGG16 networks.

behave similarly. PGD in the white-box setting decreases accuracy the most, while the black-box version has a gentler slope as the epsilons increase until it catches up with FGSM at the third epsilon. This might be explained by the lower number of iterations calculated for PGD, which optimizes well in a white-box setting, but possibly less so in the black-box. Even though the networks are similar, they are still not the exact same, which might explain this behavior. However, the white-box PGD continues to be the strongest attack overall.

5.3.2 Defenses

As mentioned in the previous section, this part of the defense involves creating networks that have been trained on filtered images and then subsequently attacking them with their own gradients. This involves combining a clean image with the adversarial noise and then filtering it with the filter corresponding to the network.

Unlike the other sections, where the filter parameters were hand-tuned, these were chosen through a parameter search of various combinations of parameters and epsilons. The best parameters were chosen based on how well they improved adversarial image accuracy while not significantly reducing original accuracy. The ideal result resembled a horizontal line from beginning to end on a graph of accuracy vs epsilons. However, not all 10 epsilons were tested every time. The results were assessed based on the performance on the zeroth epsilon and the final epsilon. This was enough information to select the best parameters for a filter while reducing computations. The search was performed on the strongest adversary, the white-box PGD. If the defense guarded against this attack, then it logically follows that it will guard against other attacks.

The parameters tested in the grid search were limited in order to not become too computationally expensive. A small amount of hand-tuning was performed before the grid-search in order to find a window where accuracy increased on the adversarial images. The parameters were set around this window in an attempt to find the most optimal combination of parameters. There were more parameters available than the ones shown below, but after hand-tuning these extra parameters, they were found to not have any significant effect on improving adversarial image accuracy and were held constant at their default values. All of the below filters were selected for their blurring effects and ability to remove adversarial noise while maintaining original accuracy. The edge preserving filters are the most ideal in that they remove the noise while retaining the image's original structure. The parameters tested are as follows:

- *TVD-ROF:* Weight of 65
  - Tested weight parameters: 45, 55, 65, 75, 85
- *TVD-Chambolle:* Weight of 130

  - Tested weight parameters: 120, 125, 130, 135, 140

- *Non-local Means:* Sigma of 10, h of 60
  - Tested sigma (expected noise variance) parameters: 5, 10, 15, 20
  - Tested h (patch weight decay) parameters: 50, 60, 70, 80

- *Anisotropic Diffusion:* K of 0.1, 20 iterations
  - Tested K (edge sensitivity) parameters: 0.1, 0.5
  - Tested iteration parameters: 10, 20, 30, 40

- *Gaussian + Median Filter* Median kernel size of 5, sigma of 20, Gaussian kernel size of 11.
  - Tested median kernel size parameters: 3, 5
  - Tested sigma parameters: 1, 5, 10, 20
  - Tested Gaussian kernel size parameters: 7, 9, 11

The images with filtering applied to the original, medium epsilon, and strongest epsilon from both attacks are shown in Figures 5.29 and 5.30. Similar to previous results, FGSM is noisier than PGD. The noise discrepancy is most apparent in the strongest epsilon between the two sets of images. All filters cause a significant amount of blurring in the images, but not as much as the smaller images. It is still clear that the image is of a dog in all filter examples, although the proximity of the dog to the camera aids in that.

White-box

This section will show the accuracy of the classifier on the two PGD and FGSM attacks when defended against using the filters described in the previous section. The accuracy obviously decreases due to extreme blurring of the image, visible in Figures 5.29 and 5.30. This can be resolved by training the network on the filtered images, which will be covered in the gray-box section. For now, the accuracies in Figures 5.31 and 5.32 reflect the performance of the model trained on clean images, with attacks generated from the gradients of said model. The

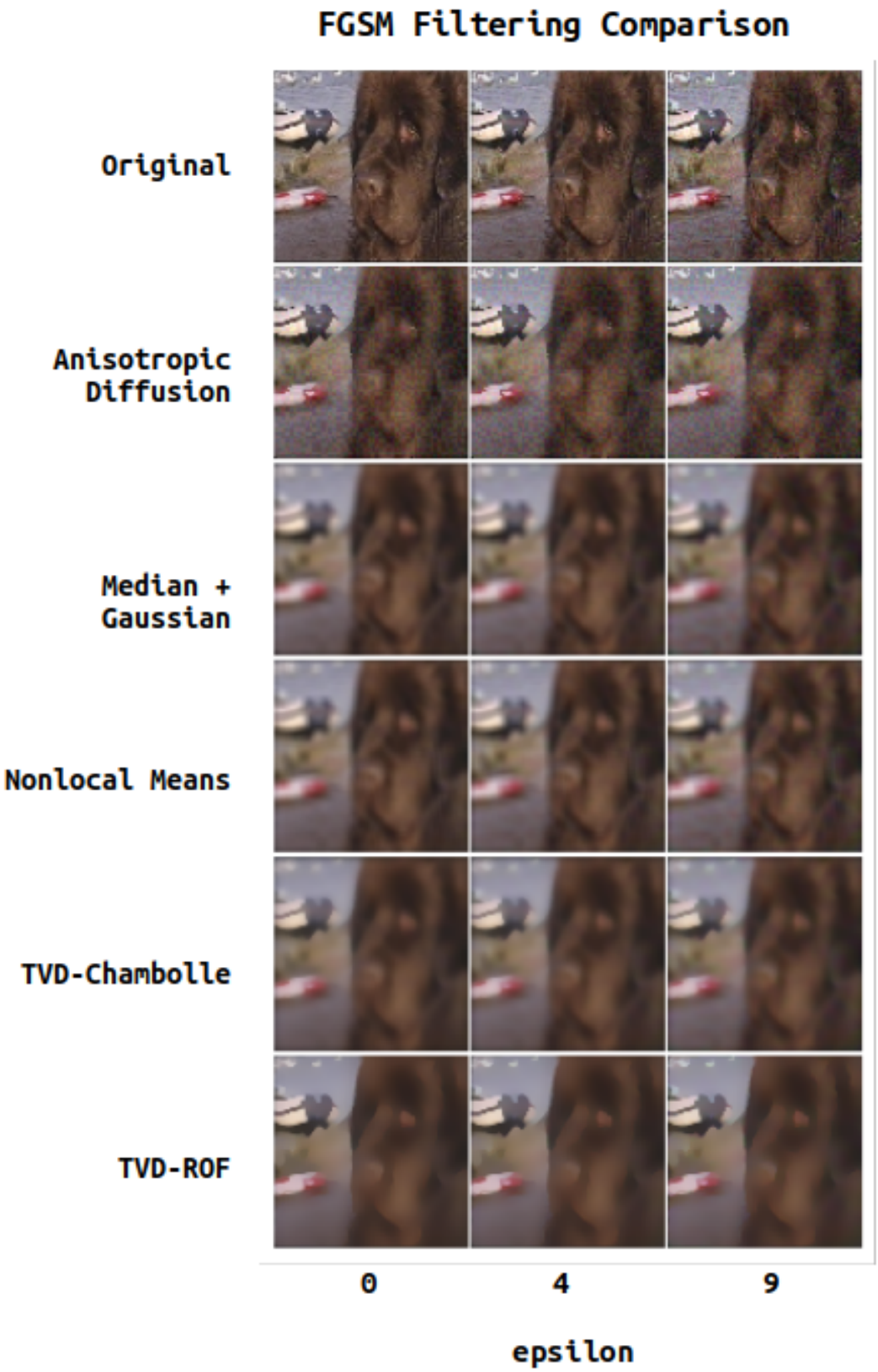

Figure 5.29: Comparison of filters applied to the original, medium epsilon, and highest strength epsilon from the FGSM attack on a random image from the CINIC-10 Large testing set.

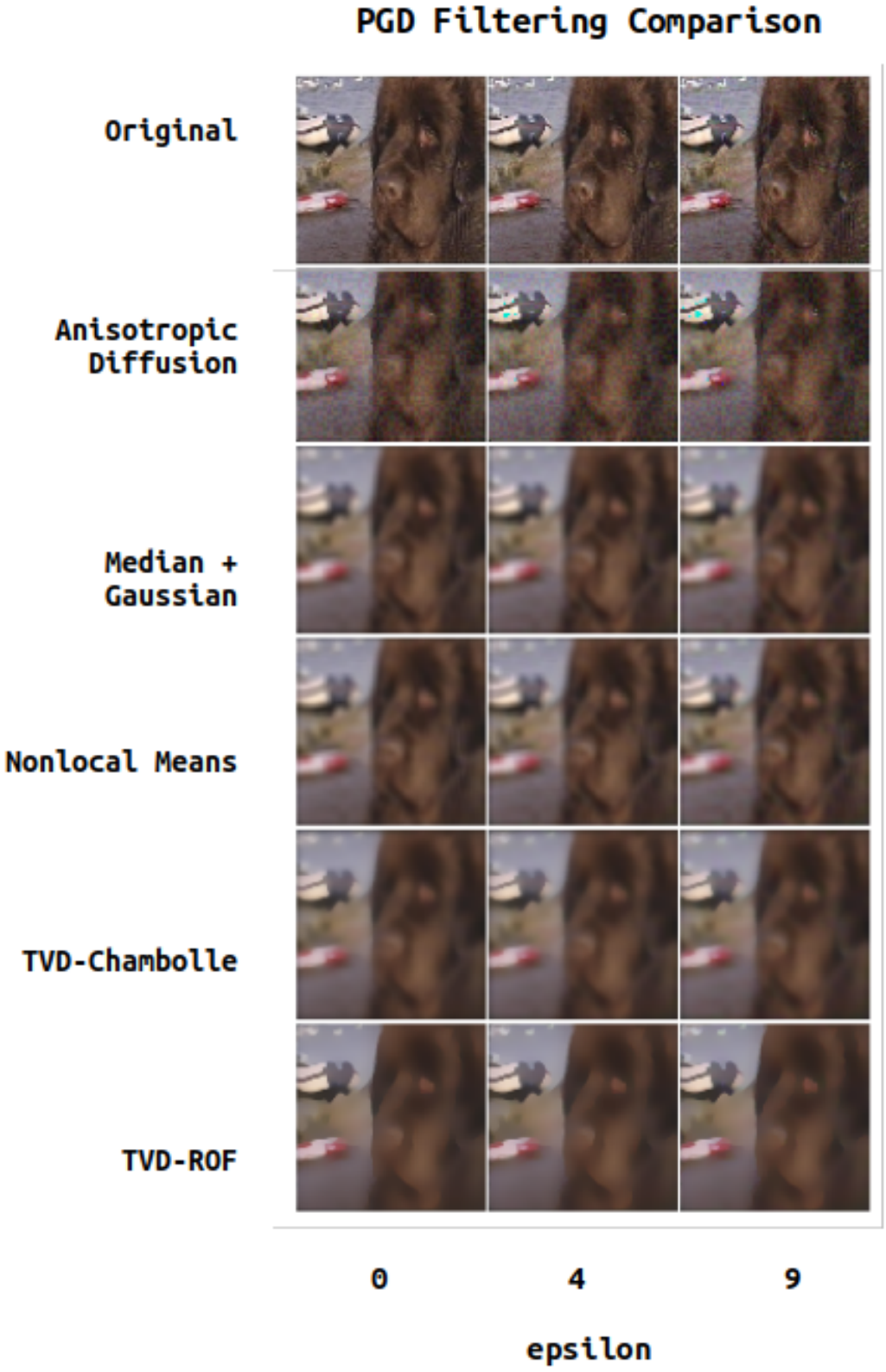

Figure 5.30: Comparison of filters applied to the original, medium epsilon, and highest strength epsilon from the PGD attack on a random image from the CINIC-10 Large testing set.

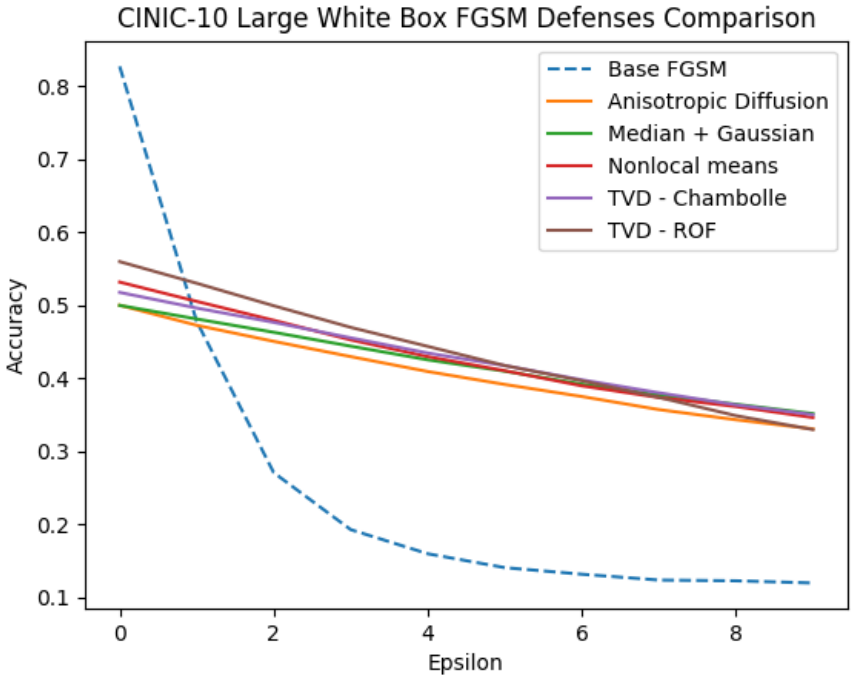

Figure 5.31: Performance of filtering defenses on increasing epsilons on the FGSM attack on the CINIC-10 Large test set.

performance increase on both attacks is nearly identical, with PGD having a slightly larger increase due to its lower initial accuracy. This is most likely because of the extreme blurring that happens within the image. As the epsilons increase, it is clear that the adversarial noise still has an effect, albeit a smaller one, on the images. If there was no effect, then all filtering defenses would be a straight line. The best performer initially is the TVD-ROF filter, while the worst performer is the resizing filter. However, at the highest epsilon, the best performer is median + Gaussian filter. This is most likely due to the double filtering that the median + Gaussian filter performs. Two layers of filtering more severely distorts the image but also more thoroughly obfuscates adversarial noise.

Black-box

Just like in the previous section, all of the black-box attacks have result in similar accuracy when filtered. Once again, this is likely due to the extreme blurring of the images. The

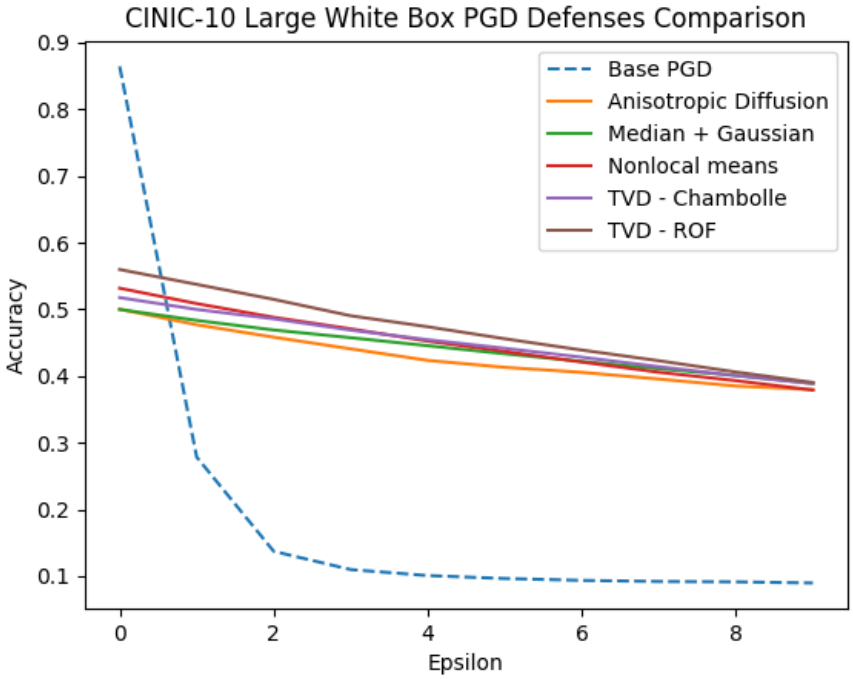

Figure 5.32: Performance of filtering defenses on increasing epsilons on the PGD attack on the CINIC-10 Large test set.

accuracy graphs are shown in Figures 5.33 and 5.34. It is interesting that the models are affected the same way by both of the attacks. Similar behavior was observed in the CIFAR-10 attacks as well. An interesting point to note is the black-box model's accuracy is more negatively affected by the image filtering than the white-box model. This might be explained by VGG16's overfitting to the ImageNet images, while R-VGG16 was significantly more regularized and therefore can generalize more. It is likely this is the case instead of the adversarial images reducing accuracy because the trends are more horizontal in the black-box graphs.

Gray-box

In an attempt to increase accuracy, new models were trained on images filtered with the most optimal combination of parameters discovered in the parameter search. The theory was that this training would increase accuracy on the initial, clean images while still defending

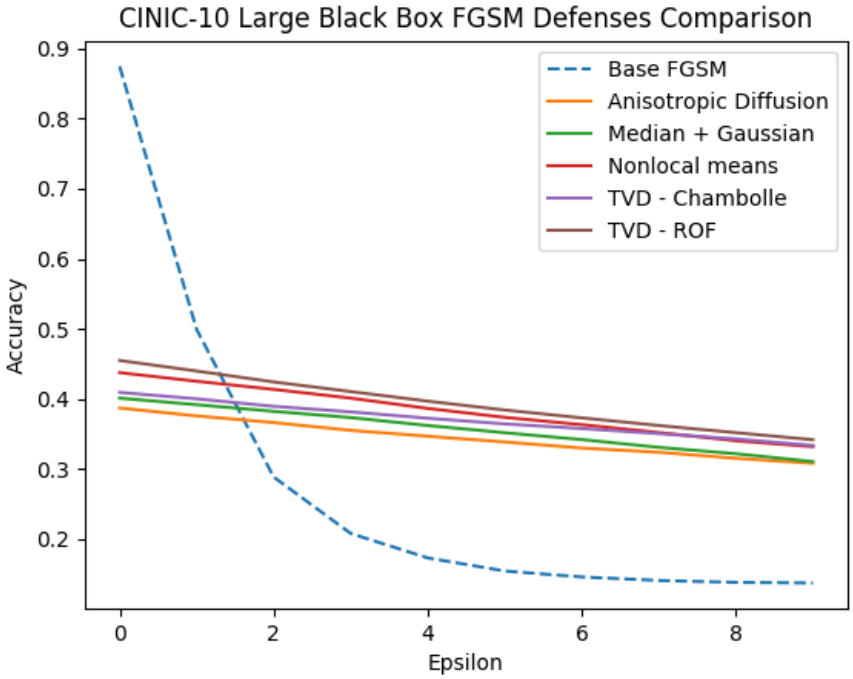

Figure 5.33: Performance of filtering defenses on increasing epsilons on the FGSM attack on the CINIC-10 Large test set in a black-box setting.

optimally against adversarial examples. All new training was performed on R-VGG16. The accuracy score after training reached roughly 60% after ten epochs on all models. This result is acceptable as it still raises the clean accuracy above what was previously seen in the white and black-box sections. In order to do a fair comparison of how well these networks defend against adversarial examples, it was necessary to create the FGSM and PGD attacks using these new networks' gradients. This is almost a white-box setting except that clean images with adversarial noise are being evaluated. The training is to increase accuracy after the appropriate filtering method is applied. A true white-box setting would involve the attacker filtering the image with the network's respective filter first, then applying the adversarial noise to this now filtered image, and then feeding it to the neural network. In this case, the attacker has access to both the filtering method and the loss function of the models. This sort of scenario will be described in the next subsection.

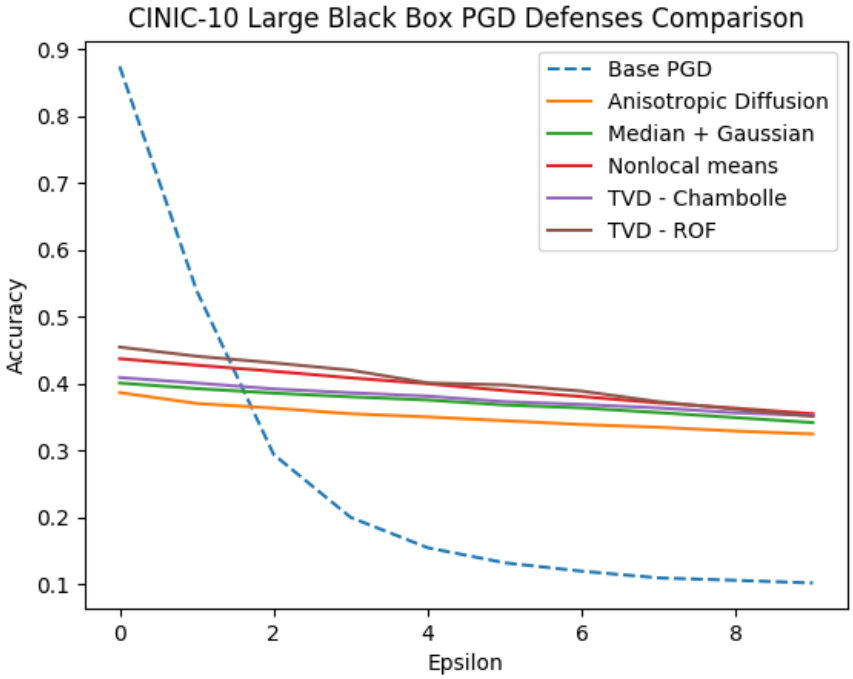

Figure 5.34: Performance of filtering defenses on increasing epsilons on the PGD attack on the CINIC-10 Large test set in a black-box setting.

The graphs in Figures 5.35 and 5.36 compare the gray-box accuracy in three ways. The dashed-dotted black line shows the accuracy of the original network when attacked with the respective adversarial method. The dashed lines show the effectiveness of the filtering defenses on the same original network as before. These groups of lines are the same ones that are shown in Figures 5.31 and 5.32. The only difference is every other epsilon was tested here instead in order to cut down on computational time. The general trend is still clearly visible however. These trends have been included here again to showcase the accuracy gain when the models are trained on the filtered images. As stated in the previous paragraph, in order for this to be a fair comparison, the models need to be attacked with their own gradients. Therefore, several new models were created and attacked. The performance gain is clearly visible. Every defense improves classifier accuracy when the model is trained on the filtered images. All adversarial accuracies on the new models were roughly the same as the base attack accuracy shown in both figures. The best performing filter in both new

instances is the median + Gaussian filter, likely because applying a sequence of filters to the images obfuscates the adversarial noise away more than just a single filter. Another successful defense was the anisotropic diffusion filter, which likely performed well due to its edge-preserving capabilities. The TVD-Chambolle filter also performs well, likely because it also preserves more edges. In general, it seems the best performing filters were the ones that managed to mask the adversarial noise while preserving image structure.

White-box on a Network Trained on Filtered Images

This section will cover a true white-box attack scenario in which the attacker has knowledge of both the model's parameters *and* the filter used in defense. This type of attack happens to counter the defenses outlined above. However, this attack is not realistic for a few reasons. It would be unlikely that the attacker would know which filter is being used, especially if a sequence is being used like the median + Gaussian filter. Additionally, the attacker would need to know the parameters being used in the attack. Finally, the resulting image looks distorted after being filtered and adversarially altered, which would make it look strange to any observer. Therefore, this method should not be considered as a serious attack.

The gray-box attacks and defenses discussed in the last section primarily dealt with applying adversarial noise to a clean image, and then filtering it before classifying it. An interesting way to decrease the network's accuracy is by first filtering the clean image with the respective filter, applying adversarial noise to the now-filtered image, and then passing it to the network for classification. The effectiveness of this method can most likely be explained by the adversarial example being fully optimized for the type of image the network classifies. If the network was trained on these filtered images, and the loss with respect to the input is maximized on such a filtered image, then the attack is most fully optimized to attack this network. The extra layer of filtering at this point does not successfully defend against the adversarial noise as much as before, and it is likely that the amount of image filtering needed to minimize the adversarial effect would be unacceptable. Even though this attack bypasses the defenses, the resulting image becomes very distorted. Therefore, this attack would not be practical in a real setting, and is only being shown here as an interesting finding.

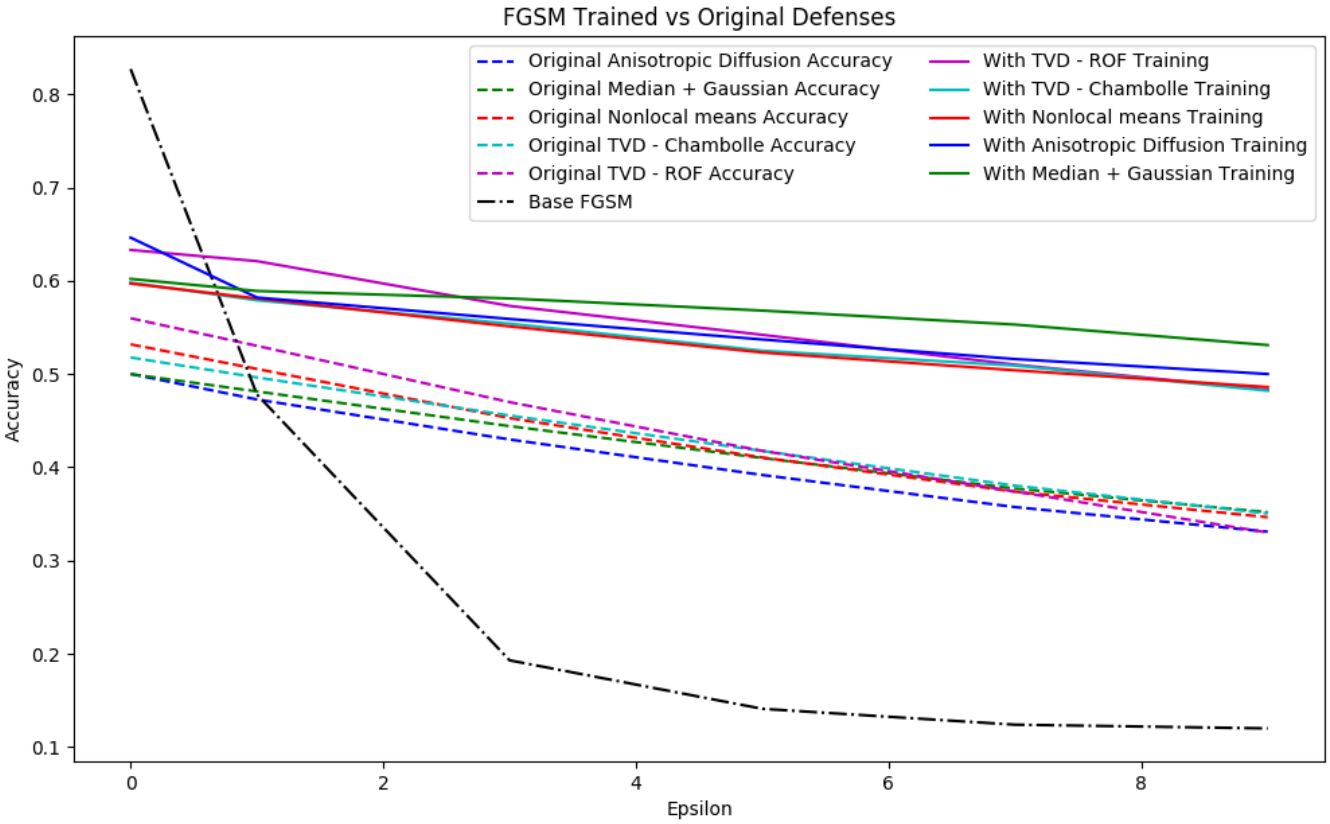

Figure 5.35: Performance of filtering defenses on increasing epsilons on the FGSM attack on the CINIC-10 Large test set after training models on the filtered images.

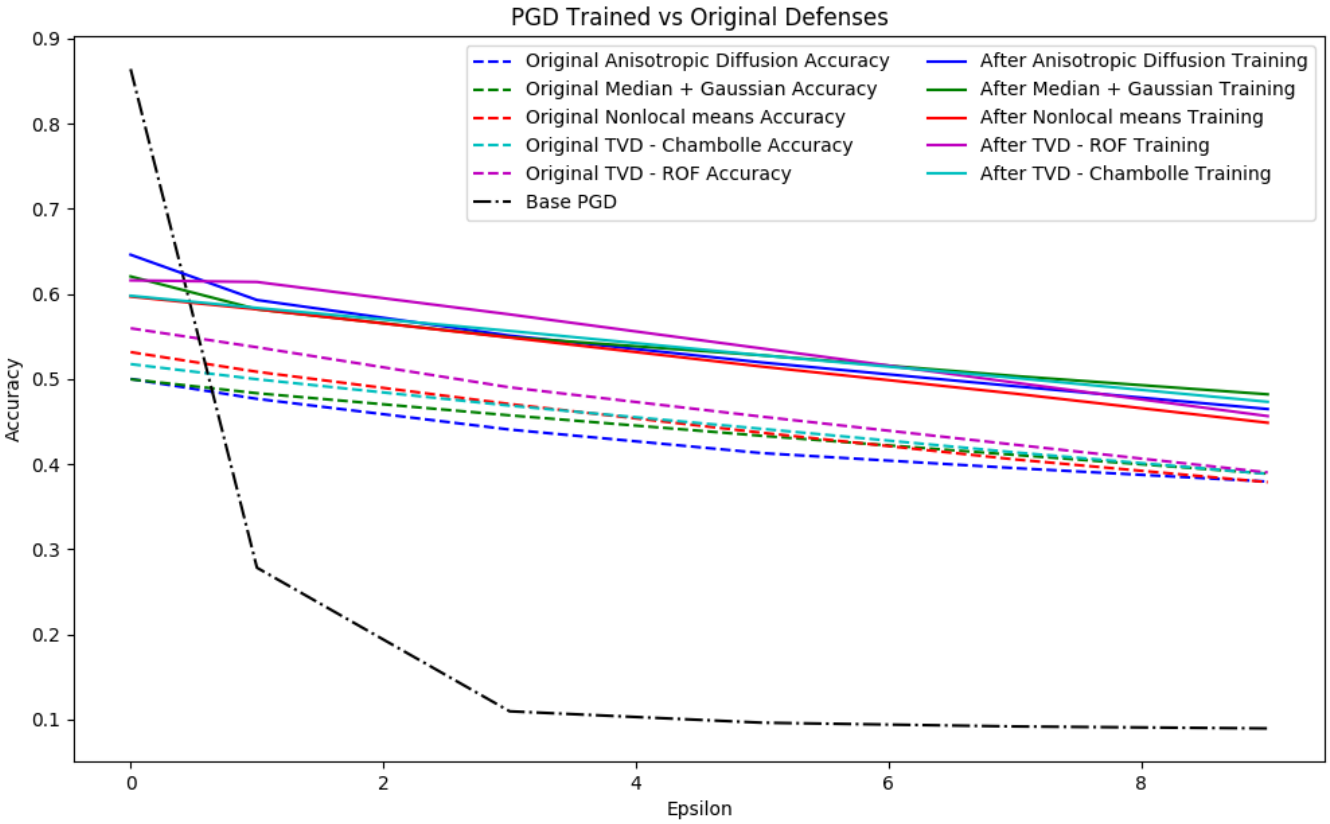

Figure 5.36: Performance of filtering defenses on increasing epsilons on the PGD attack on the CINIC-10 Large test set after training models on the filtered images.

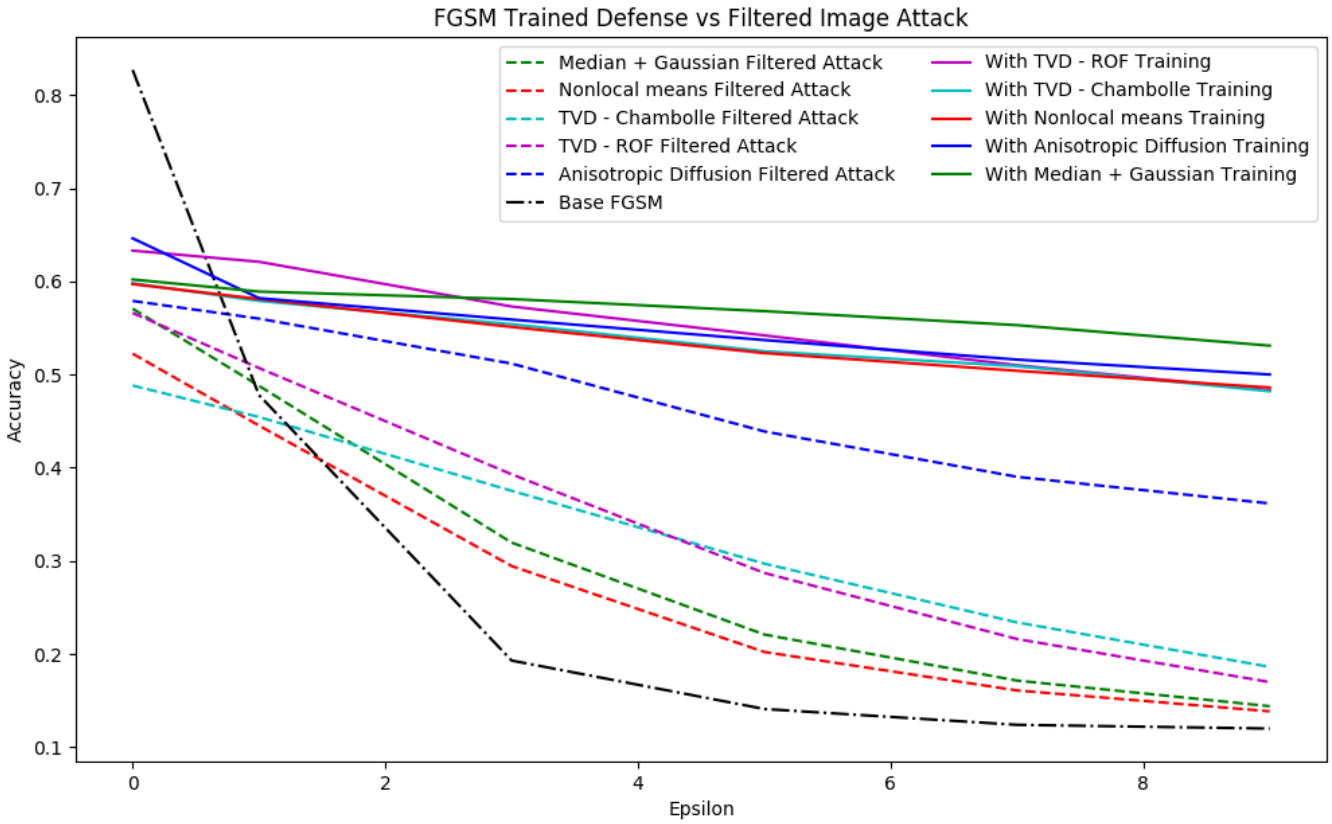

Figure 5.37: Comparison of trained filter defenses and filtered image FGSM attacks.

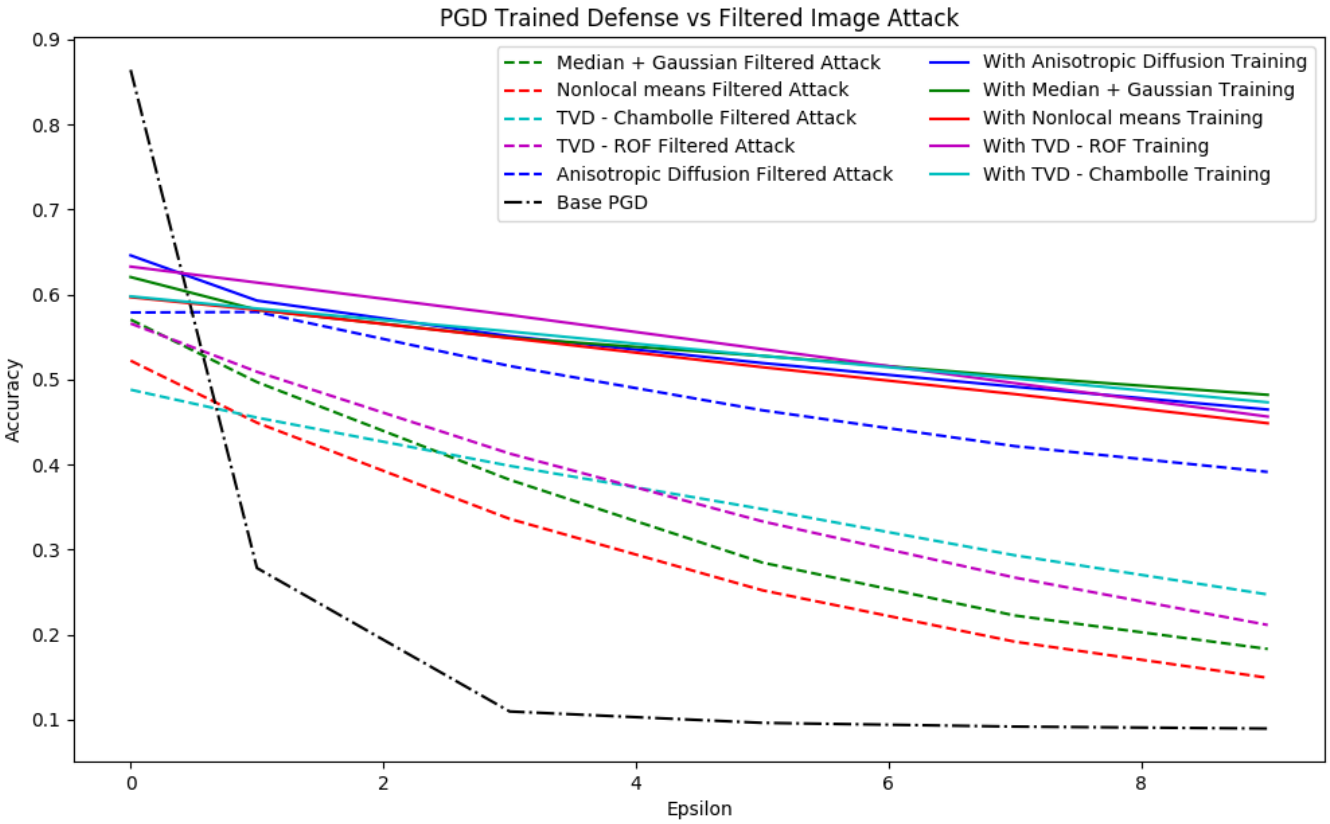

Figure 5.38: Comparison of trained filter defenses and filtered image PGD attacks.

Chapter 6

Conclusions

This thesis has shown that the effect of adversarial examples can be minimized via image filtering. A classifier, whether it is a SVM or a neural network, can be made more robust to adversarial examples with a careful transformation of the input images. This filtering must be kept within a window that does not severely reduce original image accuracy while simultaneously successfully masking adversarial perturbations in the input images. For smaller images, resizing has been shown to be an effective technique, likely because it removes a significant amount of adversarial information in proportion to the size of the image. To mask the transformation in larger images, a more radical transformation is required to minimize the adversarial effect. The best performing filter was a sequence of two filters: the median filter and the Gaussian filter. These two filters did not perform well on their own, but combined they became the most effective defense. This is most likely because the image is masked two times which results in removing even more of the adversarial noise than a single standalone filter. As an avenue for future work, it would be interesting to examine how a sequence of carefully selected filters would serve to increase classifier robustness. A sequence of filters in combination with filtered image training would likely be a powerful defense against adversarial images.

The best performing filters in all sections were the ones that most heavily blurred the images while retaining the image's edges. For example, the TVD-Chambolle filter outperformed the TVD-ROF filter because it better preserves edges and detail while filtering out more unnecessary noise. The worst performer was the non-local means filter, which

likely retained more unneeded information in the image. However, all filters successfully improved adversarial accuracy, with average classification accuracy at roughly 55% on the white-box PGD attack after filtered image training. This a substantial improvement over the initial 7% accuracy rate of the PGD attack. Unfortunately the filtering came with an initial dip in accuracy, with filtered training having an average accuracy of roughly 60% on the test set, much lower than the initial 87% accuracy of R-VGG16. However, this is part of the accuracy-robustness trade-off. What was lost in initial accuracy is improved on with adversarial accuracy.

An explanation for why the blurring filters worked to remove adversarial noise is explained in the paper by Ilyas et al. [25]. They explain that "adversarial examples are not bugs, they are features". What they mean by this is that too often models use non-robust features to generalize on a set of images. The reason why some models perform so well is not because they "learn" the structure of objects, but instead can use imperceptible features, not relevant to the subject of the image, to generalize on a data set. For example, a human knows that a cat is a cat based on its ears, the shape of its snout, etc... The models commonly used in image classification do not use these features, and instead use "highly predictive, yet imperceptible features". Adversarial examples can take advantage of these features to change the classification of an image while not significantly altering it appearance. They prove this by transforming CIFAR-10 into a data set containing only robust features and achieving high accuracy on both clean and adversarial images. Similarly with the removal of noise in these experiments, it is most likely that these image filters remove non-robust features, as well as adversarial noise, from the background and foreground of the images, leaving only the basic structure of the image as a result. For example, the anisotropic diffusion filter is commonly used in medical imaging as a means to remove noise from a image so that the physician can examine the basic structure only. Every filter used here performs a similar task; removing unnecessary noise from an image while retaining the original image structure. This is most likely why the blurring filters performed as well as they did.

The results of the trained models in the gray-box section were not as uniform, likely because of the different amount of loss each model incurred while training. Some models achieved lower training accuracy than others, meaning that a higher percentage of the data

set was available to attack. Even with R-VGG16's regularization, not all of the gradients in the data set were non-zero, so some images could not be altered. An avenue for future work would be to focus more resources on training the models with the filtered images to increase accuracy as much as possible so that filter comparisons are more fair. This would allow for more uniform results and higher accuracy if the masking from the image filtering is effective enough.

Avenues for future work would include using the full data set instead of a subset. The limited computational resources unfortunately required that only a subset of the data be used, but it would be interesting to see how the results look on a larger collection of images. There could be more candidates for adversarial examples and therefore more uniform results when performing comparisons.

Additionally, some interesting defense options could include altering the image input in more transformative ways. To more effectively mask the adversarial noise and retain image structure only, it would be interesting to try some transformations such as taking the derivative of the image, the Fourier transformation, or using the Wiener filter to guess the adversarial noise and reconstruct the image from it. Also, according to Goodfellow et al. [19] the high dimensionality of the input causes the example to be effective in the first place, so it would be interesting to use principal component analysis to reduce the image's dimensionality.

Training the models on these suggested transformations is a requirement to keep initial accuracy high. With these models comes the potential for ensemble classification. If the models agree on the image classification after transformation time, then it is likely that the image classification has been successfully defended. Otherwise, a scenario for adversarial example detection would be simple to employ in such an ensemble.

Overall, filtering the images to be more robust against adversarial examples was effective but at the cost of original accuracy. Based on the results of the experiments, the most effective defense is one that simplifies the image, obscuring the adversarial noise, while preserving the image structure and removing unnecessary features. The defense should be tested on the strongest attack available to make it robust to all attacks. A highly accurate

model trained on these filtered images should be used to receive the attacks to increase classification accuracy.

The existence of adversarial examples exposes the flaw in how convolutional neural networks "see" the world. Although they can be extremely accurate on image recognition tasks, the fact that some small perturbations in the input can cause an otherwise accurate classifier to completely misclassify an image with high confidence exposes that these networks are learning the "wrong" things. These proposed defenses do not fix the fundamental issue of adversarial examples, but are just one temporary solution of many as the field of computer vision and deep learning matures.

Bibliography

[1] Akhtar, N. and Mian, A. (2018). Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430. 5, 22

[2] Bhandare, A., Bhide, M., Gokhale, P., and Chandavarkar, R. (2016). Applications of convolutional neural networks. *International Journal of Computer Science and Information Technologies*, 7(5):2206–2215. 1

[3] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. 28

[4] Brendel, W., Rauber, J., and Bethge, M. (2017). Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*. 14

[5] Buades, A., Coll, B., and Morel, J. M. (2004). On image denoising methods. 18

[6] Carlini, N. and Wagner, D. (2017). Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE. 5, 12, 20

[7] Chambolle, A. (2004). An algorithm for total variation minimization and applications. *Journal of Mathematical imaging and vision*, 20(1-2):89–97. 18

[8] Chang, T.-J., He, Y., and Li, P. (2018). Efficient two-step adversarial defense for deep neural networks. *arXiv preprint arXiv:1810.03739*. 8

[9] Chen, J., Jordan, M. I., and Wainwright, M. J. (2019). Hopskipjumpattack: A query-efficient decision-based attack. *arXiv preprint arXiv:1904.02144*, 3. 14

[10] Chen, P.-Y., Zhang, H., Sharma, Y., Yi, J., and Hsieh, C.-J. (2017). Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26. 13

[11] Chollet, F. et al. (2015). Keras. https://keras.io. 41, 48, 58, 59

[12] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*. 55

[13] Darlow, L. N., Crowley, E. J., Antoniou, A., and Storkey, A. J. (2018). Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*. 40, 41

[14] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*. 41, 44

[15] Efros, A. A. and Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. 20

[16] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. (2018). Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1625–1634. 3

[17] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874. 27

[18] Gao, J., Wang, B., Lin, Z., Xu, W., and Qi, Y. (2017). Deepcloak: Masking deep neural network models for robustness against adversarial samples. *arXiv preprint arXiv:1702.06763*. 23

[19] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*. 2, 5, 6, 9, 20, 22, 120

[20] Graese, A., Rozsa, A., and Boult, T. E. (2016). Assessing threat of adversarial examples on deep neural networks. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 69–74. IEEE. 2

[21] Guo, C., Rana, M., Cisse, M., and Van Der Maaten, L. (2017). Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*. 20

[22] He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778. ix, 55, 58

[23] He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer. 55, 58

[24] Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154. 1

[25] Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. (2019). Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, pages 125–136. 119

[26] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*. 54

[27] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. ix, 40, 42, 45

[28] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105. 1

[29] Kurakin, A., Goodfellow, I., and Bengio, S. (2016). Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*. 3, 20

[30] Laidlaw, C. and Feizi, S. (2019). Functional adversarial attacks. In *Advances in Neural Information Processing Systems*, pages 10408–10418. 5

[31] LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database. 26, 40

[32] Lee, S. and Lee, J. (2018). Defensive denoising methods against adversarial attack. 21

[33] Lim, S.-H., Young, S., and Patton, R. (2016). An analysis of image storage systems for scalable training of deep neural networks. viii, 42

[34] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*. 5, 6

[35] Molnar, C. (2019). *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/. 4, 6

[36] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016). Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582. 11, 20

[37] Narodytska, N. and Kasiviswanathan, S. P. (2016). Simple black-box adversarial perturbations for deep networks. *arXiv preprint arXiv:1612.06299*. 13

[38] Papernot, N., McDaniel, P., and Goodfellow, I. (2016a). Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*. viii, 26, 27

[39] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. (2016b). The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE. 11, 22

[40] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830. 25, 27

[41] Perona, P. and Malik, J. (1990). Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on pattern analysis and machine intelligence*, 12(7):629–639. 19

[42] Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789. 2

[43] Rudin, L. I., Osher, S., and Fatemi, E. (1992). Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268. 17

[44] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252. 1

[45] Sharif, M., Bhagavatula, S., Bauer, L., and Reiter, M. K. (2016). Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 acm sigsac conference on computer and communications security*, pages 1528–1540. 3

[46] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. 59

[47] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*. 2, 3, 4, 5, 9

[48] Xu, W., Evans, D., and Qi, Y. (2017). Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*. 23

[49] Yamashita, R., Nishio, M., Do, R. K. G., and Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629. 1

Vita

Natalie Elizabeth Bogda was born in St. Joseph, Michigan in 1995 to Marina and Alex Bogda, both of whom immigrated here from the Soviet Union in pursuit of the American dream. She graduated from the Webb School of Knoxville in 2013. She earned a Bachelor's of Science degree from the University of Tennessee, Knoxville with a major in Computer Science in 2018. She discovered her passion for machine learning and computer vision during this time, and continued onto graduate school, performing research under the guidance of Dr. Amir Sadovnik in the Understanding Visual Understanding (UVU) lab. After earning her master's degree, she will start work in industry as a data scientist.