A BROAD SPECTRUM DEFENSE AGAINST ADVERSARIAL EXAMPLES

by

Sean McGuire

A Thesis

Submitted to the
Department of Electrical & Computer Engineering
College of Engineering
In partial fulfillment of the requirement
For the degree of
Master of Science in Electrical & Computer Engineering
at
Rowan University
June 20, 2022

Thesis Chair: Robi Polikar, Ph.D., Department Head, Department of Electrical & Computer Engineering

Committee Members:
Umashanger Thayasivam, Ph.D., Professor, Department of Mathematics
Ravi Ramachandran, Ph.D., Professor, Department of Electrical & Computer Engineering
Ghulam Rasool, Ph.D., Professor, Department of Electrical & Computer Engineering

Dedication

This thesis is dedicated to my parents. For their persistent love and encouragement.

Acknowledgements

Throughout the writing of this dissertation I have received a great deal of support and assistance.

I would first like to express my sincere gratitude to my thesis advisor, Dr. Robi Polikar for his continuing support and guidance throughout this process. Your feedback and expertise was invaluable in the development of this thesis.

I would like to thank Muhammed Umer, Glenn Dawson, and Jacob Epifano for the stimulating research discussions leading to breakthroughs in this work.

I would also like to thank my parents for their constant support and encouragement. I am deeply grateful to my friends and family for their unwavering support and belief in me.

The work was partly supported by the US Department of Education through the Graduate Assistance in Areas of National Need (GAANN) program, Award Number P200A180055.

Abstract

Sean McGuire
A BROAD SPECTRUM DEFENSE AGAINST ADVERSARIAL EXAMPLES
2021-2022
Robi Polikar, Ph.D.
Master of Science in Electrical & Computer Engineering

This work focuses on developing a new adversarial machine learning defense capable of detecting and rejecting adversarial examples. Adversarial examples are data points crafted to fool a model into giving an incorrect prediction. Everyday, machine learning is used to solve more problems. As our dependence on these models increases, it is vital to recognize their vulnerability to adversarial samples. This defense, called simply *Broad Spectrum Defense (BSD)*, is designed to work across a wide range of datasets and attacks while requiring as few hyperparameters as possible. Another benefit of the proposed BSD over existing defenses is its reliance only on test data: Unlike most other approaches, BSD does not train its detectors using adversarial data, removing an inherent biases present in other works. Extensive set of experiments showed that BSD outperforms existing detector based methods such as MagNet and Feature Squeezing. We believe BSD will inspire a new direction in adversarial machine learning to create a robust defense capable of generalizing to existing and future attacks.

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation: Adversarial Machine Learning Defenses

Machine Learning models are now an indispensable tool used across a wide array of fields from medicine to business to defense among many others. Adversarial machine learning explores the process of attacking and defending these models. It is vital that we recognize the inherent vulnerability of these models and develop strong defenses to secure them against malicious attacks.

1.2 Adversarial Machine Learning

Adversarial machine learning (AML) refers to scenarios where an adversary, presumably with malicious intent, is present in the environment. The adversary seeks to manipulate the data to compromise the model. An AML framework typically includes two actors: the adversary (the attacker) and the model creator (the defender). The attacker's goal may simply be to cause the model to generally perform poorly, or perhaps cause the model to misclassify a specific set of instances. The defender, on the other hand, wants to ensure that the model is secure and robust to attacks. The adversary may be aware of a potential defender and attempt to craft an attack that then bypasses the defense. This cat and mouse game continues perpetually. It is widely accepted that the attacker is at an advantage as defenders often react to new attacks, but there has not been much work on developing a proactive defense which can generalize to a wide spectrum of attacks.

1.3 Problem Statement

Machine learning models are susceptible to adversarial attacks, which can exploit inherent vulnerabilities of a classifier. Due to these vulnerabilities, an adversary can easily perturb a sample very slightly and force an incorrect classification of that sample. As more machine learning models are implemented in critical tasks, it is vital that a defense is constructed to secure these models.

While new defense mechanisms have been developed – and continue to be developed – many of these existing defenses are reactionary, and only work on a subset of attacks or datasets for which they are specifically designed. Additionally, some defenses work by learning to approximate attacks, but those defenses are typically trained on one type of attack. In such cases, it is unrealistic to assume that a defense designed to mitigate one type of attack will generalize to different types of attacks. An effective defense must generalize across a variety of attacks and datasets without being biased by any form of training on any given attack.

1.4 Scope of Thesis

This thesis introduces the formulation of a broad spectrum defense (BSD) that is capable of detecting adversarial samples. The primary focus of this work is on neural networks, but BSD can be suitably modified to work with any classification model. This work demonstrates the proposed defense's ability to detect adversarial samples on a wide variety of attack types and datasets, and further shows that it outperforms many of the existing works. An analysis of the defense is performed under a gray box scenario that assumes the attacker has access to a realistic amount of information about the model.

1.5 Research Contributions

A proactive defense strategy is proposed, developed and evaluated in this effort. The proposed broad spectrum defense:

1. introduces a proactive approach providing considerable – often significant – improvement over many of the existing evasion attacks;

2. is capable of generalizing to future attacks;

3. scales to large models and datasets, compatible with networks of any size and can analyze data with little delay;

4. has the ability to work in an online setting, as it does not require any retraining of the existing classifier;

5. is data agnostic, that is, it can be used with image and non image data;

6. can be suitably modified to work with non-neural network type classifiers;

7. while primarily designed for evasion attacks, it has the potential to generalize to poisoning attacks.

1.6 Organization of the Thesis

Chapter 2 provides an overview and background for neural networks and adversarial machine learning, and also introduces the important terminology used throughout this thesis. In Chapter 3, existing adversarial attack and defense approaches are discussed in detail. Chapter 4 introduces the broad spectrum defense and discusses the modules used to construct the defense including the Class Divergence Detector. Chapter 5 covers a discussion of the experiments performed, the results of these experiments, as well as comparisons between BSD and existing approaches. Chapter 6 concludes this thesis and explores possible avenues for future work.

Chapter 2

Background

2.1 Adversarial Machine Learning

Adversarial machine learning includes the process of attacking and defending machine learning models. Machine learning models are inherently vulnerable to adversarial attacks, an adversarial attack describes the process of generating a sample which is misclassified even though it is highly similar to samples that are classified correctly [15]. Defending against such attacks has proven to be a difficult challenge as every time a defense is proposed, other attacks appear that thwart the defense. This cat and mouse game continues endlessly, but it appears that the attacker always has the upper hand. Before exploring specific attacks and defenses one must understand the environment in which an attacker operates, along with the attackers goals. This section will introduce a high level taxonomy used to describe adversarial attacks and defenses.

2.1.1 Attack Taxonomy

The attack taxonomy is used to describe the nature of the attacks based on the amount and nature of information available to the attacker. The first attack taxonomy was formulated by Barreno *et al.*, whose taxonomy introduced the now commonly used terminology such as *causative (poisoning), explorative (evasion), targeted and indiscriminate* [1]. Barreno's taxonomy primarily reflects the attacker's intentions. The next taxonomy was developed by Biggio *et al.*, whose taxonomy then introduced the concepts of perfect knowledge (white box), limited knowledge (gray box), and no-knowledge (black box) attacks [3]. Biggio's taxonomy, on the other hand, reflects the amount of information that is available to the attacker. Depending on such amount and nature of information available, the attacker can craft stronger or more strategic attacks. An agreed upon taxonomy is helpful in describing the intended environment for an attack or defense.

2.1.2 Attack Taxonomy with Respect Attacker's Available Information

2.1.2.a Black Box Attacks. Attacks designed based on the amount of information available to the attacker starts by the assumption of little or no information being available to the attacker, introducing the idea of a zero knowledge scenario, also known as the *black box attacks*. In the black box setting, the attacker does not know anything about the model architecture or its parameters. In most black box formulations, however, the attacker is assumed to know, possibly, about the application domain, and may have access to the test data. Given one or more samples from the test distribution, an attacker has an initial point used to construct adversarial examples. Without this point, the problem becomes incredibly difficult as the adversary would not have a ground truth that could be perturbed. Evaluating defenses in a black box scenario reveals performance of the model when an attacker can only probe the classifier by passing in test data points and receiving a classification. The attacker can utilize this feedback to craft an attack point with more impact. The simplest black box attacks can be done by perturbing the original test data to the point of misclassification. Two of these naïve methods are additive uniform noise and Gaussian blur [11]. These two types of attacks perturb the sample by adding Gaussian or uniform noise to the data. If the samples classification remains unchanged, the magnitude of noise is increased until misclassification occurs. It is important to evaluate a defense on these naïve methods as a robust defense should prove effective against against a large variety of attacks.

2.1.2.b White Box Attacks. In the opposite scenario of *white box attacks* the attacker has complete access and knows everything there is to know about the model. That level of information includes the model itself, its parameters, training and test data as well as any defenses used along with their parameters. A white box attack is unrealistic, of course, but it represents the worst case scenario from the model's (and therefore the defender's) perspective. Some examples of white box attacks include Fast Gradient Sign Method (FGSM), Projected Gradient Decent (PGD), the Carlini Wagner attack, and many other that will be introduced in the next chapter.

2.1.2.c Gray Box Attacks. A more realistic - and often used - attack type is the so-called *gray box attacks*. Gray box attacks describe any scenario along the spectrum between complete knowledge (white box) and zero knowledge (black box) cases. Some of these variations include when the attacker is aware of the model (or perhaps even the defense applied), but not its parameters. In this case the attacker would attempt to construct the model (or its defense) itself with some arbitrary parameters, attack the model and transfer these adversarial samples onto the target classifier. Another example of gray box attack can allow the attacker to access all parameters of the classification model, but not its defenses. The attacker may then attempt to craft white box attacks against the classifier and attempt to transfer them to the defended model. The gray box scenario often includes the use of surrogate models. A surrogate model is a model selected by the attacker; the attacker trains the surrogate and creates attack samples against it. Finally, the attacker transfers these samples to the target model. Some examples of gray box attacks include the above mentioned attacks such as FGSM, PGD, and the Carlini Wagner attack. In a gray box scenario, these attacks would be performed against a surrogate model (a model thought to approximate that used by the defender). In the gray box scenario described in this work, the adversary has complete knowledge of the model, but the adversary has no knowledge of the defense, so in this case the attacker launches a “white box” attack against the model, but the attack is really “gray box” as the samples are operating on a defended model.

2.1.3 Attack Taxonomy with Respect to Attacker’s Goals

2.1.3.a Targeted Attacks. In a targeted attack, an adversary is interested in forcing a misclassification to a specific class. For example a targeted attack may involve an adversary seeking to have all stop signs classified as speed limit signs. In a targeted attack, the attacker has a narrow goal of just impacting or classifying samples into a single “target” class. Any of the above mentioned attacks can function as targeted attacks by slightly modifying the attacks parameters.

2.1.3.b Untargeted (Indiscriminate) Attacks. An untargeted attack occurs when the adversary seeks to force a misclassification to any class and does not care the class into which the sample is misclassified. An untargeted attack is successful if the sample is classified as any class besides the correct class. Untargeted attacks are able to generate adversarial samples with smaller perturbations than their targeted counter parts. This can be understood by considering a classification problem with 100 different classes. The targeted adversary may need to perturb one image towards the classification boundary of a class which resides on the opposite side of the decision surface. An untargeted attack can perturb the image until it crosses any decision boundary which can result smaller distortions. Some of the above mentioned attacks are originally implemented in an untargeted form, so these require no modification, while other attacks that are natively "targeted" can be made to be "untargeted" by performing attacks against every class and selecting the sample with the smallest perturbation.

2.1.3.c Poisoning Attacks. In a poisoning attack, the adversary has the ability to insert a number of samples into the *training set*. The samples inserted into the training set can be used to completely devastate the entire model, or they can be used to target a specific region of the model. The first mention of learning with adversarial data in the training set can be traced back to Kearns *et al.* in the 1998 paper titled *Learning in the Presence of Malicious Errors* [18]. However the concept of a poisoning attack, or intentionally comprising a machine learning model at training time was first introduced in 2008 by Nelson *et al.* [31]. A poisoning attack aims to manipulate the behavior of a given model typically by introducing maliciously crafted data points into the training data. A poisoning attack can have different goals: for example, one can seek to introduce the greatest amount of error into the model by crafting attack samples which will cause the misclassification of test samples, as was the case with the poisoning attack against the spam filter in Nelsons work. One downside of the poisoning attack that seeks to compromise the classifier – from the attacker's perspective – is that the attack may be obvious. When the model is attacked, the classification performance may be significantly degraded, which can then be easily detected.

A more modern approach to poisoning attacks is a targeted poisoning attack. In a targeted poisoning attack, the adversary seeks to control a small region of the feature or label space by inserting malicious samples into the training set defining a specific region of that space. After training, the model will return a high test performance overall, but unbeknownst to the victim, the adversary has inserted a *backdoor* which can be exploited. The purpose of the *backdoor attack* is to insert samples into the training set which have some (possibly imperceptible) pattern. The model then learns to associate this pattern with a given class as chosen by the attacker. After training is complete, the adversary can then insert any data point with the backdoor and cause the intended and targeted misclassification. This process was first demonstrated by Chen *et al.* in 2018 [7].

There exists many real life situations in which a backdoor attack can be effectively used. One example is facial recognition: if the adversary has inserted backdoor samples with pink sunglasses with the incorrect label as an *authorized user*, an unauthorized user can then walk through security with pink sunglasses and bypass the facial recognition model. A different example comes in the form of autonomous vehicles. Autonomous vehicles often use deep learning models to detect and classify objects in their surroundings. If a backdoor is added to the model which associates the presence of a sticker with a speed limit sign, the sticker can be placed on “Stop” signs and the model will then recognize a stop sign as a speed limit sign, with potentially deadly consequences. Physical backdoors were explored by Eykholt *et al.* where they demonstrated that they can make backdoor patterns that mimic normal events in the real world such as stickers on the stop sign as shown in Figure 1 [10].

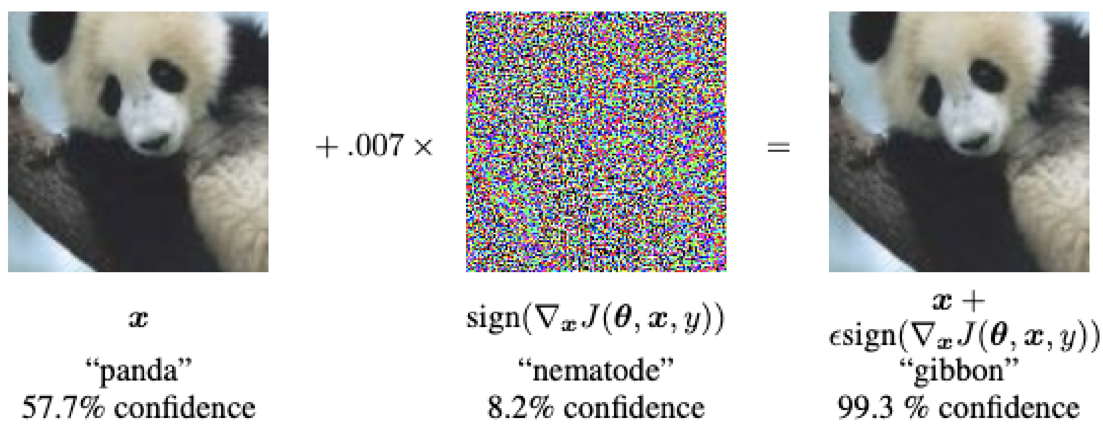
Figure 1
Example of a Backdoor Attack on Physical Object from [10]

2.1.3.d Evasion Attacks. The focus of this thesis is on developing defenses against a broad spectrum of evasion attacks. Evasion attacks are therefore explained in more detail below, with specific strategies and types of evasion attacks described in Chapter 3.

Unlike poisoning attacks, evasion attacks typically occur at test time. In an evasion attack, the adversary seeks to perturb a sample, pass it to the model for evaluation with a deliberate goal of having it misclassified. The first mention of an adversary at test time in the machine learning domain comes from Dalvi *et al.* [8]. Dalvi mentions the presence of an adversary and a game that occurs between the adversary and the classifier, in which the classifier attempts to protect itself against the attack (describing the presence of the defender). This cat and mouse game describes the iterative nature of the creation of adversarial attacks and the response of creating robust defenses. Dalvi explains in detail how an adversary can craft samples that can be misclassified using dynamic programming.

More recently, however, evasion attacks have been brought to center stage by Goodfellow et al's work in the – now widely cited – panda example, illustrated in Figure 2 [15]. The figure demonstrates that the original image of a panda can be perturbed with some small modification to pixel values, resulting in the classification of a gibbon.

Figure 2
Example of Evasion Attack taken from [15]



As mentioned above, one key difference between evasion and poisoning attacks is the time of attack: test time for evasion attacks and training time for poisoning attacks. As a result, poisoning attacks seek to corrupt a model while it is being trained, typically with the goal of having an overall poorly performing classifier. Evasion attacks, on the other hand, are crafted to misclassify specific test samples while performing well on other samples.

The adversarial sample is typically generated by applying some perturbation to the original data. Ideally, the adversarial sample should look indistinguishable from the original data, while being misclassified by the model. The goal of misclassification is achieved by applying carefully crafted perturbations to the sample, where the perturbation is designed such that an appropriate distance metric – between the original and the perturbed instance – is minimized, while the loss function of the model is maximized.

The strength of an evasion attack can be described using two metrics. The first metric is the distance between the original and the perturbed sample, computed using the L_P norm as described in Equation 2.1.

$$||x - x^{'}||_P = (\sum_{i=1}^{n} |x_i - x_i^{'}|^P)^{\frac{1}{P}} \tag{2.1}$$

In Equation 2.1, x and $x^{'}$ are the original and perturbed instances, respectively, and n is the dimensionality of x. A small value means that the perturbed sample ($x^{'}$) is very similar to the original (x), while a larger value describes a more heavily perturbed sample. The most commonly used metrics are the L_0, L_2, L_∞ norms. The L_0 norm refers to the number of non zero elements in the difference as shown in Equation 2.2, where we define $0^0 = 0$. When minimizing the L_0 norm, the attack is attempting to modify the fewest number of pixels to create as strong of an attack as possible.

$$||x - x^{'}||_0 = \sum_{i} |x_i - x_i^{'}|^0 \tag{2.2}$$

The L_2 norm measures the Euclidean distance between x and x' as shown in Equation 2.3. The L_2 distance is not sensitive to the changes in individual pixels and instead looks at average mean squared distance across the entire sample. An attack that uses L_2 norm tends to apply smaller perturbations across many pixels.

$$||x - x^{'}||_2 = \sqrt{\sum_{i=1}^{n} |x_i - x'|^2} \tag{2.3}$$

The L_∞ norm measures the maximum distance between any two pixels $||x-x'||_\infty = \max(|x_1 - x_1'|, ..., |x_n - x_n'|)$. The L_∞ distance returns the maximum perturbation for a given pixel across the entire sample, for example, an image.

$$||x - x^{'}||_{\infty} = max(x - x^{'}) \tag{2.4}$$

Adversarial evasion attacks attempt to create a sample which is misclassified by the model while looking almost identical to the original sample. To keep the adversarial sample as close to the original as possible, many attacks use the L_P norm to restrict the perturbation. Evasion attacks differ in optimization problems, but most attacks minimize the L_P norm to restrict the amount of perturbation which can be applied. So although the attack functions differ, the attacks work with the same underlying similarity metric, for this reason it is vital to evaluate defenses against L_P norm based attacks. Some newer evasion attacks move away from utilizing the same similarity metric of an L_P norm. These attacks develop different similarity metric which lead to the development of structurally different attack points. As the non-L_P norm attacks often succeed in evading the classifier, these samples must also be evaluated.

The second metric used to describe evasion attacks is the attack success rate. The attack success rate is essentially the rate at which the adversarial attack achieves its goal. In evasion attacks, the goal can be an untargeted attack, or a targeted attack. In an untargeted attack, the attacker's only objective is to force the sample simply to be misclassified as any class other than the correct class. More specifically, let the set (X, Y) contain pairs of data and labels, where Y is the set of all classes. A sample has the label $y_o \in Y$, here y_o is the original (true) label of the sample. In an untargeted attack, the adversary adds some perturbation to the data $x' = x + \epsilon$ to achieve the goal $y_{ut} \in Y \wedge y_{ut} \neq y_o$, where the new label y_{ut} is any class other than the true label Y_o. In a targeted attack, the attacker also introduces a perturbation $x' = x + \epsilon$, but in this case the attacker seeks a misclassification of y_t selected by the attacker to be any specific label other than the original y_o. In this work, we develop BSD against untargeted attacks, as such attacks result in stronger adversarial examples. It is important to note that in some attacks the "untargeted" version is the process of performing a targeted attack against all classes and selecting the class that resulted in misclassification with the lowest distance.

2.1.4 Autoencoders

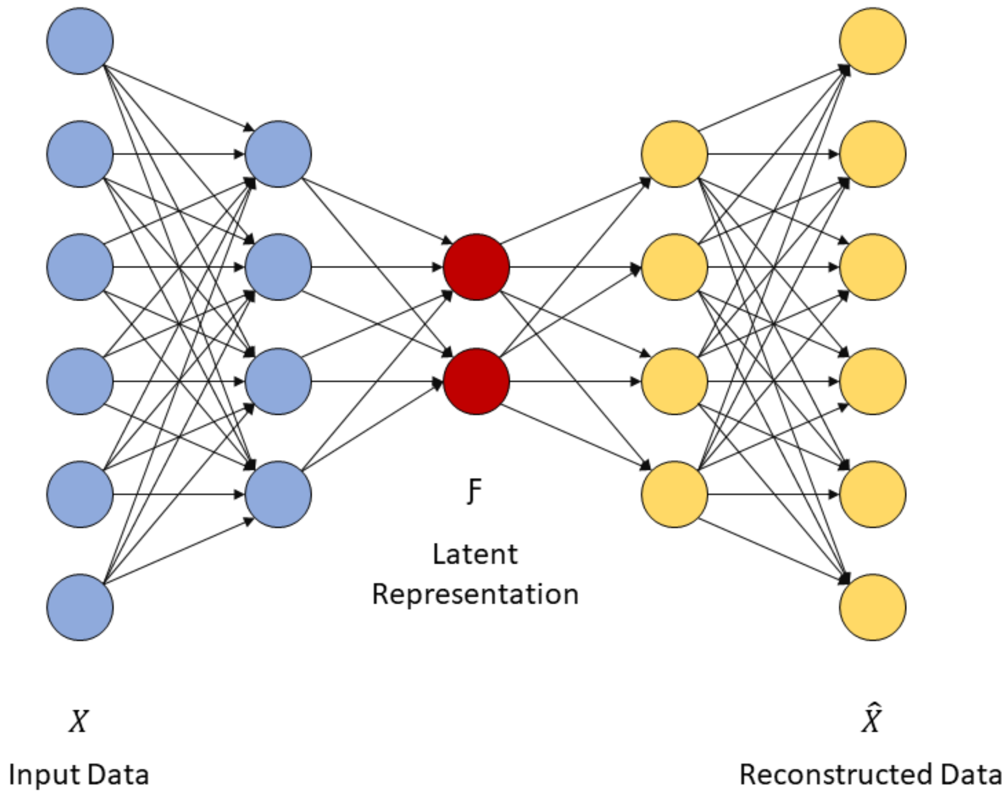
Autoencoders, in the form of an encoder-decoder combination, were first introduced by DeMers in 1992 [9]. Autoencoders seek to learn the representation of a set of data, and reconstruct said data. Autoencoders have many practical applications such as noise removal, data generation, dimensionality reduction, and image compression. Autoencoders are a specific neural network structure, which seeks to take data X and reduce it to a latent representation $\mathcal{F}$ using an encoder as shown in Equation 2.5. This latent representation is then passed through decoding layers that results in the reconstructed sample $\hat{X}$, which is shown in Equation 2.6.

$$\phi : X \rightarrow \mathcal{F} \tag{2.5}$$

$$\psi : \mathcal{F} \rightarrow \hat{X} \tag{2.6}$$

Figure 3 shows the process of passing data X through the encoding layers shown in blue, generating the latent representation $\mathcal{F}$ shown in red, which is then passed to the decoding layers shown in yellow, yielding the reconstructed data $\hat{X}$.

Figure 3
Visual Depiction of Autoencoder



The encoder layers are standard, full interconnected feed forward neural network layers, with each encoding layer having fewer nodes than the previous layer. The reduction in nodes over the layers is what allows the data to be compressed into a reduced latent representation $\mathcal{F}$. For example, if the data originally has 100 features appearing at the input layer, there may be a layer of 75 nodes followed by a layer of 50 nodes in the encoding section of the autoencoder. Therefore, the latent space is then constructed with 50 values as opposed to the 100 features used to originally describe the data. The encoder can be made of any number of hidden layers. After the data are encoded into its latent representation $\mathcal{F}$, they are decoded into the reconstructed sample $\hat{x}$. The decoder also consists of several hidden layers, with increasing number of nodes from one layer to the next. The increase in the number of nodes is what allows for the compressed latent data to be expanded back to the original size.

The goal of the autoencoder is to learn the weights needed to compress x into some latent space $\mathcal{F}$ and generate a reconstructed sample $\hat{x}$ to be as close to x as possible. To keep $\hat{x}$ as close as possible to x, the autoencoder is trained to minimize some reconstruction error, or difference, between x and $\hat{x}$, where this difference can be computed using any L_P norm. In this work, we use the L2 norm. Equation 2.7 shows $\hat{x}$ can be substituted with the forward propagation of sample x through the encoder and decoder. Replacing $\hat{x}$ allows one to define the loss in terms of the weights of the encoder, decoder and sample x. In Equation 2.7 W and b refer to the weights and biases of the encoder, whereas W' and b' refer to the weights and biases of the decoder. With the loss in terms of X and the weights, the network can be trained using backpropagation. In summary, an autoencoder is a specific neural network architecture trained to minimize the difference between input and output, and contains a latent representation of the data.

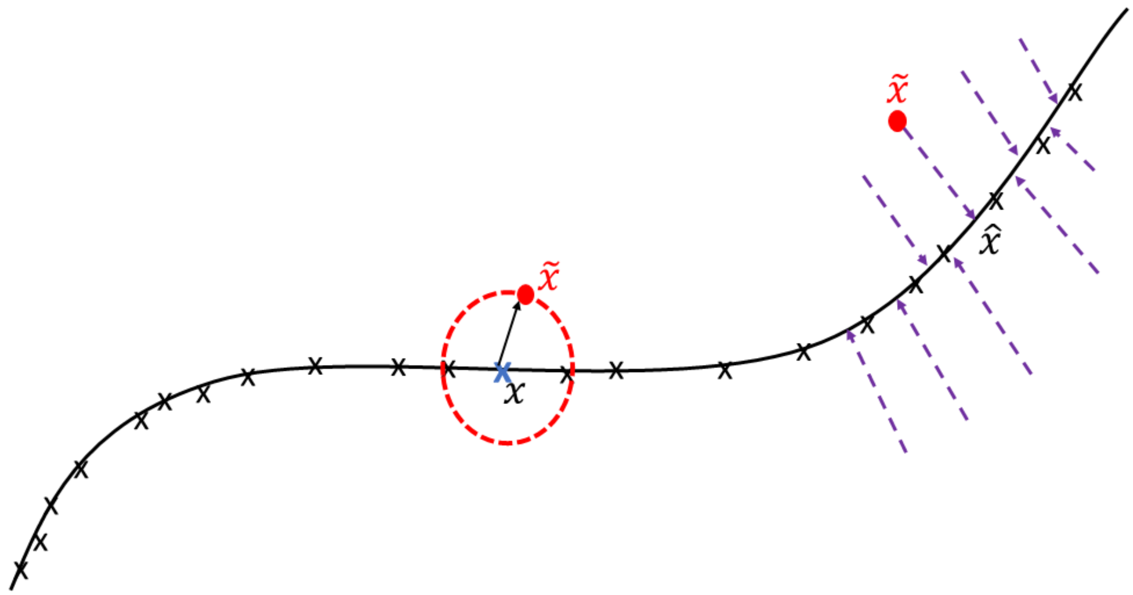
$$L(x, \hat{x}) = ||x - \hat{x}||_p = ||x - \sigma(W'(\sigma(Wx + b)) + b')||_p \tag{2.7}$$

2.1.5 Denoising Autoencoder

Earliest discussions on denoising autoencoders were provided by Vincent *et al.* in 2008 [38]. Denoising autoencoders take in a corrupted input $\widetilde{x}$, and pass it through the autoencoder resulting in the auto-encoded sample $\hat{x}$. The goal is to obtain an $\hat{x}$ that is very close to the original uncorrupted sample x with respect to some distance metric. The corruption used by the authors involved setting random inputs to zero. Denoising autoencoders are not restricted to a single form of corruption, however, and have been shown to be effective with respect to Gaussian noise, uniform noise, and randomly setting values to zero [38]. An important behavior of denoising autoencoders – which is employed in our broad spectrum defense – is the ability of the autoencoder to project data back towards the manifold. Figure 4 shows the black **x** marks laid on the data manifold and the corrupted $\widetilde{x}$ that lie further away from the manifold. The blue **x** in Figure 4 shows a data point laying on the data manifold. To train the autoencoder, the sample is corrupted by adding noise to the data, this noise is bound by some radius from the original point (blue **x**), demonstrated by the dotted red circle in Figure 4. When the $\widetilde{x}$ sample is passed through the denoising autoencoder, it is brought closer to the data manifold, as indicated with the dotted purple arrows in Figure 4. $\widetilde{x}$ can be moved along the dotted purple arrow towards the data manifold. Denoising autoencoders are trained with the goal of removing noise from the data. Since adversarial data can be seen as adversarial noise, denoising autoencoders may be able to mitigate the impact of an adversarial example. We will discuss how we use denoising autoencoders in the development of the broad spectrum defense (BSD) in Chapter 4.

Figure 4

Demonstration of Denoising Autoencoder adapted from [38]



Chapter 3

Related Work: Evasion Attack and Defense Strategies

In order to develop a broad spectrum defense against evasion attacks, it is important to understand the underlying premise and general characteristics of evasion attacks. Therefore, in this chapter, we first review a wide spectrum of evasion attacks that have been developed over the last several years. This chapter will explore the strengths and weaknesses of each attack and the conditions under which they can be most effective. We then look at some of the more effective defensive strategies developed against evasion attacks and review their strengths and shortcomings.

3.1 Terminology and Expressions

Much of the existing work on evasion attacks introduce and use their preferred nomenclature in referring to certain parameters and variables, even though similar parameters might have been defined by others for different attack strategies. In order to make their comparison easier to follow, we first present a unified set of nomenclature for variables and parameters as summarized below in Table 1.

Table 1

Table of Variables

Variable	Name	Description
x	Original data point	Unmodified test sample
y	Original label	Unmodified label of the sample
x'	Adversarial sample	Sample with added adversarial perturbation
t	Target class	Class adversary desires for misclassification
I	Iterations	Number of iterations to run a loop
k	Confidence parameter	Confidence of selected adversarial example
θ	Network parameters	Parameters of neural network classifier
$Z(x)$	Logits resulting from x	Output of last layer of network before softmax
η	Perturbation	Changes added to image by adversary
α	Scaling factor	Parameter which scales perturbation
β	Elastic net weighting	Weighting between L_1 and L_2 norms
ϵ	Perturbation bound	Upper bound to restrict perturbation
f	Neural network classifier	Neural network used for classification
$f(x)$	Softmax output	Softmax output values of sample x
$\hat{x}$	Autoencoded version of x	The result of passing x through the autoencoder
C	Number of classes	Number of output classes and nodes
M	Number of features	Number of features or pixels in an image
Υ	Maximum distortion	Number of pixels acceptable to modify image
$\mathcal{F}$	Flow field	Flow field defined over pixels of an image

3.2 Evasion Attacks

We now review some of the most common and well-established evasion attack strategies. Many of these attacks are so-called *gradient-based* attacks, as the attack strategy is based on minimizing (or maximizing) a cost function along its gradient. For each attack strategy, we describe the underlying premise, its algorithmic pseudo-code, as well as its strengths or shortcomings as appropriate. Note that an attack's relative utility is often related to its date of conception and more recent attacks tend to generate stronger adversarial examples.

In all cases, the attacker creates the attack point x' as a perturbation to a genuine test data sample in the form of $x' = x + \eta$, where η is the perturbation.

Attack samples were generated with the help of the Adversarial Robustness Toolbox [32].

3.2.1 Fast Gradient Sign Method (FGSM)

The Fast Gradient Sign Method (FGSM) is a simple yet effective attack method developed by Ian Goodfellow *et al.*. [15] FGSM utilizes the gradient of the cost function used in the training of the classifier model (typically a neural network) to create adversarial examples. Equation 3.1 shows the general formulation of FGSM, where η is the perturbation added to the genuine test sample x, x' is the resulting attack point, y is the correct label, and α is a factor for scaling the amount of perturbation added to the data. $\boldsymbol{J}(\theta, x, y)$ is then the cost function used to train the neural network, where θ are model parameters. We note that, when used in a typical image data, η perturbs all pixels that contribute to increasing the cost.

$$\eta = \alpha \ \mathrm{sign}(\nabla_x \boldsymbol{J}(\theta, x, y)) \tag{3.1}$$

FGSM is very sensitive to the α value as this scales the magnitude of the perturbation: lower values typically result in a low success rate (weak attack), whereas values that are too high result in obvious and easy to detect attack points. For this reason, one should attempt the attack with a set of various α values and verify that the resulting perturbed samples are highly similar to the original unperturbed samples. The FGSM attack computes the gradient of the cost function given the sample and label, the $sign$ is then taken and multiplied with α so the perturbation η will be α or $-\alpha$. This η value is then added to the pixel to create the modified adversarial example. Algorithm 1 demonstrates the FGSM attack in two steps, in one step the gradient of the cost function is computed and η is set to $\pm\alpha$, finally, η is then applied to the image by adding it to the pixel value. The FGSM attack was one of the first evasion attacks developed. The attack modifies the images pixels by increasing and decreasing each pixel by a value of η causing an increase in the cost function eventually leading to misclassification. As the FGSM attack can be implemented with low cost and it quickly generates adversarial samples, it can be used to rapidly evaluate the baseline efficacy of any adversarial defense.

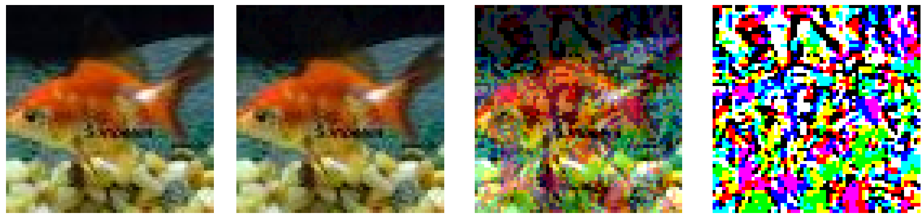
Algorithm 1 FGSM Attack Algorithm
J cost function
θ model parameters
α step size
η adversarial perturbation

procedure FGSM($\alpha, \boldsymbol{J}(\theta, x, y)$)
    $\eta = \alpha$ sign$(\nabla_x \boldsymbol{J}(\theta, x, y))$
    $x' = x + \eta$
    return x'

Figure 5 shows an example image from the TinyImageNet dataset with the FGSM attack applied with different α values. Note that as α is increased the perturbation becomes more obvious.

Figure 5

FGSM Attack Applied to an Example Image from TinyImageNet



(a) *Original Image* **(b)** α *= 0.03* **(c)** α *= 0.3* **(d)** α *= 3*

3.2.2 Jacobian Saliency Map Attack (JSMA)

Another gradient-based method is the Jacobian Saliency Map Attack (JSMA) [33]. In this attack, saliency maps are constructed on the inputs of the neural network based on the forward derivatives, which reveal enough information to craft strong adversarial examples. The saliency map shows how much each pixel contributes towards the prediction of the class for a given image. Knowing the pixels that contribute to a classification, the attack can then modify these pixels forcing the image to be misclassified. The JSMA attack is an L_0 attack, which attempts to minimize the number of modified elements, or pixels.

The first step in JSMA is to compute the forward derivative of the network with respect to the sample x as shown in Equation 3.2. In this equation x_i refers to the i^{th} pixel or feature of image x, and j refers to the j^{th} hidden layer of the network (f), up to the total number of hidden layers represented by N. The number of pixels range from 1 to M. In JSMA the derivative of the network is taken directly to determine the contribution of the input pixels on the output classification.

$$\nabla f(x) = \frac{\partial f_j(x)}{\partial x_i}_{\; i:1,\ldots,M; j:1,\ldots,N} \tag{3.2}$$

Next, a saliency map – as shown in Equation 3.3 – is computed using the gradient for every input feature and the target class. This means that a matrix of size $N \times C$ is needed, where N is the number of pixels in an image and C is the number of classes. For each element of this matrix, the forward derivatives of the network with respect to x are calculated recursively. This forward propagation of the gradient demonstrates the input components' contribution to the output classification. Algorithm 2 shows in line 5 that the pixel with the maximum value i_{max} in the saliency map S is selected for modification.

$$S(x,t)[i] = \begin{cases} 0, \text{ if } \frac{\partial f_t(x)}{\partial x_i} < 0 \text{ or } \sum_{j \neq t} \frac{\partial f_j(x)}{\partial x_i} > 0 \\ (\frac{\partial f_t(x)}{\partial x_i})|\sum_{j \neq t} \frac{\partial f_j(x)}{\partial x_i}|, \text{ otherwise} \end{cases} \tag{3.3}$$

The attacker's goal is to increase the probability of misclassification of the target class, and decreasing the weighting of any other classification. If the gradient of a pixel given the target class is negative, then the saliency map is assigned a zero for that feature. Additionally, if the pixel's gradient with respect to any other class is positive, the saliency value is also set to zero as we would not want to increase the change of any classification other than the target. However, if the gradient of the pixel given the target class is positive, this positive gradient value is assigned to the sailency map for the given pixel, then by changing this pixel we can steer the classification towards the target class. All of these values generate the saliency map shown in Equation 3.3. The saliency map demonstrates what pixels should be increased or decreased to classify the sample as the target class. The last step involves a user defined parameter, Υ, which controls how many pixels can be modified so the image remains recognizable to humans.

For an untargeted attack, the attack is performed for each given class with the exception of the true class and the class resulting in the lowest L_0 norm is selected.

JSMA is a computationally expensive attack, requiring large amount of memory to run on larger datasets. For this reason we are unable to perform JSMA on the large neural network used for classification of ImageNet.

Algorithm 2 JSMA Attack Algorithm

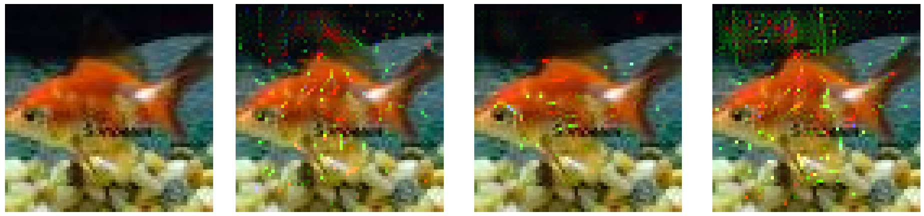
f neural network
σ magnitude of change to introduce to the feature
t target class
Υ maximum distortion
δ_x number of features currently changed from the original

Input: $x, t, f, \Upsilon, \sigma$
$x' \leftarrow x$
$\eta = 0$
while $f(x') \neq t$ and $||\eta|| < \Upsilon$ **do**
    Generate saliency_map S using Equation 3.3
    Modify x'_{imax} by σ s.t. $i_{max} = \text{argmax}_i S(x', t)[i]$
    $\eta \leftarrow x' - x$
return x'

Figure 6 shows an example image from the TinyImageNet dataset with the JSMA attack applied with different σ values. As σ is increased, the perturbations become more obvious.

Figure 6

JSMA Attack Applied to an Example Image from TinyImageNet



(a) *Original Image* **(b)** σ = *0.001* **(c)** σ = *0.01* **(d)** σ = *0.1*

3.2.3 Projected Gradient Decent (PGD)

PGD is the process of performing multiple update steps descending the negative loss function [27]. By performing this optimization, the loss is increased leading to a sample's misclassification. PGD attacks show that by taking multiple FGSM steps and projecting the sample back to some ϵ ball, attack point placement can be optimized resulting in a more powerful sample in exchange for additional computational cost. In Equation 3.4, x^t is the current placement of the sample, x^{t+1} refers to the updated attack point after step t, P is the projection of the sample back towards the ϵ ball around the original sample, α is the step size, and $\nabla_x \boldsymbol{J}(\theta, x, y)$ is the gradient of the cost as seen in FGSM.

$$[h]x^{t+1} = P(x^t + \alpha\ sign(\nabla_x \boldsymbol{J}(\theta, x, y))) \tag{3.4}$$

If only one step of PGD is performed, then the attack equates to FGSM. The power of PGD comes from taking multiple steps ascending the loss function and projecting back towards the original sample. The projection step of PGD prevents the optimization from generating samples that are perceptually different from the original target sample. PGD outperforms FGSM and results in stronger adversarial samples, and hence a stronger attack. PGD's stronger performance comes at a higher computational expense, but PGD does scale to large architectures, unlike JSMA that requires a much larger amount of available memory.

Algorithm 3 PGD Attack Algorithm

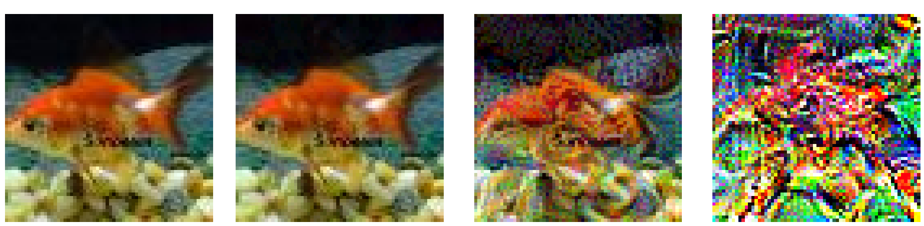
I number of iterations
α step size of attack
$\nabla_x \boldsymbol{J}(\theta, x, y)$ gradient of the cost
ϵ the radius of the ball which attack points are projected to
S a random location inside the ϵ ball around the sample

Input: $\alpha, \epsilon, \boldsymbol{J}(\theta, x, y), S$
$x' \leftarrow S$
for $i = 0$ to I **do**
    $x' = x' + \alpha$ sign$(\nabla_{x'} \boldsymbol{J}(\theta, x', y))$
    $x' = \max(\min(x', \epsilon), -\epsilon)$
return x'

Figure 7 shows an example image from the TinyImageNet dataset with the PGD attack applied with different ϵ values. Note that the smaller ϵ values result in very minor changes to the original image while the largest ϵ value results in a very obvious adversarial example.

Figure 7

PGD Attack Applied to an Example Image from TinyImageNet



(a) *Original Image* **(b)** ϵ = *0.03* **(c)** ϵ = *0.3* **(d)** ϵ = *3*

3.2.4 DeepFool

DeepFool is a more recent attack strategy that finds the decision boundary for each class, after finding the decision boundaries it projects the sample onto the closest classes decision boundary and then the sample is pushed past the boundary forcing a missclassification [30]. Algorithm 4 demonstrates the process for creating a DeepFool adversarial example. The DeepFool attack works by iterating until the classification of the original sample x differs from the classification of the generated adversarial sample x'. For ever class y' in the set of all classes C besides the correct class, the difference in gradients are computed and the difference in logit values are computed. For example line 4 of Algorithm 4 shows $w'_k \leftarrow \nabla F_k(x') - \nabla F_{k(x)}(x')$, here the difference in gradients between the adversarial sample and a given adversarial class are subtracted from the gradients of the original sample and original class. This same process is repeated in line 5 for the output values associated with each class. The class that has the closest logit values divided by gradients is then declared $\hat{l}$, which is the class with the closest decision boundary. Next, we calculate r_i, the minimal vector to project x onto the closest decision boundary computed in line 6. This computed r_i value is added to x' and the loop is repeated. When the sample is misclassified the loop is finished and the sample is finally generated by generating the final perturbation η by summing over all r values and the final adversarial sample $x + \eta$ is returned.

The DeepFool attack takes a different approach from the previously mentioned attack as instead of performing gradient decent on the loss function, the DeepFool attack finds the closest decision boundary, perturbs the sample towards that boundary and recursively repeats the process until the sample is misclassified. The DeepFool method generates strong adversarial examples with moderate computational complexity and scales well to large architectures. The DeepFool attack is considered a very strong attack and is very useful in bench-marking adversarial defenses.

Algorithm 4 DeepFool Attack Algorithm
f model
x original sample
w' stores the difference in gradients
f' store the difference in logit values
C set of all classes
y Original class
$\hat{l}$ closest class to the original
I Maximum Number of iterations

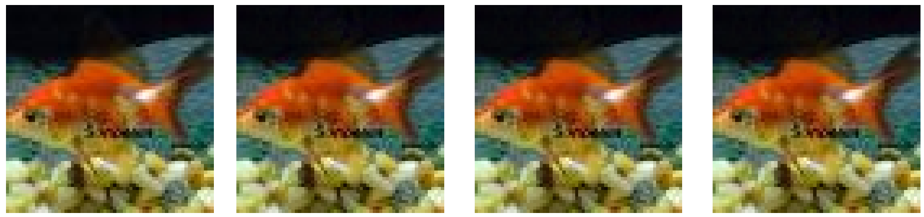
Input: x, f

$x' \leftarrow x, i \leftarrow 0$
while $\text{argmax}\, f(x') = y$ and $i < I$ **do**
  for $y' \in C$ where $y' \neq y$ **do**
    $w'_{y'} \leftarrow \nabla f_{y'}(x') - \nabla f_y(x)$
    $f'_{y'} \leftarrow f_{y'}(x') - f_y(x)$
  $\hat{l} \leftarrow \underset{y' \neq y}{\text{argmin}} \frac{|f'_{y'}|}{||w'_{y'}||^2}$
  $r_i \leftarrow \frac{|f'_{\hat{l}}|}{||w'_{\hat{l}}||^2} w'_{\hat{l}}$
  $x' \leftarrow x' + r_i$
  $i \leftarrow i + 1$
$\eta = \sum_i r_i$
return $x + \eta$

Figure 8 shows an example image from the TinyImageNet dataset with the DeepFool attack applied with different maximum iteration (I) values. In this case the parameter of maximum iteration has no impact on the sample generated as a valid adversarial example is found before 10 iterations so the samples show with 10, 100, and 1000 are the same.

Figure 8

DeepFool Attack Applied to an Example Image from TinyImageNet



(a) *Original Image* **(b)** *I = 0.03* **(c)** *I = 0.3* **(d)** *I = 3*

3.2.5 Carlini Wagner Attack

The Carlini Wagner (CW) attack attempts to simultaneously solve the min-max problem of minimizing the distance from the original sample to the adversarial example while also maximizing the likelihood that the sample is misclassified [5]. The CW attack differs from other attacks in one critical way: instead of simply constraining the attack to a certain perturbation, the amount of perturbation is dynamically solved to achieve an optimal trade off between attack strength and detectability. The distance metric utilized can be any of the L_0, L_2, or L_∞ norms. The Carlini-Wagner attack solves the multifaceted optimization problem of minimizing the distance between the adversarial sample and the true sample while also forcing the classification of the sample to the target class. One of the most sensitive parameters in the CW attack is the confidence k. A confidence value of zero returns adversarial samples that have a small difference in logit values between the most probable class and second most probable class. This small difference between output values can be interpreted as the classifier having low confidence that the prediction is correct. For example, there may only be a one percent difference between output nodes of a network. A higher confidence (k) value results in a larger gap between the most probable and second most probable classes. For example if a confidence of 0 is selected, and the original class of the attack sample is class-3, the sample may be classified as class-4 with the corresponding (highest) output being .5 and the second highest output (being that of class-3) with a value of .4. In this case the model would report the desired class, but with low "confidence". In certain critical classification operations, individuals analyze the outputs of a neural network and in this case if the attacker uses a low k (confidence value) it could tip off a knowledgeable end user.

Equation 3.5 shows the formulation of the Carlini Wagner L_2 attack. In this formulation, the $||\frac{1}{2}(\tanh(\omega) + 1) - x||_2^2$ term minimizes the difference between the adversarial sample and the original sample. This distance ensures the created adversarial sample is highly similar to the original sample. The $c \cdot F(\frac{1}{2}(\tanh(\omega) + 1))$ term describes the strength of the adversarial example. The c term is found by performing a line search over values of c to find the best solution to minimizing Equation 3.5. The c term encourages the solver to minimize both portions of Equation 3.5 simultaneously instead of optimizing over each term sequentially. Equation 3.5 represents the adversarial example x' as $\frac{1}{2}(tanh(\omega) + 1)$ where ω is the variable we are solving. The change of variables is used to ensure that the new adversarial example will have values between 0 and 1. This is ensured as $-1 \leq \tanh(\omega) \leq 1$, so it follows that $0 \leq x + \eta \leq 1$. The adversarial sample x' is equal to $x + \eta$ and also equates to $\tanh(\omega)$. The function F is applied in Equation 3.5 and is explained in Equation 3.6. Equation 3.6 is selecting the maximum logit value $Z(x')_i$ where the selected class i cannot be that of the target class t. This condition ensures the sample is classified as the target class. The logit value of the desired target class $Z(x')_t$ is subtracted from the largest logit value $Z(x')_i$. The difference in logit values is then compared to the confidence k, if the difference in logit values is lower than $-k$, the value returned is $-k$. When optimizing this attack, the parameter k encourages the solver to find an adversary x' which is classified as t with a high confidence, this higher confidence often comes with a larger distance from the original sample so this trade off must be considered.

$$\text{minimize } \left(||\frac{1}{2}(\tanh(\omega)+1) - x||_2^2 + c \cdot F(\frac{1}{2}(\tanh(\omega)+1)) \right) \tag{3.5}$$

$$F(x') = \max\{-k, \max_{i \neq t}[Z(x')_i] - Z(x')_t\} \tag{3.6}$$

The Carlini Wagner attack currently stands as one of the strongest evasion attacks. Due to the simultaneous minimization of the distance and maximization of the strength of the samples, the attack generates perceptually similar adversarial examples that cause the classifier to incorrectly classify the sample with very high confidence. The added power of the Carlini Wagner attack comes with added computational overhead compared to a simple attack like FGSM, but the added computational expense is often warranted to generate such quality and effective adversarial examples.

Algorithm 5 Carlini Wagner Attack Algorithm

x original sample
k confidence parameter
t target class
Z Logit Values (Pre softmax)
I Iterations
η perturbation

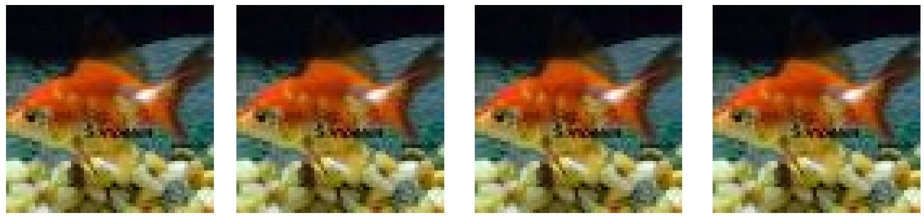
Input: x, k, t, I, Z

```
for j in Binary Search Steps do
    Select c by minimize (||η||_2^2 + c · f(x + η)))
    1/2(tanh(ω) + 1) = x' = x + η
    for i = 0 to I do
        minimize (||1/2(tanh(ω) + 1) − x||_2^2 + c · F(1/2(tanh(ω) + 1)))
        Perform optimization step with Adam and updating ω
x' = tanh(ω)
return x'
```

Figure 9 shows a sample image from the TinyImageNet dataset with the CW attack applied with different confidence (k) values. In this case the images look extremely similar, but all of the images generated are imperceptibly different. Recall that higher k values encourage the solver to generate examples with larger differences on the output of the neural network while smaller k values allow smaller differences between the outputs of the network. In this case, all of the generated examples look identical to the human eye.

Figure 9

Carlini Wagner Attack Applied to a Sample Image from TinyImageNet



(a) *Original Image* **(b)** k *= 0* **(c)** k *= 10* **(d)** k *= 100*

3.2.6 Elastic Net Attack

Elastic net attack is a special case of the Carlini Wagner attack [6]. Equation 3.7 shows the objective function that is minimized to find an elastic net attack adversarial sample. The $c \cdot \max\{-k, \max_{i \neq t}[Z(x')_i] - Z(x')_t\}$ term of this equation is identical to the Carlini Wagner attack, in the $\beta||x' - x||_1 + ||x' - x||_2^2$ term instead of just minimizing the L2 distance, the elastic net representation is used where the L_1 and L_2 norms are minimized and the weighting is decided by β. In the elastic net attack, the change of variable approach used by Carlini no longer works due to the addition of the L_1 norm.

$$g(x) = c \cdot \max\{-k, \max_{i \neq t}[Z(x')_i] - Z(x')_t\} + \beta||x' - x||_1 + ||x' - x||_2^2 \tag{3.7}$$

$$x' = S_\beta(x' - \alpha \nabla g(x')) \tag{3.8}$$

$$[S_\beta(z)]_i = \begin{cases} \min\{z_i - \beta, 1\}, & \text{if } z_i - x_{0i} > \beta \\ x_{0i}, & \text{if } |z_i - x_{0i}| \leq \beta \\ \max\{z_i + \beta, 0\}, & \text{if } z_i - x_{0i} < -\beta \end{cases} \tag{3.9}$$

For this reason, the iterative shrinkage-thresholding algorithm (ISTA) is used. ISTA, described by Equation 3.9, performs an additional step of shrinking and thresholding at each iteration. The elastic net attack can generate adversarial examples stronger than the Carlini Wagner attack as it considers both the L_1 and L_2 norms, while also shrinking and thresholding the perturbation $(x' - \alpha\nabla g(x'))$ at every step. The ISTA process is similar to the projection back to the ϵ ball explained in the PGD attack. Utilizing both L_1 and L_2 norms to constrain the attack resulting in a sample with low total perturbation over the image (L_1) and low average perturbation over the image (L_2). The β parameter shown in Equation 3.9 is used shrink the deviation from the original pixel value if the deviation is greater than β and does not change the pixel value if the deviation is less than β. The pesudocode of elastic net attack is listed in Algorithm 6. Lines 5 to 7 of the algorithm demonstrate that the best adversarial example for each original sample is selected as a sample that is misclassified with the lowest distortion metrics. The distortion metric used for the EAD attack can be a combination of L_1 and L_2 (Elastic-Net) or the L_1 distortion relative to x.

The Elastic Net attack was developed as an extension of the Carlini Wagner attack so it brings all of the strengths of the Carlini Wagner attack such as generating high quality imperceptible adversarial examples. The Elastic Net attack also introduces another parameter, β; if this parameter is tuned correctly, the Elastic Net attack can outperform the Carlini Wagner attack. However, if this parameter is not tuned correctly, the Elastic Net attack can generate weaker attack samples. If one wishes to generate the strongest adversarial attack with the Elastic Net attack, they should ensure that the β parameter is tuned correctly by performing a search over a range of β values to find the value that works best for your classifier and data. If one is constrained by computational cost, however, it would be best to use the Carlini Wagner attack instead of the Elastic Net attack.

Algorithm 6 Elastic Net Attack Algorithm
β Elastic Net Norm Weighting
x original sample
α step size
I maximum number of iterations

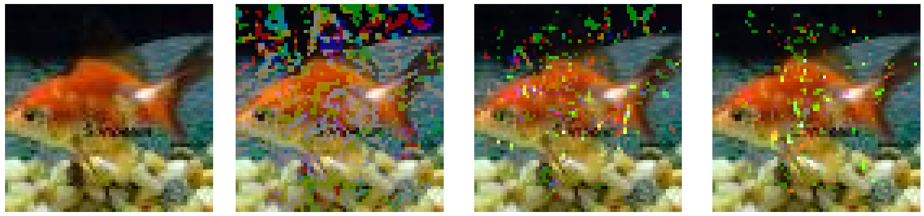
Input: x, β, α, I

Initialization: $x^{(0)} = y^{(0)} = x_0$
for $j = 0$ to $I - 1$ **do**
  $x^{(j+1)} = S_\beta(y^{(j)} - \alpha\nabla g(y^{(k)}))$
  $y^{(j+1)} = x^{(j+1)} + \frac{j}{j+3}(x^{(j+1)} - x^{(j)})$
$X = \{\text{argmax}(f(x^{(j)})) \neq \text{argmax}(f(x))\}_{j=1}^{I}$
for $x' \in X$ **do**
  select x' with lowest Elastic Net or L_1 distance from x
return x'

Figure 10 shows a sample image from the TinyImageNet dataset with the EAD attack applied with different confidence (β) values. As the β value is increased the L_1 norm is weighted more heavily and the L_2 norm is given less weight. This is demonstrated by Figure 10 as sub-image *b* shows modified values over many pixels and sub-image *d* with a β of 1 shows stronger modifications made to fewer pixels.

Figure 10

Elastic Net Attack Applied to a Sample Image from TinyImageNet



(a) *Original Image* **(b)** *β = 0* **(c)** *β = 0.5* **(d)** *β = 1*

3.2.7 Spatial Attack

One example of an attack that does not utilize an L_P norm is the Spatial Attack [39]. All of the previously mentioned attacks use some form of an L_P norm to constrain or limit the perturbation to make sure that the generated adversarial sample looks similar to the original. The spatial attack is the first attack presented that is limited to image data, all previously mentioned attacks work on any form of data. To create spatial adversarial examples, the authors begin by defining a per-pixel flow field $\mathcal{F}$ to create adversarial sample x' using pixels from the input x. The flow field defines a mapping of how the image will be perturbed. Let $x'^{(i)}$ denote the *i*-th pixel of the image with flow field coordinates $(u'^{(i)}, v'^{(i)})$. The amount of displacement in each image dimension is optimized using the flow vector $\mathcal{F}_i := (\Delta u^{(i)}, \Delta v^{(i)})$. This flow vector relates the adversarial pixel $x'^{(i)}$ to the corresponding pixel in the original image $x^{(i)}$. If the flow vector is solved around for $x^{(i)}$ then, $(u^{(i)}, v^{(i)}) = (u'^{(i)}+\Delta u^{(i)}, v'^{(i)}+\Delta v^{(i)})$. The values of $(u^{(i)}, v^{(i)})$ exist on a continuous spectrum so the differentiable bilinear interpolation is used to create the adversarial image.

Now $x'^{(i)}$ can be calculated as:

$$x'^{(i)} = \sum_{q \in N(u^{(i)}, v^{(i)})} x^{(q)}(1 - |u^{(i)} - u^{(q)}|)(1 - |v^{(i)} - v^{(q)}|) \tag{3.10}$$

$N(u^{(i)}, v^{(i)})$ contains the four neighboring pixels from $(u^{(i)}, v^{(i)})$ corresponding to the top-left, top-right, bottom-left, and bottom-right pixels. Equation 3.10 then yields the adversarial image when calculated over all pixels.

Other works rely on the use of an L_P norm to constrain the perturbation, in this work a new regularization loss, the L_{flow} loss is used to minimize the local distortion within the image. Given an image x, the optimal flow field $\mathcal{F}^*$ is obtained by minimizing:

$$\mathcal{F}^* = \operatorname*{argmin}_{\mathcal{F}}(L_{adv}(x, \mathcal{F}) + \tau L_{flow}(\mathcal{F})) \tag{3.11}$$

Here, the two terms represent the attackers two goals. The first term L_{adv} encourages the adversarial examples to be misclassified by the target classifier. The second term L_{flow} exists to minimize the local distortion. The τ parameter exists to allow tuning of the trade off between the two terms. This τ term resembles the c term from the Carlini Wagner attack as both terms control the strength vs detectability trade off.

The L_{adv} term is constructed to represent the goal, in a targeted attack, $\operatorname{argmax} f(x') = t$ where t is the target class which does not equal the ground truth label y. To reiterate, in an untargeted attack, the target can be any label besides the ground truth. The objective function for L_{adv} follows the implementation by Carlini and Wagner.

$$L_{adv}(x, \mathcal{F}) = \max(\max_{i \neq t} [Z(x')_i] - Z(x')_t, -k) \tag{3.12}$$

In Equation 3.12 $Z(x)$ represents the logit values on input x, $Z(x)_i$ represents the *i*-th element of the logit vector also known as the logit value of the *i*-th class. k is used to define the confidence level of the attack. The k parameter controls the gap between confidences of the attack class and all other classes. At low k values, the difference between any other class and the target class must be at least a value of k. As the k value increases the targeted class will result in a higher logit value while the other classes' logit values decrease. This larger separation between logit values implies that the classifier believe the adversarial example with high confidence, which is where the confidence parameter k gets its name.

L_{flow} is computed using the sum of spacial movement distance between any two adjacent pixels. Given a pixel p and its neighbors $N(p)$, L_{flow} is defined as:

$$L_{flow}(\mathcal{F}) = \sum_{p}^{all\ pixels} \sum_{q \in N(p)} \sqrt{||\Delta u^{(p)} - \Delta u^{(q)}||_2^2 + ||\Delta v^{(p)} - \Delta v^{(q)}||_2^2} \tag{3.13}$$

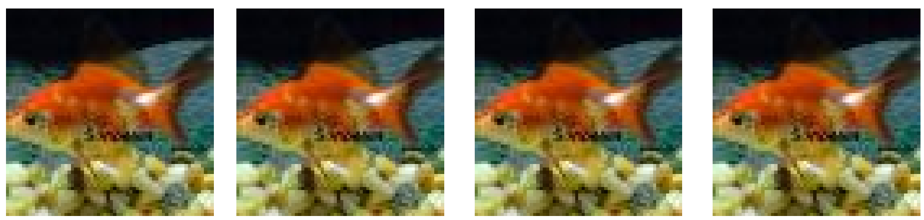
By minimizing Equation 3.13 the perceptual quality can be maintained as adjacent pixels tend to move in similar directions and distances. Equation 3.13 quantifies the difference between pixels and their neighbors in the flow field. The difference is found by summing the squares of $\Delta u^{(p)} - \Delta u^{(q)}$ and $\Delta v^{(p)} - \Delta v^{(q)}$ and taking the square root. The process is repeated for every pixel and its corresponding neighbors.

The Spatial Attack is capable of generating strong adversarial examples using a method substantially different from traditional L_P norm attacks. In developing a broad spectrum defense, one must verify the defense offers improved robustness against non-L_P norm attacks. Testing against non-L_P norm attacks reveals information about how well the defense can generalize to unseen attacks. The strength of the Spatial Attack comes at the cost of additional computational overhead as the loss function now involves computing statistics over all pixels in the image in addition to the loss function used by Carlini and Wagner.

Figure 11 shows the original test image and the result of applying the spatial attack with different τ values. All images shown appear identical demonstrating that varying the τ parameters does not significantly modify the perception of the generated adversarial sample.

Figure 11

Spatial Attack Applied to an Example Image from TinyImageNet



(a) *Original Image* **(b)** τ = *1* **(c)** τ = *10* **(d)** τ = *100*

3.2.8 Shadow Attack

The Shadow Attack is another form of a non-L_P norm attack. In the Shadow Attack, the adversarial example is constrained by three different penalty terms [13]. Similar to the Spatial attack, the constraints are designed assuming that the input is an image. Equation 3.14 introduces the equation used to find the η or the perturbation added to the original sample to yield the adversarial sample $x' = x + \eta$. Equation 3.14 shows that the loss is to be maximized while minimizing the three terms controlled by λ_c, λ_{tv}, and λ_s. The λ values ensure that aspects of the adversarial image remain very close to the original image yielding a strong sample. Applying the aforementioned constraints force the sample to look indistinguishable from the original while forcing an increase in the loss causing the misclassification of the sample.

$$\max_{\eta}[L(\theta, x + \eta) - \lambda_c C(\eta) - \lambda_{tv} TV(\eta) - \lambda_s Dissim(\eta)] \tag{3.14}$$

The $TV(\eta)$ term constrains the total variation across the image, encouraging the smoothness within the image. Following the terminology used above, η describes the perturbation added to the data. The total variation term is defined in Equation 3.15, the anisotropic total variation describes the difference in vertical and horizontal components of the perturbation.

$$TV(\eta_{i,j}) = \text{anisotropic-TV}(\eta_{i,j})^2 = (\sum_{i,j}|\eta_{i+1,j} - \eta_{i,j}| + |\eta_{i,j+1} - \eta_{i,j}|)^2 \tag{3.15}$$

The $C(\eta)$ constrains the change in mean of each color channel. $C(\eta)$ is defined in Equation 3.16. In this equation, the element wise absolute value is taken of each channel and the average of the absolute values is computed.

$$C(\eta) = ||Avg(|\eta_R|), Avg(|\eta_G|), Avg(|\eta_B|)||_2^2 \tag{3.16}$$

The last penalty term, $Dissim(\eta)$ focuses on maintaining the color balance of the image. This term will keep the perturbations to the red, green, and blue channels similar so the resulting perturbation to the image will minimally disturb the color balance and will result in a darker or lighter pixel. There are two forms of the $Dissim(\eta)$ term, the first method generates a single array to represent all color channels, and is called 1-channel, and in this case $Dissim(\eta)$ = 0 as all channels change together. The other case is where all three channels can be modified, in this case the $Dissim(\eta)$ is defined in Equation 3.17.

$$Dissim(\eta) = ||(\eta_R - \eta_G)^2, (\eta_R - \delta_B)^2, (\eta_G - \eta_B)^2|| \tag{3.17}$$

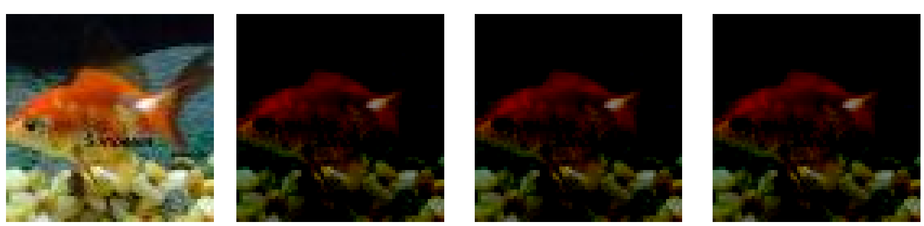
Equation 3.17 shows that the dissimilarity in the 3-channel scenario is quantified as the squared difference between each pair of color channels. This metric encourages the perturbations applied to each color channel to be similar to the perturbations applied to other channels.

It is important to note that the Shadow attack does not directly constrain the L_P norm of the generated adversarial samples. The Shadow attack ensures the adversarial image looks similar to the original with out explicitly restricting the L_P norm. Due to the potentially high L_P norm values, the attack can be effective against defenses that only evaluate robustness against L_P norm attacks.

Figure 12 shows an example image from the TinyImageNet dataset with the Shadow attack applied with different total variation (λ_{tv}) weightings. In this case the parameter of total variation did not significantly change the output image and in all cases the images generated appeared to be a darker version of the original (hence the name, *shadow attack*).

Figure 12

Shadow Attack Applied to an Example Image from TinyImageNet



(a) *Original Image* **(b)** λ_{tv} = *0.3* **(c)** λ_{tv} = *0.6* **(d)** λ_{tv} = *0.9*

3.3 Existing Defense Theory

Existing defenses vary greatly in their approaches. Some defenses attempt to utilize information about attacks to defend the feature space of the model. Other defenses try to flag and remove potential adversarial samples. One commonality is the trade off between model security and impact on non-adversarial data, a defense that can correctly identify a majority of adversarial samples comes at the expense of incorrectly identifying benign samples as adversarial. One example of this trade off is the tuning of evasion detectors. Detectors trained to detect evasion attacks attempt to remove adversarial samples, these defenses do not need to return a classification if the sample is deemed adversarial. Some detectors can detect these adversaries at a high rate, but the high detection rate comes at the expense of falsely detecting some non-adversarial data as well.

3.3.1 Adversarial Training

Adversarial training (AT) is a defensive method that relies on crafting attack samples and training the classifier on these attack samples. The central premise of this defense is that by training on the attack samples, the defender can hope to define and learn the feature space around these adversarial samples and – when one appears – the model will return the proper classification. One potential problem with adversarial training is that it can be considered as a reactive approach, where the defender is utilizing information about existing and known attacks to craft a defense. Since adversarial training is a reactive approach, it is unlikely that the defense would generalize to new attacks, as it was trained specifically with one type of attack. Goodfellow *et al.* argue that the adversarial samples used to train the defense appropriately approximate the adversarial space[15]. The adversarial space is incredibly large and it is unlikely that adversarial training would provide adequate coverage over every "nook and cranny" in the entire adversarial space. The AT defense has shown strong performance against samples that were slightly perturbed using strong attacks such as the Carlini Wagner attack, but they can be vulnerable to fundamentally different attacks such as Spatial and Shadow attacks. To address ATs vulnerability to Spatial attacks, Zhang and Wang proposed AT while incorporating spatial attack samples [41]. Their paper also showed that in exchange for the added robustness, overall performance decreased. Zhang and Wang demonstrate that adversarial training is a reactive defense that relies on knowledge of existing attacks to make the defense robust.

3.3.2 Detectors

A different approach to adversarial example defenses utilizes detectors. Detector-based defenses aim to remove adversarial points as opposed to adversarial training, which attempts to return the correct classification for adversarial samples. As the detectors do not have to return a classification, the idea is to detect anomalies that are suspected to be adversarial. Detectors also offer a simpler solution to adversarial examples as they do not have to give a classification. For some attack samples with larger perturbations, it can be very difficult to recover the correct classification. One example of such situation would an image of random values. It is impossible to know the correct classification, but in this case, a detector based defense can simply reject the sample.

There are several detector-based defenses with varying levels of success. Typically, the performances of detector-based approaches are dataset specific – some performing well only on more simplistic datasets such as MNIST or CIFAR10, whereas others only perform well against datasets such as ImageNet. Another problem with some detector-based defenses is they may work well against L_P attacks such as the Carlini Wagner attack, but they may not work well against a spatial attack or shadow attack that is not constrained on the L_P norm. A comparison of different detector methods is listed in Table 2. This table shows that most detector-based defenses are trained using existing adversarial examples. Using adversarial examples to train the detectors has the consequence of constraining the defense to perform well only on similar style attacks, limiting the defenses' ability to generalize to other existing or future attacks. Each defense in Table 2 was also evaluated against the Carlini Wagner attack, whose results are also displayed in Table 2. Defenses selected for comparison in this work had to meet the criteria of performing well on Carlini Wagner attack and not requiring adversarial data to train. These criteria resulted in the comparison of Feature Squeezing, MagNet, and the (proposed) broad spectrum defense.

Table 2

Comparison of Detector Based Defenses

Defense	**Effective on Carlini Wagner**	**Trained Using Adversarial Data**
Hendrycks's [17]	x	✓
Li's [22]	x	x
Grosse's [16]	✓	✓
Gong's [14]	✓	✓
Bhagoji's [2]	x	✓
Feinman's [12]	x	✓
Metzen's [29]	x	✓
RBF-SVM [25]	✓	✓
Feature Squeezing (FS) [40]	✓	x
Noise Reduction (NR) [23]	✓	✓
Steganalysis [24]	✓	✓
MagNet [28]	✓	x
Broad Spectrum Defense	✓	x

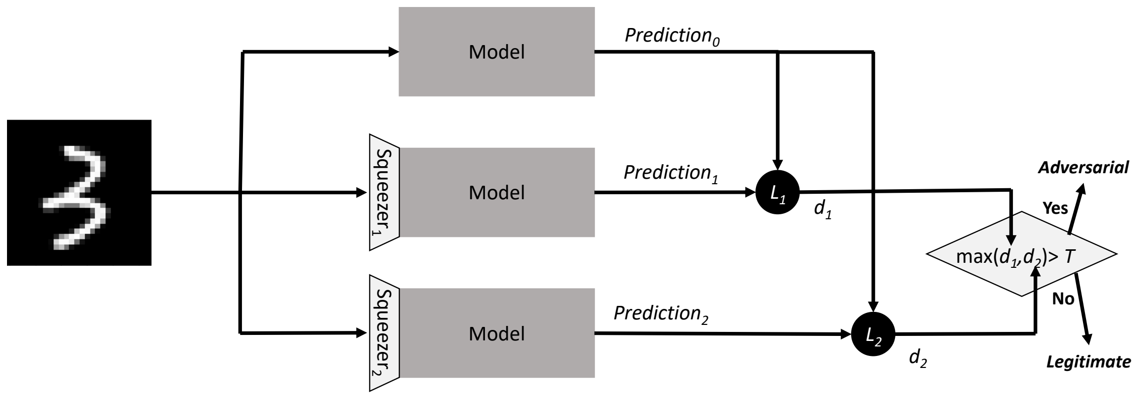
3.3.2.a Feature Squeezing. Feature Squeezing is a defense with a demonstrated strong performance against all datasets for L_0 and L_2 norm attacks [40]. Feature Squeezing is not trained using adversarial examples, and therefore no inherent bias is introduced when deploying this defense. Feature squeezing works by applying filters to images to create a new image, with the hope that the new image will remove adversarial perturbations. The authors decide to apply two styles of filters, a color depth filter and spatial filters. The color depth filter simply reduces the number of bits available for all colors, for example a gray-scale image is 8-bit and contains 256 values for each pixel, a reduced version could be a 4-bit image that contains 16 values for each pixel. Color images by default are 24-bit (8 bit over 3 channels), which contain about 16 million different colors. Bit depth reduction can bring this down to 12-bit or 4096 colors.

Feature squeezing also uses a variety of spatial filters such as Gaussian smoothing, mean smoothing, or median smoothing. These methods are considered local methods as they make use of nearby pixels to smooth the pixel in question. These filters are applied pixel by pixel across the whole image. Other spatial filters utilized include non-local methods that smooth over a broader area. For example, non-local methods find similar patches over an area of the image and replace the center of the patches with the average of the similar patches.

The Feature Squeezing defense, depicted in Figure 13, uses the squeezers to preprocess the data, generating additional predictions for each sample. For each filter, the outputs of the neural network are taken and subtracted from the outputs of the neural network on the original sample. If the squeezed images output minus the original images output exceeds the threshold, the sample is rejected as adversarial.

Figure 13

Feature Squeezing Block Diagram adapted from [40]

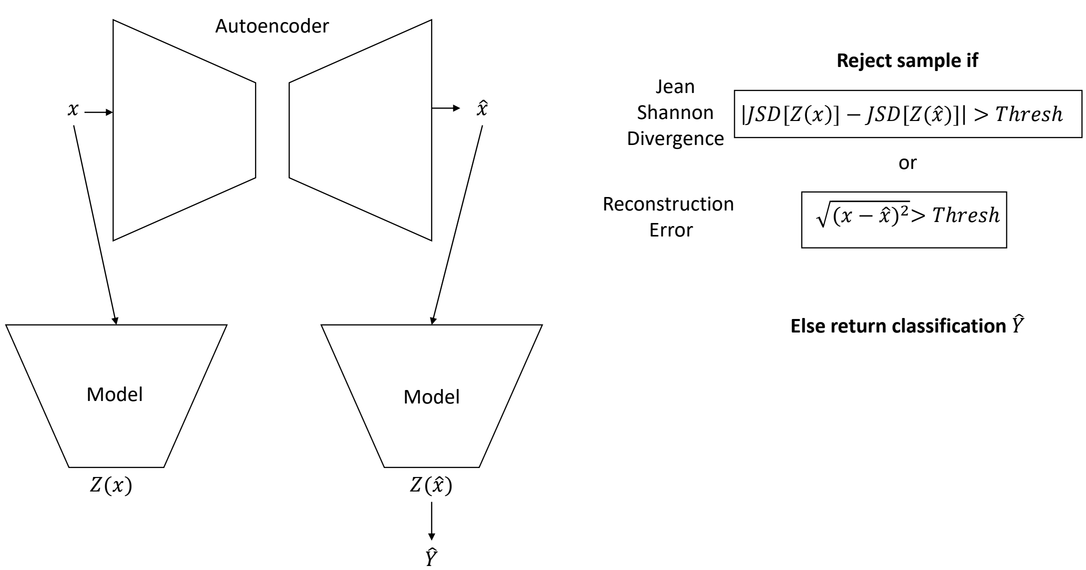


In describing their feature squeezing approach, the authors demonstrate that when the correct combination of filters are selected for any given dataset, feature squeezing is robust to L_2 and L_0 attacks [40]. Unfortunately, they also show that even with the best possible filter selection, this defense only recognizes 20.8% and 55% of Fast Gradient Sign Method (FGSM) and Basic Iterative Method (BIM) samples on CIFAR-10 (L_∞ attacks). FGSM and BIM are not computationally expensive, and are often some of the first attacks attempted. On the ImageNet dataset, the FGSM and BIM based adversarial samples once again evade the defense with rates of 43% and 64.4%. Feature squeezing demonstrates strong performance on other attacks such as the Carlini Wagner attack and Jacobian Saliency Map Attack (JSMA). Feature Squeezing makes an excellent benchmark defense as it defends against the Carlini Wagner attack across a variety of datasets and does not require adversarial data to create the defense.

3.3.2.b MagNet: A Dual Detector Approach. MagNet is a detector based defense that seeks to either remove or reform adversarial examples. MagNet seeks to detect and remove adversarial examples without training the model on adversarial data. As the defense does not have access to adversarial data, it must attempt to approximate the boundary between the adversarial and legitimate data. This defense passes a test sample x through an autoencoder yielding $\hat{x}$, the auto-encoded version of x. The reconstruction error between $\hat{x}$ and x is then calculated and compared to a threshold. This reconstruction error can also be described as the distance between $\hat{x}$ and x. A legitimate sample would be expected to have a small reconstruction error as the autoencoder was trained to minimize the distance between the input and outputs of the autoencoder. An adversarial sample may exist in a subspace of the autoencoder that is not well defined and can return a larger reconstruction error. The second detector used in MagNet is called the Jean Shannon Divergence (JSD) detector. The JSD detector calculates the Jean Shannon Divergence between $F(x)$ and $F(\hat{x})$, where $F(x)$ refers to the softmax values at the output of the network. If the difference in JSD values exceeds a threshold then the sample is deemed adversarial and is removed. The threshold is defined as a percentage, and the percentage selected corresponds to the amount of acceptable false positive detections. The JSD detector is added to remove samples with low reconstruction errors that induce a large change in the outputs of the network. The samples found by the JSD detector have an unusually high change in the outputs of the network, this usually implies that the sample is adversarial causing this massive shift in output values. If the sample passes both detectors, then the resulting classification $\hat{Y}$ is considered the predicted classification.

Figure 14 shows a block diagram of the MagNet defense. The diagram shows the the autoencoder, and the model (also known as the classifier). The original sample x is passed through the model yielding the classification $Z(x)$. x is also passed through the autoencoder yielding the auto-encoded version of x, $\hat{x}$. The auto-encoded sample is also passed through the same model yielding the classification $Z(\hat{X})$. The diagram demonstrates using the sample, an autoencoder, and softmax outputs of the model, adversarial samples can be detected. If a sample is not detected as adversarial by the JSD or reconstruction error detectors then the final classification is returned as $Z(\hat{x})$.

Figure 14
MagNet Block Diagram



MagNet and the detectors used in MagNet are explored in detail in chapter 4.

3.3.2.c Adaptive Noise Reduction. A different detector-based defense is Adaptive Noise Reduction, a defense that modifies an original image by applying transformations, and then compares the resulting images' classification to the classification of the original [23]. The idea here is that if an image is adversarial by applying some filters, some image close to the original can be recovered and the classification for the adversarial and original will not match. In the adaptive noise reduction defense different filters are applied to the image, this generates new slightly different images. Adaptive noise reduction states that if the classification of these new images differs from the original image, the sample is likely adversarial. The rationale is that neural networks should be robust to small changes within an image and the network would be expected to retrieve the correct classification. Adversarial samples are caused by minor imperceptible changes to the image and the filters can disturb the delicately crafted adversarial sample and force a change in classification.

Adaptive noise reduction has many hyperparameters that need to be selected to fit the problem. To select these hyperparameters, the authors use a validation set to generate FGSM attack points and tune the parameters to these attack points. This tuning process has the unintentional consequence of biasing the defense towards detecting FGSM generated adversarial samples and/or samples that behave similar to those generated by FGSM. This shortcoming was demonstrated when the authors found that the defense fails against simple attacks such as the L0 (Jacobian Saliency Map Attack) JSMA. JSMA samples are relatively easy to craft and they bypass the adaptive noise reduction defense at ease - a significant problem for the defense. Another issue is that the epsilons used were hand picked to be very small numbers. Recall from Section 3.2.2 that the epsilon values control how much the attack perturbs the sample. The theory is that if the epsilon is too large the resulting attack point (adversarial sample) can be recognized by a human, but adaptive noise reduction shows that as epsilon increases the detection rate decreases. This defense does not have any way of handling larger epsilon values that may be slightly visible but still difficult for a human to detect. Larger epsilon attack samples may still evade a humans perception and should be evaluated against. In this paper the authors only demonstrate performance on the Carlini Wagner attack from $k = 0$ to $k = 4$, whereas other works have shown results for $k = 0$ to $k = 40$. The authors have shown that as the confidence is increased to 4 the detection rate of this method diminishes. Adaptive noise reduction does not include any mechanisms to handle these attack samples with larger perturbations so it would be expected that the downward trend would continue towards $k = 40$. This paper also states that adaptive noise reduction does not perform well against larger perturbations such as those induced by L_0 attacks such as the JSMA. Based on performance against L_0 attacks, it appears that adaptive noise reduction would also suffer in performance against spatial and shadow attacks as they tend to introduce even larger perturbations than L_0 attacks.

3.4 Shortcomings of Current Approaches

3.4.1 Certified Robustness Within an ϵ Ball

One major limitation of existing adversarial defenses is their inability to work across a variety of attacks with a variety of ϵ values. Many of the more recently proposed defenses attempt to argue some level of certifiable robustness. Certifiable robustness shows a defense's resilience to attack within some ϵ ball of the original image. In practice, certifiable robustness helps to defend against stronger attacks, such as the Carlini Wagner attack, but as soon as the epsilon parameter of the attack is increased the defense no longer offers any claims on robustness. Certifiable robustness relies on that epsilon ball, but some attacks do not utilize distance metrics when crafting points. One example of such an attack is the shadow attack. In shadow attack, the loss is maximized but the constraint is hand crafted to keep the attack image similar to the original [13]. This local distortion can create rather high ϵ values, but these attack samples can remain indistinguishable to humans. Ghiasi *et al.* demonstrate that by using the shadow attack, they can even create samples that evade defenses that were supposed to be robust within the ϵ ball.

3.4.2 Training on Adversarial Data

Another issue with many existing defenses is the requirement that the training data for the defense must include adversarial examples. Using adversarial examples during training allows for high accuracy for detecting similar types of adversarial examples, that use similar attacks. When a defense strategy relies on adversarial data, the defender is explicitly showing (and training) the defense the kind of examples the adversary may use. This logic is fundamentally flawed as it is impossible to predict a priori and subsequently represent the entire adversarial space. One simple example of the vastness of the adversarial space is the non-L_P norm attacks. These attacks are structurally different than L_P norm based attacks. Non-L_P norm attacks bypass defenses trained with L_P norm attack data achieving high success rates.

A more effective approach to create a robust defense that can generalize is to define the region of true data, rather than that of the adversarial data. Any data that exist outside of the "true data" region can then be deemed adversarial.

3.4.3 Inability to Generalize to Other Datasets

It is perhaps unreasonable to expect a given defense to work on all datasets. In particular, certain defenses that work on simpler datasets may not work on more complex datasets. For example, some simple defenses work very well on the MNIST dataset, but do not scale to larger datasets such as ImageNet and CIFAR-100 [16] [14] [17] [2] [12]. However, perhaps more curiously, the opposite problem also occurs: for example, a detector based defense using Steganalysis (the process of detecting concealed messages inside another message) was proposed by Liu *et al.* [24]. This defense performs very well on ImageNet, but the defense fails on the simpler / smaller MNIST and CIFAR-10 datasets, as they cannot provide enough samples to construct the features required for the defense.

Chapter 4

A Broad Spectrum Defense Against Evasion Adversarial Attacks

The goal of the proposed defense is to detect and remove adversarial attack points, and return the correct classification for the remaining data. A desired characteristic for our proposed defense is the ability to generalize across a variety of models, datasets, and attacks. We strive to create a truly robust defense capable of generalizing to existing and future attacks. While the performance against all attacks may not be perfect, we seek to develop a defense that offers meaningful improvement over the lack of a defense in all cases. As such, we refer to our approach as broad spectrum defense.

4.1 Inspiration for Proposed Defense

Our inspiration for the broad spectrum defense (BSD) comes from *MagNet: a Two-Pronged Defense against Adversarial Examples* [28]. MagNet has demonstrated itself as a strong defense against evasion attacks on the MNIST and CIFAR10 datasets. The formulation of the MagNet defense also allows for a great deal of flexibility as the defense can scale to larger datasets and networks. The added overhead comes in the form of training a more robust autoencoder. The performance of MagNet against a variety of attacks has been evaluated, and it has been found that the performance suffers on non-L_P norm attacks [37]. Carlini and Wagner also demonstrated that the MagNet defense can be easily compromised if the attacker is aware that the defense is applied [4] .

The *adaptive noise reduction* defense also uses a concept that has been explored in this work [23]. The adaptive noise reduction defense applies filters to the images and compares the classification of the unfiltered image to the classification of the filtered image. If the two classifications do not match, then the sample is rejected as adversarial. Adaptive noise reduction shows strong performance on samples attacked with small perturbations, but the defense is weak against larger perturbations.

4.1.1 Training Autoencoders

As stated in chapter 3, MagNet requires a trained denoising autoencoder. The amount of noise used in training the autoencoder is an important parameter as too much noise added will result in a fuzzy output making it difficult for the classifier to generalize. If this occurs the accuracy on the auto-encoded test set will be significantly lower than the accuracy on the unmodified test set. A different problem occurs if very little noise is added: in this case the autoencoder will learn weights that result in a near perfect duplicate of the original image. If the autoencoder learns a 1 to 1 mapping of the input, it is not changing the image at all, and is not contributing to the overall defense performance. There exists a Goldilocks zone – so to speak – where the performance on the test data is not dramatically decreased, while the autoencoder is modifying the image enough to detect adversarial samples with the Jean Shannon Divergence and reconstruction detectors.

4.1.2 Setting Thresholds

Both detectors in MagNet relies on a threshold. The threshold determines the tradeoff between false positives (improperly detecting a real sample as adversarial) and false negatives (missing an adversarial sample). The threshold, defined as β, is the percentage of the training set that is acceptable as false positives. A β of 0.0 sets the thresholds to the largest value such that no training samples will be detected as adversarial. A low β sets the thresholds to result in small amount of false positives, but may not be as sensitive to capturing adversarial data. As β is increased, the sensitivity of the defense is also increased at the cost of a lower specificity. Another important note is that the β is defined per detector so a β of 0.01 will detect 1% of the training data with the reconstruction error detector and will detected 1% of the data with the JSD detector. In other words, although each detector detects 1% of the training data, the total false positives can range between 1% to 2% – 1% if all samples are detected by both detectors and they are the same, or 2%, if the samples detected by the two detectors are all different, and between 1% and 2% if in case of a mixture of the two. It is also vital to evaluate the false positive rate of the defense at test time as quantifying the false positive rate is an important aspect when considering applying a defense.

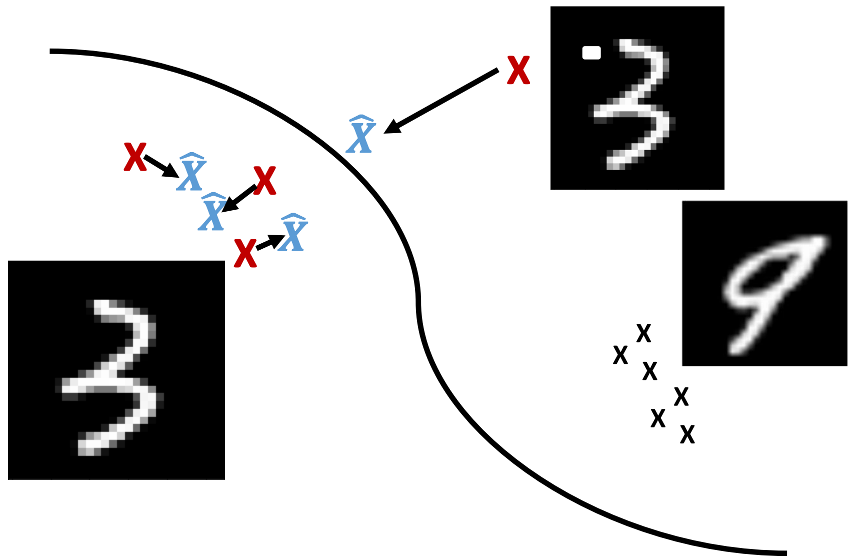
4.1.3 Reconstruction Error Based Detectors

The reconstruction error detectors aim to detect samples that have abnormally high reconstruction errors. More specifically, a sample x is passed through an autoencoder yielding $\hat{x}$. If the L_P norm between x and $\hat{x}$ is greater than a user-defined threshold, then the sample is deemed adversarial and removed from the data-stream. This threshold is established by applying a specified false positive threshold β to the training set. The reconstruction error threshold is selected based on the β value following practices explained above.

The autoencoders are trained so all genuine samples result in the smallest reconstruction error possible. A non-adversarial sample x passed through the autoencoder yields the modified sample $\hat{x}$. The reconstruction error is computed from $|\hat{x} - x|_P$ and is expected to be less than or equal to the threshold as the sample is not adversarial. When an adversarial sample x' passes through the autoencoder, however, the reconstructed image $\hat{x'}$ should modify x' so that $\hat{x'}$ lies closer than x' to the correct classes classification boundary. The reconstruction error on x' should ideally result in a detection. Controlling the false positive rate controls how many samples are detected, which allows for detection of more adversarial examples at the expense of detecting more real samples as false positives.

Figure 15 demonstrates the reconstruction error detector. In this figure, the red X refers to an original sample, the blue $\hat{X}$ refers to the auto-encoded sample and the arrow is the reconstruction error of the sample. The upper left of the figure shows the samples that belong to class 3, the reconstruction error of these samples are small therefore they are not detected as adversarial. In the top right, an adversarial sample (a modified ”3”) can be seen as it is across the decision boundary and far away from the cluster of data points that belong to class 3. The adversarial sample in the top right is passed through the autoencoder resulting in the $\hat{X}$ seen closer the boundary. The arrow pointing from the original sample to the auto-encoded sample is much larger than that of the true data points. This large reconstruction error would result in the detection of this sample.

Figure 15
Demonstration of Reconstruction Error Detector



4.1.4 JSD Detector

The Jean Shannon Divergence (JSD) detector is also used in MagNet. The JSD detector attempts to analyze the outputs of $f(x)$ and $f(\hat{x})$: if the divergence between them is greater than some threshold, the sample is removed as adversarial. The output of $f(x)$ is equal to the softmax of the logits, or the values contained in the last layer of the neural network classifier. Softmax is defined in equation 4.1 where l_i refers to the i^{th} element of the output vector. As the JSD detector requires softmax outputs, this detector is limited to neural networks and classifiers that can provide such probabilistic outputs. The JSD detector looks for a significant difference between the distributions of outputs. This detector works well for detecting an adversarial sample that has dramatically different output values on $f(x)$ and $f(\hat{x})$. For example, if an image of 7 is perturbed and classified as a 9 with high confidence, but after passing the sample through the autoencoder it is classified as a 9 with much lower confidence, the sample would be flagged by the JSD detector. Adversarial examples seek to modify the classification of a target sample, while achieving this goal of modifying the classification the outputs of the classifier must be modified. The JSD detector works to analyze the outputs of the classifier checking the difference between $f(x)$ and $f(\hat{x})$ values against the threshold and detecting the sample if the difference is above the threshold.

$$\text{softmax}(l_i) = \frac{exp(l_i)}{\sum_{j=1}^{n} exp(l_j)} \tag{4.1}$$

Equation 4.2 shows the Jean Shannon Divergence between two distributions P and Q, where $\boldsymbol{D}$ is the KL divergence. The number resulting from this calculation quantifies the difference between the distributions. A smaller JSD implies that the distributions are close together and a larger JSD implies a larger difference between (dissimilar) distributions. The JSD detector relies heavily on the threshold selected, which is selected following the same methodology explained above.

$$JSD(P||Q) = \frac{1}{2}\boldsymbol{D}(P||M) + \frac{1}{2}\boldsymbol{D}(Q||M) \tag{4.2}$$

$$M = \frac{1}{2}(P+Q) \tag{4.3}$$

As Meng et al. have found, the softmax often saturates towards the highest probability class, which heavily skews the distribution of the outputs [28]. To prevent the saturating of the softmax outputs, a temperature variable, T, is added to the softmax, which softens the probabilities and reduces over saturation to the most likely class. This modification is applied to the softmax and is shown in equation 4.4.

$$\text{softmax}(l_i) = \frac{exp(l_i/T)}{\sum_{j=1}^{n} exp(l_j/T)} \tag{4.4}$$

4.2 Proposed Defense: Broad Spectrum Defense (BSD)

The proposed defense does not rely on any existing attacks to define the region in which the adversarial samples may reside. Instead we attempt to bound the region in which genuine data reside, any data found outside of the bounded genuine data would then be considered adversarial data. The proposed defense is constructed by combining different detectors to attempt to cover as much of the adversarial space as possible, while also minimizing the amount of overlap with the distribution of genuine data. The broad spectrum defense builds off of the MagNets' detectors described above.

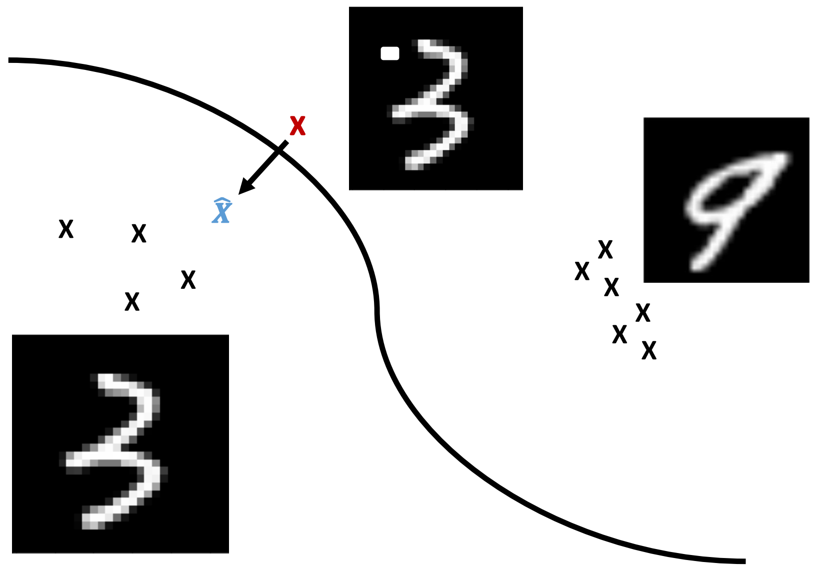
We refer to our proposed approach as broad spectrum defense (BSD), which uses an autoencoder to measure the error between the genuine input sample x and its reconstructed output $\hat{x}$. For genuine data, such an error should be small. A large error, i.e., a large distance between a sample and its reconstructed image (based on training with genuine data) is an indication of an adversarial sample. Hence, such samples with high reconstruction errors are deemed adversarial and removed from the data stream. Such a reconstruction error detector should ideally remove all adversarial examples that yield high reconstruction error when passed through the autoencoder. We retain the use the Jean Shannon divergence (JSD) detector from MagNet as well. As explained above, the JSD detector evaluates the difference between the softmax outputs of the neural network $f(x)$ and $f(\hat{x})$. If the JSD between the two is above a threshold determined from the training data, the sample is deemed adversarial and rejected. As done in MagNet, the JSD detector is applied with two different temperature values, 10 and 40. The softmax values of $f(x)$ and $f(\hat{x})$ are divided by each temperature and a detector is built for each. The temperature values are utilized to help avoid the saturation that occurs near high and low values of the softmax function. We note that these samples are removed in a way that only relies on the true distribution of genuine data points used to train the autoencoder, and hence no existing attacks are used in the training of the BSD. In other words, BSD does not require or rely on the availability of any adversarial data to be used for training.

4.2.1 Class Divergence Detector (Unique to BSD)

The Class Divergence Detector (CDD) is a new detector we add to enhance MagNet and provide additional coverage against a broader spectrum of adversarial attacks. As expected, additional coverage comes at the expense of a higher false positive rate. In chapter 5 we demonstrate this trade off. To further restrict the adversary's attack space, we add an additional detector to MagNet, the Class Divergence Detector (CDD). To explain the CDD we must first review some terminology. During inference time, the classifier - when presented with the input x - produces the label y. The classifier is then presented with $\hat{x}$, or the auto encoded representation of the sample, resulting in $\hat{y}$. The idea of this detector is that by transforming the sample with the autoencoder, the classification of this new data $\hat{x}$ would result in the same classification if the sample is not malicious, or would result in a different classification for adversarial examples. A true sample and its auto-encoded variant are expected to be very close in distance and therefore result in the same classification. An adversarial sample carefully manipulates the data to change the classification, but remain close to the decision boundary. Passing this adversarial sample through the autoencoder may result in a change in classification and this will result in a detection. The CDD does not have any hyperparameters to tune, and is only a function of the classifier and the auto-encoded sample. In the case of a benign sample, the classification from the original sample will equal the classification of the auto-encoded sample, $y == \hat{y}$, and the sample will not be detected. In the case of an adversarial sample, the classification from the original sample will not equal the classification of the auto-encoded sample, $y \neq \hat{y}$, and the sample will be detected. Figure 16 demonstrates how the class divergence detector functions. The red X marks the location of the adversarial example. In this case it will be classified as a "9" as the adversary has forced the sample across the boundary by adding the extra white pixels. After the sample is passed through the autoencoder, it now resides at the blue $\hat{x}$, note that now it is on the side of the decision boundary which will be classified as a "3". The change in classification from the original sample to the auto-encoded sample is what allows for the detection of the data. It is very possible that this sample could have bypassed MagNets reconstruction error detector (the distance auto-encoded sample is close to the adversarial sample), and JSD detector (the output softmax values from $f(\hat{x})$ could be close to $f(x)$). For example, if the sample was classified as "9" with similar probability to "3", then passing it through the autoencoder yielded a classification of "3", the CDD would detect the sample as the class has changed, but the JSD may not detect it if the distributions are highly similar. Through our experiments in Chapter 5 we show that the CDD does in fact capture these samples missed by MagNets detectors.

Figure 16

Demonstration of Class Divergence Detector

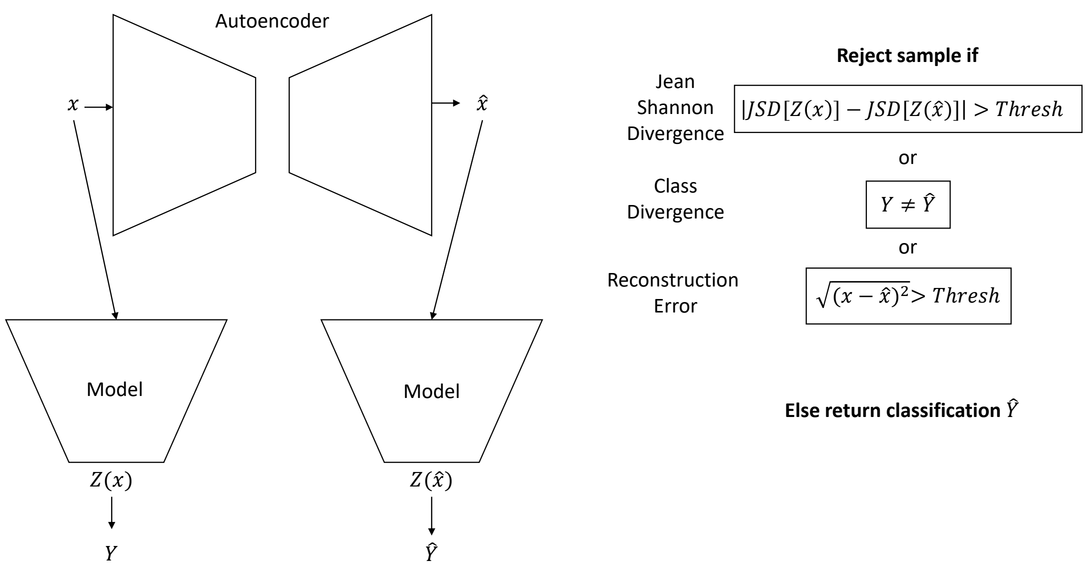


4.2.2 BSD Summary

For an adversarial sample to defeat BSD, the adversarial sample must be bound in reconstruction error, must satisfy $y == \hat{y}$ (the classification of the test sample must match the classification of the auto-encoded sample), and still maintain the JSD between $f(x)$ and $f(\hat{x})$ that is below the thresholds established on the training dataset. The resulting defense is therefore able to detect adversarial examples with very small perturbations, all the way up to examples with large perturbations. Hence the proposed BSD is designed to detect a wide array of adversaries, while also minimizing false positives.

Figure 17 shows a block diagram of the broad spectrum defense. At inference time, the sample x is passed through the autoencoder yielding the auto-encoded sample $\hat{x}$. The sample and its auto-encoded variant are then passed through the model yielding two sets of logits and predictions. Using the logits, predictions, and auto-encoded sample, the test point can be evaluated against the reconstruction error, Jean Shannon divergence, class divergence detectors and, if the sample is not flagged as adversarial, its classification $\hat{y}$ is returned.

Figure 17
Broad Spectrum Defense Block Diagram



Algorithm 7 Broad Spectrum Defense

x original sample
AE autoencoder
Z classifier Logit Values (Pre softmax)
f classifier
β percentage of training data acceptable as false positives

Input: x, AE, Z, f, β

Establish T_{recon} and T_{JSD} from training set using β
$\hat{x} = AE(x)$
$y = f(x)$
$\hat{y} = f(\hat{x})$
$E_{recon} = \sqrt{(x - \hat{x})^2}$
$E_{JSD} = |JSD[Z(x)] - JSD[Z(\hat{x})]|$
if $y == \hat{y}\ \ \&\ \ E_{recon} < T_{recon}\ \ \&\ \ E_{JSD} < T_{JSD}$ **then**
    Return classification y for sample x
else
    Sample is detected as adversarial

Chapter 5

Experiments, Results and Comprehensive Analysis of BSD

5.1 Metrics Used to Evaluate Robustness

It is difficult to evaluate the robustness of a detector-based defense as it is difficult to make any theoretical robustness claims given we are not able to map the entire adversarial attack surface. In some works, the authors discuss robustness with respect to some ϵ ball of a given data point. For example, Ghiasi *et al.* [13] demonstrate that robustness within an ϵ ball results in classifiers vulnerable to larger, non-L_P norm attacks, such as the Shadow attack and Spatial attack. We take a more holistic and empirical approach to evaluate robustness by generating a variety of attacks using different distance metrics and methodologies. Evaluating the proposed defense against a wide variety of existing attacks paints a picture of the utility of this defense. The proposed broad spectrum defense (BSD) does not utilize any information from existing attacks, therefore BSD is not biased towards any attack style, allowing better generalization to other future attacks. The accuracy on adversarial data is reported as the *adversarial accuracy*, the rate at which adversarial samples are detected or classified correctly, as shown in Equation 5.1. In Equation 5.1 $\mathcal{D}$ refers to the set of adversarial examples detected by the defense. This equation computes the success rate of the defense, recall that a detector defense succeeds by detecting and identifying the adversarial sample as adversarial. Also recall that MagNet, BSD, and Feature Squeezing pass the samples through filters and/or autoencoders and then classify these samples. If a sample passes the detector (i.e., not deemed or caught as adversarial), it passes through the filters and then the model for a final classification. This process gives the defense an opportunity to filter / rectify a potential adversarial sample that happened to pass the detector, so that its final classification is in fact correct. If the defense is able to rectify the adversarial example resulting in a correct classification, the defense has succeeded on that sample. The *adversarial accuracy* metric quantifies the success rate of remedying (detecting or correctly classifying) an adversarial sample.

$$Acc_{adv} = \frac{\sum_{i=1}^{N} y_i = argmax[f(x_i)] \text{ or } x_i \in \mathcal{D}}{N} \tag{5.1}$$

5.1.1 True / False Positive Trade Off

When evaluating the performance of a detector, the false positive rate is an important consideration. A false positive occurs when a non-adversarial sample is detected (and hence declared) as adversarial. An optimal defense would have a zero false positive rate while detecting all of the adversarial samples (also known as *true positives*). In reality, a trade off exists in which a zero false positive rate would result in missed adversarial samples (also known as *missed detection (false negative)*). On the other hand, a 100% success rate at detecting adversaries (i.e., zero missed detection) would result in a non-zero false positive detection rate. In this work, the trade off is controlled by the parameter β. The β parameter controls the user's threshold for false positives, i.e., how much of the data is acceptable as false positive detections. This parameter has a valid range of 0 to 1: with $\beta = 0$, the detectors are selected as to not detect any of the data in the training set (as adversarial). With $\beta = 0.01$, the thresholds are established such that 1% of the data in the training set are flagged as adversarial for each detector.

In this work we compute measures of sensitivity and specificity using separate adversarial and non adversarial data. More specifically, we evaluated BSD in two distinct environments to quantify performance. First we evaluate the defense in an environment that consists of only adversarial (positive) test examples. In this case we find the sensitivity, the true positives (detected as adversarial or classified correctly) divided by the number of all test samples (all of which are adversarial) 5.2. The sensitivity reveals how well the defense is able to identify adversarial samples. As described earlier, Equation 5.1 or the adversarial accuracy is a measure of sensitivity as it is measuring the rate at which the defense succeeds on positive samples.

$$Sensitivity = \frac{TP}{TP + FN} \tag{5.2}$$

We then evaluate the defense in an environment consisting of only benign (clean) samples. In this case our experiments reflect the specificity or the rate at which benign images are correctly identified as benign.

$$Specificity = \frac{TN}{TN + FP} \tag{5.3}$$

To demonstrate how many false positives are detected, each defense is evaluated against the benign test set. On such a benign (clean) test set, any *detection* (i.e., any instance flagged as adversarial) is considered as a misclassification - a false positive. Benign accuracy is computed as the ratio of all correctly classified samples that were not detected (as adversarial) to the total number of samples, as demonstrated in equation 5.4. Ideally, the set of detected samples would be empty on the test set (zero false positives). Equation 5.4 captures the performance loss incurred by incorrectly detecting test samples. Note, Equation 5.4 reflects the sensitivity of the defense, or its ability to correctly identify benign samples without flagging them.

$$Acc_{benign} = \frac{\sum_{i=1}^{N}[y_i = argmax[f(x_i)] \text{ and } x_i \notin \mathcal{D}]}{N} \tag{5.4}$$

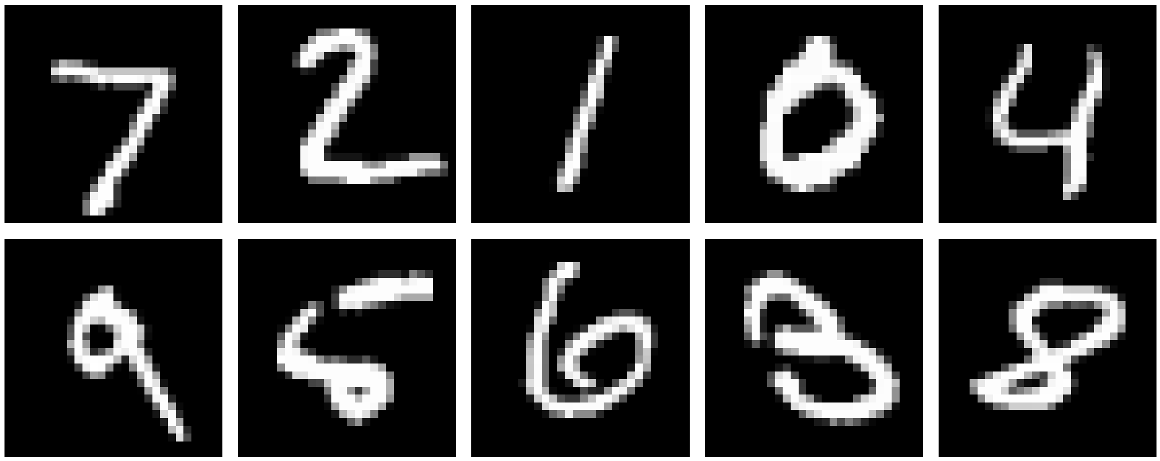
5.2 Datasets Used

Various datasets were utilized to evaluate the performance of BSD. The datasets vary in resolution of images, number of channels, number of classes, and number of images per class. Sample images from each dataset are shown below.

5.2.1 MNIST

The MNIST dataset contains hand written digits 0 to 9, leading to a 10-class classification problem. The MNIST dataset contains 70,000 gray scale 28x28 images, examples are shown in Figure 18. This dataset is split into two subsets: 60,000 images for training and 10,000 images for testing. [21] The MNIST dataset has been used for years to benchmark the performance of simple image classifiers [20], and achieving above 99% testing accuracy on MNIST is possible. MNIST is also widely used in adversarial machine learning as adversarial samples can be generated and visualized rapidly with low computational overhead. Many works have focused on attempting to secure a MNIST classifier against all adversarial samples, but none have succeeded yet [36].

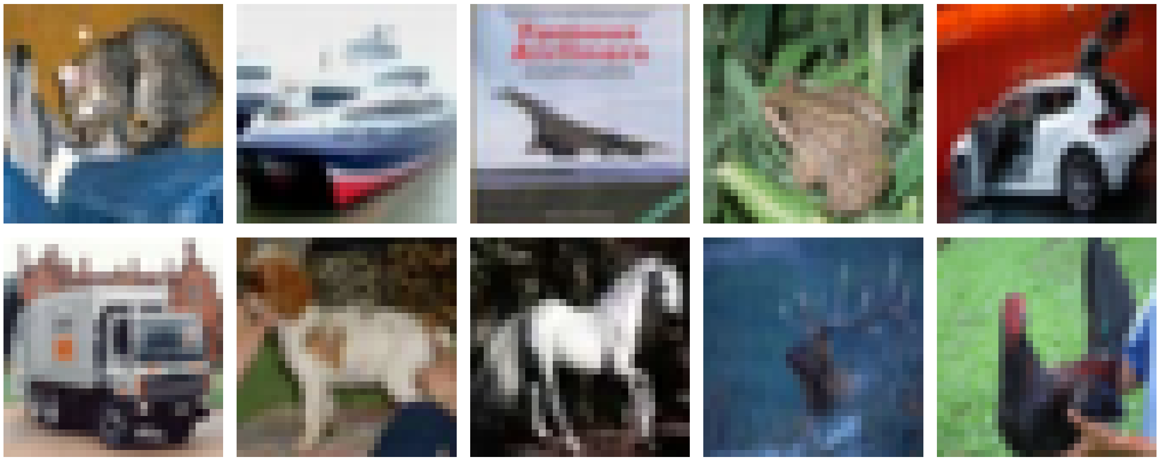
Figure 18
MNIST Dataset



5.2.2 CIFAR10

CIFAR10 dataset contains 32x32 color images of 10 objects: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck [19]. Examples of CIFAR10 images are shown in Figure 19. This data set consists of 50,000 training images and 10,000 test images, that breaks down to 5000 training images per class and 1000 test images per class. Compared to MNIST, CIFAR10 is a color dataset with 3 channels, and CIFAR10 increases the resolution to 32x32 from 28x28. This dataset offers an incremental increase in complexity from MNIST without taking too large of a step.

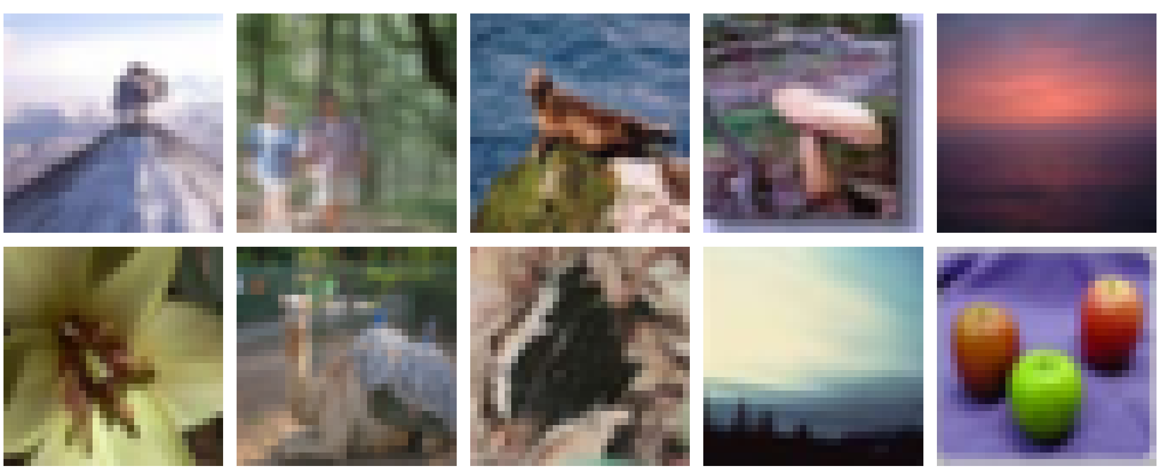
Figure 19
CIFAR10 Dataset



5.2.3 CIFAR100

CIFAR100 is very similar to CIFAR10 as it contains 32x32 color images of 100 classes, representative examples are shown in Figure 20. The dataset still consists of 50,000 training images and 10,000 testing images making it a more difficult task than CIFAR10. In CIFAR100 there are 500 training images per class and 100 test images per class. [19]

Figure 20
CIFAR100 Dataset



5.2.4 TinyImageNet

The TinyImageNet dataset consists of 200 classes with 500 training images each (100,000 total) and 50 test images per class (10,000 total). These images are 64x64 resolution downsampled from the original 256x256 images. A sampling of TinyImageNet are shown in Figure 21. The TinyImageNet problem doubles the number of classes from CIFAR100, and also doubles the resolution of the images, resulting in a more challenging classification problem that requires modern neural network architectures to achieve acceptable results. [35]

Figure 21
TinyImageNet Dataset

5.2.5 ImageNet

ImageNet is one of the largest publicly available datasets with 1,261,406 training images (668 to 3047 per class) and 50,000 images provided for testing (50 per class). The images contained in ImageNet come from 1000 different classes and have a resolution of 256x256. Some representative examples of ImageNet are shown in Figure 22. In this work, we used the Large Scale Visual Recognition Challenge 2012 version of the dataset. The feature space of ImageNet is considerably larger than TinyImageNet as it has five times the number of classes and four times the resolution. ImageNet is used to demonstrate how well classifiers, attacks, and defense can scale towards real life problems. [35]

Figure 22
ImageNet Dataset

5.3 Description of Experiments

5.3.1 Defenses for Comparisons

Feature Squeezing and MagNet are used to compare and contrast the performance of the Broad Spectrum Defense to existing defenses. MagNet was chosen in part because it is a strong defense, and in part is a subset of the BSD, whereas Feature Squeezing was selected because it has been shown to be one of the best performing detector-based defenses. Another reason why we selected these two defenses is because they are not trained with adversarial data. This attribute is critical and necessary to create a generalizable defense. In the paper describing Feature Squeezing, the authors performed a direct comparison and found that MagNet significantly outperformed Feature Squeezing on MNIST, and Feature Squeezing outperformed MagNet on CIFAR-10, indicating a lack of clear winner between the two [40]. Other defense approaches were excluded for reasons such as training with adversarial data, only functioning on specific datasets, or only effective on a small subset of adversarial attacks. Table 2 in Chapter 3 has shown that MagNet, Feature Squeezing and BSD are the only defenses that meet this criteria.

In forming the comparison, MagNet and BSD are both controlled by the same β parameter allowing for a one-to-one comparison. As the FS defense is created without the β parameter, we omit this defense from the benign accuracy evaluation. The benign accuracy evaluation was performed to allow for a comparison between MagNet and BSD, this comparison shows how much additional error is introduced by using the Class Divergence Detector. For the Feature Squeezing defense, we defer to the author provided parameters for each dataset presented in their work. We provide an adversarial performance comparison between FS, MagNet, and BSD.

5.3.2 Adversarial Evaluation

The adversarial robustness for Feature Squeezing, MagNet, and BSD is determined by evaluating each against a variety of attacks. Each attack was crafted using the same dataset used to train the same classifier, with the same attack data passed to each defense. To show how well each defense generalizes, attacks of different norms (L_0, L_2, L_∞) were utilized. In addition to the traditional L_P norm attacks, another set of attacks, that do not utilize the L_p norm (shadow and spatial attacks), were also used. All defenses and attacks were evaluated on MNIST, CIFAR10, CIFAR100, TinyImageNet, and ImageNet datasets. BSD and MagNet were applied with a β value of 0.05, this β value was selected to match the expected False Positive rate given in the Feature Squeezing paper [40]. The aforementioned experiments provides a comprehensive comparison of defenses across a wide spectrum of attacks and environments.

5.4 MNIST

5.4.1 Experimental Setup

On the MNIST dataset high classification accuracies can be achieved using relatively simple convolutional neural network architectures. In these experiments, we used a convolutional neural network with parameters specified in Table 3. In Table 3, the first column specifies the layer type and the second column shows the layer parameters, such as number of filters, number of nodes, or kernel size. We chose this specific network, because it was used in prior works and serves as a benchmark classifier [34] [28].

The MNIST dataset was normalized to zero mean with -1 to 1 bounds. The bounds were selected to line up with the output of the autoencoder: the activation function used on the final layer of the autoencoder was the hyperbolic tangent (tanh) function, whose range is also bounded between -1 to 1.

Table 3

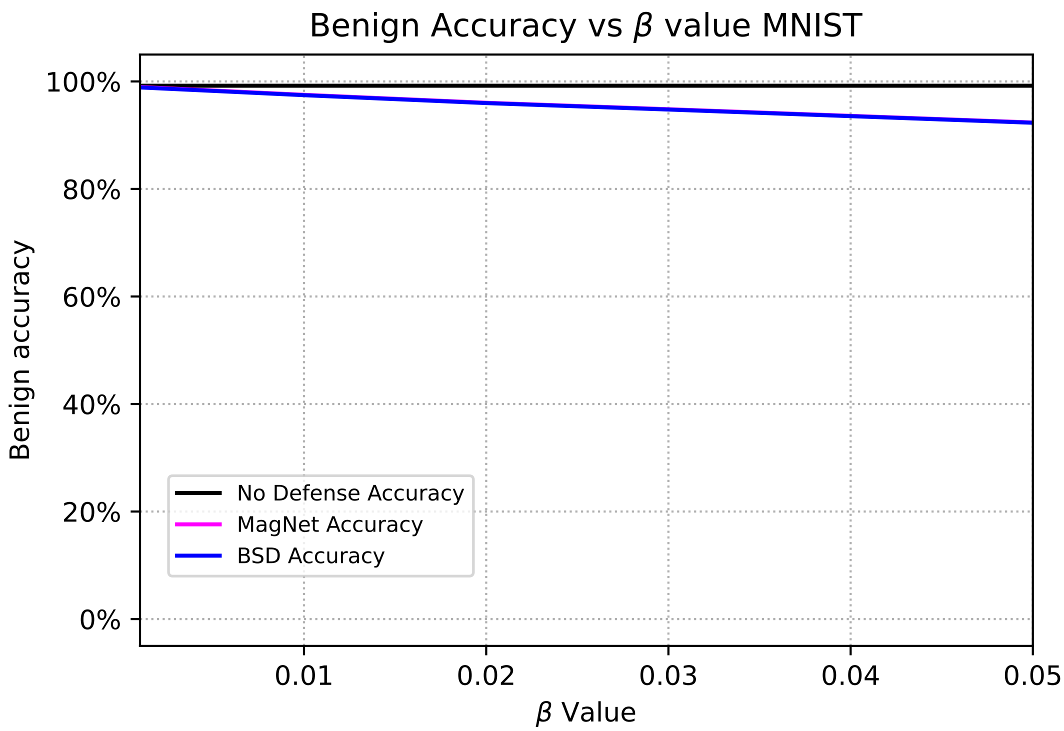
MNIST Classifier Architecture

Layer Type	Layer Parameters
Relu Convolution	32 filters (3x3)
Relu Convolution	32 filters (3x3)
Max Pooling	2x2
Relu Convolution	64 filters (3x3)
Relu Convolution	64 filters (3x3)
Max Pooling	2x2
Relu Fully Connect.	200 units
Relu Fully Connect.	200 units
Softmax	10 units

5.4.2 Benign Accuracy Evaluation

The benign accuracy for MNIST is shown in Figure 23. The blue line (BSD) and pink line (MagNet) show the benign accuracy of each defense on the unmodified test set of the MNIST dataset. We are able to form a one to one comparison between MagNet and BSDs' benign accuracy as they are both controlled by the same parameter β. Figure 23 shows that as the β value is increased towards 0.05, the accuracy drops from 100% to 95%, an expected behavior. The "No Defense Accuracy" shown in Figure 23 refers to the accuracy of the classifier operating on the same (non-adversarial) test set as the MagNet and BSD defenses. Note that the accuracy of the classifier on the test data, or the "No Defense Accuracy" is independent of the β value and serves as a baseline (zero false positive detections). Increasing β increases the quantity of false positives in the test set, so we would expect a drop in performance on the clean test data as β is increased. On MNIST, both defenses have an identical false positive rate at each β value (as shown by overlapping blue and pink lines), indicating that the Class Divergence Detector has not increased the number of false positive detections on MNIST.

Figure 23
Impact of Detectors on MNIST



5.4.3 Performance on All Attacks

Table 4 shows the adversarial accuracy (Equation 5.1) of BSD, MagNet, and FS against a variety of attacks. It is important to note that the attacks used include a variety of L_0, L_2, L_∞, and non-L_p norm based attacks. The attacks were all white box attacks, the strongest – if unrealistic – attacks possible, generated with complete knowledge of the classifier, parameters, and data. The attacks were not aware, however, of any defenses potentially applied. A defense that is truly robust should offer protection against any adversarial attack regardless of the distance metric or formulation. The table shows that in all cases, BSD outperforms MagNet and FS on this dataset. It is important to note that Figure 23 has shown that BSD can be applied without introducing additional false positive detections over MagNet, and applying BSD can slightly improve adversarial accuracy over the wide variety of attacks shown in Table 4.

Table 4

MNIST Adversarial Accuracy at β = 0.05

Attack Type	**BSD**	**MagNet**	**FS**	**No Defense**
FGSM	**99.45 %**	98.79 %	93.74 %	9.93 %
JSMA	**94.45 %**	94.42 %	76.98 %	9.70 %
PGD	**100.00%**	**100.00%**	91.45%	14.45%
DeepFool	**99.44 %**	99.42 %	76.98%	0.00 %
Carlini Wagner	**91.38%**	91.33%	89.48%	9.8%
EAD	**96.56%**	95.26%	96.02%	0.00 %
Spatial	**86.16 %**	85.68 %	76.41 %	0.00 %
Shadow	**99.32 %**	99.22 %	97.95 %	8.79 %

5.5 CIFAR10

5.5.1 Experimental Setup

To evaluate the performance of BSD on a more complex dataset, the results on the CIFAR10 data set were analyzed. A classifier trained to 81.7% accuracy (under no attack) was used to evaluate the defense. Table 5 shows the architecture of a simple convolutional neural network. This CNN architecture we used is identical to the one used in defensive distillation and related works. [34] [28]. In Table 5, the first column refers to the layer type, the second column shows the layer parameters such as number of filters, number of nodes, or kernel size.

To mimic the work done in *On the Limitations of MagNet Defense Against L-1 Based Adversarial Examples*, the autoencoder contains 3 layers, the first two are kernel size of 3x3 with 256 filters, the last layer is kernel size of 3x3 with 3 filters [26]. The CIFAR10 data was normalized to zero mean with -1 to 1 bounds.

Table 5

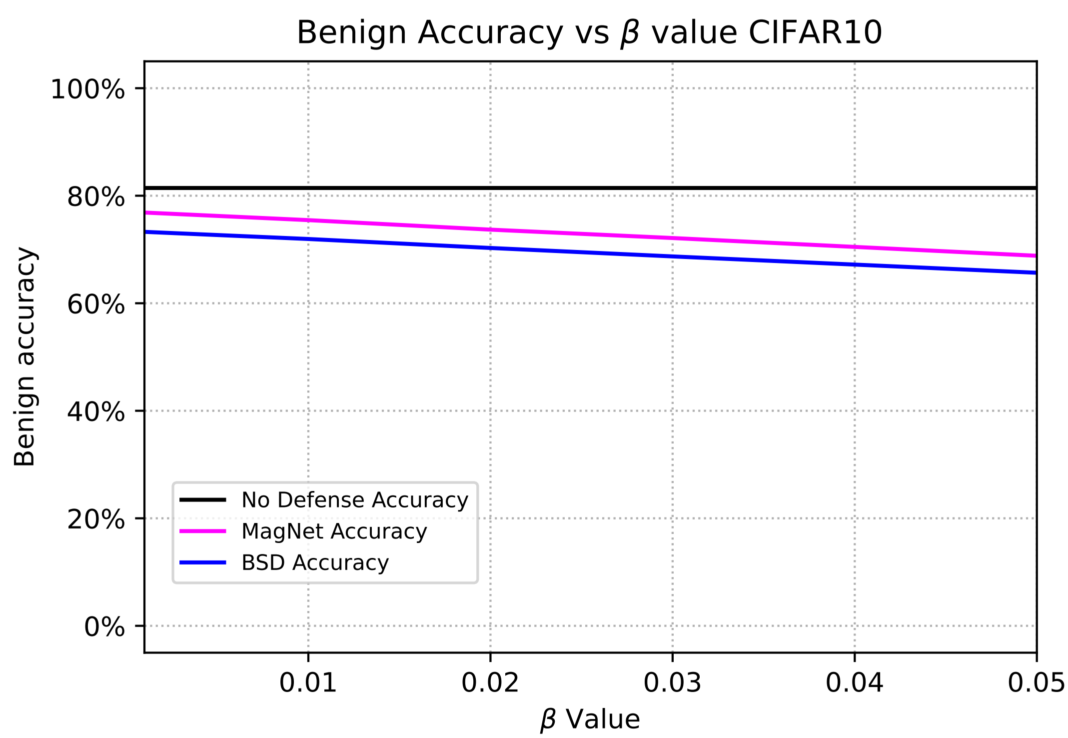
CIFAR10 Standard Classifier Architecture

Layer Type	Layer Parameters
Relu Convolution	64 filters (3x3)
Relu Convolution	64 filters (3x3)
Max Pooling	2x2
Relu Convolution	128 filters (3x3)
Relu Convolution	128 filters (3x3)
Max Pooling	2x2
Relu Fully Connect.	256 units
Relu Fully Connect.	256 units
Softmax	10 units

5.5.2 Benign Accuracy Evaluation

The benign accuracy (Equation 5.4) of the defense on the non-adversarial test set is evaluated to determine the quantity of false positives. A sweep is performed over β values to demonstrate the change in benign accuracy on the test set as the β parameter is increased. Figure 24 shows that with a β of 0, 76% accuracy is achieved with MagNet and 73% is achieved with BSD. The figure also shows that the No Defense Accuracy, or the accuracy of the classifier on the benign test set with no defense applied, is 81%. The result is expected as MagNet and BSD follow the same slope meaning that the β dependent components are detecting the same samples. The β dependent components are the reconstruction error based detector and the JSD detector. The constant difference between MagNet and BSD - as seen in Figure 24 – is attributed to the Class Divergence Detector. As BSD includes an additional detector, it will detect more clean samples as false positives than MagNet, but – as we will see below – the extra detector will also catch more adversarial samples than MagNet. As the threshold is increased, the accuracy on the test set declines linearly – as expected – similar to the results observed on the MNIST set.

Figure 24
Impact of Detectors on CIFAR10



5.5.3 Performance on Wide Spectrum of Attacks

Different adversarial attacks were performed against the neural network used for classification, some of these attacks are fundamentally different in how they construct adversarial examples. Table 6 shows that in most cases BSD offers a gain in accuracy over other defenses - often with wide margins. This gain is most pronounced in the spatial and shadow attacks. This result is expected as the spatial and shadow attacks do not optimize using a L_P norm, instead they utilize different similarity metrics to ensure the adversarial images look like the original images. The table demonstrates that BSD offers an 32% gain on the shadow attack over Feature Squeezing and an 23% gain over MagNet. Since the primary difference between MagNet and BSD is the class divergence detector (CDD), the 23% performance gain can be attributed to the CDD. Another interesting take away from the table is that MagNet and BSD both achieve 99.90% adversarial accuracy on PGD while FS only achieves a 1.02%. Table 6 also shows that Elastic Net and Spatial attacks were unable to develop strong attacks as the accuracy of the classifier without a defense is as high as 82.62% and 84.47 %. In both of these cases, applying BSD still leads to an increase in performance as some of the adversarial samples successfully crafted are detected.

Table 6

CIFAR10 Adversarial Performance at β = 0.05

Attack Type	BSD	MagNet	FS	No Defense
FGSM	**43.85 %**	31.84 %	38.87 %	18.75 %
JSMA	**97.36 %**	69.33 %	62.99 %	2.15 %
PGD	**99.90%**	**99.90%**	1.02%	23.53%
DeepFool	**86.43%**	73.34%	39.89%	8.1%
Carlini Wagner	**81.84 %**	73.44 %	73.44 %	0.00 %
Elastic Net	**90.63 %**	77.34 %	77.54 %	82.62 %
Spatial	**91.11 %**	79.30 %	83.11 %	84.47 %
Shadow	**77.15 %**	53.61 %	44.92 %	45.51 %

5.6 CIFAR100

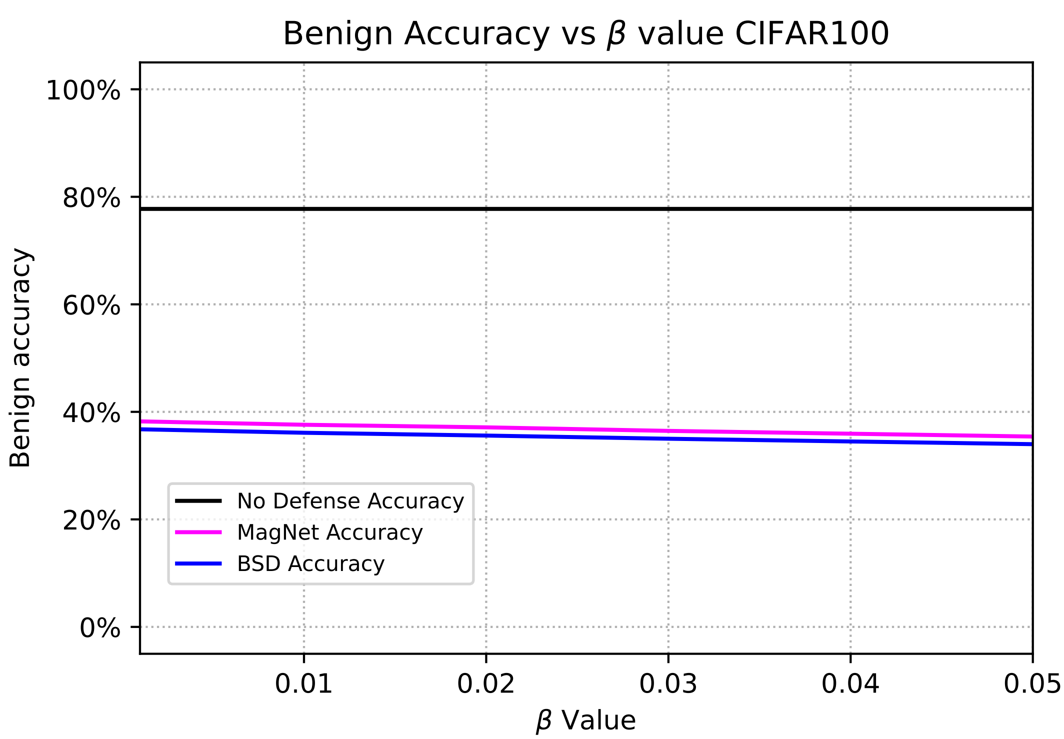
CIFAR100 is a significant increase in difficulty over CIFAR10 due to increase in number of classes from 10 to 100. As a result, a simple neural network can only yield results in the 20-30% range. Therefore, we used a Wide Res-Net, which achieved a 79% accuracy with benign test data. The autoencoder used follows the same architecture used in CIFAR10, but is retrained for CIFAR100. The same preprocessing steps are also repeated, the data was normalized to zero mean with -1 to 1 bounds.

5.6.1 Benign Accuracy Evaluation

As shown in Figure 25, BSD – which adds the class divergence detector to MagNet – results in a slight benign accuracy (Equation 5.4) decrease of 1.5% compared to MagNet. As β is increased, we see the expected linear downtrend of benign accuracy of MagNet and BSD. The surprising result in Figure 25 is the 40% drop caused by applying MagNet and BSD to CIFAR100. This extreme reduction in benign performance can be attributed to the fact that the target classifier used is a wide-resnet containing 36 million parameters is able to achieve the 79% accuracy on the test set. Due to CIFAR100s relatively low support of only 500 training images per class, the wide-resnet fits the dataset very tightly. This tight fit is fine for classification of the test set, however in MagNet and BSD, the images are passed through an autoencoder and are then classified. This process of passing the images through the autoencoder causes enough disturbance to the image to chance the classification resulting in the 40% benign performance. A possible remedy to the benign performance drop involves retraining the autoencoder to introduce less modification in the samples. Through experimentation we were unable to find a suitable autoencoder, once an autoencoder would remedy the benign accuracy, the defense would lose ability to detect adversarial samples.

Figure 25

Accuracy on Clean Data for CIFAR100 Wide ResNet



5.6.2 Evaluation on Adversarial Data

A variety of attacks were launched against the CIFAR-100 dataset. Table 7 demonstrates that on the CIFAR100 dataset, BSD out performed MagNet and Feature Squeezing on every attack. In the case of FGSM, MagNet and BSD both achieved 100% adversarial accuracy. BSD demonstrates very high performance gains of over 30% on JSMA, DeepFool, Carlini Wagner, Elastic Net Attack (EAD), and Spatial attack. Although BSD does provide excellent coverage against adversarial attacks on the CIFAR100 dataset, one must also consider the extremely high cost in benign accuracy shown in Figure 25.

Table 7

CIFAR100 Simplified Adversarial Performance at β = 0.05

Attack Type	**BSD**	**MagNet**	**FS**	**No Defense**
FGSM	**100 %**	**100 %**	18.4 %	2.34 %
JSMA	**98.93 %**	45.89 %	19.04 %	0.01 %
PGD	**99.90%**	99.70%	8.67%	2.92%
DeepFool	**92.09%**	40.82%	35.04%	18.65%
Carlini Wagner	**85.84 %**	42.48 %	31.95 %	0.00 %
EAD	**93.16**	48.43	16.06%	4.79%
Spatial	**94.14 %**	45.31 %	13.91 %	12.54 %
Shadow	**92.18 %**	74.80 %	3.00 %	2.8 %

5.7 TinyImageNet

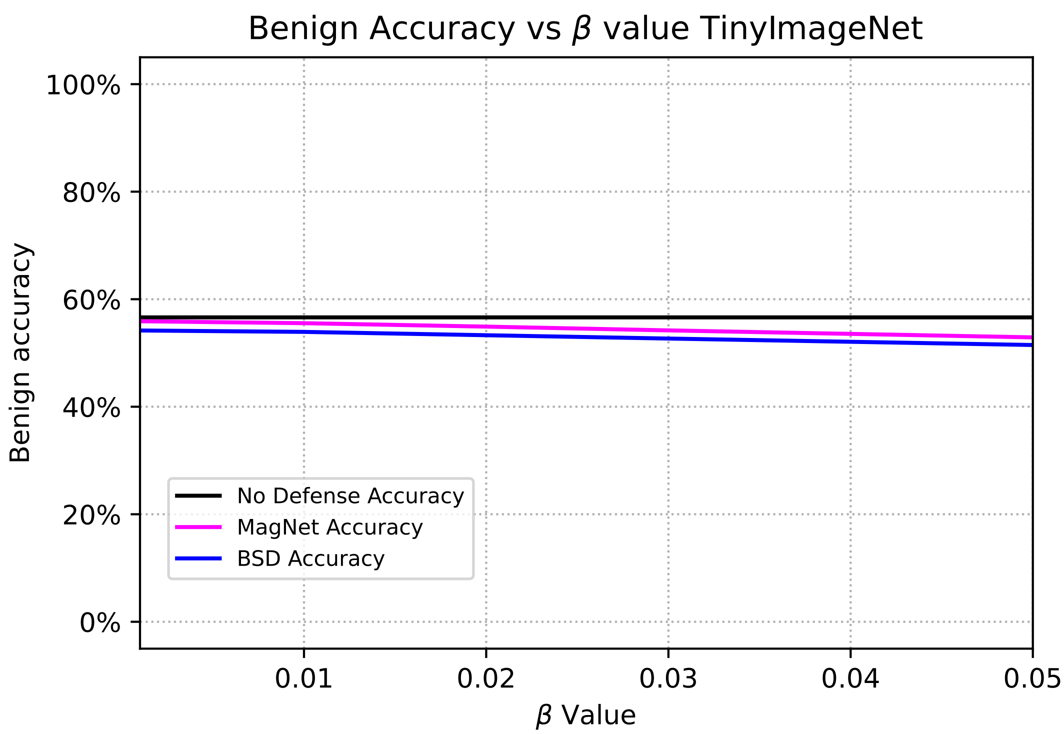
On TinyImageNet dataset, the Feature Squeezing defense was found to only detect between 5 and 15% of adversarial examples for any given attack. We have utilized the authors' implementation of Feature Squeezing and used the same hyperparameters the authors have selected for the TinyImageNet dataset. We believe the low performance of FS comes from the fact that we have selected to evaluate the defenses on a ResNet18 network consisting of 11.6 million parameters. The authors' of Feature Squeezing used a MobileNet architecture made up of only 5 million parameters. The FS defense relies on applying filters to images and examining the classifications, a very tightly fit model (like ResNet18) will have decision boundaries drawn closer to the samples than a model with a looser fit. Due to the tight fit of the ResNet, we believe that the changes introduced by the filters of FS are not able to push the adversarial samples across the decision boundary towards the correct class. This inability to change the classification of samples by applying filters results in the low 5% to 15% performance found on TinyImageNet.

5.7.1 Benign Accuracy Evaluation

A larger autoencoder is needed on this dataset since the feature space is considerably larger compared to other datasets, and a more complex network is therefore required to learn the underlying data distributions. Figure 26 demonstrates that BSD and MagNet can be applied to TinyImageNet with minimal impact on the clean data performance. Note how with a β of zero MagNet results in 1% loss in benign accuracy and BSD introduces a 2% loss in benign accuracy (the additional 1% is attributed to the Class Divergence Detector).

Figure 26

Accuracy on Clean Data for TinyImageNet



5.7.2 Evaluation on Adversarial Data

To evaluate the TinyImageNet dataset on adversarial images we followed the same procedure used on the other datasets. We performed the same set of attacks over the test set of TinyImageNet and evaluated the adversarial accuracy of BSD, MagNet, and FS defenses to compare them and highlight the differences in performance. This comparison is shown in Table 8. As described earlier, the results for Feature Squeezing are between 5 and 15% against every attack. It appears that for larger datasets, Feature Squeezing may need to be tweaked to the specific model used for classification to achieve the best performance. This is important to note as the goal of BSD is to design a defense that can be applied to any model and any dataset without requiring the tuning of parameters. Table 8 shows that in most of the attacks on TinyImageNet, MagNet is very close in adversarial accuracy or ties the accuracy of BSD. In EAD and Spatial Attack the difference is more pronounced as BSD achieves a 5.47% and 7.71% increase in adversarial accuracy.

Table 8

TinyImageNet Adversarial Performance at β = 0.05

Attack Type	**BSD**	**MagNet**	**FS**	**No Defense**
FGSM	**100.00 %**	**100.00 %**	9.75%	0.20 %
JSMA	**99.41 %**	97.85 %	14.26%	0.29 %
PGD	**100.00%**	**100.00%**	4.35%	2.54%
DeepFool	**67.68 %**	63.37 %	6.98%	0.00 %
Carlini Wagner	**66.68 %**	61.94 %	5.09%	0.25 %
EAD	**73.24%**	67.77%	5.7 %	58.59%
Spatial Attack	**48.24 %**	40.53 %	5.92 %	11.52 %
Shadow Attack	**86.81 %**	83.69 %	4.59%	21.48 %

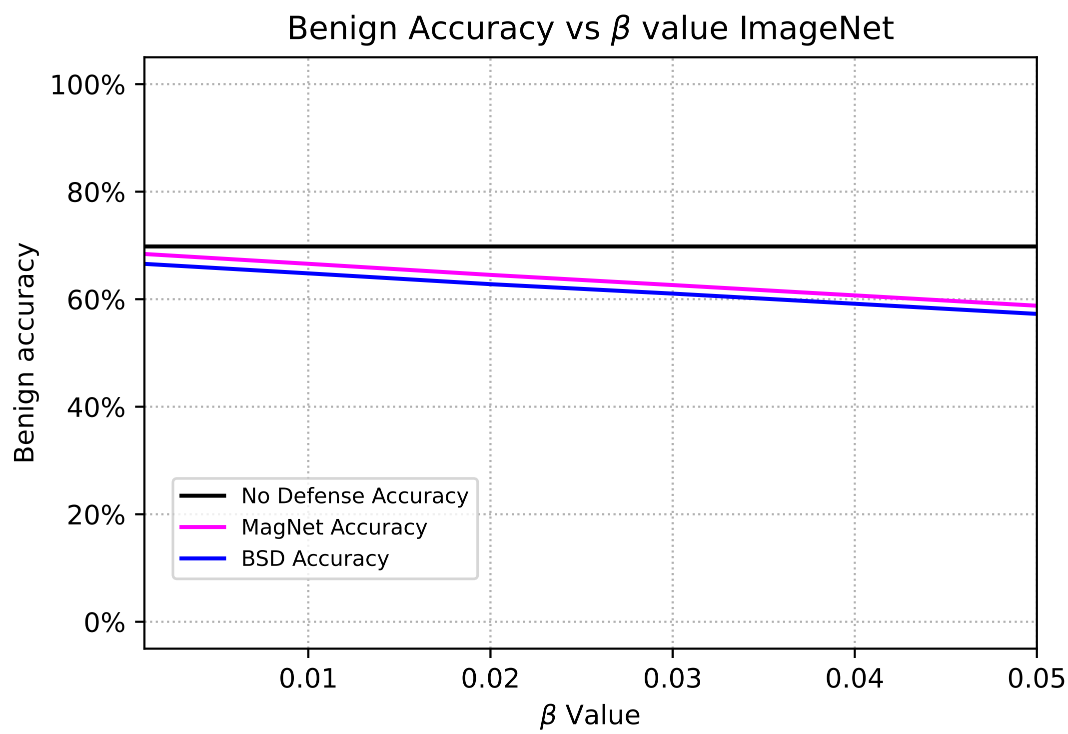
5.8 ImageNet

The ImageNet dataset consists of very large 256x256 images. This dataset is used to test the upper bound of BSD and see if it can continue to defense against adversarial attacks in such a vast feature space. On the ImageNet dataset, the Feature Squeezing behavior mimicked the results on TinyImageNet data as Feature Squeezing was only able to detect between 5% and 13% of adversarial examples on any given attack. Once again, this could be attributed to the fact that we used the authors selected hyper-parameters instead of attempting to tune the hyper parameters to the ResNet18 model used for classification. Once again, this is an interesting result as the goal of BSD is to design a defense capable of generalizing to any dataset and defending any model without adjusting hyper-parameters.

5.8.1 *Benign Accuracy Evaluation*

On ImageNet, we train a large autoencoder to learn to denoise the samples. Figure 27 shows that on ImageNet the classifier achieves a 69.81% accuracy on the clean test dataset. At a β value of zero, BSD achieves a benign accuracy of 66.57% and MagNet results in a benign accuracy of 68.41%. As the β parameter is increased, the benign accuracy quickly decreases. This decrease is attributed to the false detections by both the JSD and the reconstruction error detector. These false detections are not coming from the CDD as the CDD is independent of the β. Futhermore, the CDD can be seen by measuring the constant difference between MagNet and BSD, in this case the CDD results in a 1.84% decrease in benign accuracy from MagNet.

Figure 27
Accuracy on Clean Data for ImageNet



5.8.2 Evaluation on Adversarial Data

Imagenet is a very large dataset with a vast feature space, for this reason certain attacks become infeasible to compute. One example is the JSMA attack, which requires the creation of a matrix of forward derivatives for each input pixel, a process that requires more than 12 gigabytes memory (the capacity of the Titan V GPU used to generate attacks), therefor JSMA was not computed on ImageNet. Table 9 shows that in all cases BSD out performs both MagNet and Feature Squeezing. In a few cases, the difference in adversarial accuracy between BSD and MagNet is more pronounced. Two of these cases are the Spatial and Shadow attacks where BSD outperforms MagNet by 7.82% and 8.6%. It appears that over all data sets, BSD provides the most performance gain on these non-L_P norm attacks.

Table 9

ImageNet Adversarial Performance at β = 0.05

Attack Type	**BSD**	**MagNet**	**FS**	**No Defense**
FGSM	**98.20 %**	97.76 %	1.10%	0.14 %
JSMA	N/A	N/A	N/A	N/A
PGD	**82.03**	80.27%	2.9%	28.32%
DeepFool	**86.32%**	81.84%	6.14%	78.51 %
Carlini Wagner	**83.98 %**	80.46 %	7.67 %	43.36%
EAD	**96.48%**	91.41 %	12.15 %	4.68%
Spatial Attack	**63.28 %**	55.46 %	8.22 %	38.87 %
Shadow	**59.38%**	50.78 %	4.29%	38.86%

5.9 Summarized Results Over Datasets

The results across all our experiments show that BSD does offer a significant improvement over MagNet and Feature Squeezing in all of the datasets shown, but this improvement comes at the cost of more false positive detections. The false positives occur due to the additional CCD detector added to MagNet. This detector will at best, flag no additional clean samples, but in reality few clean samples are detected as adversarial. The benefit of applying BSD over MagNet becomes more pronounced as the complexity of the dataset is increased. For example, applying BSD to MNIST offers only slight improvement overall. Applying BSD to CIFAR10 results in a larger performance gain, and finally applying BSD to TinyImageNet and ImageNet result in the largest performance gains over MagNet and FS. These results show that the class divergence detector appears to scale with complexity of datasets and models well. This result also makes sense as more complex models are fit tightly, and a small disturbance can change the classification of the sample. The class divergence detector was designed to solve this exact problem.

When looking at each defense by attack, it becomes apparent that BSD offers significant accuracy gains on Shadow and Spatial attacks. This is promising as these attacks were designed using non-L_P similarity metrics resulting in fundamentally unique attacks. BSD's ability to improve performance against Shadow and Spatial attacks demonstrate promise that BSD may also provide some level of robustness against future yet to be developed attacks.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

6.1.1 Overview

The Broad Spectrum Defense is designed to be a robust defense capable of generalizing across a wide variety of evasion attacks on any neural network and dataset. There are a variety of evasion attacks constructed with different metrics, methods, and loss functions. All of these attacks share the same goal: to generate a data point indistinguishable from the original while forcing a misclassifciation from the network. Many defenses have been developed in recent years, but a majority of these defenses focus on a subsection of the problem such as defending against all L_2 norm attacks, or defending against all L_P norm attacks. Some defenses utilize adversarial data in training, these methods are directly biased towards one type of attack: the one on which they are trained. These type of defenses do not generalize well to other types of attacks that may be developed in the future. As such, these defenses are considered to be *reactive approaches* as they were developed in response to a specific attack. The most effective evasion defense will be one which is not reactive, but rather proactive. An optimal adversarial defense should be able to detect and reject adversarial samples generated with any attack style on any dataset without flagging legitimate samples.

6.1.2 Contributions

We designed the Broad Spectrum Defense to defend against a wide spectrum of evasion attacks and subsequently to improve the performance of a network on any dataset when under attack. The Broad Spectrum Defense can scale to protect any size classifier and is not biased by adversarial data in the training phase. As adversarial data is not used in the creation of this defense, we believe that the defense will generalize well to future adversarial attacks providing some level of protection. The Broad Spectrum Defense falls short of an optimal evasion defense as it does not achieve the ideal but unrealistic goal of detecting 100% of attack samples from every attack without flagging (detecting) any legitimate data as adversarial. We observed that BSD is able to outperform Feature Squeezing and MagNet – two state of the art approaches – in all explored cases. Although Broad Spectrum Defense is not an optimal solution to the problem of detecting evasion samples, it appears to contribute an additional level of robustness against existing attacks on all datasets.

In this work, we evaluated BSD against non-L_P norm attacks, demonstrating the difficulty of defending against such attacks. The unique nature of using different non-L_P attacks allow for the evaluation of attack samples that fall outside of the normally explored L_P norm attack space. The approach of using these non-L_P norm attacks demonstrates the importance of trying to define and reject the entire adversarial space as opposed to trying to define the adversarial space explicitly using existing attacks. This work highlights the need for an adversarial defenses capable of defending any dataset and classifier against any (or, at least a broad spectrum of) attacks. The adversarial defense crafted to meet these three criteria may even generalize to future attacks. BSD lays the foundation for creating and evaluating such a defense.

6.2 Future Work

Future work should include designing an attack effective against this defense. Carlini has demonstrated that modifying the Carlini Wagner attack allows it to defeat MagNet, the accuracy of BSD can be significantly degraded through this same white box attack[4]. A true white box attack, can be viewed as a worse case scenario, although it is very unrealistic. BSD does not claim to be robust to a white box attack, improving its white box performance is a possible avenue for future work.

The attack scenario analyzed in this work falls under “gray box” as the attacker has complete access to the classifier, but is unaware of the defense. The “gray box” scenario is realistic as an attacker can easily obtain a set of pretrained models and formulate attacks against them. Often the classifier used by the defender is one of these pretrained models, or the adversarial samples generated for the pretrained model are effective against the defenders model. Future work can also involve benchmarking this defense against future gray box attacks and attempting to build out the modular nature of this defense to increase its robustness. When adding additional detectors one must weigh the increase in robustness against the increase in false positive detections.

We hope that this work will inspire the creation of more proactive defenses that are robust against L_P and non-L_P norm attacks. A robust defense should work on data of varying scale from datasets as small as MNIST up to datasets as large (and potentially larger) than

ImageNet. A robust defense should be compatible with any neural network of any size from a simple Convolutional Neural Network up to and beyond a complex EfficientNet model. Another consideration for this potential defense is to manage the trade-off between the ability to detect adversarial samples and the detection of false positives. A truly robust and provably complete adversarial defense sounds like an impossibility, but BSD shows that this impossible goal can be deconstructed into finite problems we can attempt to solve.

References

[1] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ASIACCS '06*, 2006(March):16–25, 2006.

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

[3] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time, 2017.

[4] N. Carlini and D. Wagner. MagNet and "Efficient Defenses Against Adversarial Attacks" are Not Robust to Adversarial Examples. pages 1–5, 2017.

[5] N. Carlini and D. Wagner. Towards Evaluating the Robustness of Neural Networks. *Proceedings - IEEE Symposium on Security and Privacy*, pages 39–57, 2017.

[6] P. Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C. J. Hsieh. EAD: Elastic-net attacks to deep neural networks via adversarial examples. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 10–17, 2018.

[7] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv*, 2017.

[8] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial classification. *KDD-2004 - Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 99–108, 2004.

[9] D. DeMers and G. Cottrell. Non-Linear Dimensionality Reduction. *Advances in neural information processing systems*, 5:580–587, 1992.

[10] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust Physical-World Attacks on Deep Learning Models. 2017.

[11] A. Fawzi, S. M. Moosavi-Dezfooli, and P. Frossard. The Robustness of Deep Networks: A Geometrical Perspective. *IEEE Signal Processing Magazine*, 34(6):50–62, 11 2017.

[12] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner. Detecting Adversarial Samples from Artifacts. 3 2017.

[13] A. Ghiasi, A. Shafahi, and T. Goldstein. Breaking certified defenses: Semantic adversarial examples with spoofed robustness certificates. pages 1–16, 2020.

[14] Z. Gong, W. Wang, and W.-S. Ku. Adversarial and Clean Data Are Not Twins. 4 2017.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–11, 2015.

[16] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel. On the (Statistical) Detection of Adversarial Examples. 2 2017.

[17] D. Hendrycks and K. Gimpel. Workshop track-ICLR 2017 EARLY METHODS FOR DETECTING ADVERSARIAL IMAGES. Technical report.

[18] M. Kearns and M. Li. Learning in the presence of malicious errors. *Proceedings of the Annual ACM Symposium on Theory of Computing*, 21(2012):267–280, 1988.

[19] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images.(2009). *Cs.Toronto.Edu*, 2009.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. *Neural Information Processing Systems (NIPS)*, 60(6):84–90, 2012.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998.

[22] X. Li and F. Li. Adversarial Examples Detection in Deep Networks with Convolutional Filter Statistics. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:5775–5783, 2017.

[23] B. Liang, H. Li, M. Su, X. Li, W. Shi, and X. F. Wang. Detecting Adversarial Image Examples in Deep Neural Networks with Adaptive Noise Reduction. *IEEE Transactions on Dependable and Secure Computing*, pages 1–14, 2018.

[24] J. Liu, W. Zhang, Y. Zhang, D. Hou, Y. Liu, H. Zha, and N. Yu. Detection based defense against adversarial examples from the steganalysis point of view. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:4820–4829, 2019.

[25] J. Lu, T. Issaranon, and D. Forsyth. SafetyNet: Detecting and Rejecting Adversarial Examples Robustly. Technical report, 2017.

[26] P. H. Lu, P. Y. Chen, K. C. Chen, and C. M. Yu. On the Limitation of MagNet Defense Against L1-Based Adversarial Examples. *Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN-W 2018*, (i):200–214, 2018.

[27] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 6 2018.

[28] D. Meng and H. Chen. MagNet: A Two-Pronged defense against adversarial examples. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 135–147. Association for Computing Machinery, 10 2017.

[29] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff. On detecting adversarial perturbations. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2 2019.

[30] S. M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:2574–2582, 2016.

[31] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia. Exploiting machine learning to subvert your spam filter. *LEET 2008 - 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, (April), 2008.

[32] M.-I. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, I. M. Molloy, and B. Edwards. Adversarial Robustness Toolbox v1.0.0. pages 1–34, 2018.

[33] N. Papernot, P. Mcdaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. *Proceedings - 2016 IEEE European Symposium on Security and Privacy, EURO S and P 2016*, pages 372–387, 2016.

[34] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. *Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016*, pages 582–597, 2016.

[35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[36] L. Schott, J. Rauber, M. Bethge, and W. Brendel. Towards the first adversarially robust neural network model on MnIsT. *7th International Conference on Learning Representations, ICLR 2019*, 3:1–16, 2019.

[37] M. Sharif, L. Bauer, and M. K. Reiter. On the Suitability of L p-norms for Creating and Preventing Adversarial Examples. Technical report, 2018.

[38] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, volume 311, pages 1096–1103, New York, New York, USA, 2008. ACM Press.

[39] C. Xiao, J. Y. Zhu, B. Li, W. He, M. Liu, and D. Song. Spatially transformed adversarial examples. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pages 1–29, 2018.

[40] W. Xu, D. Evans, and Y. Qi. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. (February), 2018.

[41] H. Zhang and J. Wang. Joint Adversarial Training: Incorporating both Spatial and Pixel Attacks. 2019.