Adversarial Example Resistant Hyperparameters and Deep Learning Networks

Eric Hulderson

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science

University of Washington

2021

Reading Committee:

Brent Lagesse, Chair

Geetha Thamilarasu

Erika Parsons

Program Authorized to Offer Degree:

Cybersecurity Engineering

University of Washington

Abstract

Adversarial Example Resistant Hyperparameters and Deep Learning Networks

Eric Hulderson

Chair of the Supervisory Committee:
Associate Professor Brent Lagesse, Ph.D.
Computing and Software Systems

Carefully crafted input has been shown to cause misclassifications in machine learning based classification systems resulting in the phenomenon of adversarial examples. Hyperparameters, the settings used to build and train machine learning models, have been shown to build machine learning models that are more resistant to adversarial examples. In this paper, we expand the research of hyperparameter saliency and incorporate deep learning architectures to compliment the field of research in addition to exploring the relationships between adversarial resistance and accuracy as well as depth. We find that hidden layer structures as well as activation function are important to resistance of adversarial perturbations, network depth provides for more robustness with some attacks while architecture influences robustness against others, and salient hyperparameter impact on accuracy is complex.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would first like to thank my family for the unwavering support they have provided throughout my research journey at the University of Washington. Their encouragement allowed me to discover the depth of my work capacity as I straddled the line between school, work, and family. I would also like to thank Cisco Systems, Inc. for providing me the opportunity, resources, and flexibility necessary to pursue an advanced degree. They are an amazing company, and I am proud to be a part of it. I would like to thank my committee for challenging my work and providing thoughtful insights. Their guidance stretched my thinking and steered my research so that I may be more successful. Finally, I would like to thank Brent Lagesse and Cody Burkard for sharing their research and allowing me the opportunity to expand upon it. It's been an amazing journey that I will never forget.

With Gratitude,

Eric Hulderson

Chapter 1. INTRODUCTION

Deep learning (DL) has continued to make significant progress across many modalities of machine learning (ML). Its implementations are not limited to computer vision and natural language processing domains and include virtual assistants, fraud detection, news aggregation, automatic game playing, personalization, and healthcare. Recently, DL has been used to detect developmental delays in children and achieved 92% accuracy in detecting Autism spectrum disorder [1], and Twitter has taken fake news detection to the next level through the acquisition of Fabula.ai enabling the social media company to detect upwards of 93% of area under the curve (AUC) [2], an aggregate measure of performance of possible classifications.

1.1 PROBLEM DEFINITION

Real-world applications and systems are increasingly being powered by DL. For instance, companies from the IT industry such as Amazon and Google as well as auto manufacturers such as Mercedes and Tesla are developing self-driving cars, which heavily utilize DL techniques including object recognition, reinforcement training, and multimodal learning [3][4]. Facial recognition is now commonplace and has been implemented by Apple as biometric authentication to unlock mobile phones as well as in highly controversial applications such as street surveillance by law enforcement [5].

While DL may claim great success in numerous applications, many of its empowered applications are life-critical, which raises great concern for safety and security. Research has shown repeatedly that neural networks (NN) are vulnerable to thoughtfully crafted input samples or adversarial examples [6]. It is possible for these samples, which have perturbations hardly recognizable to the human eye, to fool well-performing NNs. Recent studies have shown that it

is possible for adversarial examples to be applied to real-world applications. It has been shown that an adversary can construct a physical adversarial example and confuse an autonomous vehicle's traffic sign recognition system by manipulating a stop sign [7].

To address the concern of safety and security of NNs, researchers have come up with numerous methods to increase the robustness of models. Adversarial training is one such method where adversarial examples are directly incorporated into the training process. While this method is promising, it can dramatically increase the training time of NNs, and it will need to be consistently updated as new adversarial examples emerge. Another method for increasing the robustness of NNs is through careful selection of hyperparameters used to build a NN. Burkard and Lagesse [8] recently showed that selection of hyperparameters to did have an impact on model robustness, although testing was limited to creating adversarial examples with Carlini and Wagner attack with the LeNet architecture and MNIST dataset.

Another practical challenge results from the often-cited tension between model accuracy and robustness [9]. The objective of adversarial training and intelligent hyperparameter selection is to create robust NNs, which is typically measured as the accuracy of a model's predictions on adversarial examples. The increased robustness often comes with a price, which is a decrease in accuracy on non-adversarial examples. Recently, Tsippras et al. [10] were able to show that there exists a trade-off between model accuracy and robustness when models were created using adversarial training. The question remains whether this trend holds true for models created using hyperparameters.

In this work, we will generate hundreds of NN models with five datasets on three convolutional neural network (CNN) architectures with commonly used hyperparameters. We will generate adversarial examples, and we will collect metrics which will allow for the

examination of NN model robustness as it relates to hyperparameter selection, model accuracy, and model depth.

Our methods are derived from the work of Burkard and Lagesse [8], and in addition to broadening the scope of experimentation, we refine and improve their techniques to detect salient hyperparameters. Our work will incorporate the use of state-of-the-art DL architectures, including Residual Networks, as well as a range of datasets with diverse characteristics. Great care will be taken in the creation of the models in our experiments as we will utilize techniques to provide for the best accuracy and lowest error rate while also preventing overfitting. Experiments will be conducted in a fully programmatic method using object-oriented methods to reduce human error and time needed for experiments. Our experiments will also feature the use of an additional method of generating adversarial examples to look for salient hyperparameters that can more broadly resist adversarial examples. More specific information on improvements to technique will be covered in the Results section.

1.2 GOALS

In this paper, we consider the ability of deep neural networks (DNN) to resist adversarial examples, also referred to as robustness, and the following questions:

- Do salient hyperparameters exist on DL architectures which impact model robustness?
- Is there a tradeoff between robustness and accuracy with salient hyperparameters?
- Is there a relationship between robustness and NN depth?

1.3 OUTLINE

The structure of this paper consists of the following: chapter 2 reviews the relevant work of attacks, defenses, and threat modeling for NNs as well as research into correlations between model accuracy, robustness, and depth. Chapter 3 will present a detailed design of the experiments conducted and identify all components used. Chapter 4 will present the logic and methodology for comparing model robustness and the results of the experiments conducted. Chapter 5 will present the contributions to the field and discuss their implications. Finally, chapter 6 will be the conclusion of the paper, and it will discuss any future research directions that may be possible because of our research .

Chapter 2. BACKGROUND

2.1 CONVOLUTIONAL NEURAL NETWORKS

CNNs are a form of DL most applied to analysis of visual imagery. As the name implies, a mathematical operation known as a convolution is used in this type of NN. A convolution is a specialized form of linear operation, and it can be generalized as an operation on two functions of a real-valued argument. Convolutional NNs differ from other NNs as they use convolution in place of general matrix multiplication in at least one of their layers [11].

The basic architecture of CNNs in comprised of many layers each with a specific purpose. A layer in a NN model is a construct in the architecture of the model that takes information from one layer and passes it to another. A CNN will contain at a minimum a convolutional layer, a pooling layer, a receptive field, weights, and a fully connected layer.

The objective of the convolutional operation is to extract the high-level features of the input. The layer takes as input an image with shape defined by image height, image width, and image depth, and it outputs feature maps with the shape defined by, feature map height, feature map width, and feature map channels. The features are extracted using a Kernel or Filter as it shifts over the input image based on the stride, and at every shift performing a mathematical operation, discretely viewed as multiplication by a matrix [11], to extract images features. The filter is also an array of numbers, and the numbers are commonly referred to as weights. The filter moves from left to right based on the stride length until the complete width is parsed, at which point it moves down and repeats the process until the entire image is traversed. The convolutional filter (input channels) must be equal to the number of channels (depth) of the input. To put another way, convolutions may be performed over more than one axis.

A pooling function in a NN further modifies the output of the NN. Its role is to replace the output and provide a summary statistic of the nearby outputs. It does this by making the representation become approximately invariant to small translations of the input [11]. This means that if the input in changed by a small amount, the valued of the pooled outputs are not changed. This is useful in cases where it is important to know if a property is present, but its exact location is not required. There are different types of pooling functions, each with different operations. Operations include max pooling, average pooling, the L^2 norm of a rectangular neighborhood, and a weighted average based on the distance from the central pixel. The summarization that pooling provides also improves the computational efficiency of the network as it report summary statistics for pooling regions based on the kernel width rather than pixel by pixel [11].

The receptive field, or field of view, is the region of input that a CNN feature is looking at, and it is important to understanding how CNNs work [12]. Any area outside of the receptive field does not affect the value of the output feature. It can be described by its center location and size. Not all pixels within a receptive field bare equal importance to the CNN's feature. The pixels more closely oriented to the center contribute more to the calculation of the output feature. Not only does the feature look at a particular region of the receptive field, but it focuses exponentially more to the middle or that region. The size of the receptive field can be increased in several ways. Stacking more layers to make a network deeper can increase the receptive field linearly. Sub-sampling can increase the receptive field multiplicatively. Both VGG networks [13] and Residual Networks [12] use a combination of these techniques.

The fully connected layer, also known as dense layer, is the last layer of the CNN where each input is connected to an output by a learnable weight. It takes an output volume, typically as a one-dimensional array, from a preceding layer, such as a pooling layer or convolutional layer, and it outputs an N-dimensional vector. The N-dimensional vector output is often the number of classes associated with the dataset in the supervised learning problem [14].

2.1.1 *Hyperparameters*

Hyperparameters are settings that can be used to control the behavior of the CNN [11]. In this work, they are broken into two categories, network hyperparameters and training hyperparameters. Network hyperparameters are any settings that pertain to the CNN architecture itself. Examples of these are convolutional kernel size, activation function, and strides. Training hyperparameters are the settings that apply to how the CNN is trained. Examples of these include batch size, momentum, and optimization function.

2.2 Related Works

2.2.1 *Adversarial Machine Learning*

Adversarial ML was first recognized by Dalvi, et al. under the term adversarial classification in 2004 [15]. They described a game played by two players, the classifier, and the adversary. The classifier's role was to learn a function of input that would allow a correct classification of input, while the adversary attempts to make the classifier misclassify input by altering the input. They create a formal framework and algorithm which automatically adapts the classifier to evolving adversary manipulations.

2.2.2 *Machine Learning Attacks*

The term adversarial example was first used by Szegedy, et al. [6] to describe NN input with imperceptible, non-random perturbations that are able to arbitrarily change a network's prediction. NN input undergoes subtle manipulations that are generally unobservable to the human eye, but it can interfere with the ML models interpretation of an input's features causing a misclassification.

Poison attacks date back to about 2008 where they were employed to disrupt an email spam filter [16]. Poisoning attacks work by modifying training data either through label modification, data injection, or data modification so that the ML model's classification boundaries are changed. These attacks can be targeted where training data is intelligently manipulated so that a specific classification boundary is changed, or untargeted where any of the model's classification boundaries are changed.

Privacy attacks are attacks on ML models where an attacker attempts to gain some type of knowledge of the system. The knowledge of the system can be membership inference, where

the attacker tries to determine if an input was part of the training data, or it can also be input inference, or model inversion, where an attacker can extract data from the training dataset used by the ML system [17]. Parameter inference, or model extraction, is the last type of privacy attack were an attacker attempts to gain information about a ML model or its hyperparameters, which can then be used to more readily create evasion attacks [18].

The idea behind a trojan attack in the context of ML is to retrain a ML model to change its behavior in some conditions while leaving existing behavior unchanged. While access to the original training dataset is not required, attackers would need access to the model, its parameters, and have the ability to retrain the model [19].

A backdoor attack is a type of poison attack which targets the integrity of the ML model. In a backdoor attack, a ML model is trained so that the classifier functions as it is intended to, but so it can also accept input that the designer is not aware of in which an attacker can trigger certain behavior [20].

Evasion attacks are the most common type of attack on ML models originally cited in 2004 [15]. These attacks involve the manipulation, or perturbation, of input to a ML classifier with the goal of triggering a misclassification. Typically, this takes place by minimally perturbing image input to a CNN such that it cannot be detected by the human eye.

Optimization attacks are a form of evasion attacks in which optimization algorithms are used to generate adversarial examples. They are often complex, precise, more often in black box scenarios, where there is no access to the ML algorithm, and effective in generating adversarial examples [21].

Another category of evasion attack is sensitivity analysis, where attackers use sensitivity analysis methods to find sensitive features necessary for classification and perturb them. To

perform sensitivity analysis, a class of algorithm is used that is able to determine the contribution of each input feature to the output [21].

Adversarial examples may also be transferable between different ML models. Rozsa et al. show in [22] experiment with multiple adversarial example generation techniques looking for the method which creates the highest quality images, and they use these images to test the robustness of Residual Networks. They examine the transferability of these attacks and find that models must have similar composition in order for the adversarial examples to remain effective in addition to demonstrating that more accurate models are less vulnerable to attack.

Serban et al. have compiled a comprehensive survey on evasion attacks in an object recognition setting in [21], in which they also discuss the potential for adversarial examples to be transferable between ML models. They postulate that it adversarial examples may be created in a white box setting, where the attacker has access to the ML model, and transferred to a black box model, where the attacker has no information about the model. Many facets of adversarial examples are discussed including its impacts on security, safety, and properties of NN robustness.

Liu et al. expand upon the research into adversarial example transferability and delve into the problems associated with it [23]. Prior to their research, little work into adversarial example transferability had been done on large systems with targeted attacks. While they examine untargeted attacks, Liu et al. focus on the problem of successfully transferring targeted attacks by creating a novel ensemble-based approach to generating adversarial examples. Using their approach, they are able to successfully attack an online black box classification system.

2.2.3 *Machine Learning Attack Defenses*

Many countermeasures have been developed to defend NNs from adversarial examples and increase the NNs adversarial resistance. These strategies include sample filtering, defensive distillation, and augmentation of training data. In this section, we will discuss the methods as well as their effectiveness.

Xiao et al. postulate in [24] that feature selection is actually making classifier security worse in the context of poisoning attacks, and they create a novel approach to feature selection that can improve classifier security against evasion attacks when specific assumptions are made about the adversary's data manipulation strategy.

Grosse et at show in [25] that highly-effective adversarial examples can be created for not only image classifiers, but for systems with grave consequences such as malware classifiers. These attacks are leveraged against different instances of malware classifiers, and potential defensive mechanisms such as distillation and adversarial training are explored with some success.

In Nguyen's paper [26], a different approach is taken to fool DNNs. Instead of perturbing images minimally with the intent to fool a classifier, the authors generate adversarial examples that to a human look like noise, but to a classifier they are recognizable with 99% certainty. The authors further examine the topic and find interesting differences between human and computer vision.

Papernot et at [8] proposed defensive distillation as a defense against adversarial examples. This defense is based on transfer learning where a teacher network is pre-trained and the knowledge from that network is transferred to a student network with fewer parameters. Soft labels produced by the teacher network are used when training the student network. They are

advantageous in training because they include predictive softmax output as part of the training data, and they contain additional information about class similarities learned by the teacher network. Classification can then be done on the student or distilled network. The claim is that this approach reduces overfitting and provides for a more secure NN, however, Papernot et al. [27] found that while it does provide for robust security, it only applies to DNNs that produce energy-based probability distributions. Further, Carlini et at [28] found that with slight modifications to a standard attack, they could easily find adversarial examples in defensively distilled networks.

Much of recent research on creating robust NNs resilient to adversarial examples has been heavily focused on adversarial training. This is the process of building adversarial examples for a model, and training with those adversarial examples with the purpose of introducing data to the model that would not otherwise be present in the environment. This is intended to train the model to correctly classify the adversarial example. Introducing this data provides the algorithm with relevant information to better fit the true discrimination function. This idea has been discussed by Goodfellow et al. [29], and the observation is made that the error rate of adversarial examples on a learning model continues to decrease after the error rate of the model on training samples converges to a minimum. This requires a focused effort to be made on implementing training features to detect this condition and halt training, such as early stop.

While adversarial training has been effective at preventing some adversarial examples, the production of adversarial examples using most methods is computationally expensive rendering impractical to compute a substantial number of samples for use with this process. The Fast Gradient Sign Method (FGSM) [29] attempts to solve this problem by easily and quickly producing a large number of samples, but the process results in lower quality examples that do

not adequately represent all possible types of malicious data. It is also based on the linearity hypothesis [29] and does not perform as well on models that are not represented well by this theory. In summary, adversarial training is effective at preventing any type of malicious data that can be produced but fails to represent all possible types of adversarial samples without a substantial amount of computation.

In [8], Burkard and Lagesse examine the effectiveness of hyperparameter selection with respect to NN robustness. They show that various hyperparameters indeed do increase a NN's resistance to adversarial examples. However, to arrive at their findings only a single NN architecture and dataset are used. The concepts explored here should be validated across a larger subset of ML systems and datasets to see if it holds true.

In Hendrycks and Dietterich's paper [30], they create benchmark datasets with the goal of identifying which classifiers should be used when safety is the highest priority of the system and also in research settings where general robustness is requirement. They examine various methods to create adversarial example resistance as well as bypassing certain protection mechanisms and found that it adds robustness to the system. They find that larger models improve robustness to common perturbations as measured in mean corruption error, a metric used to gauge how much stronger a model is against noise compared to AlexNet.

Papernot et al. examine the defense strategy of gradient masking, or obfuscated gradient, in their work [31] as well as other strategies. Many defensive mechanisms fall into the category of gradient masking, a technique that attempts to hide the directions into which the model is sensitive. One approach to this is the creation of a model that has no useful gradient. As an example, a model could be created using nearest neighbor classifier instead of a NN to inhibit an attacker's ability to generate adversarial examples, but it has been shown that models created

with nearest neighbors are also vulnerable to attack [29]. Athalye et at go on to develop three attack techniques to circumvent three different types of obfuscated gradient and conclude that ML practitioners should not rely on obfuscated gradients as a means to adversarial resistant ML models [32].

Another novel attack defense proposed by Zantedeschi et at is the combination of use of network hyperparameter activation unit bounded RELU, or BRELU, and training with Gaussian augmented data [33]. BRELU, bounded rectified linear unit, was first described by Bakhteri and Liew [34] and prescribes the addition of an upper bound as prescribed by the UAT (universal approximation theorem) to the RELU activation unit as RELU is unbounded for non-negative input. The advantage of the defense proposed by Zantedeschi et al. is that the training component is less computationally expensive that standard adversarial training, which allows for larger sets of directions to be explored around inputs that can be effective against a larger range of attacks.

The contribution provided by Berntsen et al. [35] is unique in that it forces the classifier to make predictions on what the input should look like based on the class under consideration and then test those predictions This is accomplished by breaking the model into three distinct stages. The first stage is estimation which estimates a vector of parameters. Using the estimated parameters, the second stage, projection, generates an image. The final stage, comparison, compares the image with the actual image and provides a tunable probability of whether the image is clean or an adversarial example.

Li and Li take a different approach to detecting adversarial examples in [36]. Instead of training a model to detect adversarial examples, they propose a technique where the adversarial example is detected based on statistics in the output of the convolutional layers. This would be

considered a passive defense versus an alternative method like adversarial training, which would be considered an active defense. In addition to this novel technique, classifiers trained in one adversarial generating technique can detect adversarial examples created with other techniques assuming the attack is using the gradient of the classifier.

2.2.4 *Threat Modeling*

Nelson [37] examines ML systems in adversarial environments with the goal of determining methods to comprehensively thread model the landscape of adversarial attacks. His analysis concludes that with a systematic approach, ML practitioners can successfully assess a ML systems vulnerability and deploy methods to strengthen their systems from attack.

In the paper [38], Barreno et at delve into the landscape of ML security by examining the different types of attacks on ML systems, the defenses available to thwart attacks, and a threat model system to better understand the adversary.

2.2.5 *Evaluating Adversarial Robustness*

In the paper 'On Evaluating Adversarial Robustness," Carlini et al. [39] impress upon researchers to be skeptical of all results obtained as it can be very easy to be deceived when performing evaluations. They have formulated a framework of recommendations that can help when performing a defense evaluation. The intent with creating this framework is not restrict the types of testing ML practitioners and researchers should perform, but to inspire others in the field with ideas of how to go about their own evaluations.

Carlini et al. create the attacks that are used in this work in [40]. These attacks, the Carlini-Wagner L0, L2, and L∞ attacks, were created to disprove the assertion that defensive distillation could reduce the success of most adversarial examples. Moreover, these attacks were

created as a means to evaluate the adversarial robustness of NNs. They are intended to serve as benchmarks for future attempts at creating adversarial resistant NNs. These attacks stand to be more than five years old at the time of writing this, yet they remain state-of-the-art and valid as the benchmarks they set out to be.

2.2.6 *Real-world Applications*

Kurakin et at show in [41] that the current threat models assumption in which an adversary can feed data directly into a ML classifier is insufficient. They proceed to show that in physical world scenarios, such as using signals from cameras and other sensors, adversarial examples are classified incorrectly.

In [42], Graese et al. look at the real world implications of adversarial examples by investigating their transferability following a normal image acquisition process. Their research shows that misclassifications are less frequent for adversarial examples following small transformations such as multiple crops, image preprocessing, and text binarization.

In [43] Kurakin et al. seek to address the problem of how to apply adversarial training to large models and datasets. In the process of applying adversarial training to large models and datasets, they observe that adversarial training creates adversarial resistance to single-step attack methods, find that single-step attack methods are best for black-box attacks, and solve the problem of 'label leak,' which causes adversarial examples to classify more accurately than clean examples.

Eykholt et al. [7] have created an attack algorithm specially designed to cause misclassifications in the real-world case of road sign classification. Their attack, called RP2 or Robust Physical Perturbations, is designed to be used in a two-stage system. Initially, they generate perturbations in a lab setting, and for the second step they apply the perturbation to road

signs. Using this system, they show it is possible to generate physical adversarial examples at varying distances and angles.

2.2.7 *Robustness Versus Accuracy*

Prediction accuracy has been a long-lasting standard to compare performance of ML models. The relationship between this standard and ML model robustness has recently become an area of focus with attacks on ML models becoming more common. Tsipras et al. [10] investigate the effects of adversarial training on the ability of a model to provide accurate classification, and they are able to demonstrate that robust models are less inept at standard generalization. Rozsa et al. [22] look at the accuracy versus robustness problem from a different angle. They generate adversarial examples using three different techniques, and they evaluate their performance against state-of-the-art models. They found that the more accurate models required greater perturbations to form adversarial examples. Their belief is that more accurate models learn feature mappings for a given classification task that make classes more separable providing increased accuracy, which is what makes them more resistant to adversarial examples.

Su et al. [9] conducted a comprehensive study and found a different result when looking at the accuracy versus robustness problem. In their work where they performed a large study of 18 classification models, they found there was a clear trade-off between accuracy and robustness. The realization was that as testing accuracy increased in performance, robustness reduces generally. Su et al. also looked at robustness as a function of model size and architecture finding that a ML model's architecture was a more accurate predictor of robustness than its size. However, Su et al. did also observe that model size, or depth in this case, did provide a slight improvement to model robustness when the models were of the same architecture.

Duesterwald et al. [44] look into some of the practical challenges that exist with adversarial training as it relates to the new parameters that this technique make available. They conclude that effectiveness of adversarial training is dependent upon the proper setting of these parameters, and that these settings are specific to the architecture and training data in the experiment environment.

Chapter 3. METHODOLOGY

In this chapter, the details of the experiments conducted in this work will be covered including the CNNs creation process, datasets used, specific NN architectures, the experiment design, metric definitions, the attack methodology, and methods for comparing results. Moreover, reasoning for choosing each element of the experiment will be described.

3.1 CONVOLUTIONAL NEURAL NETWORKS

CNNs are chosen for this work as they are easier to use to conduct the experiments outlined in this paper. For the sake of detecting salient hyperparameters, it is easier to perturbate images to mislead a CNN classifier than it is to perturbate files with the intention of misleading a ML-based malware classifier. Regardless of the context of the ML experiment, the information derived from the results should be able to be universally applied.

Our CNN robustness testing is focused on three different architectures: LeNet, ResNet50, and VGG16. Each are supported by the Keras DL API, which will allow for code portability between experiments on different architectures. Other CNN architectures were considered for this experiment including ResNet101, ResNet152, and Inceptionv3, but as these CNNs are well known, their deeper-learning capabilities required significant training time and compute resources. The following sections provide additional details on the structure of these CNNs as well as additional motivation for use.

3.1.1 *LeNet*

The LeNet-1 architecture represents a very early CNN. It is a basic architecture that has been well studied, and it is the basis of many modern architectures including GoogLeNet [45].

Created by LeCun et al. in 1998, it is composed of two convolutional layers each followed by a max pooling layer terminated by a single dense, fully connected layer with ten nodes [46]. LeCun et al. continued to develop on top of this architecture and quickly came up with LeNet-4, LeNet-5, and boosted LeNet. LeNet-4 saw the addition of another fully connected layer in between the last max pooling layer and the output layer. The new fully connected layer had 120 nodes. The LeNet architecture saw another revision in the form of LeNet-5, as pictured in figure 3.1, which had the addition of another fully connected layer (F6 in figure 3.1) between the 120 node, fully connected layer, and the 10-node output layer. The new fully connected layer had a total of 84 neurons.

This NN architecture has been chosen as a baseline for testing more complex, deeper NNs. It offers the flexibility to allow for changes to many hyperparameters while still providing a reasonable level of accuracy. The LeNet architecture tested in this work more closely resembles that of LeNet-4 as it has two fully connected layers, a 500-node fully connected layer and an output layer with the number of nodes equal to the number of classes in which the dataset it is trained on. We captured the default parameters of the NN in the appendix.

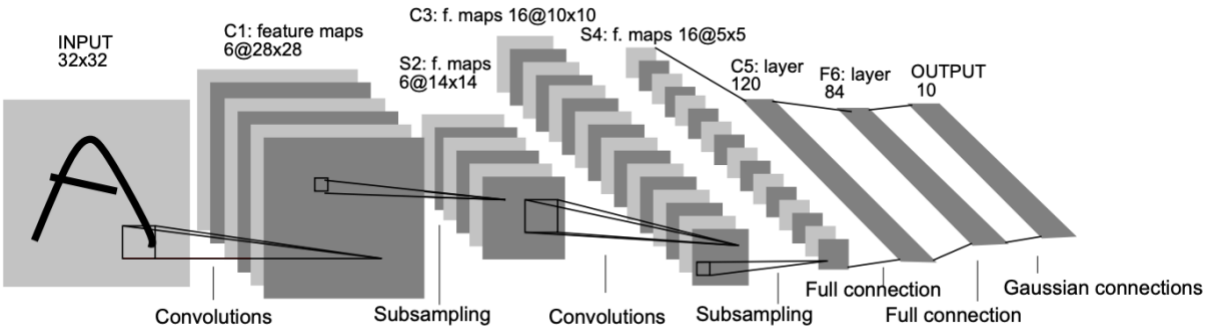

Figure 3.1 - LeNet-5 Architecture, An Early CNN [46]

3.1.2 *VGG16*

The VGG16 architecture was created by Simonyan and Zisserman [13], and it was the first runner-up in the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The NN, boasting 138 million parameters, was originally setup to accept a 224 x 224 RBG image. The images are passed through a series of convolutional layers, each which incorporate a 3 x 3 filter, a distinct filter size for a CNN. Note that the 3 x 3 filter is the smallest of its size to allow for capture of the concept of left/right, up/down, and center. Not all convolutional layers are followed by a max pooling layer as is the case with the LeNet architectures. Instead, convolutions are grouped as shown in figure 3.2 from conv1 to conv5, with each group ending with a max pooling layer. The max pooling layers with the VGG16 architecture are designed to preserve spatial resolution. Each max pooling layer utilizes a max pooling filter size of 2 x 2 and it incorporates a stride of 2. Interestingly, the convolutional layers implement a stride of 1. The VGG16 architecture implements three hidden layers. The first two layers have 4096 nodes, while the last layer has 1000 nodes, which is equal to the number of classes in the ImageNet dataset.

VGG16 was chosen for this work as it is a popular, well studied CNN, and it is approximately three-fold deeper than LeNet. The VGG16 architecture as it was tested in this work allowed for processing of smaller images (48 x 48) as well as processing of grayscale images.

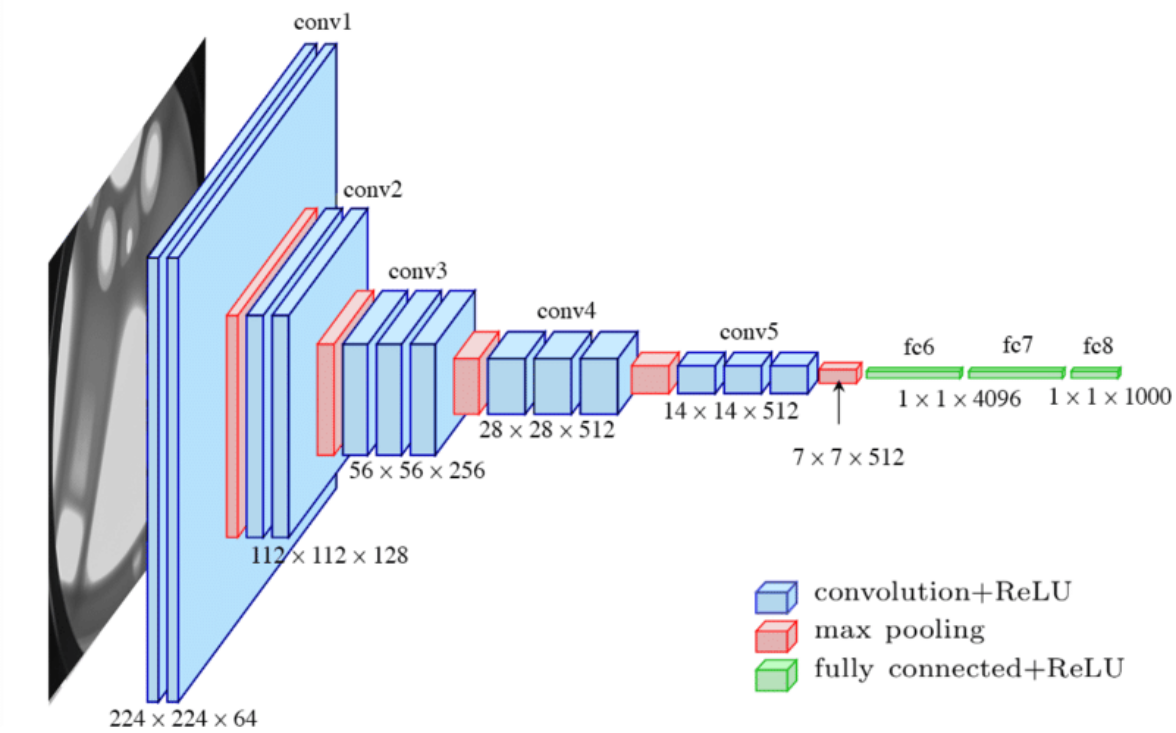

Figure 3.2 - VGG16 Architecture [47]

3.1.3 *ResNet-50*

ResNet, short for Residual Network, was developed by He et al. and was inspired by VGG [48]. It was the winner of the 2015 ILSVRC. Much like VGG16, the baseline architecture is comprised of convolutional layers with a 3 x 3 filter. The development of the ResNet architecture included two design rules. First, when the output feature map size is the same, the layers should have the same number of filters. Second, if the feature map size is halved, the number of filters is doubled to preserve the computational complexity of every layer. Using a stride of 2, convolutional layers directly perform down sampling. The network utilizes an average pooling layer and a fully connected layer of 1000. Average pooling calculates the average value of a section of the feature map, while max pooling calculates the max value of a

section of the feature map. The use of an average pooling layer is unique of the models selected for this work. ResNets employ other unique features referred to as shortcuts where layers can be bypassed when certain criteria are met. These shortcuts, or identity shortcuts, can be used when the input and output are of the same dimension. Shortcuts may also be used with increased dimensions, but only when the addition of padding of zeros for increased dimensions occur. Another shortcut, the projection shortcut, is used to match dimensions with a 1 x 1 convolutional filter. Each shortcut described here use a stride of 2.

The ResNet50 architecture was chosen for this work due to its size, popularity, and ease of use. At over 23 million parameters and 50 layers, it is nearly 3 times as deep as the VGG16 architecture, although it has more than 100 million less parameters. Additionally, with the architecture being the winner of the 2015 ILSVRC, it has been well studied in academic context making it a good fit for this work.

The ResNet50 architecture, as well as Residual Networks of other sizes, are visually described by figure 3.3. As evident by the figure, all ResNets have in common the conv1 layer, which uses a convolutional kernel size of 7 x 7 and a stride of 2. They also utilize a max pooling layer with a 3 x 3 filter and stride of 2, and they end with an average pooling layer and a fully connected output layer. How the various ResNets differ is how the layers conv2 through conv5 are implemented and repeated.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3.3 - Five Stages of the Residual Network Architectures [48]

3.2 DATASETS

A total of five datasets are used in the experiments in this work. They include MNIST, Fashion MNIST, CIFAR10, GTSRB, and MNIST Sign Language. Each dataset has been well explored in the ML community, which allows the work contained here to be more easily comparable to other works. The datasets were chosen as they had different characteristics and were also easy to use. The image sizes within the datasets are small making them more manageable for the experiments defined here as they will require less computational time. The number of classes within the datasets are also different which will be useful for making attack comparisons. The range of datasets includes at a minimum 10 different classes and a maximum of 43 different classes with various image sizes also represented. Three of the datasets are grayscale (1 channel) while the remaining two datasets are RGB (3 channel). Table 6.1 summarizing the datasets used in this paper can be found in the appendix.

3.2.1 *MNIST*

The MNIST dataset is a database of handwritten digits ranging from zero to nine. The dataset has 60,000 training examples and 10,000 test examples. Images in the dataset are grayscale, which means they are a range of gray shades expressed with a value from zero to 255. The dimension of MNIST images is 28x28x1.

The MNIST dataset is a subset of a larger dataset created by NIST (National Institute of Standards and Technology), which has been size-normalized and centered to the fixed-size image. It is based on NIST's Special Dataset 3 (SD-3) and Special Dataset 1 (SD-1). Originally, SD-3 was designated as the training dataset, and SD-1 was designated the test dataset. However, inconsistencies became obvious between SD-1 and SD-3 with SD-3 being considered cleaner and easier to recognize. The reason for this can be traced to the source of the SD-3, which was created from Census Bureau employees, while SD-1 was created from high school students. Early learning experiments using the dataset prompted testers to conclude that the result should be independent of the choice of training set and test among the complete set of samples. The rational conclusion was to build a new dataset by combining samples from both SD-1 and SD-3. MNIST is comprised of 30,000 training samples from SD-1 and 30,000 training samples from SD-3. It also contains 5,000 test samples from SD-1 and 5,000 test samples from SD-3.

MNIST was originally developed by Chris Burgess and Corinna Cortes using bounding-box normalization and centering. The dataset was later adapted by Yann LeCunn using centering by center of mass within a larger window [49]. The LeCunn dataset is what we chose to implement in our testing.

The MNIST dataset is a popular dataset and is good for people who would like to experiment on real data without spending a lot of time with preprocessing and formatting. This

dataset is easy to achieve high accuracy as well as low error in prediction. This is in large part why we have chosen this dataset for testing.

3.2.2 *Fashion MNIST*

The Fashion MNIST dataset was created based upon Zalando's clothing article images. It is comprised of 60,000 training images, 10,000 test images, and 10 classes. MNIST is often the first dataset that researchers attempt to prove out ML models with the thought that if it doesn't work with MNIST, it will fail with other datasets. The truth may be contrary, however, in that if it works well with MNIST, it may still fail with other datasets. Many researchers consider the MNIST dataset to be too easy with even classic ML algorithms able to achieve accuracy of 97%. It is also been stated that MNIST is overused with
Ian Goodfellow, a Google Brain research scientist, calling for people to move away from its use [50].

Fashion MNIST was created to serve as a drop-in replacement of the MNIST dataset as it has the same number of images and classes as well as dimensions. We chose to use this dataset in this work as it is an easy drop-in replacement to MNIST, while still providing for the use of real-world data that has high level of complexity.

3.2.3 *CIFAR10*

CIFAR10 (Canadian Institute for Advanced Research) is a dataset comprised of 60,000 color images representing 10 real-life objects with each object is represented with 6,000 images. The dataset is randomized and broken up into 50,000 training images and 10,000 test images. Each image has dimension of 32x32 and is in RGB color. This means that color is expressed with red,

green, and blue colors with an intensity based on a value from zero to 255. As there are three colors layers, the image dimension is effectively 32x32x3.

The CIFAR10 dataset is a subset of the 80 million tiny images dataset, which were collected by Alex Krizhesvsky, Vinod Nair, and Geoffrey Hinton. The latter is no longer available online as it was reported to contain derogatory terms as categories and offensive images.

The CIFAR10 dataset is commonly used to train computer vision algorithms and is one of the most widely used datasets for ML [51]. It is a good dataset to use for people who do not want to spend time with preprocessing and formatting, and since all images are in color, it provides a different perspective on model robustness obtained through the test methods in this work.

3.2.4 *GTRSB*

The GTRSB (German Traffic Sign Benchmark) dataset is a large dataset comprised of color images of German traffic signs. The dataset boasts 43 classes and more than 50,000 images with various dimensions. For the purposed of testing in this work, we have designated 39,209 images as training images and 12,630 images as test images.

GTRSB was created as part of a single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) held in 2011. It was created to address a lack of datasets which address recognition challenges in real-world problems with high industrial relevance. While the dataset does have classes that closely resemble others, it also provides a wide range of variation between classes in terms of color, shape, imagery, and text. Moreover, images within the same class exhibit variations in appearances due to changes in illumination, rotations, weather conditions, or partial obfuscation [52]. Figure 3.4 contains

examples of all 43 classes of the GTRSB dataset. Notice how some of the images appear to have natural obfuscations such as glare from the sun, blur from the image being captured in motion, or low lighting. These traits lend real-world characteristics to the dataset.

This dataset was chosen for use in this work for many reasons. First, the dataset contains 43 classes, which is a departure from the standard 10-class dataset. Second, the dataset contains color images which will allow for result comparison with the other color dataset, CIFAR10. Lastly, the dataset has implications to recognition challenges in real-world problems with high industrial relevance, namely autonomous vehicles.

Figure 3.4 - German Traffic Signal Recognition Benchmark [52]

3.2.5 *Sign Language MNIST*

Sign Language MNIST is a dataset inspired by the Fashion MNIST dataset, and it was designed to be a drop-in replacement for MNIST. As with Zalando's Fashion MNIST, the motivation for creating MNIST Sign Language was to create a dataset with more complexity to better challenge computer vision and be more applicable to real-world applications. The dataset is a multi-class problem with 24 classes which represent 24 of the 26 letters in American Sign Language. Figure 3.5 provides a visual representation of the dataset. Note that two letters, J and Z, are not included as they require motion to visually convey. The dataset is broken into 27,455 training images and 7,172 test images, which only represents about half that of standard MNIST, but otherwise remaining similar by being 28x28 pixel image with grayscale values from 0 – 255.

The Sign Language images originate from hand gestures of multiple users against various backgrounds, which resulted in 1,704 non-cropped, color images. To extend the dataset, an image pipeline was created through use of ImageMagick that allowed images to be converted to grayscale, resizing, cropping to hands-only, and creating more than 50 image variations based on filters and data augmentation. The filters used include 'Mitchell,' 'Robidoux,' 'Catrom,' 'Spline,' and 'Hermite,' and data augmentation parameters employed include 5% random pixilation, +/-15% brightness/contrast shifts, and rotations of 3 degrees [53].

The Sign Language MNIST dataset was chosen for this work for its potential for real-world applications, its ease-of-use, and due to its non-standard number of classes. Additionally, since the base image set which Sign Language MNIST is based is relatively small and its extended using filters and data augmentation, it will be interesting to compare results against datasets with a larger, unique data source.

Figure 3.5 - Sign Language MNIST Classes [53]

3.3 Attack Algorithms

There exists a variety of methods that could be employed to create an adversarial example. Each method has various characteristics that sets itself apart from the others. These characteristics include knowledge or lack of knowledge of the NN under attack, whether the attack is targeting a class within the scope of the NN classifier, the attack strategy, and the level of performance of the attack. With these characteristics in mind, the approaches developed by Carlini and Wagner [40] and Chen et al. [54] will be used to evaluate the models created in this work.

3.3.1 *Carlini and Wagner*

The Carlini and Wagner attack, or the C&W attack, is a white box attack with both targeted and untargeted attack capability. This powerful and popular attack employs optimization methods [21] to search for very small perturbations and generates high-quality adversarial examples. The

attack is targeted and contains a parameter that allows the user to set the attack confidence. The attack confidence is a scalar value which allows for determination of the best- and worst-case adversarial examples. By using the lowest possible attack confidence, it should be possible to find the lowest L2 distance between the source sample and target class using targeted capabilities. This helps to quantify the vulnerability between classes.

Figure 3.6 is an example of the C&W attack from experiments conducted with ResNet50 on the Fashion MNIST dataset. The figure shows an image from the source class trouser on the left. In the center are the perturbations that were added by C&W, and only to the point where the classification probability for shirt is larger than trouser. The image on the right is the perturbated image of trouser misclassified as a shirt. Note that the image is clearly recognizable as trouser and not shirt. Also, note the precise perturbation and minimal alteration of the image. We provide a similar image perturbated with EAD in the next section.

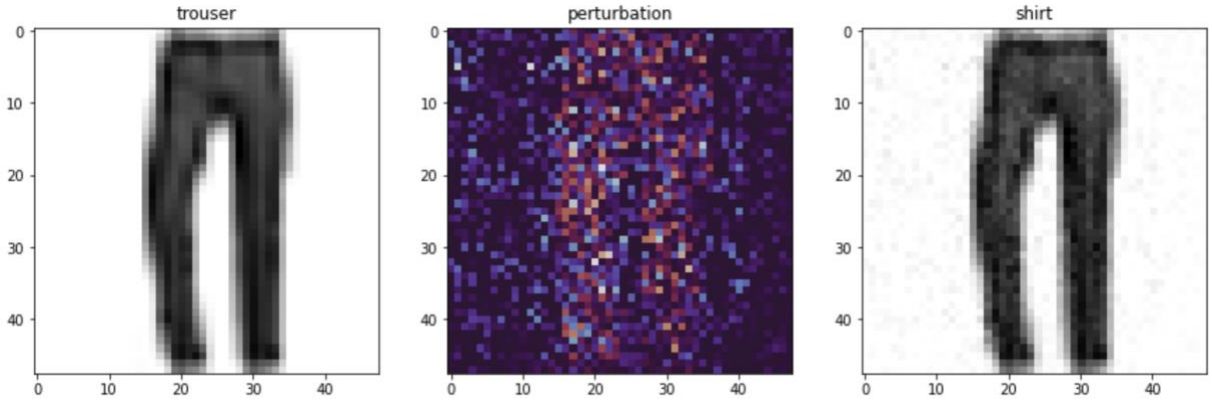

Figure 3.6 - C&W Attack with Fashion MNIST on ResNet50

3.3.2 *Elastic-net Distance*

The Elastic-net Distance attack [54], or EAD attack, is a method of creating adversarial examples that was inspired by the C&W attack. As the name suggests, it uses elastic-net regularization, a combination of penalty functions used for feature selection in high-dimensionality problems, to discover sensitive features. The algorithm is considered a sensitivity analysis attack and has targeted attack capabilities [21].

The EAD attack was chosen for this work for a few reasons. The attack using sensitivity analysis versus optimization algorithms to find sensitive features to perturb, which will allow for study of hyperparameter selection against a different class of perturbations. Algorithmically, the attack is a drop-in replacement for the C&W attack, which allows for easy use within the same code base. This also means the attack is white box and targeted so it can provide the same quantifiable result as the C&W attack. Moreover, the attack parameterizes the attack confidence to allow for determination of best- and worst-case adversarial examples.

The images in figure 3.7 were captured during experimentation with ResNet50 with Fashion MNIST. The images are in the exact same context as figure 3.6 were an image representing trouser is perturbated in such a way to cause a misclassification of shirt. The perturbation is applied just until the probability of the image being a shirt is greater than the image being trouser. Note that the perturbations are much noisier and more irregular than the C&W attack. While the image still clearly shows trouser, the image appears to be altered to a larger degree than with the C&W attack.

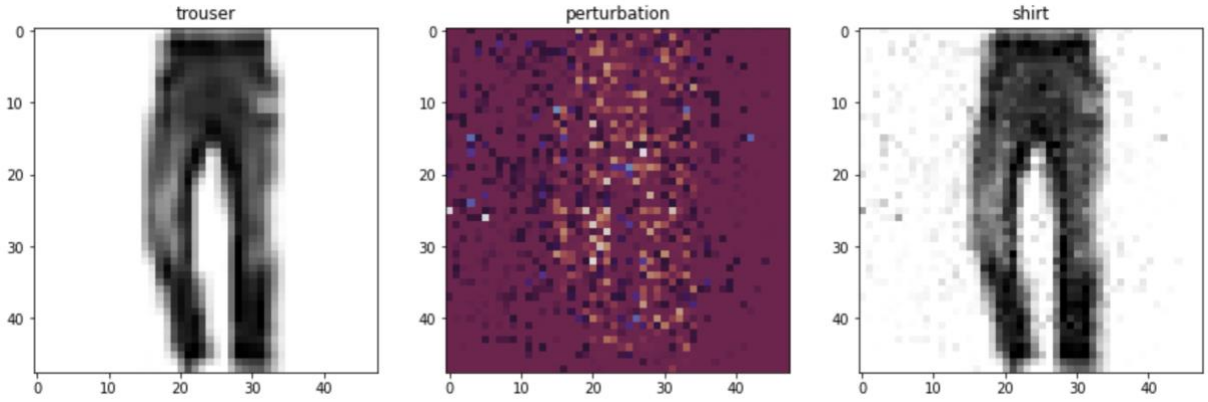

Figure 3.7 - EAD Attack with Fashion MNIST on ResNet50

3.4 METRIC DEFINITION

Many metrics will be collected throughout the course of this work as they will be used to interpolate NN robustness as they relate to a model's composition in terms of hyperparameter selection, network depth, and training.

3.4.1 *Distortion*

Distortion in the context of adversarial examples is the property of an image that describes the deviation of observed pixels from their original coordinates on a 2D plane, and it is measured as a percentage of the pixel deviation from the original image.

3.4.2 *L2 Distance*

Adversarial examples are usually perturbated in a manner so that changes to the original image are not detectable to the human eye. There are various means to measure the degree to which an adversarial image is perturbated. The L2 distance is the Euclidian distance between two images [6]. We will utilize L2 distance in this work as the means to measure the minimum amount of distortion applied to the original image to cause a misclassification.

3.4.3 *Robustness*

Robustness is a term used in this work to describe a property of a NN that prevents it from being misled into incorrectly classifying an image that has been altered with the intent to cause a misclassification.

3.4.4 *Accuracy*

Accuracy is a performance indicator of a NN, and it is a measure of probability for evaluating classification models. Accuracy can be determined by dividing the number of predictions by the total number of predictions [55].

3.4.5 *Achieving Minimum Validation Accuracy*

Achieving the minimum validation accuracy was a problem with some dataset and CNN architecture combinations. The following techniques were put into practice in some cases. This is important as these methods are often used in the real world to achieve desired accuracy in CNN models.

3.4.5.1 Data Augmentation

Image data augmentation is a preprocessing method for expanding a training dataset by creating modified images of the dataset. The Keras DL NN library provides this capability with the ImageDataGenerator class [56]. When this technique is employed, validation accuracy often improves considerably. Although, this technique does not guarantee increased validation accuracy.

3.4.5.2 Early Stopping

Early stopping is a technique where NN training is halted once a monitored metric has stopped improving [57]. This technique was utilized to accomplish two things; to prevent overfitting and to save time. Overfitting is when a NN models the training data to the extent that it negatively impacts the performance of the model. When overfitted, the NN memorizes the various peculiarities of the training data so well that it is unable to find a general predictive rule [58]. Early stopping also saves time by stopping the training of a NN once it stops improving. If training did not cease, the NN would continue to be trained to per its epoch configuration.

3.4.5.3 Learning Rate Scheduler

The purpose of a learning rate scheduler is to adjust the learning rate during training by reducing the learning rate by a pre-defined schedule. It has been well established that adjust the learning rate during NN training can increase the performance of a NN [59].

3.4.6 *Softmax*

Softmax is a function which transforms a vector of real numbers into a vector of probabilities. It takes input that can be positive, negative, zero, or greater than one and converts that input into probabilities between 0 and 1, with the sum of the probabilities being equal to 1 [60].

Chapter 4. RESULTS

4.1 EXPERIMENTAL SETUP

This work examines the impact of hyperparameters on NN model robustness when attacked with C&W attack and EAD attack using a diverse range of publicly available datasets. The depth of this testing, which includes the creation of hundreds of models with varying hyperparameters, will also allow a look into tradeoffs between model accuracy and robustness as it relates to hyperparameters with increased resistance to adversarial examples. Lastly, the breadth of NN models used in this work with an increasing level of convolutional layers will allow for opportunity to quantify model robustness as it relates to NN depth.

4.1.1 *Average Test Sample*

To quantify the robustness of a NN model and compare that result to that of another NN model with a slight change in hyperparameter setting, given the experiments are being conducted with the same dataset and NN architecture, the same set of images samples must be used for generation of adversarial examples.

Not just any samples should be used to conduct these experiments. The samples should be an average representation of each of the classes of the dataset used in the experiment to determine the NN models robustness. We could use a singular dataset class as our sample to perturbate, but that would not provide an accurate measurement of a NN models robustness. The reason for this is that some dataset classes may more easily be perturbated than others to fool a NN model. There are two reasons for this, and they can be summarized as the similarities of images between the classes and the variation of images within the same class. Using the MNIST

dataset as an example, it isn't difficult to imagine that it is easier to perturbate the number 3 so that it gets misclassified as the number 8 than it is to perturbate the number 1 to cause a CNN to misclassify it as the number 8. Moreover, within the MNIST dataset itself, due to variations in the handwritten samples, there may exist samples of the same class that are more easily perturbated than others. Using samples from the MNIST class 1 as illustrated in figure 4.1, it is feasible that the sample furthest to the right may more easily perturbated to fool a CNN to classify it as the number 7 than other samples from the same class.

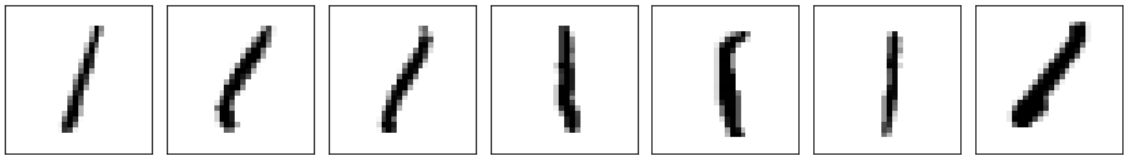

Figure 4.1 - MNIST Samples of Class 1

One option to solve the problem of similarities of images from different classes and variations of images within the same class is to generate attacks using all images in the dataset. Robustness metrics acquired from attacking the CNN could then be averaged across all dataset sample experiments. While this may provide the most accurate account for calculating the robustness of a CNN, it is extremely time- and compute-intensive and falls outside of the time allotted to this work. A more reasonable approach and the approach we take in this work is to find the average test sample for each dataset and NN architecture pair.

The average test sample is found by building 100 CNNs using the default parameters specified in the methods section. These models will serve as a reference point to generate metrics to compare model robustness against for models created with various shifting hyperparameters. Once the CNNs are compiled and trained, the most accurate model is attacked 100 times each using a different class sample set. An exception to this is the Sign Language

MNIST dataset experiments where only 70 attacks and class samples sets where used due to its smaller test sample size. The way these CNNs are attacked, each class is perturbated in such a way, depending on the specific attack methods characteristics, to cause the CNN to misclassify the image to each of the dataset's classes, apart from when the classes are the same. In this work, we will refer to the representant class as the source class and the class being attacked as the target class.

For each of the 100 attack data sample sets, the L2 distances are captured for each source class – target class attack. The mean L2 distance for each source class is then calculated, and from this the adversarial class distance can be found by deriving the mean L2 distance across all 100 attacks for the source class. The average sample can then be determined by finding the sample within the source class set that is closest to the mean L2 distance of the source class. This is easily achieved by finding the absolute value of the minimum L2 distance of the difference of the adversarial class distance and the adversarial sample distance. This process of finding the average sample is followed for all combinations of adversarial attack, dataset, and CNN architecture defined in this work. Figure 4.2 shows the average samples found using the C&W attack, Fashion MNIST dataset, and the LeNet-1 architecture.

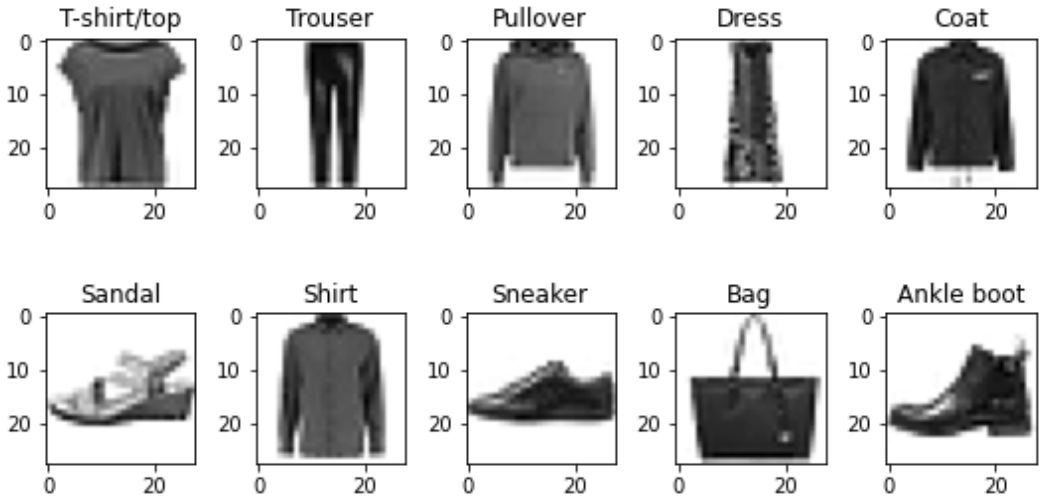

Figure 4.2 – Average Fashion MNIST Samples Derived
From C&W Attack and LeNet Architecture

4.1.2 *Minimum Validation Accuracy*

For an attack to be initiated against a CNN in both the building of the 100 baseline models and the hyperparameter shifting experiments, a minimum validation accuracy must be achieved in the model training process. This qualifier was created to serve two purposes. First, it will save time and resources. Adversarial ML attacks can consume a lot of time and compute resources to accomplish their goal, which in the context of this experiment is a premium considering hundreds of CNN models must be created and attacked to obtain results for one CNN architecture, dataset, and attack tuple. Multiply this by three CNN architectures, five datasets, and two adversarial attack methods, and the test time becomes extensive. The second reason is relevance. When training various CNN architectures while shifting hyperparameters one-by-one, often CNNs achieve poor validation accuracy or do not converge no matter what additional measures are taken, and their use would not align with real world scenario. Attacks on these models were avoided through use of a minimum validation accuracy.

The minimum validation accuracy used in the scope of this work is within 10% of the mean baseline model validation accuracy as listed in table 6.2 in the Appendix. Note the minimum and maximum values in the table and how consistent the values are for LeNet and VGG16, while the values have a wider distribution for ResNet50.

ResNet50 proved to be a challenging architecture to generate consistently accurate baseline models. This is apparent by referencing the accuracy for the Fashion MNIST dataset as well as the range of accuracies gathered for GTRSB in figure 4.3. He et al. [48] were able to show in their work that adding more layers resulted in an increase in training and test error. However, they were unable to offer an explanation to this. Due to the amount of time and resources needed to build the models necessary to conduct the experiments in this work, the accuracy of these datasets against this architecture were deemed acceptable.

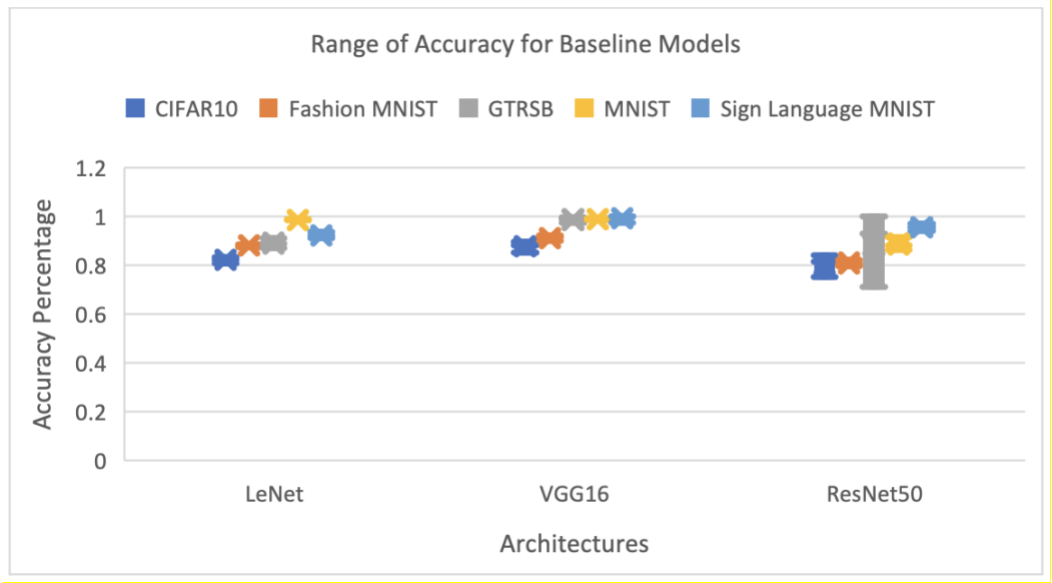

Figure 4.3 – Plot of Minimum, Mean, and Maximum Accuracy for Baseline Architectures

4.1.3 *Hyperparameter Testing*

To determine the impact of a specific hyperparameter, CNNs are created each with the tuning of only a single parameter with the hyperparameters class. Using the hyperparameter class activation unit as an example from table 6.3 in the Appendix, a CNN is created for each of the five parameters, elu, linear, relu, sigmoid, and tahn, within the hyperparameter class. The CNN with tuned hyperparameter is then attacked and the L2 distances for each source class – target class pair are captured. This will allow for comparison of characteristics, namely robustness, between the CNN with tuned hyperparameter and the CNN baseline models created for each dataset and CNN architecture pair. From the L2 distances recorded, the mean L2 distance, the minimum L2 distance, and the maximum L2 distance are derived as well as the standard deviation of these measurements.

4.1.4 *Generating Metrics*

To quantify the robustness of a NN, a standard deviation, a measure of dispersion relative to its mean [61], is generated that is specific to the CNN, dataset, and attack combination. To gather the data required to calculate this metric, the 100 baseline models are attacked, and with each attack, the L2 distance for each source and target class are recorded, as well as the adversarial image and the image distortion. Using the recorded L2 distances, we then calculate the mean L2 distance, the minimum L2 distance, the maximum L2 distance, and their mean value across 100 baseline model attacks. The standard deviation is then found for the mean of the mean L2 distances, the mean of minimum L2 distances, the mean maximum L2 distances, and the mean of the standard deviations. The total standard deviation for the baseline models is then calculated as the sum of each of these standard deviation values. It should be noted that the number of models

used to determine the mean L2 distances of the baseline models was increased in comparison to the original work.

Table 6.5 in the Appendix shows the standard deviations of the mean distance, minimum distance, maximum distance, and the standard deviation of the L2 distance metrics. Note that the values are quite small for the LeNet architecture models meaning that characteristics of the LeNet baseline models were very similar. However, as the architectures become more complex, as is the case with VGG16 and ResNet50, the characteristics of the baseline models become more diverse, especially when the attack was EAD.

Expanding upon the minimum L2 distance and the maximum L2 distance, they can each be thought of in terms of source class and target class. The minimum L2 distance represents a class which is most likely very similar to the target class and thus requires a minimum amount of perturbation to generate a misclassification. The maximum L2 distance represents a source class that is much different than the target class and thus requires a high amount of perturbation to cause a misclassification. An example of minimum L2 distance using the MNIST dataset may be the minimal amount of perturbation required to make a 1 misclassify as a 7. On the other hand, an example of maximum L2 distance again using the MNIST dataset may be the large amount of perturbation required to misclassify a 1 as an 8.

4.1.5 *Evaluating Hyperparameter Impact*

The measure of the impact a hyperparameter has is determined using the standard deviation from the baseline models. To find the number of standard deviations that are present in the shifted hyperparameter CNN, for each L2 distance metric as well as its standard deviation, the difference of the baseline model L2 distance metrics and the shifted hyperparameter L2 distance metrics are divided by the respective baseline model L2 distance metric standard deviation. The

quotients are then summed resulting in a total standard deviation value, which can be compared to the total number of standard deviations present in the baseline model.

To convey hyperparameter impact, we rely on discrete valuations of high, medium, and low. These values are used to categorize and provide reasonable separation between a broad range of values, so they are more easily consumed versus standard deviations of L2 distance. For an impact to be regarded as high, the shifted hyperparameter classes' maximum total standard deviation must be at least eight standard deviations above the baseline or higher. For an impact to be considered medium, the number of shifted-hyperparameter classes' maximum standard deviations must be at least four and less than eight. An impact designation of low means the shifted-hyperparameter classes' maximum standard deviation was equal to the baseline model baseline and less than four.

It should be noted that the method used in this work to evaluate hyperparameter impact is an improved method based on that created by Burkard and Lagesse [8]. In their work, hyperparameter impact was determined by calculated the standard deviation of the total standard deviations for each hyperparameters within the hyperparameter class. We have determined that it is better to use the maximum value of the total standard deviation of the hyperparameter class to determine its impact for two reasons. First, while their method does indeed work for finding hyperparameter classes that do have higher impact over the baseline models, it only does so for hyperparameter classes that have high variability within the total standard deviation measure of the individual hyperparameters within the class. If the total standard deviation is higher than the baseline models range, but the variability of the total standard deviations is low, than the impact asserted for this hyperparameter class may also be below the threshold. When the maximum total standard deviation is used, this problem is

eliminated. Second, when the standard deviation of the total standard deviation of a hyperparameter class is used to gauge impact, the impact metric is abstracted an additional layer from the original unit of measure, the standard deviation of the baseline model. Using the maximum total standard deviation from the hyperparameter class preserves the metric, which is more intuitive.

4.2 HYPERPARAMETER IMPACT

The following section summarizes the results of impact of hyperparameter selection on CNN robustness. Experiments were conducted using two well-known evasion attacks on three popular CNN architectures using five publicly available datasets with varying characteristics. Due to the amount of information to convey in this section, it will be organized by attack-CNN architecture.

4.2.1 *LeNet – C&W*

The most consistent performing hyperparameter classes in the experiment series using the LeNet architecture and the C&W attack were bias initialization and max pooling filter. Each of these hyperparameter classes had at least one hyperparameter that provided increased robustness over the baseline model baseline in all the five datasets tested. Many of the bias initializers showed increased robustness in some of the datasets, but they failed to consistently show higher mean L2 distances when compared to the baseline model baseline with one exception, the ‘ones’ bias initializer. The ‘ones’ bias initializer hyperparameter showed highly elevated mean L2 distance to the factor of 100 in all datasets. For max pooling filter size, the best performing max pooling filter size was inconsistent. The LeNet CNN performed best with the CIFAR10 and Sign Language MNIST datasets when using a 4 x 4 filter while the Fashion MNIST and GTRSB datasets application to the CNN produced better results using 5 x 5 and 2 x 2 filter size

respectively. The mean L2 distances were the highest when the CNN max pooling filter was 6 x 6 for MNIST.

The hyperparameter classes activation unit and batch size also scored highly showing improved mean L2 distance in three out of five datasets. The activation unit which produces the highest mean L2 distance is the sigmoid function, and it consistently achieved high marks for all datasets tested in this series of experiments.

Improvements in mean L2 distance attributed to batch size generally favored 128 with the 2 out of 3 datasets where is scored highly showing the most improvement. The exception to this is with MNIST, where the highest mean L2 distance achieved was with using the highest batch size tested, 512.

Table 4.1 - Comparison of Hyperparameters with Impact in Majority of Datasets in LeNet-C&W Experiments

LeNet – C&W						
Parameter	Specific	Dataset Mean L2 Distance				
		CIFAR10	Fashion MNIST	GTRSB	MNIST	Sign Language
LeNet Model	NA	0.155	0.581	0.558	1.244	0.244
Activation Unit	Elu		0.228		1.061	0.161
	Linear		0.282		1.502	0.123
	Relu		0.389		0.470	0.249
	Sigmoid		143.745		167.714	135.571
	Tahn		0.389		0.943	0.140
Batch Size	32		0.259	0.452	0.976	
	64		0.938	0.354	0.773	
	128		1.306	1.184	1.861	
	256		0.449	0.603	1.464	
	512		0.825	0.456	2.026	
Bias Initializer	Variance Scaling	0.157	1.047	0.824	1.513	0.775
	Ones	35.166	93.826	177.635	65.256	83.322
	Random Uniform	0.127	0.535	0.805	2.982	0.340
	Truncated Normal	0.214	1.332	0.586	2.330	0.366
	Zeros	0.106	0.383	0.914	1.318	0.109
Max Pooling Filter Size	2 x 2	0.097	1.044	1.488	1.082	0.191
	3 x 3	0.265	0.544	0.534	2.652	0.231

	4 x 4	0.388	0.840	0.840	5.876	0.971
	5 x 5	0.112	1.501	NA	2.781	NA
	6 x 6	0.218	1.125	NA	6.449	NA

4.2.2 *LeNet – EAD*

Results for the experiment series conducted with the LeNet architecture and the EAD attack were very similar to that of the prior series of experiments. The most consistent performing hyperparameter classes were bias initializer and max pooling filter with each hyperparameter class having a minimum one hyperparameter that provided increased robustness over the baseline model baseline in at least four of the five datasets tested. The best mean L2 distance scores for the bias initializer class were the 'ones' initializer achieving a mean L2 distance 100 times that of the baseline model baseline in most of the datasets. Some of the other bias initializers did show some improvement, but it did not show consistency across datasets. The max pooling filter size improved for the mid-to-larger filter sizes in the range of 4 x 4 to 6 x 6. When compared to the C&W attack on LeNet, the only common result is the 5 x 5 filter with MNIST.

The hyperparameter classes activation unit and strides also performed well in this series of experiments with both generating robust results in three out of five datasets tested. Sigmoid again showed the highest levels of mean L2 distance achieving a mean L2 distance of 216 against the MNIST dataset. In comparison, the mean L2 distance average for the baseline models is 1.2078.

The result for strides is like that of max pooling filter size where mid-to-larger strides showing improved adversarial resistance to that of smaller strides.

Table 4.2 - Comparison of Hyperparameters with Impact in Majority of Datasets in LeNet-EAD Experiments

LeNet - EAD						
Parameter	Specific	Dataset Mean L2 Distance				
		CIFAR10	Fashion MNIST	GTRSB	MNIST	Sign Language
LeNet Model	NA	0.278	0.498	0.724	1.208	0.254
Activation Unit	Elu		0.915		1.045	0.188
	Linear		0.486		0.930	0.259
	Relu		0.290		0.359	0.244
	Sigmoid		203.386		216.025	28.440
	Tahn		0.482		0.619	0.273
Bias Initializer	Variance Scaling	0.210	1.109	0.825	1.363	0.355
	Ones	8.782	132.179	146.623	120.423	114.385
	Random Uniform	0.420	0.823	0.494	0.853	0.219
	Truncated Normal	0.352	1.766	0.507	3.143	0.593
	Zeros	0.221	0.761	0.964	2.670	0.126
Max Pooling Filter Size	2 x 2	0.356	1.360	0.669	1.608	
	3 x 3	0.284	0.303	0.960	1.785	
	4 x 4	0.277	0.812	1.471	0.917	
	5 x 5	0.488	2.213	NA	4.821	
	6 x 6	0.657	1.316	NA	4.050	
Strides	1	0.273	0.510		0.861	
	2	0.576	1.166		2.062	
	3	NA	0.616		3.713	

4.2.3 *VGG16 – C&W*

The most consistent performing hyperparameter class in the series of experiments conducted with VGG16 and the C&W attack are bias initialization, convolutional kernel size, and max pooling filter size with each class acquiring above baseline model baseline mean L2 distance for four out of five datasets. The bias initializer ‘ones’ outperformed the other hyperparameters within the class for all datasets and reached its highest value of mean L2 distance at over 221 when the GTRSB dataset was applied to the architecture and attack pairing. This is a magnitude of 10 higher than the mean L2 distance recorded for the baseline models. Increased adversarial

resistance favored smaller convolutional kernel size with a value of 2 x 2 achieving the highest mean L2 distance for all models meeting the testing criteria. The max pooling hyperparameter achieved its highest security rating at a size of 5 x 5 for all dataset meeting the requirements for attack.

The number of dense nodes when there are two hidden layers, which is the default for VGG16, also showed some reactivity in this experiment series for three out of five datasets. Increased CNN robustness occurred with the number of dense nodes was 512 for Fashion MNIST and MNIST, but the mean L2 distance was higher for Sign Language MNIST at 4,096.

Table 4.3 - Comparison of Hyperparameters with Impact in Majority of Datasets in VGG16-C&W Experiments

VGG16 – C&W						
Parameter	Specific	Dataset Mean L2 Distance				
		CIFAR10	Fashion MNIST	GTRSB	MNIST	Sign Language
LeNet Model	NA	3.7054	5.7549	26.6472	10.3353	6.5593
Bias Initializer	Variance Scaling		4.817	27.349	26.935	10.658
	Ones		71.793	221.794	90.170	53.048
	Random Uniform		10.917	25.440	5.930	8.384
	Truncated Normal		2.861	36.335	3.178	3.450
	Zeros		2.592	24.100	7.848	7.420
Dense Nodes (Two layers)	512		12.949		28.441	4.754
	1024		4.035		10.033	2.436
	2048		3.738		1.145	4.506
	3072		2.178		7.038	11.070
	4096		5.083		6.637	14.852
Convolutional Kernel Size	2 x 2	8.395	15.455		38.531	17.633
	3 x 3	2.406	2.936		10.022	6.811
	4 x 4	6.468	1.866		21.160	6.640
	5 x 5	3.208	3.238		1.219	8.238
	6 x 6	5.636	4.500		24.581	6.628
	7 x 7	7.179	3.962		10.464	6.932
	8 x 8	3.064	1.530		4.482	7.987
	9 x 9	3.018	1.634		16.467	5.550
Max Pooling Filter Size	2 x 2	1.724		21.838	2.938	7.740
	3 x 3	7.599		32.231	8.898	25.377

	4 x 4	14.665		26.706	10.339	16.630
	5 x 5	34.773		160.344	33.527	58.064
	6 x 6	7.358		23.620	10.278	5.156

4.2.4 *VGG16 – EAD*

The results for the series of experiments using the VGG16 architecture and EAD attack provided limited results with only one dataset showing a marked improvement over the baseline. Experiments with the GTRSB dataset showed that the kernel initialization class and max pooling filter size class improved mean L2 distance over the baseline. The kernel initializer 'truncated normal' offered the highest increase in mean L2 distance achieving a value just over 1284, which is less than twice that in comparison with the baseline. In the max pooling filter size class, a smaller filter size of 3 x 3 offered the best improvement of the class. This value is just slightly elevated in comparison with the baseline model baseline.

Table 4.4 - Comparison of Only Hyperparameters with Impact in Datasets in VGG16-EAD Experiments

VGG16 – EAD						
Parameter	Specific	Dataset Mean L2 Distance				
		CIFAR10	Fashion MNIST	GTRSB	MNIST	Sign Language
LeNet Model	NA	735.1307	337.3082	788.2167	427.0382	265.7694
Kernel Initializer	Glorot Uniform			2.501		
	HE Normal			859.477		
	Orthogonal			5.577		
	Random Uniform			520.318		
	Truncated Normal			1284.336		
	Zeros			NA		
Max Pooling Filter Size	2 x 2			743.530		
	3 x 3			901.814		
	4 x 4			571.923		

	5 x 5		578.056		
	6 x 6		261.039		

4.2.5 *ResNet-50 – C&W*

Bias initialization had the most consistent result in the ResNet50 and C&W series of experiments accruing above L2 distance baseline in four out of five datasets. Of the bias initializers, 'ones' consistently had the highest resilience to the C&W attack.

Activation, loss, and strides also showed above L2 distance baseline scores in two out of three datasets. Of the activation units, Relu scored the best in both CIFAR10 and MNIST datasets. Sigmoid activation was out of scope in this experiment as models trained with this activation did not meet minimum validation accuracy. Parameters within the loss hyperparameter were inconsistent with sig cross having the best score for Fashion MNIST and Cos having the best score for MNIST. The stride parameter was consistent between Fashion MNIST and MNIST with a single stride achieving the best score.

Table 4.5 - Comparison of Hyperparameters with Highest Impact in Datasets in ResNet50-C&W Experiments

ResNet50 – C&W						
Parameter	Specific	Dataset Mean L2 Distance				
		CIFAR10	Fashion MNIST	GTRSB	MNIST	Sign Language
LeNet Model	NA	9.376	38.54	27.08	18.715	43.478
Activation Unit	Elu	3.654			2.634	
	Linear	NA			0.665	
	Relu	33.918			77.894	
	Sigmoid	NA			NA	
	Tahn	NA			0.021	
Bias Initialization	Variance Scaling	5.070	42.292	23.089	4.846	
	Ones	292.425	241.447	225.379	93.647	
	Random Uniform	22.673	43.199	15.938	21.700	

	Truncated Normal	16.581	43.217	20.235	7.505	
	Zeros	3.123	19.262	41.005	42.809	
Loss	Sig Cross		94.878		12.158	
	Hinge		46.119		19.367	
	Log Loss		15.685		3.826	
	Mean Pair		2.224		14.099	
	MSE		12.933		34.870	
	Cos		22.291		54.458	
Strides	1		233.312		45.240	
	2		44.218		20.106	
	3		25.223		16.135	

4.2.6 *ResNet-50 – EAD*

Results for the ResNet50 architecture and EAD attack are limited with only one dataset showing better than L2 distance baseline scores for L2 mean distance. HE normal kernel initialization and convolutional kernel size of 6 x 6 both showed slightly elevated mean L2 distance for the MNIST dataset.

Table 4.6- Comparison of Only Hyperparameters with Impact in Datasets in ResNet50-EAD Experiments

ResNet50 – EAD						
Parameter	Specific	Dataset Mean L2 Distance				
		CIFAR10	Fashion MNIST	GTRSB	MNIST	Sign Language
LeNet Model	NA	349.179	479.624	339.771	174.379	358.051
Kernel Initializer	Glorot Uniform				97.526	
	HE Normal				209.453	
	Orthogonal				62.557	
	Random Uniform				26.037	
	Truncated Normal				108.907	
	Ones				NA	
Convolutional Kernel Size	2 x 2				235.883	
	3 x 3				159.326	
	4 x 4				123.898	
	5 x 5				111.436	

	6 x 6				269.109	
	7 x 7				245.353	
	8 x 8				58.390	
	9 x 9				142.448	

4.2.7 *Robust Hyperparameters*

The following section explores the results of hyperparameter experiments across all experiment sets from the prior section, and it is broken up into three subsections: consistent, less consistent, and inconsistent. This information is summarized in table 6.7 in the Appendix.

4.2.7.1 Consistent

The four hyperparameters that showed improvements in mean L2 distance the most consistently are activation, bias initialization, convolutional kernel size, and max pooling filter size. Consisting solely of network hyperparameters, these hyperparameters are deemed best not only in terms of impact consistency across all of experiments, but they had the most reliability in terms of setting. Table 4.7 captures the results of the consistent hyperparameters across all experiments.

Activation displayed high impact in 15 out of the 30 dataset/architecture/attack experiment combinations. Of those 15 impact results, 14 were ranked as high and the with the remaining impact ranked as low. Activation disproportionately had an affect with datasets Fashion MNIST, MNIST, and Sign Language MNIST occurring four times. A single occurrence of increased robustness also occurred with CIFAR10. Activation had in impact with either attack being used, and it allowed for improved mean L2 distance in LeNet and ResNet50 architectures. Moreover, with ResNet50, attacks were unsuccessful, even when the attack

confidence was increase incrementally from 1 to 5 for each attack. Initially, the results were interpreted as unsuccessful as extremely low L2 distances were recorded (approaching zero), yet the distortion returned from the attack was extremely high. Re-evaluation of these results thus conclude that the attacks failed (additional tests with attack confidences higher than 5 were not performed due to time constraints). Figure 4.4 illustrates this point and shows that distortion increases near linearly, yet each of L2 distances drops to zero. No impact was recorded with activation and the VGG16 architecture. Generally, the most common activation unit recording impact was Sigmoid, although RELU had impact in the single result with CIFAR10. At closer inspection of the ReLU activation result set, Sigmoid did provide for higher resistance to adversarial examples, but as the validation accuracy was low for Sigmoid activation, the result Sigmoid result was excluded.

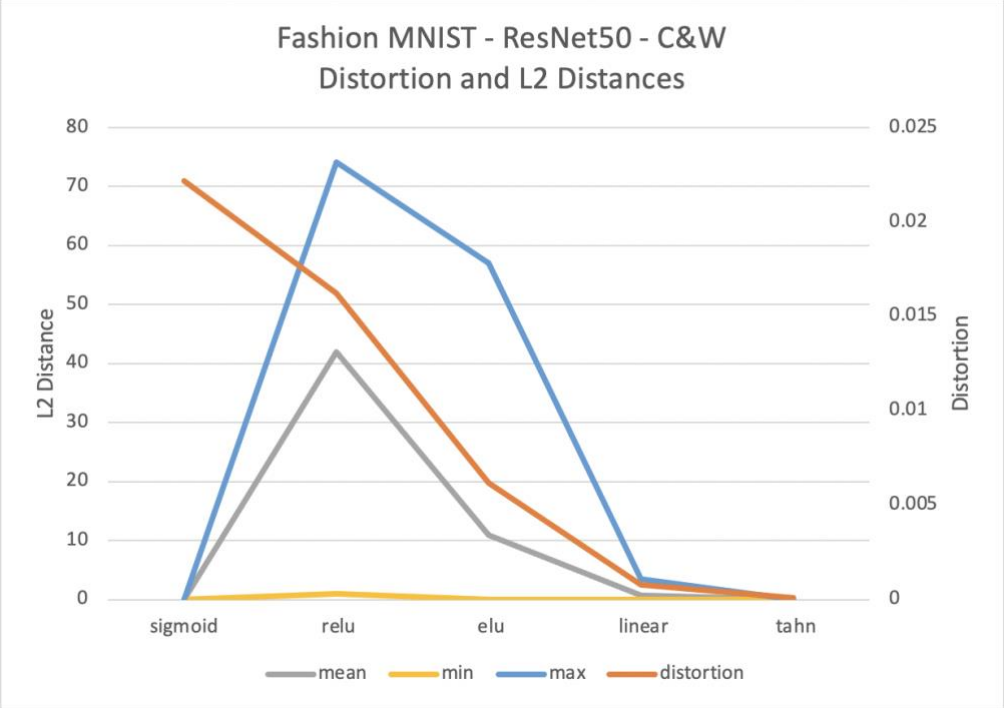

Figure 4.4 – Sigmoid L2 Distance Extremely Low When Compared with Increasing Values of Image Distortion

The impact of bias initialization on robustness was high in all experiments where an elevated mean L2 distance was recorded, which was 18 out of a total of 30 dataset/architecture/attack tuples. High impact occurred with all datasets at least once, in all architectures, and using both C&W and EAD attacks. The only experiments where 'ones' bias initialization did not have impact is with EAD attacks against VGG16 and ResNet50 architectures. Each instance of elevated mean L2 distance occurred when the bias initialization 'ones' parameter was used. This makes the bias initialization 'ones' parameter the most robust hyperparameter tested to improve CNN resistance to adversarial examples.

Max pooling filter size was the next most consistent hyperparameter with high impact in 11 experiments and low impact in three experiments. All elevated impacts occurred at least once against all datasets and with both the C&W and EAD attacks. Interestingly, max pooling filter size only improved mean L2 distance against the LeNet and VGG16 architectures. The max pooling filter size was not perfectly consistent across all results, however. Although, the most common filter size which improved resilience was 4 x 4 and 5 x 5.

Convolutional kernel size displayed impact in nine of the 30 dataset/architecture/attack tuples. Two of the impacts were rated as high, two as medium, and five as low. Impacts occurred at least once against all datasets, in all three architectures, and in with both the C&W and EAD attack, with the exception being the EAD attack against VGG16. The convolutional kernel size was mixed amongst impacted experiments with sizes 2 x 2, 3 x 3, 4 x 4, and 5 x 5 being recorded. The filter size with the highest occurrence was 2 x 2 making up more than half of results showing impact. Additionally, the convolutional kernel size 2 x 2 was common against all results with the VGG16 architecture.

Table 4.7 – Hyperparameters Which Consistently Had Impact Across All Experiments

Parameter	C&W																EAD															
	LeNet					VGG16					ResNet50						LeNet					VGG16					ResNet50						
	CIFAR10	Fashion MNIST	GTRSB	MNIST	Sign Language	CIFAR10	Fashion MNIST	GTRSB	MNIST	Sign Language	CIFAR10	Fashion MNIST	GTRSB	MNIST	Sign Language		CIFAR10	Fashion MNIST	GTRSB	MNIST	Sign Language	CIFAR10	Fashion MNIST	GTRSB	MNIST	Sign Language	CIFAR10	Fashion MNIST	GTRSB	MNIST	Sign Language		
Activation Unit		***		***	***						*	***	***	***	***			***		***	***							***	***	***	***		
Bias Initializer	***	***	***	***	***		***	***	***	***	***	***	***	***			***	***	***	***	***												
Convolutional Kernel Size		*			**	*	*		**	*			***					*												***		
Max Pooling Filter Size	*	*	*	***	***	***		***	***	***							***	***	***	***				***									

4.2.7.2 Less Consistent

The next group of hyperparameters which showed an increased level of adversarial resistance are the training hyperparameters strides, batch size, momentum, loss, and the network hyperparameter single layer dense nodes. These hyperparameters consistently displayed impact, although less consistently than the prior group, yet no pattern could be discerned to predict which hyperparameter setting produced the best impact or in what dataset/architecture/attack combination it performed best.

Strides showed increased mean L2 distance in seven of the 30 dataset/architecture/attack tuples experiments. One of the impacts ranged as high, while two ranked as medium impact, and four impacts were rated as low. Strides had impacts with experiments with all architectures and with both the C&W and EAD attacks. Strides mostly impacted experiments with Fashion MNIST and MNIST with each occurring three times. One instance of impact was also recorded with the CIFAR10 dataset, although the impact was low. Stride values of one and two correlated with impact with the C&W attack, while stride values of two and three showed improved adversarial resistance with the EAD attack.

Batch size showed low impact in both the LeNet and VGG16 architectures. However, impact was disproportionately weighted towards the LeNet architecture with only one

occurrence in the VGG16 architecture. No impact results were shown for the ResNet50 architecture. All datasets experienced elevated mean L2 distance with batch size at least once with Fashion MNIST being the only dataset to elicit a result twice. The low impact was recorded with both the C&W and EAD attack. Although, the C&W attack recorded a result twice as many times as the EAD attack. It is also worth noting that the only impact result occurring on the VGG16 architecture, was recorded with the C&W attack. Batch size was inconsistent across experiments recording an impact with size 32 being recorded most frequently. Batch size 128 was recorded twice, and batch size 512 recorded once.

The training hyperparameter momentum showed modest mean L2 distance with one high, two medium, and two low impact results. Impact results occurred when either the C&W or EAD attack was employed. Of the five datasets tested, only Fashion MNIST, GTRSB, and Sign Language MNIST showed any improvements with shifts in the momentum hyperparameter. Moreover, the ResNet50 architecture did not show additional mean L2 distance leaving only LeNet and VGG16 recording impact. Momentum values varied from experiment to experiment with no clear pattern in their demonstration of high adversarial resistance.

Loss function met the minimum criteria to have results referenced in this section with four elevated L2 distance results. One of the results was high impact with the C&W attack against the LeNet architecture and Fashion MNIST dataset. The other results were of low impact. Various loss functions assisted to achieve these results and are MSE, cos, and sig cross. Of these loss functions, only cos occurred more than once. It’s worth noting that loss function only had an impact against the LeNet and ResNet50 architectures and the Fashion MNIST and MNIST datasets.

The number of dense nodes with a single hidden layer is the last hyperparameter that appeared to show moderate consistency across all experiment dataset/architecture/attack tuples with one medium impact and 4 low impact results. These results were recorded with both the C&W and EAD attack, and they occurred with both LeNet and VGG16 architectures. No impact was shown with the ResNet50 architecture. Of the datasets tested, only Fashion MNIST, MNIST, and CIFAR10 showed that the number of dense nodes with a single hidden layer made and improvements to robustness. Impacts were shown with both MNIST and CIFAR10 twice, while they were only recorded with Fashion MNIST once. There was not a consistent number of dense nodes that provided increased robustness with the values ranging from 512 to 4096 although 512 did occur more than once.

4.2.7.3 Inconsistent

This final group of hyperparameters showed impact in three or less of the experiments conducted. The hyperparameters that make up this group are activity regularization L2, number of dense layers, number of dense nodes in two layers, dropout on dense layer, dropout on input layer, filters, kernel initialization, kernel regularization L1, kernel regularization L2, optimization, and dense nodes at three layers.

Activity regularization showed impact in two experiments both using EAD attack and the LeNet architecture. The hyperparameter setting was inconsistent between impact results with Fashion MNIST and MNIST datasets. Three impact results were recorded in the experiments shifting the number of dense layers, and they were low in each case. Two impact results with five layers occurred against CIFAR10 once with C&W against VGG16 and again with the EAD against LeNet. Three layers had low impact with EAD, LeNet, and Fashion MNIST. Three low impact results occurred with number of dense nodes in two layers with C&W and VGG16 with

512 nodes improving Fashion MNIST and MNIST and 4096 nodes improving Sign Language MNIST. Two low impact results occurred with dropout on dense layer with 40% and 20% logged against C&W, VGG16, and CIFAR10 and EAD, LeNet, and Fashion MNIST experiments respectively. Three results were recorded with dropout on input layer all occurring against C&W attack. Two medium impacts were recorded with VGG16 and CIFAR10 and MNIST at 10% and 40% respectively. A low impact result of 30% occurred with ResNet50 and CIFAR10. A single low impact result was recorded with filters per CNN with 512 filters having impact against EAD, LeNet, and Fashion MNIST. Kernel initialization is interesting while it only showed three results, all three were high impact and with the EAD attack. He normal had impact with LeNet and Fashion MNIST, random uniform worked well with VGG16 and GTRSB, and random uniform had impact with ResNet50 and MNIST. Kernel initialization L1 had a single low impact result with C&W, ResNet50, and Fashion MNIST. Kernel regularization L2 had three medium impact results each occurring with Fashion MNIST. Setting were nonuniform with .08 on C&W and LeNet, .1 with C&W and ResNet50, and .06 with EAD and LeNet. Three optimization results were recorded with the C&W attack. Two low impact results with adam and adamax VGG16 and Fashion MNIST and Sign Language MNIST respectively. A medium impact result was recorded with adam and LeNet and MNIST. Number of dense nodes at three layers had two results with 1024 nodes. The first is a high impact result with C&W, VGG16, and CIFAR10, and the second being a low impact result with EAD, LeNet, and GTRSB dataset.

4.2.7.4 Ones Bias Initialization

Goodfellow et at concluded, through the application of the universal approximator theorem, that only structures with a hidden layer should be trained to resist adversarial perturbations [29]. Li et al. made note of an issue called 'dying RELU' [62] where a RELU unit always outputs zero, and when the RELU unit enters this state, it is unlikely to recover as the function gradient at zero is also zero preventing gradient descent learning from altering the weights. Li et al. suggested a trick to overcome this issue by initializing the bias with a small constant value such as 0.01. This ensures the RELU unit fires on initialization preventing 'dying' or 'dead' RELU. Moreover, Jozefowicz et at [63] state that LTSM (Long Short-term Memory) commonly initialize the bias with ones.

Considering that all the NN architectures tested in this work by default use RELU activation and initialize the bias with zero, it is our intuition that nodes within the hidden layer are subject to dying RELU, and that the initialization of the bias with a constant value one may prevent this, which allows for more active neurons increasing the robustness of the model. To test this intuition, we created five new LeNet models using ones bias initialization and an increasing number of nodes, from 100 to 2000, in the hidden layer. If in the experiment result we see increased L2 distances as the nodes increased, we can conclude that more active nodes provides increased robustness against adversarial examples. We can also compare this result against the L2 distance trend in the hyperparameter shifting experiment where hidden layer nodes were increased against the default LeNet configuration with zeros bias initializations.

The two graphs in figure 4.5 illustrate the trend of increased mean L2 distance as the number of nodes increases in the hidden layer when ones bias initialization is used along with RELU activation. The L2 distance is represented by the y-axis while the x-axis represents the

number of nodes. While the max L2 distance does not appear to change much, the minimum L2 distance substantially increases with the increase in nodes in the hidden layer, especially with the EAD attack. The graphs in figure 4.5 clearly show that the increased number of nodes in the hidden layer, with the ones bias initialization, provide for more resistance to adversarial perturbations. The two graphs in figure 4.6 have bias initialization of zero, and they may be subject to dying RELU as they show flat to subtle decrease in mean L2 distance as the number of nodes increase. They do not appear to be resistant to adversarial examples at all as the L2 distances in every respect, minimum, mean, and maximum, are very low.

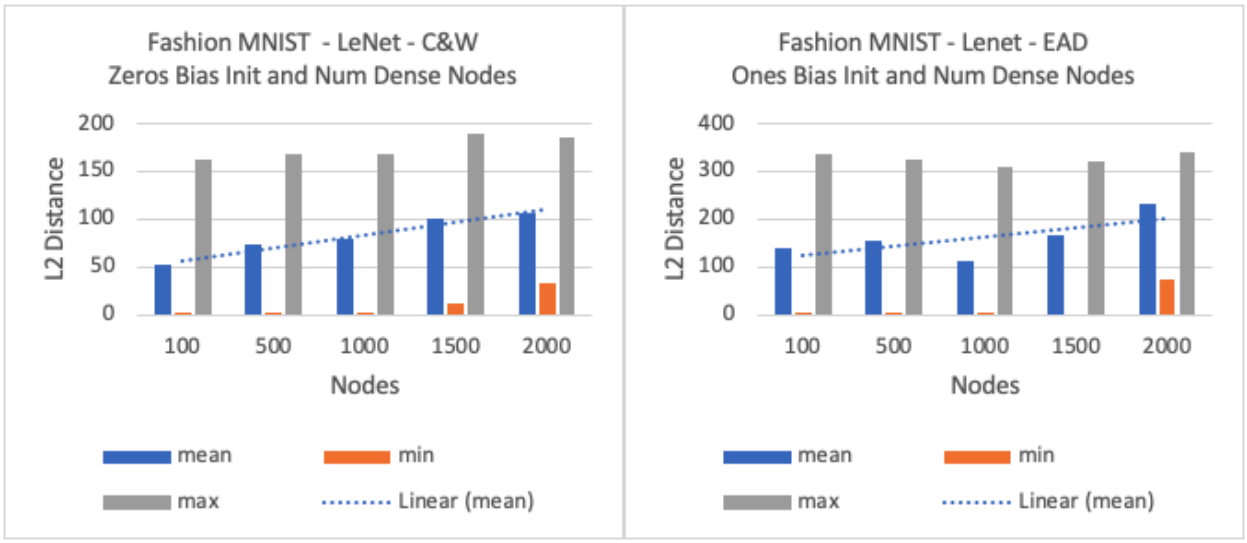

Figure 4.5 – Trend of Mean L2 Distance Increases When Nodes Increase with Ones Bias Initialization

Figure 4.6 - Trend of Mean L2 Distance Decreases When Nodes Increase with Zeros Bias Initialization

4.2.7.5 Sigmoid Activation

Sigmoid activation showed the highest levels of adversarial resistance in both the LeNet and ResNet models in our experiments when combined with the C&W attack. This observation is true when comparing mean, minimum, and maximum L2 distance as well as image distortion. Adversarial resistance was also high with LeNet and ResNet50 models in combination with the EAD attack, but in some experiments zeros bias initialization provided for the highest adversarial resistance in at least one measure.

An explanation to why Sigmoid activation has high levels of adversarial resistance was not easy to obtain. Many experiments were conducted exploring linearity, node activation, etc. Our intuition, however, is that the mechanism behind Sigmoid activation's high adversarial resistance has to do with the small derivative it produces. This small derivative is a product of the condensing of input into the space between 0 and 1, which produces a curious property such that large changes in the input result in a small change in the output. This explains the smooth

gradient in figure 4.7. Notice how details of the pullover are much harder to discern in comparison to activation functions such as RELU. Further explanation into what causes Sigmoid activation's adversarial robustness may serve as a subject for future work.

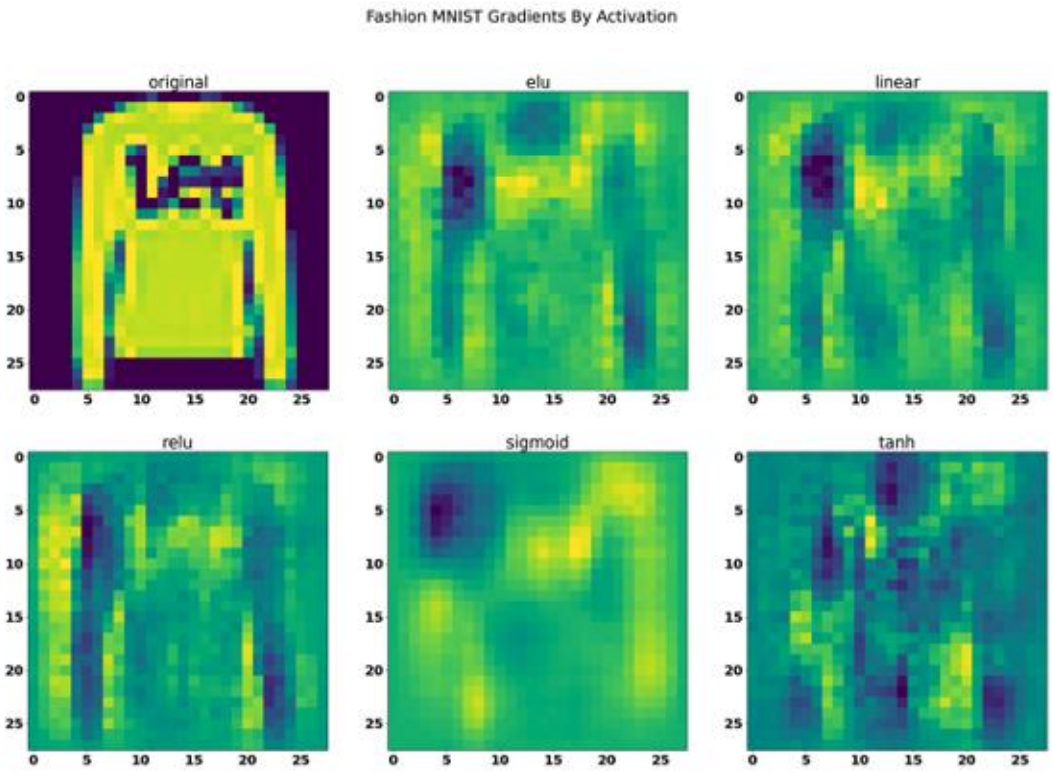

Figure 4.7 - Fashion MNIST Gradients Produced by Each Tested Activation Function Based on Test Image of Class Pullover

4.2.7.6 VGG16 and EAD Attack

Examining table 4.7, it is obvious that not many hyperparameters had effect on the robustness of the VGG16 architecture when the EAD attack was used regardless of dataset. While Sigmoid activation did provide ample robustness in other experiments, it failed to converge and provide a decent level of accuracy when building the VGG16 models and thus the models were never attacked. Ones bias initialization did provide additional robustness. However, examining the

VGG16 baseline models for the EAD attack, which serve as the baseline to create the standard deviation metric, it is apparent that there was a large variation in robustness. This large variation accounted for a large standard deviation in L2 distances for the baseline models in which the ones bias initialization fell below to be classified as a high, medium, or low impact hyperparameter.

4.3 Neural Network Accuracy vs Robustness

In this section, we examine the relationship between NN properties accuracy and mean L2 distance with the goal of determining whether there is a tradeoff between NN robustness and accuracy. To quantify this relationship, we applied a correlation function to the accuracy and mean L2 distance metrics gathered for experiments with high impact hyperparameters bias initialization, activation function, max pooling filter size, and convolutional kernel size. By using a correlation metric, we were also able to look for patterns that may exist between architecture, dataset, and attack. As a note, in the comparisons of hyperparameters for ResNet50, image distortion values were used in place of mean L2 distance.

Figure 4.8 depicts the results of our examination of the correlation between accuracy and robustness. The correlation function describes how accuracy influences model robustness using a value between -1 and 1. Values approaching -1 indicate an inverse correlation while values that are closer to 1 indicate a correlation. Values that are near 0 indicate that there is no relationship between accuracy and mean L2 distance.

Some patterns are discernable from closely inspecting the data in figure 4.8. Bias initialization shows a strong correlation with the LeNet architecture, although it is inversely correlated with ResNet50 and VGG16 architectures. Activation is more closely aligned to

neutral for the ResNet50 architecture apart from the C&W attack with the CIFAR10 dataset. In the other architecture where it was prevalent, results were polarized between correlation and inverse correlation by dataset. The Sign Language MNIST dataset shows a strong correlation while fashion MNIST and MNIST show a strong inverse correlation. Kernel size shows as correlated with the LeNet architecture. With the ResNet50 architecture, kernel size has a neutral result as well as a strongly correlated result. VGG16 mostly has inversely correlated relationships of accuracy and robustness with kernel size, except C&W attack on the Sign Language MNIST which is neutral. Pool size was nearly split by attack with a subtle lean of correlation inversely within the LeNet architecture with the C&W attack being strongly correlated and EAD being inversely correlated. The exception to this is the C&W attack against the Sign Language MNIST dataset which is inversely correlated. Results in VGG16 are mostly correlated, but one experiment, the C&W attack on CIFAR10, has a slight inverse correlation.

After careful examination, we could see a pattern immerge with correlation of accuracy and robustness. While not perfect, bias initialization and kernel size tended to track upon architectural lines. We conclude that the relationship between salient hyperparameters and accuracy is a complicated one.

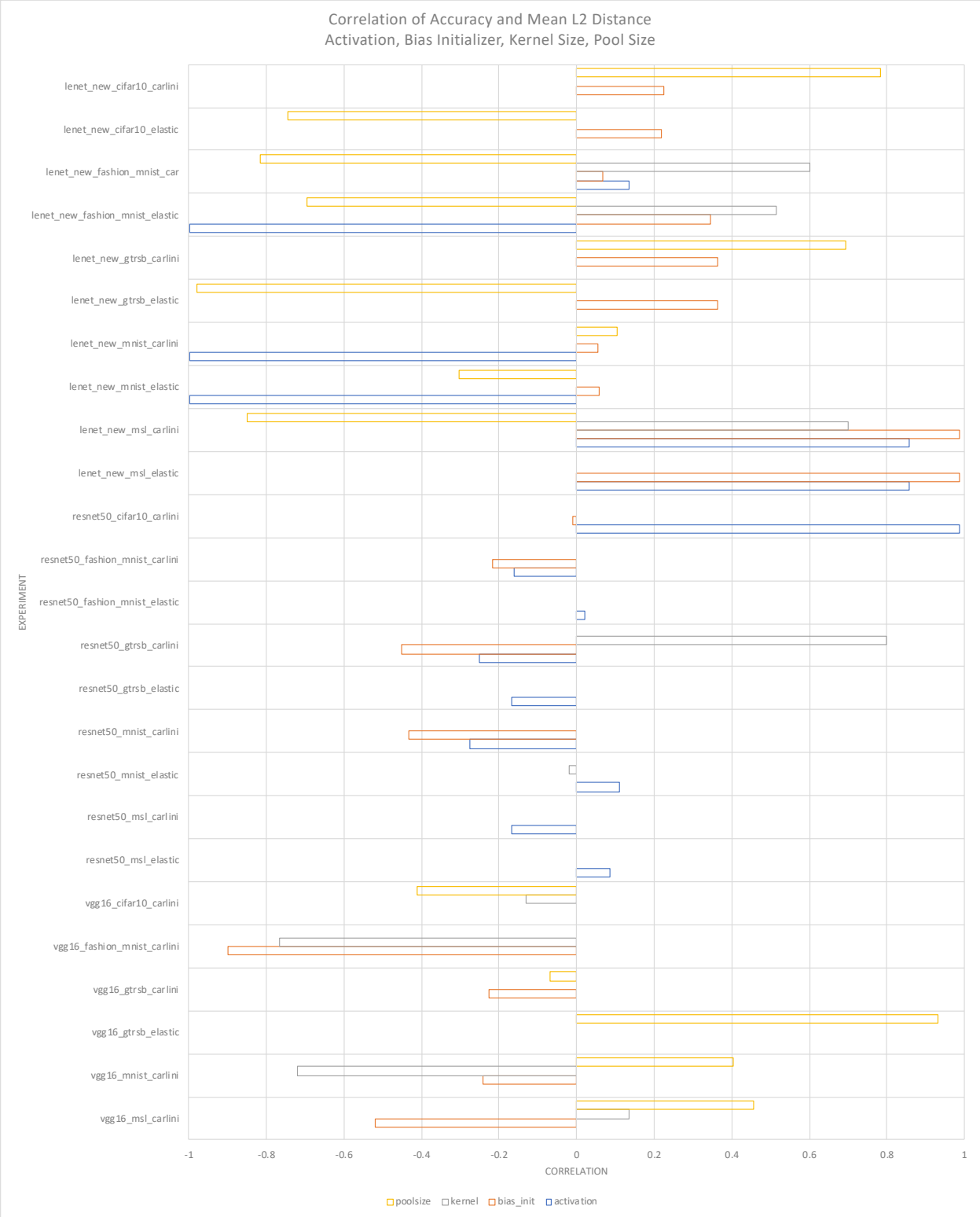

Figure 4.8 – Correlation of Robustness and Accuracy Using Mean L2 Distance

4.4 Neural Network Depth vs Robustness

The selection of LeNet, VGG16, and ResNet50 architectures for this work allowed for the opportunity to examine CNN robustness as a property of NN depth as each model represents a deeper NN by a factor of roughly three. Using the data gathered building the 100 baseline models for each architecture, we were able to look at the various L2 distance metrics collected as a function of progressive NN depth.

Figure 4.9 shows the trend for robustness as it relates to depth for all datasets using the C&W attack. The trend line displayed in the figure is based on a linear equation of the average mean L2 distances recorded when creating the baseline models for each architecture. Note that with all combinations involving the C&W attack, NN depth does show to be a predictor of model robustness.

Examining the effects of the EAD attack on progressively DNNs showed slightly different results in figure 4.10. In the case of datasets CIFAR10, GTRSB, and MNIST, the linear trend based on the mean L2 distance was not as pronounced as with the C&W attack sets. Additionally, for these datasets, robustness increased from LeNet to VGG16 and decreased from VGG16 to ResNet50. Datasets Fashion MNIST and Sign Language MNIST display a more dramatic increase in linear increase of L2 distance as well as robustness as the NN depth increased.

It is our observation that generally, NN robustness does increase with model depth. However, there are variations to this pattern between models with different architectures as it relates to robustness against different evasion attacks. L2 distances consistently increased when the C&W attacks was used, and, while the mean L2 distance trend did show a modest increase for the EAD attack, L2 distance were still larger is most the datasets tested in VGG16 when

compared to ResNet50. This data does point to model depth as being an indicator of model robustness, but it appears to show that architecture plays a strong role in broad resistance to adversarial perturbations.

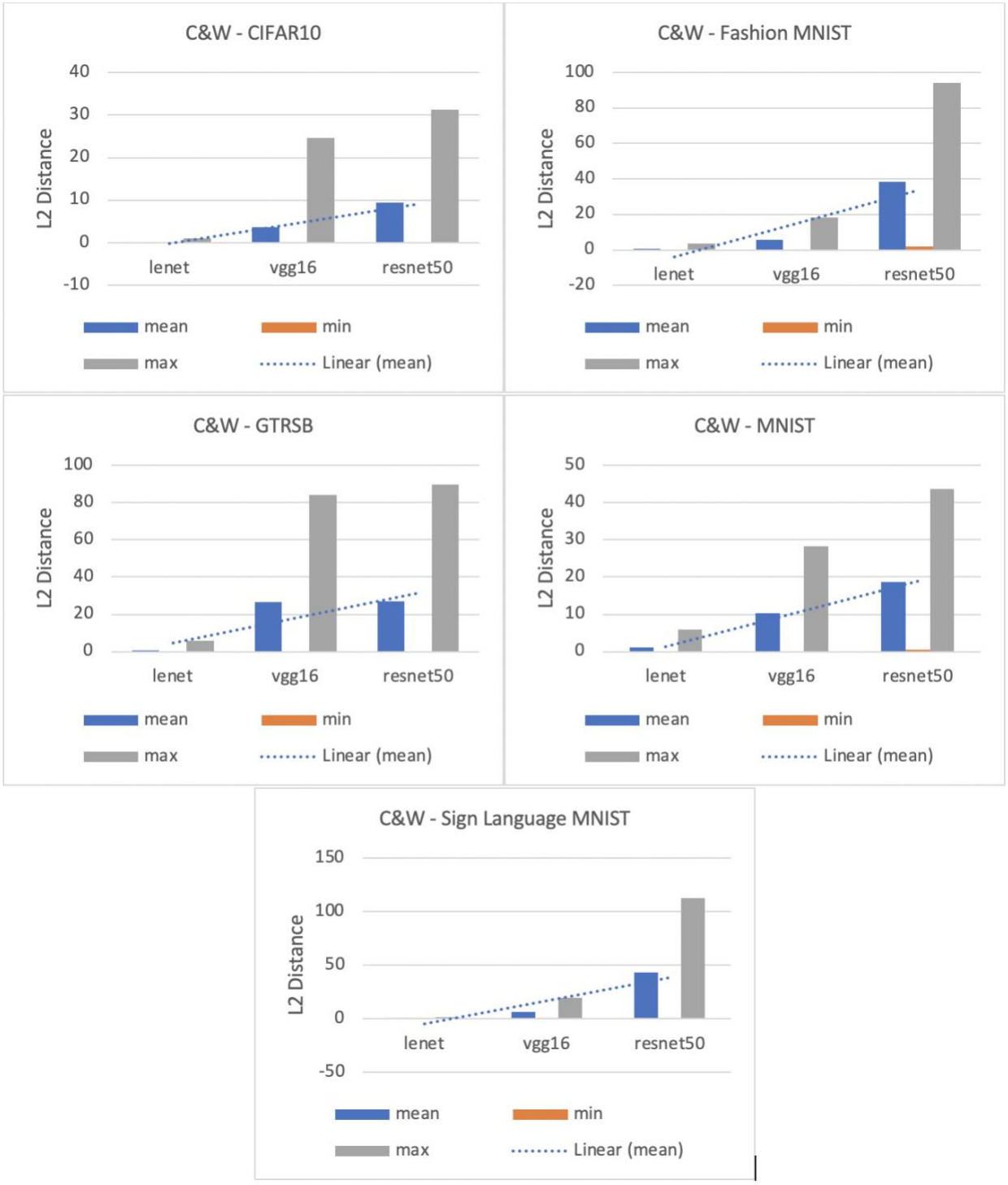

Figure 4.9 – L2 Distance Versus Model Depth Increasing from Left to Right – C&W Attack

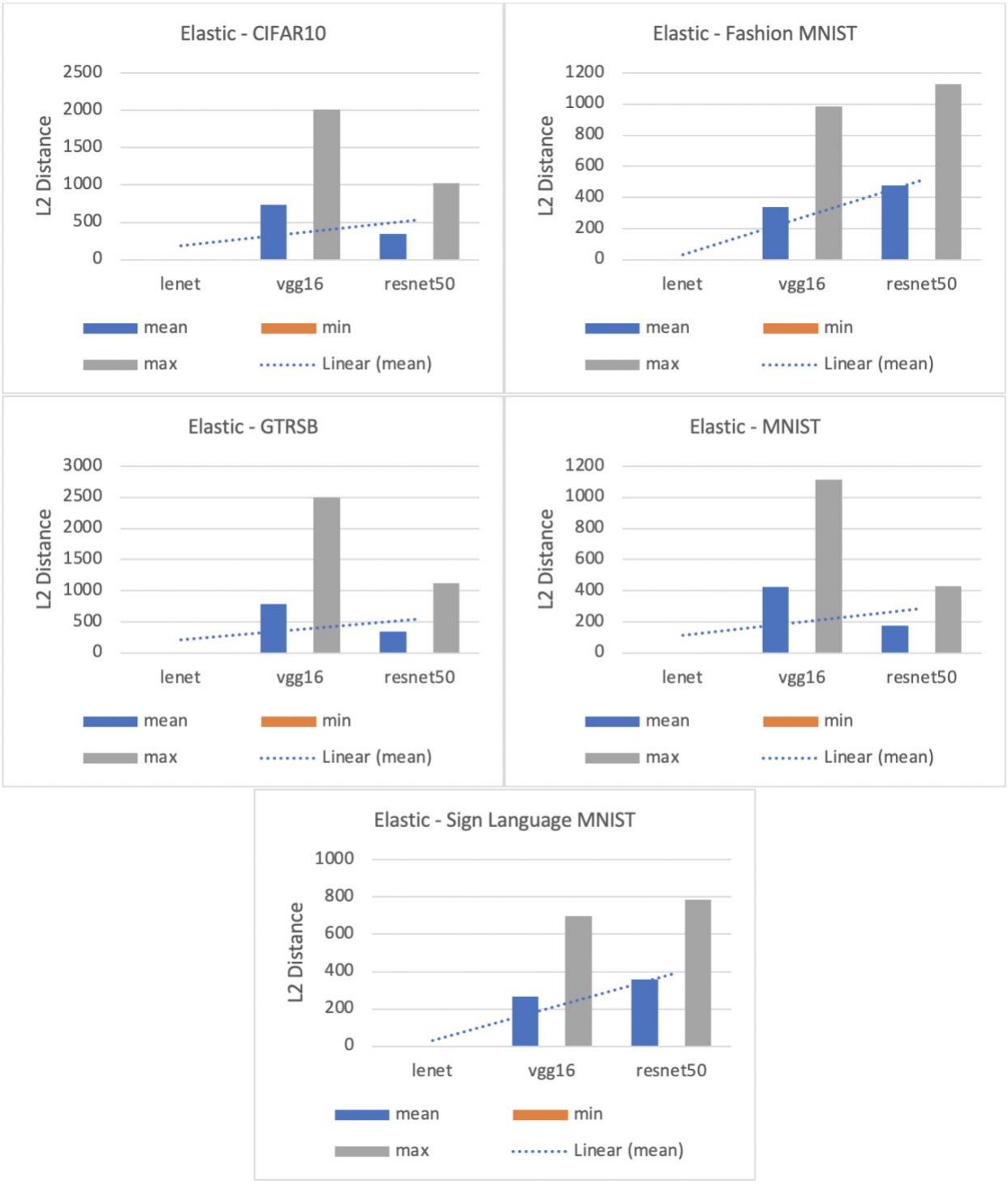

Figure 4.10 - L2 Distances Versus Model Depth
Increasing From Left to Right – EAD Attack

Chapter 5. DISCUSSION

NNs are often built with accuracy being the only means to which its performance is measured. With the degree to which ML technology has proliferated industries and has been integrated into life-critical applications, it is also important to consider performance indicators which measure ML security.

The contributions that this work makes to the field of defense against adversarial examples are straightforward. While most hyperparameters used to create NNs do not influence robustness outside of the normal variation experienced when multiple models, a few hyperparameters do consistently improve a NNs ability to resist adversarial examples, and in some cases considerably. In the case of bias initialization, we were able to show that using non-zero, constant value shows increasing robustness as the number of nodes increase in the hidden layer thereby confirming that only structures with a hidden layer should be trained to resist adversarial perturbations [29]. We suggest the mechanism at play here is the prevention of dying RELU through the use of ones bias initialization.

There are several assumptions and limitations that go along with our work. First, we consider only two attacks, and we see differences in how hyperparameters impact robustness with each attack. Given that there are many attacks in the adversarial example landscape, there may be results which may force us to reconsider our conclusions here. Second, we are comparing depths of different NN architectures. A more thorough approach to investigate this would be to test models are varying depths and of the same architecture. As an example, experiments of models within the same architecture and of varying depth may include ResNet32, ResNet50, and ResNet152. Another assumption we make is with the inability of either C&W or EAD to generate adversarial examples with ResNet50 models with Sigmoid activation even with

a confidence of 5. We concede it may be possible the result is wrongly interpreted when we assume the attack failed with L2 distances approaching zero with high distortion. Lastly, to implement many of the hyperparameters, design considerations had to be made against the default architectures that didn't include the hyperparameter in question, and there may be differences in how hyperparameters are implemented. Additionally, VGG16 and ResNet50 in literature use much larger images than what was tested with in this work. In the interest of time, image sizes were not increased to 224x224 for VGG16 and ResNet50 experiments as the computational time required to conduct our experiments would increase substantially. Given that an image of 224x224 has more than 50 times the number of pixels as a 30x30 image, we presume that experiments with the larger image would take between 50 to 100 hours to complete on a 10-class dataset whereas our experiments took between 1 and 2 hours.

The implications of this work in the context of cybersecurity could be used in the application of principles of a secure development lifecycle, something akin to Microsoft's SDL or OWASP's SDLC, where security and privacy considerations are placed into the phases of the development cycle providing assurance activities that allow for the development of secure features. It seems logical that this approach could be applied to machine learning development, and techniques discussed in this work could be used as assurance activities to develop more secure machine learning models. With the knowledge that the choice of hyperparameters used for a NN does affect its robustness, development cycles could be allocated to test a NN's resistance to adversarial examples in addition to honing accuracy.

Short of incorporating this information into secure development lifecycle, how can the results of these experiments immediately be applied to provide for more robust NNs? Sigmoid activation should first be considered for use as it provided the highest level of adversarial

resistance in terms of L2 distance. However, it is also slow to converge resulting in extended model training times. The later can be resolved using batch normalization, but in limited testing we found that this may decrease the amount of adversarial resistance. Sigmoid activation was also second to ones bias initialization in high-impact frequency across experiments therefore it may not work in many applications. Case in point, it did not work with VGG16. Relu activation with ones bias initialization should be the next consideration. This combination of hyperparameters showed the next highest levels of robustness, and those values tended to increase when the number of nodes in the hidden layer(s) increased. Moreover, Relu activation with ones bias initialization most frequently had a high impact on robustness in our experiments. Focus on max pooling layers should come next as it is the hyperparameter with the third most frequent results. Our experiments conclude that larger max pool sizes are better. Generally, a max pooling setting of (5, 5) or (4, 4) provided for the best results. The last hyperparameter providing consistent results that could be applied to strengthen NNs against attack is kernel size. We find with this hyperparameter, smaller is generally better.

Throughout this work we highlight the hyperparameter and setting which provided the highest level of resistance against adversarial examples. If the suggested hyperparameter and setting cannot be used, we have table 6.8 in the Appendix listing next-best hyperparameter settings. While not receiving the title of best setting in its respective hyperparameter class, they nonetheless registered has having high, medium, or low impact as explained in our Experimental Setup section.

Chapter 6. CONCLUSION

In this paper, we have generated hundreds of NN models with five datasets on three CNN architectures, including two state-of-the-art architectures that placed well in the ILSVRC, with commonly used hyperparameters. We were able to generate adversarial examples in most cases, and in the process, we collected various metrics allowing for the examination of model robustness as it relates to hyperparameter selection, model accuracy, and model depth.

By analyzing collected metrics, such as the L2 norm, as well as standard deviations derived from the baseline models, we have observed there are common hyperparameters that exist across architecture, dataset, and adversarial generation method that improve resiliency to attack. We believe that Sigmoid activation, ones bias initialization, a larger max pooling filter size, and a smaller convolutional kernel size each can improve robustness when used to build a CNN.

Sigmoid activation and bias ones initialization provide the highest levels of resistant to adversarial perturbations. We believe that Sigmoid activation creates robust models due to the small derivative it produces, which may obfuscate the derivative for white box attack algorithms. The mechanism that we believe increases adversarial resistance with bias initialization is hidden layer node activation. We were able to show that with a constant, non-zero bias initialization, resistance to adversarial examples increase as the number of hidden layer nodes increased.

A review of the L2 norm metrics collected from the dataset and architecture baseline models allowed for investigations into robustness versus model depth. We arrived at the conclusion that model depth does have an impact on adversarial examples resistance, but the architecture also seems to play a large role given the differences in behavior between the attacks tested.

When considering the accuracy achieved in the baseline models in relation to the accuracy of the models with the most salient hyperparameters, we conclude that the result is complicated. In some cases, we see that the accuracy increased with the implementation of robust hyperparameters, and in models with the same architecture and different dataset we see decreases in accuracy versus the baseline.

6.1 Future Work

The conclusions we arrive at in this work help to better understand the salient hyperparameters for improving the robustness of NNs, and we have helped shed some light into why these hyperparameters have the impact on robustness that they do. In the process, we have encountered design aspects that fall out-of-scope for this project as well as curious results that may be interesting options for future work. Some examples of directions of future work include the expansion of the tests to include more hyperparameters such as the BRELU or PRELU activation or the use of additional adversarial example generation methods. Additional topics could examine the inability of C&W and EAD attacks to succeed against ResNet50 models using Sigmoid activation and push the attack confidences to higher levels. VGG16 and ResNet50 also appeared to show adversarial resistance for the EAD attack which did not present in the LeNet model. Lastly, examination of robustness achieved when mixing the more adversarial resistant hyperparameters as well as understanding the exact mechanism that allows for Sigmoid activations would be useful.

BIBLIOGRAPHY

[1] Q. Tariq *et al.*, "Detecting Developmental Delay and Autism Through Machine Learning Models Using Home Videos of Bangladeshi Children: Development and Validation Study," *J Med Internet Res*, vol. 21, no. 4, Apr. 2019, doi: 10.2196/13822.

[2] F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein, "Fake News Detection on Social Media using Geometric Deep Learning," *arXiv:1902.06673 [cs, stat]*, Feb. 2019, Accessed: Apr. 07, 2021. [Online]. Available: http://arxiv.org/abs/1902.06673

[3] "Self-driving Vehicles | One Hundred Year Study on Artificial Intelligence (AI100)." https://ai100.stanford.edu/2016-report/section-ii-ai-domain/transportation/self-driving-vehicles (accessed Aug. 18, 2021).

[4] J. Dow, "Mercedes cars will be powered by NVIDIA AI for self-driving starting 2024," *Electrek*, Jun. 23, 2020. https://electrek.co/2020/06/23/mercedes-cars-will-be-powered-by-nvidia-ai-for-self-driving-starting-2024/ (accessed Aug. 18, 2021).

[5] A. Gebhart, "Facial recognition: Apple, Amazon, Google and the race for your face," *CNET*. https://www.cnet.com/home/smart-home/facial-recognition-apple-amazon-google-and-the-race-for-your-face-facebook/ (accessed Aug. 18, 2021).

[6] C. Szegedy *et al.*, "Intriguing properties of neural networks," *arXiv:1312.6199 [cs]*, Feb. 2014, Accessed: Apr. 07, 2021. [Online]. Available: http://arxiv.org/abs/1312.6199

[7] K. Eykholt *et al.*, "Robust Physical-World Attacks on Deep Learning Models," *arXiv:1707.08945 [cs]*, Apr. 2018, Accessed: Apr. 07, 2021. [Online]. Available: http://arxiv.org/abs/1707.08945

[8] C. Burkard and B. Lagesse, "Can Intelligent Hyperparameter Selection Improve Resistance to Adversarial Examples?," *arXiv:1902.05586 [cs, stat]*, Feb. 2019, Accessed: Apr. 07, 2021. [Online]. Available: http://arxiv.org/abs/1902.05586

[9] D. Su, H. Zhang, H. Chen, J. Yi, P.-Y. Chen, and Y. Gao, "Is Robustness the Cost of Accuracy? -- A Comprehensive Study on the Robustness of 18 Deep Image Classification Models," *arXiv:1808.01688 [cs]*, Mar. 2019, Accessed: Apr. 07, 2021. [Online]. Available: http://arxiv.org/abs/1808.01688

[10] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "Robustness May Be at Odds with Accuracy," *arXiv:1805.12152 [cs, stat]*, Sep. 2019, Accessed: Apr. 07, 2021. [Online]. Available: http://arxiv.org/abs/1805.12152

[11] "Deep Learning." https://www.deeplearningbook.org/ (accessed Apr. 16, 2021).

[12] W. Luo, Y. Li, R. Urtasun, and R. Zemel, “Understanding the Effective Receptive Field in Deep Convolutional Neural Networks,” *arXiv:1701.04128 [cs]*, Jan. 2017, Accessed: May 05, 2021. [Online]. Available: http://arxiv.org/abs/1701.04128

[13] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv:1409.1556 [cs]*, Apr. 2015, Accessed: Apr. 07, 2021. [Online]. Available: http://arxiv.org/abs/1409.1556

[14] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights Imaging*, vol. 9, no. 4, pp. 611–629, Jun. 2018, doi: 10.1007/s13244-018-0639-9.

[15] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma, “Adversarial classification,” in *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’04*, Seattle, WA, USA, 2004, p. 99. doi: 10.1145/1014052.1014066.

[16] B. Nelson *et al.*, “Exploiting Machine Learning to Subvert Your Spam Filter,” p. 9.

[17] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, “Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing,” *Proc USENIX Secur Symp*, vol. 2014, pp. 17–32, Aug. 2014.

[18] G. Ateniese, G. Felici, L. V. Mancini, A. Spognardi, A. Villani, and D. Vitali, “Hacking Smart Machines with Smarter Ones: How to Extract Meaningful Data from Machine Learning Classifiers,” *arXiv:1306.4447 [cs, stat]*, Jun. 2013, Accessed: Aug. 06, 2021. [Online]. Available: http://arxiv.org/abs/1306.4447

[19] Y. Liu *et al.*, “Trojaning Attack on Neural Networks,” p. 17.

[20] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, “Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning,” *arXiv:1712.05526 [cs]*, Dec. 2017, Accessed: Feb. 25, 2019. [Online]. Available: http://arxiv.org/abs/1712.05526

[21] A. Serban, E. Poll, and J. Visser, “Adversarial Examples on Object Recognition: A Comprehensive Survey,” *arXiv:2008.04094 [cs]*, Sep. 2020, Accessed: Aug. 06, 2021. [Online]. Available: http://arxiv.org/abs/2008.04094

[22] A. Rozsa, M. Günther, and T. E. Boult, “Are Accuracy and Robustness Correlated?,” *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 227–232, Dec. 2016, doi: 10.1109/ICMLA.2016.0045.

[23] Y. Liu, X. Chen, C. Liu, and D. Song, “Delving into Transferable Adversarial Examples and Black-box Attacks,” *arXiv:1611.02770 [cs]*, Nov. 2016, Accessed: Feb. 26, 2019. [Online]. Available: http://arxiv.org/abs/1611.02770

[24] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, “Is Feature Selection Secure against Training Data Poisoning?,” in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 1689–1698. Accessed: Mar. 06, 2016. [Online]. Available: http://machinelearning.wustl.edu/mlpapers/paper_files/icml2015_xiao15.pdf

[25] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, “Adversarial Perturbations Against Deep Neural Networks for Malware Classification,” *arXiv:1606.04435 [cs]*, Jun. 2016, Accessed: Jul. 14, 2016. [Online]. Available: http://arxiv.org/abs/1606.04435

[26] A. Nguyen, J. Yosinski, and J. Clune, “Deep neural networks are easily fooled: High confidence predictions for unrecognizable images,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 427–436. Accessed: Dec. 05, 2016. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7298640

[27] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks,” *arXiv:1511.04508 [cs, stat]*, Mar. 2016, Accessed: Apr. 07, 2021. [Online]. Available: http://arxiv.org/abs/1511.04508

[28] N. Carlini and D. Wagner, “Defensive Distillation is Not Robust to Adversarial Examples,” *arXiv:1607.04311 [cs]*, Jul. 2016, Accessed: Apr. 07, 2021. [Online]. Available: http://arxiv.org/abs/1607.04311

[29] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and Harnessing Adversarial Examples,” *arXiv:1412.6572 [cs, stat]*, Mar. 2015, Accessed: Apr. 07, 2021. [Online]. Available: http://arxiv.org/abs/1412.6572

[30] D. Hendrycks and T. Dietterich, “Benchmarking Neural Network Robustness to Common Corruptions and Perturbations,” *arXiv:1903.12261 [cs, stat]*, Mar. 2019, Accessed: Apr. 30, 2020. [Online]. Available: http://arxiv.org/abs/1903.12261

[31] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples,” *arXiv:1602.02697 [cs]*, Feb. 2016, Accessed: Jul. 14, 2016. [Online]. Available: http://arxiv.org/abs/1602.02697

[32] A. Athalye, N. Carlini, and D. Wagner, “Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples,” *arXiv:1802.00420 [cs]*, Feb. 2018, Accessed: Feb. 23, 2019. [Online]. Available: http://arxiv.org/abs/1802.00420

[33] V. Zantedeschi, M.-I. Nicolae, and A. Rawat, “Efficient Defenses Against Adversarial Attacks,” *arXiv:1707.06728 [cs]*, Jul. 2017, Accessed: Feb. 23, 2019. [Online]. Available: http://arxiv.org/abs/1707.06728

[34] R. Bakhteri and S. S. Liew, “Bounded activation functions for enhanced training stability of deep neural networks on visual pattern recognition problems”, Accessed: Aug. 08, 2021. [Online]. Available:

https://www.academia.edu/28108405/Bounded_activation_functions_for_enhanced_training_stability_of_deep_neural_networks_on_visual_pattern_recognition_problems

[35] H. Berntsen, W. Kuijper, and T. Heskes, "The Artificial Mind's Eye: Resisting Adversarials for Convolutional Neural Networks using Internal Projection," *arXiv preprint arXiv:1604.04428*, 2016, Accessed: Nov. 24, 2016. [Online]. Available: http://arxiv.org/abs/1604.04428

[36] X. Li and F. Li, "Adversarial Examples Detection in Deep Networks with Convolutional Filter Statistics," *arXiv:1612.07767 [cs]*, Oct. 2017, Accessed: Apr. 07, 2021. [Online]. Available: http://arxiv.org/abs/1612.07767

[37] B. Nelson, "Behavior of Machine Learning Algorithms in Adversarial Environments," PhD Thesis, University of California at Berkeley, 2010.

[38] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?," in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, 2006, pp. 16–25. Accessed: Nov. 17, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?id=1128824

[39] N. Carlini *et al.*, "On Evaluating Adversarial Robustness," *arXiv:1902.06705 [cs, stat]*, Feb. 2019, Accessed: Apr. 23, 2020. [Online]. Available: http://arxiv.org/abs/1902.06705

[40] N. Carlini and D. Wagner, "Towards Evaluating the Robustness of Neural Networks," *arXiv preprint arXiv:1608.04644*, 2016, Accessed: Nov. 24, 2016. [Online]. Available: https://arxiv.org/abs/1608.04644

[41] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv:1607.02533 [cs, stat]*, Feb. 2017, Accessed: Apr. 07, 2021. [Online]. Available: http://arxiv.org/abs/1607.02533

[42] A. Graese, A. Rozsa, and T. E. Boult, "Assessing Threat of Adversarial Examples on Deep Neural Networks," *arXiv preprint arXiv:1610.04256*, 2016, Accessed: Nov. 24, 2016. [Online]. Available: https://arxiv.org/abs/1610.04256

[43] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial Machine Learning at Scale," *arXiv preprint arXiv:1611.01236*, 2016, Accessed: Nov. 24, 2016. [Online]. Available: https://arxiv.org/abs/1611.01236

[44] E. Duesterwald, A. Murthi, G. Venkataraman, M. Sinn, and D. Vijaykeerthy, "Exploring the Hyperparameter Landscape of Adversarial Robustness," *arXiv:1905.03837 [cs, stat]*, May 2019, Accessed: Aug. 17, 2021. [Online]. Available: http://arxiv.org/abs/1905.03837

[45] C. Szegedy *et al.*, "Going Deeper with Convolutions," *arXiv:1409.4842 [cs]*, Sep. 2014, Accessed: May 26, 2021. [Online]. Available: http://arxiv.org/abs/1409.4842

[46] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.

[47] M. Ferguson, R. ak, Y.-T. Lee, and K. Law, "Automatic localization of casting defects with convolutional neural networks," Dec. 2017, pp. 1726–1735. doi: 10.1109/BigData.2017.8258115.

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015, Accessed: Apr. 07, 2021. [Online]. Available: http://arxiv.org/abs/1512.03385

[49] "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges." http://yann.lecun.com/exdb/mnist/ (accessed Apr. 07, 2021).

[50] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," *arXiv:1708.07747 [cs, stat]*, Sep. 2017, Accessed: Aug. 04, 2021. [Online]. Available: http://arxiv.org/abs/1708.07747

[51] "AI Progress Measurement," *Electronic Frontier Foundation*, Jun. 12, 2017. https://www.eff.org/ai/metrics (accessed Apr. 07, 2021).

[52] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, vol. 32, pp. 323–332, Aug. 2012, doi: 10.1016/j.neunet.2012.02.016.

[53] "Sign Language MNIST." https://kaggle.com/datamunge/sign-language-mnist (accessed Apr. 07, 2021).

[54] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh, "EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples," p. 8.

[55] "Classification: Accuracy | Machine Learning Crash Course," *Google Developers*. https://developers.google.com/machine-learning/crash-course/classification/accuracy (accessed Aug. 17, 2021).

[56] K. Team, "Keras documentation: Image data preprocessing." https://keras.io/api/preprocessing/image/ (accessed Aug. 16, 2021).

[57] K. Team, "Keras documentation: EarlyStopping." https://keras.io/api/callbacks/early_stopping/ (accessed Aug. 16, 2021).

[58] T. Dietterich, "Overfitting and Undercomputing in Machine Learning," *Computing Surveys*, vol. 27, pp. 326–327, 1995.

[59] V. P. Plagianakos, G. D. Magoulas, and M. N. Vrahatis, “Learning Rate Adaptation in Stochastic Gradient Descent,” in *Advances in Convex Analysis and Global Optimization: Honoring the Memory of C. Caratheodory (1873–1950)*, N. Hadjisavvas and P. M. Pardalos, Eds. Boston, MA: Springer US, 2001, pp. 433–444. doi: 10.1007/978-1-4613-0279-7_27.

[60] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation Functions: Comparison of trends in Practice and Research for Deep Learning,” *arXiv:1811.03378 [cs]*, Nov. 2018, Accessed: Aug. 17, 2021. [Online]. Available: http://arxiv.org/abs/1811.03378

[61] “Standard Deviation,” *DeepAI*, May 17, 2019. https://deepai.org/machine-learning-glossary-and-terms/standard-deviation (accessed Aug. 17, 2021).

[62] “CS231n Convolutional Neural Networks for Visual Recognition.” https://cs231n.github.io/convolutional-networks/ (accessed Aug. 09, 2021).

[63] R. Jozefowicz, W. Zaremba, and I. Sutskever, “An Empirical Exploration of Recurrent Network Architectures,” p. 9.

APPENDIX A

Table 6.1 - Dataset Characteristics

Name	**Training Samples**	**Test Samples**	**Classes**	**Dimensions**	**Channels**
MNIST	60,000	10,000	10	28 x 28	1
Fashion MNIST	60,000	10,000	10	28 x 28	1
CIFAR10	50,000	10,000	10	32 x 32	3
GTSRB	39,209	12,630	43	30 x 30	3
MNIST Sign Language	27,455	7,172	24	28 x 28	1

Table 6.2– Baseline Model Minimum, Mean, and Maximum Accuracy by Architecture and Dataset

Baseline Model Accuracy

Datasets	CNN Architectures								
	LeNet			VGG16			ResNet50		
	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
CIFAR10	0.808	0.824	0.836	0.853	0.883	0.898	0.752	0.814	0.843
Fashion MNIST	0.875	0.881	0.886	0.900	0.915	0.923	0.797	0.809	0.823
GTRSB	0.872	0.890	0.912	0.975	0.991	0.997	0.713	0.932	1.000
MNIST	0.985	0.987	0.989	0.985	0.991	0.994	0.865	0.883	0.916
Sign Language MNIST	0.910	0.924	0.936	0.976	1.000	1.000	0.942	0.957	0.970

Table 6.3 - Network Hyperparameters Tested

Network Hyperparameter	Parameter Range
Activation Unit	Elu, Linear, Relu, Sigmoid, Tanh
Activity Regularization	L1_L2(.01), L1_L2(.02), L1_L2(.03), L1_L2(.04), L1_L2(.05), L1(.01), L1(.02), L1(.03), L1(.04), L1(.05), L2(.02), L2(.04), L2(.06), L2(.08), L2(.10)
Bias Initializer	Zeros, Ones, Random Uniform, Truncated Normal, Variance Scaling
Convolutional Kernel Size (Dimensions)	2 x 2, 3 x 3, 4 x 4, 5 x 5, 6 x 6, 7 x 7, 8 x 8, 9 x 9
Kernel Initializer	Zeros, Glorot Uniform, Random Uniform, Orthogonal, Truncated Normal, HE Normal
Kernel Regularization	L1_L2(.01), L1_L2(.02), L1_L2(.03), L1_L2(.04), L1_L2(.05), L1(.01), L1(.02), L1(.03), L1(.04), L1(.05), L2(.02), L2(.04), L2(.06), L2(.08), L2(.10)
Max Pooling Filter Size (Dimensions)	2 x 2, 3 x 3, 4 x 4, 5 x 5, 6 x 6
Number of Dense Layers	1, 2, 3, 4, 5
Number of Filters per CNN	32, 64, 128, 256, 512
Number of Hidden Nodes	512, 1024, 2048, 3072, 4096
Percent Dropout on Input Layer	10%, 20%, 30%, 40%, 50%
Percent Dropout on Dense Layer	10%, 20%, 30%, 40%, 50%
Stride Size	1, 2, 3

Table 6.4 - Training Hyperparameters Tested

Training Hyperparameter	Parameter Range
Batch Size	32, 64, 128, 256, 512
Loss	SigCross, Hinge, Log, Mean Pairwise, MSE, COS
Momentum	.1, .3, .5, .7, .9
Optimizer	SGD, RMS, Adadelta, Adam, Adamax, Nadam

Table 6.5 - Standard Deviations of Distance Metrics Across 100 Models

Architecture	Dataset	Attack	Mean Distance	Minimum Distance	Maximum Distance	Standard Deviation
LeNet	CIFAR10	C&W	0.0995	0.0058	0.5027	0.1032
		EAD	0.1077	0.0069	0.6391	0.0969
	Fashion MNIST	C&W	0.2835	0.0151	2.1172	0.366
		EAD	0.2097	0.017	1.4057	0.2648
	GTRSB	C&W	0.2443	8.40E-11	3.2364	0.2484
		EAD	0.2955	0.0004	5.0175	0.3731
	MNIST	C&W	0.7166	0.0804	2.9637	0.7109
		EAD	0.6786	0.075	3.4589	0.7427
	Sign Language MNIST	C&W	0.1133	0.0034	0.7478	0.1007
		EAD	0.0786	0.0051	0.5255	0.0852
VGG16	CIFAR10	C&W	1.7266	0.027	13.6854	2.4534
		EAD	249.7023	0.1011	201.3164	99.0923
	Fashion MNIST	C&W	4.2474	0.0758	10.8775	4.1145
		EAD	130.0061	0.3067	138.6188	73.5207
	GTRSB	C&W	8.1111	0.01602	21.7241	7.8707
		EAD	143.3066	0.0002	139.5619	35.4931
	MNIST	C&W	6.7086	0.429	15.5627	5.9864
		EAD	145.3306	0.3801	14.8212	41.6238
	Sign Language MNIST	C&W	3.6161	0.0195	9.5645	3.8562
		EAD	63.5364	0.0101	1.5581	17.6971
ResNet50	CIFAR10	C&W	8.8782	0.1945	27.2459	9.9137
		EAD	133.855	0.0079	103.3631	64.7757
	Fashion MNIST	C&W	34.0991	11.6394	57.1071	18.3788
		EAD	142.3906	0.714	32.7246	28.3736
	GTRSB	C&W	10.4574	0.09843	31.7554	9.6296
		EAD	68.1375	0.0031	43.1978	21.4673

	MNIST	C&W	11.0002	1.5721	19.112	6.7322
		EAD	52.5665	0.0068	12.9205	15.9815
	Sign Language MNIST	C&W	17.0443	0.7427	31.6419	10.6035
		EAD	89.8299	0.0499	11.0932	14.6769

Table 6.6 – Default Hyperparameter Settings for All Architectures

Network Hyperparameter	**LeNet**	**VGG16**	**ResNet50**
Activation Unit	Relu	Relu	Relu
Activity Regularization	None	None	None
Bias Initializer	Zeros	Zeros	Zeros
Convolutional Kernel Size (Dimensions)	5 x 5	3 x 3	7 x 7
Kernel Initializer	Glorot uniform	He normal	He normal
Kernel Regularization	None	None	None
Max Pooling Filter Size (Dimensions)	2 x 2	2 x 2	3 x 3
Number of Dense Layers	1	2	1
Number of Filters per CNN	32	64	64
Number of Hidden Nodes	500	4096	# of classes (output)
Percent Dropout on Input Layer	None	None	None
Percent Dropout on Dense Layer	None	50%	None
Stride Size	1	1	2

Training Hyperparameter	**LeNet**	**VGG16**	**ResNet50**
Batch Size	128	128	128
Loss	SigCross	Categorical crossentropy	Categorical crossentropy
Momentum	0.9	0.9	0.9
Optimizer	Adam	Adam	Adam

Table 6.7 – Hyperparameters with Highest Impact Across All Experiments

Parameter			Activation Unit	Activity Regularizer L2	Batch Size	Bias Initializer	Dense Layers (Number of)	Dense Nodes (Two Layers)	Dropout on Dense Layer	Dropout on Input Layer	Filters per CNN	Kernel Initializer	Kernel Regularizer L1	Kernel Regularizer L2	Convolutional Kernel Size	Loss	Momentum	Optimization	Max Pooling Filter Size	Dense Nodes (One Layer)	Strides	Dense Nodes (Three Layers)
C&W	LeNet	CIFAR10				ones													4, 4			
		Fashion MNIST	Sigmoid		128	ones								0.08	3, 3	MSE			5, 5		2	
		GTRSB			128	ones											1		2, 2			
		MNIST	Sigmoid		512	ones												adam	4, 4	512		
		Sign Language	Sigmoid			ones									5, 5				4, 4			
	VGG16	CIFAR10					5		0.4	0.1					2, 2				5, 5	512		1024
		Fashion MNIST				ones		512							2, 2		7	adam		4096		
		GTRSB				ones													5, 5			
		MNIST				ones		512		0.4					2, 2				5, 5		1	
		Sign Language			32	ones		4096							2, 2		5	adamax	5, 5			
	ResNet50	CIFAR10	Relu			ones				0.3												
		Fashion MNIST	*Sigmoid			ones							0.01	0.1		Sig Cross					1	
		GTRSB	*Sigmoid			ones									4, 4							
		MNIST	*Sigmoid			ones										Cos					1	
		Sign Language	*Sigmoid																			
EAD	LeNet	CIFAR10			32	ones	5												5, 5	1024	2	
		Fashion MNIST	Sigmoid	0.04	32	ones	3		0.2		512	he normal		0.06	4, 4		3		5, 5		2	
		GTRSB				ones											5		4, 4			1024
		MNIST	Sigmoid	0.02		ones										Cos			4, 4	3072	3	
		Sign Language	Sigmoid			ones																
	VGG16	CIFAR10																				
		Fashion MNIST																				
		GTRSB										random uniform							5, 5			
		MNIST																				
		Sign Language																				
	ResNet50	CIFAR10																				
		Fashion MNIST	*Sigmoid																			
		GTRSB	*Sigmoid																			
		MNIST	*Sigmoid									random uniform			2, 2							
		Sign Language	*Sigmoid																			

Impact High Medium Low

Table 6.8 - Experiments with More Than One High Impact Result

Attack	Architecture	Dataset	Hyperparameter	High	Medium	Low
C&W	LeNet	fashion mnist	Bias Initialization	Ones		Variance Scaling
			Max Pooling			(6, 6), (5, 5)
			Loss	MSE		Soft Cross
		mnist	Max Pooling	(6, 6), (4, 4)	(5, 5)	(3, 3)
		sign language	Bias Initialization	Ones	Variance Scaling	
	ResNet50	cifar10	Activation			Relu
		fashion mnist	Kernel Size			0.01, 0.02
		gtrsb	Kernel Size	(4, 4)	(3, 3)	(2, 2)
		mnist	Bias Initialization	Ones		Zeros
	VGG16	cifar10	Triple Dense Layers	1024, 3072		
			Max Pooling	(5, 5), (4, 4)	(6, 6)	
		fashion mnist	Bias Initialization	Ones		Random Uniform
			Momentum		7	9
		mnist	Kernel Size		(2, 2)	(6, 6)
			Bias Initialization	Ones		Variance Scaling
EAD	LeNet	cifar10	Batch Size			32, 512
		fashion mnist	Kernel Size			(4, 4), (6, 6)
			Bias Initialization	Ones, Truncated Normal		Variance Scaling
			Max Pooling	(5, 5)	(6, 6)	(2, 2)
		mnist	Bias Initialization	Ones		Truncated Normal
			Max Pooling	(6, 6), (5, 5)		
		sign language	Bias Initialization	Ones		Truncated Normal
			Activation	Sigmoid		
	VGG16	gtrsb	Max Pooling	(5, 5)	(6, 6)	