

**ANALYSIS OF MULTIPLE ADVERSARIAL ATTACKS ON
CONVOLUTIONAL NEURAL NETWORKS**

A Thesis Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Burcum Eken
August 2022

Copyright © 2022 by Burcum Eken.
All rights reserved.

ACKNOWLEDGEMENTS

I would like to thank my dear family, who has always been by my side and has supported me unconditionally. I couldn't have made it this far without their sacrifices.

I would like to thank my professor and graduate advisor Dr. Seddik Djouadi for his guidance and advice throughout my thesis work. I extend my gratitude to Dr. Kevin Tomsovic and Dr. Stella Sun for willing to serve as my committee members.

Lastly, I would like to also thank the Republic of Turkey Ministry of National Education for giving me this wonderful opportunity and opening me up to new ideas.

ABSTRACT

The thesis studies different kind of adversarial attacks on Convolutional Neural Networks by using -ERCOT- electric load data set in order to fool deep neural networks. With the improvement of Deep Learning methods, their securities and vulnerabilities have become an important research subject. An adversary who gains access to the model and data sets may add some perturbations to the datasets, which may cause significant damage to the system. Six types of adversarial attacks are used and compared to each other for this research. As mentioned, these attacks can damage systems and private information of individuals, and even cause larger attacks. In the related study, a convolutional neural networks model was established by converting electrical load forecasting data sets into the images. The established model can be used not only on this data, but also can expanded it to other applications. With the implementation of the adversarial attacks, the accuracy value has been greatly affected, showing the security vulnerability of artificial intelligence systems that include convolutional neural networks. The aim of this research is to find out how these attacks affect state-of-the-art accuracy and how vulnerable convolutional neural networks can be. A comparison of various attack strategies is carried out with comparison.

TABLE OF CONTENTS

CHAPTER ONE INTRODUCTION AND THESIS OUTLINE.....	1
CHAPTER TWO BACKGROUND OF ADVERSARIAL ATTACKS AND LITERATURE REVIEW.....	4
2.1 Introduction.....	4
2.2 Fast Gradient Sign Method	5
2.3 Project Gradient Descent	7
2.4 Deep Fool.....	10
2.5 One Pixel.....	11
2.6 Sparse Fool.....	13
2.7 Basic Iterative Method (BIM).....	17
CHAPTER THREE CONVOLUTIONAL NEURAL NETWORKS	19
3.1 Perceptron	19
3.2 Multilayer Perceptron	21
3.3 Convolutional Neural Networks.....	25
3.3.1 Convolutional Layers.....	26
3.3.2 Pooling Layer	31
3.3.3 Fully Connected Layer.....	35
3.3.3.1 Softmax Cross-Entropy.....	37
3.3.3.2 Minimizing Loss.....	38

CHAPTER FOUR DATA COLLECTION AND PREPARATION.....	42
4.1 Data Collection.....	42
4.2 Convolutional Neural Networks Structure.....	44
CHAPTER FIVE EXPERIMENTATION RESULTS	49
CHAPTER SIX CONCLUSION	55
LIST OF REFERENCES.....	57
VITA	63

LIST OF TABLES

Table 5-1 The accuracy for classification and the rate of success of the attacks on Convolutional Neural Network.....	53
--	----

LIST OF FIGURES

Figure 2-1 The first image which belongs to the West Data misclassified as the data of the Far West Station because of the FGSM attack.	8
Figure 2-2 The illustration of Sparse Fool method [14]	16
Figure 3-1 A model of a neuron that places an emphasis on the structure that links the axon and the dendrite [25].	20
Figure 3-2 The graphical representation of a perceptron.	22
Figure 3-3 Depicts the network of perceptrons that constitutes an MLP.	23
Figure 3-4 One of the images that belongs to the Far West Station in 2015.	27
Figure 3-5 Typical CNN Architecture[36]	28
Figure 3-6 It shows the 3x3 matrix multiplication.	30
Figure 3-7 (a)The resulting the first convolution (b)The result of the feature map	32
Figure 3-8 Maximum and Average pooling example [38].	34
Figure 3-9 A graph illustrating the connections between the two fully connected layers[40].	36
Figure 4-1 Electric load data is converted to RGB model.	43
Figure 4-2 The pixel values of a grayscale image are scaled from 0 to 255.	45
Figure 4-3 One of the grayscale images belongs to the Far West Station in 2019.	46
Figure 4-4 The architecture of the convolutional neural networks used in this thesis	47
Figure 5-1 The rates of training loss and accuracy have been achieved during the training time of CNN.	50
Figure 5-2 The difference between attacked and clean data.	51

CHAPTER ONE

INTRODUCTION AND THESIS OUTLINE

With the increase of machine learning applications, cyber-attacks on these applications have also increased in recent years. Even if the machine learning models are used in many fields including transportation and communications, social media, product recommendations, dynamic pricing, and fraud detection, they are still vulnerable to cyber-attacks [1].

Numeric vectors are accepted as inputs by Machine Learning algorithms [2]. When an input is designed in a particular way to obtain the incorrect result from the model, it can be called an Adversarial Attack. Adversarial Attacks applied to Machine Learning models are one of the cyber-attacks. When machine learning models become very sensitive to minor changes in inputs and in Machine Learning security, exploiting this sensitivity in order to change the algorithm's behavior is a major issue, and may cause significant damage to the system [3]

Adversarial examples can be dangerous for computer systems based on convolutional neural networks to perform data processing and analysis. For instance, a computer program that reads handwritten numbers can wrongly interpret them as belonging to a different number [4]. In addition, adversarial examples have the potential to fool a content filter that is present on a website. A user may publish content that is in violation of the guidelines for that website; but, if some adversarial noise was applied to the image, the content filter could not appropriately identify that image as being in violation of the criteria. These attacks may also be used in the real world, but with considerably less effectiveness

because of the environment in which they are used. For example, stickers and painted "trap" stop signs allowed Eykholt et al. [5] to manipulate stop signs and cause them to fail. The signs were misinterpreted as 'Speed Limit 45' signs, which is their intended categorization. Deep Neural Networks could not learn as humans see and have difficulty detecting harmful data.

Adversarial attacks can be classified into two categories: targeted and untargeted attacks. The goal of a target class owned by the targeted attack is to make the dataset misclassified by creating adversarial examples. On the other hand, the untargeted attack does not have a target class to misclassify the image. The goal of an untargeted attack is to make the classifier give an incorrect result [6]. It has been shown that while untargeted attacks are not as effective as targeted attacks, but they require less time. Even though targeted attacks are more useful in changing the model's predictions, they are costly in time [7].

The purpose of this thesis is to analyze different gradient-based adversarial attacks on a convolutional neural networks model designed to classify whether data from ERCOT [8] belongs to West Station or Far West Station and to showcase how much attacks may affect the model.

We explore the effect of adversarial attacks on Deep Learning model and give an idea for future work in order to design more robust models that could resist adversarial attacks. The main contribution of this work is as follows:

- The effects of six types of gradient-based adversarial attacks, including FGSM [9], BIM [10], PGD [11], Deep Fool [12], One Pixel [13], Sparse Fool [14] are provided.
- It is shown how previously proposed adversarial attacks mathematically affect the classifier and how much they affect the accuracy of the model.

Chapter two discusses related works and the background and introduces briefly the background of adversarial attacks and describes in detail the six gradient-based attacks used for the experiment. Chapter three introduces convolutional neural networks. Chapter four describes the images how the data is converted to the images to be used as a model for all attacks. Finally, in chapter five, the experimental results are described and followed by future work and a conclusion.

CHAPTER TWO

BACKGROUND OF ADVERSARIAL ATTACKS AND LITERATURE REVIEW

2.1 Introduction

Since the work of Szegedy [15], many strategies to create adversarial examples have been developed. Most strategies work by minimizing the distance between an image and its adversarial counterpart, which causes the classifier to misclassify the data. Many strategies use the gradients of the model to determine the perturbations required to generate an adversarial example.

The first adversarial example, presented by Szegedy, made use of the fact that neural networks are vulnerable to minor perturbations in their inputs. They hypothesized that the explanation was due to high non-linearity within the neural networks layers, resulting in a ‘blind spot’ within the training part.

The authors looked for exposing this blind spot utilizing box-constrained L-BFGS to minimize the following cost function with respect to r .

$$loss(f(x + r), l) + c \cdot |r|$$

Equation 2.1

In the Equation 2.1, the original image is represented by the variable x , adversarial perturbations added to the image is represented by the variable r , which guarantees to stay in the range $[0,1]$, l represent the target class and c balances the distances between images and predictions. The authors were able to use this technique to maximize loss in the model and derive optimal perturbations for a convolutional neural network.

Many researchers have developed a number of strategies to create adversarial examples since its beginnings. They all have the same purpose in order to fool a classifier while breaking the proposed defenses. Image misclassification is the topic of at least twenty adversarial attack techniques at the time of this writing [16]. This thesis concentrates on rapid, gradient-based attacks despite the variety of adversarial attack methods. This is because it is not feasible to use most of these methods since the strongest attacks take a long period of time to compute and apply to the images. Instead, this thesis focuses on Fast Gradient Sign Method [9], Project Gradient Descent [11], Deep Fool [12], One Pixel [13], Sparse Fool [14] and Basic Iterative Method (Iterative Fast Gradient Sign Method) [10] used to attack the images and focuses on a comparison of the success of these attacks on convolutional neural networks. For the experiment, the background of selected six different gradient-based attacks is described in the following sections.

2.2 Fast Gradient Sign Method

With the Fast Gradient Sign Method, or FGSM Szegedy et al. [15] have built on the prior work in which they theorized that the success of adversarial attacks is due to the strong non-linearity of the convolutional neural networks. Instead, Goodfellow et al. [9] show that the linear behavior of a neural network's layer in high dimensional spaces of the network's layers is more than sufficient to generate adversarial examples. Because of the linear structure of these networks, the authors say, a modest input can increase linearly overall of the input dimensions, resulting in a substantial change in the output. They refer to this as "accidental steganography" and conclude that if the input has a high enough dimensionality, any networks can have adversarial examples [9].

Linear models are vulnerable to adversarial attacks since they allow a perturbation to multiply with high dimensionality in the input data set. Equation 2.2 provides an explanation for this behavior. Using an image with an 8-bit channel, Goodfellow et al. demonstrate how a simple adversarial example could be created in the article. It is expected that any change in a pixel value smaller than the precision of the image, which in this case is a pixel value less than $1/255$, would have no effect on how the classifier classifies an image. This indicates that as long as $\|\eta\|_\infty < \epsilon$, where ϵ is the maximum possible value under the precision and the perturbation η , the classification outcome should be unaffected. Theoretically, this modification should be rejected. Nonetheless, in high dimensional spaces the perturbation η will be multiplied by the weight vector ω , which has high dimensionality n and has the same average magnitude m . If the perturbation is selected to be sign (ω), this will cause the activation to expand by as much as ϵmn . To put it another way, should the extremely small perturbations be designed to cause to maximize the loss in input x , these perturbations will accumulate and cause a significant change in the output of the model.

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

Equation 2.2

Because of this linear behavior, Goodfellow et al. were able to develop a rapid and cheap method to generate adversarial examples by utilizing the gradient of the model. The FGSM could be used to generate adversarial example x' , as shown in the formula below. x represents the original image, ϵ represents the amount by which the pixel intensity should be shifted. $\nabla_x J$ represents the gradient of the loss with respect to the input x , the vector of

true labels is y , θ represents the vector of model parameters, and the cost function of the model is $J(\theta, x, y)$.

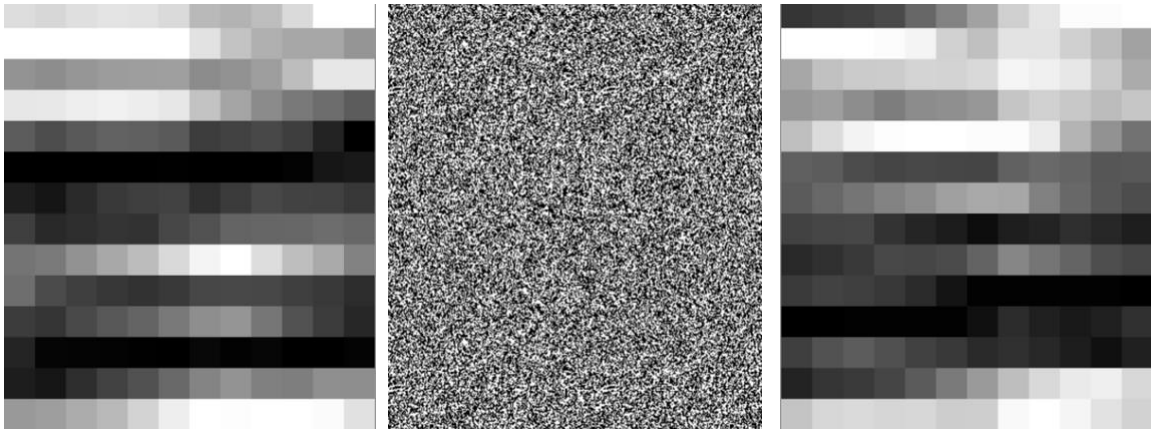
$$x' = x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y))$$

Equation 2.3

This equation makes it easier to generate adversarial examples for high-dimensional inputs. The principle of the FGSM is to maximize the loss function with respect to the input for the purpose of impact on the activations within the model's layers by changing just the input. In this technique, the model's other parameters remain constant while the input is updated. In the Figure 2-1, an example of FGSM attack in action on an image from the dataset used in this research is shown. Due to the adversarial noise, this data belongs to the West Station, this image is misclassified as Far West Station.

2.3 Project Gradient Descent

Project Gradient Descent or PGD which is another kind of adversarial attacks used in this thesis [11]. It was developed by Madry et al. in attempt to produce the most resistant deep learning model feasible against *first-order adversaries*. A first-order adversary possesses first-order knowledge of the network being attacked, which means that it has access to the model's gradient and can maximize the loss relative to the model's input. Additionally, FGSM is a first-order adversary, but PGD is a far stronger attack than FGSM.



Original Image

FGSM Adversarial Noise

Adversarial Image

Classification: West Station. at:0.009

Misclassification: Far West

Figure 2-1 The first image which belongs to the West Data misclassified as the data of the Far West Station because of the FGSM attack.

This is because PGD is known as an iterative version of FGSM or I-FGSM, with the main difference being that PGD begins in a tiny ‘ball’ area surrounding the gradient relative to the input [17].

When it comes to result, PGD is considered a constrained optimization problem because, while it takes steps, it checks to ensure that it does not move outside of the L -ball, which is often an L_∞ or L_2 ball. The difference in pixels between the adversarial image and the original image is the L -ball, where the difference is measured in pixels. The L_2 ball represents the distance between the original image and the adversarial image in terms of the Euclidean distance. The L_∞ distance indicates the difference between the largest pixels in the images. If the gradient steps performed throughout the process lead outside of the L -ball, the result must be reflected to the constrained area. The purpose of adhering to this limitation is to protect the adversarial image from being too visually different from the original image. This constraint ensures the creation of the adversarial example, which fools the neural network but seems to viewer to be the original class.

Equation 2.4 shows the PGD. PGD is different from FGSM because PGD computes the gradient of the cost function just once and then applies the transformation. PGD moves in tiny steps, computing a new gradient for each step. It adds a little perturbation to each pixel in the input at each step. As a result, PGD is the most effective method for determining the maximum of the loss function, which makes PGD a stronger attack than FGSM and explains why Madry et al. say that it is a ‘universal’ first-order adversary,

implying that a neural network resilient to its attacks is also resistant to all other first-order attacks.

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

Equation 2.4

x^{t+1} is the image with current time step t , S represents the perturbation distance from x , the constraint is Π_{x+s} that must be reflected, and α is the small perturbation. The remaining terms are the same with equation for the FGSM.

2.4 Deep Fool

The Deep Fool was created by Moosavi et al. [12] as a way to generate minimal perturbations in order to misclassify the data and determine the resilience of neural networks against adversarial attacks. Since DeepFool attack is an untargeted attack, it does not force the input to be in a specific class. These two attacks are motivated in the paper: the need to develop a more straightforward and precise approach for evaluating robustness, as well as the desire to construct an image with more ideal, less noticeable perturbations.

The DeepFool attack is theoretically fundamental. Starting with a binary classification problem:

$$f(x) = w^T x + b$$

Equation 2.5

The authors assumed that the nature of neural networks is linear, and the DeepFool attack splits the class space into hyperplanes and projects the current input onto the current class, i.e., the class space closest to the current class. Known as the minimum perturbation, they end up dramatically surpassing this value in order to generate an adversarial example. The result is an image that is practically indistinguishable from the original image due to the fact that the adversarial perturbations have been designed to be as minimum as possible. This technique also includes an iterative strategy in order to account for the non-linearity of neural networks, which is repeated until an adversary is created. It is proven to generate an adversarial example by using the L_2 norm in an iterative process until

$$\text{sign}(f(x)) \neq \text{sign}(f(x + r))$$

Equation 2.6

where r is the smallest amount of perturbation required. The authors also suggest that their method could be expanded to the multiclass classification and claimed that their approach could be used with the L_∞ norm as well. The DeepFool attack was able to generate adversarial examples using smaller perturbations compared to GoodFellow, because the average perturbations were five times smaller than the perturbations of the FGSM attack. As a result, The DeepFool computes a more optimal adversarial perturbation and can be considered as one of the sophisticated attacks.

2.5 One Pixel

In a One Pixel attack, it is necessary to maliciously alter an image's classification from its original label to the target model by manipulating a single pixel [13]. The One

Pixel attack is detrimental to the ability of Deep Learning model-based information systems to provide a high level of performance guarantee. In certain cases, modifying just one pixel in an image might cause the classification of the image to be changed to an irrelevant label, resulting in performance degradation of Deep Learning model-based applications and even more significant effects. So, the classification of images is altered to completely unrelated labels with the single change of a pixel [13]. The explanation of above, generating adversarial images can be defined as a constraint-based optimization problem. The authors assume a vector where each scalar element containing one pixel represents an input image. f which receives n -dimensional inputs is the target image classifier ($x = x_1, \dots, x_n$) and t is the correct classification of the original image. x 's probability belonging to the class t is thus $f_t(x)$. The vector $e(x) = e_1, \dots, e_n$ represents an additive adversarial perturbation in accordance with the input vector x , the target class t and maximum modification restriction L . The aim of the adversaries in the targeted attacks is to discover the optimal solution $e(x)^*$ to the following problem:

$$\begin{aligned}
& \underset{e(x)^*}{\text{maximize}} && f_{adv}(x + e(x)) \\
& \text{subject to} && ||e(x)|| \leq L
\end{aligned}$$

Equation 2.7

The problem means identifying two values: 1) Identifying the dimensions that need to be perturbed and 2) The strength of the alteration for each dimension that corresponds to that dimension. The authors described their approach that is slightly different is showed in the Equation 2.8 where d is a small number taken as $d = 1$ for One Pixel Attack.

$$\begin{aligned}
& \text{maximize} && f_{adv}(x + e(x)) \\
& && e(x)^* \\
& \text{subject to} && ||e(x)||_0 \leq d
\end{aligned}$$

Equation 2.8

Previous studies frequently alter a part of all dimensions, however in this technique just d dimensions are modified, with the remaining dimensions of $e(x)$ being set to zeros in all other dimensions.

2.6 Sparse Fool

The Sparse Fool Attack which is introduced by Dezfooli [14] is inspired by geometry that uses of the low mean curvature of the borders to compute adversarial perturbations in a fast and efficient manner. The l_p minimization adversarial perturbations are the most frequent form of adversarial perturbations for image classification tasks because they are the easiest to evaluate and optimize. The authors define as adversarial perturbation, for a provided classifier and an image $x \in \mathbb{R}^n$, the smallest perturbation r which causes the predicted label $k(x)$ of the classifier to be such that:

$$\min ||r||_p \text{ s. t. } k(x + r) \neq k(x)$$

Equation 2.9

Most known adversarial attack algorithms are able to address the optimization problem. When perturbations are sparse, the goal is to lower the amount of perturbed pixels

required to deceive the network. This corresponds to reducing the value of $\|r\|_0$ in the Equation 2.9. For NP-hard problems, there is no guarantee that a global minimum could be reached generally [18]–[20]. There are several approaches [20], [21] to avoid this problem's computational burden, the most popular of which is the l_1 relaxation, the equivalent convex l_1 problem may be used to approximate the minimization of $\|r\|_0$ under linear constraints [22], [23]. So, the authors seek an efficient technique to take advantage of such relaxation in order to solve the optimization problem in Equation 2.9.

After addressing the optimization problem's generic form, sparse adversarial perturbations can be produced as:

$$\begin{aligned}
& \underset{r}{\text{minimize}} && \|r\|_1 \\
& \text{subject to} && k(x + r) \neq k(x) \\
& && l \leq x + r \leq u
\end{aligned}$$

Equation 2.10

where $l, u \in \mathbb{R}^n$ denote the lower and upper bounds of the values of $x + r$, such that $l_i < x_i + r_i \leq u_i, i = 1, \dots, n$. Since the authors look for a quick and effective resolution of the problem, they concentrate on the decision boundary's geometric properties, especially on its curvature.

The color green in Figure 2-2 denotes the l_2 -Deep Fool adversarial perturbations generated after each iteration [14]. The method converges after two rounds, with a full

perturbation of $r = x^{(2)} - x^{(0)}$. The minimum perturbation r is modified by projecting the current datapoint $x^{(i)}$ onto the predicted hyperplane using the l_1 projection method:

$$r_d \leftarrow \frac{|w^T(x^{(i)} - x_B)|}{|w_d|} \cdot \text{sign}(w_d),$$

Equation 2.11

Where d represents the index of the highest absolute value of w for which there has not been a previous use.

$$d \leftarrow \arg \max_{j \notin S} |w_j|$$

Equation 2.12

Right before going to the next iteration, the authors confirm that the values of the subsequent iteration $x^{(i+1)}$ are correct. As a result, they utilize the projection controller $Q(\cdot)$, which realigns the out-of-bounds values of the updated point by projecting $x^{(i)} + r$ onto the box-constraints specified by l and u and the new iteration $x^{(i+1)}$ becomes $x^{(i+1)} \leftarrow Q(x^{(i)} + r)$. It is worth noting that the boundaries l, u are not restricted to representing an image's dynamic range but may be generalized to satisfy any equivalent constraint. Algorithm 1 is a summary of the algorithm that can be used to clarify the linearized problem.

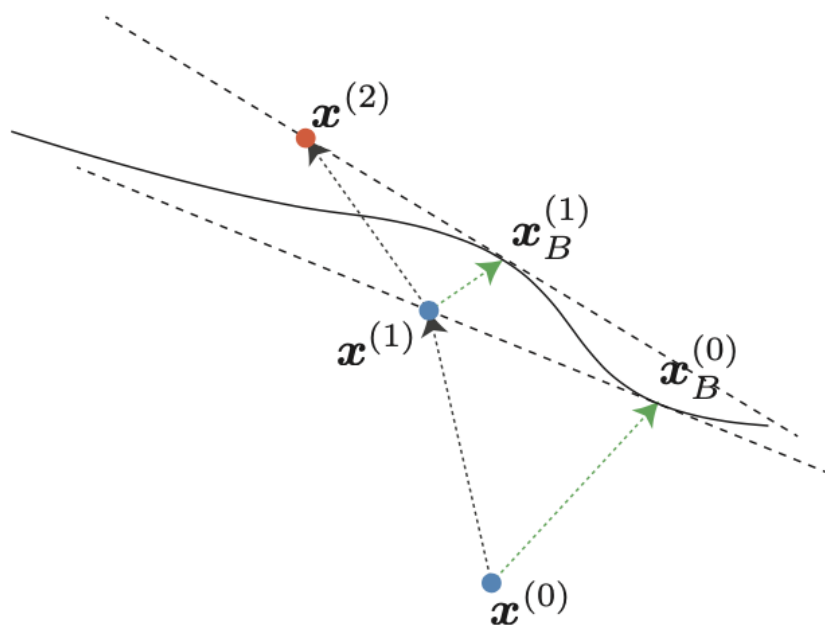


Figure 2-2 The illustration of Sparse Fool method [14]

Algorithm 1: SparseFool**Input:** image x , projection operator Q , classifier f .**Output:** perturbation r

1. Initialize: $x^{(0)} \leftarrow x, i \leftarrow 0$
2. **while** $k(x^{(i)}) = k(x^{(0)})$ **do**
3. $r_{adv} = \text{DeepFool}(x^{(i)})$
4. $x_B^i = x^{(i)} + r_{adv}$
5. $w^{(i)} = \nabla f_{k(x_B^i)}(x_B^i) - \nabla f_{k(x^{(i)})}(x_B^i)$
6. $x^{(i+1)} = \text{LinearSolver}(x^{(i)}, w^{(i)}, x_B^i, Q)$
7. $i \leftarrow i + 1$
8. **end**
9. **return** $r = x^{(i)} - x^{(0)}$

2.7 Basic Iterative Method (BIM)

The Basic Iterative Method aims at extending FGSM's logic [10]. Using a smartphone, the researchers illustrate how adversarial examples may fool a neural network in a real-world scenario. This method generates samples, by means of applying FGSM k times with a small size $a \geq \epsilon/k$ and each step's feature values are clipped to make sure that they do not go beyond ϵ of the source example, which means that the values were higher than the maximum perturbations threshold of ϵ . This method is also referred to as the Iterative-FGSM or IFGSM. BIM is showed mathematically in Equation 2.13 [10].

$$x_{adv}^0 = x$$

$$x_{adv}^{N+1} = \text{Clip}_{x,\epsilon}\{x_{adv}^N + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x_{adv}^N, y))\}$$

x : benign input

x_n^{adv} : n^{th} adversarial example

a : step size

$Clip_{x,\epsilon}$: Clipped values when $x^{adv} > \epsilon$, s.t. $\epsilon > 0$

Equation 2.13

The final adversarial examples were created after several iterations of the technique were performed.

CHAPTER THREE

CONVOLUTIONAL NEURAL NETWORKS

In this chapter, the neural networks and its terminology are briefly explained, and some examples are given.

3.1 Perceptron

The development of neural networks and the area of deep learning originated with hard approximation for how the brain processes [24]. An enormous number of neurons and synapses collaborate to accomplish the brain's duties and tasks that are required. It is helpful to begin with the simplest version of a neural network, a perceptron, which represents a single neuron, in order to better explain large neural networks.

When a neuron gets fired, a stimulus generally triggers it and signals are being sent to the neuron's axon to another neuron's dendrites, which is known as synaptic transmission. After, the new neuron can activate and induce another neuron gets to be fired, performing the process again in the system. A perceptron graphically is represented in Figure 3-1.

An input vector of real-valued inputs is passed to a perceptron, which calculates a linear combination of these inputs and then outputs either 1 if the result is larger than a certain threshold, or -1 if the result is less than that threshold. Furthermore, given a set of inputs x_1 through x_n , the perceptron computes the output $o(x_1, \dots, x_n)$,

$$o(x_1, \dots, x_n) = \begin{cases} 1, & \text{if } w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n > 0 \\ -1, & \text{otherwise} \end{cases}$$

Equation 3.1

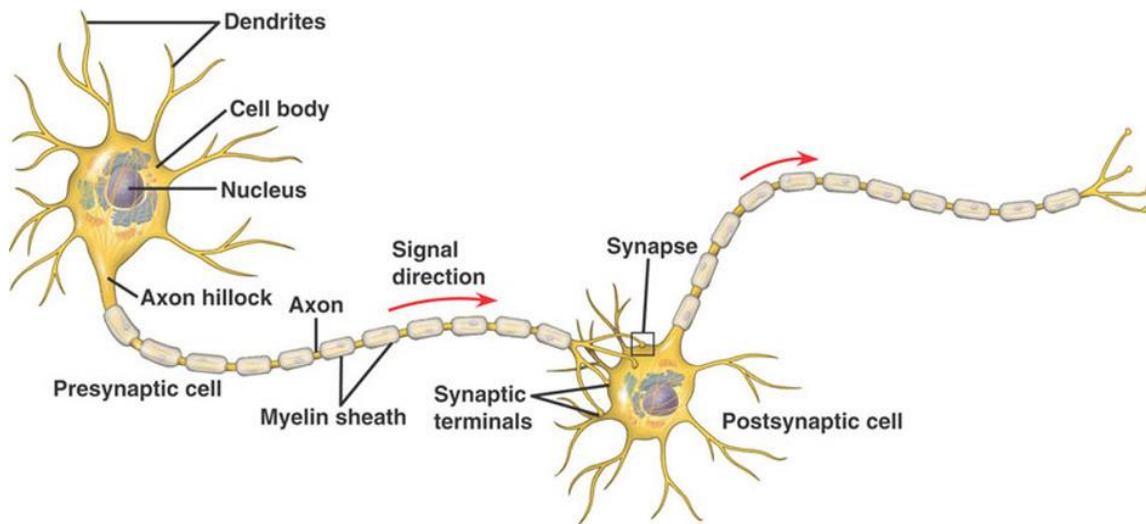


Figure 3-1 A model of a neuron that places an emphasis on the structure that links the axon and the dendrite [25].

where w_i is a real-valued constant or *weight* that specifies the contribution of each input x_i to the perceptron output [24].

The amount $(-w_0)$ is the threshold that must be exceeded by the weighted combination of inputs $w_1x_1 + \dots + w_nx_n$ for the perceptron to output 1. This process can be seen in the Figure 3-2.

3.2 Multilayer Perceptron

The perceptron model that has been looked at so far can only shows linear hypotheses. To surpass linearity, it can be looked at the multilayer perceptron (MLP), which is a generalization of the perceptron and what most people call a neural network. The focus of this discussion will be on the fully linked feedforward MLPs, also called the standard or vanilla MLP. Other neural networks, such convolutional neural networks, may be made to operate as special cases of the ordinary MLP, as shown in the following section. The network of perceptrons that composes an MLP can be seen in Figure 3-3 [26]. In addition, if it is considered each perceptron to be a "neuron," it can be understood where the term "neural networks" came to be used. The terms "nodes" and "neurons" will be used interchangeably in the following discussion of perceptrons. A first "hidden layer" of perceptrons is provided with the inputs, with each perceptron receiving each input individually (where does "fully connected" come from?). After that, each perceptron's output is sent through a non-linear thresholding function.

In order to make a neural network more comprehensive, fully connected layer can be stacked fully connected layers. By doing so, the approximation quality of the neural

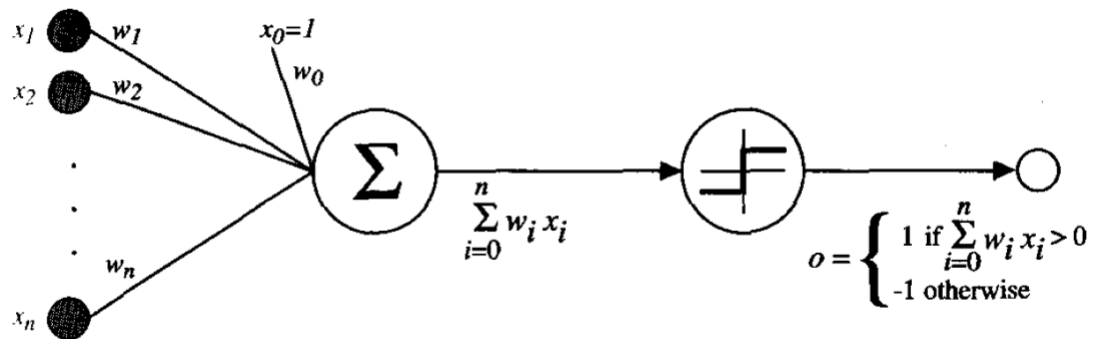


Figure 3-2 The graphical representation of a perceptron.

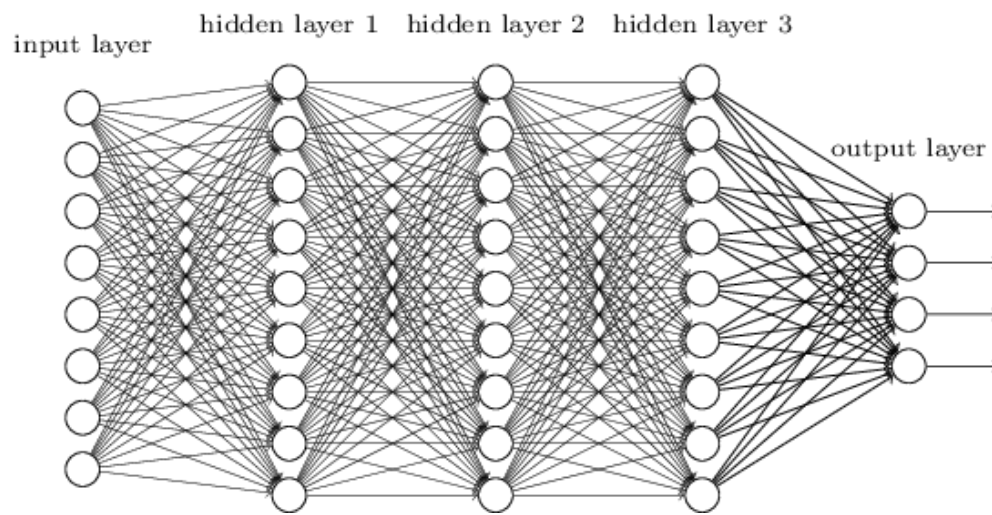


Figure 3-3 Depicts the network of perceptrons that constitutes an MLP.

network may be increased. However, the number of stacked linear fully connected layers does not matter, a linear model would be the result. Non-linearity can be achieved by applying threshold function such as ReLU [27], tanh or sigmoid [28]. By definition, an activation function determines whether or not to activate ("fire") a neuron. It introduces non-linearity into a neuron's output. Without activation functions, a neural networks is nothing more than a linear regression model [29] .

The ReLU function is the threshold that is most used in practice today [30]. It is defined as:

$$ReLU(x) = \begin{cases} x, & \text{if } (x) \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Equation 3.2

Sparsity is a result of ReLU [30]. It also correlates into more models with increased predictive ability and reduced noise or overfitting. Neurons are more likely to interpret significant information in a sparse network. For instance, a neuron capable of recognizing facial images of human should never be triggered if the image seems to be of a structure [31]. Consequently, when an input is given, just a small number of the nodes are not zero (they are active), and a linear function shows every potential subset. This makes it possible for gradients to flow more smoothly, which makes the process of training go faster. Additionally, the ReLU is easier to figure out than the hyperbolic and exponential versions [30]. In this thesis, ReLU is used as an activation function.

3.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) were presented over two decades ago as a version of vanilla MLPs that were modeled after the visual cortex of animals [32]. The fundamental logic was that animals' visual cortex is composed of layers of simple and complex cells, and as an image is processed via the cortex, increasingly rich features of the image are discovered [33]. Pooling layers (complex cells) and convolutional layers (simple cells) are form of CNNs and to train even high level and complex features, convolutional layer filters have been used [28]. CNNs are designed specifically for processing grid-like data including information such as one-dimensional audio signals, two-dimensional images, and three-dimensional movies. Since the images are used in this thesis, a discussion about images is presented:

An image is composed of pixels. Three color components: red, green, and blue - make up each pixel in the RGB model. Each element's value is typically between 0 (no color) and 255 (maximum saturation) [34]. Due to the higher cost of creating a color image than a grayscale image, color photos have been transformed to grayscale images, for example, in order to reduce the cost of generating low-cost edition books. Additionally, a color-blind viewer must have a picture that is of the same quality as a color image in order to interpret the information in the same way that a normal person does when viewing a color image. Consequently, there are number of image processing applications that need to change a color image to a grayscale image for different reasons [35]. If each Red, Green, and Blue color has a value of 8 bits, then the RGB color space has a value of 24 bits and provides 16,777,216 colors. A 24-bit representation of the color of a pixel in a color picture.

The grayscale picture is represented by an 8-bit brightness value. The brightness of each pixel in a grayscale image ranges from 0 to 255. In order to convert a color image to a grayscale image, the RGB (24-bit) data must be converted to grayscale values (8 bit) [35]. Since the gray scale images are used in this thesis, one of the Far West Images in 2015 is represented in the Figure 3-4. The details are provided in the data preparation section.

As it can be seen in Figure 3-5, the architecture of CNN is the typical for all models [36].

To understand what a CNN is, it is necessary to first analyze the structure of a single layer. In general, each layer of a CNN is composed of a collection of learnable "filters" (which correspond to the network's weights), a non-linear threshold (similar to that found in MLPs), and sometimes a pooling (also known as subsampling) operation.

Overall, after taking in a single input image, the model applies convolutional and pooling layers to it before generating the output image through a sequence of fully connected layers. These are described in the following subsections.

3.3.1 Convolutional Layers

A convolutional layer extracts features by convolving the input with a filter. This produces a feature map suitable for training the networks. Simple shapes such as edges or more sophisticated shapes such as structures are considered features. Convolution techniques can be used to make convolutional layers that can be used to implement neural networks to image data.

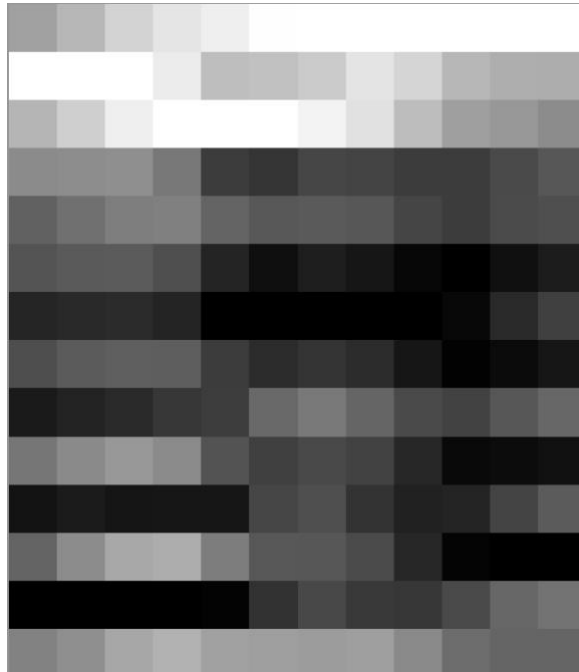


Figure 3-4 One of the images that belongs to the Far West Station in 2015.

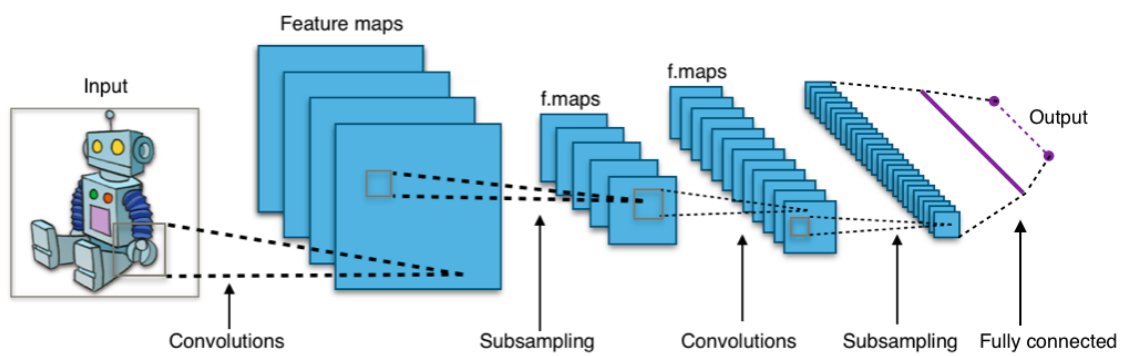


Figure 3-5 Typical CNN Architecture[36]

Convolution is the key operation. In the presence of a $n \times n$ filter (which is just a weight matrix):

$$F = \begin{bmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nn} \end{bmatrix}$$

And an image's $n \times n$ patch:

$$I = \begin{bmatrix} i_{11} & \cdots & i_{1n} \\ \vdots & \ddots & \vdots \\ i_{n1} & \cdots & i_{nn} \end{bmatrix}$$

The term "convolution" refers to the process of combining the filter and the picture patch:

$$F * I = \sum_{j=1}^n \sum_{k=1}^n w_{jk} i_{jk}$$

Equation 3.3

The goal is to slide the filter over the image, convolving it with each patch of the image as it does so. By selecting a stride, the amount of pixels moved may be changed by the filter each time it slides (so with a stride of 1 the filter sees every possible patch of the image). Each convolution generates an output, and the sum of all the outputs from a filter's convolutions with an image generates a feature map, or the output of that layer that also has a matrix structure.

In Figure 3-6, between a 3 by 3 section of the input image's matrix and a 3 by 3 filter matrix, a dot product convolution (called multiplication) is performed. The components of the resultant matrix are combined, and the result is the output value of the feature map ("destination pixel"). The input matrix is shifted over by the filter, which performs the dot product convolution on every leftover combination of 3 by 3 sized areas,

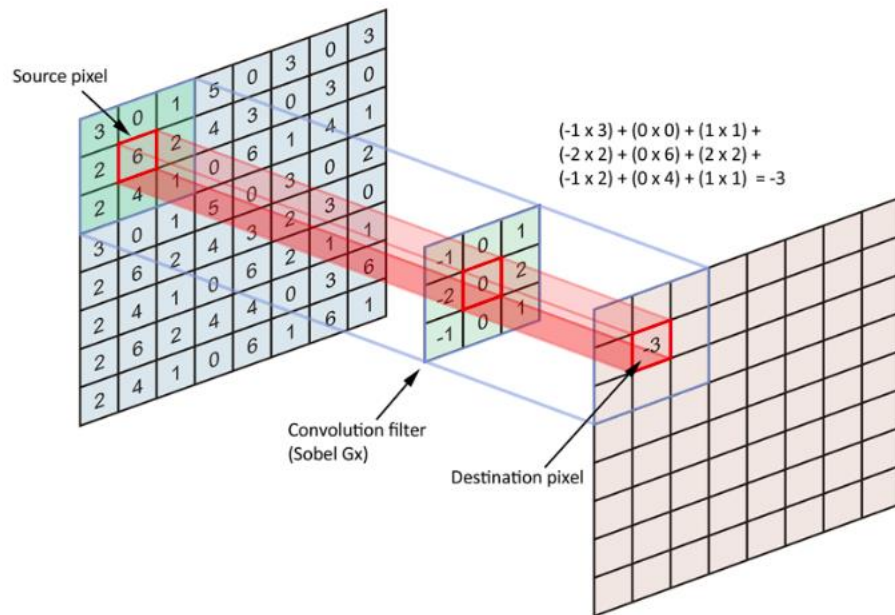


Figure 3-6 It shows the 3x3 matrix multiplication.

thus completing the feature map.

A concrete example will help to illustrate this process. The outcome of convolution of a 3x3 filter is shown in portion of Figure 3-7 below:

$$F = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

With an image patch

$$I = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix},$$

As indicated on the feature map by

$$\mathbf{F} \star \mathbf{I} = 1 * 1 + 1 * 0 + 1 * 1 + 0 * 0 + 1 * 1 + 1 * 0 + 0 * 1 + 0 * 0 + 1 * 1 = 4$$

Equation 3.4

The resulting feature map is then shown in section b of Figure 3-7 after the filter is convolved across the full image.

The generated feature map is then subjected to a non-linear thresholding operation by element. As with MLPs explained section 3.2 Multilayer Perceptron, the ReLU function is the most frequently used threshold for CNNs. As a last step, the feature map is put through a pooling process described in the following subsection.

3.3.2 Pooling Layer

Similar to the strides, pooling is another technique for lowering the dimensionality of a layer, especially its height and width while conserving its depth. This is advantageous

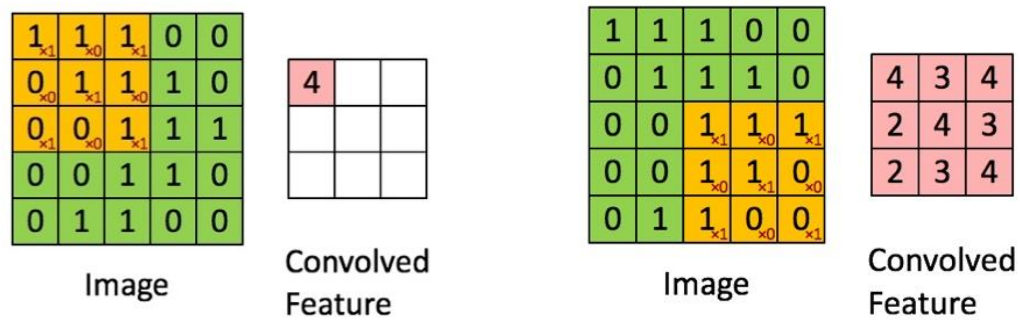


Figure 3-7 (a)The resulting the first convolution (b)The result of the feature map
A convolution of an image with a filter is demonstrated [37].

because it reduces the amount of CPU power necessary to analyze the data while still extracting the most prominent characteristics from feature maps. This time, though, they use simpler functions instead of an activation function, bias and weight to get the same results. Pooling layers are classified into two types: maximum pooling and average pooling. Maximum pooling layers are used in conjunction with convolutional layers to enable down sampling of the feature maps obtained from the convolutions. Maximum pooling is necessary to ensure that the model is resilient to the image content's position. Pooling is typically 2 by 2 with a stride of 2, which means that it will take the maximum value of each 2 by 2 patch in the feature map and move it two pixels at a time, returning a collection of smaller feature maps.

The values that are contained within the patch for each channel are averaged out when using the average pooling method. Pooling is commonly performed in 2 *by* 2 patches of a two-dimensional feature map with a stride of (2,2). Calculating the average for each region of the feature map is considered pooling. This implies that each of the feature map's 2 *by* 2 squares are down sampled to the square's average value. An example of maximum and average pooling can be seen in the Figure 3-8 [38].

Both pooled and networks without pooling implement and regulate deformation stability in large part due to the smoothness of learned filters [39]. Because of this reason and since very large size images, for example millions of images, could not be provided, it is not required to use a pooling layer for this thesis.

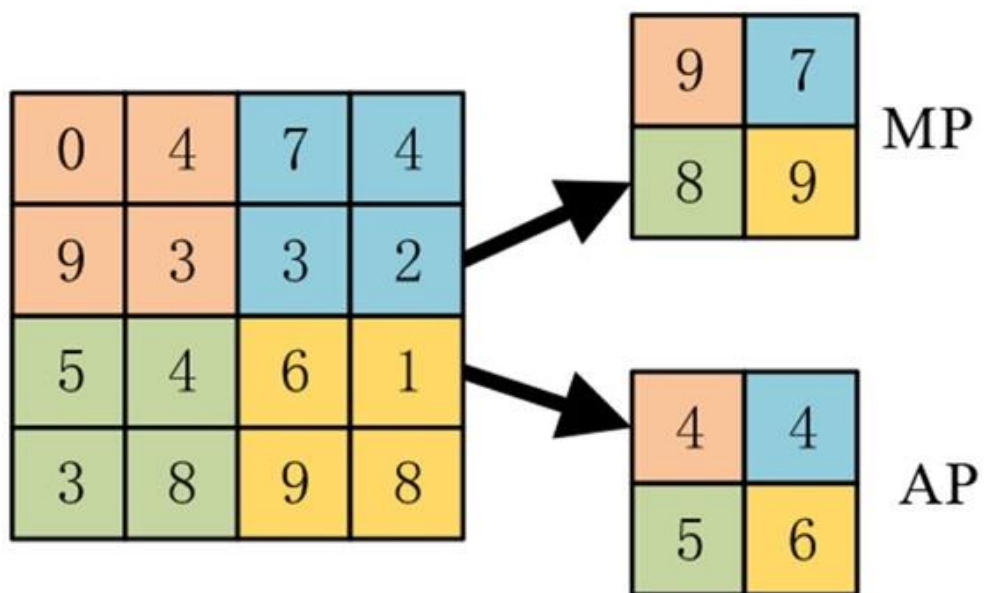


Figure 3-8 Maximum and Average pooling example [38].

3.3.3 Fully Connected Layer

As implied by the name, the previous layer's nodes need to be connected to the next layer's nodes in order for the two to be fully connected. In order to construct the output matrix, the input matrix must first be arranged in the form of a column vector. This column vector is then put into a collection of layers that are fully connected to one another, which means that classification occurs at fully connected layers [40].

Assume there are two sequential layers: $l^{(k-1)} \in \mathbb{R}^{m^{(k-1)} \times 1}$ and $l^{(k)} \in \mathbb{R}^{m^{(k)} \times 1}$. The weight matrix that links these layers would be written as $w^{(k)} \in \mathbb{R}^{m^{(k-1)} \times m^{(k)}}$ to make sure that they are all connected. Figure 3-9 illustrates this structure [40]

The bias term ($b^{(k)} \in \mathbb{R}^{m^{(k)}}$) is also included in most fully connected layers in order to account for the constants of the system. Simply by applying weight and bias, a fully connected layer's output, $o^{(k)}$, could be easily determined utilizing the layer function $\psi^{(FC)}$, which is shown below.

$$o^{(k)} = \psi_{(K)}^{(FC)}(o^{(k-1)}) = (o^{(k-1)})^T w^{(k)} + b^{(k)}$$

Equation 3.5

$\psi_{(K)}^{(FC)}$'s computational complexity is

$$\mathcal{O}(\psi_{(K)}^{(FC)}) = \mathcal{O}(m^{(k-1)}m^{(k)})$$

Equation 3.6

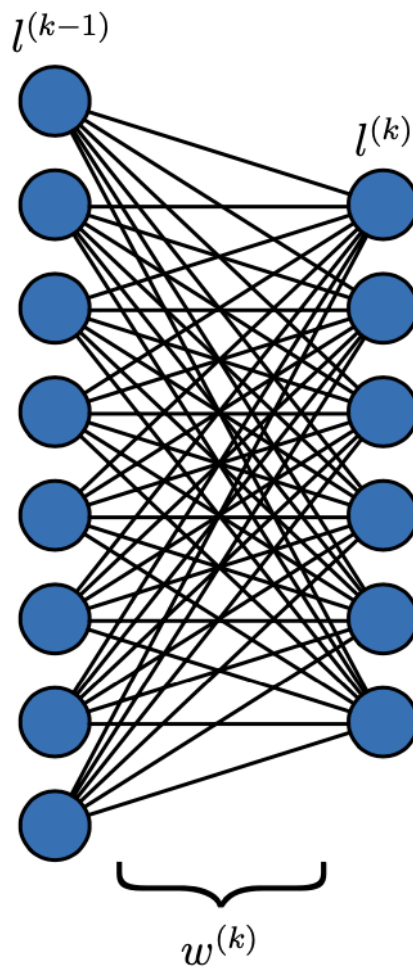


Figure 3-9 A graph illustrating the connections between the two fully connected layers[40].

Activation functions (like tanh or ReLu explained under section 3.2) are applied to each fully connected layer, while the output full connected layer is processed through SoftMax. Before creating a final class output in the SoftMax layer, fully connected neural networks are the fully connected layer that update weights between neurons.

3.3.3.1 Softmax Cross-Entropy

By using loss or cost function, the quality of an approximation can be quantified. One way to better understand the idea of loss would be a salesperson's loss. Considering a customer paying a maximum of \$20 for a particular product, should the salesperson sell this item for \$14, the salesperson loses \$6 in potential profit. Or should the salesperson sell this item for \$24, the customer will not buy it and it costs the salesperson \$20. In this case, the salesperson's objective would be to minimize the loss in order to maximize the profit. One of the common loss functions is Softmax Cross-Entropy, which is described as when two approximations are compared, the one with the lower loss performs better at approximating the data. So, it is used to solve classification problems in which we are attempting to determine which class our data belongs to. The Softmax Cross-Entropy initially uses the input to determine the class probabilities. The definition of Softmax Cross-Entropy is:

$$p(i|\hat{y}_n) = \frac{e^{\hat{y}_{n,i}}}{\sum_{j=1}^N e^{\hat{y}_{n,j}}}$$

Equation 3.7

Then it is compared to the expected output ($y_n \in \mathbb{R}^N$), the loss of Softmax Cross-Entropy may be described by the Equation 3.8 [28]:

$$\mathcal{L} = CE(\hat{y}_n, y_n) = - \sum_{i=1}^N y_{n,i} \log(p(i|\hat{y}_n))$$

Equation 3.8

In Softmax Cross-Entropy, the softmax is used to turn the outputs of the nodes into probabilities. Therefore, it makes sense to use it for classification problems when the output data reflects probability distribution [41]. GoodFellow et.al gives an in-depth description of this technique [28].

3.3.3.2 Minimizing Loss

To get more accurate approximations, the parameters of the neural networks can be optimized. One frequently used technique for optimizing these parameters is Adam, which was introduced by Kingma et. al [42]. It is an optimization method that iteratively updates network weights using training data. When the authors provide the algorithm, they also detail the several advantages of utilizing Adam to solve non-convex optimization issues. The approach is simple to develop, computationally efficient, requires minimal memory, and is ideally suited for issues with a large amount of data and/or parameters. Adam is quite simple to configure, with the default configuration parameters resolving most issues. Additionally, the strategy is applicable to non-stationary targets and issues involving extremely noisy and/or sparse gradients [42], which proves the success of the attacks used in this thesis since the attack changes the gradient or makes the dataset noisy. Adam optimizer employs a hybrid of two gradient descent algorithms:

- **Momentum**

This approach accelerates gradient descent by taking the 'exponentially weighted average' of the gradients into account. If the averages are used, the algorithm moves faster toward the minima which is the best solution.

$$w_{t+1} = w_t - \alpha m_t$$

Equation 3.9

where,

$$m_t = \beta m_{t-1} + (1 - \beta) \left[\frac{\delta L}{\delta w_t} \right]$$

Equation 3.10

- **Root Mean Square Propagation (RMSP)**

Tieleman and Hinton [43] introduced RMSP to accelerate mini-batch learning. The learning rate in RMSP is adaptive depending on the moving average of the magnitudes of previous gradients. In other words, RMSP computes a moving average of the squares of the most current gradients, represented by (v) . As a result, recent gradients are given more weight.

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t + \varepsilon)^{\frac{1}{2}}} * \left[\frac{\delta L}{\delta w_t} \right]$$

Equation 3.11

where,

$$v_t = \beta v_{t-1} + (1 - \beta) * \left[\frac{\delta L}{\delta w_t} \right]^2$$

Equation 3.12

That is, the next value of w should be divided by the previous value of w by moving average of squares that have been squared recently for that parameter w , multiply it by α , and then subtract it from the previous value of w .

Additionally, RMSP performs well with large and redundant datasets [44].

- **Mathematical Explanation of Adam Optimizer**

Adam's method updates the exponential moving averages of the gradient(m_t) and squared gradient(v_t), which represent the first and second moment estimations, respectively.

By simply combining these two formulas in above, Equation 3.13 and Equation 3.14 can be obtained. As seen below, the hyper-parameters $\beta_1, \beta_2 \in [0, 1]$ govern the exponential decay rates of these moving averages:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[\frac{\delta L}{\delta w_t} \right]$$

Equation 3.13

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\delta L}{\delta w_t} \right]^2$$

Equation 3.14

Given that m_t and v_t which are first and second moment estimations, respectively are both set to 0 (using the abovementioned approaches), it is seen that they have a tendency to be 'biased towards 0' when both β_1 and β_2 converge to 1 ($\beta_1 \& \beta_2 \approx 1$). This optimizer

corrects for this issue by computing 'bias-corrected' $\widehat{m}_t$ and $\widehat{v}_t$ values. This is also done to maintain control of the weights while attempting to approach the global minimum, hence avoiding excessive oscillations when near it. The following formulas are used:

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\widehat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Equation 3.15

Intuitively, the gradient descent algorithm can be adjusted after each iteration to ensure that the process persists regulated and unbiased throughout. So, the bias-corrected weight parameters can be used in place of the usual weight parameters. When these parameters are incorporated into the general equation with updated parameters, we obtained:

$$w_t = w_{t-1} - \eta \left(\frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \varepsilon} \right)$$

Equation 3.16

CHAPTER FOUR DATA COLLECTION AND PREPARATION

4.1 Data Collection

There are two data sets utilized in this thesis that will be discussed in this part. The selection of these data sets was based on their increasing complexity and is obtained from the ERCOT website. The datasets are preprocessed to produce the adversarial images as explained below.

The ERCOT region has eight stations, where two Far West and West regions are chosen to implement CNN and the adversarial methodologies as discussed previously. Each of these 5-year electrical data sets from 2015 to 2021 is considered as a signal and these electrical data sets which are available on ERCOT website are converted into the images.

Since the number of images is not required when taken on a monthly basis, and the image sizes is very small when taken daily, they have therefore been taken on a weekly basis. A weekly data set consists of 168 hourly datasets. 14×12 images were created in order to place the 168 rows of hourly data exactly in the image. Since the 1-year period consists of 8784 hours of data, a total of 52 images with a size of 168 were created. While a pixel represents a row of electrical data, Python programming and its libraries such as Numpy are used to generate the pixels, which means that it is placed automatically by code. The image values have been converted into (Numpy) arrays and colored images, 14×12 size have been created as seen in Figure 4-1.

Since working with grayscale images is more convenient than (RGB) color images and such a large size of the images would demand a considerable amount of processing

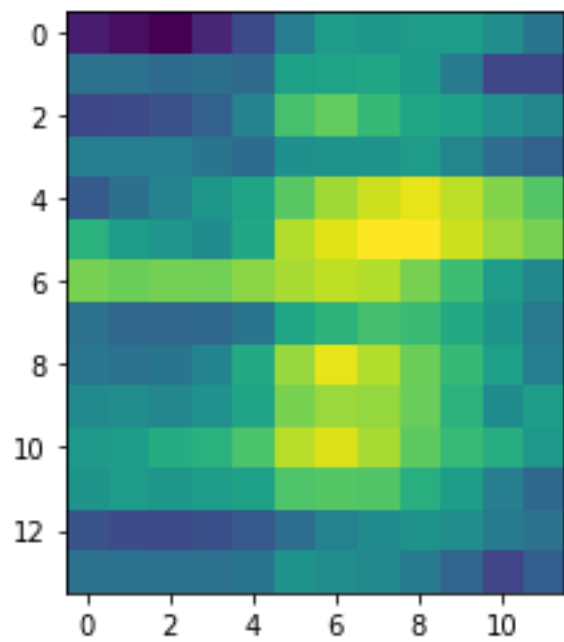


Figure 4-1 Electric load data is converted to RGB model.

time and these 24-by-24-pixel pictures already need far more processing power than the 8-by-8-pixel images, so increasing their size will simply exponentially lengthen the duration of the experiment, as explained under the Convolutional Neural Networks section, the RGB 24-bit images we obtain are converted to grayscale 8-bit images. Grayscale images have a range of 0 to 255 for the brightness of each pixel. So, these values are scaled between 0 and 255 and as can be seen in Figure 4-2.

Hundreds of grayscale images, where the values are seen as similar and the actual image is as given in Figure 4-3, are used to create the CNN model and to attack this image data set in order to generate adversarial images. Since these obtained images are different for each electrical data, they can be clearly noticed by deep learning models, although those of the West and Far West stations are barely noticeable to the human eye. The West station data is called *Class 0*, the Far West station data is *Class 1* in order to be able to classify and generate adversarial images from the created data set.

4.2 Convolutional Neural Networks Structure

The architectures of the convolutional neural networks used for each dataset are described in this section and are shown in Figure 4-4. Each data set's neural network has the same neural networks and there was a white-box attack on each data set's neural network. This model is divided up by fundamental components that include convolutional layers and fully connected layers. These are the most important components that go into making up a convolutional neural network. A convolutional layer extracts features by convolving the input with a filter. To train the network, a feature map is generated. Structures and basic forms like edges are examples of features.

	0	1	2	3	4	5	6	7
0	24.0983	64.1763	96.4937	118.219	128.892	147.371	148.098	168.571
1	105.769	151.566	165.966	149.754	123.716	45.9217	25.5379	41.6524
2	155.404	195.296	219.728	243.833	250.064	255	255	255
3	69.5677	95.314	103.278	90.0643	62.3384	29.0201	58.8955	97.6042
4	122.744	143.027	162.982	158.955	146.072	126.122	99.9659	89.4783
5	7.3679	32.0043	62.6501	95.0672	76.6453	73.37	101.559	107.859
6	188.214	210.711	193.74	171.894	163.131	149.515	128.671	133.064
7	82.1189	113.51	141.61	135.942	123.912	64.958	42.7995	49.7132
8	75.1309	109.555	121.373	116.034	104.877	96.6753	89.2612	116.701
9	123.49	175.8	204.222	198.875	188.49	159.964	121.356	129.968
10	255	245.986	255	255	255	217.859	166.302	180.825
11	116.733	100.779	60.3548	45.8775	24.9878	15.718	31.6285	54.3991
12	240.115	255	252.411	243.442	231.847	190.702	125.717	117.026

Figure 4-2 The pixel values of a grayscale image are scaled from 0 to 255.

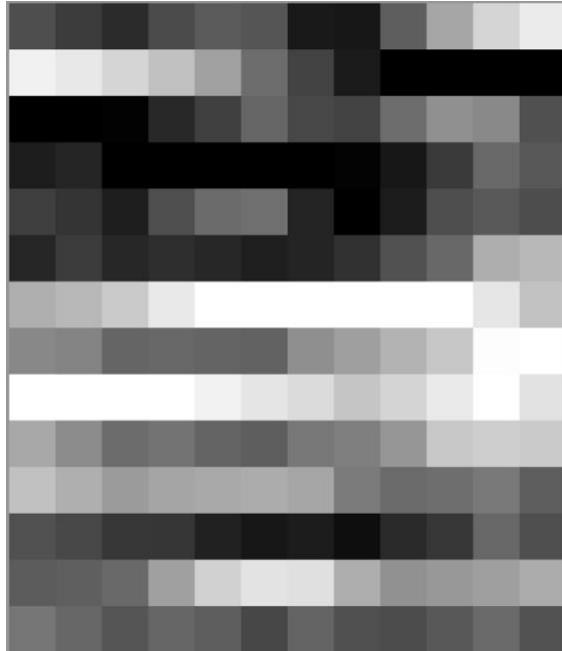


Figure 4-3 One of the grayscale images belongs to the Far West Station in 2019.

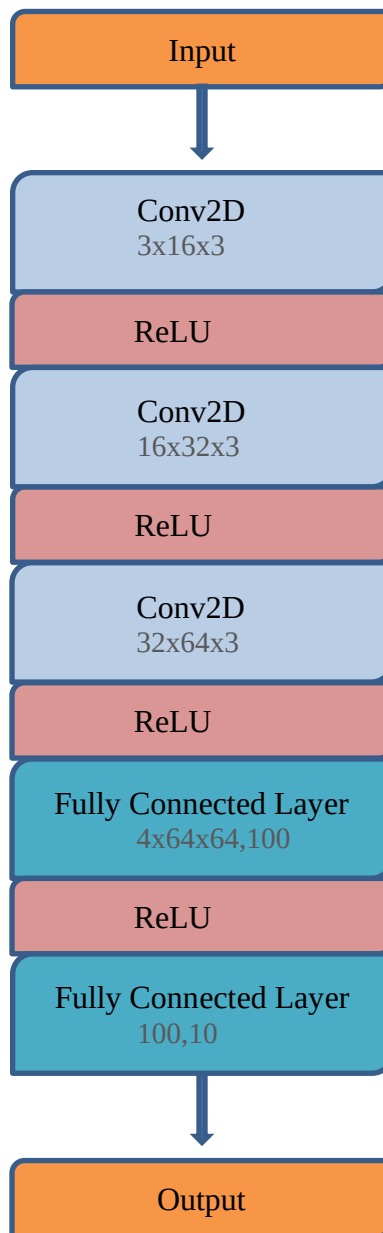


Figure 4-4 The architecture of the convolutional neural networks used in this thesis.

Structures and basic forms like edges are examples of features. Deeper within a convolutional neural network, more complicated characteristics are often extracted. In the linear layers, which are fully connected neural networks, the weights between neurons are modified before the softmax layer makes a final decision about the class.

As explained above, the inputs are normalized between 0 and 255. In addition to the normalization of the input, Rectified Linear Activation Unit, or ReLU is used as the activation function. This was selected based on claims that it can generate more accurate and timely results than conventional activation functions [30]. As a result, for electrical load data sets converted into images, the CNN structure is created with 30 epochs using 3 convolutional layers and 2 fully connected (linear) layers. The data of the West Station is classified, *class 0* and the Far West Station is classified, *class 1* as outputs. We successfully distinguish whether an image given to CNN belongs to *class 0* or *class 1*.

CHAPTER FIVE

EXPERIMENTATION RESULTS

In this research, some adversarial machine learning methods are used to show the vulnerability of convolutional neural networks and compare the adversarial attacks on CNN. To do that, as explained under the data preparation section, convolutional neural networks is used, and adversarial images are generated by the adversarial methods and the model is evaluated by using both attacked and normal data.

To train the model, every image is scaled to a value between 0 and 255. The neural networks model is trained in 30 epochs and this network's learning rate was set incredibly low, at 0.1%. Figure 5-1 illustrates this network's training process. The run achieves a high accuracy on the training set. Observing the model, it is evident that the accuracy does increase, reaching a peak of about 92% accuracy on the data set around the 30th epoch. With the increasing importance of adversarial attacks, different methods have been developed by researchers. 'Torchattacks' [45] which is a library for PyTorch [46] tool that lets us use adversarial attacks to create new adversarial examples and that incorporates many of the developed methods. All adversarial attacks were applied to two data sets in order to show their successes on the vulnerability of Convolutional Neural Networks by implementing this library, torchattacks. The same ϵ which is a literally small amount of adversarial noise at 0.009% added to the images is used for all attacks and the same step size, 20, -if attack has step size as a parameter-, is used for all of them, to get a fair comparison. In Figure 5-2, the difference between clean data and attacked data can be seen, by adding some perturbations to the clean image, the adversarial images are generated to

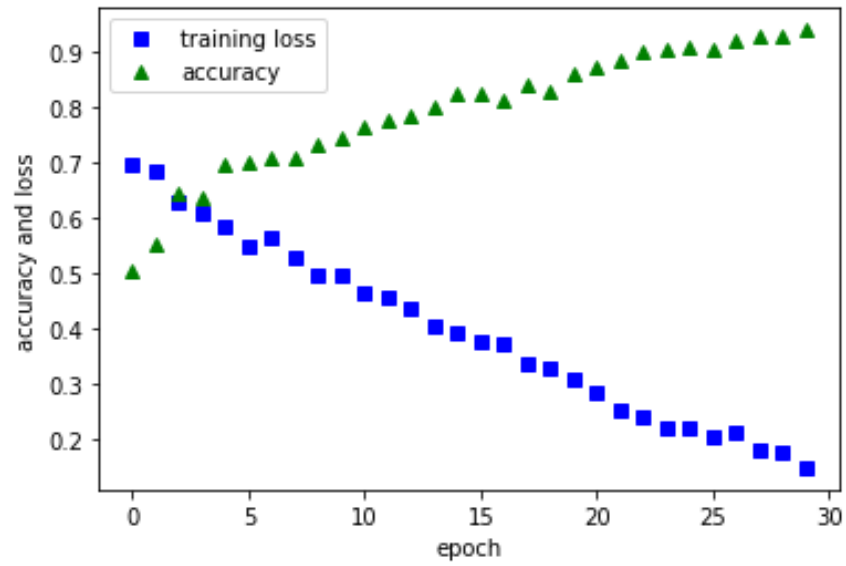


Figure 5-1 The rates of training loss and accuracy have been achieved during the training time of CNN.

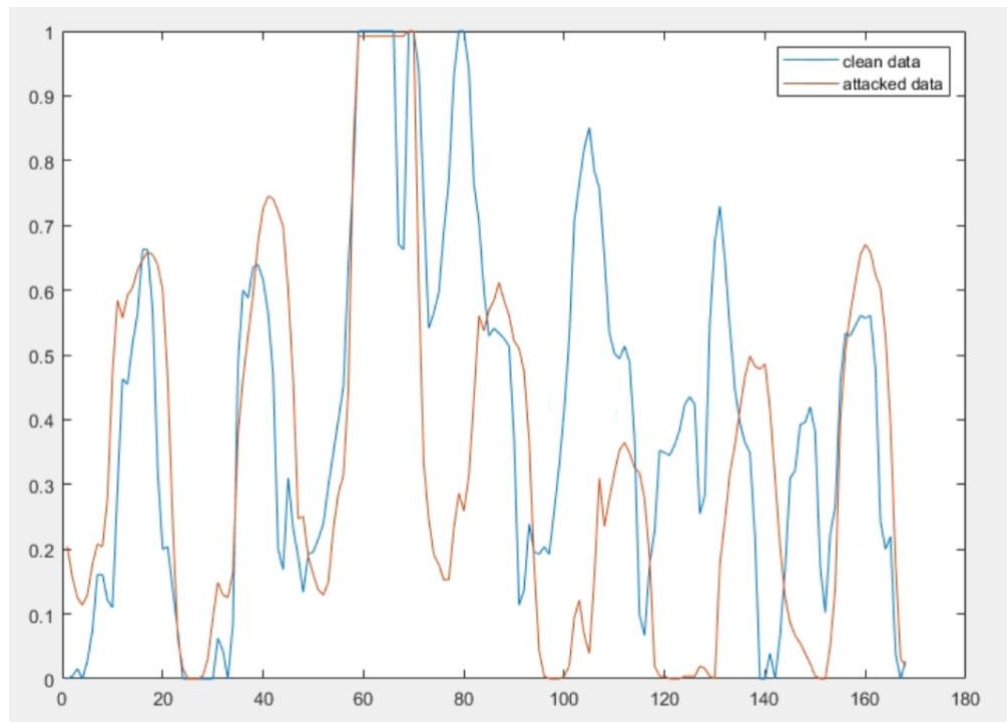


Figure 5-2 The difference between attacked and clean data.

manipulate the result of the classification.

First, the results were about the similar for PGD and FGSM, but as ϵ is increased, the accuracy dropped much more quickly for PGD than for FGSM. This also shows that PGD is a more powerful attack than FGSM as claimed. BIM also which is Iterative-FGSM also affects accuracy more than the FGSM as expected. The results can be seen in the Table 5-1.

One Pixel change one of the pixels completely and the rate of the attack success is expected to be higher than the others. For Sparse Fool attack, the low mean curvature of the decision boundary is taken advantage of which means that this attack is based on geometry and controls how sparse the perturbations are. Another attack, Deep Fool, produces adversarial samples with fewer perturbations and greater rates of misclassification than other approaches. Because one of the untargeted attacks is Deep Fool, which is meaning of not pushing the input towards a particular class.

Since the authors motivate to generate an image with optimum and less detectable perturbations, the result of Deep Fool also supports the claim.

Using TorchAttacks

Since TorchAttacks has been used for the experiment, it is important to underly some advantages and disadvantages:

- Lower-level operations such as multiplications for convolutions are not required to be implemented. Because of this, we were able to work on more advanced implementations like creating adversarial samples.

Table 5-1 The accuracy of classification and the rate of success of the attacks on Convolutional Neural Networks.

Attack Type	Standard Accuracy	Adversarial Accuracy
PGD Attack	93.41%	68.96%
FGSM Attack	90.50%	71.98%
Deep Fool	94.37%	32.42%
One Pixel	92.03%	24.59%
Sparse Fool	97.94%	64.71%
BIM	94.64%	71.21%

- Most operations work well on a wide range of platforms and devices. It would demand a considerable processing time in order to optimize the model for the usage of memory and processor, if the model was built in C++. In this situation, it would take a lot of time to compare different methods and models.
- It is complicated to alter existing operations and create one in TorchAttacks due to the lack of documentation.
- While implementing some attacks, we always face some errors. It is because their parameters are different and had to be changed constantly during the training time.

CHAPTER SIX CONCLUSION

With the convolutional neural networks model developed in this study, six adversarial attacks were applied to the model and data set and the results were presented. The creation of a deep learning model and the realization of adversarial attacks fall within the scope of white-box attacks. The white-box PGD attack is the one that does the most damage, which makes sense. It was made to be the ultimate first-order adversary, and therefore the strongest white-box attack that has been published thus far. The white-box FGSM attack comes in second, which is also expected. The white-box attack will always be more powerful than others because it has access to the model's loss function and can adjust the input to maximize the model's loss. All attacks also succeed in confusing the model and causing it to misclassify, which also shows the vulnerability of Convolutional Neural Networks. The existence of adversarial examples shows a vulnerability in the viewpoint of convolutional neural networks. Modern CNNs can be tricked into making bad decisions by making small changes to the inputs.

A successful offense necessitates an equally effective defense. For instance, whenever an FGSM attack was implemented, the authors were inevitably testing their own defense against an FGSM attack. When a defense is produced against an attack, another attack that is a powerful attack emerges in order to break it. After that, a more powerful defense is designed to mitigate the attack, and an even stronger attack suddenly arises to break through this defense.

Since the FGSM, a great deal of progress has been made in the area of defenses and adversarial attacks, and it would be beyond the scope of this thesis to discuss all these developments. In the literature, there are some defense methods such as adversarial

training, distillation, feature squeezing, ensemble methods, etc. Some of them can be applied to the results of this thesis in the future. In addition to this, new mitigation strategies and detection algorithms could be also studied for these attacks in the future.

LIST OF REFERENCES

- [1] M. Musser and A. Garriott, “Machine Learning and Cybersecurity AUTHORS HYPE AND REALITY,” 2021.
- [2] Institute of Electrical and Electronics Engineers and Manav Rachna International Institute of Research and Studies, Proceedings of the International Conference on Machine Learning, Big Data, Cloud and Parallel Computing : trends, prespectives and prospects : COMITCON-2019 : 14th-16th February, 2019.
- [3] A. , Chakraborty, M. , Alam, V. , Dey, D. , Mukhopadhyay, and A. Chattopadhyay, “Adversarial Attacks and Defences: A Survey,” ArXiv, abs/1810.00069., 2018.
- [4] A. Graese, A. Rozsa, and T. E. Boulton, “Assessing threat of adversarial examples on deep neural networks,” in Proceedings - 2016 15th IEEE International Conference on Machine Learning and Applications, ICMLA 2016, Jan. 2017, pp. 69–74. doi: 10.1109/ICMLA.2016.44.
- [5] K. Eykholt et al., “Robust Physical-World Attacks on Deep Learning Visual Classification,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Dec. 2018, pp. 1625–1634. doi: 10.1109/CVPR.2018.00175.
- [6] M. Ozdag, “Adversarial attacks and defenses against deep neural networks: A survey,” in Procedia Computer Science, 2018, vol. 140, pp. 152–161. doi: 10.1016/j.procs.2018.10.315.
- [7] S. Liang, X. Wei, S. Yao, and X. Cao, “Efficient Adversarial Attacks for Visual Object Tracking,” Aug. 2020, [Online]. Available: <http://arxiv.org/abs/2008.00217>

- [8] “The Electric Reliability Council of Texas,(2015 to 2021), The Hourly Load Data, Retrieved from: Load_Historical”.
- [9] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and Harnessing Adversarial Examples,” Dec. 2014, [Online]. Available: <http://arxiv.org/abs/1412.6572>
- [10] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” Jul. 2016, [Online]. Available: <http://arxiv.org/abs/1607.02533>
- [11] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards Deep Learning Models Resistant to Adversarial Attacks,” Jun. 2017, [Online]. Available: <http://arxiv.org/abs/1706.06083>
- [12] S. M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Dec. 2016, vol. 2016-December, pp. 2574–2582. doi: 10.1109/CVPR.2016.282.
- [13] J. Su, D. V. Vargas, and K. Sakurai, “One Pixel Attack for Fooling Deep Neural Networks,” IEEE Transactions on Evolutionary Computation, vol. 23, no. 5, pp. 828–841, Oct. 2019, doi: 10.1109/TEVC.2019.2890858.
- [14] A. Modas, S.-M. Moosavi-Dezfooli, and P. Frossard, “SparseFool: a few pixels make a big difference,” Nov. 2018, [Online]. Available: <http://arxiv.org/abs/1811.02248>
- [15] C. Szegedy et al., “Intriguing properties of neural networks,” Dec. 2013, [Online]. Available: <http://arxiv.org/abs/1312.6199>

- [16] N. Akhtar and A. Mian, “Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey,” *IEEE Access*, vol. 6. Institute of Electrical and Electronics Engineers Inc., pp. 14410–14430, Feb. 16, 2018. doi: 10.1109/ACCESS.2018.2807385.
- [17] T.-J. Chang, Y. He, and P. Li, “Efficient Two-Step Adversarial Defense for Deep Neural Networks,” Oct. 2018, [Online]. Available: <http://arxiv.org/abs/1810.03739>
- [18] T. Blumensath and M. E. Davies, “Iterative Thresholding for Sparse Approximations.”
- [19] M. Nikolova, “Description of the minimizers of least squares regularized with l_0 -norm. Uniqueness of the global minimizer,” *SIAM Journal on Imaging Sciences*, vol. 6, no. 2, pp. 904–937, 2013, doi: 10.1137/11085476X.
- [20] A. Patrascu and I. Necoara, “Iteration complexity analysis of random coordinate descent methods for ℓ_0 regularized convex problems,” Mar. 2014, [Online]. Available: <http://arxiv.org/abs/1403.6622>
- [21] M. Nagahara, D. E. Quevedo, and J. Ostergaard, “Sparse packetized predictive control for networked control over erasure channels,” *IEEE Transactions on Automatic Control*, vol. 59, no. 7, pp. 1899–1905, Jul. 2014, doi: 10.1109/TAC.2013.2294622.
- [22] E. J. Candes and T. Tao, “Decoding by linear programming,” *IEEE Transactions on Information Theory*, vol. 51, no. 12, pp. 4203–4215, Dec. 2005, doi: 10.1109/TIT.2005.858979.

- [23] D. L. Donoho, “Compressed sensing,” IEEE Transactions on Information Theory, vol. 52, no. 4, pp. 1289–1306, Apr. 2006, doi: 10.1109/TIT.2006.871582.
- [24] T. M. (Tom M. Mitchell, Machine Learning.
- [25] M. Radice, “Basic Structure of the Human Nervous System,” Available: <https://mikerbio.weebly.com/structure--function.html>, 2019.
- [26] M. Nielsen, “Neural Networks and Deep Learning.” [Online]. Available: <http://neuralnetworksanddeeplearning.com>
- [27] A. F. Agarap, “Deep Learning using Rectified Linear Units (ReLU),” Mar. 2018, [Online]. Available: <http://arxiv.org/abs/1803.08375>
- [28] I. Goodfellow, Y. Bengio, and A. Courville, “Deep Learning,” MIT Press, 2016.
- [29] S. , & A. A. Sharma, “Activation functions in neural networks. towards data science,” 2017.
- [30] X. Glorot, A. Bordes, and Y. Bengio, “Deep Sparse Rectifier Neural Networks,” 2011.
- [31] D. Liu, “A Practical Guide to ReLU,” Available: <https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7>, Dec. 22, 2017.
- [32] Y. LeCun, “Gradient-based learning applied to document recognition,” IEEE.
- [33] D. H. Hubel and T. N. Wiesel, “RECEPTIVE FIELDS AND FUNCTIONAL ARCHITECTURE OF MONKEY STRIATE CORTEX,” 1968.
- [34] R. C. Gonzalez and R. E. (Richard E. Woods, Digital image processing. Prentice Hall, 2008.

- [35] C. Saravanan, “Color image to grayscale image conversion,” in 2010 2nd International Conference on Computer Engineering and Applications, ICCEA 2010, 2010, vol. 2, pp. 196–199. doi: 10.1109/ICCEA.2010.192.
- [36] V. Solegaonkar, “Convolutional Neural Networks,” <https://towardsdatascience.com/convolutional-neural-networks-e5a6745b2810>, Apr. 15, 2019.
- [37] UFLDL Tutorial, “Feature Extraction Using Convolution,” Deep Learning Tutorial, 2022.
- [38] S. H. Wang et al., “Multiple sclerosis identification by 14-layer convolutional neural network with batch normalization, dropout, and stochastic pooling,” *Frontiers in Neuroscience*, vol. 12, no. NOV, Nov. 2018, doi: 10.3389/fnins.2018.00818.
- [39] A. Ruderman, N. C. Rabinowitz, A. S. Morcos, and D. Zoran, “Pooling is neither necessary nor sufficient for appropriate deformation stability in CNNs,” Apr. 2018, [Online]. Available: <http://arxiv.org/abs/1804.04438>
- [40] Z. R. Ramsundar B., *TensorFlow for Deep Learning* . 2018.
- [41] P. Golik, P. Doetsch, and H. Ney, “Cross-Entropy vs. Squared Error Training: a Theoretical and Experimental Comparison.”
- [42] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” Dec. 2014, [Online]. Available: <http://arxiv.org/abs/1412.6980>
- [43] T. Tieleman and G. Hinton, “Divide the Gradient by a Running Average of Its Recent Magnitude,” *COURSERA: Neural Networks for Machine Learning*, 2012.

- [44] Geoffrey Hinton with Nitish Srivastava Kevin Swersky, “Neural Networks for Machine Learning ,” UToronto and Coursera , 2012.
- [45] H. Kim, “TorchAttacks.” 2020.
- [46] Inc. Facebook, “PyTorch.” 2018.

VITA

Burcum Eken was born in Ankara, TURKIYE in 1995. She earned a Bachelor of Science degree at the Gazi University, Ankara with a major in Computer Engineering in 2017 and continued studying Computer Engineering as a graduate student at University of Tennessee, Knoxville.