A REVIEW ON ADVERSARIAL ATTACKS AGAINST NEURAL NETWORKS AND THEIR DEFENSE METHODS

A THESIS SUBMITTED TO
THE FACULTY OF ARCHITECTURE AND ENGINEERING
OF
EPOKA UNIVERSITY

BY

SAJDI MUDA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JULY, 2021

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name Surname: Sajdi Muda

Signature:

ABSTRACT

A REVIEW ON ADVERSARIAL ATTACKS AGAINST NEURAL NETWORKS AND THEIR DEFENSE METHODS

Muda, Sajdi

M.Sc., Department of Computer Engineering

Supervisor: Dr. Arban Uka

Co-Supervisor: Dr. Ali Osman Topal

Machine Learning (ML) is nowadays the core of technology and the main study area for different situations. It is indeed common to be the main subject of attackers at this kind of position. For instance, we can take as example the ML component of the last generation cars which have the autopilot functionality. Some would even think of damaging the ML component of the car system and try to perturbate the model to make the vehicle crash. In this case would be a real disaster, which of course a task at this level is being handled by car manufacturers. This situation is called usually as Adversarial Attack on Machine Learning Model.

The necessity of protection against adversarial attacks is indeed crucial and thus we do have a variety of methods handling this. As attacks are so different to a ML model, defense methods are according to them. In this thesis study we will show main attack methods and their corresponding defenses. At the end we will show and compare the abovementioned methods effectiveness.

Keywords: Machine Learning, Adversarial Attack, Adversarial defense, Deep Neural Networks

ABSTRAKT

RISHIKIM MBI SULMET NË RRJETAT NEURALE DHE MENYRAT E MBROJTJES NDAJ TYRE

Muda, Sajdi
Master Shkencor, Departamenti i Inxhinierisë Kompjuterike
Udhëheqësi: Dr. Arban Uka
Udhëheqësi i përbashkët: Dr. Ali Osman Topal

Mësimi automatik “Machine Learning” është tanimë bërthama e teknologjisë dhe një nga tematikat më kryesore të studimeve shkencore. Është padyshim normale që në një pozitë të tillë të bëhesh një pikë tërheqje për dashakeqësit dhe sulmuesit. Mund të marrim si shembull këtu komponentin e “mësimit automatic” të makinave të gjeneratës së fundit të cilat kanë funksionalitetin autopilot të inkorporuar. Disa mund të shkojnë aq larg sat ë mendojnë edhe sulmimin e komponentit të “mësimit automatik” dhe në këtë mënyrë ta nxjerrin sistemin e drejtimit jashtë kontrollit. Në këtë rast do të ishte një fatkeqësi e vërtetë, por që padyshim një problematikë e këtij lloji është menduar nga prodhuesit e makinave. Kjo dukuri njihet ndryshe edhe si “Sulmet kundërshtare” (Adversarial Attack).

Nevoja për mbrojtje ndaj sulmeve të tilla është padyshim kritike dhe rrjedhimisht ne kemi një varietet të madh metodave që merren me këtë dukuri. Në këtë tezë ne do të tregojme llojet kryesore të sulmeve dhe respektivisht mënyrat e mbrojtjes ndaj tyre. Gjithashtu ne do të bëjmë dhe një krahasim të performancave të këtyre metodave të ndryshme.

Fjalë Kyçe: Mësimi automatik, Sulme kundërshtare të mësimit automatik, Mbrojtje ndaj sulmeve, Rrjete neurale.

ACKNOWLEDGEMENTS

I would like to thank both my supervisor and co-supervisor, Dr. Arban Uka and Dr. Ali Osman Topal for their continuous guidance, encouragement and support during all the stages of my thesis. They helped me a lot to achieve the best throughout these years and for this thesis.

I am also deeply thankful to all the professors of Epoka University, who each of them was a good guidance to this journey.

I am especially grateful and thankful to my family for their continuous support and motivation. I am here because of them!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ML	Machine Learning
DNN	Deep Neural Network
CNN	Convolutional Neural Network
FGSM	Fast Gradient Sign Method
BIM	Basic Iterative Method
NLP	Natural Language Processing
GAN	Generative Adversarial Networks
HP	High Pass filter
LP	Low Pass filter
PCA	Principal Component Analysis
TCM	Target Class Method
AVS	Automated Vehicle System
PCA	Principal Component Analysis
API	Application Programming Interface
UPN	Unsupervised Pretrained Network
RNN	Recurring Neural Network
FC	Fully Connected layer

CHAPTER 1

INTRODUCTION

Although computers and machines are not human beings, Machine Learning approaches seem to make them vulnerable and being deceived by other systems or models. The usage of Machine Learning in security systems and vulnerable points of our everyday life, makes it much more important and thus much more attractive for adversaries. Systems such as autonomous cars, malware or other security detection platforms, face recognition along with biometric recognition, voice recognition [1] and much more are great to have, but at the same time we must ensure the security behind them. Lately the security aspects of Machine Learning input selection have been considered deeply by researchers.

Deep Learning is an evolutional branch of machine learning that makes possible to learn from the input data through many processing layers. Since the discovery of Deep Learning methods, it has been found to be the most powerful solution for large computational learning processes. The solution it provides is remarkable although the hardware architecture that needs to be built on top of must be enormous and efficient [2].

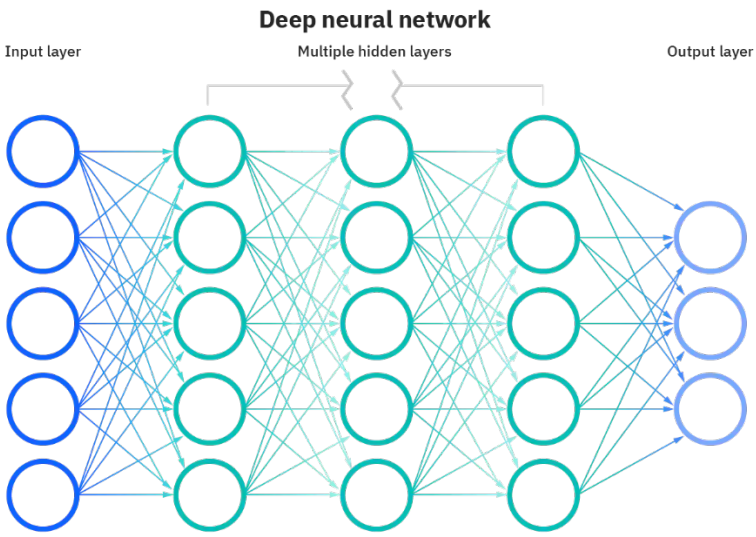


Figure 1. Deep Neural Network architecture[1]

Although Neural Networks and Deep Learning offer such nice learning curves and solid solution, they do fail in front of the adversarial attacks. The heart of Deep Learning Algorithms is built on top of interconnected layers and hidden layers of neurons which extract data from the given input.

There are different attacking methodologies that tend to ruin a ML model, but the most common, easy, and indistinguishable one is indeed feeding the model with adversary input modified at the finest level so that human eye could never catch the peculiarity as explained in the recent research of Chakraborty et. al. [1]. The taxonomy of these attacking methodologies will be expanded in the following chapters.

[1] Figure taken from https://www.ibm.com/cz-en/cloud/learn/neural-networks

1.1. Adversarial attack

Pitropakis et. al. have done a magnificent work by collecting information about the taxonomy of the adversarial attacks at [3]. As they explain in their recent work, a DNN adversarial can be either Black Box or White Box, and according to that the attack phase changes. In our work we will dive deeper to show how and when to prevent these kinds of attacks.

It is indeed important to note that DNNs weakest spot happens to be the image recognition task as Szegedy et. al. [4] proves. Additionally, Wang et. al. at their work [5] states that according to the research lately DNN is not robust at Natural Language Processing (NLP). There are some defenses mentioned in that work that are worth studying for this purpose. DNN does not suffer at all when it comes to voice recognition thus, DNN can be used without a doubt in this direction.

Shilin et. al. [6] categorizes all adversarial attacks according to their typology on the area of usage (ex. Computer Vision, NLP etc). The weakest spot of ML seems to be Computer Vision again, especially due to its large input features and computations.

Arguably, the worst disadvantage of DNN is their “black box” nature. In other words, you do not know how or why your DNN came up with a certain output. For instance, when you put an image of a dog into a DNN and it predicts it to be a cat, it is very hard to understand what caused it to arrive at this prediction. When you have features that are distinguishable by a human eye it is much easier to understand why the model failed in that specific input. By comparison, algorithms like Decision Trees are very interpretable. This is very important because in some domains, interpretability is critical to be able to improve the performance of the model. [7]

For our concern, Image recognition task must be considered to experiment for its robustness and further decide whether it can hold back towards adversarial inputs. Attack oriented experiments will be tested in our work for this purpose.

1.2. Structure of Thesis

The structure of this thesis will be composed of 5 chapters. In the first one we will present the adversarial problem, its threats, and the importance of researching in this direction. A short description of this work will be explained in this chapter.

In Chapter 2 we show all recent trends and research in this field. In this chapter we will also show and describe the taxonomy of adversarial attacks. Further in the chapter the defense methods will be explained.

In Chapter 3 we explain our methodology and the approach we have regarding this problem. As this thesis will be mainly a survey rather than finding a new solution, we will explain how we will evaluate the methods described in the previous chapter. Dataset that has been used and DNN structure will be shown at the end.

Chapter 4 will be a showcase of our practical work and a comparison between results obtained through all methods explained in this work.

Our last Chapter is Chapter five where we conclude this our work and give some future recommendations prior to the results of the experiments. The challenges that we have faced in this work will be listed in this chapter for future improvements.

CHAPTER 2

LITERATURE REVIEW

Despite the recent advances in the field of machine learning, its applications and models, vulnerability of these models to adversarial attacks is a matter of great importance during the last years. Since the process of inputs to a model is becoming more and more dynamic in the terms of collection, it is indeed a crucial task ensuring the security and the veracity of the data. Data collection or transmission targeting can be examples of the adversarial methods to affect a machine learning model, which lead to manipulating the input or the model, as shown in Figure 2. Models are threatened by adversarial attacks and can be protected by the adversarial defense methods.

In this chapter, we will show a review of the previous research work about the taxonomy of the adversarial attacks. Then, we will focus on the classification and the most common types of the attacks and defense methods.

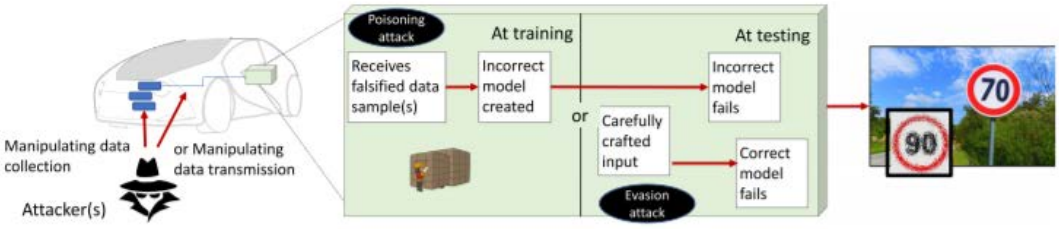


Figure 2. Different avenues for attacking a system.[2]

Pitropakis et al in their work at [3], propose a taxonomy, which classifies many recent studies and research about adversarial attacks against the machine learning models. It provides also the breaking down process of the features used to classify

[2] Figure taken from [3], page 4, figure 1

different attacks by introducing several phases that each of the attack features are uniquely associated with. Authors state the fact that ML is supposed to be trained using datasets that are representative and trustworthy to obtain the results we are looking for. The matter here is that outsiders and other actors would want to break the trustworthiness of the dataset. No matter what the cost is, we should offer a robust technique that either detects these changes or will not be fooled from a bunch of modified micro parts of the inputs.

In another survey of Zhou et al. [8] it was presented the approaches that describe the behavior of adversarial machine learning as game theoretic. The adversary is the attacker and the player in this case is the learning system.

The intention of attacking a model sometimes may not be just perturbating to a desired point, rather learn as much as possible about the architecture of that model. Duddu in his work, [9] states that the adversary's aim is to gather the sensitive information of the system architecture to be able to steal data afterwards.

In another research on adversarial attacks [10], authors show that Deep Neural Networks (DNN) are very sensitive to changes in the input, meaning that accuracy of the model's prediction changes if the input image changes. To prove this, the authors added noise to the input to fool the model, and it resulted to the weakness of the network.

2.1. Adversarial Threat model and Taxonomy

In this section we represent with the overall taxonomy of the adversarial example, and how should one build the ML model to overcome these issues. We will include also some real-life scenarios to be able to fully understand the case. The adversary may try to use not only one, but many week spots or vulnerabilities. A full protection must consider

protecting all possible vulnerabilities which will be listed below in the forthcoming subtitles.

2.1.1. Adversarial attack surface

Chakraborty describes the machine learning system as a data processing pipeline [11]. Thus, we should define the attack surface based on this processing pipeline where the adversary can attack. The sequence of operations can be described as:

a) Collection of input data from sensors or data repositories

b) Transferring the data in the digital domain

c) The transformed data processed by the ML model; output produced.

d) Action based on the output.

Below in figure 3, the generic pipeline of an Automated Vehicle System (AVS) is presented visually.

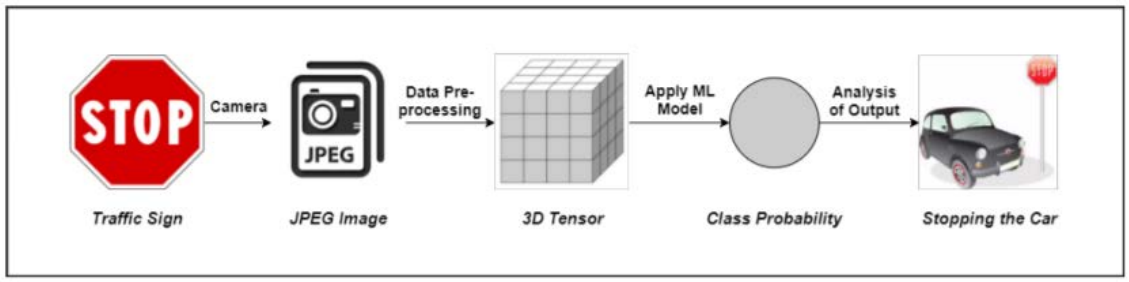


Figure 3. Generic pipeline of an Automated Vehicle System[3]

The attack surface can be identified based on the data processing pipeline. To ruin the ML model, adversary has two main entry points to the processing pipelines, and

[3] Figure taken from [1], page 28, figure 3

they are: a) Data collection and b) Data processing where each of them have their own challenges and opportunities.

An adversary's intention or goal may be divided generally into 4 categories as follows:

1. ***Confidence Reduction:*** The adversary tries to reduce the confidence of prediction for the target model. This is generally achieved by lowering the confidence level for each possible class and misclassifying all of them no matter what the output is. For instance, the sign "STOP" would be predicted with 60% confidence rather than the pure model that was with 90% confidence. The intention behind this attack is simply to make a model useless. In many cases this kind of attack can be used by the model creator itself to test the model and reach the perfection (if possible)
2. ***Misclassification:*** The adversary tries to alter the output classification of an input example to any class different from the original class. This method is quite similar with the first one, but with the only difference that here the intention is not just to lower confidence, but to predict another class. For instance, the sign "STOP" would predict another sign such as "45 speed limit".
3. ***Targeted Misclassification:*** The attacker tries to evade the model on predicting a class no matter what the input is. For instance, all traffic signs should predict the sing "90 speed limit".
4. ***Source/Target Misclassification:*** The adversary attempts to map the real outputs with another class to mislead the model. The difference with the targeted misclassification is that in this case the adversary wants to map all classes to predict each of them some other class. For instance, sing "STOP" will predict "45 speed limit" and sign "turn left" would predict "turn right".

2.1.2. Threat model

The intention of an adversary may vary from task to task, but in overall we can group them as below:

- Evasion attack
- Poisoning attack
- Exploratory attack

Evasion Attack: This kind of attack occurs during the testing phase and the adversary has no intention at all ruining the ML model, but rather he tries to fail the model during action. A real-life example to this problem may be as follows:

A model for AVS whenever gets a "STOP" sign as input must stop the vehicle and obey the rules. Instead, an adversary knowing somehow the architecture of the model evades this normal flow by changing the stop sign image at some pixels (Described in the figure 4)

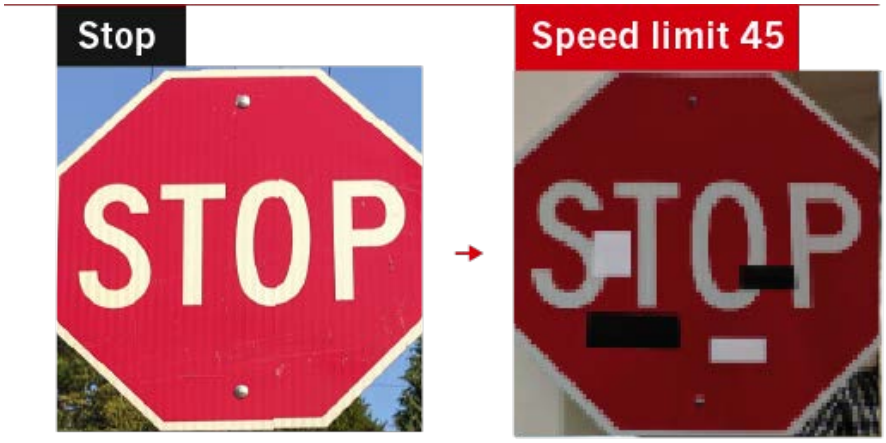


Figure 4. Stickers in the second image made ruined the ML Model.[4]

[4] Figure taken from https://bdtechtalks.com/2018/12/27/deep-learning-adversarial-attacks-ai-malware/ai-adversarial-attack-stop-sign/

Poisoning attack: Is the second type of thread model and is indeed a very difficult one due to the poisoning happening while training the model, thus many models can be fooled or mistrained. If a mistrained model remains unseen and that specific one runs for testing case or in production, then a disaster may occur. The data collected for the task of training the model, should be "poisoned" (some additional noise in the data that misleads the model).

This kind of model will be full of error regardless the input it will have, and besides it wont show at all the accuracy it was prepared for. At figure 5 you can see the schema how poisoning attack occurs in the normal learning process.

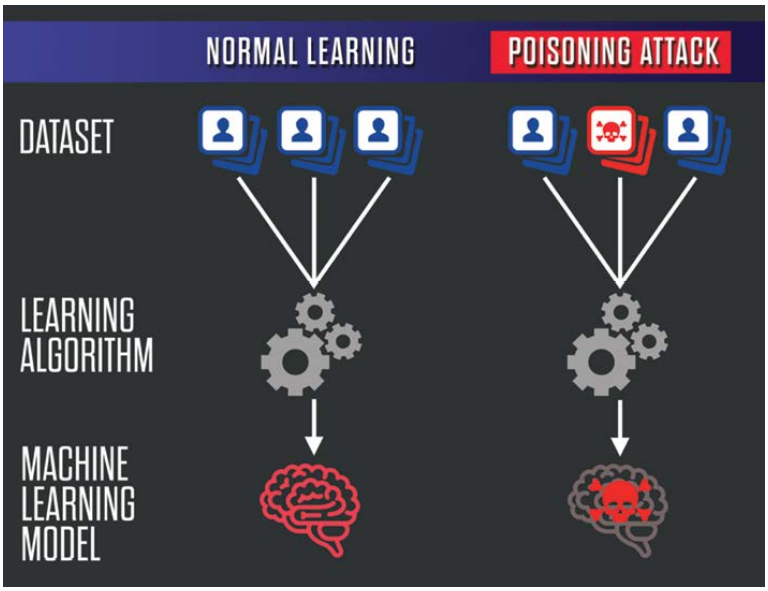


Figure 5. Poisoning attack schema[5]

Exploratory attack: This kind of attack is a bit more visionary than the previous ones. The real aim of this type of adversary is to not harm the model at all, instead to learn as much as possible regarding the learning algorithm of the underlying system and pattern in training data.

[5] Figure taken from https://analyticsindiamag.com/what-is-poisoning-attack-why-it-deserves-immediate-attention/

Table 1. Attack Summary

Adversarial model	*Type of attack*
Exploratory Attack	Model Inversion Membership Inference attack Model Extraction via APIs Information Inference
Evasion Attack	Adversarial Examples Generation Generative Adversarial Networks (GAN) GAN based attack in collaborative learning Intrusion Detection Systems Adversarial Classification
Poisoning Attack	Support Vector Machine Poisoning Poisoning on collaborative filtering systems Anomaly Detection Systems

In Table 1 are listed some type of attacks categorized into 3 attack models explained above. This classification is done by Chakraborty et. al. [11] after an extensive research regarding the taxonomy of the adversarial methods on ML.

The information that the adversary has regarding the system, or the algorithm defines the thread model too. Below we will list the capabilities that one can possess in different stages.

2.1.3. Adversarial capabilities

The capability term refers to the amount of information or space for action that is available to the attacker. The aforementioned attack types are strongly connected with the capabilities. An adversary that has access to the model (poisoning attack) is automatically kind of stronger compared to the adversary that has access only to the testing phase (evasion attack). The capabilities of an adversary are commonly separated into two main groups: [11]

- Training phase capabilities
- Testing phase capabilities

Training phase capabilities: Attacks during the training phase are stronger and tend to influence or corrupt the whole model. This is usually achieved by altering the model based on the capabilities listed as follows: [3]

1. *Data Injection:* The attacker does not have any access to the dataset which the model is training on, neither have access to the algorithm, but may have access to inject some new corrupted data (maybe created solely for this task) and in this way the model will be corrupted.
2. *Data Modification:* The attacker has access only to the dataset and in this way he can modify current inputs by editing them physically and the whole model will perturbate at the desired level of perturbation.
3. *Logic Corruption*: The attacker do not deal with the data or data preprocessing. He directly tries to mess the algorithm on the run so that even though data is correct and not corrupted, system is tended to fail too.

Testing phase capabilities: Adversarial attacks at the testing time do not tamper with the targeted model but rather forces it to produce incorrect outputs. The effectiveness of such attacks is determined mainly by the amount of information available to the adversary about the model. [11] This capability can be split in two main groups, Black Box and White Box.

1. *White Box:* The adversary has full knowledge regarding the model used for learning and its architecture. Attacker has knowledge about the training dataset and the algorithm used for that task. Simply put the adversary is an insider who is too powerful, and the adversarial attack may be so hidden to be revealed, which is the real disaster.
2. *Black Box*: The adversary in this case does not have any knowledge regarding the algorithm, dataset, architecture of the model and the learning model itself. There are a ton of methods that these types of attackers do to approach the real model and compromise it, but the most important of them are 3 and listed as follows.

a. *Non-adaptive Black-Box attack*: Adversary in this case knows only the training set of the model and does not have any clue regarding the model and the algorithm. To discover the whole process, the attacker randomly uses a new algorithm and a new model architecture to train on the dataset that he already knows. Iterative he changes the model parameters or the algorithm to approach as much as possible to the real model. At the end, after discovering the model the attacker uses normal White-Box techniques to make adversary samples.

b. *Adaptive Black-Box attack*: This type of attacker does not have any clue regarding the training process, but he can access the target model as an oracle. Then the "oracle" queries the target model to generate a carefully selected dataset which then will tell more about the model and the training process itself. At the end, the White-Box techniques are followed.

c. *Strict Black-Box attack*: At this point, the adversary does not know anything, but just by following the input output pairs of the model may create an idea of the model, just like the adaptive attack idea. This kind of attack may be useful only for large set of input-output pairs, which means a lot of time consuming over the model.

All the above information regarding the evasion/poisoning adversarial attacks and their respective methodology (white/black box) are explained graphically regarding the complexity and capability.

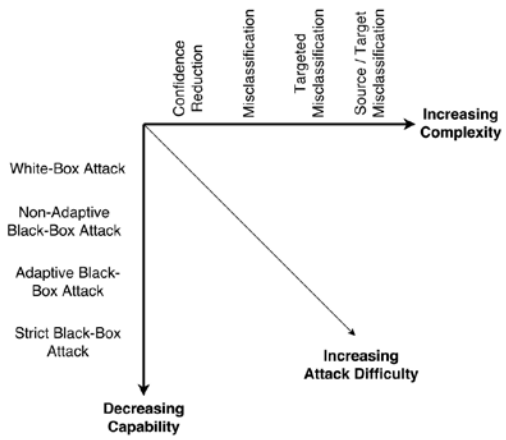


Figure 6. Evasion attack difficulty according to adversarial capabilities[6]

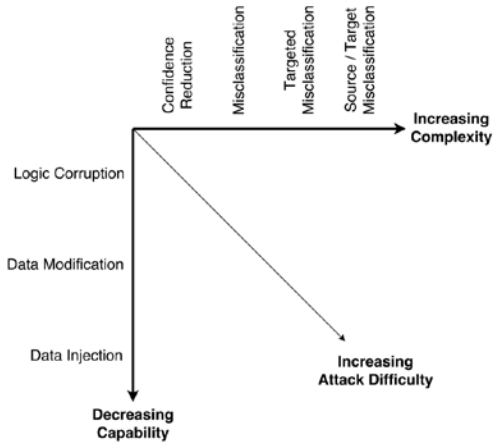


Figure 7. Poisoning attack difficulty according to adversarial capabilities[7]

[6] Figure taken from [1], page 30, figure 5 a)

[7] Figure taken from [1], page 30, figure 5 b)

2.2. Adversarial attacks

In Figure 8 there is a nice representation of the Adversarial examples which categorizes the types of attacks and defenses.

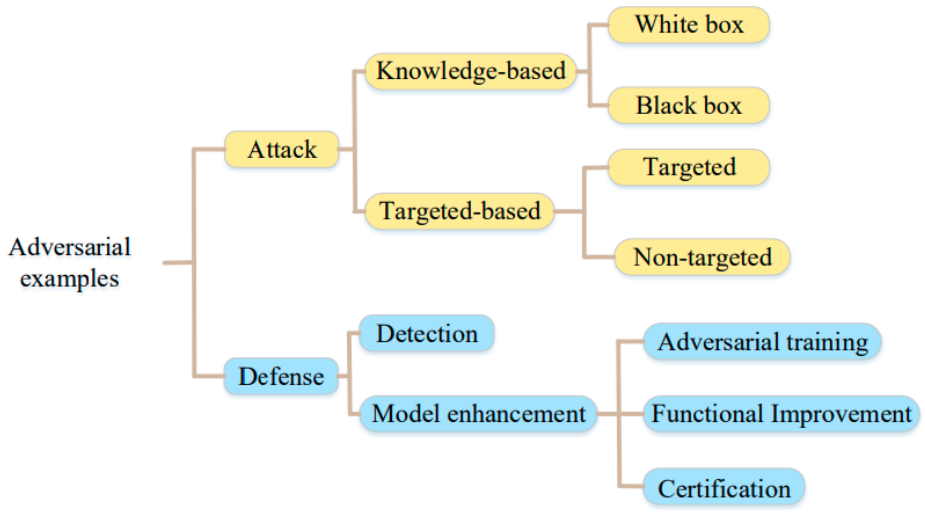


Figure 8. Classification of adversarial attacks and defenses [8]

Mainly the attacks are done by changing the inputs, for example such as adding noise to images, adding chars or words to texts etc, even a single pixel in some cases can ruin the whole ML model [12]. Most of them cannot be caught by human eye and need a very thorough expertise to discover.

2.2.1. Adversarial input (sample) generation

This type of attack occurs simply by manipulating the input for the model on test or while training (Evasion and Poison attack respectively). For this task there are a ton of algorithms that precisely handle this issue. We will list a few of them which are the

[8] Figure taken from [5], page 4, figure 3

most important among the whole list determined by the latest research and surveys [13] [14] [15] [16] [17] [18].

Fast Gradient Sign Method (FGSM): An efficient solution is introduced by Goodfellow et. al. [10].

They proposed a fast gradient sign methodology which calculates the gradient of the cost function with respect to the input of the neural network. Simply put, for an input image, the method uses the gradients of the loss with respect to the input image to create a new image that maximizes the loss. This new image is called the adversarial sample/image. The adversarial examples are generated using the following equation:

$$X* = X + \epsilon * sign(\nabla x\ J(X,yt))$$

Here, J is the cost function of the trained model, ∇x denotes the gradient of the model with respect to a normal sample X with correct label yt, and ϵ denotes the input variation parameter which controls the perturbation's amplitude. [11]

An intriguing property here, is the fact that the gradients are taken with respect to the input image. This is done because the objective is to create an image that maximizes the loss. A method to accomplish this is to find how much each pixel in the image contributes to the loss value and add a perturbation accordingly. This works pretty fast because it is easy to find how each input pixel contributes to the loss by using the chain rule and finding the required gradients. Hence, the gradients are taken with respect to the image. In addition, since the model is no longer being trained (thus the gradient is not taken with respect to the trainable variables, i.e., the model parameters), and so the model parameters remain constant. The only goal is to fool an already trained model.

Since then, the FGSM has been improved and many variations of it has been rolled out from different research. The most well-known variations of FGSM are as follows:

- *Basic Iterative Method (BIM):* This is the most straightforward FGSM variation yet one of the best. This method generates adversarial examples iteratively using a small step size. [19]

$$X^{0*} = X;\ X^{n+1*} = Clip_{X,e}\{X^{n*} + \alpha * sign(\nabla x\, J(X^{n*}, y_{true}))\}$$

- *Target Class Method (TCM):* This method maximizes the probability of some specific target Y_{target} which in the most cases is not the real class. The equation is not so different than the real FGSM approach:

$$X* = X + \epsilon * sign(\nabla x\, J(X, Y_{target}))$$

2.2.2. Generative Adversarial Attack (GAN)

Goodfellow et al [10] introduced generative adversarial networks whose aim is to generate samples like the training set, having almost identical distribution. The GAN is composed of two deep learning networks which are as following: A discriminative deep learning network and a generative deep learning network. The role of discriminative network is to distinguish between samples taken from the original database and those generated by GAN. The generative network is first initialized with random noise. Its role is to produce samples identical to the training set of the discriminator. Formally, generative network is trained to maximize the probability of discriminative network making a mistake. This competition leads both models to improve their corresponding accuracy at highest levels possible. We can denote model as D for discriminative network and G for generative network. The entities and adversaries are in a constant duel where one (generator) tries to fool the other(discriminator), while the other tries to prevent being fooled. The authors defined the value function V (G,D) as

$$\min G \max D\ V(D,G) = Ex{\sim}pdat\ a(x) \log D(x) + Ez{\sim}pz(z) \log(1 - D(G(z)))$$

where $p_д(x)$ is the generator's distribution, pz (z) is a prior on input noise variables. The objective is to train D to maximize the probability of assigning correct

label to training and sample examples, while simultaneously training G to minimize it. It was found that optimizing D was computationally intensive and on finite data sets, there was a chance of over fitting. Moreover, during the initial stages of learning when G is poor, D can reject samples with relatively high confidence as it can distinguish them from training data. So, instead of training G to minimize log (1 − D(G(z)), they trained G to maximize log (D(G(z)).

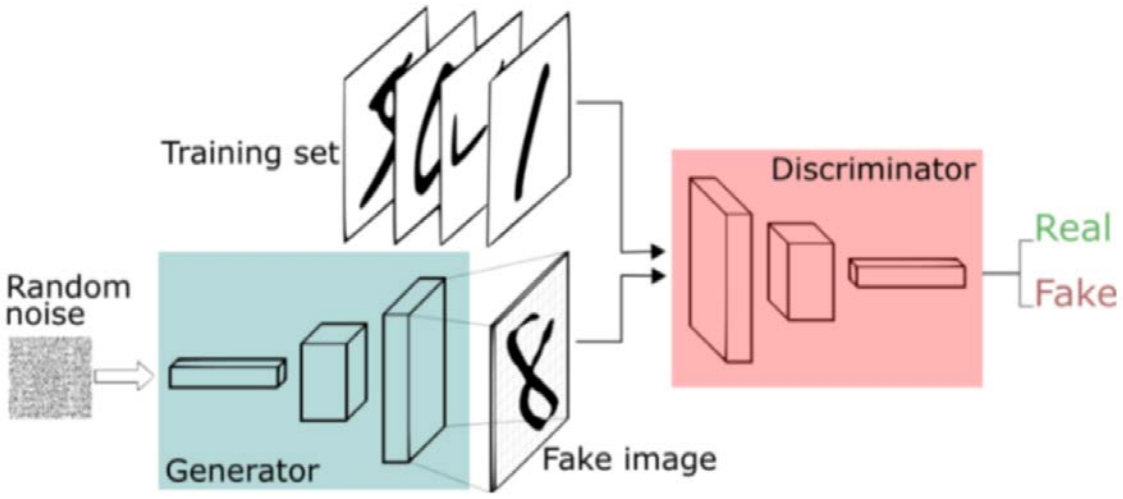


Figure 9. GAN attack schema[9]

2.2.3. GAN in collaborative Deep Learning

B. Hitaj et al [20] introduced a GAN method for collaborative learning framework. The motivation behind this technique is to gather as much information as possible from an honest victim inside the Collaborative Deep Learning process. It is a type of attack used solely in the collaborative deep learning framework where each of users contribute to the training process of the main model.

[9] Figure taken from https://towardsdatascience.com/a-game-theoretical-approach-for-adversarial-machine-learning-7523914819d5

The authors showed that the attack can be carried out in Convolutional Neural Networks which are themselves pretty difficult to invert or even when the parameters are hidden using differential privacy. In the white-box setting, the adversary acts as an insider within the privacy-preserving collaborative deep learning protocol. The motive of the adversary is to extract tangible information about the labels that do not belong to his dataset.

2.3. Adversarial defenses

Generalization of the solution regarding this high-level difficulty task is nearly impossible due to its variability in usage. Maybe we ca generalize a solution for Computer Vision in a specific direction, but the problem for NLP, sound processing etc is still completely another type. Until now there are some research that mention robust networks can be built to achieve impressive accuracy, even though there may be adversarial samples in the training set [16] [21]. In another research we find proved that no network is capable of being fully protected just by self-arrangement and without any kind of additional method [22].

Below in this chapter we will list some of most well-known and proved methods regarding the protection against adversarial attacks regarding the type of attack it is. According to the typology of the defense method they can be grouped mainly into:

- Adversarial training (Immunity to adversarial inputs)
- Denoise functions

2.3.1. Adversarial training

This methodology regarding the defense against adversaries results very handy and with quite an improvement in performance [23] [4] [24]. The principle is to poison the model by we and make it immune regarding adversarial samples. This method is likely a brute force algorithm type. The defender should create by his own all the

necessary adversarial samples. Although the results show that this method seems to be robust on first level attack, it fails on more than 1 level attacks such as the black box attack, where the adversarial tries more than one method to get into the model.

2.3.2. Denoise functions

There are some already robust methods that handle the task of denoising the noised images (or any kind of digital signal) successfully. The most straightforward among all *Principal Component Analysis* (PCA) which has shown to be effective in a recent work of Jere et. al. [25]. Authors mention in that study that PCA is an effective defense against certain adversarial attacks on smaller datasets. This works well when the dataset and number of features are small, however, for larger datasets with larger inputs this method becomes computationally inefficient as the size of the data matrix scales quadratically with the size of dataset.

Authors propose in that study a better solution to PCA. Their intention is to make a dimensionality reduction of an image so that for larger dataset and for images with many features, the defensive PCA would not cost so much in terms of computation. The results are quite good but not as impressive as required, simply put from an accuracy of 98% in a binary classifier, defensive PCA resulted in a 93% accuracy of the same dataset.

Another method used for denoising is the Low Pass (LP) and High Pass (HP) filter. The LP is often called as the “smoothing” filter. It lowers the signals of an image that are above a threshold. With this technique we may get rid of additional noises but the accuracy is not so promising. The HP in the other hand is the opposite. It sharpness the images to make the features more obvious. These two methods are very dataset specific and are not as robust as needed for a successful learning model.

The last but not least, is the *denoise autoencoder* method. Autoencoder is a type of network architecture composed of two identical but opposite networks which are respectively encoder and decoder networks. As shown in the figure 10 below, the first part of the autoencoder is the encoder network which reduces the features of an input

sample, lets take as a case an image. So, from 32x32 image (1024 pixel) the encoding process may reduce the features up to 100 or even further, 10 pixel. Then by doing the symmetric function of the encoding process, the decoder tries to reconstruct the sample from the output of the encoder. From the example mentioned the image encoded into 100 numbers will be reconstructed to 1024 pixels as it was previously. The reason behind doing this methodology is simple: By encode-decode we try to remove the ambiguous and extra features which in our case are adversarial features to the inputs (some extra pixels ore noises).

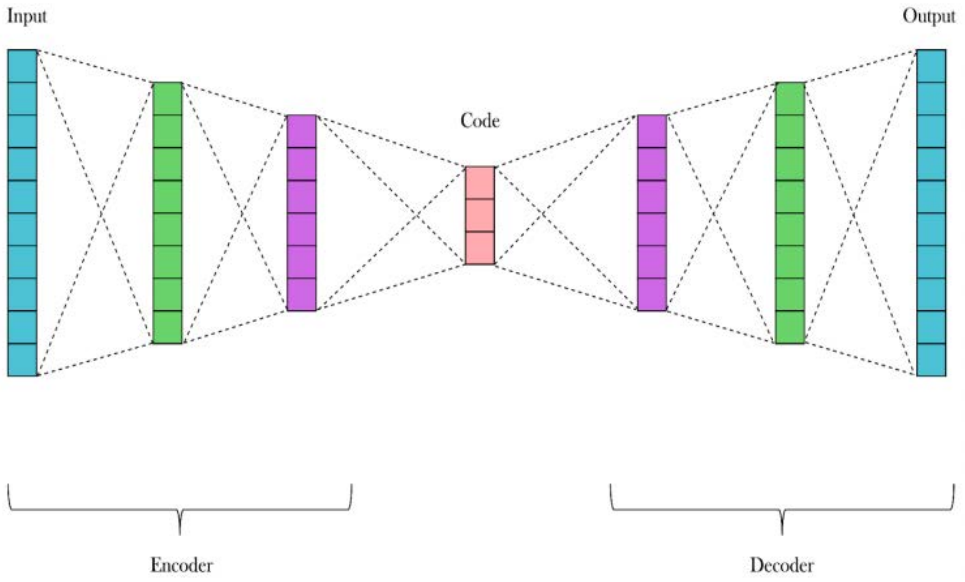


Figure 10. Autoencoder schema [10]

[10] Figure taken from https://medium.com/analytics-vidhya/creating-an-autoencoder-with-pytorch-a2b7e3851c2c

CHAPTER 3

METHODOLOGY

3.1. Methodology

We use both Qualitative and Quantitative research in this thesis. The thesis aim is to provide a survey on some types of adversarial attacks on neural networks and to deeply analyze the accuracy metrics under these attacks in comparison with the accuracy of the base model, meaning CNN model. Moreover, we will provide the respective adversarial defenses against these attacks. For this reason, here you will find implemented both types of research.

Qualitative Research

Qualitative Research approach requires a detailed observation and is concerned mainly with the process, rather than the outcomes. This type of research is descriptive, the researcher is interested in the process, meaning and understanding of the problem.

Taking into consideration this approach, we will apply a Qualitative Research design in this thesis to have a better representation of the facts and methods used in fulfillment of our thesis aim.

Quantitative Research

Quantitative research is said to be empirical in nature and is known as scientific paradigm. This approach uses experiments and numbers to ensure the validity of the problem. The researchers use quantitative information to establish generalizable facts about a topic. [26] Our approach of reporting the evaluation metrics in neural networks, making comparisons with model under adversarial attacks and reporting numeric conclusions is what we call Quantitative research in this work.

3.2. Dataset

In this work we will use CIFAR-10 dataset. The name stands for Canadian Institute For Advanced Research and it was developed together with CIFAR-100. The dataset is used for object recognition in computer vision field, and it was collected by CIFAR institute researchers, Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. [27] It consists of 60000 images, with size 32x32. The images are colored photographs of objects, smaller than a typical photograph.

The dataset is divided into 50000 images for training and 10000 for testing, each division is part of five training batches and one test batch. The train batches contain 5000 images from each class randomly selected, but some training batches contain more images that belong to a class than another. The test batch 1000 images from each class in random order. There are 10 classes, associated with integer values, as shown in the table below.

Table 2. Dataset tables

Label	***Class***
0	airplane
1	automobile
2	bird
3	cat
4	deer
5	dog
6	frog
7	horse
8	ship
9	truck

The classes are mutually exclusive. The figure 11 below shows the classes of the dataset and 10 random images from each class.

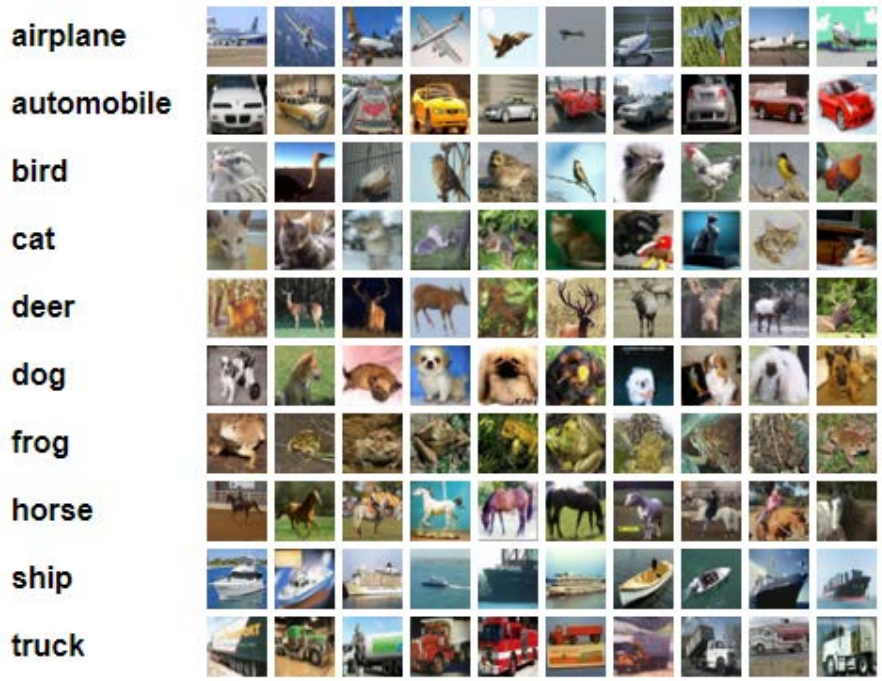


Figure 11. Dataset classes [11]

The archive contains the train and test batches, each of them is a Python "pickled" object, produced with cPickle. The python3 version which opens such a file and returns a dictionary is given below.

```
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

Listing 1. Unpickle dataset batch

In this work, the dataset is loaded using the Keras API, which will be later explained.

[11] Figure taken from https://www.cs.toronto.edu/~kriz/cifar.html

3.3. Libraries and Tools

The experiments in this work are conducted by using Python Scripting language, which is powerful for ML and it provides some beneficial packages for ML models. In this section, we will list and describe the most used Python packages and tools that we have used in our experiments for this thesis.

Google Colaboratory (Colab) – an interactive environment that allows to write and execute Python code. Execution takes place in Google's cloud servers, so that users can use Google hardware, no matter of the power of their machine. It is used extensively for ML community. In our case, we started to train the model in Colab, but we faced some issues, related with execution time. To train the model, it took more 10 hours and this was not possible in Colab, it stopped the execution after 12 hours. A solution was Colab Pro, which was not available in our country, Albania, so I decided to go on with Python IDE, PyCharm installed in our personal computer.

PyCharm – is a handy Python IDE for professional use. It can organize the project into directories and files and support all the python libraries. That is the IDE we have used for our experiments' codes.

Tensorflow – is and open-source platform for developing and training machine learning models and deep neural networks research.

Keras – is and API used for deep learning framework written in Python, which runs on top of the ML platform, Tensorflow. The core structures of Keras are layers and models.

NumPy – is a Python library used for working with arrays and stands for Numerical Python. By reading the image as a NumPy array, various image processing can be performed using NumPy functions.

Matplotlib – is the library used for image visualizations, plotting of graphs and any kind of interactive visualizations.

3.4. Deep Neural Networks

Deep Neural Networks (DNN) have become the methods of speech classification and image classification. They are neural networks with more than 2 layers or better described as networks with an input layer, an output layer and at least one hidden layer in between. The neurons are interconnected like neurons in the human brain.

The first layer neurons consume the input data and their output is used in the next layers to provide a final output. Each layer computes a small function, called activation function. This function decides what to be passed to the next neuron.

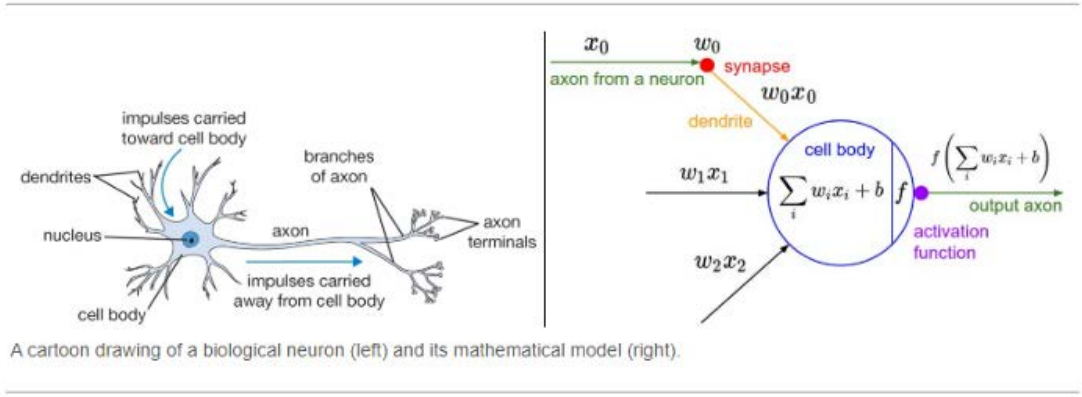


Figure 12. Activation function illustration [12]

The connection between neurons of successive layers has a weight, which defines the influence of the input to the output for the next neuron. The initial weights are

[12] Figure taken from https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253

selected randomly, then they are updated during iterations. A learning mechanism optimizer updates these wights that will result into correct prediction of the outcome.

There are three broad classes of deep neural network architectures, the convolutional neural networks (CNNs), unsupervised pretrained networks (UPNs), and recurrent neural networks (RNNs). Among these, the CNN are commonly used in image classification. That is what we are using with CIFAR-10 dataset, so it will be explained in much more details below.

3.4.1. Convolutional Neural Network

CNNs or ConvNets are similar to Neural Networks, made up pf layers with neurons and weights. They are composed of multiple layers of artificial neurons. The flow of data between layers is as in a DNN. Moving deeper in the layers of CNN, they detect higher-level features such as objects, faces, and more. Their applications vary from image and video recognition, image classification, medical image analysis, computer vision and natural language processing.

CNN architecture consists of two main parts:

- a convolutional tool, used for separating and identifying the image features to be used in the process (Feature Extraction)
- a fully connected layer used for predicting the image class from the output of the process and the features from previous stage.

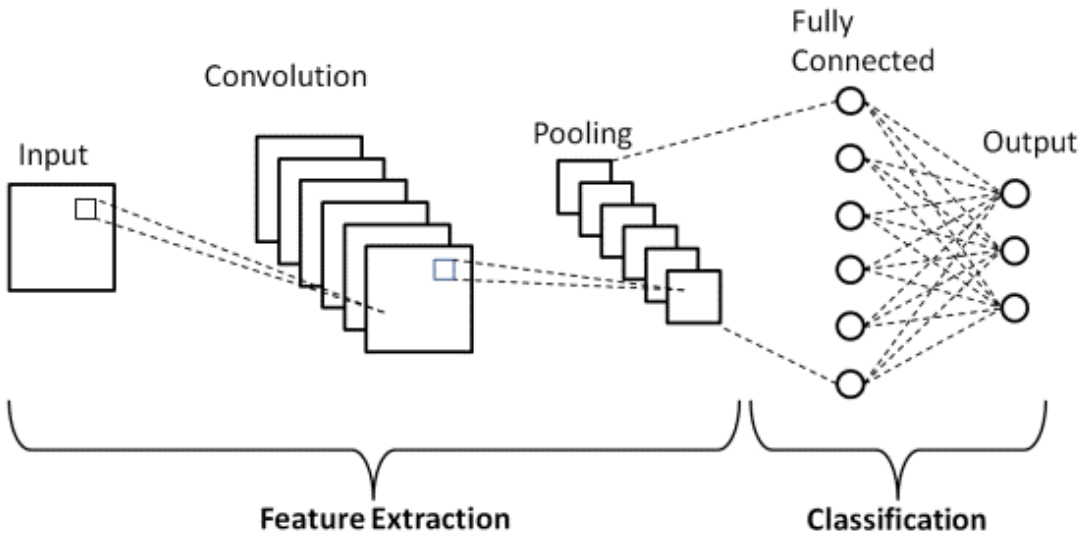


Figure 13. CNN Architecture [13]

The layers that make up CNN are three types, convolutional layers, pooling layers and fully connected layers (FC). The first layer used to extract features from the input images is the convolutional layer. The output of this layer is called Feature Map and serves to other layers to learn several other features. Convolutional layers are followed by Pooling layers. The purpose of such layer is to decrease the size of the convolved feature map to reduce the computational costs. While, in Max Pooling, from the feature map is taken the largest element. This type of layer is the middle of convolutional fully connected layer. The last type of CNN layers are FC, consisting of the weights and biases along the neurons and connects the neurons between two layers. They are the last layers of the CNN and placed before the output layer.

In addition to these, there is also Dropout layer and Activation functions. Dropout layer is used to overcome overfitting. Overfitting occurs when the model works so well on the training data, which leads to negative impact in the model's performance. The dropout layer drops some neurons from the neural network during the training phase,

[13] Figure taken from https://www.upgrad.com/blog/basic-cnn-architecture/

which reduces the size of the model. For example, a dropout of 0.2, means that 20% of the nodes will be dropped randomly from the network. Activation function decides which information of the model goes in forward direction and which one should not. The most used activation functions are ReLU, Softmax, tanH and Sigmoid functions.

3.5. Deep Neural Networks under adversarial attacks

Deep learning models are very vulnerable to adversarial examples, that are input images generated by perturbations and which fool the network. In this work, we will focus on gradient based attacks that create adversarial examples. The two attacks, Fast Gradient Sign Method (FGSM) and Basic Iterative Method (BIM) are the methods we will use to add perturbation in our dataset images to compare the results with the base model results (meaning no attacking). The goal is to ensure misclassification. We compare the result using the most common type metrics, accuracy.

To protect and increase model robustness towards these attacks, defenses are implemented. We will use adversarial training, as a standard brute force approach. Adversarial examples will be used to train the model. The purpose of such strategy is to ensure that the model will predict the same class for original and attacked images. The other method is denoising autoencoders, which denoise the perturbated images, that we later use on the network.

CHAPTER 4

IMPLEMENTATION AND RESULTS

4.1 Experiments

In this part we will show how the experiments were conducted and we will summarize the results in terms of different metrics. We have used CIFAR-10 dataset and the codes were run in PyCharm IDE, using TensorFlow package. This section is divided in three parts, building the base model, fooling the model using adversarial attacks and implementing some defenses to those attacks. For each of them we will explain the experiments and their results.

We will use accuracy as the most common used metric for benchmarking. The neural network accuracy will be compared among the base model, the attacks we will implement and the defense methods.

4.1.1. Base model

We have built a deep neural network, CNN type, as the benefits of multiple layers stand in the fact that they can learn features at various levels of abstraction. We have built a six layered CNN, which has next a flatten layer. As there are 10 classes, the output layer is a dense layer with 10 nodes. Activation function is Softmax activation.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
activation (Activation)      (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
activation_4 (Activation)    (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
activation_5 (Activation)    (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 10)                20490
=================================================================
Total params: 309,290
Trainable params: 308,394
Non-trainable params: 896
_________________________________________________________________
```

Figure 14. Base model summary

As we have built and trained the model, we have plotted the first 25 images from both train and test dataset along with their corresponding class, using Python library Matplotlib.

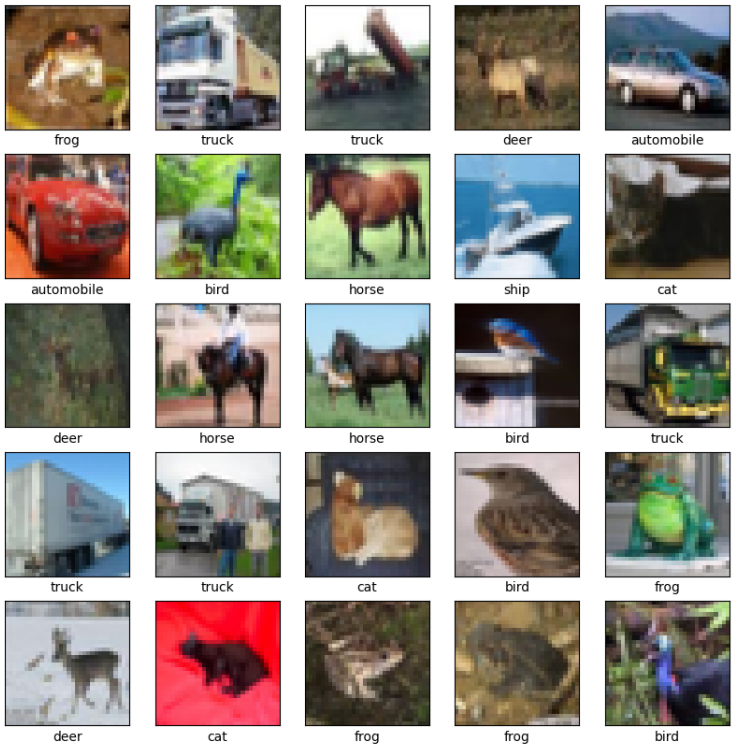


Figure 15. Dataset train images

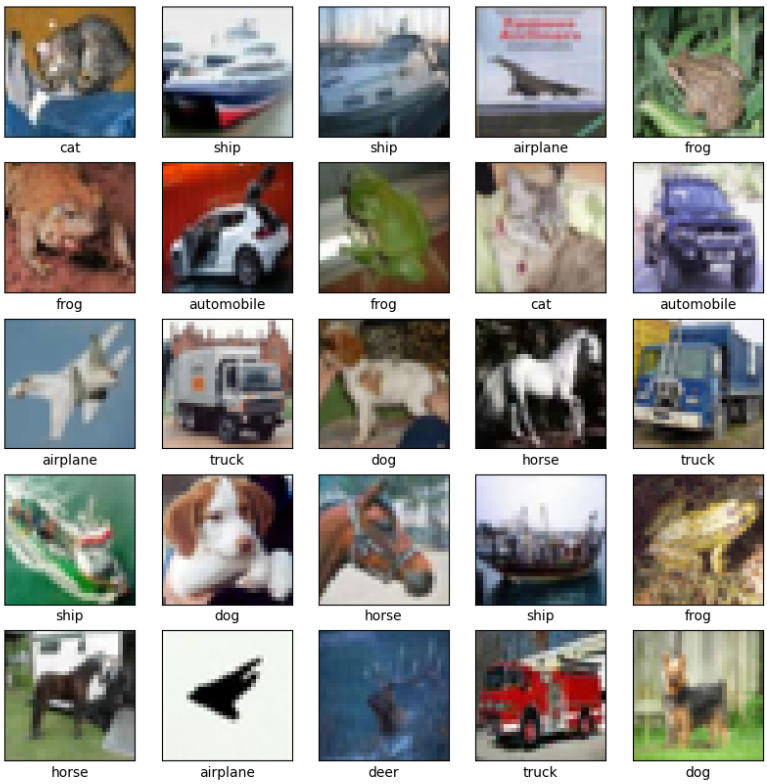


Figure 16. Dataset test images

We trained the model with batch size of 64 and number of epochs of 125. This model achieved a pretty good result, which are reported in the table below.

Table 3. Base model test results

Test Accuracy	***88.780***
Loss	0.457

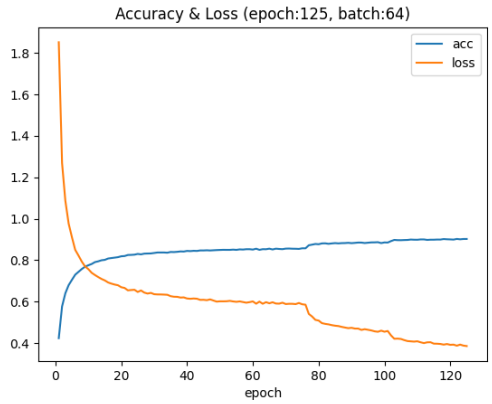
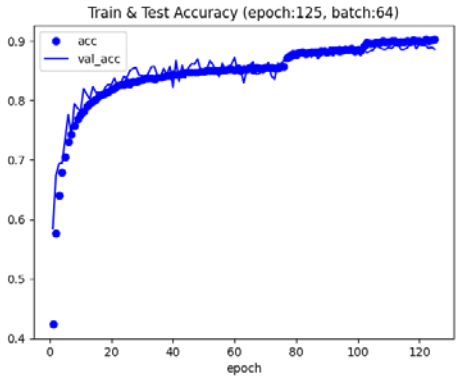


Figure 17. Accuracy, Loss vs. Epochs



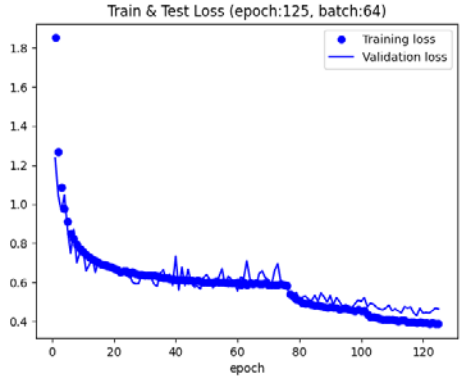


Figure 18. Train, test accuracy vs. epochs

Figure 19. Train, test loss vs. epochs

4.1.2. Fast Gradient Sign Method attack implementation

This method is a white box attack, using it leads to misclassification. In this case, the attacker has complete access to the model being attacked. It works by using the gradients of the neural network in order to create adversarial example. In our case we have fooled a pre trained model. The first step of this attack is to create perturbations, which will misrepresent the image, creating an adversarial image. The gradients are taken with respect to image. We have tried this for three different epsilons values to perturbate both train and test images, as illustrated below.

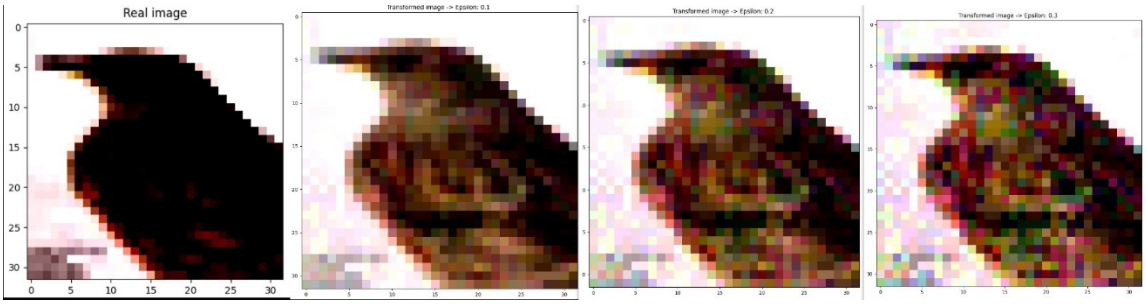


Figure 20. a) Train image 1 - FGSM b) eps=0.1 c) eps=0.2 d) eps=0.3

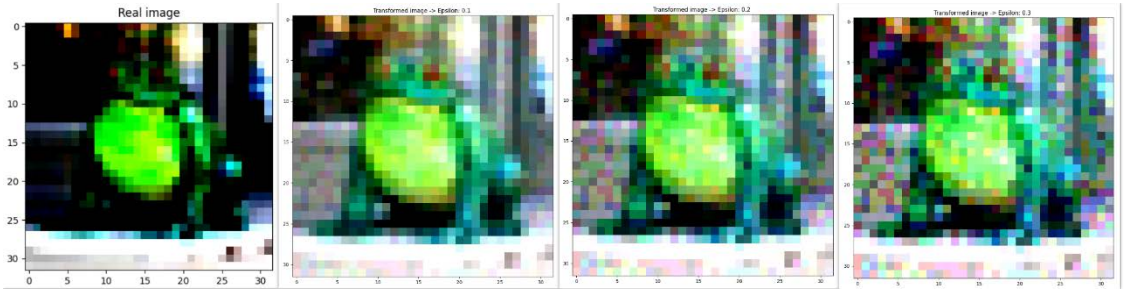


Figure 21. a) Train image 2 - FGSM b) eps=0.1 c) eps=0.2 d) eps=0.3

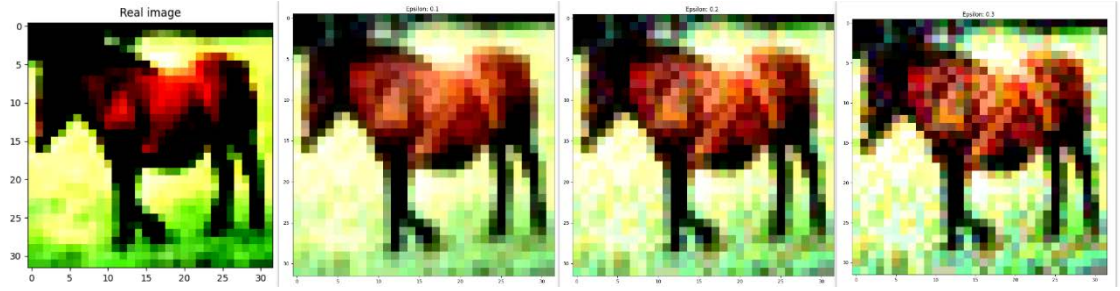


Figure 22. a) Train image 3 - FGSM b) eps=0.1 c) eps=0.2 d) eps=0.3

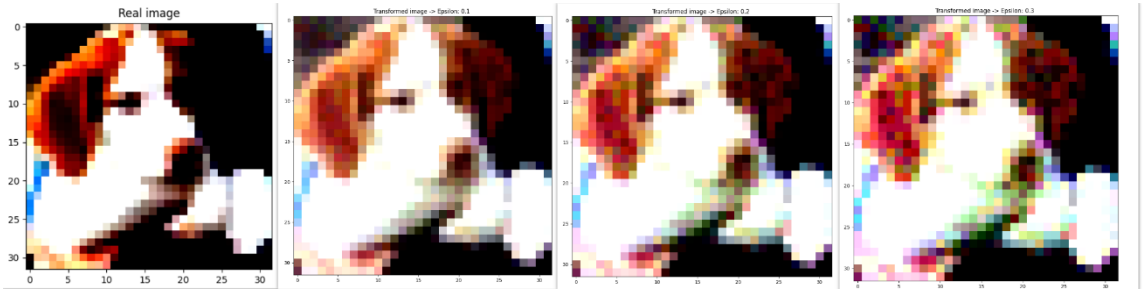


Figure 23. a) Test image 1 - FGSM b) eps=0.1 c) eps=0.2 d) eps=0.3

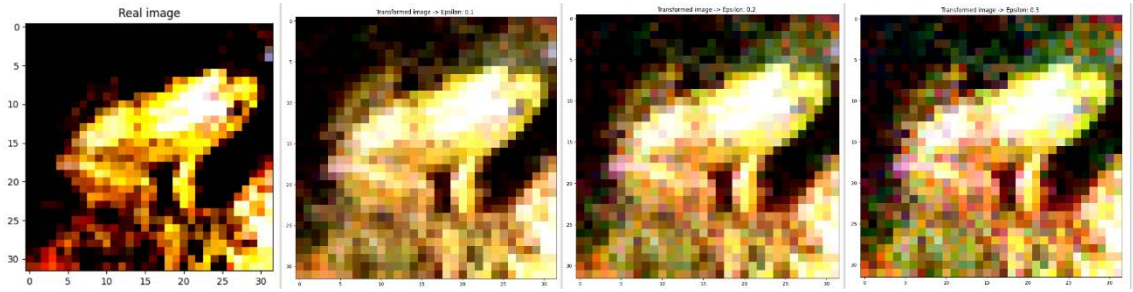


Figure 24. a) Test image 2 - FGSM b) eps=0.1 c) eps=0.2 d) eps=0.3

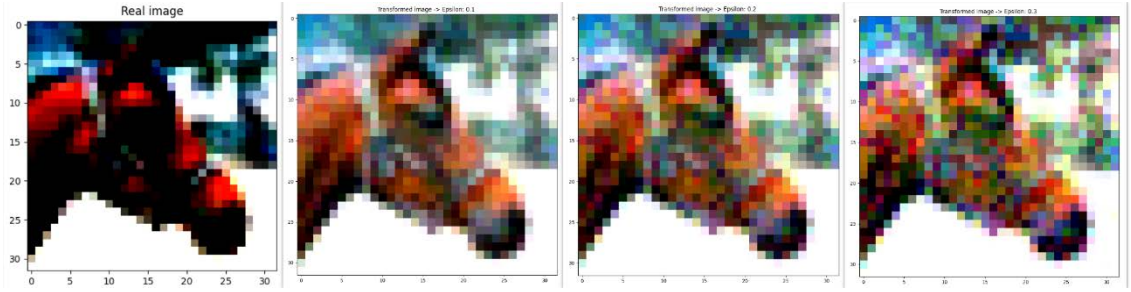


Figure 25. a) Test image 3 - FGSM b) eps=0.1 c) eps=0.2 d) eps=0.3

Firstly, we evaluated the pretrained model on the real images. Then, we add perturbations to both train and test images, as shown in the figures from 19 to 24 shown above. Adversarial examples were generated with three different epsilon magnitudes. We evaluate both train and test accuracies and loss in the pretrained model with the

adversarial images and compare the results. Table 4. summarizes the results of FGSM attack.

Table 4. FGSM attack results

	Base model	***Epsilon = 0.1***	***Epsilon = 0.2***	***Epsilon = 0.3***
Train Accuracy	93.7%	5.7%	5.4%	6.6%
Test Accuracy	88.6%	5.1%	4.9%	6.4%
Train Loss	0.29	8.01	8.09	8.1
Test Loss	0.46	8.34	8.17	8.19

From the table 5, we notice that both train and test accuracy of the model evaluated with adversarial train and test images, drop drastically. Attacking the input images with the lowest value of epsilons 0.1, lowers the train accuracy from 93.7 to 5.7, with a difference of 88%. While the test accuracy drops by 83%.

Perturbated images with epsilon 0.2 result to be even worse, as value of noise increases, and the images become more foolish for the model. Epsilon value of 0.3 on images affect the accuracies to be a little bit greater in value than perturbated images with epsilon 0.1 and 0.2 but is still very low compared with the base model accuracy.

Regarding train loss and test loss, model evaluation on adversarial samples increases the loss from 0.29 to 8.01 and from 0.46 to 8.34, respectively for perturbated images by 0.1 epsilon value. The same increase for loss is noticed on the case of noised images with the epsilon magnitude of 0.2 and 0.3.

What we conclude from these results is that no matter what value of epsilon is applied to perturbate the images, these adversarial examples are enough to drop the accuracy of the model at a highest point.

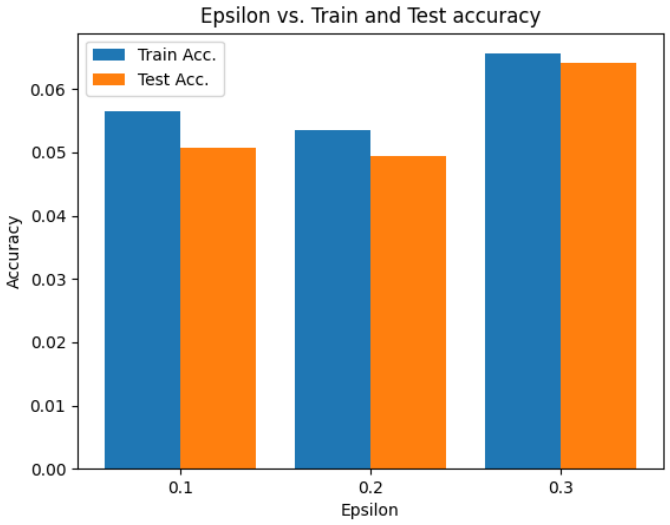


Figure 26. Accuracy vs. Epsilon for FGSM attack

Figure 25 shows the values of train and test accuracy of model evaluated with perturbated images by different epsilon magnitudes.

4.1.3. Basic Iterative Method attack implementation

This attack method produces adversarial images as well, as an iterative version of FGSM. The difference is that we apply it multiple times with a small step size. The pixel values of intermediate results are clipped after each step to ensure that they are in a neighborhood of the original image. In our work, we have previously trained the model with clean images of the CIFAR-10 dataset. Then the train images and the test images are attacked by this method for epsilons 0.1, 0.2 and 0.3, respectively. We have set 40 number of iterations. For each value of epsilon, we have plotted the images to have a better representation of the real and perturbated images.

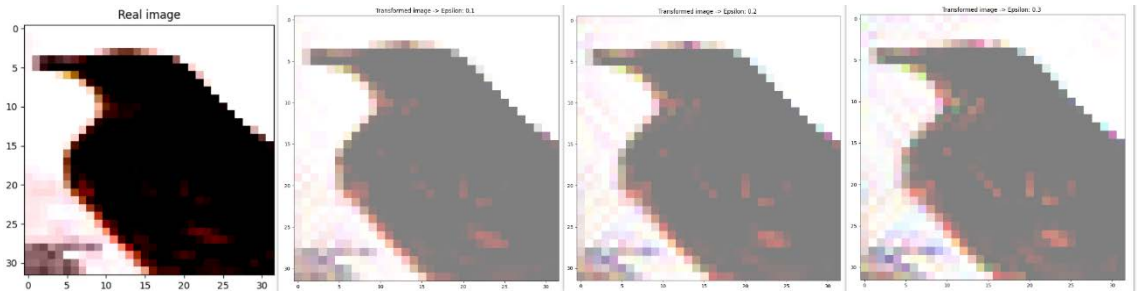


Figure 27. a) Train image 1 - BIM b) eps=0.1 c) eps=0.2 d) eps=0.3

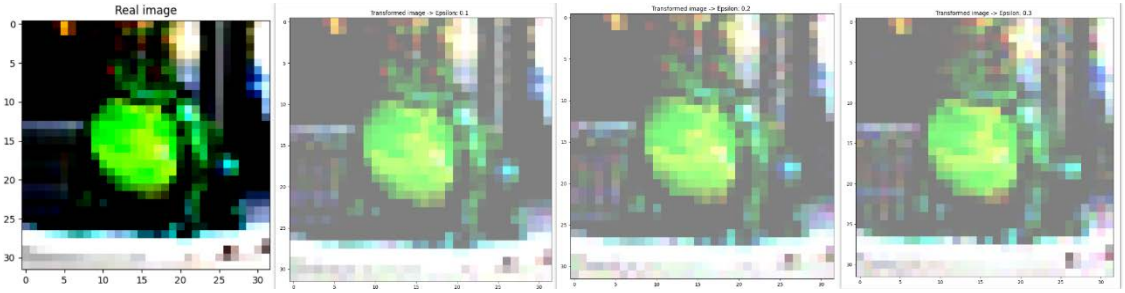


Figure 28. a) Train image 2 - BIM b) eps=0.1 c) eps=0.2 d) eps=0.3

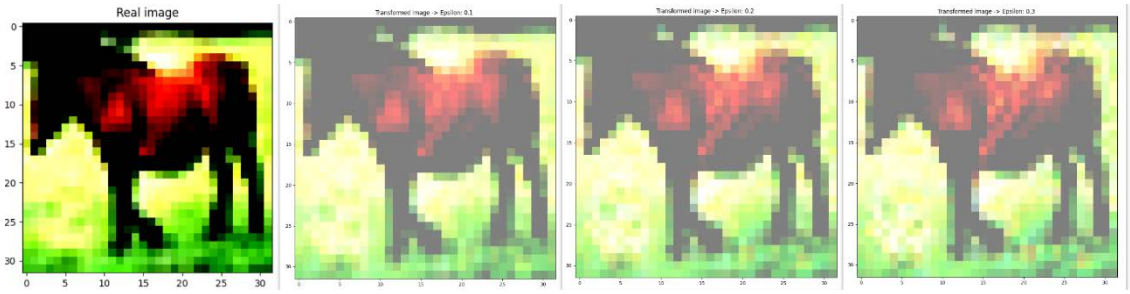


Figure 29. a) Train image 3 - BIM b) eps=0.1 c) eps=0.2 d) eps=0.3

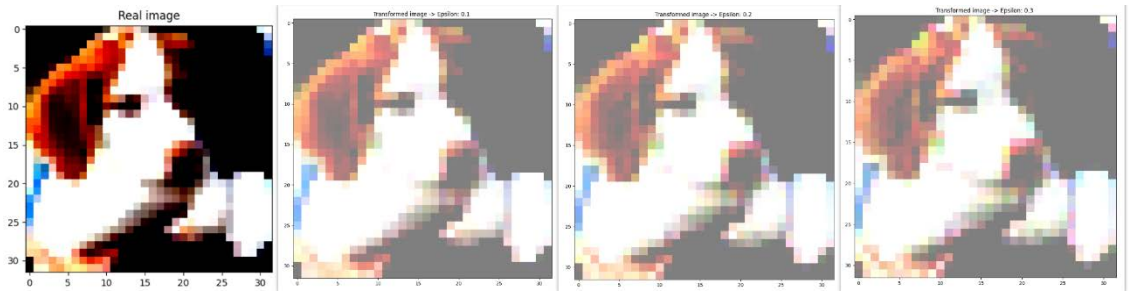


Figure 30. a) Test image 1 - BIM b) eps=0.1 c) eps=0.2 d) eps=0.3

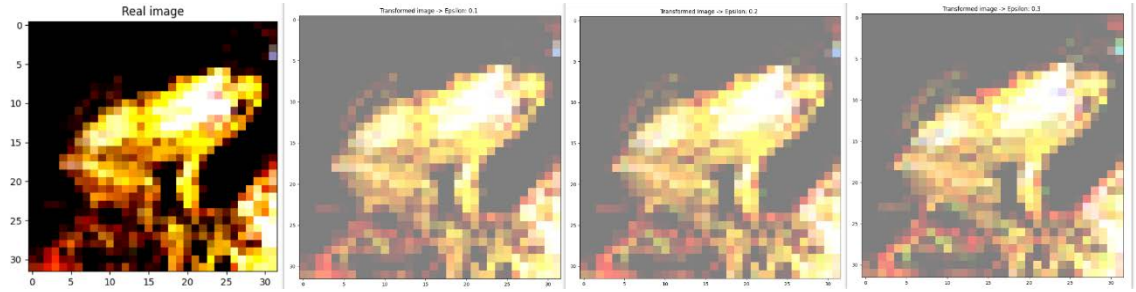


Figure 31. a) Test image 2 - BIM b) eps=0.1 c) eps=0.2 d) eps=0.3

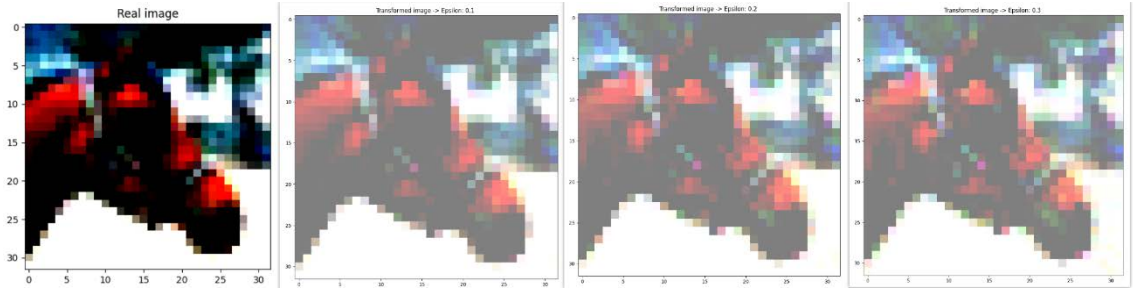


Figure 32. a) Test image 3 - BIM b) eps=0.1 c) eps=0.2 d) eps=0.3

We calculated train and test accuracy and loss with the model pretrained on adversarial images created with BIM attack. The results are summarized in the table below.

Table 5. BIM attack results

	Base model	***Epsilon = 0.1***	***Epsilon = 0.2***	***Epsilon = 0.3***
Train Accuracy	93.7%	1.2%	0.2%	0%
Test Accuracy	88.6%	1.2%	0.2%	0.1%
Train Loss	0.29	13.01	20.38	24.52
Test Loss	0.46	13.12	20.46	24.6

From the results of the table 5, we conclude that BIM adversarial attack affect the accuracy very bad. The loss from base model to epsilon 0.1 magnitude with BIM attack is 92% in train accuracy and 85% in test accuracy. Train loss value and test loss have increased from 0.29 to 13.01 and from 0.46 to 13.12, respectively. Attacking the train and test images of CIFAR-10 dataset with epsilon value of 0.2 and 0.3 is even worse, or we can say that the accuracy is 0%. Loss is logically increased. The figure below illustrates the relation of epsilon value on the perturbations of the images and the train and test accuracies.

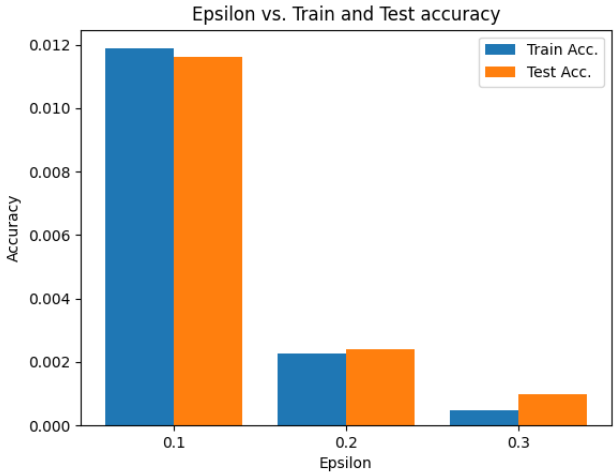


Figure 33. Accuracy vs. Epsilon for BIM attack

This basic iterative method attack performs even worse than the model evaluated with adversarial examples, which are generated by the other gradient based attack, previously shown, FGSM. Furthermore, it required a much more computation power than FGSM as it is iterative.

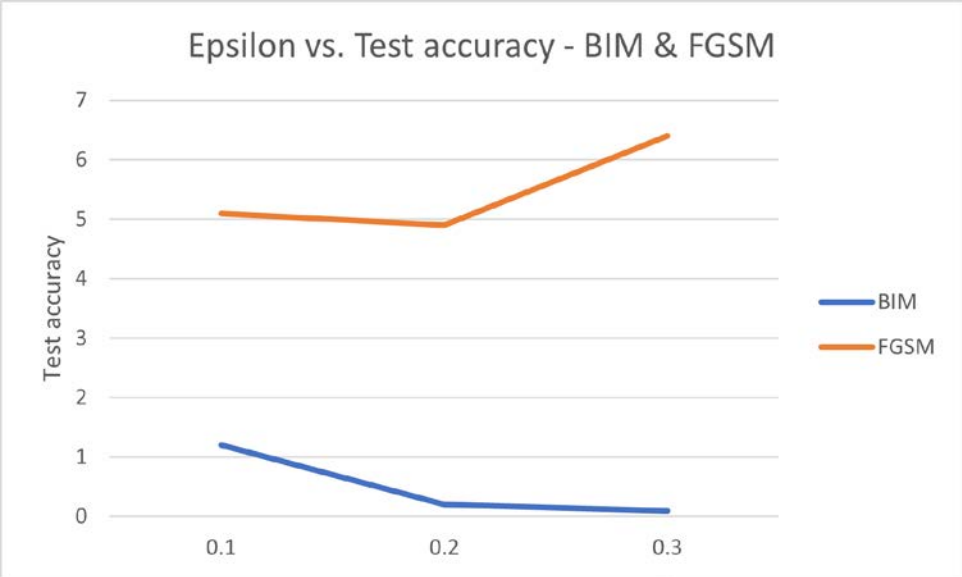


Figure 34. BIM vs. FGSM

4.1.4. Adversarial training defense implementation

This method consists of including in the model's training data the adversarial examples, making possible for the model to classify them. The training takes both clean and adversarial samples. In our work, we have implemented FGSM adversarial training and BIM adversarial training (with three different epsilons values). The results are summarized and compared below.

Table 6. FGSM Adversarial training (epsilon=0.1)

	Train accuracy	***Test accuracy***	***Train Loss***	***Test Loss***
Base model on adversarial data	5.7%	5.1%	8.01	8.34
Defended model on clean images	90.7%	85.2%	0.38	0.59
Defended model on adversarial data	94.2%	89.9%	0.28	0.44

Table 7. FGSM adversarial training (epsilon=0.2)

	Train accuracy	***Test accuracy***	***Train Loss***	***Test Loss***
Base model on adversarial data	5.4%	4.9%	8.09	8.17
Defended model on clean images	91.4%	86.6%	0.35	0.52
Defended model on adversarial data	95.1%	91.6%	0.25	0.38

Table 8. FGSM adversarial training (epsilon=0.3)

	Train accuracy	***Test accuracy***	***Train Loss***	***Test Loss***
Base model on adversarial data	6.6%	6.4%	8.18	8.19
Defended model on clean images	91.1%	86.5%	0.36	0.52
Defended model on adversarial data	93.1%	89.8%	0.32	0.43

As we have previously reported, the base model reached a 93.7% train accuracy and 88.6% test accuracy. Then, we evaluated the model with adversarial data perturbated using FGSM attack and with epsilons of 0.1; 0.2 and 0.3. The accuracies in the three cases dropped in a significant way, while loss values were up. Next step was to implement adversarial training. We trained the model with both clean, regular images and FGSM adversarial images, generating in this way train dataset of 100000 images, twice the train images of CIFAR-10 dataset (with all three values of epsilons). We evaluated the model trained with these data, on both the original images and on the adversarial images. The results were pretty well and has a significant increase compared with the FGSM results. For all values of epsilons, the model trained on clean images increased the train accuracy and the test accuracy. Whereas the train and test loss was dropped. The defended model performed even better, approaching in this way the base model accuracy with slight changes, which from the observations we see that the higher the value of epsilon, the lower the accuracy of the defended model. But, in overall this defense method performed really well on this CNN attack with FGSM method and being very close to the base model results. Figures below also illustrate the results.

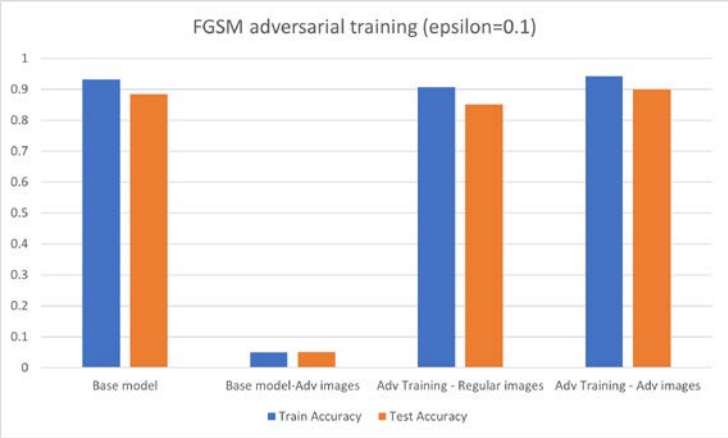


Figure 35. FGSM adversarial training results (eps=0.1)

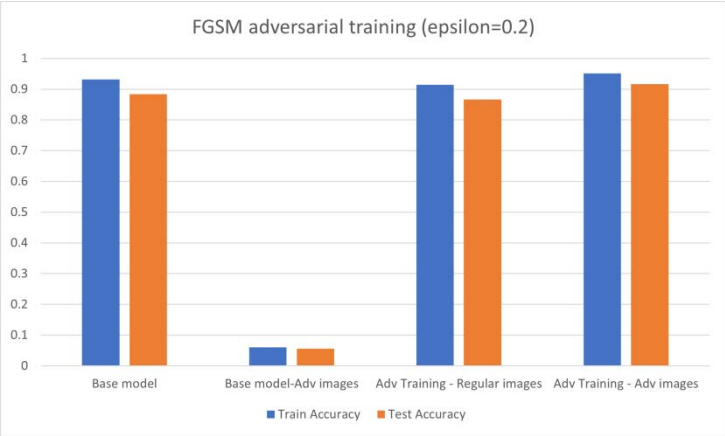


Figure 36. FGSM adversarial training results (eps=0.2)

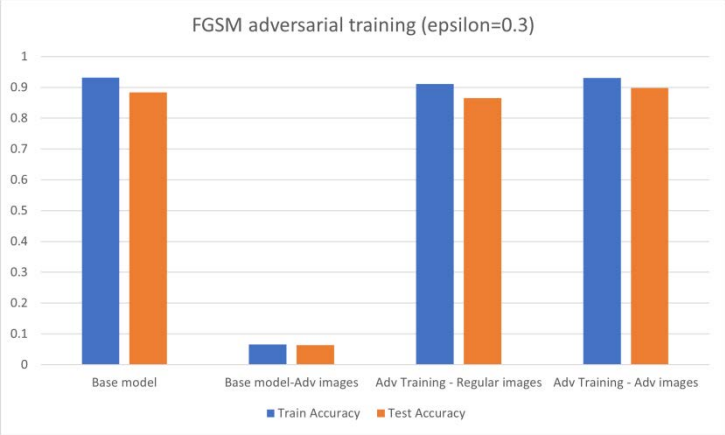


Figure 37. FGSM adversarial training results (eps=0.3)

As it was stated before, we have implemented BIM adversarial training and the results are illustrated and listed below.

Table 9. BIM adversarial training (epsilon=0.1)

	Train accuracy	***Test accuracy***	***Train Loss***	***Test Loss***
Base model on adversarial data	1.2%	1.2%	13.09	13.12
Defended model on clean images	91.6%	87.7%	0.35	0.49
Defended model on adversarial data	79.6%	74.4%	0.69	0.9

Table 10. BIM adversarial training (epsilon=0.2)

	Train accuracy	***Test accuracy***	***Train Loss***	***Test Loss***
Base model on adversarial data	0.2%	0.2%	20.38	20.46
Defended model on clean images	91.2%	87.2%	0.36	0.49
Defended model on adversarial data	80.3%	75.3%	0.67	0.85

Table 11. BIM adversarial training (epsilon=0.3)

	Train accuracy	***Test accuracy***	***Train Loss***	***Test Loss***
Base model on adversarial data	0%	0.1%	20.52	24.6
Defended model on clean images	90.3%	86.2%	0.39	0.53
Defended model on adversarial data	79.9%	75.4%	0.69	0.86

We conducted the experiment to test how BIM adversarial training with three epsilon magnitudes (0.1; 0.2; 0.3) affects model accuracy. The accuracies of the model evaluated on pure adversarial data generated with BIM attack in the three cases dropped in a significant way, while loss values were increased, as previously reported. For this adversarial training defense method, we trained the model with both clean, regular images and BIM adversarial images, generating in this way train dataset of 100000 images, twice the train images of CIFAR-10 dataset (with all three values of epsilons).

We evaluated the model trained with these data, on both the original images and on the adversarial images. The results were pretty well and had a significant increase compared with the pure BIM attack results. For all values of epsilons, the model trained on clean images increased the train accuracy and the test accuracy. Whereas the train and test loss were dropped. The defended model accuracy was increased in a high rate, but comparing with FGSM adversarial training, BIM adversarial training did not perform as well as its accuracy. Despite this, we can say that BIM adversarial training reached a good score when the model was trained with both clean and adversarial images. The figures below illustrate the FGSM adversarial training on real and adversarial data against the BIM adversarial data.

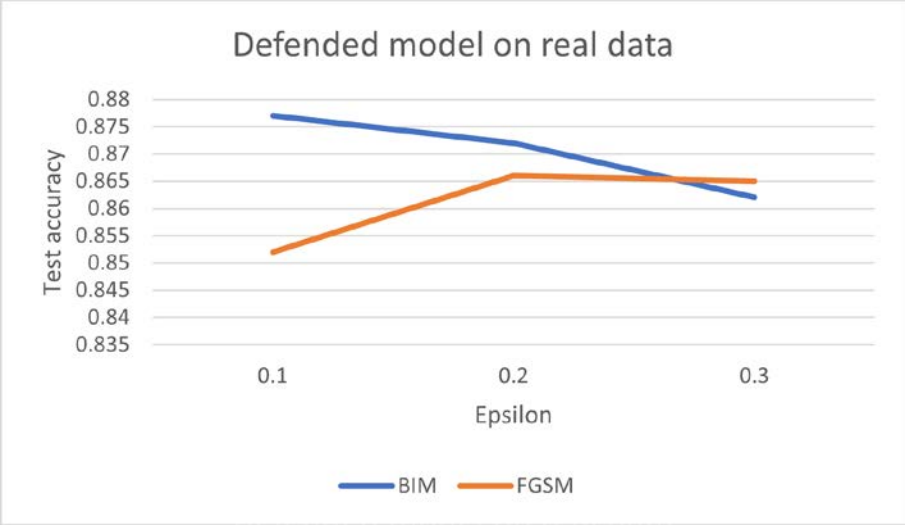


Figure 38. BIM vs. FGSM adversarial training on real data

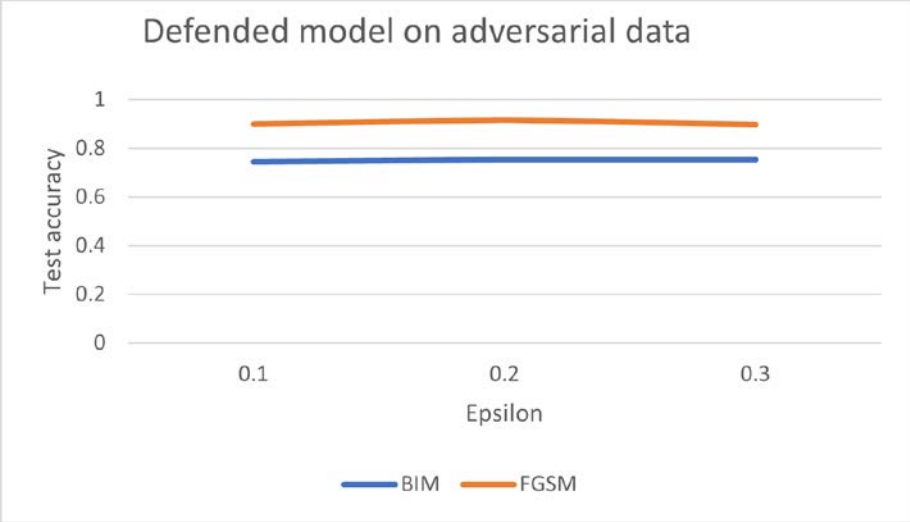


Figure 39. BIM vs. FGSM adversarial training on adversarial data

4.1.5. Denoising autoencoders defense implementation

We use autoencoders as a mean for denoising images before feeding them to the standard neural network. They have the architecture of a neural network. In this part, we will describe how we built the autoencoder. We define a convolutional block and a deconvolutional block. Convolutional blocks consist of 3 operations: 2D convolution, batch normalization and ReLu activation. Deconvolutional blocks consist of 3 operations: 2D transposed convolution, batch normalization and also ReLu activation. We compile the autoencoder using mse loss and adam optimizer and we fit the autoencoder to the adversarial training set. The adversarial images are perturbated uding FGSM as described in the sections above with 3 different epsilon values: 0.1; 0.2; 0.3.

After the denoising autoencoder is trained, we predict the clean train and cleaned test and we plot the images and evaluate the results, which are illustrated and reported below.

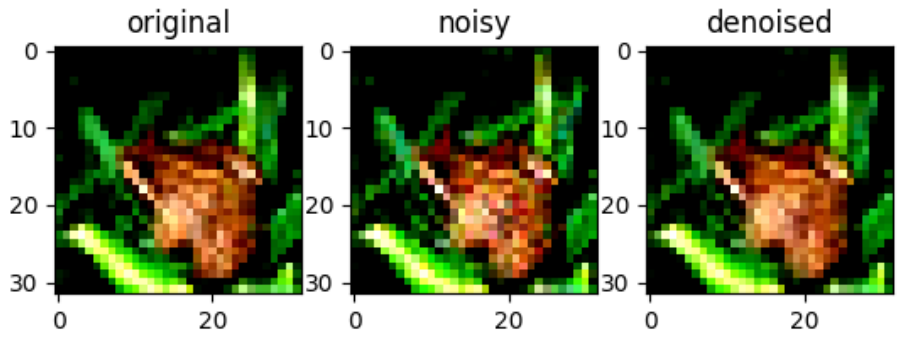


Figure 40. a) Real b) noised - FGSM (eps=0.1) c) denoised using autoencoder

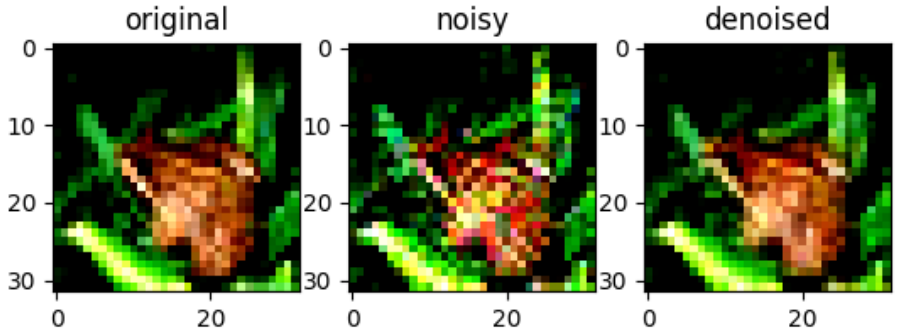


Figure 41. a) Real b) noised - FGSM (eps=0.2) c) denoised using autoencoder

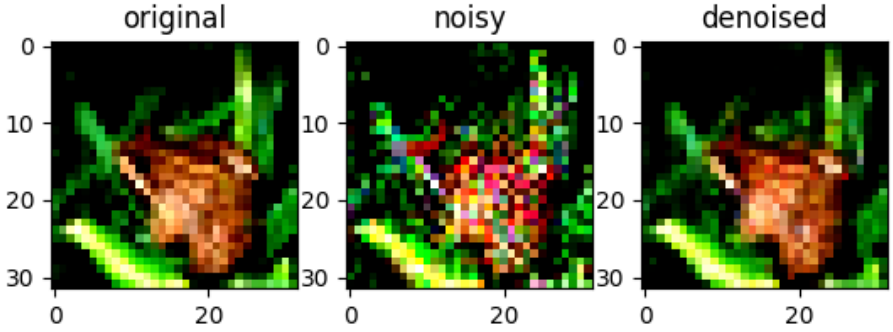


Figure 42. a) Real b) noised - FGSM (eps=0.3) c) denoised using autoencoder

Table 12. Denoising autoencoder to FGSM (epsilon=0.1)

	Train accuracy	***Test accuracy***	***Train loss***	***Test loss***
Base model on adversarial data	5.7%	5.1%	8.01	8.34
Model on denoised images	34.5%	33.4%	2.71	2.84

Table 13. Denoising autoencoder to FGSM (epsilon=0.2)

	Train accuracy	***Test accuracy***	***Train loss***	***Test loss***
Base model on adversarial data	5.4%	4.9%	8.09	8.17
Model on denoised images	44.2%	37.8%	2.06	2.37

Table 14. Denoising autoencoders on FGSM (epsilon=0.3)

	Train accuracy	***Test accuracy***	***Train loss***	***Test loss***
Base model on adversarial data	6.6%	6.4%	8.18	8.19
Model on denoised images	48.5%	40.5%	1.86	2.21

From the information given in the tables above, we can state that the accuracy of the model evaluated on cleaned images by the denoising autoencoder, is increased considerably. But that is not the desired accuracy compared to the base model, because the maximum it can reach is almost 40% (test accuracy), while the base model has a test accuracy around 88%.

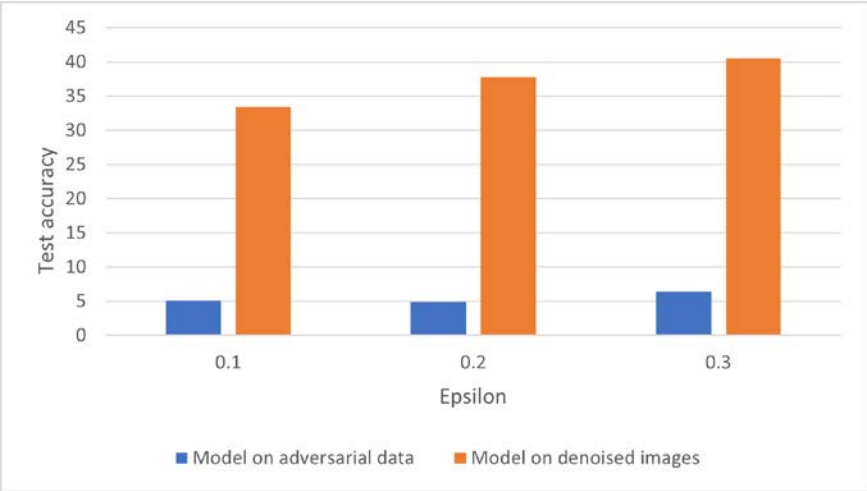


Figure 43. Test accuracy for model on adversarial and denoised data

The adversarial training works better on this attack as a defense mechanism to the input images, as illustrated in the figure below.

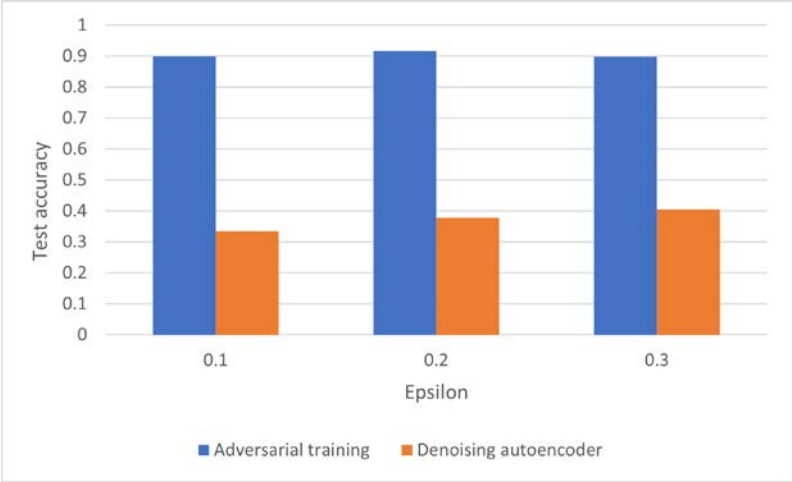


Figure 44. Adversarial training vs. Autoencoder test accuracy with FGSM

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1. Conclusions

The aim of this thesis was to provide a comprehensive review on the results of applying adversarial attacks on deep neural network using CIFAR-10 dataset and trying the defense methods against these attacks. We used CNN as a base model and then experimented with two attacks, and two defenses, to report and analyze accuracy of the model and how the model tends to be misclassified.

The deep neural networks are really vulnerable to adversarial examples. A small perturbation in the input sample, “destroys” the model, leading to misclassification. Except of the attacks and defenses we have implemented for our DNN model, there are a lot more, some of them of same kind, white-box and some are classified as black-box. The defense methods are of a great variety as well, but there are specific to different types of attacks, meaning that all defenses cannot make the model robust. They depend on the type of the attack they are trying to protect the model from.

5.2. Future Work

Firstly, we suggest to further dig into the defense mechanisms. In this work, we have given some insights for the defenses against only FGSM and BIM. But there can be methods that can make the model more robust even from the non-gradient attack approaches. Furthermore, we suggest the autoencoder to be set up in a way that it

increases more the model accuracy. As our focus was untargeted attacks, as a future work we can ensure that the defenses perform well for targeted attacks as well.

REFERENCES

[1] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, Depbdeep Mukhopadhyay, "A survey on adversarial attacks and defences," *CAAI Transactions on Intelligence Technology,* 2020.

[2] Maurizio Capra, Beatrice Bussolino, Alberto Marchisio, Muhammad Shafique, Guido Masera, Maurizio Martina, "An Updated Survey of Efficient Hardware Architectures for Accelerating Deep Convolutional Neural Networks," *future internet,* 2020.

[3] P. G. A. L. Pitropakis, "A taxonomy and survey of attacks against machine learning," *Elservier,* 2019.

[4] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, Rob Fergus, "Intriguing properties of neural networks," *Computer Vision and Pattern Recognition,* 2014.

[5] Wenqi Wang, Run Wang, Lina Wang, Zhibo Wang, Aoshuang Ye, "Towards a Robust Deep Neural Network in Texts: A Survey," *IEEE,* 2021.

[6] Shilin Qiu, Qihe Liu, Shijie Zhou and Chunjiang Wu, "Review of Artificial Intelligence Adversarial Attack and Defense Technologies," *Applied Sciences MDPI,* 2019.

[7] Jure Sokoli´c, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues, "Robust large margin," *Transactions on Signal Processing,* 2017.

[8] K. X. Zhou, "A survey of game theoretic approach for adversarial machine

learning," *Wires Data Mining Knowl. Discov.*, 2018.

[9] Duddu, "A Survey of Adversarial Machine Learning in Cyber Warfare," *Defence Science Journal*, 2018.

[10] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy, "Explaining and Harnessing Adversarial Examples," *CoRR abs*, 2014.

[11] ANIRBAN CHAKRABORTY, MANAAR ALAM, VISHAL DEY, ANUPAM CHATTOPADHYAY, DEBDEEP MUKHOPADHYAY, "Adversarial Attacks and Defences: A Survey," *ACM*, 2018.

[12] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai, "One Pixel Attack for Fooling Deep Neural Networks," *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, 2019.

[13] Ninghao Liu, Mengnan Du, Ruocheng Guo, Huan Liu, Xia Hu, "Adversarial Attacks and Defenses: An Interpretation Persepective," *arXiv*, 2020.

[14] "Adversarial Attacks and Defenses in Deep Learning," *Elsevier*, *Engineering*, 2020.

[15] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, Jianguo Li, "Boosting Adversarial Attacks with Momentum," *arXiv*, 2018.

[16] Guangling Sun, Yuying Su, Chuan Qin, Wenbo Xu, Xiaofeng Lu and Andrzej Ceglowski, "Complete Defense Framework to Protect Deep Neural Networks against Adversarial Examples," *Mathematical Problems in Engineering*, 2020.

[17] Austin Short, Trevor La Pay, Apurva Gandhi, "Defending Against Adversarial Examples," *Sandia National Laboratories*, 2019.

[18] A. Goel, "An Empirical Review of Adversarial Defenses," *arXiv,* 2020.

[19] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio, "Adversarial Machine Learning at Scale," *arXiv,* 2016.

[20] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz, "Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning," *arXiv,* 2017.

[21] Aleksander M˛adry, Aleksandar Makelov, Ludwig Schmidt and Dimitris Tsipras, "Towards Deep Learning Models Resistant to Adversarial Attacks," *arXiv,* 2019.

[22] Nicholas Carlini and David Wagner, "MagNet and “Efficient Defenses Against Adversarial Attacks” are Not Robust to Adversarial Examples," *arXiv,* 2017.

[23] Uri Shaham, Yutaro Yamada, and Sahand Negahban, "Understanding Adversarial Training: Increasing Local Stability of Neural Nets through Robust Optimization.," *arXiv,* 2015.

[24] Chunchuan Lyu, Kaizhu Huang, and Hai-Ning Liang, "A Unified Gradient Regularization Family for Adversarial example," *IEEE International Conference on Data Mining, ICDM,* 2015.

[25] Malhar Jere, Sandro Herbig, Christine Lind, Farinaz Koushanfar, "Principal Component Properties of Adversarial Samples," *arXiv,* 2019.

[26] O. P. Atieno, "AN ANALYSIS OF THE STRENGTHS AND LIMITATION OF QUALITATIVE AND QUANTITATIVE RESEARCH PARADIGMS," *Problems of Education in the 21st Century,* 2009.

[27] T. University, "The CIFAR-10 dataset," [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html. [Accessed 01 07 2021].