NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Adversarial Attacks and Defences: Threats and Prospects Analysis

Aristi M. Papastavrou

Supervisor: **Konstantinos Chatzikokolakis,** Associate Professor

ATHENS

September 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιθέσεις και Άμυνες χρησιμοποιώντας Adversarial Μηχανική Μάθηση: Ανάλυση κινδύνων και Προοπτικών

Αρίστη Μ. Παπασταύρου

Επιβλέπων: **Κωνσταντίνος Χατζηκοκολάκης,** Αναπληρωτής Καθηγητής

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2022

BSc THESIS

Adversarial Attacks and Defences: Threats and Prospects Analysis

Aristi M. Papastavrou
S.N.: 1115201800154

SUPERVISOR: **Konstantinos Chatzikokolakis,** Associate Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιθέσεις και Άμυνες χρησιμοποιώντας Adversarial Μηχανική Μάθηση: Ανάλυση κινδύνων και Προοπτικών

Αρίστη Μ. Παπασταύρου
Α.Μ.: 1115201800154

ΕΠΙΒΛΕΠΩΝ: **Κωνσταντίνος Χατζηκοκολάκης,** Αναπληρωτής Καθηγητής

ABSTRACT

As users in the new age of data, most of the new technologies that we use in our day-to-day life are based on machine learning models that are now more than ever, extremely complex. Furthermore innovative technologies and state-of-the-art networks that used to be considered safe, and effective appear to be unstable to small, well sought, perturbations of the images.Despite the importance of this phenomenon, no effective methods have been proposed to accurately compute the robustness of state-of-the-art deep classifiers to such perturbations on large-scale datasets.This makes it difficult to apply neural networks in security-critical areas.
As a result exploiters,could easily trick the model into making incorrect predictions or give away sensitive information. Fake (adversarial) data could even be used to corrupt models without us knowing. The field of adversarial machine covers both sides of the coin.

Adversarial machine learning is the study of the attacks on machine learning algorithms, and of the defenses against such attacks. A recent survey exposes the fact that practitioners report a dire need for better protecting machine learning systems in industrial applications. In this thesis we studied adversarial attacks that are more likely to fool from everyday users to big companies due to the very realistic result they create and the difficulty in finding out the error if you aren't familiar with the data.Take as an example smart-automated cars. The everyday driver trusts the vendors that they have programmed the car well so that when its in autopilot, it will know how to analyze the signals in the road and know when to start, stop,go. What happens though when a malicious programmer applies a black-box attack to the database of said car and mislabels the STOP sign to GO so to confuse the cars computer?

In fact, very small and often imperceptible perturbations of the data samples are sufficient to fool state-of-the-art classifiers and result in incorrect classification. So after taking into account the newly discovered gaps of Deep Neural Networks regarding noise perturbations, what does it take to create a well structured attack that would lower the models confidence? Is it possible to create defence mechanisms that can secure our networks from the adversarial and some times, dynamic, attacks?

The goal of this thesis is to further examine, analyze and understand the mechanism, reasoning behind adversarial attacks against machine learning models, as well as the effectiveness of each attack to the same target,database.Also we will be presenting a hybrid version of the famously known boundary attack, propose optimizations for different attack strategies and evaluate some known defence mechanisms against said adversarial attacks.

Throughout this thesis, a plethora of plots and boards will be provided to the reader, to enhance his/her understanding of this study via experiments.

SUBJECT AREA: Adversarial Machine Learning, Data Protection

KEYWORDS: Adversarial Machine Learning, Neural Networks,Perturbations, Data Privacy, Classification, Mislabeling, Adversarial Attacks and Defences

ΠΕΡΙΛΗΨΗ

Ως χρήστες στη νέα εποχή των δεδομένων, η πλειονότητα των νέων τεχνολογιών που χρησιμοποιούμε στην καθημερινότητα μας, βασίζονται σε εξαιρετικά σύνθετα μοντέλα μηχανικής μάθησης. Οι καινοτόμες τεχνολογίες και τα δίκτυα αιχμής που παλαιότερα θεωρούνταν ασφαλή και αποτελεσματικά, φαίνεται να είναι ασταθή σε μικρές και συνάμα δυναμικές, διαταραχές. Παρά το γεγονός αυτό, δεν έχουν προταθεί αποτελεσματικές μέθοδοι που να υπολογίζουν με ακρίβεια την ευρωστία των προηγμένων νευρωνικών δικτύων, υπό την απειλή τέτοιων διαταραχών. Ως αποτέλεσμα, κακόβουλοι προγραμματιστές, θα μπορούσαν με την παραγωγή πλαστών (adversarial) δεδομένων να ξεγελάσουν το μοντέλο για να κάνει λανθασμένες προβλέψεις ή να αποκαλύψουν ευαίσθητες πληροφορίες υπο την άγνοια μας.

Η (adversarial) μηχανική μάθηση είναι η μελέτη των επιθέσεων σε αλγόριθμους μηχανικής μάθησης και των αμυνών έναντι τέτοιων επιθέσεων. Μια πρόσφατη έρευνα αποκαλύπτει το γεγονός ότι οι επαγγελματίες αναφέρουν την απόλυτη ανάγκη για καλύτερη προστασία των συστημάτων μηχανικής μάθησης σε βιομηχανικές εφαρμογές. Σε αυτή την πτυχιακή μελετήσαμε επιθέσεις που είναι πιο πιθανό να ξεγελάσουν από καθημερινούς χρήστες έως μεγάλες εταιρείες λόγω του πολύ ρεαλιστικού αποτελέσματος που δημιουργούν και της δυσκολίας να εντοπιστεί το σφάλμα εάν κάποιος δεν είναι εξοικειωμένος με τα δεδομένα. Αναλογιστείτε τα έξυπνα - αυτοματοποιημένα αυτοκίνητα. Ο καθημερινός οδηγός εμπιστεύεται τους πωλητές ότι έχουν προγραμματίσει καλά το αυτοκίνητο, ώστε όταν βρίσκεται σε αυτόματο πιλότο, να ξέρει πώς να αναλύει τα σήματα στο δρόμο και να ξέρει πότε να ξεκινήσει, να σταματήσει, να φύγει. Τι συμβαίνει όμως όταν ένας κακόβουλος προγραμματιστής εφαρμόζει μια επίθεση στη βάση δεδομένων του εν λόγω αυτοκινήτου και αλλάζει την σημασία του συμβολου STOP σε GO έτσι ώστε να μπερδέψει τον υπολογιστή του αυτοκινήτου; Στην πραγματικότητα, πολύ μικρές και συχνά ανεπαίσθητες διαταραχές των δειγμάτων δεδομένων επαρκούν για να παραπλανηθούν οι ταξινομητές τελευταίας τεχνολογίας και να οδηγήσουν σε εσφαλμένη ταξινόμηση. Αφού λοιπόν ληφθούν υπόψη τα πρόσφατα ανακαλυφθέντα κενά των Νευρωνικών Δικτύων σχετικά με τις διαταραχές θορύβου, τι χρειάζεται για να δημιουργηθεί μια καλά δομημένη επίθεση που θα μείωνε την εμπιστοσύνη των μοντέλων; Είναι δυνατόν να δημιουργηθούν αμυντικοί μηχανισμοί που να προστατεύουν τα δίκτυά μας από τέτοιο είδος δυναμικών επιθέσεων;

Στόχος αυτής της διπλωματικής εργασίας είναι να εξετάσει περαιτέρω, να αναλύσει και να κατανοήσει τον μηχανισμό, τη συλλογιστική πίσω από τις αντίπαλες επιθέσεις κατά μοντέλων μηχανικής μάθησης, καθώς και την αποτελεσματικότητα κάθε επίθεσης στον ίδιο στόχο, βάση δεδομένων. Επίσης θα παρουσιάσουμε μια υβριδική έκδοση της περίφημης οριακής επίθεσης, προτείνουν βελτιστοποιήσεις για διαφορετικές στρατηγικές επίθεσης και αξιολογούν ορισμένους γνωστούς αμυντικούς μηχανισμούς έναντι των εν λόγω επιθέσεων αντιπάλου. Καθ' όλη τη διάρκεια αυτής της πτυχιακής, μια πληθώρα διαγραμμάτων και πινάκων θα παρασχεθεί στον αναγνώστη, προκειμένου να κατανοήσει την διεξαχθείσα έρευνα μας, μέσω πειραμάτων.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Μάθηση, Προστασία Δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Μηχανική Μάθηση, Νευρωνικά Δίκτυα, Θόρυβος, Ταξινόμηση, Προστασία Δεδομένων, Κατηγοροποιηση, Adversarial Επιθεσεις και Άμυνες

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

1. INTRODUCTION

1.1 Adversarial Machine Learning Overview

Data in the 21st Century is like Oil in the 18th Century: an immensely, untapped valuable asset. Like oil, for those who see Data's fundamental value and learn to extract and use it there will be huge rewards. We're in a digital economy where data is more valuable than ever. It's the key to the smooth functionality of everything from the government to local companies. Without it, progress would halt. According to estimates, all data in 2022 could reach 44 zettabytes. Nowadays, machine learning models in computer vision are used in many real-world applications, like self-driving cars, face recognition, cancer diagnosis, or even in next-generation shops in order to track which products customers take off the shelf so their credit card can be charged when leaving.The increasing accuracy of these machine learning systems is quite impressive, so it naturally led to a veritable flood of applications using them. Although the mathematical foundations behind them were already studied a few decades ago, the relatively recent advent of powerful GPUs gave researchers the computing power necessary to experiment and build complex machine learning systems. Today, state-of-the art models for computer vision are based on deep neural networks with up to several million parameters, and they rely on hardware that was not available just a decade ago.
One of the ways privacy could be breached, is by exploiting inherit vulnerabilities found in the structure and nature of many ML models a, deployed in public. This massive data evolution, has created many data breaches and security issues that have not yet been resolved. Example (and point of the whole thesis) of such exploitations are the adversarial attacks.As of today, attacking a machine learning model is easier than defending it. State-of-the art models deployed in real-world applications are easily fooled by adversarial examples if no defense strategy is employed, opening the door to potentially critical security issues. For example, Deep neural networks (DNN) have achieved state of the art performance for image classification tasks. However, state of the art DNNs have vulnerabilities to adversarial attacks. They are susceptible to small perturbations that are made in a meaningful way, which is not random noise.

Due to this gap in DNNs and inherited the risk of breaching privacy has motivated many senior coders and scientists to try and come up with new defences.The attacker crafts the attack, exploiting all the information they have about the model.So how a hybrid type of training or having more robust models could help defend our networks? How easily can we generate hybrid attacks and is it possible to defend against them all?
These are some of the questions, we are trying to answer in this Thesis.

1.2 Adversarial Attacks

Cyber attacks often cause data corruption and intentional tampering by an unexpected source, which could be crucial elements in the training data for deep neural networks. Although these models are popular for their accuracy and performance for computer vision tasks (such as classification detection, and segmentation), they are known to be extremely vulnerable to adversarial attacks. In this type of attack, the adversary induces minor but systematic perturbations in key model layers such as filters and input datasets as shown in Figure 1.1 . Even though this minor layer of noise is barely perceptible to human vision, it may cause drastic misclassification in critical scene elements such as road signs and traffic lights. This may eventually lead to AV crashing into other vehicles or pedestrians.
On the same note, one of the main reasons that ML Models are used in data exploitation is to help with the prediction, classification of data clusters (we could say that they work as classifiers). Prediction in data is about fitting a shape that gets as close to the data as possible.Most adversarial attacks usually aim to deteriorate the performance of classifiers on specific tasks, essentially to “fool” the machine learning algorithm.
Adversarial attacks are primarily of two types: (1) White-box and (2) Black-box. An attack is considered to be *White-box*, when we have complete knowledge of internal workings and details, while *Black-box* is an attack where we acquire limited knowledge of the model, with no labeled data sometimes.
An adversarial example refers to specially crafted input which is designed to look ”normal” to humans but causes misclassification to a machine learning model. Often, a form of specially designed ”noise” is used to elicit the misclassifications. The methods studied (and implemented/experimented with) for this thesis are given in the below list:

- ”white-box attacks:”
  - Fast Gradient Sign Method (FGSM)
  - DeepFool
  - Carlini and Wagner (C&W)
- ”black-box attacks:”
  - Boundary Attack

As a generic example, let us take a GoogLeNet trained on ImageNet to perform image classification as our machine learning model. Below you have two images of a panda that are indistinguishable to the human’s eye. The image on the left is one of the clean images in ImageNet dataset, used to train the GoogLeNet model. The one on the right is a slight modification of the first, created by adding the noise vector in the central image. The first image is predicted by the model to be a panda, as expected. The second, instead, is predicted (with very high confidence) to be a gibbon.

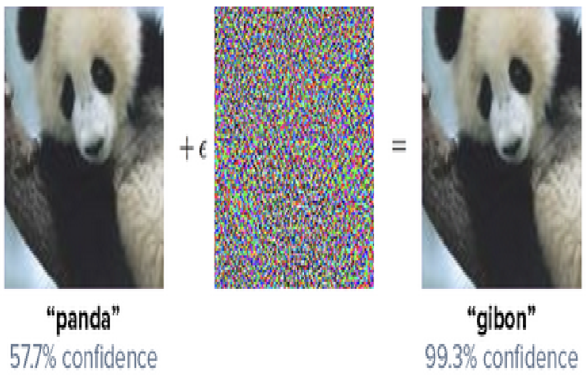


Figure 1.1: Example of adversarial attack: minor perturbations introduced to the training data cause misclassification of a critical traffic sign i.e. Yield instead of Stop sign. This incorrect prediction can be hardly perceptible to the human eye and thus have dangerous repercussions for autonomous vehicles.

1.3 Thesis' Goal

The main goal of this thesis is to understand the basic mechanisms of Adversarial Machine Learning,both from the attackers and defenders point of view. Our aim is to create a thorough analysis of the possible threats but also prospects that Adversarial ML proposes.

Specifically, regarding the attack portion of the thesis, we elaborate on famous white-box and black-box attacks.The following chapters aim to thoroughly explain, how each attack method that we implemented works, the mechanics of every stage of each attack and go in depth on the reason the attack works, firstly on a theoretical basis and, secondly, according to experimental results, statistics, plots, boards, etc.Finally, we propose optimizations (and even introduce our proposal for a hybrid version of a famous black-box attack).

As for the defence part of our research, the sector of defence mechanisms for adversarial attacks might not have evolved as much, we still provide the reader with a high level overview of proposed defences and some experimental results from our research based on the theory behind those defences.In the same spirit, we discuss about vulnerabilities in neural networks that exist and make them susceptible to such attacks, explore on how to defend against said attacks and create modules that could be incorporated in a models structure and make it more robust.

2. BACKGROUND KNOWLEDGE

In 1959, Arthur Samuel, a computer scientist who pioneered the study of artificial intelligence, described machine learning as “the study that gives computers the ability to learn without being explicitly programmed.”
Alan Turing’s seminal paper (Turing, 1950) introduced a benchmark standard for demonstrating machine intelligence, such that a machine has to be intelligent and responsive in a manner that cannot be differentiated from that of a human being.
Machine Learning is an application of artificial intelligence where a computer/machine learns from the past experiences (input data) and makes future predictions. The performance of such a system should be at least human level.

Machine Learning (ML) is one of the most advanced and continuously growing fields of our time. Some implementations of machine learning use data and neural networks in a way that mimics the working of a biological brain.It focus expands on creating prediction models and using statistical algorithms for predicting the behavior of an object, given a subset of the object’s features.

Before diving in to the core part of the thesis, of the attacks and defences in Adversarial ML, we will present some basic principles and structures that we use throughout the chapters.

2.1 Adversarial examples in deep learning

An adversarial example is a sample of input data which has been modified very slightly in a way that is intended to cause a machine learning to misclassify it.
Adversarial examples are thought to be a good aspect of security to work on because they represent a concrete problem in AI safety that can be addressed in the short term, and because fixing them is difficult enough that it requires a serious research effort. Reinforcement learning agents can also be manipulated by adversarial examples, according to new research from UC Berkeley, OpenAI, and Pennsylvania State University.The research shows that widely-used RL algorithms, such as DQN, TRPO, and A3C, are vulnerable to adversarial inputs. These can lead to degraded performance even in the presence of pertubations too subtle to be percieved by a human, causing an agent to move a pong paddle down when it should go up, or interfering with its ability to spot enemies in Seaquest.

Adversarial examples, most of the times, are generated by minimizing the following function with respect to r:

$$\boldsymbol{loss(\hat{f}(x+r), l) + c* \mid r \mid}$$

In this formula, x is an image (represented as a vector of pixels), r is the changes to the pixels to create an adversarial image (x+r produces a new image), l is the desired outcome class, and the parameter c is used to balance the distance between images and the distance between predictions. The first term is the distance between the predicted outcome of the adversarial example and the desired class l, the second term measures the distance between the adversarial example and the original image. This formulation is almost identical to the loss function to generate counterfactual explanations. There are additional constraints for r so that the pixel values remain between 0 and 1.

Adversarial examples are hard to defend against because it is difficult to construct a theoretical model of the adversarial example crafting process. Adversarial examples are solutions to an optimization problem that is non-linear and non-convex for many ML models,

including neural networks. Because we don't have good theoretical tools for describing the solutions to these complicated optimization problems, it is very hard to make any kind of theoretical argument that a defense will rule out a set of adversarial examples.
Adversarial examples are also hard to defend against because they require machine learning models to produce good outputs for every possible input. Most of the time, machine learning models work very well but only work on a very small amount of all the many possible inputs they might encounter.
Every strategy we have tested so far fails because it is not adaptive: it may block one kind of attack, but it leaves another vulnerability open to an attacker who knows about the defense being used. Designing a defense that can protect against a powerful, adaptive attacker is an important research area.

2.2 Data Labeling

In machine learning, data labeling is the process of identifying raw data (images, text files, videos, etc.) and adding one or more meaningful and informative labels to provide context so that a machine learning model can learn from it. For example, labels might indicate whether a photo contains a bird or car, which words were uttered in an audio recording, or if an x-ray contains a tumor. Data labeling is required for a variety of use cases including computer vision, natural language processing, and speech recognition..

2.3 Classification

In machine learning, classification refers to a predictive modeling problem where a class label is predicted for a given example of input data.From a modeling perspective, classification requires a training dataset with many examples of inputs and outputs from which to learn.
A model will use the training dataset and will calculate how to best map examples of input data to specific class labels. As such, the training dataset must be sufficiently representative of the problem and have many examples of each class label.Below we will list some of the classifiers that we will use in the attacks and defences of this thesis:

- **Decision Boundary** is the region of problem space , where the classifier predictions are ambiguous. Decision boundary will play a great role in our attacks as the core of the algorithms is based on adding perturbations strategically near the decision boundary so that we cause misclassification.
- **Binary Classifier** refers to those classification tasks that have two class labels. It uses one single decision boundary to predict the binary class of an input datapoint (0, 1).
- **Multi-Class Classification** refers to those classification tasks that have more than two class labels.Unlike binary classification, multi-class classification does not have the notion of normal and abnormal outcomes. Instead, examples are classified as belonging to one among a range of known classes.
- **Multi-label classification** refers to those classification tasks that have two or more class labels, where one or more class labels may be predicted for each example.

2.4 Regularizers for Single-step Adversarial Training

Regularization is a form of regression, that constrains/ regularizes or shrinks the coefficient estimates towards zero. In other words, this technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting.A simple relation for linear regression looks like this. Here Y represents the learned relation and β represents the coefficient estimates for different variables or predictors(X).The fitting procedure involves a loss function, known as residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function:

$$RSS = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2$$

Regularization, significantly reduces the variance of the model, without substantial increase in its bias. So the tuning parameter λ, used in the regularization techniques described above, controls the impact on bias and variance. As the value of λ rises, it reduces the value of coefficients and thus reducing the variance. Till a point, this increase in λ is beneficial as it is only reducing the variance(hence avoiding overfitting), without loosing any important properties in the data. But after certain value, the model starts loosing important properties, giving rise to bias in the model and thus underfitting. Therefore, the value of λ should be carefully selected.

2.5 Deep Neural Networks

Deep Learning is a sub-field of machine learning in Artificial intelligence (A.I.) that deals with algorithms inspired from the biological structure and functioning of a brain to aid machines with intelligence.At its simplest, a neural network with some level of complexity, usually at least two layers, qualifies as a deep neural network (DNN), or deep net for short. Deep nets process data in complex ways by employing sophisticated math modeling.

A simplified version of Deep Neural Network is represented as a hierarchical (layered) organization of neurons (similar to the neurons in the brain) with connections to other neurons. These neurons pass a message or signal to other neurons based on the received input and form a complex network that learns with some feedback mechanism.Think of each individual node as its own linear regression model, composed of input data, weights, a bias (or threshold), and an output. The formula would look something like this:

$$\boldsymbol{\sum_{i=1}^{m} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias}$$

The general structure of a (deep) neural network is like below:

- **Input weights**: The weights that resemble to each one of the input features, plus one bias weight. These are also called **parameters** of the perceptron.
- **Hidden Layers**: he main computation of a Neural Network takes place in the hidden layers. So, the hidden layer takes all the inputs from the input layer and performs the necessary calculation to generate a result. This result is then forwarded to the output layer so that the user can view the result of the computation

- **Aggregation function**: Is usually a sum function that sums up the information the perceptron has acquired from input.
- **Non Linearity**: A non linear function like sigmoid, ReLU, ELU, etc, which enables the perceptron to detect non linear patterns in features without the use of any input transformation.

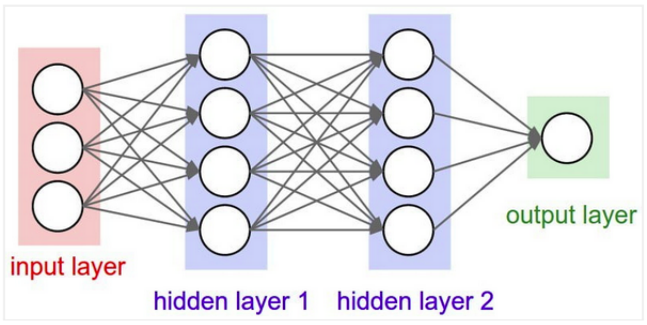


Figure 2.1: Two or more hidden layers comprise a Deep Neural Network

As we have already stated, neural networks can learn non linear patterns with the use of a non linearity transformation.

In our case, we care more about solving classification problems with DNNs, so we will make use of the cross entropy loss function

$$\boldsymbol{J}(\boldsymbol{W}) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log(f(x^{(i)};\boldsymbol{W})) + (1-y^{(i)}) \log((1-f(x^{(i)};\boldsymbol{W}))),$$

and it’s multi-class variation.

2.5.1 Convolutional Neural Networks

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better than a Feed Forward Network.
Usually CNNs are used with media-type inputs (images,video,sound etc.) but can also be used with time series and any other input, from which features could be extracted. At the very end of a convolution and min/max pooling layer sequence, there is often a series of densely connected layers that learn the features and generate the predicted values, or as prefered in our case, predicted classes.
CNNs are defined by 2 basic layers

- **Feature Learning** which includes: Convolution, Pooling layers. Convolutional Filters aim to extract features from the data and might even reduce the feature dimensions, while pooling layers aim to downsample the feature maps after a convolution layer and are used for generalization reasons.
- **Classification** which includes: Flatten, Fully connected layers and Softmax.

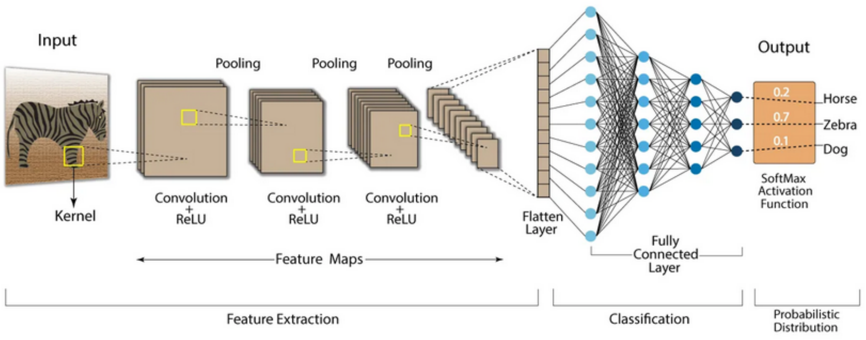


Figure 2.2: Convolutional Neural Network

3. BOUNDARY ATTACK

3.1 Overview

In this chapter, we will go into depth about one of the newest black-box attacks in adversarial machine learning, boundary attack. As mentioned in the introduction, cyber-security threats on Autonomous vehicles (AV) can cause serious safety and security issues as per the “Safety First” industry consortium paper published by twelve industry leaders.The boundary attack is introduced as a category of the decision-based attack [2], which is relevant for the assessment of model robustness. We chose to dive deeper into this particular attack, because as it is referenced by the creators of the FoolBox[3] such decision based attacks (like the boundary attack) are applicable to the real world closed-source,black-box models of autonomous cars,need less knowledge and are easier to apply than transfer-based attacks and are more robust to simple defences than gradient- or score based attacks. Boundary attacks usually require a large set of model queries for obtaining a successful human indistinguishable adversarial example. To improve the efficiency of the boundary attack, it must be combined with a transfer-based attack. The biased boundary attack, significantly reduces the number of model queries with the combination of low-frequency random noise and the gradient from a substitute model. Similar to other transfer-based attacks, a biased boundary attack depends on the transferability between the target model and the substitute model.

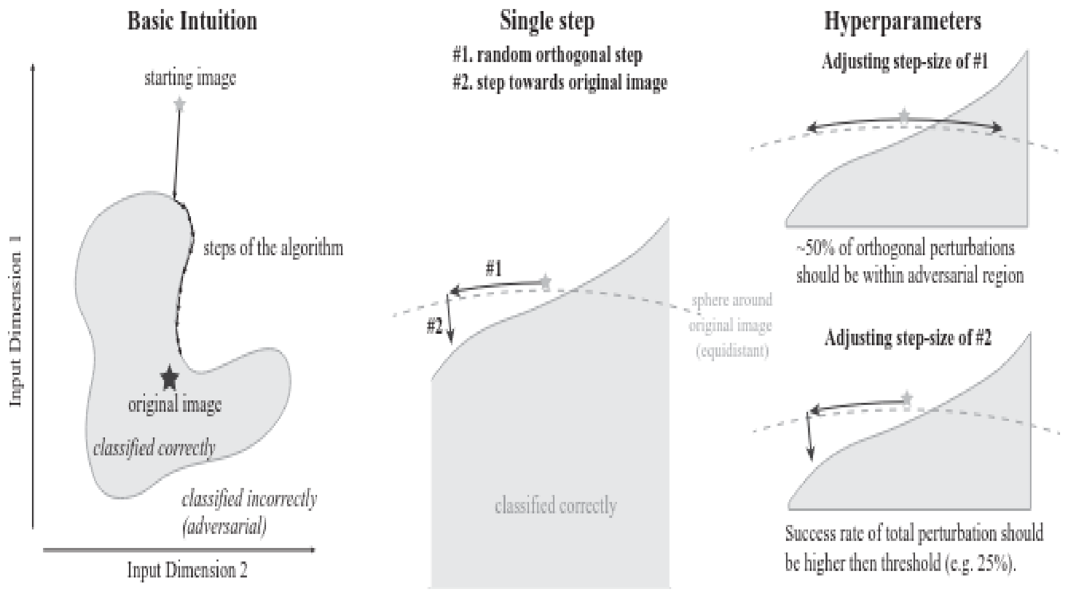

Basic Intuition
starting image
steps of the algorithm
Input Dimension 1
original image
classified correctly
classified incorrectly (adversarial)
Input Dimension 2
Single step
#1. random orthogonal step
#2. step towards original image
#1
#2
sphere around original image (equidistant)
classified correctly
Hyperparameters
Adjusting step-size of #1
~50% of orthogonal perturbations should be within adversarial region
Adjusting step-size of #2
Success rate of total perturbation should be higher then threshold (e.g. 25%).

Figure 3.1: Basic Intuition of Boundary Attack

The basic intuition behind the boundary attack algorithm is depicted in the above figure: the initialization step of the algorithm is based on finding a point that is adversarial and close to the bound of the two pictures under scope and then performs a random walk along the boundary between the adversarial and the non-adversarial region such that (1) it stays in the adversarial region and (2) the distance towards the target image is reduced. In other words we perform rejection sampling with a suitable proposal distribution P to find progressively smaller adversarial perturbations according to a given adversarial criterion c(.). The basic logic of the algorithm is described in Algorithm 1, each individual building block is detailed in the next subsections of this chapter.

3.2 Attack Specifics

3.2.1 Initialization Step

Assuming we start our attack from the starting image, our goal is to add pertrubations/noise to it such as that we start from any sample that is classified by the model as being from the target class. In general, starting points should be close. Intuitively, it makes sense to pick points that are already close to that goal. The optimization procedure would then merely refine them, requiring less iterations to arrive at an adversarial example or produce a better one in the same number of steps. The most straightforward approach is to search a large data set (e.g.ImageNet) for images of the target class and then pick the one with the lowest distance to the original.Starting points should reduce dimensionality. Optimization often suffers from very large search spaces. It is therefore desirable to reduce this search space and to concentrate only on specific dimensions. Notably, Brunner et al. demonstrate that attacks gain efficiency by concentrating only on pixels that differ between the current image and the original, ignoring the rest. A suitable starting point should facilitate this, ideally by not replacing the original entirely but only small regions of it.The attack can then be limited to these regions.

3.2.2 Our proposition for a better initialization step (Hybrid Boundary Attack)

In a perfect scenario, we would have smooth boundaries like the ones pictured in Figures 3.1 and 3.2. However after experimenting with multiple photos and datasets we figured out that bounds are a lot more complex and actually a bound between two pictures can overlap multiple times, meaning that we might have a convex curve type of bound (In order to visualize better the type of bound under scope, you can see Figure 3.2). In this case if we take the first adversarial point found by the algorithm, we will end up with a very time-costly attack, that starts with a significantly big error . So instead of taking the first possible adversarial step as proposed by the basic algorithm (which is a actually the first point where the photos misclassify), we can let our algorithm explore if there are more points in further overlaps of the bound, where we still get an adversarial point but achieve minimal distance from the edge of the target image.

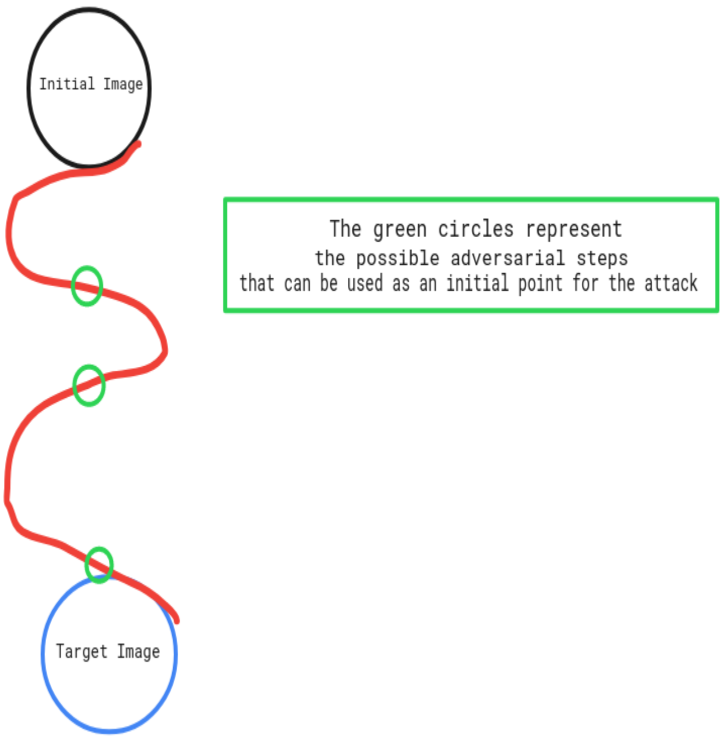

Initial Image
The green circles represent
the possible adversarial steps
that can be used as an initial point for the attack
Target Image

Figure 3.2: Multi-curved bound that was the inspiration behind the above optimization

Here we provide a code snippet of our proposal:

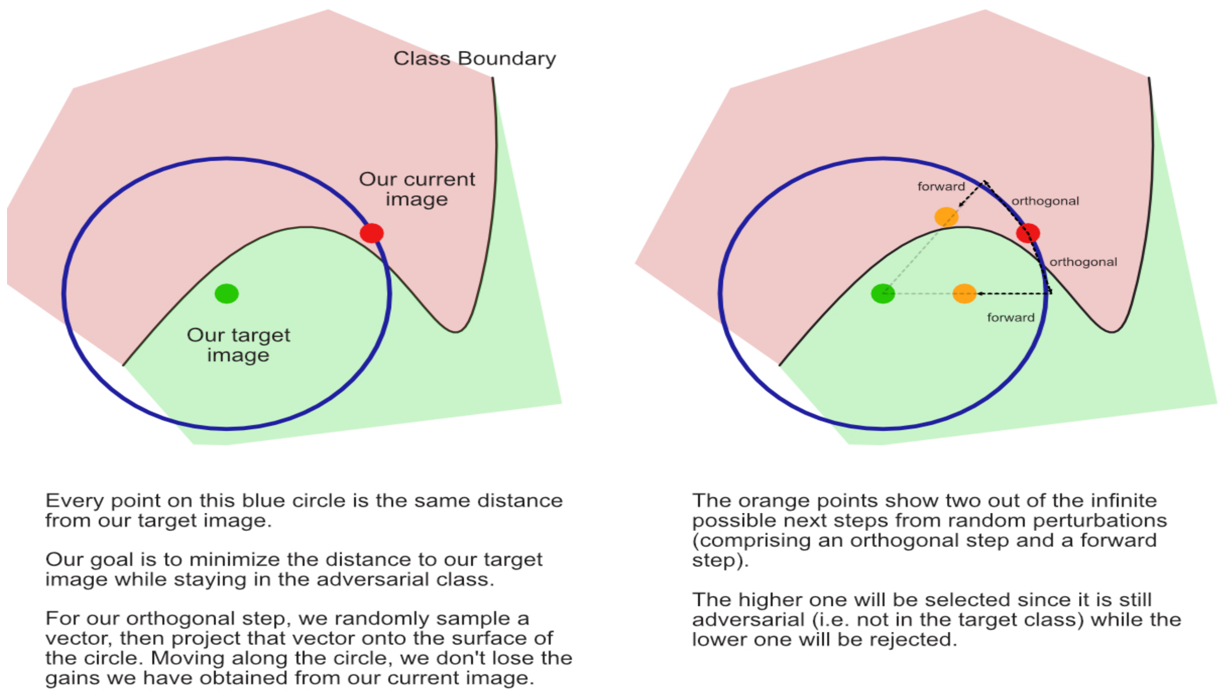
```
    while True:
        trial_sample = self.adversarial_sample +
        ↪  forward_perturbation(self.epsilon, self.adversarial_sample,
        ↪  self.target_sample) - self.prev_dist
        prediction = self.classifier.predict(trial_sample.reshape(1, 224,
        ↪  224, 3))
        self.n_calls += 1
        dist = distance.euclidean(prediction,
        ↪  self.classifier.predict(self.target_sample))

        if self.repetitions == 0:
          self.prev_dist = dist
        if dist < self.prev_dist:
          self.repetitions += 1
          continue
        self.prev_dist = dist
        if self.repetitions > 1000 or self.no_bound_found > 100:
          break
        if np.argmax(prediction) == self.attack_class:
            self.adversarial_sample = trial_sample
            self.boundaries_found += 1
        else:
          print("Didn't Find bound")
          self.epsilon *= 0.9
          self.no_bound_found += 1
          break
        self.repetitions += 1
```

3.2.3 Proposal Distribution

The efficiency of the algorithm crucially depends on the proposal distribution P, i.e. which random directions are explored in each step of the algorithm.It's crucial that whatever distribution we choose, it has to follow some constraints:

- 1) the perturbed sample lies within the input domain, $o_i^{\sim k-1} + h^k, i \in [0, 255]$.
- 2) he perturbation has a relative size of δ,$\| h^k \|_2 = \delta \times d(o, o^{\sim k-1})$.
- 3) the perturbation reduces the distance of the perturbed image towards the original input by a relative amount e, $d(o, o^{\sim k-1}) - d(o, o^{\sim k-1} + h^k) = \epsilon \cdot d(o, o^{\sim k-1})$.

The optimal proposal distribution will generally depend on the domain and / or model to be attacked, but for all vision-related problems tested here a very simple proposal distribution worked surprisingly well. Firstly, we sample from an iid Gaussian distribution $h_i^k \sim N(0, 1)$ and then rescale and clip the sample such that (1) and (2) hold. In a second step we project h^k onto a sphere around the original image o such that $d(o, o^{k-1} + h^k) = d(o, o^{k-1})$ and (1) hold. We denote this as the orthogonal perturbation and use it later for hyperparameter tuning. In the last step we make a small movement towards the original image such that (1) and (3) hold. For high-dimensional inputs and small δ,ε (delta and e step respectively) the constraint (2) will also hold approximately.



Every point on this blue circle is the same distance from our target image.

Our goal is to minimize the distance to our target image while staying in the adversarial class.

For our orthogonal step, we randomly sample a vector, then project that vector onto the surface of the circle. Moving along the circle, we don't lose the gains we have obtained from our current image.

The orange points show two out of the infinite possible next steps from random perturbations (comprising an orthogonal step and a forward step).

The higher one will be selected since it is still adversarial (i.e. not in the target class) while the lower one will be rejected.

Figure 3.3: Description of the orthogonal perturbations

3.2.4 Hyperparameter Tuning

In general, boundary attack only requires two hyperparameters: $\epsilon(e_{step}), \delta(delta_{step})$. While in many implementations of the boundary attack, they leave it up to the user to initialise these two hyperparameters dynamically, we chose to set the values depending on the loops of the code.Specifically, we multiply epsilon with 0.5 each time we apply forward pertrubations at the same point (but only if we haven't surpassed 500 e_steps). As for delta, we calculate the mean score from the predictions and we act accordingly as seen below:

Algorithm 1 Tuning of the Boundary Attack

```
$d\_score = np.mean(predictions == self.attack_class)$
if $d_score > 0.0$ then
    if $d_score < 0.3$ then
        $delta* = 0.9$ $d_score > 0.7$
        $delta/ = 0.9$
        $adversarial\_sample = np.array(trial\_samples)[np.where(predictions == attack\_class)[0][0]]$
        break
    end if
else
    $delta* = 0.9$
```

The adjustment is inspired by Trust Region methods. In essence, we first test whether the orthogonal perturbation is still adversarial. If this is true,then we make a small movement towards the target and test again. The orthogonal step tests whether the step-size is small enough so that we can treat the decision boundary between the adversarial and the non-adversarial region as being approximately linear. If this is the case, then we expect around 50% of the orthogonal perturbations to still be adversarial. If this ratio is much lower, we reduce the step-size δ, if it is close to 50% or higher we increase it. If the orthogonal perturbation is still adversarial we add a small step towards the original input. The maximum size of this step depends on the angle of the decision boundary in the local neighbourhood. If the success rate is too small we decrease ϵ, if it is too large we increase it. Typically, the closer we get to the original image, the flatter the decision boundary becomes and the smaller ϵ has to be to still make progress. The attack is converged whenever ϵ converges to zero.

3.2.5 Adversarial Criterion

A typical criterion by which an input is classified as adversarial is **misclassification**, i.e. whether the model assigns the perturbed input to some class different from the class label of the original input. Another common choice is **targeted misclassification** for which the perturbed input has to be classified in a given target class. Other choices include top-k misclassification (the top-k classes predicted for the perturbed input do not contain the original class label) or thresholds on certain confidence scores. Outside of computer vision many other choices exist such as criteria on the word- error rates. In comparison to most other attacks, the Boundary Attack is extremely flexible with regards to the adversarial criterion. It basically allows any criterion (including non-differentiable ones) as long as for that criterion an initial adversarial can be found (which is trivial in most cases)

Algorithm 2 Minimal version of the Boundary Attack

```
Ensure: Data: original image o, adversarial criterion c(.), decision of model d(.)
Ensure: Result: adversarial example $\sim$ o such that the distance $d(o, o^{\sim}) = \parallel o - o^{\sim} \parallel_2^2$ is
  minimized initialization: $k = 0, o^{\sim 0} \sim U(0,1) s.t. o^{\sim 0}$ is adversarial;
  while k < maximum number of steps do
      draw random perturbation from proposal distribution $h_k \sim P(o^{\sim k-1});$
      if $o^{\sim k-1} + h_k$ is adversarial then
          $o^{\sim k} = o^{\sim k-1} + h_k;$
      else
          $o^{\sim k} = o^{\sim k-1}$
      end if
      $k = k + 1$
  end while=0
```

However as mentioned in the overview, we can get much better results with only a few alterations in the algorithm, by creating a Fast Adaptive Boundary Attack [6]. So the new algorithm will be like below:

Algorithm 3 Fast Adaptive Boundary Attack

```
Ensure: Input : x_orig original point, c original class,N_restarts, N_iter, α_max, β, η, ε, p
Ensure: Output :x_out adversarial example
  $u \leftarrow$ inf
  for $j = 1$, …, N_restarts do
      if $j = 1$ then
          $x(0) \leftarrow x_{orig}$;
      else
          $x(0) \leftarrow$ randomly sampled s.th. $\| x(0) - x_{orig} \|_p = minu, e/2$;
      end if
      for $i = 0, \dots, N_{iter} - 1$ do
          $s \leftarrow argmin_{l \neq c} \frac{|f_l(x^{(i)}) - f_c(x^{(i)})|}{\|\nabla f_l(x^{(i)}) - \nabla f_c(x^{(i)})\|_q}$
          $\delta(i) \leftarrow proj_p(x^{(i)}, {}_s, C)$
          $\delta(i)_{orig} \leftarrow proj_p(x^{(i)_{orig}}, \pi_s, C)$
          $a = min\{\frac{\|\delta^i\|_p}{\|\delta^i\|_p + \|\delta^i_{orig}\|_p}\} \in [0, 1]$
          $x(i+1) \leftarrow proj_C((1-)(x(i) + h\delta^{(i)}) + (x_{orig} + h\delta(i)_{orig}))$
          if $x(i+1)$ is not classified as c then
              if $\|$x$^{(i+1)} - x_{orig} \|_p < u$ then
                  $x_{out} \leftarrow x^{(i+1)}$
                  $u \leftarrow \| x^{(i+1)} - x_{orig} \|_p$
              end if
              $x^{(i+1)} \leftarrow (1-)x_{orig} + x^{(i+1)}$
          end if
      end for
  end for
```

3.3 Conclusion

Concluding this chapter, of the Boundary Attack method, we dove into the functionality of this attack and how it is applicable in the real world black box models,the autonomous cars. We described fairly every step of the algorithm, through visual properties and simplified mathematical background.Further more we suggested another available alteration of the boundary attack, the Fast Adaptive Boundary Attack which we also analyzed above. Also we paid attention to the dynamic hyperparameter tuning of the variables δ, ϵ that play a big role on the performance of the attack.

Also a significant point in this chapter was our proposal for a hybrid version of the boundary attack. We showcased that bounds in real life objects aren't always as smooth as the ones described on the paper of Brendel[3]. As a result we proposed a method where we choose an even closer adversarial point on the bound between the initial and target image. This attack will exploit the same vulnerabilities as the original attack, but converge faster than the original attack (about 2.5 hours less in execution time)

4. FAST GRADIENT DESCEND ATTACK

4.1 Overview

In this section, we will get into details about the Fast Gradient Sign Method (FGSM)[20][10] to create adversarial images.

The FGSM exploits the gradients of a neural network to build an adversarial image, similar to what we've done in the untargeted adversarial attack and targeted adversarial attack tutorials. Essentially, FGSM computes the gradients of a loss function (e.g., mean-squared error (MSE) or categorical cross-entropy) with respect to the input image and then uses the sign of the gradients to create a new image (i.e., the adversarial image) that maximizes the loss. The result is an output image that, according to the human eye, looks identical to the original, but makes the neural network make an incorrect prediction!

The fast gradient sign method works by using the gradients of the neural network to create an adversarial example. For an input image, the method uses the gradients of the loss with respect to the input image to create a new image that maximises the loss. This new image is called the adversarial image. This can be summarised using the following expression:

$$adv\_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

where:

- adv_x : Adversarial image.
- x : Original input image.
- y : Original input label.
- ϵ : Multiplier to ensure the perturbations are small.
- θ : Model parameters.
- J : Loss.

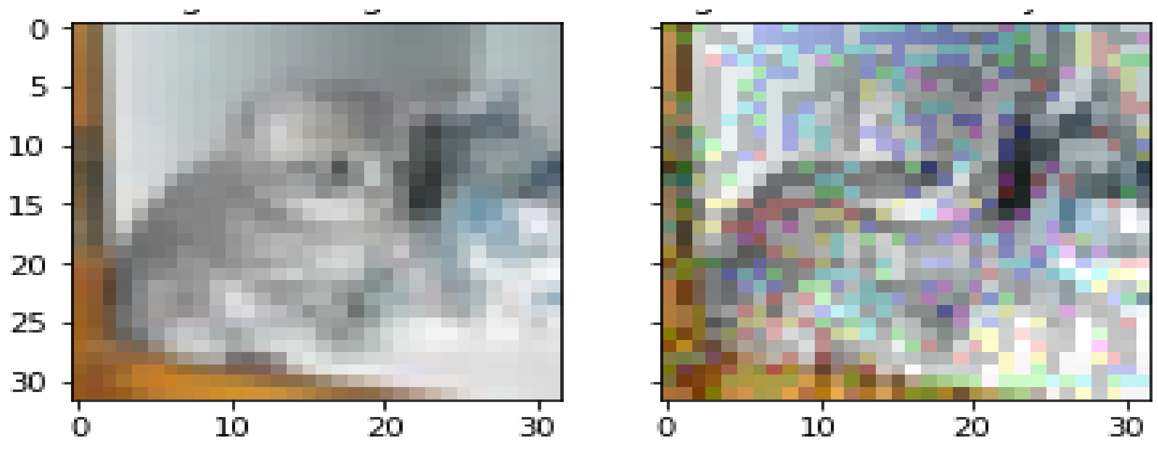


Figure 4.1: FGSM attack using CNN model on CIFAR-100.

An intriguing property here, is the fact that the gradients are taken with respect to the input image. The objective is to create an image that maximizes the loss. A method to accomplish that is to find how much each pixel in the image contributes to the loss value, and add a perturbation accordingly. This strategy works in fast paces 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.

4.2 Attack Specifics

4.2.1 Adversarial Step

Since FGSM is one of the best attacks taking into account the minimal time needed to execute due to the fact that the gradients are taken from the input image and not from the train model process, it only made sense that we used the optimal packages of python to speedup the process even more. So for that we used the **tensorflow.keras**. The FGSM is fast because it makes a linear approximation of a deep model and solves the maximization problem analytically, in closed form. Despite this approximation, it is still able to reliably fool many classifiers for computer vision problems, because deep models often learn piece-wise linear functions with surprisingly large pieces. The basic function of the attack is the point where we create the adversarial image using the FGSM attack. The code for that is attached below:

```python
def generate_adversaries(model, baseImage, label, epochs=50):

  for step in range(0, epochs):
    baseImage = tf.cast(baseImage, tf.float32)
    delta = tf.Variable(tf.zeros_like(baseImage), trainable=True)

    with tf.GradientTape() as tape:
      tape.watch(baseImage)
      adversary = baseImage + delta
      pred = model(adversary)
      loss = tf.keras.losses.MSE(label, pred)

    gradients = tape.gradient(loss, baseImage)
    delta.assign_add(clip_eps(delta, eps = EPS))
    optimizer.apply_gradients([(gradients,delta)])
    signed_grad = tf.sign(gradients)

  return signed_grad
```

4.2.2 Hyperparameter Tuning

Some notes that we took into account, while finetuning our model/attack were the below:

- FGSM step size should be small,so that the adversarial examples don't cluster near the boundary
- Preferably when initialising the CNN model we use a maxpooling layer after every convolutional layer which helps with the dimension reduction and the better fit of the training data.
- The training process should be implemented for a small amount of epochs
- fine tuning should be done with dropout rates of 0.3 and more, as anything less gave a noticeable higher training loss

After many experiments, $8-10$ epochs were the optimal number of epochs for the training process. However by adding early-stopping to the training process we can avoid any pitfalls because of overfitting due to the long training.

4.2.3 Our proposition for general Optimizations

According to the keras documentation, most of the times we use pre-trained model for the FGSM attack, like ResNet,MobileNetV2 etc. However, after some experimenting we figured out that if we exchange the pre-trained model for a custom cnn model, we get much higher accuracy and a noticeable small training loss. So we created a basic CNN model of four convolutional layers (conv2d), 3 maxpool layers (MaxPooling2D) which helps with the dimension reduction of our model ,dropout of rate 0.3 and finally a layer of softmax activation function. Adding this alteration to our attack gave us the following statistics:

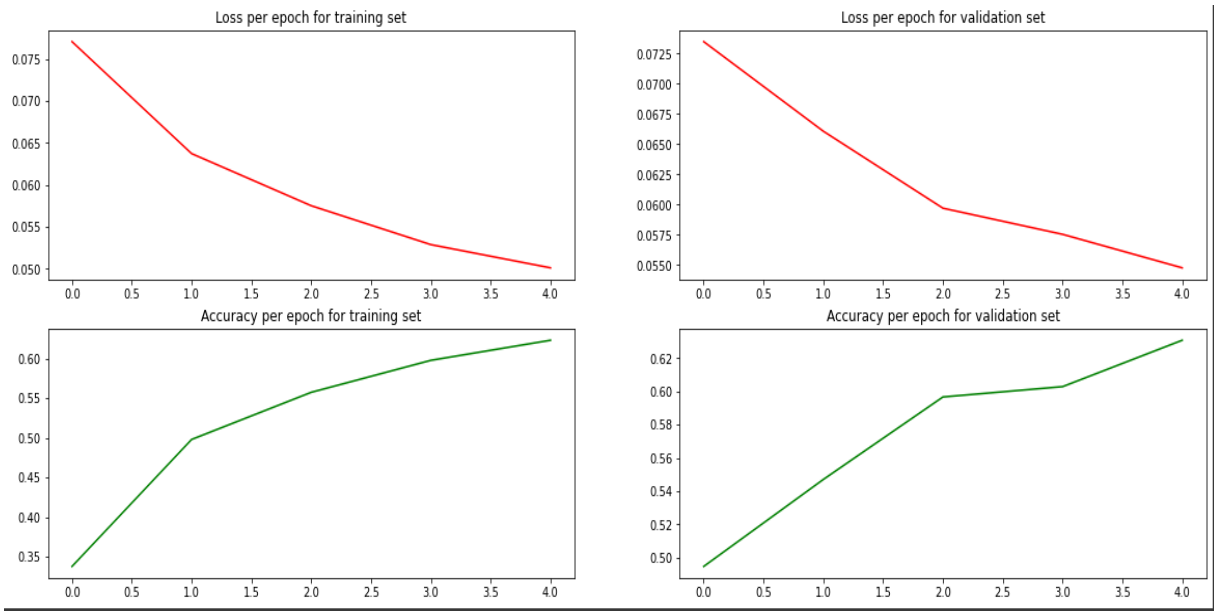


Figure 4.2: Statistics of FGSM attack using a custom made CNN model.

4.3 Conclusion

The earliest and simplest method to generate adversarial examples is the Fast Gradient Sign Method (FGSM). This non-iterative method generates examples in one step and leads to robust adversaries. However due to it's simplicity and the over-complexity of nowadays neural networks it has stopped been actively used since 2000's.
But how effective is this attack in tricking the network? Recall that we want to produce images which cause a prediction of a wrong class at a high confidence. For FGSM, a stronger perturbation does not necessarily lead to more confident or successful adversaries.
In theory, increasing the magnitude of the perturbation should lower the confidence of the model in predicting the correct class. However, after experimenting on the adversarial images that we produced, we noticed some exceptions. Interestingly, the confidence in the original correct class of many images first drops quickly, then increases again with increasing adversarial perturbations. In some occasions, the model's confidence even reaches back to clean levels at the highest levels of perturbance.

5. DEEP-FOOL ATTACK

5.1 Overview

So far we've seen different kinds of adversarial attacks, that although they differ in difficulty (for example boundary attack has far more complec calculations as we saw that FGSM) however they all had the same goal: exploit DNNs, which as we saw are susceptible to small perturbations that are made in a meaningful way, which is not random noise.One more thing to remember is that these models don't look at the shape of an object so much as look at the texture of an object. This is one of the major reasons these models fail due to small perturbations in the image [7].

At this point we came across the paper DeepFool: a simple and accurate method to fool deep neural networks [14] which introduces an efficient method to compute the minimal perturbations necessary to cause a classifier to misclassify an image, which they call DeepFool. DeepFool is a state-of-the-art ,untargeted white box attack method for efficiently crafting adversarial examples, which is based on pushing an input to its closest decision boundary, approximated in the vicinity of the input according to an efficient linear approximation. A detailed explanation of this attack algorithm can be found in the next sections of this chapter.DeepFool is also one of the few adversarial attacks that evolved and is now used in the audio domain [16].

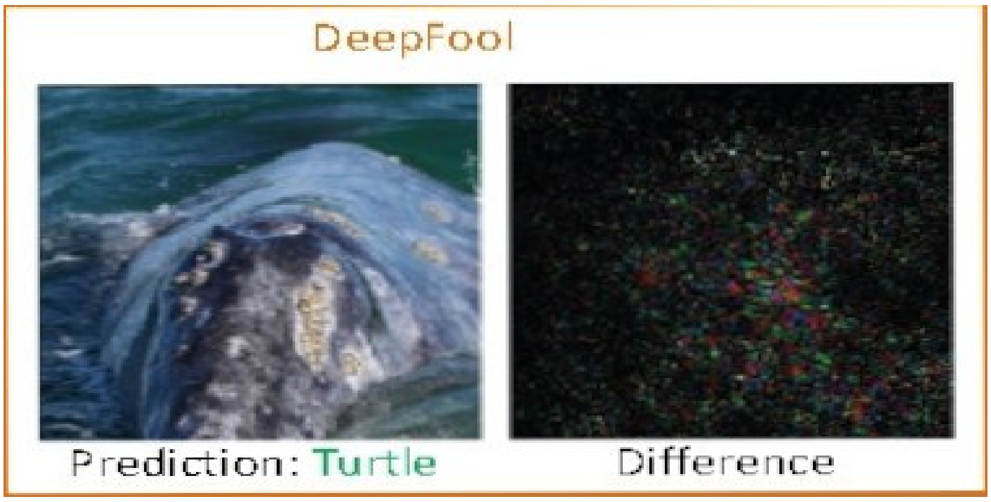


Figure 5.1: Example of DeepFool attack where whale gets misclassified as turtle.

5.2 Attack Specifics

5.2.1 DeepFool for binary classifiers

Firstly the first and most basic version of DeepFool is the one that uses a binary classifier. In order to obtain the minimal pertrubation in this case, so that the classifier of f misclassifies the input image of x, the pertrubation of $r * (x)$ must project the input image of x orthogonal to the hyperplane of the binary classifier. However misclassification is not guaranteed since models are non-linear in nature (the binary classifier deepfool was created with the idea that models are linear). To alleviate this issue, the algorithm works iteratively and adds the previous perturbation to the next perturbation, which is performed until the label changes or max iterations are reached. Additionally, convergence could end up close to zero, so a parameter called overshoot η is used as a scalar $(1 + h)$ for the perturbation of $r * (x)$, so it would go past zero and make the class label change.
Below is the full pseudocode for the binary - classifier DeepFool attack:

Algorithm 4 DeepFool for binary classifiers

```
Ensure: Input: Image x, classifier f.
Ensure: Output: Pertrubation $\hat{r}$
1: Initialize $x_0 \leftarrow x, i \leftarrow 0.$
2: while $sign(f(x_i)) = sign(f(x_0))$ do
3:     $r_i \leftarrow -\frac{f(x_i)}{\|\nabla f(x_i)\|_2^2}\nabla f(x_i)$
4:     $x_{i+1} \leftarrow x_i + r_i$
5:     $i \leftarrow i + 1$
6:     if $sign(f(x_i)) \neq sign(f(x_0))$ then return $\hat{r} = \sum_i r_i$
7:
```

However, for high-dimensional non-linear decision spaces, which is the general case of DNNs, estimating the decision boundaries is a complex task, which makes this optimization intractable in practice. Due to this limitation, the DeepFool algorithm provides a strategy to approximate $r*$ by efficiently approximating the decision region of the model.

5.2.2 DeepFool for multiclass cases

In general DeepFool algorithm for a multi-class classifier can be considered multiple binary classifiers. The decision boundary as shown below can be considered a polyhedron, which is made from the intersection of multiple hyperplanes. The minimum perturbation needed would be from the closest hyperplane to x_0. Given there are multiple classes, the loss and backpropagation would need to be computed for each class label allowed by the function. When finding the minimum perturbation, the difference must be taken between the computed values for each label and the computed values for the label of the original prediction.

The only difference to the binary case is that it requires the differences of the classifier output and the gradients of the outputs, as well as taking the absolute value of the model output.
Below is the full pseudocode for the multiclass DeepFool attack:

In order to visualize better the geometry behind the different models and how differently each classifier approaches the problem, we can use the image below where on the left we

Algorithm 5 DeepFool for multiclass cases

```
Ensure: Input: Image x, classifier f.
Ensure: Output: Pertrubation $\hat{r}$
  Initialize $x_0 \leftarrow x, i \leftarrow 0.$
  while $\hat{k}(x_i) = \hat{k}(x_0)$ do
      for $k \neq \hat{k}(x_0)$ do
          $w'_k \leftarrow \nabla f_k(x_i) - \nabla f_{\hat{k}(x_0)}(x_i)$
          $f'_k \leftarrow f_k(x_i) - f_{\hat{k}(x_0)}(x_i)$
      end for
      $\hat{l} \leftarrow argmin_{k \neq \hat{k}}(x_0) \frac{|f'_k|}{\|w'_k\|_2}$
      $r_i \leftarrow \frac{|f'_{\hat{l}}|}{\|w'_{\hat{l}}\|_2^2} w'_{\hat{l}}$
      $x_{i+1} \leftarrow x_i + r_i$
      $i \leftarrow i + 1$
      if $\hat{k}(x_i) \neq \hat{k}(x_0)$ then return $\hat{r} = \sum_i r_i$
```

have an adversarial example for binary classifier, while on the right we have an adversarial example for multiclass classifier.

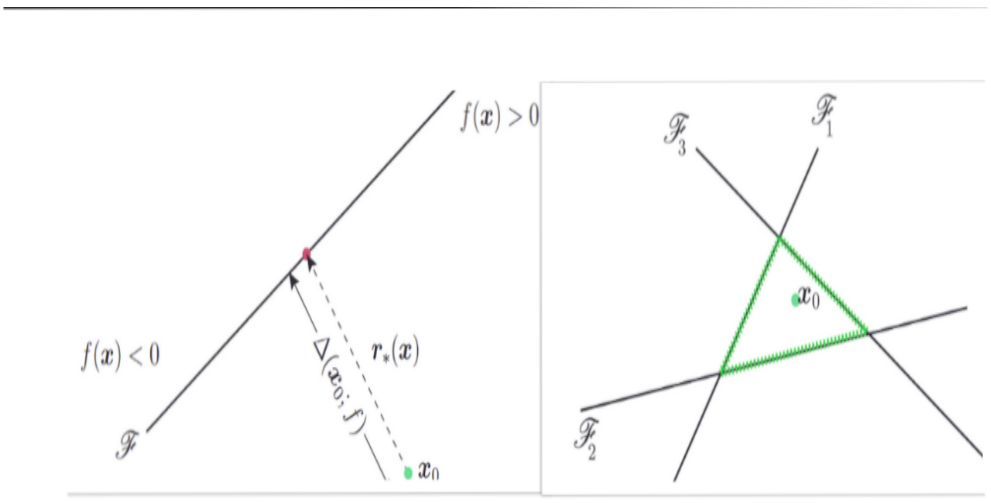


Figure 5.2: On the left:Adversarial Example for Binary Classifier. On the right:Adversarial Example for Multi-Class Classifier

5.2.3 Proposition for optimized search of adversarial bound

DeepFool relies on the assumption that we can accurately approximate the boundaries of the decision region of the classifier in the proximity of a given input using a linear approach. Based on this assumption, at each step, the input is moved in a greedy way towards the (estimated) closest boundary, using the perturbation r_i. However, there could exist alternative perturbations that, with the same amount of distortion employed by DeepFool, can reach a different decision region, or even reach it with less distortion. Being able to find such shortcuts in an efficient way can increase the effectiveness of the attack, or reduce the required amount of perturbation to fool the model. So if we use any search algorithm [17] known and test the perturbations projected in the sphere of radius $\| r_i \|_2$ and centered at the current point x_i, we might find a better result that is capable of changing the output of the model, with no additional regard for the incorrect class that is produced. Thus, the proposed methodologies are particularly well suited for untargeted attacks, as in targeted attacks we focus on reaching a particular output rather than producing any possible incorrect class.

Also like we saw in the optimizations of FGSM, many programmers prefet to use pretrained models like resnet34 etc. However here as well we noticed that if we create a custom made CNN model like the one described in the FGSM optimizations we get higher accuracy and smaller training loss than with a pretrained model as we can tune the cnn class input, filter sizes, dropout rates to the input databases needs.

5.2.4 Hyperparameter Tuning

Due to the fact that the attacks described are similar structure wise (but differ greatly in the math background - perturbations generating) the hyperparameters that we can tune and actually see better results are some of the below:

- dropout rate ≥ 0.3
- use of SGD optimizer if model used is CNN
- early stopping to avoid overfit
- increase of number of layers and associated sizes to reduce high bias

5.3 Conclusion

In this chapter we explored the background of DeepFool attack. In simple words, The DeepFool algorithm searches for an adversary with the smallest possible perturbation, finds the projected distance to the closest decision boundary and iteratively changes the adversary until the class has changed. The results are less perceptible adversaries.
Specifically, the algorithm imagines the classifier's decision space being divided by linear hyperplane boundaries that divide the decision to select different classes. It then tries to shift the image's decision space location directly towards the closest decision boundary. However, the decision boundaries are often non-linear, so the algorithm completes the perturbation iteratively until it passes a decision boundary.

The results show that DeepFool creates adversarial images with the lowest perturbation, but requires more computation time when compared to FGSM as the second is able to create adversarial images fast, but it has more perturbations than DeepFool. DeepFool requires more computation time due to its iterative process in order to find the minimal perturbation.

Furthermore, we commented on the inherited problems that DeepFool faced at its early stages where although the attack did have a high performance when using binary classifiers, it had to evolve by using multiclass classifiers in oder to cover the variety of models. That's why we introduced the algorithm of DeepFool for multiclass case.
In this chapter like in the ones that came before, we proposed various optimizations that we explored during the creation of this thesis.Except from using a CNN model instead of pretrained ones like resnet, we also introduced[17] the option of creating a hybrid version of deepfool that uses local or global search to find the next optimal perturbation.

6. CARLINI WAGNER ATTACK

6.1 Overview

In this section we are going to study the last adversarial attack of this thesis: The Carlini & Wagner attack which is currently one of the best known algorithms to generate adversarial examples and it was published in IEEE S&P 2017 [4]. In order to understand the different additions that Carlini and Wagner brought to adversarial attacks we have to do a quick recap of basic definitions. For starters, adversarial examples as described in Intriguing properties of neural networks [15], the optimization problem to craft adversarial examples was formulated as:

$$minimize : D(x, x + \delta)$$
$$such\ that : (1)C(x + \delta) = t\ and\ (2)\ x + \delta \in [0, 1]^n$$

where:

- x= input image
- δ = perturbations
- D = distance metric between the adversarial and real image
- C = Classifier function
- n= dimensions
- t = target class
- Constraint 1 makes sure that the image is indeed misclassified
- constraint 2 makes sure that the adversarial image is valid i.e. it lies within the normalized dimensions of x.

Traditionally well known way to solve this optimization problem is to define an objective function (generally but not necessarily a loss function that penalizes as we move further from satisfying the constraints) and to perform gradient descent on it; which guides us to a optimal point in the function. However, The formulation above is difficult to solve because $C(x + \delta) = t$ is highly non-linear (the classifier is not a straight forward linear function).

However Carlini and Wagner proposed that the first constraint ($C(x + \delta) = t$) is altered in order to satisfy another condition as well. Specifically if $C(x + \delta) = t$ is satisfied then $f(x + \delta) \leq 0$ is also satisfied, where f is the objective function. In the next section we are going to look into the attack specifics as well as the optimized choice for the objective function as proposed by the creators.

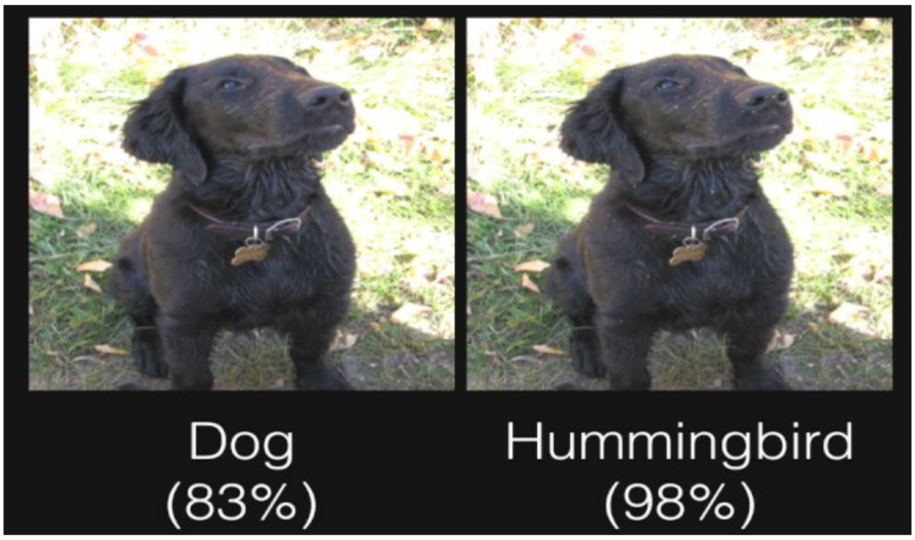


Figure 6.1: Result of the carlini wagner attack showcased at the 38th IEEE Symposiumon Security and Privacy

6.2 Attack Specifics

6.2.1 Basic Algorithm Specifications

First and foremost we have to look into the choice of objective function, as proposed by the creators Carlini and Wagner.Conceptually, the objective function tells us *how close we are getting to being classified as t*. The most basic definition for an objective function would be $f(x+\delta) = [1-C[x+\delta]_T]$ where $C[x+\delta]_T$ is the probability of $x+\delta$ being classified as t. If the probability is low, then the value of f is closer to 1 whereas when it is classified as t, is equal to 0. However such a simplistic function won't be able to cope with high, multiclass, non-linear models.

For this reason Carlini and Wagner in their paper [4] tested a list of 7 different objective functions:

- $f_1(x') = -loss_{F,t}(x') + 1$
- $f_2(x') = max_{i\neq t}(F(x')_i) - F(x')_t)^+$
- $f_3(x') = softplus(max_{i\neq t}(F(x')_i) - F(x')_t) - log(2)$
- $f_4(x') = (0.5 - F(x')_t^+$
- $f_5(x') = -log(2F(x')_t - 2)$
- $f_6(x') = max_{i\neq t}(Z(x')_i) - Z(x')_t)^+$
- $f_7(x') = softplus(max_{i\neq t}(Z(x')_i) - Z(x')_t) - log(2)$

where s is the correct classification, (e)+ is short-hand for $max(e, 0)$, $softplus(x) = log(1+exp(x))$, $loss_{F,s}(x)$ is the cross entropy loss for x, Z(x') is the logit (the unnormalized raw probability predictions of the model for each class / a vector of probabilities) when the input is an adversarial.

The best results were obtained when using the $f_6(x')$

So now the original problem reformulates as:

$$minimize : D(x, x + \delta) + c * f(x + \delta)$$
$$such\ that : x + \delta \in [0, 1]^n$$

Notice that we have adjusted some of the above formula by adding a constant ($c > 0$); we have done this only so that the function respects our definition. This does not impact the final result,as it just scales the minimization function.

Through experiments we discovered that the best constant is often found lying between 1 or 2.

Finally they proposed that the constraint of the original formula:

$$x + \delta \in [0, 1]^n$$

, is altered. Specifically we have to apply the method *change of variable* so that we optimize over w instead of δ.

The final reformulation of the attack will now be:

$$minimize : D(\frac{1}{2}(tanh(w) + 1), x) + c * f(x + \frac{1}{2}(tanh(w) + 1))$$
$$such\ that : tanh(w) \in [-1, 1]$$
$$where\ f(x') = max(max\{Z(x')_i : i \neq t\} - Z(x')_t, -k)$$

The CW attack is the solution to the optimization problem (optimized over) given above using Adam optimizer. To avoid gradient descent getting stuck, we use multiple starting point gradient descent in the solver.

6.2.2 Hyperparameter Tuning and model optimizations

As mentioned before, Carlini and Wagner attack is currently the best adversarial attack out there since it has the highest performance to most datasets. Beacuase of the simplicity of the attack and its great results it's difficult to find many optimizations. However some small tweaks that seemed to have some small effect on the attacks accuracy and performance are the below:

- A small modification can be made to launch non-targeted attacks:

$$min \parallel \delta \parallel_p^2 \ s.t. \ g(x+\delta) \neq y \& x+\delta \in X$$

- apply the method *change of variable* so that we optimize over w instead of δ.Specifically w is given by:

$$\delta = \frac{1}{2} * (tanh(w)+1) - x$$

- instead of using VGG16 for the model that will be used for the attack, we will substitute it with a cnn model or even better a recurrent cnn (or even a fast recurrent cnn model[8]). Specifically any type of cnn model, as they have been dominant model types for image recognition but are less robust than transformers. Let's note here that the more robust a model is, we have a much lower probability of the attack being effective [19].
- most of the times the most effective value for the constant using in the algorithm is in the interval: $[1, 2]$
- after experimenting with metric functions such as L0, L2 and Linf we concluded that L2 has the best outcome to find the optimal adversarial point closer to the bound of the clean image and the targete image.

6.3 Conclusion

Summarizing what was presented in the Carlini-Wagner chapter, we presented and analyzed all aspects of the algorithm structure as well as different metrics and experiments that were shocased by the authors themseves. In brief words, the Carlini-Wagner attack is a regularization-based attack with some critical modifications which can resolve the unboundedness issue.

The L2 attack version of CW (which after experiments understood that is the most effective out of the three),uses a logits-based objective function which is different from all existing L2 attacks, and avoids the box constraint by changing variables.

After evaluating the Carlini and Wagner attack we find it often produces adversarial examples are on average within 11.6% of the minimally-distorted example when using the L_{inf} norm,and within 5.1% of the minimally-distorted example when using L2 (we consider here just the terminated experiments,to draw a meaningful conclusion). In particular,iterative attacks perform substantially better than single-step methods, such as the fast gradient method. This is an expected result and is not surprising: the fast gradient method was designed to show the linearity of neural networks, not to produce high-quality adversarial examples.

7. EVALUATION OF ATTACKS

7.1 Overview

Reaching the end of the attack section of this thesis, it's time to showcase our statistics and evaluate the methods that we used.At this point the experiments conducted where almost identical from one attack process to another. Specifically, we conducted experiments using the same datasets in order to test accuracy, and similar neural networks to train our models. For example, for the FGSM, Carlini Wagner and Deepfool we experimented with ResNet and VGG16, however we ended up creating a custom-simple cnn model which proved to work far better than the pretrained models for our dataset samples.Furthermore, since our focus is on the attack function itself, its important that we keep the architecture of the models similar so that we can evaluate them on similar terms at the end. Before jumping in to statistics and further info, we are going to briefly present the datasets that we used for our experiments.

7.2 Datasets

7.2.1 MNIST

The MNIST[1] database of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.
It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28×28 field.
With some classification methods (particuarly template-based methods, such as SVM and K-nearest neighbors), the error rate improves when the digits are centered by bounding box rather than center of mass.

7.2.2 CIFAR-10

The CIFAR-10 dataset[2] consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. In order to set the same slice of classes for our experiments , we used a list of specific labels:
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

[1]https://yann.lecun.com/exdb/mnist/
[2]https://www.cs.toronto.edu/ kriz/cifar.html

7.2.3 CIFAR-100

This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a ”fine” label (the class to which it belongs) and a ”coarse” label (the superclass to which it belongs).

7.3 Metrics, Optimizers, Hyperparameter Setup and Early Stopping Class

Since we used the cnn model in DeepFool, FGSM and Carlini Wagner (we used ResNet for the boundary attack mostly because boundary attack, due to the fact that its a blackbox attack, takes much more time to finish),we tuned it with the same hyperparameters (which seemed to work for all three attacks). So in more detail we tuned the models as followed:

- Optimizers selected: torch.optim.Adagrad or torch.optim.Adam (both worked but torch.optim.Adagrad) seemed to perform better
- criterion: SparseCategoricalCrossentropy()
- dropout rate = 0.3
- used one maxpooling layer for each Con2d layer we added to the models state
- used tf.keras.losses.MSE() to calculate the loss of the gradients of each loop from the adversarial generator step
- batches in training: 128
- epochs: 10
- applied earlystopping class in order to avoid overfitting of our model (and thus making it less robust)

As mentioned above, earlystopping is a technique which, as the name suggests, stops the models training usually based on a given threshold, checked at the end of every epoch. Most common threshold is the difference of the validation error/score between epochs, given the validation set is present (more possible in whitebox attacks given that we have access to a validation set). Earlystopping is one of the most famous and effective ways of avoiding poor generalized models, since it monitors the progress of the training and when the latter is starting to fade out, stops the training, since there is no reason to continue iterations. Although earlystopping was created as a concept to help avoid over-fitting of models, it’s easy to understand that by eliminating different inherited vulnerabilities of the model and making it more robust, we decrease the probabilities of the respective attacks being effective. Below, we present you with a code-snippet of the custom Early Stopping class that we created for the needs of our attack models:

```python
class EarlyStopping:
    def __init__(self, patience=7, delta=0.05):
        """
        Args:
            patience: How long to wait after last time validation loss
    ↪   improved.
            delta : Minimum change in the monitored quantity to qualify as an
    ↪   improvement.
        """
        self.patience = patience
        self.counter = 0
        self.best_score = None
        self.early_stop = False
        self.val_loss_min = np.Inf
        self.delta = delta
    def __call__(self, val_loss, model):
        score = -val_loss

        if self.best_score is None:
            self.best_score = score
        elif score < self.best_score + self.delta:
            self.counter += 1
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_score = score
            self.counter = 0

        return self.early_stop
```

7.4 Attacks Evaluation

7.4.1 Boundary Attack

Boundary attack was the only black-box attack that we implemented for this thesis. Although one would expect that whitebox attacks would have significantly better results and efficiency from the blackbox one, boundary attack statistics prove this statement wrong. If we could mention one disadvantage for this attack strategy, that would be the time duration. Although the rest of the attacks (depending on the dataset that they get trained on) have a low time expectancy (max 30 minutes if dataset complicated, or cnn is trained with more than 10 epochs), boundary attack needs almost 8 hours to finish executing. However, if we compare the output photos per 10 steps of the algorithm, we will see that we have a very truthful looking result in about 1.30-2 hours. It is understandable though to have such high execution times, as gradient calculations are time consuming, and the fact that we dont know the model that we attack nor the database makes things even more difficult.

In our implementation we added a feature of image saving to a newly created folder of google drive where we save image instances during the attack execution, so that we can visualize the evolution of the attack, how much noise/perturbations are added in each loop, and when do we have convergence.

In the technical aspect of the boundary attack, we used ResNet models and executed the boundary attack version with our optimization per starting adversarial point.Also for the experimental that we will showcase, we used the MNIST database. So for this experiment we used as initial image an instance of the number 2 from the MNIST dataset and as target an instance of number 4.It;s worth mentioning that for quicker convergence we could have chosen numbers that are easier to look like eachother after adding some perturbations (like 3 and 8).

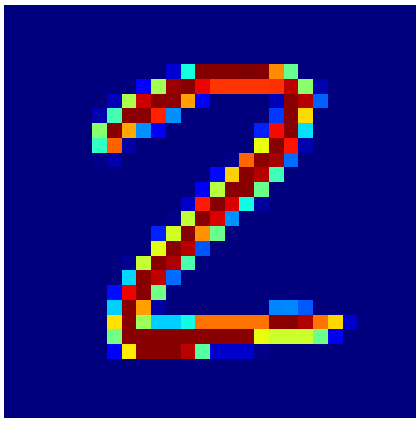


Figure 7.1: Initial Image Labeled as: MNIST_2

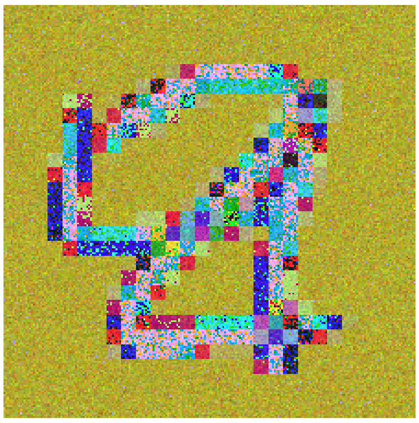


Figure 7.2: step 800

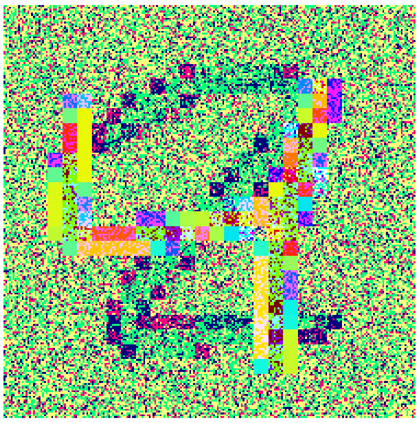


Figure 7.3: step 2300

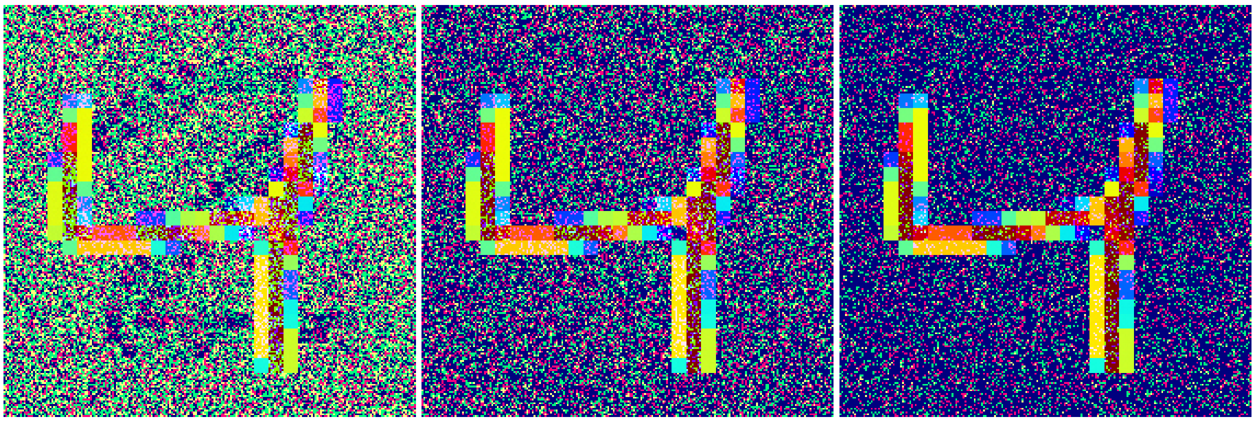


Figure 7.4: step 4.150 **Figure 7.5: step 7.500** **Figure 7.6: step 10000**

Figure 7.7: Adversarial Images, in different steps during the attack,Labeled as: MNIST_2

As we can see from the short sample of images that we present here, the misclassification worked already from the first steps of the algorithm (which is logical taking into account that the model traces the target picture with noise from the initial photo.) The last picture is a clear image of the MNIST instance of 4 but we can still detect some slight noise/perturbations in the background of the photo. If we let it reach to an even lower loss (about 2 more hours of execution), we would have results that fooled the naked eye completely.

We tested the attack on images from both CIFAR-10 and MNIST. In both cases the attack succeeded in the same time frame impressive results. The MSE loss for MNIST was a little higher in all occasions but the difference was not that big between the two datasets nor important as visually we got similar-quality results.

Table 7.1: Boundary Attack MSE loss scores depending on epochs of model

MSE/epochs	1000 epochs	10000 epochs	15000 epochs
CIFAR-10	16781.41	112.11	72.08
MNIST	19959.93	214.59	94.65

7.4.2 Fast Gradient Descend Attack

Gradient descent is currently untrendy in the machine learning community, but there remains a large number of people using gradient descent on neural networks or other architectures from when it was trendy in the early 1990s. However its a simple yet effective attack, especially with older datasets of the machine learning world (like cifar-10 and mnist)

As it is known FGSM first calculates the gradient of J (J is the loss function) with respect to the input x, and then modifies each dimension of the input based on the sign of the gradient. For example, if the gradient is positive for some pixel, its value is increased. If the gradient is negative, its value is decreased. The amount of modification is determined by the parameter ϵ. A high ϵ will increase the chance of misclassification, but the resulting image will look very different from the original image. A low ϵ will produce an output image that is very similar to the input image, but the chance of misclassification becomes low as well.With only using one iteration, and having the ability to compute the perturbations for a batch of images at once, this allows the FGSM method to perform faster than the DeepFool method. However the algorithm is not designed to find the minimal perturbations, so it would result in larger perturbations and also requires manual adjustment of the parameter of ϵ to get desired results.

Now, if we look closer to the first two rows of images provided for this section, we can observe that the background pixels that used to be dark blue are now mixed with some lighter blue pixels, and some parts of the digit that used to be redhave turned a lighter shade of red/orange.

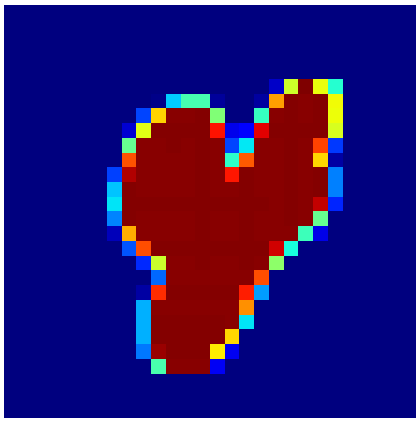


Figure 7.8: Initial Image Labeled as: MNIST_4

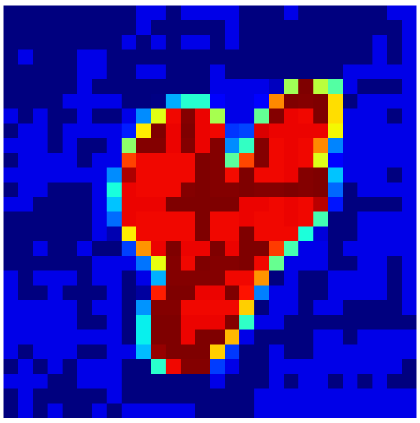


Figure 7.9: Adversarial Image Labeled as: MNIST_9

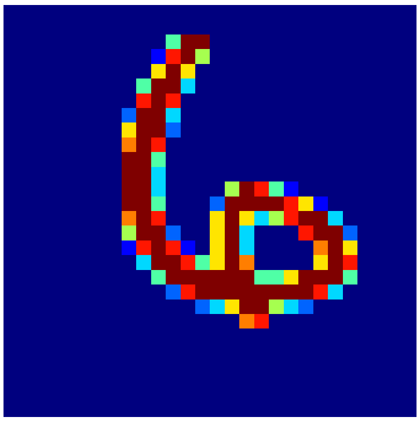


Figure 7.10: Initial Image, Labeled as: MNIST_6

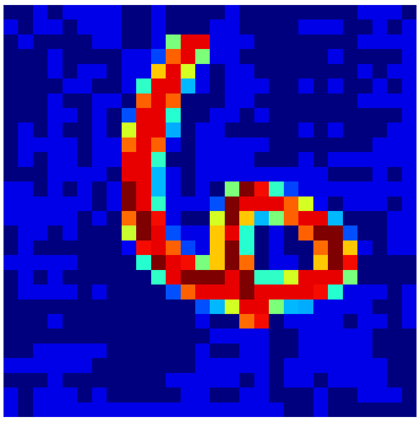


Figure 7.11: Failed Adversarial Image, Labeled as: MNIST_6

One downside of the FGSM is that the manipulated images are often perceptible for humans for anything but the smallest changes in pixel values. This may be because this method can only modify pixel values upwards or downwards a constant value rather than a seemingly random value. Additionally, manipulations using this technique are particularly noticeable around the darker areas of an image because the relative magnitude of manipulation compared to the original image's pixel values. This can be improved by using iterative methods.
Although Fast Gradient has higher probability of not achieving missclassification as effectively, as the other attacks under scope,it has a high accuracy when using CNNN models and low MSE loss in contrary to boundary attack which as we saw needed 8 hours of execution time and still the MSE loss was over 50.

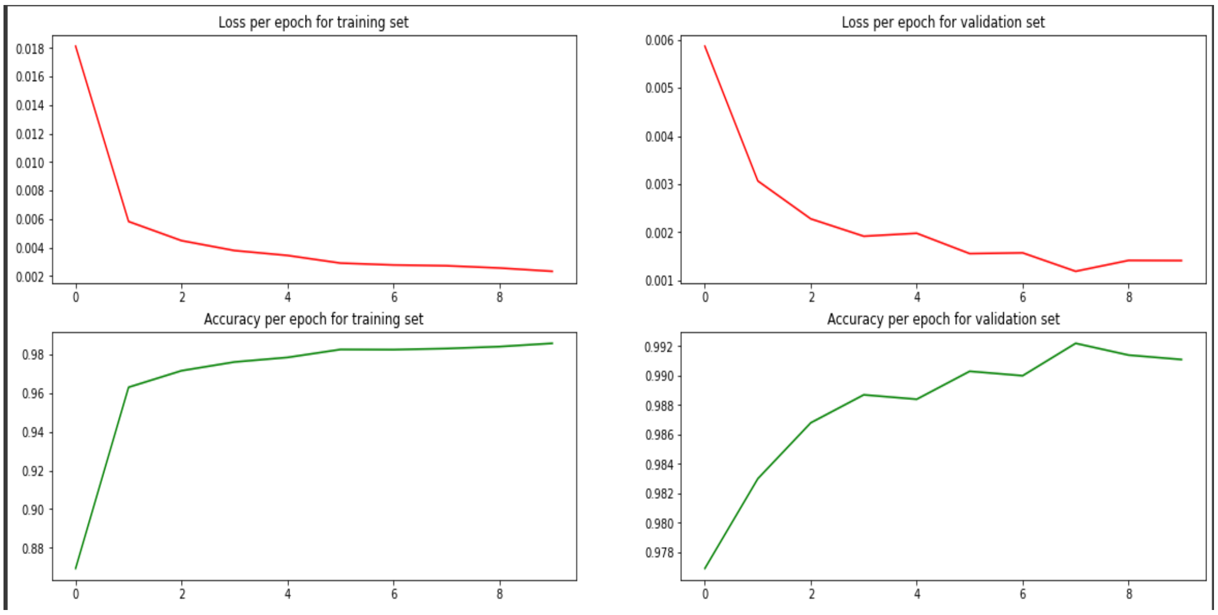


Figure 7.12: Statistics of FGSM attack using MNIST sataset.

7.4.3 DeepFool

The methods which we have analyzed so far use the gradient to increase the loss which allows them to approximate the optimal perturbation. DeepFool finds the projected distance to the closest decision boundary and iteratively changes the adversary until the class has changed. The results are less perceptible adversaries.DeepFool attacks are 100% successful and are always imperceptible.
The following figures (7.12 to 7.15) are 2 examples of applying deepfool attack on cifar-10 and mnist dataset, where on the left we can view the original image while on the right the resulting adversarial image.
By applying this attack we decrease the network's confidence from almost 100% down to 14%. The adversarial image appears almost identical to the initial one.We hypothesize that the reason for this is that DeepFool searches the closest decision boundary and slightly overshoots it.

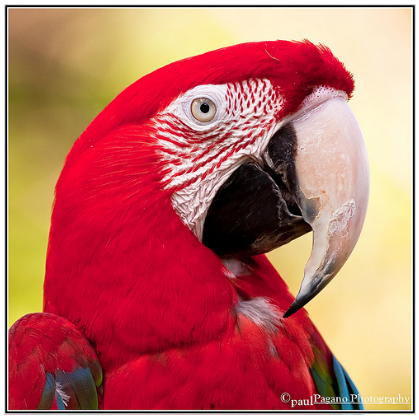


Figure 7.13: Initial Image using CIFAR-10, Labeled as: bird

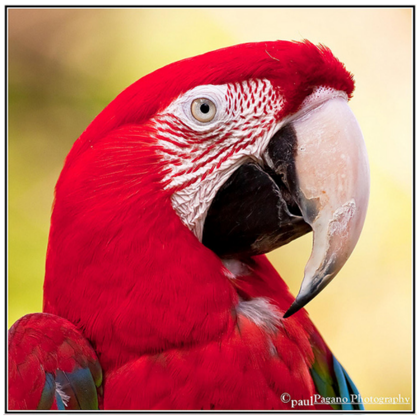


Figure 7.14: Adversarial Image, Labeled as: shrew

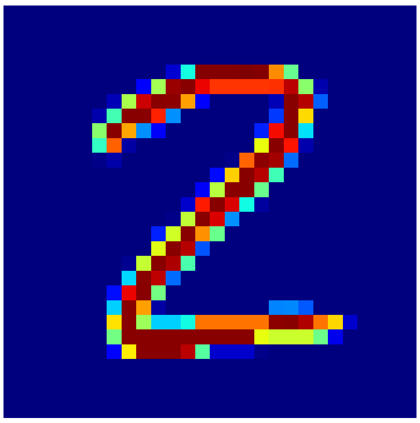


Figure 7.15: Initial Image, Labeled as: two

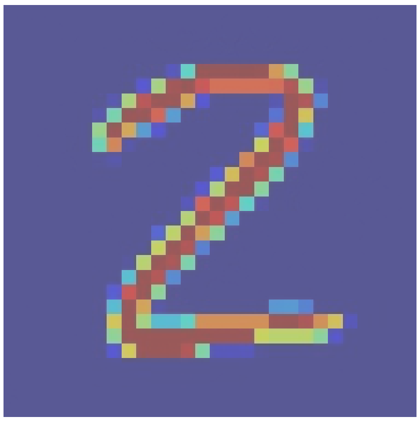


Figure 7.16: Adversarial Image, 8 loops performed, Labeled as: seven

Table 7.2: DeepFool Attack statistics

Classifier	Test error	$\hat{p_{adv}}$	time
CIFAR-10	0.17	3.0×10^{-2}	$1min53s$
MNIST	0.08	2.0×10^{-1}	$1min42s$

Although DeepFool has high performance rates in general and it converges in only a few steps, we have to keep in mind that the decision boundaries are often non-linear, so the algorithm completes the perturbation iteratively until it passes a decision boundary. Furthermore, this method generates very subtle perturbations in contrast to the coarse approximations of the optimal perturbation vectors generates by FGSM.
To sum up, DeepFool can be used as a valuable tool to accurately assess the robustness of classifiers. Empirically that DeepFool converges in a few iterations (i.e., less than 3) to a perturbation vector that fools the classifier, which makes it a good opponent against the rest of the whitebox attacks.

7.4.4 Carlini-Wagner

As we mentioned in the Carlini and Wagner chapter of this thesis, the referenced attack has proven to be a state-of-the-art method for the sector of whitebox attacks.
Although this attack algorithm can generate stronger/ higher quality adversarial examples ,the cost to generate them is also higher (because of the way optimization problem is formulated). Even though, Carlini suggests some relaxations in the paper to reduce the cost, it is still costly.
Compared to attack techniques like the Fast Gradient Sign Method (FGSM), another major difference here is that in the original formulation of the CW attack problem, the authors do not specify a threshold that sets a limit to the maximum distortion that is allowed, while in FGSM is done with a parameter ϵ. What this means is that, the attack here always succeeds.

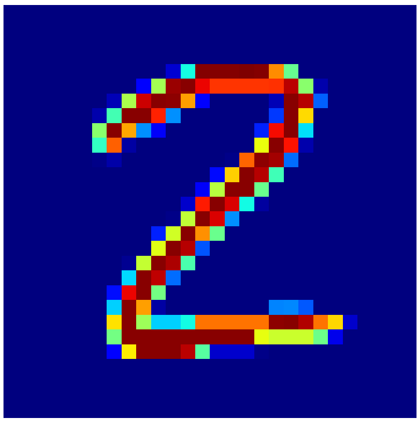


Figure 7.17: Initial Image, Labeled as: analog clock

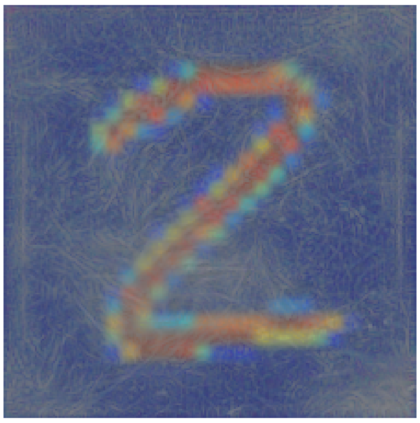


Figure 7.18: Adversarial Image,Labeled as: lion

When we first saw the results of the labeling done, we thought something went wrong, as the labeling for the initial image appeared to be classified as analog clock. However after some research we found out that this is a common occasion, when using the MNIST dataset paired with a VGG16 model. Although this happens, the attack is still successful as in the adversarial image the model classified the photo as lion! Below we created a diagram to showcase the percentages of classification for each photo (as we used a multiclass classifier)

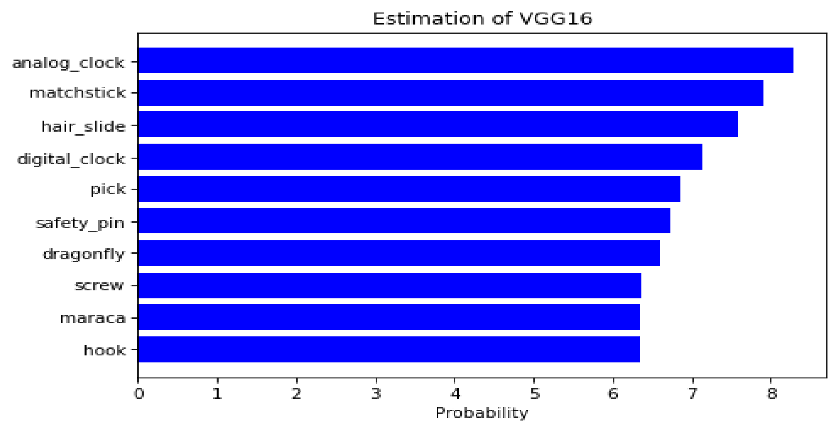


Figure 7.19: Classifications of original image

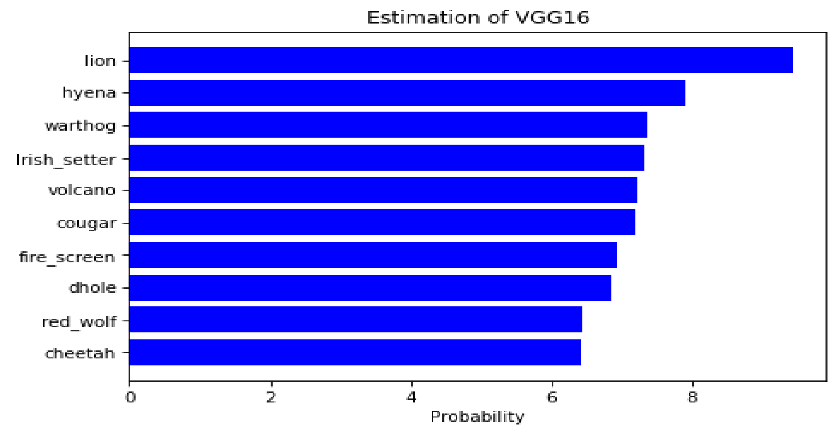


Figure 7.20: Classification predictions of adversarial image

Although the adversarial image appears a little more blurry than the original one, it doesn't necessarily mean something about the effectiveness of the attack. This could easily be optimized if we used a different model (for example a cnn, r-cnn, fast rcnn, which we mentioned that optimize carlini-wagners results and performance). However we wanted to showcase the modularity of the attacks and the easy application to all models (e.g. In this experiment we used the pretrained model tf.keras.applications.VGG16())

7.4.5 General observations and comparisons

Reaching the end of the evaluation chapter, we conclude with some notes regarding the comparison of the above models. Firstly, in the scenario where there is no available information regarding the model, dataset and in general have no knowledge of the targeted model, boundary attack is the only plausible choice from the group of attacks we have experimented on, as it is the only black box attack. It is highly effective as shown by the experimental statistics. However, one fundamental drawback is the high execution time (about 5 to 8 hours depending the images and type of boundary attack). That being said, boundary attack is still a higly effective method with final results that are both realistic and deceiving to the user.

On the other hand, FGSM (Fast Gradient Signed Method) is currently considered an outdated method, due to its higher probability of failing to misclassify. FGSM is a fairly computationally inexpensive attack that worked against most of the sample images. However, it has bigger failure rates against more complex images with high initial confidence. Another downside is that the perturbations added to the final images are more noticeable comparing to other attacks, which is an immediate result of the way te algorithm chooses to place the perturbations.
Nevertheless, FGSM is the optimal choice when performing adversarial training as a defence mechanism. Fast Gradiend methods can generate large quantities of adversarial examples in short time periods, compared to other attacks, therefore allowing the defender to have a bigger sample to ”poison” his dataset with.

Finally, regarding the Carlini-Wagner and Deep-Fool attacks they’re both state of the art attacks. as they can create a deceiving perturbated image in a short time frame. Although costly, Carlini-Wagner is a far more certain choice due to its structure that we described in chapter 6. DeepFool is based on searching the closest decision boundary and slightly overshooting it, which doesn’t necessarily guarantee successful results. Empirically though it has almost as high rates of performance as Carlini-Wagner.
However both attacks are mostly preferred when performing a white-box attack, as they are quick and efficient.

The code and research that was implemented for this thesis can be found in the following link: `https://github.com/AristiPap/Adversarial_ML_Research`

8. DEFENCE MECHANISMS

Up to this point, we explored and presented various types of adversarial attacks. We experimented and evaluated each attack on different databases in order to visualize which attack has better results and in general is more effective on different occasions. As we saw at the initial chapters of the thesis, adversarial attacks are a newly created sector with high effectiveness, either we are referring to whitebox or blackbox attacks. However due to the smart and innovative techniques that they use it has been proven a difficult task to defend properly against them.However after extensive research we found out that there are 4 methods that have proven to be quite effective on the attacks that we previously reviewed.These attacks often have tradeoffs in terms of computational cost and effectiveness, just as their defense counterparts do. On the flip side of the coin, we have defenses like the ones referenced below, which attempt to break down the input image and recreate it to remove noise or PCA which only retains the most important dimensions of the input sample. More specifically, these methods are:

- Adversarial Training
- Random Resizing and Padding
- Principal Component Analysis
- Increase of Robustness of models used for the training process

In this chapter we will document these four ways of defence for adversarial attacks and even implement a defence mechanism that works effectively against FGSM attacks at a high percentage. We will firstly describe the implementation details and the theory behind why the above methods seems to work as well as explore scientific papers and bibliography that document different and complex defence experiments.Furthermore, we will present couple of points on why, some of the defences, might fail against the previously referenced attacks. Finally we will showcase the implemented defence method and elaborate with statistics,plots and experimental results.

8.1 Adversarial Training

8.1.1 Overview

As it is easily understood by the title itself, adversarial training simply speaking means that while the training is going on we also generate adversarial images with the attack which we want to defend and we train the model on the adversarial images along with regular images. Put simply, you can think of it as an additional data augmentation technique.Adversarial training is one of the most effective approaches to defending deep learning models against adversarial examples. Unlike other defense strategies, adversarial training aims to enhance the robustness of models intrinsically. During the last few years, adversarial training has been studied and discussed from various aspects.

A referenced by Madry [12] in order to have a successful defence tactic we consider an adversarial variant of standard Empirical Risk Minimization (ERM), where our aim is to minimize the risk over adversarial examples:

$$h^* = argmin_{h \in H}\ E_{(x,y_{true})\sim D}[max_{\|x^{adv}-x\|_{\inf}\leq\epsilon} L(h(x^{adv}), y_{true}]$$

Madry argues that adversarial training has a natural interpretation in this context, where a given attack is used to approximate solutions to the inner maximization problem, and the outer minimization problem corresponds to training over these examples. Note that the original formulation of adversarial training [12] [9], which we use in our experiments, trains on both the “clean” examples x and adversarial examples x^{adv}.Although adversarial training is thought to be the most effective method in practice for any defence mechanism, it’s still a long way to go for adversarial training to handle adversarial attacks perfectly. Prevailing adversarial training methods can produce a robust model with worst-case accuracy of around 90% on MNIST. For slightly more challenging datasets, e.g., CIFAR-10, adversarial training only achieves around 45% , which is far from satisfactory. Additionally, adversarial training leads to the degraded generalization ability of deep learning models.

8.1.2 Adversarial Training with Adaptive ϵ

As we witnessed in the algorithms of the various adversarial attacks, the parameters of the methods are predefined and fixed during training. However it is argued by many engineers that individual data points might have different intrinsic robustness and different distances to the classifier’s decision boundary. However, adversarial training with fixed ϵ treats all data equally.Considering the individual characteristic of adversarial robustness, researchers propose to do adversarial training at the instance level.If we take into account the *Instance Adaptive Adversarial Training (IAAT)*[1], where ϵ is selected to be as large as possible, ensuring images within ϵ-ball of x are from the same class. This strategy helps IAAT relieve the trade-off between robustness and accuracy, though there is a slight drop in robustness. After IAAT, another process was creates: Customized Adversarial Training (CAT) [5] which further applies adaptive label uncertainty to prevent over-confident predictions based on adaptive ϵ.

Adversarial training with adaptive ϵ is a good exploration.However, empirical evidence shows many standard datasets are distributionally separated, or the distances between the inner classes are larger than ϵ used for attacks. This reflects the limitation of current adversarial training methods on finding proper decision boundaries.

8.1.3 Tuning of Variants for Adversarial Training

There are many tweaks and transformations that we can apply to adversarial training to make it more effective (depending also on the datasets that we are handling).Other variants of adversarial training are summarized that seem to be modified in different experiments are the learning objectives of vanilla adversarial training, like adversarial distributional training where a distribution-based min-max problem is derived from a general view. Also a famous alteration of this defence mechanism is bilateral adversarial training [18] where the model is training on both perturbed images and labels. Some models also replace the fundamental components of their implementation for performance reasons, like hypersphere embedding, and smoothed ReLU function.

8.1.4 Random Resizing and Padding

With this process, given an image,we have to randomly resize the image (of all 4 sides) and then pad the image randomly. This strategy was used from the team that won 2nd place on NeurIPS competition, hosted by Google Brain[21]. The methodology they used was the below:

- Set resizing range $\in [299, 331]$
- Set padding size $= 331 \times 331 \times 3$
- Average the prediction results over 30 such randomized images
- Where for each such randomization you also flip the image with a 0.5 probability

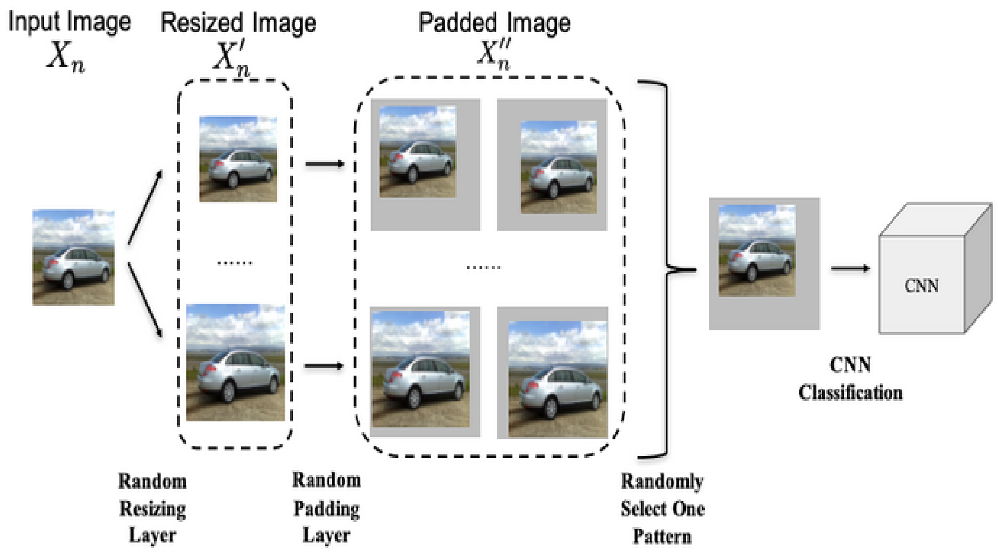


Figure 8.1: Visualization of the random padding layer pads in the pipeline of the randomization-based defence attack

8.2 Principal Component Analysis (PCA)

Principal Component Analysis is a technique for dimensionality reduction that identifies patterns in data based on the correlation between features. It then transforms the data by projecting it onto a set of orthogonal axes with equal or fewer dimensions. The goal of this is to extract the highest impact dimensions of some data, so that the noise, which would not change those dimensions greatly, can be filtered out. It has been shown that PCA is an effective defense against certain adversarial attacks on smaller datasets, where n is the number of samples in the dataset and d the number of features (rows × columns) of each sample. 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. However after searching in the bibliography available we found a method that overcomes this problem[11]. Additionally, this method is independent of the dataset size.Furthermore, our method also has the added advantage that it requires no knowledge of the dataset which makes it more versatile.

Algorithm 6 Finding the (k, p) point for a given neural network f (·), input image x, top scoring class on input image c(x), and maximum number of principal components n. We reconstruct the image from components 1 through i and find the point at which the dominant class is no longer dominant

```
$k \leftarrow n$
$topper \leftarrow argmax f(x)$
while $topper == c(x)$ do
    $k = k - 1$
    $x^* = P_{inv,row}(C_{1:k})$
    $topper = argmax f(x^*)$
    if $topper \neq c(x)$ then
        $p = p(x^*)[topper]$
        return (k,p)
```

8.3 Increase of Robustness of models used for the training process

In general, adversarial robustness is the model's performance on test data with adversarial attacks, adversarial test accuracy of classification.We expect to know the model's performance in the worst case scenario, so the adversary should be strong enough and launch attacks in white-box settings, and the adversary has full knowledge of the model, such as architectures, parameters, training data, etc.

As we saw before, the more robust a model is, the more difficult it is for it to be susceptible to attacks (for example although Carlini and Wagner is the best attack currently out there, it is very expensive and time consuming than other attacks. So in a robust model there is a possibility that carlini would stop before having a result (this in case we added specific number of iterations as a threshold)).
Some ways to make the model more robust are the following:

- use transformers instead of cnn models as it has been proven for transformers to be far more robust [19]
- use a model that's resistant to outliers. Tree-based models are generally not affected by outliers, while regression-based models are. If you are performing a statistical test, try a non-parametric test instead of a parametric one.
- use a robust error metric: Switching from mean squared error to mean absolute difference reduces the influence of outliers
- Artificially cap data at some threshold.
- Transform data - If data has a very pronounced right tail, try a log transformation.

8.4 Simple implementation of adversarial training for FGSM attack

After exploring all the possible defence mechanisms we came to the conclusion that the easiest but also the most effective strategy is *adversarial training*. This method can be easily applied to any of the previous attacks that we took into account, however after personal experiments I noticed that FGSM had the highest accuracy and efficiency when adding adversarial training. The only thing that needs to change is that except from finetuning our model on the val dataset that we had in the beginning, we now create a dataset of adversarial images, and after fitting the trained model on the actual dataset, then we finetune it to the adversarial dataset.

The results that we got from adding adversarial training to the FGSM attack are the following:

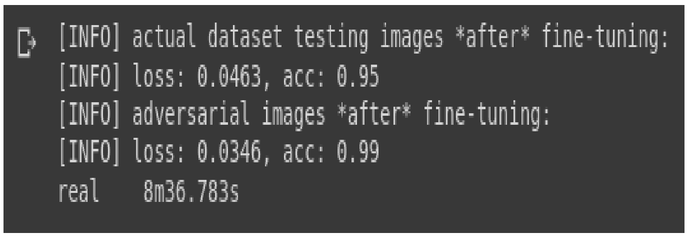

```
[INFO] actual dataset testing images *after* fine-tuning:
[INFO] loss: 0.0463, acc: 0.95
[INFO] adversarial images *after* fine-tuning:
[INFO] loss: 0.0346, acc: 0.99
real    8m36.783s
```

Figure 8.2: Results from adding adversarial training to the FGSM attack model

Initially, our CNN obtained 98.92% accuracy on our testing set. Accuracy has dropped on the testing set by $\approx 0.5\%$, but the good news is that we're now hitting 99% accuracy when classifying our adversarial images, thereby implying that:

- Our model can make correct predictions on the original, non-perturbed images from the MNIST dataset.
- We can also make accurate predictions on the generated adversarial images (meaning that we've successfully defended against them).

8.5 Observations on (Failed) Defence trials

Madry [12] used PGD, an iterative and stronger version of FGSM, for adversarial training. This scheme is found to be effective for images created from similar attack algorithms. Specifically, the network can accurately classify most of adversarial samples, if the L_{inf} distance between the original and the adversarial pair is bounded by some parameter e. However, Schott [13] showed that, while the adversarially trained network is resistant to attacks using L∞ distance, it is still vulnerable to other adversarial samples created using other distance metrics.

Specifically, we experimented with said defence methods, on all four attacks that we presented in this thesis. As expected, Madry and Schott's coclusions were verified: FGSM showcased high rates of accuracy on adversarial images, meaning that adversarial training is actually an effective defence method. However, the rest of the attacks (Boundary, Hybrid Boundary, DeepFool and CarliniWagner) don't seem to be affected by such methods. In detail, as it can be seen in the following plots, the accuracy of the model in all three cases is almost stable, we can only notice a decrease of max 10%. However the only downside is that we have an increase in the executional time, which is normal as we have "poisoned" the original dataset with perturbed photos, which is harder to classify.

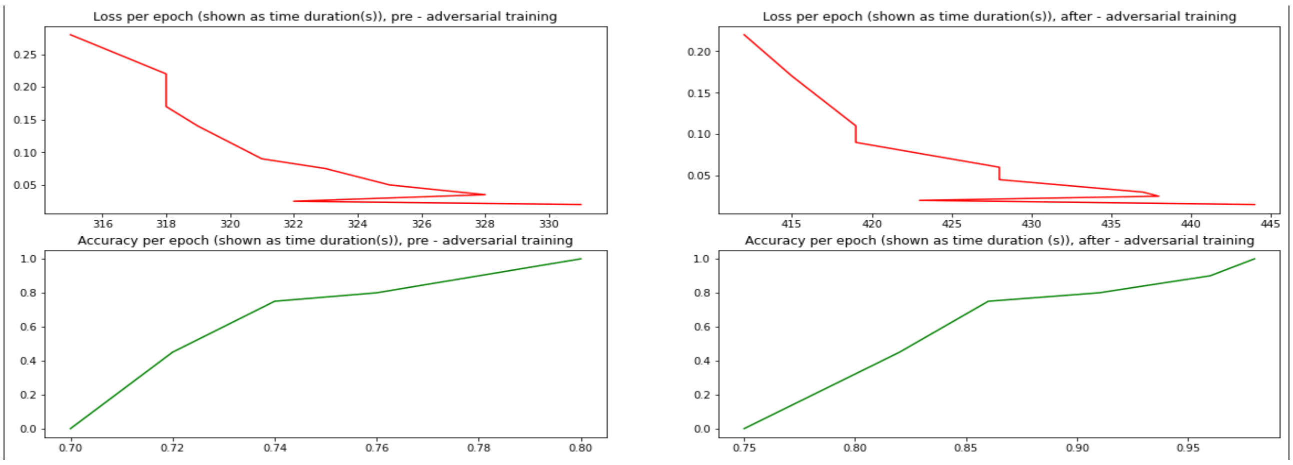


Figure 8.3: Results from adding adversarial training to the FGSM attack model

9. CONCLUSIONS AND FUTURE WORK

The goal of this thesis was to explore the big downfall that adversarial machine learning caused to the deep neural networks. After showcasing some of the most famous attacks that are currently used to cause damage in multiple universal datasets , it is clear that Adversarial Machine learning is a sector with big potential and much more to give in the advancements of deep neural networks.

Its safe to say that Adversarial ML is the future of Datascience and cybersecurity as we know it. After exploring only a couple of the available methods that could exploit any dataset available, we noticed that the way each attack strategy creates perturbations is unique and sometimes dynamic (like boundary attack, our proposal of the hybrid boundary attack and Carlini wagner strategy). Just by creating an alteration in the calculation of a constant or by using local/global search we are able to find promising or even optimal adversarial points. As a result, due to the vast variety of different perturbations generated, it is almost impossible to create defences and secure models enough so that they are covered from all types of attacks. In our case just by making an alteration in the initial step of allocating the first possible adversarial point, we created a much more effective version of the boundary attack.

However, due to the fact that Adversarial Machine Learning is a fairly new sector of ML, we still haven't seen many innovations in the defence sector. In our experimental studies we figured out some advancements that have been referenced in different bibliography like adversarial training and increase of model robustness, we created a custom Early stopping class in order to avoid model overfitting etc. These methods seemed to be working great in some of our attack processes like FGSM and even DeepFool,in some cases, but unfortunately they are still not capable enough to cover all the different variations of adversarial attacks effectively.

Considering the research we did for this thesis, some of our future work ideas would be to construct a dynamic tuning system for the DeepFool attack (or even FGSM) as in that case these attacks would be able to outperform Carlini-Wagner and at a very low cost. Furthermore after creating a hybrid version of the Boundary attack, next step would be to find an effective way to lower the execution time, making it the ultimate choice when using Black-box attacks.Finally, the end goal is to create enhancements in the defense sector of adversarial machine learning that could perhaps be inspired by using the notions of gradients, DNNs and higher knowledge on model performance. After all ethical hacking is a process of detecting vulnerabilities in an application, system, or organization's infrastructure that an attacker can use to exploit an individual or organization, in order to take action and strengthen the defence mechanisms available.

ABBREVIATIONS - ACRONYMS

FGSM	Fast Gradient Sign Method
ML	Machine Learning
CNN	Convolutional Neural Networks
DNN	Deep Neural Networks
CAT	Customized adversarial training
FAB	fast adaptive boundary attack
CSV	Comma Separated Values
SGD	Stochastic Gradient Descent

BIBLIOGRAPHY

[1] Yogesh Balaji, Tom Goldstein, and Judy Hoffman. Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets, 2019.

[2] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models, 2018.

[3] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models, 2018.

[4] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks, 2017.

[5] Minhao Cheng, Qi Lei, Pin-Yu Chen, Inderjit Dhillon, and Cho-Jui Hsieh. Cat: Customized adversarial training for improved robustness, 2020.

[6] Francesco Croce and Matthias Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack, 2020.

[7] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness, 2021.

[8] Ross Girshick and Microsoft Research. Fast r-cnn, 2015.

[9] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

[10] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies, 2017.

[11] Malhar Jere1, Sandro Herbig, Christine Lind, and Farinaz Koushanfar. Principal component properties of adversarial samples, 2019.

[12] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[13] Lukas Schott, Jonas JRauber, Matthias Bethge, and Wieland Brendel. Towards the first adversarially robust neural network on mnist, 2018.

[14] Moosavi-Dezfooli Seyed-Mohsen, Fawzi Alhussein, and Frossard Pascal. Deepfool: a simple and accurate method to fool deep neural networks, 2016.

[15] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014.

[16] Jon Vadillo and Roberto Santana. Universal adversarial examples in speech command classification, 2021.

[17] Jon Vadillo, Roberto Santana, and Jose A. Lozano. Exploring gaps in deepfool in search of more effective adversarial perturbations, 2020.

[18] Jianyu Wang and Haichao Zhang. Bilateral adversarial training: Towards fast training of more robust models against adversarial attacks, 2019.

[19] Zeyu Wang, Yutong Bai, Yuyin Zhou, and Cihang Xie. Can cnns be more robust than transformers?, 2022.

[20] Xiaosen Wang1, Yichen Yang, and He-Kun Deng, Yihe. Adversarial training with fast gradient projection method against synonym substitution based text attacks, 2020.

[21] Cihang Xie, Zhishuai Zhang, Alan L. Yuille, Wang Jianyu, and Ren Zhou. Mitigating adversarial effects through randomization, 2018.