Università
Ca'Foscari
Venezia

Master's Degree

in Computer Science

Final Thesis

Certifying Robustness of Machine Learning Models Utilizing Adversarial Attack Transferability

Supervisor
Prof. Stefano Calzavara

Assistant supervisor
Lorenzo Cazzaro

Graduand
Shehzad Inayat
888374

Academic Year
2022 / 2023

Index

Abstract

Machine learning has become increasingly popular for its ability to learn from data, identify patterns and make logical decisions with little or no human intervention, allowing humans to rapidly develop models that can analyze extraordinarily large and ever-increasing volumes of data. Machine learning models, for instance, Convolutional Neural Networks (CNNs), received attention due to their purposeful use in a wide variety of areas, such as self-driving cars and cyber security. However, recent studies have shed light on how such systems can be compromised by test time evasion attacks, i.e., carefully engineered adversarial examples with imperceptible perturbation, raising security concerns about using such models in safety-critical systems. Furthermore, adversarial examples may exhibit the transferability property, i.e., adversarial examples crafted for one model may evade also potentially unknown models, that makes attacks practical even in the black-box setting. Machine learning models need to present satisfiable performance also in adversarial settings, thus it's crucial to evaluate faithfully their robustness against evasion attacks. Since in real-world scenarios (black-box settings) target models may not be directly accessible and it may be difficult to verify their robustness, we propose a framework that allows the analyst to evaluate efficiently the robustness of target models by leveraging simple well-known surrogate models and the transferability of adversarial attacks. Our proposal consists in combining the information about the robustness of surrogate models evaluated on a test set using different logical gates to approximate the robustness of the target model, hoping that the information about the robustness of surrogate models transfers to the target model. In addition, along with the measure of transferability for each model, we explore the correlation between other information available to the analyst and the best gate, in order to suggest a strategy to identify the best aggregation function in different settings. The preliminary experimental evaluation on MNIST dataset using different machine learning models shows the possibility of approximating effectively the robustness of target models via surrogate models.

Keywords: Machine Learning, Adversarial Machine Learning, Supervised Learning, Adversarial Attacks, Transferability, Robustness, Security

Chapter 1

Introduction

Machine learning (*ML*), a branch of *Artificial Intelligence* (*AI*), is a unique approach that effectively looks at the use of computational algorithms and allows a machine to learn from data, turning the observed data into a usable model which needs not to be reprogrammed, and leveraging the training data to improve its performance on a specific set of tasks [1]. The main goal of machine learning models is to perform complex tasks or solve problems, the way humans do, by emulating human behavior. Machine learning models have widely been adapted in different domains, for medical diagnosis, classification, image processing, prediction, etc., due to their powerful and unparalleled flexible capabilities in dealing with evolving input in a variety of applications and their accurate results. However, despite being extremely powerful, these models can fall prey to malicious adversaries [2]. It has been shown, in previous research, that these models are sensitive to adversarial perturbations and when applied to the input data, eventually mislead the model. Security concerns arise, for using these models in safety/security-critical systems, following how these supposedly robust models can be fooled fairly easily with unnoticeable noise applied to the original input. Take, for instance, self-driving vehicles that use these models and rely on the surrounding and traffic signs to drive the vehicle. The performance of these models is dependent on factors such as illumination, weather, or other which influence it [2]. However, even in perfect illumination or weather conditions, inputs with a careful human imperceptible noise can force them to misclassify the signs. In [3], *Papernot et. al.* show how a carefully engineered adversarial input image, that looks the same as the original, can force the models to predict, what originally is a stop sign, as a yield sign which can put a vehicle in a dangerous situation, successfully carrying out an evasion attack.

The phenomenon of *adversarial examples* is not as new as it seems. As pointed out, the attacks against machine learning date to around 2004 when adversarial examples were studied in the context of spam filtering showing that linear classifiers could be easily made to do mistakes by a few carefully crafted changes in the context of spam emails [4]. The newness came from being discovered in the context of deep learning when it was found while trying to understand the decision-making process of neural networks, that they appear to possess adversarial examples that can mislead them [5] bringing the phenomenon of adversarial examples into the light. The process of feeding delicately perturbed inputs to a classifier at test time that feels the same as the original image to the human eye but throws away the respective classifier is known as an *Evasion attack*. In this kind of attack, the adversary uses the test data, the data which the model uses to make predictions, as opposed to data used to train models that come in the category of *Poisoning attacks*. The classifier is fixed in the case of a black-box evasion attack and neither the structure nor its parameters can be accessed by the attacker. [6], [3] shows how the algorithms working behind self-driving cars for road sign recognition can be evaded by slightly perturbing the road sign image or just putting stickers on the road sign in desired spots.

We are in an era where security is constantly updated and it might not be possible to have direct access to the target model in most cases, either to its structure or its parameters, leaving the attacker no option but to attack the model in a black box setting. *Black-box attacks* are the kind of attacks where the adversary has very little or possibly no knowledge of the target model and are usually designed (in the case of surrogate model training) to query the target model on inputs and observe the outputs to infer knowledge about the target model as opposed to white-box attacks where the attacker can have complete access to the model, the training and test data, the architecture and the parameters as well as the weights that are learned by the model, at training time. These attacks might seem to limit the attacker's capabilities; however, there is the more practical fact that, in real-world scenarios, the models are inaccessible to an adversary, hence, they are the attacks mostly seen in the practical world. Apart from the fact that an attacker has access to querying the model, the querying access might be as well limited to a specific number of queries, limiting the attacker's aptitude even further.

While black-box adversarial attacks limit the ability of the adversary, there is another novel phenomenon, *transferability*, which can make black-box attacks extremely effective. Adversarial examples crafted for one model can elude the model, nonetheless, it is not limited to evading that specific model but can evade other models as well. Transferability is the ability of an attack, generated against one specific model, to be significantly effective against another potentially unseen model. Since, possibly in real-world scenarios, a black-box attacker might be limited in terms of queries he can make to the target, the marvel of transferability overcomes this problem for the attacker. The first empirical black-box attacks, utilizing transferability, were conducted by *Papernot et al*, attacking real-world remotely hosted deep neural networks, achieving significant success, as well as, finding the extreme capability of black-box attacks being able to evade strategies previously found to make adversarial attack crafting harder [3]. The strategy is to train a surrogate model, available to the adversary locally, with data where the instances are generated synthetically by the adversary and labeled by the target network. The next step is to craft adversarial attack examples for the surrogate model, fed to the target model for prediction, and see them succeed in misleading the target classifier. There has been extensive research work of the same kind but on large-scale datasets [7] and for boosting the transferability [8]–[10].

To make models, to be deployed for problem-solving, robust against these black-box attacks, it is crucial to understand what causes the phenomenon of transferability to happen. The reason is that machine learning models have an inborn or natural vulnerability to adversarial attacks, also, the substitute model complexity plays a major role in the transferability of the attacks [11]. Furthermore, the size of the gradient of the input, the gradient alignment of the substitute and target model, and, in addition, the variance of the loss landscape have a big impact on the transferability. In addition, the robustness measure is focused on by some in the literature as a metric for the security assessment of machine learning models. Furthermore, existing literature has focused on formal methods-based verification techniques and robustness certification techniques based on verification and some other methods to give robustness guarantees.

1.1 Problem Description

Dealing with the threat of adversarial attacks has been in the spotlight for quite some time and research work has been done in this regard to make machine learning models robust to attacks. However, to cope with the threat of adversarial attacks, it is vital to test the models for *robustness*, a measure of the extent to which a model can overcome or withstand adverse conditions (in this case, adversarial attacks), before deploying them. Since the accessibility to an ML model in a black-box setting is very limited, usually only query access [3], it might be possible to utilize transferability, for the fact that adversarial examples transfer between models, to perceive the measure of the robustness of the target model. In this work, we study whether using surrogate models and their transferability, we can infer the robustness information of the target model in black-box settings.

1.2 Contributions

We summarize our main contributions as follows:

- We propose a framework for certifying the robustness of machine learning models using an ensemble of surrogate models, used simultaneously and their results combined using four logic gates and utilizing their transferability to the target models. An analyst who wants to evaluate the robustness of a model in black-box settings because the model is inaccessible to him can use our framework to provide robustness guarantees to the owner or an attacker using our framework can gather robustness information of the target model that disclosed.

- We present preliminary experimental evaluation, gained as a result of conducting an extensive assessment, to show that surrogate models can be productively used, using our strategy, to certify the robustness of the target model.

1.3 Organization

We organize this thesis as follows:

- In chapter 2, we discuss the background of topics needed as a basis for this thesis. We talk deeply about supervised learning, classification, and the three classification models used in this thesis (Support Vector Machines (SVM), Logistic Regression (LR), and Neural Networks (NNs)). Also, we give a deeper introduction to Adversarial Machine Learning, adversarial examples, Transferability as well as black-box and white-box evasion attacks.

- In chapter 3, we introduce and define stability and robustness, the importance of establishing the security of ML models, the measure of robustness and how robustness can be certified as well as the challenges in computing robustness in black-box settings.

- In chapter 4, we talk in detail about our approach to compute the robustness of an ML model using surrogate models in black-box settings.

- In chapter 5, we put forward our experimental assessments for the development of this thesis.

- Last but not the least, we conclude the thesis by talking briefly about our contributions and future research directions.

Chapter 2

Background

The term ‘Machine Learning’ was first coined by an American researcher, *Arthur Lee Samuel,* the pioneer of artificial intelligence and computer gaming, who stated it as, the field of study allowing the machines to learn without the requirement to unequivocally reprogram it [12]. Arthur's study not only defined machine learning but also included programming a machine in such a way that the resultant behavior corresponds to how humans behave and constitutes learning. Data was found to be the driving force in helping the machines learn, and, by the mid-2000s, with success stories amassing from extensive research, it was discovered that the data can be way stronger than theoretical models [13]. The evolution of computing technologies has brought at our disposal, machines, that are more powerful than ever before. It became possible to utilize these powerful machines and machine learning algorithms to identify complex and concealed patterns that would not be easy for humans, or even possible, with the amount of data that is available to us contemporarily. In layman or simple terms, machine learning develops models from training data and automates the process of adapting to a multiplex of facts and figures to perform tasks or solve problems.

Machine learning algorithms can be divided into categories depending on the purpose they are used for, but the main categories are as follow:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

For empirical analysis of this thesis, supervised learning models are used so we will discuss here only the supervised learning paradigm.

2.1 Supervised Learning

Supervised learning is a class of problems that acquire the input-output relationship information of a system given a training dataset consisting of instances and their true labels [14]. Also cited as Learning with a Teacher as well as Inductive Machine Learning [15], [16], the objective of supervised learning is to build concise models that learn the mapping between the inputs and outputs given supervised data. After the training is over, the learned model is tested on a test set where only the test input vectors (without their corresponding labels) are provided to the model, allowing the model to predict the test examples, leading to uncovering the precision or correctness of the model by comparing the predicted labels with the true labels.

During supervised training, the instances (usually denoted by $x_i \; where \; i = 1 \ldots n$) are fed to the model being trained to get outputs. The goal is to minimize the error on the entire dataset and in case of prediction error (higher difference between predicted output and the ground truth) during the training process, the parameters of the model are adjusted for minimized error. However, minimizing the training error does not guarantee that the model will perform equally well on the test data which is unseen and new to the classifier. The poor performance may be due to various reasons one of which is overfitting. When a model overfits, despite its good performance on training data, it is not able to generalize well to data unincluded in the training set, therefore, a balance between error minimization and complexity of the model is necessary. Take, for instance, Support Vector Machine (SVM), which maximizes the decision boundary leading to good generalization [17].

2.1.1 Classification

Classification is a supervised learning problem where the problem to solve is to find a mapping between instances and a set of pre-defined class labels or targets. Formally,

Definition 1[18]**.** Given a labeled set of instance x ∈ S (S being the training set), along with their formal target labels y from a data distribution D which is unknown, and an inducer I, the objective of the classification task is to produce a classifier with generalization error as low as possible.

Generalization error is the rate of misclassification concerning the distribution and can be written as [18]:

$$\varepsilon(I(S), D) = \sum_{(x,y)\in U} D(x, y).\, L(y, I(S)(x))$$

Where $I(S)$ is the model while $L(y, I(S)(x))$ is the loss function of the model

$$L\big(y, I(S)(x)\big) = \begin{cases} 0 \;\; if \; y = I(S)(x) \\ 1 \;\; if \; y \; ! = I(S)(x) \end{cases}$$

Machine learning classification algorithms make use of input training data to predict the likelihood of falling into one of the pre-defined classes. In the classification algorithm, a discrete output function (y) is mapped to the input variable (x). For the sake of simplicity, we will use $f()$ as the notation for the learned model or classifier.

$$y = f(x), \text{ where } y = \text{categorical output}$$

Performance evaluation is crucial for the sake of uncovering the quality of the learned model. There are many metrics for the evaluation of performance however most commonly and frequently used is the *accuracy* and its complement *error rate* [19].

$$Accuracy = \frac{Correctly\ predicted\ instances}{Total\ number\ of\ instances}$$

$$Error\ rate = \frac{Incorrectly\ classified\ instances}{Total\ number\ of\ instances}$$

Accuracy and error rate complement each other so finding one, the other can be computed by deducting the computed from 1.

$$Error\ rate = 1 - Accuracy$$

2.1.2 Classification Algorithms

Classification algorithms are used to categorize data into a class or category. It can be performed on both structured and unstructured data. Classification can be of three types: binary classification, multiclass classification, and multilabel classification. There are many algorithms used for classification, but we will discuss here the models specifically used for the motive of experimenting for this thesis.

2.1.2.1 Support Vector Machine (SVM)

Support vector Machines (SVMs) are one of the most robust prediction methods used for binary classification where the input vectors are mapped to a high dimensional feature space [17]. Given a set of labeled training examples, builds a model that assigns new examples to one or the other category. The training examples are mapped into a feature space and separated using a hyperplane that is equidistant as well as which maximizes the distance between the data points of two classes.

The hyperplane is found using support vectors which are the points nearest to the hyperplane and influences its position and orientation. The reason, for finding a hyperplane that is at a maximum distance from the data points of the two classes, is that the future points are assigned a class with maximum confidence.

Let's suppose we have some points, represented by $x \in \mathbb{R}^D$ in vector space representing feature vectors. However, this maybe too simple so we want to map this to a more complex non-linear feature space transforming the feature space, i.e., $\mathbb{R}^D \rightarrow \mathbb{R}^M$, where each of these feature vectors x is mapped to a transformed basis vector $\emptyset(x) \in \mathbb{R}^M$.

How are the feature vectors separated? Feature vectors are separated using a decision boundary, called hyperplane, separating these points into their respective classes.

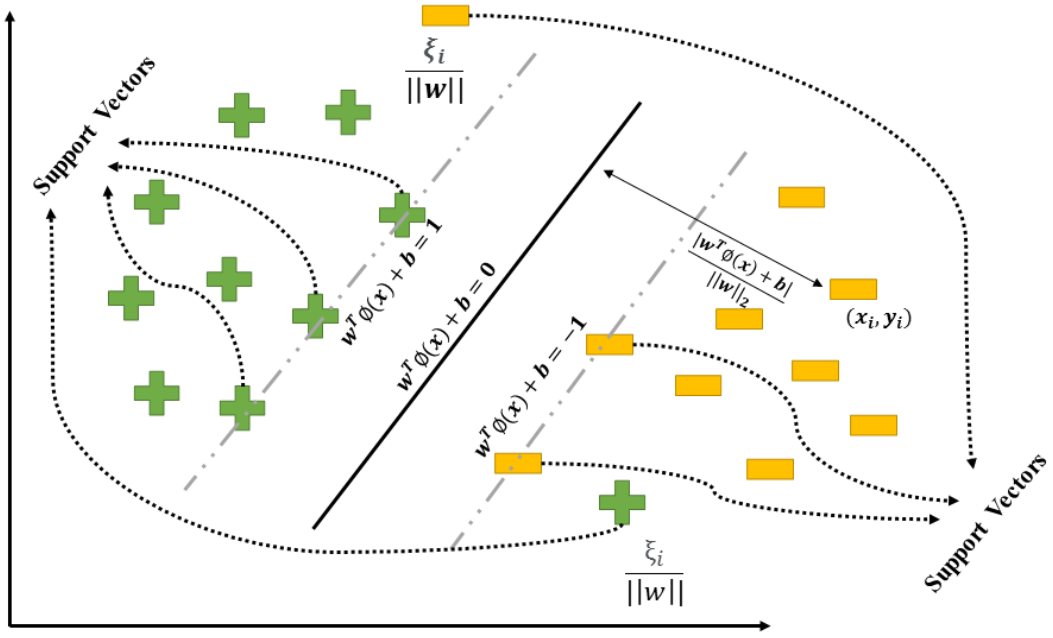

Fig 1: SVM decision boundary maximization

Consider the case when the data is perfectly separable in a binary classification problem. Nonetheless, there can be different possibilities of hyperplanes all separating the data with cent percent accuracy. What SVM does is it chooses the hyperplane which maximizes the distance to the nearest points on either side of it. These points are called support vectors and play an important role in choosing the maximum-margin hyperplane.

The classes usually in binary classification are 0 and 1. As the value of the hyperplane is 0, the value of the hyperplane for feature vectors in the feature space would be greater than 0 for one group of data and less than 0 for another. Furthermore, during the testing time, the product of the predicted value and the actual value would be greater than 0 for correct classification otherwise

less than 0. Since we are in a condition where the classes are perfectly separable, all the samples would be classified correctly.

Let's consider the case now when the data is not perfectly separable with a straight line. In this case, a decision boundary that perfectly classifies feature vectors into their respective class will make the model overfit the data. *Vapnik, 1995 []* introduced the concept of slack variables to tackle the situation when the data is not perfectly separable, i.e., when the noise present seeds the flap of the target classes, relaxing the hard margin constraints and inducing a soft margin. Mistakes are allowed in the case of imperfectly separable data by introducing slack variable denoted by the Greek letter ξ, valued greater than or equal to zero for all, which serves as a penalty for incorrect classification. A data point classified correctly will have a slack value of 0 and 1 otherwise. The variable C serves as a regulator which regulated the value of slack variables. The higher the C is, the more complex the decision boundary becomes.

The choice of linear classifiers can be problematic sometimes because it is often the case when the data is not linearly separable. With the introduction of *Kernel Trick* (*Vapnik and Cortes, 1995*), it's possible to have maximum margin hyperplanes with linear models and non-linear decision functions.

To understand the essence of SVM, classification, one needs to grasp four basic concepts: *(i) the separating hyperplane, (ii) the maximum-margin hyperplane, (iii) the soft margin, and (iv) the kernel function* [20].

2.1.2.2 Logistic Regression

Logistic Regression (LR), originally developed by *Joseph Berkson* as a general statistical model [21], despite its name is a classification model, is an efficient method for binary and linear classification problems and has a very good performance not only with binary or linearly separable problems but with multiclass problems as well. When the dependent variables are binary, it is LR that is most appropriate to be used. Logistic regression is used as a tool for predictive analysis describing the correlation between a binary dependent variable and one or more independent variables.

Logistic regression, also known as the *logit model*, uses a logistic (sigmoid) function to model binary dependent variables and is often used for solving classification problems as well as predictive analysis. Logistic regression calculates the probabilities of events to occur based on the available set of independent variables, for instance, head or tail in case of tossing a coin. The value of the dependent variable is usually between 0 and 1 for the fact that the outcomes are probabilistic. Logistic regression uses logit transformation on the odds, dividing the success probability by the failure probability.

Given points $x \in \mathbb{R}^D$ representing feature vector in feature space, the logistic or sigmoid function $F(x) = e^x/e^x + 1$ make sure that the values are between 0 and 1. Logistic Regression is called a classification model because it starts as a linear equation. Due to the presence of log-odds, we use the sigmoid function to help get the output values as a probability. These probabilities can

then be effectively used to conduct classification tasks. For instance, we want to predict whether a student will successfully pass the exam or fail it. The probability, to predict whether that student will pass, has to be higher than a specific threshold. If the probability is higher than that threshold, we can say that the student will pass. However, if the probability is lower, we can predict that the student will fail.

We could assume the function $p(x)$ to be linear but the outputs are probability distribution which is bounded while the function is unbounded. To solve this problem, logit transformation comes into play limiting it between 0 and 1, therefore, we will consider the function as $log\ p(x)/(1-p(x))$.

Setting up a threshold is important for the logistic regression to work as a linear classifier. As an example, let's suppose the threshold is 60% or 0.6 and we have two classes, 1 and 0. The outcome probability needs to be equal to or higher than 0.6 to be classified as 1.

$$y = \begin{cases} 1 & if\ \ p \geq 0.6 \\ 0 & otherwise \end{cases}$$

2.1.2.3 Neural Networks (NN)

Neural networks are algorithms or methods, modeled on the human brain, and, in machine learning, they are designed for recognizing patterns in data. The inspirations for NNs are taken from the human brain imitating closely the signaling of biological neurons. Since they mimic the way neurons in the human brain signal and make decisions, they are also known as Artificial Neural Networks (ANNs).

Artificial neural networks (ANNs) are made up of layers of junctions, comprising input and output layers at the start and end respectively. Between the input and output layers, there are deep layers where the computation happens. The junctions or nodes are connected each with its weight and a threshold that causes the nodes to activate and signal. Having the output of a node higher than a specific threshold, the neuron signal to other corresponding nodes.

Neural Networks learn by analyzing the training data to be able to be used for problem-solving or performing tasks. Once they are trained or they have found the decision boundary, they can perform tasks, such as classification efficiently and at much faster speeds than humans can perform.

Each node in the network can be thought of as itself a linear regression model composed of input data, weights, threshold, and output.

$$\sum w_i x_i + b = w_1 x_1 + w_1 x_1 + \cdots + b$$

$$f(x) = \begin{cases} 1, & \sum w_1 x_1 + b \geq 0 \\ 0, & \sum w_1 x_1 + b < 0 \end{cases}$$

These ANNs learn weights (the parameters of the neural network that transform the input data) which constitute learning. A trained neural network is one where the initially assigned weights are optimized. The optimization procedure starts by initializing random weights that are usually small in magnitude. The weights show the strength of the connection between the neurons and are important in the fact that the higher the weight of a neuron is, the more it contributes to the final output. Given an input to the neural network, it is multiplied with the weights, and if a node's output is greater than a threshold, after the passage down the activation function, it signals to other neurons. The process continues with the higher-weighted neurons signaling to one another in a forward way and the neural networks that process in this manner are known as Feed Forward Neural Networks.

To know an estimation of how far we from our desired solution are, a *loss function* is used. Generally, *mean squared error* is chosen as the loss function for regression problems and *cross-entropy* for classification problems. Let's take a regression problem and its loss function be a mean squared error, which squares the difference between *actual* (y_i) and *predicted value* ($\hat{y}_i$).

$$M_i = (y_i - \hat{y}_i)^2$$

The loss function is calculated for the entire training dataset and their average is called the *Cost function*.

$$Cost = M = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

2.1.2.4 Deep Neural Networks (DNNs)

Deep Neural Networks (DNNs) are Artificial Neural Networks (ANNs) having multiple hidden layers. They have the same component as the ANNs and work in the same way, but, the difference

is that the inputs are passed through several deep or hidden layers before outputting a result. DNNs can contain many more hidden layers, even hundreds or thousands. With the high volume and dimensions of data available nowadays, it would not be possible to use simple ANNs to analyze them and perform tasks efficiently and this is where DNNs shine, being able to explore and scrutinize data much more effectively.

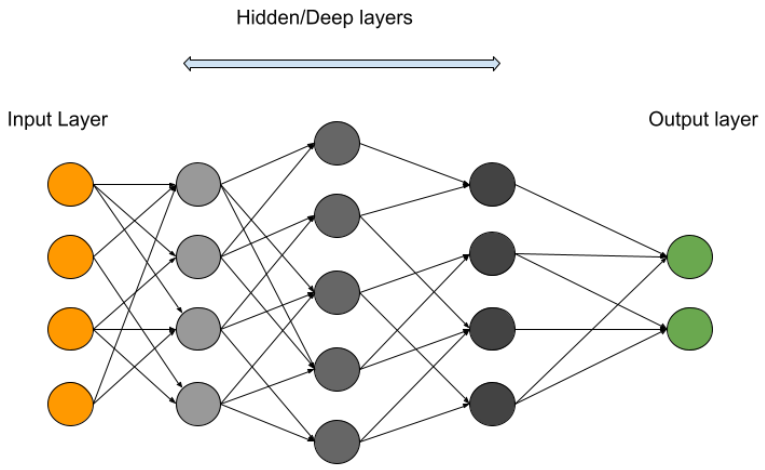

Fig 3: Example of Deep Neural Network Architecture

2.1.2.5 Convolution Neural Networks (CNNs)

Convolutional neural networks (CNNs) are Feed Forward Neural Networks that contain additional layers, known as convolution layers, before the deep or hidden layers. CNNs shine when used for image recognition, as their performance is topnotch with computer vision tasks. These networks harness matrix multiplication, to pinpoint patterns within images.

The convolutional layers are the first layers, after the input layers, where the convolution operations are performed and are the building blocks of CNNs. These layers are comprised of specific size filters applied to an area of the image, calculating the dot product between the input pixels and the filter. The process is repeated, shifting the filter by a specific stride, until the entire image is covered and dot products are calculated. As a result, we get what is known as a Feature Map comprising the output of the filter applied. The feature maps are then transformed using ReLU (Rectified Linear Unit) activation function, mapping the negative inputs as zeros, establishing non-linearity to the neural network model.

Pooling layers, also known as down sampling, are part of convolution neural networks and conduct dimensionality reduction, reducing the number of parameters in the input. Similar to the

convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array.

While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layers are the final layers, before the final output layer, to which the output of the convolution layers is fed. However, adding more convolution layers drastically increase the complexity of the CNN, identifying greater portions of the image. The initial layers take note of low-level features such as edges. As the input data is passed to the later layers, more concrete elements are recognized focusing on high-level features such as shapes, that are specific in order to recognize the input.

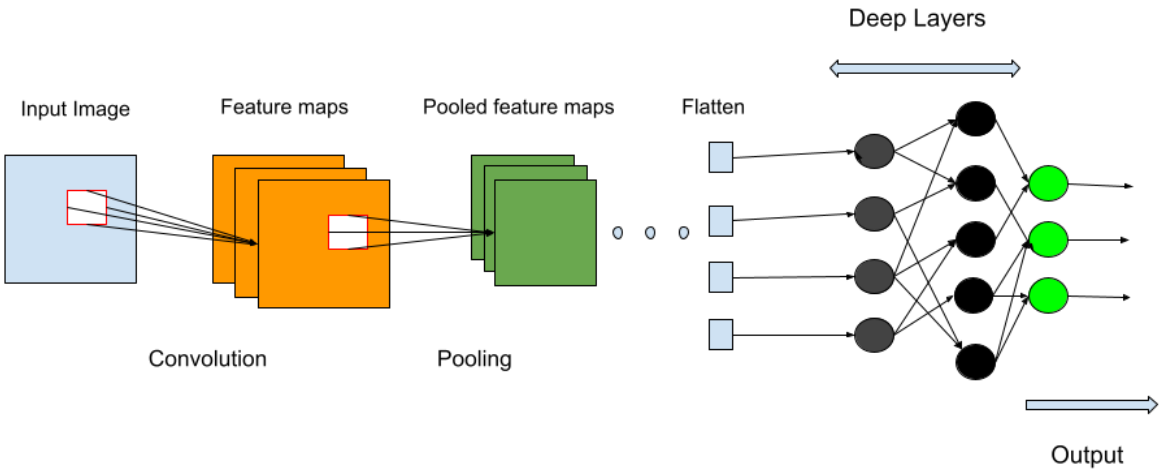

Fig 4: Example of Convolution Neural Network

2.2 Adversarial Machine Learning

Machine learning models can be sometimes overly complicated, due to their ambiguous learning nature, which might lead to a poor understanding of how they output results. Take, for instance, Deep Neural Networks (DNNs) which have unparalleled accurate and state-of-the-art performance on the problems to which they are applied. However, their eloquence, which is the reason for their great performance, can lead them to learn in an explicable way, making it hard to understand their learning manner as well as making it a weakness that can be exploited by an attacker.

But why do these adversarial examples exist? There have been different theories presented and one of the first to put forward was *Szegedy et al*, explaining that deep neural networks learn a discontinuous mapping between inputs and outputs, as well as poor generalization of the model

causing the network to fail to predict correctly when given an indistinguishably perturbed input [5]. However, there is much non-linearity in machine learning, the activation functions like ReLU and Sigmoid are straight in the middle for preventing the gradient to vanish. There are a lot of linear functions all perpetuating one another's input in the same direction, and, applying small perturbations to inputs, accumulates into massive differences on the other end of the network. Another most commonly accepted reason, as argued in [22], could be that the model never fits data perfectly, test accuracy is always lower than perfect, and there will always be adversarial pockets of inputs that exist between the boundary of the classifier and the actual sub-manifold of the sampled data.

2.2.1 Allocation of Attacks

The attacks, against a machine learning model, by an attacker, can be different in different scenarios and is dependent on features, such as the effect on the classifier, the kind of violation, and or the kind of attack. In [23] *Biggio et al* give the taxonomy of the attacks that a machine-learning model might confront. The attacks are based on features such as *influence* (the effect that the attack or the attacker has on the machine learning classifier), *security violation* (the type of violation of the security that an attacker can cause), and *specificity* (what particularly an attacker wants to do).

The *influence* can be of two types, either causative or exploratory. If it is causative, the attacker wants to constrain or put hurdles in the path of the classifier to make it slip away. Otherwise, in the case of exploratory, the attacker doesn't interfere with the classifier, but he tries to gain knowledge about the classifier which might be trained.

The *security violation* can be of three types depending upon the attacker and his motive. An attacker can violate Integrity by fooling the classifier into giving access to the attacker and realizing some malicious inputs into legitimate. Additionally, availability violations can be caused by keeping busy the classifier with unwanted and successfully conducting denial-of-service attacks. Furthermore, a violation of privacy can bring privileged details regarding the classifier into the attacker's hands.

The attack *specificity* stands for if the attacker is focusing on misclassifying an instance into a specific class, targeted, or the attacker just intends to misclassify the instances, exploratory.

2.2.2 Modelling Adversary

Modeling the threats is crucial in understanding the kind of attacks that a deployed classifier might face, the attackers' strategies that he could use, and what could be at risk when the attacks happen. *Biggio et al* [4] proposed a framework for modeling threats against supervised learning, considering the goal of the attacker, his knowledge, capability, and the strategies he might wield to attack.

The attacker's goal is based on two of the features, as described in the previous section, one of which is a security violation and another being specificity. The attackers can have different motives for attacking and depend on the kind of security that the attacker plans to breach as well as if the attacker is considering some specific targets or it is untargeted. An attack could be targeted or untargeted along with breaking integrity, availability, or privacy.

Another point that is worth considering is the attacker's knowledge which is important for deciding what kind of attacks the adversary could carry out.

White-box attacks. This is the situation when the adversary or the attacker has complete knowledge of the model or classifier deployed, including the training data, the architecture of the model, the objective function, as well as the parameters and weights learned, etc. Since the attacker has full knowledge of the classifier, he can carry out an attack fairly easily and efficiently.

Grey-box attacks. These types of attacks come in between white-box and black-box attacks where the attacker has some partial knowledge about the system, such as the feature representation and the algorithm used to train the model but do not have any access to the data used for training as well as the weights learned by the trained model. However, even if the training dataset is not available to the attacker, he can make a synthetic data set made up of instances collected by the attacker and labeled by the target model.

Black-box attacks. For most real-world scenarios, white-box assumptions or complete access to a classifier may be unrealistic, most probably only query access to it. A black-box attack is the kind of attack where the information about the models is hidden, the attacker has no access to it, but only querying access to the models is possible where the inputs are synthetically crafted and queried to the oracle, which acts as an opaque endpoint, to get the outputs predicted by the model.

Nonetheless, the attacker can still use the knowledge of the algorithms that are good for specific tasks, e.g., for image recognition or generally working with images for classification, CNNs are used, so the attacker might guess the trained model, but still, since he has no other knowledge, he cannot guess the architecture of the target CNN.

The attacker's capability is dependent on one of the features that *Biggio et al* proposed, as described above, that is the feature of influence. If the influence of the adversary is causative, he can have control over both the data used for training as well as the data used for testing. Conversely, if the influence is exploratory, the attacker can only exploit the test time data.

2.2.3 Attack Types

The two most common types of attacks that an adversary can carry out, as described in [4], are test time *Evasion* and *Availability* attacks. The evasion attacks are carried out at test time where the adversary influences the test set violating integrity, as opposed to poisoning attacks which can be used for violating integrity as well as availability, where the adversary has an influence over the training set and can inject malicious instances into the dataset used for training to affect the prediction capability of the classifier negatively.

2.2.4 Evasion Attacks

Evasion attacks are the kind of attacks that are carried out at test time since the attacker influences the test data. Given a trained classifier, the attacker manipulates the input data to deceive the classifier into misclassifying them. The formulations below are taken from.[11]

The optimization problem in the case of a white-box evasion attack, given an instance x along with its label y and l being the loss function and x' being the adversarial example can be written as:

$$\max_{x'} \quad l(y, x', w)$$
$$s.t \quad ||x' - x||_p \leq \varepsilon$$
$$x_{lb} \preccurlyeq x' \preccurlyeq x_{ub}$$

Here, since the attack is white-box and the attacker has complete knowledge of the classifier, the parameters are assumed to be known to him. For a black-box attack, since the attacker doesn't have any knowledge of the classifier and in this case, the adversary can make use of the surrogate models for effectively carrying out evasion attacks on the target model leveraging the property of *transferability* [3]. The parameters of the target model are not known to the attacker but by using surrogate models, the parameters of the surrogate model will make up for target model parameters that are unknown.

The instinct behind the evasion attack is that the adversary, instead of minimizing the loss as done in the training phase, maximizes the loss on the perturbed sample with the true label to force the classifier to misclassify it. The perturbation value, usually denoted by ε (epsilon) is the bound on the perturbation and the difference between the natural instance and perturbed sample should not increase the value of ε as well as an additional constraint, known as box constraint, is on the perturbed sample itself bounding its values [11].

2.2.4.1 Adversarial Examples

An adversarial example refers to a specifically engineered input that is designed to look the same as the original input to humans but misleads the machine learning model.

Let x be a natural instance with label y and the classifier under attack be C. An adversarial example is such that:

$$\hat{x} = x + \delta$$
$$C(\hat{x}) \; ! = C(x) = y$$

Where δ is the amount of perturbation or noise that is applied to the image. For instance, when an imperceptible noise is applied to the image of a 'panda', causes the classifier to misclassify it as 'gibbon' [22].

2.2.4.2 Adversarial Examples Generation

Given an input, an adversarial example is generated by applying carefully crafted noise to the input which forces the classifier to make the wrong prediction. But how are the adversarial examples generated? Let us start with what an entity wants to achieve with training, for instance, a neural network. The formulations below are taken from [24].

Let x be a sample and y be its target class. Classifiers are trained by feeding them data, the instances, and the labels, in the form (x, y). After the classifier is trained, and successfully found a decision boundary, the next step is how the classifier performs on unseen data. Let C be a trained classifier, then the prediction function would be of the form:

$$C_\theta : X \rightarrow \mathbb{R}^m$$

Where h_θ maps the input space to the output space which is m dimensional with m number of classes. The vector θ stands for the parameters of the model and is what is typically optimized when training the model.

A loss function is necessary to deduce how well the network model the training data and maps the predictions made by the model and the true labels to a non-negative number.

$$l(C_\theta(x), y)$$

In the function above, $C(x)$ is the predictions of the model and y are the true labels. Since we want to optimize the parameters and minimize the loss, we use it as an optimization problem.

$$\min_\theta \frac{1}{n} \sum_{i=1}^{n} l(C_\theta(x_i), y_i)$$

The problem is solved using an optimization algorithm, such as gradient decent, computing the gradient of the loss, which is calculated by the back-propagation algorithm, concerning the parameters of the model and fine-tuning the parameters in the opposite direction.

For crafting an adversarial example, the opposite, of what is done above, needs to be done. We have to maximize the loss, so the classifier is forced to make the wrong prediction.

$$\max_{\|\delta\| \in \varepsilon} l\,(C_\theta(x + \delta), y)$$

In the above formulation, ε is the allowed perturbation, and, since we want our adversarial image to be close to the natural image, the optimization is one over the perturbation applied to the image.

In literature, there have been different techniques proposed for the generation of adversarial examples. Adversarial examples can be generated using two settings, white box, in which the adversary has access to the model, and, black-box, where the attacker has no access to the model and usually the model querying is allowed but that is limited as well. Below are some of the famous methods proposed for the generation of adversarial examples.

2.2.5 Attack Methodologies

2.2.5.1 Substitute Model Attack

Substitute model attack is a black-box attack methodology where the adversary has no access to the model, i.e., parameters or weights of the network, gradient directions, training set, as well as network architecture in the case of a neural network, and the objective is to train a substitute or surrogate model. The attacker only has query access to the model and exploits the transferability property to carry out attacks. The attack was demonstrated by *Papernot et al* [3]against deep neural networks (DNNs) by training substitute deep neural networks (DNN).

The attack works as follows:

1. A substitute or replicated training set is made by synthetically crafting training samples.

2. The substitute training set samples (X) are fed to the target DNN for prediction and getting the corresponding labels (Y)

3. A substitute DNN model is chosen and trained using the dataset crafted in the form (X, Y)

4. Attack the surrogate network by crafting adversarial examples using a white-box attack method, such as FGSM or PGD. The adversarial examples crafted for the surrogate model are likely to throw away the target model making it misclassify the adversarial inputs.

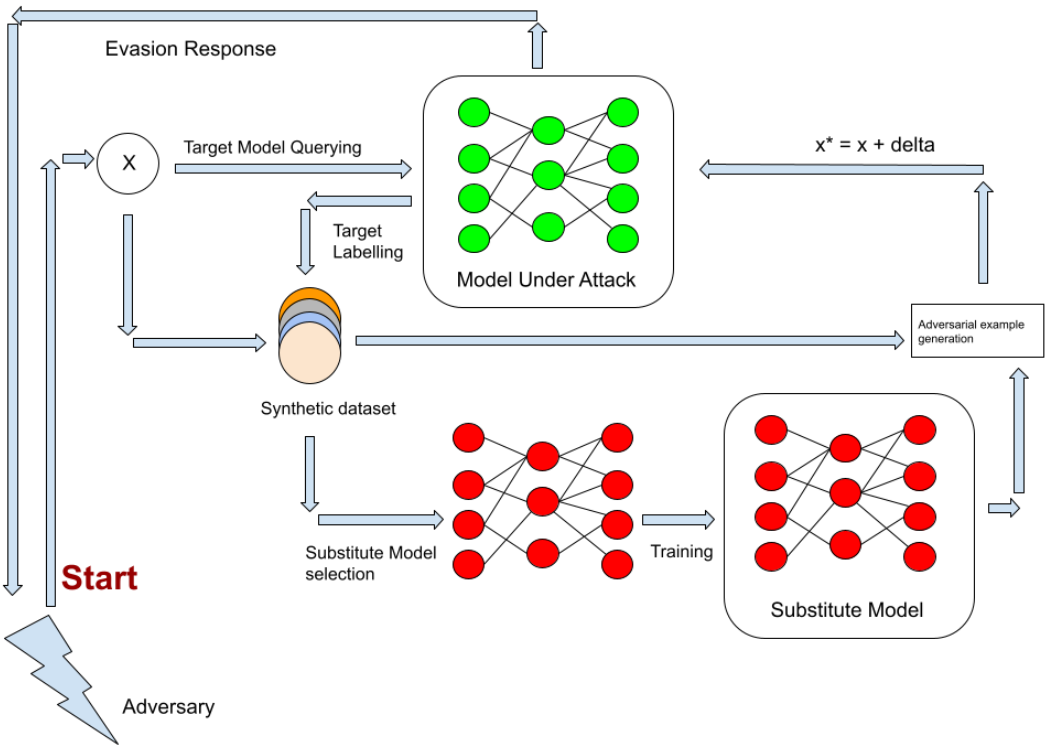

Fig 5: Substitute model Attack

2.2.5.2 Projected Gradient Decent (PGD)

Projected gradient decent is one of the most efficient and effective attacks in the white box settings. Since it's a white-box attack, the attacker has full access to the classifier, most importantly, its gradients. Gradients being vectors are the directions in which the model is sensitive, pointing to the direction of the highest increase of a function. The attacker is much more capable and powerful in white-box settings and can craft adversarial examples fairly efficiently as opposed to black-box or real-world scenarios where the attacker has to rely on the property of transferability to execute attacks. Projected gradient descent can be considered one of the most veritable white-box attacks as it is argued that making a neural network robust against a PGD attacker, makes it robust against all or a wide range of first-order adversaries [25].

PGD, as compared to general *gradient decent*, which moves in the direction opposite to the direction of gradients to minimize a function and has no constraints on variables, as well moves in the direction opposite to gradients to minimize the function but is subject to constraints. With each step taken in the direction negative to the gradient, we have to project onto the feasible set. So, finding an adversarial example with PGD can be thought of as a *constrained optimization problem.* The objective is to find perturbations that maximize the loss of a classifier on inputs while keeping the perturbations in the specified range, ε. As the aim of crafting an adversarial example is to maximize the loss of a model on that sample, PGD attempts to find such perturbations keeping the perturbation size less than or equal to a specified amount referred to as *epsilon* (denoted by ε). However, for adversarial attacks in real-world scenarios, the perturbations applied to the inputs are such that they can not be recognized by humans.

The PGD algorithm is an iterative method that starts by initialization of a uniform random perturbation in the l_p ball around the sample and then iterate the updates [24], written as:

$$\delta := Proj(\delta + \eta \nabla_{\delta} l(h_{\theta}(x + \delta), y))$$

Here $Proj$ denotes the projection onto l_2 or l_∞ and η denotes the step size. Additionally, we have the choice to set the step size as well as the number of iterations.

2.2.6 Defenses

Defense mechanisms have been proposed in former research works to make machine learning models robust or resistant to adversarial attacks. Most of the literature used regularization and data augmentations to defend against, however, they were relatively simple approaches and couldn't defend against strong adversaries. More considered sophisticated methods, such as *Gradient Masking* has also been shown not to suffice for the fact that an attacker can access gradients using other ways [22](e.g., training surrogate models), and *Defensive Distillation* which was shown to be broken by *Carlini and Wagner* [26]. However, the method of adversarial training is the most efficient one which can prove to be a reliable defense for increasing the robustness against adversarial attacks.

2.2.6.3 Adversarial Training

Adversarial training is a defense mechanism attempting to make neural networks robust or improve their robustness against adversarial attacks by training the network with adversarial examples. The problem of adversarial sample crafting, a problem of inner maximization, is to find

the most effective adversarial examples that maximize the loss of the classifier on that sample and is solved by well-designed adversarial attacks such as Fast Gradient Sign Method (FGSM) or Projected Gradient Decent (PGD) [22], [27].

$$\max_{\|\delta\| \in \varepsilon} l\left(C_\theta(x + \delta), y\right)$$

The problem of outer minimization is to minimize the loss during the training procedure. Let D be an adversarial dataset of input-output pairs (adversarial samples along with their true labels), we can write:

$$\min_{\theta} \frac{1}{\| D \|} \sum_{x,y \in D} \max_{\|\delta\| \in \varepsilon} l\left(C_\theta(x + \delta), y\right)$$

Adversarial training is a min-max problem solved by finding adversarial samples which maximize the prediction loss of the classifier while training the same classifier on the adversarial examples crafted in a way that minimizes the training loss.

Adversarial training has been shown to be one of the most effective methods of defense achieving avant-garde performance.

2.2.7 PGD Adversarial Training

PGD adversarial training was proposed by *Madry et al* [25]for the training of adversarially robust networks. The idea is the same as discussed above, to train the model with both natural and adversarial examples generated by PGD. When tested for robustness, adversarially trained CNNs and ResNets [28] prove to be resistant with improved robustness against many first-order L_∞ attacks in both black-box and white-box settings.

For experimental evaluations, we have used PGD as a method of choice for crafting adversarial examples.

Chapter 3

Adversarial Robustness

With the discovery of adversarial examples (*Szegedy et al, 2013*), inputs that are specifically crafted by an adversary to mislead machine learning systems or models, questions have been raised about the security of these models and their use in critical systems. Deep neural networks are shown to be much more sensitive to perturbations due to their natural vulnerability to these perturbations. Although there has been defense research work done in the regard to making the models robust to adversarial attacks, it's still steady and not enough, as compared to the research work that has been done in understanding the adversarial attacks which is very vast. Furthermore, most of the proposed techniques or defense mechanisms to defend against the threat of adversarial examples were soon shown to be either insufficient or invalid for the reason of invalid implementation [29].

Robustness, in general, refers to the ability to cope with unexpected or harsh conditions and remain in good health. When looked at in the view of a system, it can be thought of as the potential of a system to resist disturbance or perturbations without being functionally affected [30]. However, we have to look into robustness in view of adversarial machine learning known as adversarial robustness. Adversarial robustness relates to the measure of the extent to which a network can withstand or is irrepressible against adversarial examples [5].

The use of machine learning models has grown immensely in recent years due to their super-human level performance, flexibility, and adaptability to complex problem-solving situations. However, their flexibility and over-expressiveness can prove to be dangerous as they can be exploited by an adversary. If we wish to deploy a machine learning model (e.g., DNNs) for problem-solving, it is important to consider the security risks and threats that the model might face to take appropriate measures to defend it. Machine learning has been used to focus more on the problems that they are well capable of solving while less attention has been paid to making them secure. Machine learning systems are not free from security threats. Designing or deploying machine learning models without effectively looking or taking into security, in the real world there exists, adversaries, intending to benefit (e.g., monetary), who can cause impairment and force the model to behave inappropriately. Currently, the biggest threat that these models are facing is the threat of adversarial attacks. With the discovery of adversarial inputs, it has been shown how these models can be fooled with imperceptible noise applied to the inputs. Machine learning models, such as DNNs, have inherent or inborn properties that can make them vulnerable to adversarial examples. ML models are used in a wide range of areas including safety-critical systems and using them without establishing proper security could be disastrous. For instance, consider a self-driving car that uses a machine learning model to recognize road signs to take necessary driving actions. An adversary can perturb the road sign to fool the ML model used by the car, by carefully putting simply small patches in desired areas of the road sign, into classifying the sign which it originally it is not [6].

If the ML model used by the car is fooled by the perturbed road sign, which will be fooled most probably if it is not tested in adversarial conditions, it can put the passenger's life at risk.

3.1 Stability vs Robustness

Stability and Robustness are sometimes misunderstood and used interchangeably. Both of these measures are used in terms of defining the security properties of machine learning models (classifiers), however, they are not the same and the differentiation between them is necessary for establishing appropriate security.

A classifier is said to be stable on a particular instance $x \in X$ if it classifies all the possible adversarial examples (that can be crafted for x to mislead it) to the same class as it classifies x to [31]. The definition of stability makes it interesting for the fact that if the classifier is stable on an instance, no adversarial evasion attacks are possible as the classifier will assign all the adversarial exploited inputs the same class label as the original input.

A classifier is said to be robust on a particular instance $x \in X$ if the prediction of the classifier on that instance is correct (the same as the mapping between x and its correct label y found by the target function given a correctly labeled set) as well as it is stable on that instance. That means that, for instance, if the classifier is robust on some set of instances, the classifier will always correctly predict them assigning them a class label to which they belong [31].

Having said that, it is obvious from the definitions that for a classifier to be robust, it needs to predict the correct class label and it has to be stable on all the feasible adversarial examples that are possible for the instances. We can say that robustness implies stability but not vice-versa. Furthermore, just the stability is not enough because the classifier not only needs to assign a constant class label to an instance, but the class label needs to be correct as well. Additionally, the notion of stability might be limited in practice when considering the stability of a classifier on a set of instances. Hence, robustness favors more when we are considering adversarial scenarios as the classifiers' predictions are expected to be correct on an instance along with its stability.

3.2 Motivations for Robustness Evaluation

Assessing the security properties of a model to be deployed is of utmost importance to make sure there isn't any security flaw in the model that could be exploited by an adversary. However, the motive behind accessing the model security could be different for different people who wish to investigate the soundness of a model before stationing the model in the real world. *Carlini et al* [29] argue that there are usually three reasons for evaluating the security of a model considering metric, such as robustness, which is the most widely used metric for establishing the security of machine learning models in the view of adversarial attacks. Evaluating robustness could be for the reason of, defending against adversaries who will attack the deployed model if they find any incentive, testing the worst-case robustness, or measuring the advancements or progress of

machine learning. However, whatever the motivation is, it is clear that robustness evaluation is of utmost importance.

3.3 Robustness Verification vs Robustness Certification

The terms, verification, and certification are sometimes used in an ambiguous manner (interchangeably for each other). Although they are as good as to be used interchangeably, there is a slight difference that forces us to use them in appropriate contexts.

Verification in general hands-on to a mechanism that allows checking formally or informally that a given model respects a certain set of specifications giving rise to the question, "Are we on the right track of building or designing?" Certification instead is the act of guaranteeing that a system or component complies with its specified requirement and is acceptable to be used for operations.

In the view of adversarial machine learning, certification of robustness refers to the surety that for a given classifier and an instance $x \epsilon X$ there does not exist any adversarial example within the l_p bounds that can force the classifier to make wrong predictions. While the verification of robustness can be thought of as a mechanism that gives theoretical guarantees (i.e., using abstract interpretation or formal verification) that the model's robustness will not come down a specific lower bound upon attacks with a specified range or kind of perturbations. The verification approaches can be divided into *complete* and *incomplete* verification. *Complete* verification is the one when the verification process is unable to verify an instance while the presence of an adversarial example around that instance is guaranteed, we can say that the verification process is *complete*. In contrast, if the process results in outputting the instance to be verified, and if there is an adversarial example that is guaranteed to exist around that sample, the verification process is *incomplete*.

The robustness verification of models as stated in [32], such as DNNs, is carried out with the objective of putting forward the robustness' theoretical lower bounds under specific disturbance/perturbation constraints against any adversary. The robust training measures are then applied with the goal to achieve robustness which is in accordance with the theoretical lower bounds. While formal method techniques are used for verification of robustness, they have been used for certification of robustness as we can see in [33] where the authors use abstract interpretation for robustness certification of neural networks. With that being said, the verification can be seen as a part of the certification process because for a model to be issued robustness certification, it needs to be verified first and empirically tested. However, it is not necessary for verification to be a part of the certification task as verification may not work when considered certification in the black-box settings. Robustness certification might be possible in black-box settings, which is the aim of this thesis, where we don't need the help of any formal method verification techniques to certify robustness.

3.4 How to Certify Robustness?

3.4.1 Robustness Measure

Robustness is the characteristic of a system to remain functional without losing its effectiveness when faced with unforeseen circumstances that are outside of its normal conditions of operations. Even though contemporary machine learning models have achieved state-of-the-art performance in solving problems, such as image recognition, they have still not achieved any milestone in being robust (struggling with being robust). These machine-learning models struggle with resilience against adversarial examples, which is an example of a *distributional shift* [34]. The phenomenon of distributional shift is a kind of a problem that arises when the data on which the model is trained is from a different distribution than the data distribution that the model will work on in the real world. Perturbed inputs or adversarial examples are not the only ones that can cause the problem of distributional shift, transformations, such as gaussian noise, can lead to the same problem as well. To increase the robustness of a machine learning model, the normal training techniques don't work, and new objectives and processes are introduced to the training procedure to make the model learn in a way that is resilient to adversarial perturbation. One such example is adversarial training where the models are trained with natural instances as well as adversarial examples.

As stated in [29], it is crucial (while measuring the robustness of a machine learning model that has been robustly trained, i.e., adversarially trained) to define the threat model. A threat model is an essential part of the defense in itself and is important for providing security guarantees the conditions under which a defense or a model with defense is secure. Defining the adversary goals, whether he wants to simply cause the classifier to make errors or he has some target class that he wants the adversarial examples to be misclassified into. It is of utmost importance to constrain the adversary, and his capabilities, for building defenses that are not avoided by an adversary who is unrestricted. Contemporarily, the majority of the defense mechanisms restrict the adversary in perturbing the inputs from the test set to a small amount. Keeping that in mind, an adversarial example (in the view as defined before) would be legitimate only if the similarity or the distance between the natural and the perturbed image is in the specified ball of radius, known as epsilon. A careful and thorough explanation of the adversary's capabilities can help measure robustness well. Robustness is usually computed over the dataset and can be between 0 and 1 showing a percentage of instances the classifier can classify correctly as well as every viable adversarial example (for each of the instances of the dataset) is not able to mislead the classifier in assigning a wrong class.

3.4.2 Existing Robustness Certification Approaches

Accessing security is not easy for the fact that the risk, adversarial risk (Madry et al, 2017), can be theoretically calculated while practically, the exact value cannot be calculated leading to an approximation of this quantity because of the dependence on the capability of an adversary. The knowledge of the adversary is either a white box, black box, or grey box. It would be wrong to

assume that the attacker might not know the details of the model or the datasets. A defense mechanism that hides their data secret (e.g., model, data, etc.) must show that the secret is easily replaceable and that the secret is not extractable [29].

In the traditional software development process, *formal methods* (a set of mathematical and logical techniques to prove the correctness of software) are used for providing mathematically proved guarantees on the software/system behavior. The formal method is broadly divided into complete formal methods and incomplete formal methods [35]. The difference between complete and incomplete formal methods is that complete methods provide soundness and completeness guarantees but require a large amount of time making them unscalable to bigger neural networks. However, incomplete formal methods provide only soundness guarantees but with the advantage over complete formal methods that it requires very little time and is scalable to larger neural networks. An extended amount of work has been done for traditional software systems leading to the development of verification tools that are well-planned, efficient, and organized. However, if we look at the use of formal methods for the verification of machine learning systems, the progress is steady, nonetheless, efforts have recently been made for adapting formal methods to be used for the verification of machine learning. Take for instance the work done in [36] for neural networks, based on the *Satisfiability Modulo Theory* (SMT) which comes under the category of complete formal methods, and the task of verification is taken as a problem of constrained satisfiability. The authors demonstrated that with the sigmoid function, the output of a feed-forward neural network with just dense layers is always between the specified bounds. In their approach, the activation function was encoded as linear approximation functions with some specified intervals and feeding them to an SMT solver along with the constraints for finding any counter-example if any exist. If counter-examples are found, the intervals are changed to the ratio between the previous interval and a refinement parameter.

Existing research works have proposed ways to certify the robustness of machine learning models with most of the work done on neural networks, such as DNNs, for their use in the vast majority of fields. Take, for instance, the work done in [37]for robustness certification of neural networks. The authors introduce a system named *RefineZono* which works by a combination of complete and incomplete verification techniques for the certification of neural networks. The formal methods used are *Mixed Linear Integer Programming* (*MILP*) which works by transforming verification into a mixed linear integer program, Linear Programming (*LP*) relaxation which is used for hard optimization problems, and *Abstract Interpretation* which allow comparing semantics mathematically to prove security properties. The approach is that the layers are encoded using the formal methods along with the constraints and the bounds are refined for the first layers using MILP relaxation, and the second layer using LP relaxation. However, for the last layers, abstract interpretation is used without any additional refinement. The works [33], [38], [39], take an incomplete formal method of abstract interpretation while [40] take a complete formal method MILP into account for robustness certification. In [38], the authors propose certification of robustness using randomized smoothing arguing that turning a classifier (which performs well under the Gaussian noise) into its smooth version using randomized smoothing proves to provide robustness guarantees over l_2 norm.

3.4.3 Comparing Our Work with Existing Literature

Adversarial perturbations and the attacks originating from them are real threats. Also, when considering black-box settings (where the attacker has no knowledge of the classifier, its parameters, the training data, and procedure, etc.), the property of transferability (having vast empirical evidence to support [3], [6]) is a fact as well, along with the reasons behind it [11].

The common thing about most of the previous research works is that their proposed mechanisms try to certify robustness with the help of formal methods, formal verification, along with making slight changes to the way such as layers in the neural network work. The majority of the works are using incomplete formal methods as a part of their robustness certification process to deal with the problem of scalability which the complete formal methods suffer from. Since white-box access to the model is needed, its typically infeasible to verify large neural networks. The robustness certifications are usually in the form of verification of security properties that are motivated by these slight changes to the structure or the way the layers in neural networks work. Furthermore, most of the works are trying to norm-bound an adversary and putting constraints to be able to not make any changes greater than the specified size of perturbations. What has been done is proposing slight changes while trying to theoretically verify it followed by some empirical guarantees to be secure within the norm bounds.

Because of the large number of adversarial threats, establishing the security properties of a classifier is of utmost importance and the metric used in the scenarios of adversarial attacks is usually adversarial robustness. What distinguishes our work from most of the work previously done is that we want to certify robustness in real-world or black-box settings. We do not provide any defense mechanism that would protect the classifier from attacks, instead, what are we doing is we intend to certify classifiers for robustness which may or may not be trained with the current state-of-the-art defense mechanism, such as adversarial training, and deployed for problem-solving, in the black-box settings using surrogate models. Since the attacks transfer, we can utilize the transferability information along with leveraging well-known neural network models as surrogates to certify the robustness of the target model. The main advantage is that we do not face the issue of scalability. Since our work is free from verification techniques (complete verification which suffer from scalability), as well as we are not accessing the target model directly, we are able to handle machine learning models, such as deep neural networks, which could be very large.

3.5 Why is black-box robustness Certification challenging?

Black-box setting refers to when the adversary or the attacker has no knowledge of the target model, including the architecture, parameters and weights, training data, etc. It is clear from the fact that since nothing is known about the model in black-box settings, it would be hard to perceive the information of robustness. In the real-world, we do not have any access to the deployed model

and the only access to the model is in the form of querying the model to get predictions, and that as well is in the form of an API where the queries are sent to the oracle.

To get accurate evaluations of the robustness of any model, it would be necessary to search for the minimum adversarial perturbation to attack the model that will significantly degrade the model's performance. There are possibilities of many different attacks that can prove to be strong, however, optimization-based attacks are the most powerful among them that utilize gradients of the loss function [29]. The gradient-based algorithms to generate adversarial examples, such as Projected Gradient Decent (PGD) [25] which can generate maximum confidence adversarial examples, work on the gradients of the model, and without gradients, it would not be possible for PGD to find adversarial perturbation that is confident enough to mislead the classifier and forcing it to misclassify an input for which the perturbation is found. Following the reasoning, as the gradients are not available to find any adversarial sample in black-box settings, which might mislead the model, it's challenging to access the model for robustness.

Apart from the inaccessibility to gradients in black-box settings, in the case of a surrogate model attack, the surrogates need to be as close as possible in architecture for the attacks to transfer more to infer something beneficial. Also, the huge amount of hyperparameters makes the certification in black-box settings complicated because a slight change can induce a different behavior. Many testing methods work well for traditional software systems, nevertheless, these testing methods might not prove to be useful when considering machine learning (since neural networks in themselves work as black boxes lacking transparency) and are under-approximated for the fact that the inputs space might be infinite and unable to be explored fully (unless the feature space is finite). Furthermore, the verification techniques formally guarantee robustness, but the problem is that they do not scale well to larger complex models. Additionally, the robustness measure itself is not reliable because of its dependence on the choice of inputs. Even if the robustness guarantees for a specific input are provided, that does not assure the robustness of any other instance from the same distribution (providing only local robustness guarantees).

Chapter 4

Robustness Certification

Neural networks are a powerful way to solve computer problems that leans from data and act in a way mimicking human behavior. Deep learning algorithms based on artificial neural networks (ANNs) have achieved great success for their powerful nature to analyze data and state-of-the-art super-human level performance in domains such as image classification. However, the performance of these models in the presence of perturbations cannot be even compared to an uninformed human [29]. The important thing is the consistency of performance which means that these models when used in real-world settings with the presence of adversaries, who could make the models fall astray using carefully crafted adversarial examples, should behave the same as they perform with natural data. The consistency of performance is even more important when using them in safety-critical systems, such as self-driving vehicles or aeronautics, etc., because, in safety-critical systems where the room for error is extremely low and inconsistent, behavior can lead to disaster (risk of harm to human life and/or system).

Robustness measures the extent to which a model can incessantly keep its performance consistent, as it performs on the natural data, without being fooled by adversarial examples. Robustness is important for the fact that these models are under constant threat of adversarial attacks which could force them to behave unexpectedly. Machine learning models, such as DNNs, can be made robust to attack by developing defense mechanisms with a thorough evaluation of the threats and the way the threats affect the models. However, although the defense designed can be argued to increase robustness in the view of some kind of attacks, we don't know how the defense will perform as we don't have robustness guarantees of the model when considering real-world situations or black-box settings where the attacker doesn't know anything about the model but can still attack using black-box attack methods. Take for instance, gradient masking where the concept is to mask gradients, so the attacker is not able to access gradients, nonetheless, this mechanism to increase robustness was shown to not suffice the fact that there exists a possibility of black-box attacks, such as square attacks [41], where the attacker doesn't need gradients to attacks or, substitute model attacks, where the adversary train substitute models to approximate the target model gradient information [22].

Robustness certification is crucial in the sense that we have a surety of the robustness of the models, we wish to deploy, for generating trust in them and have peace of mind that their performance will not degrade, keeping adversaries and adversarial attacks in mind, even in the worst-case scenarios. In this chapter, we explain our approach to robustness certification in detail.

4.1 Framework for Robustness Certification

The framework developed uses the black-box technique of attacking the target model via multiple surrogate models. The robustness information for each surrogate is taken independently and combined to approximate the robustness of the target model. The target and surrogate models are trained and then fed to the framework for robustness certification of the target model. The first step is to attack the models via Projected Gradient Decent (PGD).

4.1.1 Attacking the Models

The classifiers are attacked using the Projected Gradient Decent (PGD) which is a white-box algorithm for generating maximum confidence adversarial examples utilizing the gradients of the model under attack (since we are using surrogate models to attack the target model, the surrogate model gradients are available to the attacker). Adversarial examples are generated using PGD and the surrogate model performance is tested on them. For each instance of adversarial examples, the prediction of the surrogate models is recorded in the form of success or failure.

4.1.1.1 Attacking Surrogates

Let D_{test} be a test set and x be an instance, such that $x \in D_{test}$, belonging to a class label $y \in Y$ where Y is the output space, S a surrogate classifier, x' be an adversarial sample where $x' = x + \delta$ and δ is the perturbation or noise found by the PGD algorithm, and R is a result function that maps the correct or incorrect prediction to a result space. The result of the surrogate models on each instance becomes:

$$R(S(x')) = \begin{cases} 1 & if\ S(x') \neq y \\ 0 & otherwise \end{cases}$$

Where $S(x')$ is the prediction of the surrogate model on the adversarial example and $S(x)$ is the prediction of the classifier S on a natural instance. The result is 1 if the prediction of the surrogate classifier on the adversarial sample is not equal to the true label that the instance belongs to.

In the same manner, the surrogates are tested on each of the instances belonging to the test set and the results are recorded.

4.1.1.2 Evaluating Transferability

When attacks for the recording of the attack results are done, the next step is to attack the model for evaluating the transferability. The measure of transferability will tell us how much of the

adversarial samples crafted transfer to the target model. This time, the full test set is not needed to be perturbed, instead, we first find the performance of the surrogates on the test set in the form of accuracy. We take all of the instances that the surrogate classifiers predict correctly. We then use these instances to craft adversarial examples for the surrogate models. After the adversarial examples are crafted, we move to the performance testing of both the surrogate, for which the adversarial samples are crafted, as well as the target model.

Let $D_{inc} = \{1, \dots, n\}$ be the set of natural unperturbed instances that the surrogate classifier misclassifies. The set of unperturbed natural instances that the classifier correctly classify becomes:

$$D_c = D_{test} - D_{inc}$$

The PGD algorithm will find the adversarial examples for this set D_c for a given surrogate classifier and tested upon the same surrogate as well as the target model to get the cardinality of instances that they misclassify. Let $x \in D_c$ be an instance with the correct label y, T be the target classifier, the transferability of adversarial samples to the target model is then computed as:

$$\mathcal{T} = \frac{Number\ of\ misclassifications\ by\ Target\ Model}{Number\ of\ misclassifications\ by\ Surrogate\ Model}$$

The same procedure is followed for every surrogate classifier that would be used against the target model to get the measure of transferability (the number or proportion of adversarial examples that transfer to the target model).

4.1.1.3 Combining Results

Since the idea is to use an ensemble of surrogate models against the target model, each of the surrogate models will have its result list which needs to be combined to get one single result list that can be compared to the target results. Since the results are in the form of 1 and 0, we will use logical gates to combine the results into one.

- **AND gate.** Here we will perform tuple-wise logical conjunction to combine the results of surrogates recorded as a result of the PGD attack. For instance., in the case of two surrogates against the target, let the results of the two surrogates be R_1 and R_2, the combination becomes:

$$R_1 = [1,\ 0,\ 1,\ 0]$$
$$R_2 = [1,\ 0,\ 0,\ 1]$$

$$AND = [1,\ 0,\ 0,\ 0]$$

The idea is that if both the surrogates fall prey to the adversarial examples, only then we will take the combined result for that instance as 1, else 0.

- **OR gate.** The logical OR operation will be performed in order to combine the results of surrogates. For example.,

$$R_1 = [\,1,\ 0,\ 1,\ 0]$$
$$R_2 = [\,1,\ 0,\ 0,\ 1]$$

$$OR = [\,1,\ 0,\ 1,\ 1]$$

In this case, even if one of the classifiers is forced to misclassify an instance x_i, doesn't matter if the other surrogate classifier predicts it correctly, the combined result would be taken as 1 (misclassification on that instance).

- **Majority Voting.** In the case of majority voting, the result would be taken supported by the majority. However, in the case of two surrogates, the result of the majority voting would be equal to the AND gate. As an example., let's consider a third surrogate classifier along with the previous two and let the result of the third classifier be R_3, the combination would be:

$$R_1 = [\,1,\ 0,\ 1,\ 0]$$
$$R_2 = [\,1,\ 0,\ 0,\ 1]$$
$$R_3 = [\,1,\ 0,\ 1,\ 1]$$

$$Majority\ Voting = [\,1,\ 0,\ 1,\ 0]$$

- **Weighted Majority Voting.** Here the idea is almost the same as majority voting but with the addition of weights. Here, the weights are the measure of transferability for each

classifier which is multiplied by the result list. One important thing here is that for weighted majority voting, the resulting space consists of -1 instead of 0. Since we will multiply the transferability measure with each respective result list, we don't want any instance result of a surrogate classifier to become 0 when multiplied by 0. Furthermore, we will do tuple-wise addition to get a single result list. For instance, let the transferability measure of the first classifier be $\mathcal{T}_1$=0.2, the second classifier be $\mathcal{T}_2 = 0.5$ and the third classifier be $\mathcal{T}_3 = 0.8$. Weighted majority voting would be:

$$\begin{aligned} R_1 &= [1, -1, 1, -1] * \mathcal{T}_1 \\ R_2 &= [1, -1, -1, 1] * \mathcal{T}_2 \\ R_3 &= [1, -1, 1, -1] * \mathcal{T}_3 \end{aligned}$$

Which after multiplication with the transferability measure becomes:

$$\begin{aligned} R_1 &= [0.2, -0.2,\ 0.2,\ 0.2\ ] \\ R_2 &= [0.5, -0.5, -0.5, 0.5] \\ R_3 &= [0.8, -0.8, 0.8, -0.8] \end{aligned}$$

Now we will do tuple-wise addition. After the addition operation, the result will look like this:

$$Weighted\ majority\ voting = [1.5, -1.5, 0.5, -0.1]$$

If the resulting number is positive or greater than 0, we will take the combined result as 1 for that instance, otherwise 0. The final result using weighted majority voting will be:

$$Weighted\ majority\ voting = [1,\ \ 0,\ \ 1,\ \ 0]$$

4.1.1.4 Attacking Target Model in White-Box Setting

For the purpose to certify the robustness of the target model, we will suppose the target model is available to be attacked in a white-box setting. It is important to note that the assumption of accessibility to the target model is just for our experimental evaluations, while the target model

may most probably not be accessible in the ideal black-box settings or real-world scenarios. Since here, the target model is available to us for empirical assessment in white-box settings, we have access to the model gradients so we can use PGD to generate adversarial examples for the target model and see the performance. In the same way, as we attacked and recorded results for the surrogate models, we will check the prediction of the target model on each instance getting a result list in the form of 1 if the prediction is wrong, else 0.

Let $x \in D_{test}$ be an instance, with class label $y \in Y$, T be a target classifier, x' be an adversarial sample where $x' = x + \delta$ and δ is the perturbation found by the PGD, and R is a result function that maps the correct or incorrect prediction to a result space. We can write:

$$R(T(x')) = \begin{cases} 1 & if\ T(x') \neq y \\ 0 & otherwise \end{cases}$$

Being able to access the target classifier would be beneficial for the fact that we will have a result list for the target model (collected as a result of predictions on the adversarial examples set crafted specifically for the target model using PGD) which can be directly compared with the combined result of the surrogate models. In addition, it would allow us to compute the accurate measure of robustness for the target model (since the target model is accessible) which in turn would be beneficial for comparing it with the robustness measure found by the surrogate models for the target model.

4.1.2 Computing the Robustness

Having collected the combined result list and the result list for the target model, we can now compute robustness. It is crucial to mention here that we aim to approximate the robustness of the target model in the best way possible via surrogate models. The robustness of the target model is computed in two ways. First, via the target model which is the actual robustness because we are computing it using the result list obtained by directly attacking the target model. Secondly, we compute the robustness via the surrogate models.

4.1.2.1 Robustness via Target Model

For computing robustness, we will use two result lists. First, the list of results would be collected by checking the predictions of the target model on the unperturbed test set while second, the list of the result obtained via checking the predictions of the target model on the adversarial samples crafted using the same test set.

We will compare the two result lists. Suppose a natural instance is misclassified by the target classifier, we will not look at the result of the target classifier on the adversarial example that was

crafted for the target model using the same instance for the fact that it's already been misclassified in the natural form, so the classifier is not robust on that instance. Instead, if the classifier classifies a benign instance correctly and the target classifier also classifies the adversarial sample crafted using the same instance correctly, we will consider the classifier to be robust on that instance.

Let $x \in D_{test}$ be an instance of the test set with the target label y, x' be the adversarial sample crafted for the target model making use of PGD using the same instance, and T be the target classifier, T is considered robust on an instance x if it also correctly classifies both x and x'. The robustness over the dataset is computed by checking the robustness of the target model on every instance and dividing the number of instances the target classifier is robust on by the total number of instances in the dataset.

$$\mathcal{R} = \frac{Number\ of\ instance\ the\ classifier\ is\ robust\ on}{Total\ number\ of\ instance\ of\ the\ dataset}$$

4.1.2.2 Robustness via Surrogate Models

The robustness via the target models is computed is the same way as it is computed for the target model. However, the difference is that in the previous section, the robustness is computed directly using the target model assuming it to be accessible while here, the robustness is computed using the combined result lists that we obtained for the surrogate models. Nevertheless, since we propose the use of multiple models, their independent results are combined using the logical gates to get single combined results. As a result, we will have four robustness measures for the fact that the robustness is measured by comparing the four combined lists (one for each gate) with the list of predictions (predictions of the target model) on an unperturbed test set.

4.1.3 Finding the Optimal Result

After we have the robustness results for all four gates, the next step is to compare them with the actual robustness that is computed via the target model and pick the result which best approximates the robustness of the target model.

The most important part is choosing the result with the gate which finestly approximates the robustness of the target model because the results depend on the choice of models and different models will have slightly different results. For instance, in the case of using surrogate neural networks against a target neural network, to get the best robustness approximation results out of surrogates, one of the important points is to construct surrogate models that are close in architecture to the target model. However, the key to understanding which gates best approximate the robustness of the target model in different situations, the hyperparameters used for the surrogate models need to be understood because the hyperparameters do affect the robustness result we obtain via the four gates. We will discuss the factor affecting the results in detail in chapter 5.

4.2 Attacker's Scheme

An analyst or an attacker can use our framework to effectively evaluate the robustness of a target model in black-box. Generally, in real-world scenarios or black-box settings, access to the model is not possible and usually, there is only query access where the model can be queried to get predictions on some inputs. So, the attacker can use surrogates to approximate the robustness of the target model.

4.2.1 Training Surrogates

It is possible to craft a synthetic dataset for training the surrogates for attacking the target model as shown in previous research [3]. The attacker can generate synthetic samples and get them labeled by the target model by querying to get a training set. The surrogates can be chosen and trained with the dataset crafted.

4.2.2 Attacking Target via Surrogates

The attacker can use the trained surrogates to attack the target model. A test set can be crafted in the same way as the training set and used to craft adversarial samples. The surrogates are available to the attacker locally to it is possible to use PGD to craft adversarial samples for surrogate models and then query them to the target model in return to infer transferability information. The transferability can tell how close the surrogates used are to the target in architecture or for instance, the parameters used. If the attacks are not transferring well, the dataset crafting, and surrogate model selection can be redone in order to make sure that the trained surrogates approximate the target model well. However, one thing to notice is that the query access to the target model might be limited, but it is possible to overcome the query limitation problems [42], [43].

4.2.3 Approximating Robustness

After having attacked the target models via adversarial examples crafted for surrogates that transfer well and calculated the transferability, the prediction results of the target model and surrogates can be recorded in a list form on the test set. In the same way, as stated in section 4.2.2.2, the robustness can be computed by getting four approximations of robustness (one for each logical gate) of the target model via surrogate models.

4.2.4 Choosing the Best Robustness Result

The last step is to find and choose the robustness result of a gate that best approximates the robustness of the target model which is dependent on models used as surrogates and their hyperparameters and most importantly their transferability. We will discuss the patterns to look for as well as the transferability measure in detail in the next chapter in order to be able to choose the right gate approximating optimally the target model robustness.

Chapter 5

Experimental Evaluations

In this chapter, we present the experiments, conducted for the development of this thesis, and the experimental results as well as discuss how the empirical evaluations conform with the proposed theory.

5.1 Preliminaries

5.1.1 Choice of Machine Learning library

We have used SecML [44] which is a library for security evaluations of machine learning models and serves the purpose. It has built-in standard datasets allowing for efficient training of machine learning models. In the bargain, it has implementations of the most popular attacks against machine learning models, such as Projected Gradient Decent (PGD), for effectively carrying out evasion attacks.

5.1.2 Dataset

The dataset used, for empirical evaluations, for the development of this thesis is the MNIST database [45] (Modified National Institute of Standards and Technology database). MNIST is a dataset of handwritten digits that was originally crafted out of the old NIST database. The dataset consists of 70,000 grey-scale images (10 classes from 0 to 9) divided into 60,000 training and 10,000 testing instances. Each image is 28*28 black and white image.

Image and optical character recognition are important problems in machine learning. Several standard datasets to experiment with machine learning algorithms have been proposed, out of which MNIST is the most widely used dataset for the fact that it is simple as well as to promote research so researchers can experiment on a common dataset and able to compare their results to one another [46]. The popularity of MNIST can be seen in the fact that almost every machine learning library has it built-in.

5.1.3 Target and Surrogate Models

There have been different machine learning models used for experimenting including simple well-known models, such as Support Vector Machines (SVMs) and Logistic Regression (LR) which are only used as surrogate models, as well as more complex models such as Neural

Networks (NNs). The NNs used include shallow neural networks and deep Convolution Neural Networks some of which are taken from [42] and used as target as well as surrogate models. For instance, we have used a well-known convolution neural network, such as the Caffe version of LeNet referring to LeNet-5 [47], so that the results are comparable to other research works.

The simple models, only used as surrogates, along with their regularization factor C are reported in the table below:

	Hyperparameters		
Models	**C**		**Max Iterations**
Support Vector Machine (SVM)	0.1	1.0	-
Logistic Regression (LR)	0.1	1.0	100

Table 5.1: Models only used as surrogates

As it is evident from the table, we will use both models in two different versions, one highly regularized and another slightly less regularized model. Regularization here is important because as we can see in the previous work, highly regularized or lower complexity models tend to perform better than high complexity or less regularized models [11]. Since we will be using multiple surrogates against the target model, we want to try different combinations of highly regularized and less regularized models to see how their regularization affects the robustness result on each logical gate calculated via surrogates.

Apart from the simple models presented above, the majority of the machine learning models used for experiments are neural networks, specifically some shallow neural networks and convolution neural networks. Presented below in the tables are neural networks that we have used for empirical evaluations:

Model 1	**Model 2**
FC (10) + ReLU FC (3) + SoftMax	FC (50) + ReLU FC (3) + SoftMax

Table 5.2: Shallow neural networks used for experiments

Model 3	Model 4
Conv(32, 3, 3) + ReLU MaxPool(2, 2) Conv(64, 3, 3) + ReLU MaxPool(2, 2) Conv(64, 3, 3) + ReLU MaxPool(2, 2) FC(1024) + ReLu FC(3) + SoftMax	Conv(32, 3, 3) + ReLU MaxPool(2, 2) Conv(64, 3, 3) + ReLU MaxPool(2, 2) Conv(128, 3, 3) + ReLU MaxPool(2, 2) FC(512) + ReLu FC(128) + ReLu FC(3) + SoftMax

Table 5.3: CNNs used in this work.

CNN	Model A	Model C	LeNet
Conv(32, 3, 3) + ReLU Conv(64, 3, 3) + ReLU MaxPool(2, 2) Dropout(0.25) FC(128) + ReLu Dropout(0.5) FC(3) + SoftMax	Conv(64, 5, 5) + ReLU Conv(64, 5, 5) + ReLU Dropout(0.25) FC(128) + ReLu Dropout(0.5) FC(3) + SoftMax	Conv(128, 3, 3) + ReLU Conv(64, 3, 3) + ReLU Dropout(0.25) FC(128) + ReLu Dropout(0.5) FC(3) + SoftMax	Conv(20, 5, 5) + ReLU MaxPool(2, 2) Conv(50, 5, 5) + ReLU MaxPool(2, 2) FC(500) + ReLu FC(3) + SoftMax

Table 5.4: Additional CNNs used in this work. Taken from [63]

There are two shallow neural network models presented in table 5.2 named Model 1 and Model 2. For evaluations, Model 1 is only used as a surrogate model while Model 2 is only used as a target model. The same goes for the two convolution neural network models in the next table. In table 5.3, Model 3 is only used as a surrogate model while Model 4 is used only as a target model.

However, in table 5.4, the four neural networks mentioned are taken from previous work [42] for the reason to make our work comparable with other previous research work.

5.2 Experimental setup

Before presenting the experimental results, here we talk about the dataset partitioning, the accuracy of each trained classifier, and the methods of crafting adversarial attacks. The experimental setups are as follows:

5.2.1 Dataset Splitting

The MNIST dataset is divided into training, with 60,000 instances, and a test set consisting of 10,000 instances. However, we are not using the full training and test set. We are considering the sets consisting of three digits, 0, 3, and 7. We call the training set D_{train} comprised of 18,319 examples and the test set D_{test} made up of 3,018 examples. We won't divide D_{test} set as we will use it to craft adversarial examples against the surrogate models to get prediction results as well as to check if they transfer to the target model. Moreover, we will also use the test set to directly attack the target model crafting adversarial examples for it and getting a prediction result list. The training set instead is partitioned into $D_{T-train}$ including 12,823 samples, for training the target model, and $D_{S-train}$ comprising 5,496 samples, for training the surrogate models, with a 70:30 split ratio where 70% of the instances of the original training set are reserved for training the target model and 30% for training the surrogate models. The training set is partitioned using stratified random sampling to maintain the original class distribution. Moreover, the training and test sets were normalized between 0 and 1.

5.2.2 Training

5.2.2.1 Surrogate and Non-Adversarial Target Models

The simple models, as can be seen in table 5.1, are trained using $C = 0.1$ as well as $C = 1.0$ and the maximum iteration for Logistic Regression in both cases is 100. The table below shows the accuracies of the trained models on the natural or benign test set.

	Hyperparameters	
Models	**C= 0.1**	**C = 1.0**
Support Vector Machine (SVM)	98.41	98.28
Logistic Regression (LR)	98.14	98.38

Table 5.5: Accuracy of simple models on the benign test set

Both of the models have been able to achieve more than 98% accuracy on the benign test set when trained with different values of hyperparameters.

As for the more complex neural network models, Model 1 from table 5.2 was trained using Stochastic Gradient Decent (SGD) as an optimizer for 15 epochs with a learning rate of 10^{-2}, momentum 0.9 with 8 instances per batch. Model 2 instead was trained for 10 epochs with Adam optimizer with a learning rate of 10^{-2} and batch size of 20. The convolution neural network models, presented in table 5.3, were both trained using SGD optimizer, however, Model 3 was trained for 15 epochs with a batch size of 32 while Model 4 was trained for 10 epochs with a batch size of 20. The table below shows the accuracy of these models on the natural test set.

	Models			
	Surrogate		**Target**	
Hyperparameters	**Model 1**	**Model 3**	**Model 2**	**Model 4**
Epochs	15	15	10	10
Batch size	8	32	20	20
Optimizer	SGD	SGD	Adam	SGD
Learning rate	0.01	0.01	0.01	0.01
Momentum	0.9	0.9	-	0.9
Accuracy	98.67	99.67	99.20	99.67

Table 5.6(a): Accuracies of neural network models

It can be noticed that only Model 1 achieved less accuracy as compared to the other three, however, it's not very bad considering it's a very small shallow neural network with just 10 filters. Other than that, the other three models achieved accuracy higher than 99% with Model 3 and Model 4 achieving the same accuracy. Mention again, Model 1 and Model 3 were used only as surrogate models while Model 2 and Model 4 were used as target models.

The models mentioned in table 5.4, have been used as surrogates, as well as some of them, have been used as target models. The table below shows which of them has been used as surrogates as well as the target model.

Table 5.6(b): Accuracies of neural network models

	Models					
	Models as surrogates				Models as Targets	
Hyperparameters	**CNN**	**Model A**	**Model C**	**LeNet**	**Model C**	**LeNet**
Epochs	15	15	15	15	12	12
Batch size	32	32	32	32	20	20
Optimizer	SGD	SGD	SGD	SGD	SGD	SGD
Learning rate	0.01	0.01	0.01	0.01	0.001	0.001
Momentum	0.9	0.9	0.9	0.9	0.9	0.9
Accuracy	99.60	99.50	99.47	99.57	99.77	99.87

5.2.2.2 Adversarial Trained Target Models

Apart from conducting experiments on non-adversarial trained target models, some of the target models presented in section 5.2.2.1 were adversarially trained to expose the performance of surrogates in approximating the robustness of the target model which are adversarially trained. For adversarial training, the method adopted was PGD Adversarial Training [25]. Training adversarial samples were crafted for target models using the set $D_{T-train}$, considering the l_2 norm bound, and the models were trained on them. The table below shows the models which were adversarial trained along with their accuracy on a natural test set.

	Adversarial Trained Target Models		
	Model 4	**Model C**	**LeNet**
Accuracy	99.80	99.73	99.90

Table 5.7: Adversarial-trained target models

In the table above, it can be seen that with adversarial training, the accuracy on the benign test set has been improved for Model 4 and LeNet. However, for Model C, the accuracy almost remains the same with a very slight fall in accuracy.

5.2.3 Evasion Attacks Crafting

We adopt Projected Gradient Decent (PGD) [25], described in section *2.2.5.2* to craft adversarial attacks for the surrogates as well as the target model. The adversarial samples were crafted with three levels of perturbations, $\varepsilon = [1.0, 1.5, 2.0]$ considering the l_2 norm.

5.3 Evaluations

The experiments are carried out with a number of surrogate models against target models (in singular) with a surrogate grouping of three, two, and one models. We run a total of eight sets of experiments testing three as well as two surrogates against a target model. A target model can be adversarial-trained or non-adversarial trained, so we experiment with both adversarial trained as well as non-adversarial-trained target models. We first present, in section *5.3.1*, three sets of experiments where the surrogate models are used against a non-adversarial trained target model. The rest, in section *5.3.2*, belong to the category of experiments where the target model is adversarially trained.

5.3.1 Non-Adversarial Trained Targets

5.3.1.1 Experiments 1

In the first experiment, SVM, LR, and Model 1 were used against Model 2. Attacks were crafted for the three surrogates, with perturbation $\varepsilon = 1.0$ and $\varepsilon = 1.5$, and then tested on the target model.

The table below shows the measure of transferability of the surrogates to the target model.

	Transferability				
	C=0.1		**C=1.0**		
$\boldsymbol{\varepsilon}$	**SVM**	**LR**	**SVM**	**LR**	**Model 1**
1.0	0.13	0.27	0.02	0.06	0.14
1.5	0.11	0.29	0.03	0.07	0.13

Table 5.8: Transferability of SVM, LR, Model 1 to Model 2

In table 5.8, it can be seen that the transferability of highly regularized simple models, SVM and LR, is very high as compared to when the models are less regularized. Moreover, the shallow neural network Model 1 and SVM are performing almost the same with SVM having slightly less transferability. In addition, LR is performing the best with the highest transferability.

The table below shows the robustness approximation results of the surrogates. The results shaded yellow represent the results that are closest to the target robustness compared to others.

	Surrogates			**Target**	**Robustness Approximation via Surrogates**				**Target Robustness**
ε	**SVM**	**LR**	**Model 1**	**Model 2**	**AND**	**OR**	**MV**	**WMV**	**Target**
1.0	C=0.1	C=0.1	NN	NN	0.90	0.75	0.85	0.85	0.86
1.0	C=1.0	C=0.1	NN	NN	0.91	0.42	0.82	0.90	0.86
1.0	C=0.1	C=1.0	NN	NN	0.86	0.72	0.79	0.79	0.86
1.0	C=1.0	C=1.0	NN	NN	0.86	0.43	0.75	0.85	0.86

Table 5.9(a): Robustness approximations results of SVM, LR, and Model 1 against Model 2

	Surrogates			**Target**	**Robustness Approximation via Surrogates**				**Target Robustness**
ε	**SVM**	**LR**	**Model 1**	**Model 2**	**AND**	**OR**	**MV**	**WMV**	**Target**
1.5	C=0.1	C=0.1	NN	NN	0.76	0.40	0.55	0.75	0.62
1.5	C=1.0	C=0.1	NN	NN	0.76	0.10	0.49	0.75	0.62
1.5	C=0.1	C=1.0	NN	NN	0.57	0.37	0.44	0.44	0.62
1.5	C=1.0	C=1.0	NN	NN	0.56	0.10	0.40	0.55	0.62

Table 5.9(b): Robustness approximations results of SVM, LR, and Model 1 against Model 2

As can be seen in table 5.9(a), when the models are highly regularized (i.e., C = 0.1), the simple majority voting and weighted majority voting gates give the best results when compared to the robustness measure directly computed using the target model. However, it is worth noticing that LR has a big impact on the result shift between gates. When its highly regularized, the gate which best approximates the target model robustness is weighted majority voting, while, when it's not as highly regularized as before, the result which best approximates target model robustness is shifted

to the AND gate. It might be for the reason that transferability does have an effect that can help us spot the gate which best approximates the target model robustness. In this case, it seems that when the transferability of surrogate models is high, the weighted majority gate will give the best result. Additionally, the table also shows the effect of regularization on the gates. For instance, when C for SVM is changed from 0.1 to 1.0, it highly affects the OR gate and has a small effect on the majority voting gate which is evident from the table, while, in the case of LR, the effected gates are AND and weighted majority voting with almost the same effect on majority voting gate as SVM. Furthermore, table 5.9(b) shows the same patterns when the models are attacked with $\varepsilon = 1.5$.

The tables below show the results when two models are used against the target model.

	Surrogates			**Target**	**Robustness Approximation via Surrogates**				**Target Robustness**
$\boldsymbol{\varepsilon}$	**SVM**	**LR**		**Model 2**	**AND**	**OR**	**MV**	**WMV**	**Target**
1.0	C=0.1	C=0.1		NN	0.90	0.77	0.90	0.89	0.86
1.0	C=1.0	C=0.1		NN	0.89	0.42	0.89	0.89	0.86
1.0	C=0.1	C=1.0		NN	0.79	0.74	0.79	0.77	0.86
1.0	C=1.0	C=1.0		NN	0.76	0.42	0.76	0.76	0.86

Table 5.9(c): Robustness approximations results of SVM, LR, and Model 1 against Model 2

	Surrogates			**Target**	**Robustness Approximation via Surrogates**				**Target Robustness**
$\boldsymbol{\varepsilon}$	**SVM**	**LR**		**Model 2**	**AND**	**OR**	**MV**	**WMV**	**Target**
1.5	C=0.1	C=0.1		NN	0.75	0.42	0.75	0.75	0.62
1.5	C=1.0	C=0.1		NN	0.75	0.10	0.75	0.75	0.62
1.5	C=0.1	C=1.0		NN	0.45	0.38	0.45	0.42	0.62
1.5	C=1.0	C=1.0		NN	0.42	0.10	0.42	0.42	0.62

Table 5.9(d): Robustness approximations results of SVM, LR, and Model 1 against Model 2

In the case of using two models, SVM and LR, as surrogates, the patterns are the same as we saw when using three models but here the difference is that when LR is highly regularized, the majority voting and weighted majority voting is over-approximating the robustness measure while when it is less regularized, the AND gate is under-approximating the robustness of target model. It might

be for the reason that using the same models as a surrogate could help better approximate the target model robustness, and since we are not using any neural network as a surrogate, we are not optimally approximating the target model robustness.

Let's see what the results are when using LR and Model 1 against Model 2. The tables below show the results.

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε		LR	Model 1	Model 2	AND	OR	MV	WMV	Target
1.0		C=0.1	NN	NN	0.90	0.83	0.90	0.89	0.86
1.0		C=1.0	NN	NN	0.86	0.75	0.86	0.84	0.86

Table 5.9(e): Robustness approximations results of SVM, LR, and Model 1 against Model 2

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε		LR	Model 1	Model 2	AND	OR	MV	WMV	Target
1.5		C=0.1	NN	NN	0.76	0.54	0.76	0.76	0.62
1.5		C=1.0	NN	NN	0.46	0.38	0.46	0.42	0.62

Table 5.9(f): Robustness approximations results of SVM, LR, and Model 1 against Model 2

When attacking with stronger models such as LR which has a higher transferability and a neural network, as can be seen in table 5.9(e) and 5.9(f), the results show the kind of the same patterns as we saw with using other models above, however, this time OR gate also show a comparable result to other gates and that maybe because the two surrogates are stronger models than SVM and will probably over-approximate robustness with other gates. In the case when attacked with a higher perturbation level, OR gate is able to approximate the target robustness to some extent, but still for other gates, robustness is over-approximated with highly regularized LR while under-approximated with less regularized LR.

The tables below show when used an SVM and a shallow neural network Model 1 as surrogates against a shallow neural network Model 2.

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM		Model 1	Model 2	AND	OR	MV	WMV	Target
1.0	C=0.1		NN	NN	0.86	0.75	0.86	0.84	0.86
1.0	C=1.0		NN	NN	0.84	0.42	0.84	0.84	0.86

Table 5.9(g): Robustness approximations results of SVM, LR, and Model 1 against Model 2

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM		Model 1	Model 2	AND	OR	MV	WMV	Target
1.5	C=0.1		NN	NN	0.56	0.40	0.56	0.54	0.62
1.5	C=1.0		NN	NN	0.50	0.10	0.50	0.50	0.62

Table 5.9(h): Robustness approximations results of SVM, LR, and Model 1 against Model 2

Looking at tables 5.9(g) and 5.9(h), the surrogate models SVM and Model 1 have been able to approximate target robustness very well with $\varepsilon = 1.0$, nonetheless, in the case of $\varepsilon = 1.5$, as the adversarial examples crafted for SVM doesn't transfer as well as LR, doesn't matter if SVM is highly regularized or not, the robustness is under-approximated.

5.3.1.2 Experiments 2

In this set of experiments, Model A, Model C, and CNN were chosen as surrogates against LeNet as the target model. The models were attacked with perturbation levels $\varepsilon = [1.0, 1.5. 2.0]$. The table below shows the measure of transferability of surrogates to the target.

	Transferability		
ε	Model A	Model C	CNN
1.0	0.53	0.28	0.40
1.5	0.58	0.37	0.38
2.0	0.60	0.29	0.36

Table 5.10: Transferability of Model A, Model C, and CNN to LeNet

Since now we are using all convolution neural networks against a target convolution network, the transferability of all three surrogates is better than before. Among the three surrogates, Model A is the one with the highest transferability followed by CNN, and lastly Model C.

The tables below show the results when three of them, as well as in groups of two, are used together to approximate the target model robustness.

					Robustness Approximation via Surrogates				Target Robustness
ε	**Surrogates**			**Target**	**AND**	**OR**	**MV**	**WMV**	**Target**
1.0	Model A	Model C	CNN	LeNet	0.98	0.96	0.97	0.97	0.96
1.5	Model A	Model C	CNN	LeNet	0.95	0.90	0.93	0.93	0.87
2.0	Model A	Model C	CNN	LeNet	0.88	0.68	0.80	0.80	0.69

Table 5.11(a): Robustness approximations results of Model A, Model C, and CNN against LeNet

As table 5.11(a) shows, the result, which best approximates the target model robustness, shifted to OR gate because we are using CNNs against a CNN and all of these three surrogates are stronger models in a sense that the adversarial examples crafted for them are transferring well to the target model.

					Robustness Approximation via Surrogates				Target Robustness
ε	**Surrogates**			**Target**	**AND**	**OR**	**MV**	**WMV**	**Target**
1.0	Model A	Model C		LeNet	0.98	0.96	0.98	0.97	0.96
1.5	Model A	Model C		LeNet	0.95	0.90	0.95	0.93	0.87
2.0	Model A	Model C		LeNet	0.85	0.75	0.85	0.82	0.69

Table 5.11(b): Robustness approximations results of Model A, Model C, and CNN against LeNet

					Robustness Approximation via Surrogates				Target Robustness
ε	Surrogates			Target	AND	OR	MV	WMV	Target
1.0	Model A		CNN	LeNet	0.98	0.96	0.98	0.97	0.96
1.5	Model A		CNN	LeNet	0.95	0.92	0.95	0.93	0.87
2.0	Model A		CNN	LeNet	0.87	0.72	0.87	0.81	0.69

Table 5.11(c): Robustness approximations results of Model A, Model C, and CNN against LeNet

					Robustness Approximation via Surrogates				Target Robustness
ε	Surrogates			Target	AND	OR	MV	WMV	Target
1.0		Model C	CNN	LeNet	0.98	0.96	0.98	0.97	0.96
1.5		Model C	CNN	LeNet	0.95	0.91	0.95	0.93	0.87
2.0		Model C	CNN	LeNet	0.85	0.70	0.85	0.77	0.69

Table 5.11(d): Robustness approximations results of Model A, Model C, and CNN against LeNet

Tables 5.11(b), 5.11(c) and 5.11(d) show two surrogates against the target model. The result in these cases is the same as with three surrogates, however, if seen the result where Model C is used, the robustness approximation is close to the target model as compared to when the other two models as well as the Model A is causing the result value to go up which might be the reason that Model A is the strongest model among the three having the highest transferability. Overall, the three models together can better approximate the target robustness with the OR gate.

Moreover, if we employ the three models singularly against the target model, as can be seen in the tables below, they provide a single robustness value which might or might not approximate the target model robustness well, and using them together we have the advantage of leveraging the logical gates for the combination of the predictions for surrogates which can help approximate well the target model robustness with one or the other logical gate.

ε	Surrogates			Target	Robustness Approximation via Surrogates	Target Robustness
1.0	Model A			LeNet	0.98	0.96
1.5	Model A			LeNet	0.93	0.87
2.0	Model A			LeNet	0.82	0.69

Table 5.11(d): Robustness approximations results of Model A, Model C, and CNN against LeNet

ε	Surrogates			Target	Robustness Approximation via Surrogates	Target Robustness
1.0		Model C		LeNet	0.97	0.96
1.5		Model C		LeNet	0.93	0.87
2.0		Model C		LeNet	0.78	0.69

Table 5.11(d): Robustness approximations results of Model A, Model C, and CNN against LeNet

ε	Surrogates			Target	Robustness Approximation via Surrogates	Target Robustness
1.0			CNN	LeNet	0.97	0.96
1.5			CNN	LeNet	0.93	0.87
2.0			CNN	LeNet	0.78	0.69

Table 5.11(d): Robustness approximations results of Model A, Model C, and CNN against LeNet

5.3.1.3 Experiments 3

This set of experiments employs Model A, LeNet, and CNN as surrogates against Model C as the target model. The adversarial perturbation levels considered, for crafting adversarial samples, here are, $\varepsilon = [1.0, 1.5. 2.0]$. The table below shows the measure of transferability, to the target model, for each surrogate concerning the three perturbation levels.

	Transferability		
$\boldsymbol{\varepsilon}$	**Model A**	**LeNet**	**CNN**
1.0	0.48	0.20	0.47
1.5	0.50	0.21	1.0
2.0	0.57	0.35	0.29

Table 5.12: Transferability of SVM, LR, Model 1 to Model 2

It can be seen in table 5.12 that Model A transferability is constant, around 50%, with all the perturbation levels, while CNN with perturbation level $\varepsilon = 1.5$, the adversarial examples crafted for it are 100% transferring to Model C. LeNet has the lowest transferability among the three.

Below are the tables of robustness approximation results.

					Robustness Approximation via Surrogates				**Target Robustness**
$\boldsymbol{\varepsilon}$	**Surrogates**			**Target**	**AND**	**OR**	**MV**	**WMV**	**Target**
1.0	Model A	LeNet	CNN	Model C	0.98	0.95	0.97	0.97	0.95
1.5	Model A	LeNet	CNN	Model C	0.95	0.84	0.94	0.93	0.76
2.0	Model A	LeNet	CNN	Model C	0.88	0.60	0.80	0.80	0.47

Table 5.13(a): Robustness approximations results of Model A, LeNet, and CNN against Model C

					Robustness Approximation via Surrogates				Target Robustness
ε	Surrogates			Target	AND	OR	MV	WMV	Target
1.0	Model A	LeNet		Model C	0.95	0.95	0.95	0.97	0.95
1.5	Model A	LeNet		Model C	0.94	0.85	0.94	0.93	0.76
2.0	Model A	LeNet		Model C	0.84	0.67	0.84	0.82	0.47

Table 5.13(b): Robustness approximations results of Model A, LeNet, and CNN against Model C

					Robustness Approximation via Surrogates				Target Robustness
ε	Surrogates			Target	AND	OR	MV	WMV	Target
1.0	Model A		CNN	Model C	0.98	0.96	0.98	0.97	0.95
1.5	Model A		CNN	Model C	0.95	0.91	0.95	0.93	0.76
2.0	Model A		CNN	Model C	0.87	0.72	0.87	0.82	0.47

Table 5.13(c): Robustness approximations results of Model A, LeNet, and CNN against Model C

Overall, the results, using this combination of models, show the same patterns as the results in *section 5.3.1.2*. The OR gate is dominating because the surrogate models are too strong as compared to the target model as well as surrogates over-approximating the robustness of the target model with increasing levels of perturbation.

We adversarial trained the target models to increase their robustness against adversarial attacks. We hope the surrogates would be able to approximate well the robustness of target models when the target models are robust to adversarial attacks. In the next section, we present the experimental evaluations with adversarial-trained target models.

5.3.2 Adversarial Trained Targets

5.3.2.1 Experiments 1

We will use simple models, SVM and LR, and a relatively complex convolutional neural network model, Model 3, against adversarial trained Model 4. The adversarial examples were crafted using the same levels of perturbation as before, $\varepsilon = [1.0, 1.5. 2.0]$. The table below shows the measure of transferability.

	Transferability				
	C=0.1		**C=1.0**		
$\boldsymbol{\varepsilon}$	**SVM**	**LR**	**SVM**	**LR**	**Model 1**
1.0	0.005	0.012	0.0018	0.005	0.065
1.5	0.0017	0.009	0.0008	0.0017	0.050
2.0	0.0028	0.008	0.0007	0.001	0.067

Table 5.14: Transferability of SVM, LR, Model 1 to Model 2

The transferability measures are very low for the reason that adversarial training made the target model much more robust to adversarial attacks. Adversarial attacks crafted with three different levels of perturbations doesn't for surrogate models don't affect the target classifier much. Since the models' transferability measures are low, they can be termed weak models. The simple models, SVM and LR have very low transferability which is around 1% for LR in the case of crafting adversarial examples with $\varepsilon = 1.0$ and lower than 1% for both with all the levels of applied perturbations. Moreover, Model 3 has better transferability but still, it's not enough.

As we saw in section *5.3.1.1*, less regularized models cause the result of best robustness approximation to be shifted to AND gate because the transferability measures were low, however, in this case, all the surrogate models have very low transferability so we would expect the AND gate to approximate the robustness of target model well. The tables below show the robustness approximation results.

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM	LR	Model 3	Model 4	AND	OR	MV	WMV	Target
1.0	C=0.1	C=0.1	Model 3	Model 4	0.96	0.76	0.89	0.94	0.99
1.0	C=1.0	C=0.1	Model 3	Model 4	0.96	0.42	0.88	0.94	0.99
1.0	C=0.1	C=1.0	Model 3	Model 4	0.95	0.73	0.79	0.94	0.99
1.0	C=1.0	C=1.0	Model 3	Model 4	0.95	0.42	0.75	0.94	0.99

Table 5.15(a): Robustness approximations results of SVM, LR, and Model 3 against ADV Model 4

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM	LR	Model 3	Model 4	AND	OR	MV	WMV	Target
1.5	C=0.1	C=0.1	Model 3	Model 4	0.88	0.41	0.71	0.83	0.88
1.5	C=1.0	C=0.1	Model 3	Model 4	0.88	0.10	0.70	0.83	0.88
1.5	C=0.1	C=1.0	Model 3	Model 4	0.84	0.37	0.45	0.83	0.88
1.5	C=1.0	C=1.0	Model 3	Model 4	0.84	0.10	0.41	0.83	0.88

Table 5.15(b): Robustness approximations results of SVM, LR, and Model 3 against ADV Model 4

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM	LR	Model 3	Model 4	AND	OR	MV	WMV	Target
2.0	C=0.1	C=0.1	Model 3	Model 4	0.70	0.14	0.41	0.65	0.67
2.0	C=1.0	C=0.1	Model 3	Model 4	0.70	0.01	0.41	0.65	0.67
2.0	C=0.1	C=1.0	Model 3	Model 4	0.65	0.12	0.17	0.65	0.67
2.0	C=1.0	C=1.0	Model 3	Model 4	0.65	0.01	0.15	0.65	0.67

Table 5.15(c): Robustness approximations results of SVM, LR, and Model 3 against ADV Model 4

As we were anticipating, the AND gate is dominating here is all the cases, except in the case when the surrogates are attacked with $\varepsilon = 2.0$ where we see the same patterns as in section *5.3.1.1* where the regularization of LR was causing the shift of best result from weighted majority voting gate to

AND gate. Overall, the models have low transferability, and, in such cases, AND gate would approximate the target models' robustness better.

It would be interesting to see how the surrogates in groups of two perform against the target model. Below are the table presenting two surrogates against the target model.

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM	LR		Model 4	AND	OR	MV	WMV	Target
1.0	C=0.1	C=0.1		Model 4	0.90	0.77	0.90	0.90	0.99
1.0	C=1.0	C=0.1		Model 4	0.90	0.42	0.90	0.90	0.99
1.0	C=0.1	C=1.0		Model 4	0.79	0.74	0.79	0.77	0.99
1.0	C=1.0	C=1.0		Model 4	0.76	0.42	0.76	0.76	0.99

Table 5.15(d): Robustness approximations results of SVM, LR, and Model 3 against ADV Model 4

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM	LR		Model 4	AND	OR	MV	WMV	Target
1.5	C=0.1	C=0.1		Model 4	0.76	0.42	0.76	0.76	0.88
1.5	C=1.0	C=0.1		Model 4	0.75	0.10	0.75	0.75	0.88
1.5	C=0.1	C=1.0		Model 4	0.46	0.38	0.46	0.42	0.88
1.5	C=1.0	C=1.0		Model 4	0.42	0.11	0.42	0.42	0.88

Table 5.15(e): Robustness approximations results of SVM, LR, and Model 3 against ADV Model 4

It seems, from table 5.15(d) and 5.15(e), that SVM and LR alone cannot approximate the robustness well when the target model is adversarially trained. The results are not even interesting at perturbation level 1.0 as only the first two results show a comparable robustness approximation with resect to the target model. As the perturbation level advance to 1.5, all the results are under-approximated.

The tables below, 5.15(f), 5.15(g), and 5.15(h), show SVM along with Model 3 as surrogates against the target Model 4, while tables 5.15(f), 5.15(g), and 5.15(h), present LR and Model 3 as surrogates against Model 4.

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM		Model 3	Model 4	AND	OR	MV	WMV	Target
1.0	C=0.1		Model 3	Model 4	0.95	0.77	0.95	0.94	0.99
1.0	C=1.0		Model 3	Model 4	0.94	0.42	0.94	0.94	0.99

Table 5.15(f): Robustness approximations results of SVM, LR, and Model 3 against ADV Model 4

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM		Model 3	Model 4	AND	OR	MV	WMV	Target
1.5	C=0.1		Model 3	Model 4	0.84	0.41	0.84	0.83	0.88
1.5	C=1.0		Model 3	Model 4	0.83	0.11	0.83	0.83	0.88

Table 5.15(g): Robustness approximations results of SVM, LR, and Model 3 against ADV Model 4

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM		Model 3	Model 4	AND	OR	MV	WMV	Target
2.0	C=0.1		Model 3	Model 4	0.70	0.41	0.70	0.65	0.67
2.0	C=1.0		Model 3	Model 4	0.65	0.15	0.65	0.65	0.67

Table 5.15(h): Robustness approximations results of SVM, LR, and Model 3 against ADV Model 4

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε		LR	Model 3	Model 4	AND	OR	MV	WMV	Target
1.0		C=0.1	Model 3	Model 4	0.96	0.88	0.96	0.94	0.99
1.0		C=1.0	Model 3	Model 4	0.95	0.75	0.95	0.94	0.99

Table 5.15(i): Robustness approximations results of SVM, LR, and Model 3 against ADV Model 4

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε		LR	Model 3	Model 4	AND	OR	MV	WMV	Target
1.5		C=0.1	Model 3	Model 4	0.88	0.70	0.88	0.83	0.88
1.5		C=1.0	Model 3	Model 4	0.84	0.84	0.84	0.83	0.88

Table 5.15(j): Robustness approximations results of SVM, LR, and Model 3 against ADV Model 4

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε		LR	Model 3	Model 4	AND	OR	MV	WMV	Target
2.0		C=0.1	Model 3	Model 4	0.70	0.41	0.70	0.65	0.67
2.0		C=1.0	Model 3	Model 4	0.65	0.15	0.65	0.65	0.67

Table 5.15(k): Robustness approximations results of SVM, LR, and Model 3 against ADV Model 4

The tables above show the same patterns and almost the same result when using the three models. It is clear that the simple models alone cannot approximate well, however, if used both or one of them together with a neural network, the robustness approximation is optimally possible using AND gate considering the models do not transfer well.

5.3.2.2 Experiments 2

In this group of experiments, we considered SVM, LR, and LeNet against adversarial-trained Model C. The table below shows the measures of transferability.

	Transferability				
	C=0.1		C=1.0		
ε	SVM	LR	SVM	LR	LeNet
1.0	0.005	0.016	0.0012	0.003	0.078
1.5	0.0035	0.012	0.0011	0.0023	0.094
2.0	0.043	0.011	0.0014	0.0036	0.10

Table 5.16: Transferability of SVM, LR, and LeNet to Model C

As we can see in table 5.16, the attacks crafted for SVM and LR don't transfer well to the target Model C and the transferability of SVM is lower than the transferability of LR except when SVM is highly regularized, and the attack perturbation is 2.0. The adversarial examples crafted for LeNet are transferring at most 10% however, the transferability of LeNet is on average between 8 to 10% for all of the three perturbations.

The tables below show the robustness approximation results.

	Surrogates			**Target**	**Robustness Approximation via Surrogates**				**Target Robustness**
ε	**SVM**	**LR**	**LeNet**	**Model C**	**AND**	**OR**	**MV**	**WMV**	**Target**
1.0	C=0.1	C=0.1	LeNet	Model C	0.97	0.77	0.90	0.96	0.98
1.0	C=1.0	C=0.1	LeNet	Model C	0.97	0.42	0.89	0.96	0.98
1.0	C=0.1	C=1.0	LeNet	Model C	0.96	0.74	0.79	0.96	0.98
1.0	C=1.0	C=1.0	LeNet	Model C	0.96	0.42	0.76	0.96	0.98

Table 5.17(a): Robustness approximations results of SVM, LR, and LeNet against ADV Model C

	Surrogates			**Target**	**Robustness Approximation via Surrogates**				**Target Robustness**
ε	**SVM**	**LR**	**LeNet**	**Model C**	**AND**	**OR**	**MV**	**WMV**	**Target**
1.5	C=0.1	C=0.1	LeNet	Model C	0.90	0.41	0.73	0.87	0.94
1.5	C=1.0	C=0.1	LeNet	Model C	0.90	0.11	0.73	0.87	0.94
1.5	C=0.1	C=1.0	LeNet	Model C	0.87	0.38	0.46	0.87	0.94
1.5	C=1.0	C=1.0	LeNet	Model C	0.87	0.1	0.41	0.87	0.94

Table 5.17(b): Robustness approximations results of SVM, LR, and LeNet against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM	LR	LeNet	Model C	AND	OR	MV	WMV	Target
2.0	C=0.1	C=0.1	LeNet	Model C	0.74	0.14	0.43	0.69	0.79
2.0	C=1.0	C=0.1	LeNet	Model C	0.74	0.01	0.43	0.69	0.79
2.0	C=0.1	C=1.0	LeNet	Model C	0.70	0.12	0.17	0.70	0.79
2.0	C=1.0	C=1.0	LeNet	Model C	0.70	0.014	0.15	0.70	0.79

Table 5.17(c): Robustness approximations results of SVM, LR, and LeNet against ADV Model C

Tables 5.17(a), 5.17(b), and 5.17(c) show a very clear pattern. When the LR is highly regularized, the AND gate dominates giving the best approximation result, however, when its less regularized, AND gate and weighted majority voting (WMV) gate give the same result. One thing to be noticed is that in the case of $\varepsilon = 1.0$ and $\varepsilon = 1.5$, switching between highly regularized and less regularized models doesn't affect the weighted majority voting gate. However, as we saw in the previous experiments, AND or weighted majority voting giving the best approximation can be connected to the low transferability of the surrogates to the target.

Let's see the case of using two surrogates against the target model.

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM		LeNet	Model C	AND	OR	MV	WMV	Target
1.0	C=0.1		LeNet	Model C	0.96	0.77	0.96	0.96	0.98
1.0	C=1.0		LeNet	Model C	0.96	0.42	0.96	0.96	0.98

Table 5.17(d): Robustness approximations results of SVM, LR, and LeNet against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM		LeNet	Model C	AND	OR	MV	WMV	Target
1.5	C=0.1		LeNet	Model C	0.87	0.41	0.87	0.87	0.94
1.5	C=1.0		LeNet	Model C	0.87	0.10	0.87	0.87	0.94

Table 5.17(e): Robustness approximations results of SVM, LR, and LeNet against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	SVM		LeNet	Model C	AND	OR	MV	WMV	Target
2.0	C=0.1		LeNet	Model C	0.70	0.14	0.70	0.70	0.79
2.0	C=1.0		LeNet	Model C	0.70	0.01	0.70	0.70	0.79

Table 5.17(f): Robustness approximations results of SVM, LR, and LeNet against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε		LR	LeNet	Model C	AND	OR	MV	WMV	Target
1.0		C=0.1	LeNet	Model C	0.97	0.89	0.97	0.96	0.98
1.0		C=1.0	LeNet	Model C	0.96	0.76	0.96	0.96	0.98

Table 5.17(g): Robustness approximations results of SVM, LR, and LeNet against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε		LR	LeNet	Model C	AND	OR	MV	WMV	Target
1.5		C=0.1	LeNet	Model C	0.90	0.73	0.90	0.87	0.94
1.5		C=1.0	LeNet	Model C	0.87	0.41	0.87	0.87	0.94

Table 5.17(h): Robustness approximations results of SVM, LR, and LeNet against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε		LR	LeNet	Model C	AND	OR	MV	WMV	Target
2.0		C=0.1	LeNet	Model C	0.74	0.43	0.74	0.70	0.79
2.0		C=1.0	LeNet	Model C	0.70	0.15	0.70	0.70	0.79

Table 5.17(i): Robustness approximations results of SVM, LR, and LeNet against ADV Model C

Overall, tables, 5.17(d) to 5.17(i), show the same patterns as before with AND and weighted majority voting gates dominating. LeNet along with LR instead of SVM show slightly better approximation result.

5.3.2.3 Experiments 3

Since SVM is not contributing much when used with other models, we decided to omit it in this group of experiments. We are using LR, LeNet, and CNN against Model C. The table below shows the measure of transferability.

	Transferability			
	LR			
ε	C=0.1	C=1.0	LeNet	CNN
1.0	0.016	0.0003	0.078	0.26
1.5	0.012	0.0023	0.094	0.92
2.0	0.011	0.0036	0.10	0.82

Table 5.18: Transferability of LR, LeNet, and CNN to Model C

Here, LR and LeNet are not transferring very well but CNN which has a very high transferability reaching as high as 92% with $\varepsilon = 1.5$ and 82% with $\varepsilon = 2.0$ which might be for the reason that CNN and Model C are very much the same in architecture.

The tables below show the robustness approximation result.

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	LR	LeNet	CNN	Model C	AND	OR	MV	WMV	Target
1.0	C=0.1	LeNet	CNN	Model C	0.98	0.89	0.96	0.97	0.98
1.0	C=1.0	LeNet	CNN	Model C	0.98	0.76	0.96	0.97	0.98

Table 5.19(a): Robustness approximations results of LR, LeNet, and CNN against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	LR	LeNet	CNN	Model C	AND	OR	MV	WMV	Target
1.5	C=0.1	LeNet	CNN	Model C	0.95	0.73	0.88	0.93	0.94
1.5	C=1.0	LeNet	CNN	Model C	0.94	0.41	0.85	0.93	0.94

Table 5.19(b): Robustness approximations results of LR, LeNet, and CNN against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	LR	LeNet	CNN	Model C	AND	OR	MV	WMV	Target
2.0	C=0.1	LeNet	CNN	Model C	0.85	0.42	0.66	0.78	0.79
2.0	C=1.0	LeNet	CNN	Model C	0.85	0.15	0.62	0.78	0.79

Table 5.19(c): Robustness approximations results of LR, LeNet, and CNN against ADV Model C

The AND and weighted majority voting gates are giving the best results, looking at tables 5.19(a), 5.19(b), and 5.19(c), and that is because we just have only one model that is transferring well. However, if we look at the OR gate, the result is not as low as when all the surrogates have low transferability as we saw previously. That is because we have one model, Model C, which is transferring very well, and the high transferability is not letting the OR gate get very low. But the other two do not have much impressive transferability and that's the reason we are not getting better results with OR gate. If all the models transfer well, as we saw in section *5.3.1.3*, the OR gate would be the one best approximating the target robustness.

The tables below show the results with two surrogates against the target.

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	LR		CNN	Model C	AND	OR	MV	WMV	Target
1.0	C=0.1		CNN	Model C	0.97	0.90	0.97	0.97	0.98
1.0	C=1.0		CNN	Model C	0.97	0.76	0.97	0.97	0.98

Table 5.19(d): Robustness approximations results of LR, LeNet, and CNN against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	LR		CNN	Model C	AND	OR	MV	WMV	Target
1.5	C=0.1		CNN	Model C	0.936	0.75	0.936	0.932	0.94
1.5	C=1.0		CNN	Model C	0.93	0.42	0.93	0.93	0.94

Table 5.19(e): Robustness approximations results of LR, LeNet, and CNN against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	LR		CNN	Model C	AND	OR	MV	WMV	Target
2.0	C=0.1		CNN	Model C	0.78	0.47	0.78	0.78	0.79
2.0	C=1.0		CNN	Model C	0.78	0.15	0.78	0.78	0.79

Table 5.19(f): Robustness approximations results of LR, LeNet, and CNN against ADV Model C

In the case of using LR with CNN against Model C, we can see that AND and weighted majority voting gates give almost the same result and consistent results with all three levels of perturbations. This might be a sign that one weak model, with low transferability, and one stronger model, with high transferability, when used together, the gates which will best approximate the target model robustness would be AND and weighted majority voting.

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε		LeNet	CNN	Model C	AND	OR	MV	WMV	Target
1.0		LeNet	CNN	Model C	0.98	0.95	0.98	0.97	0.98

Table 5.19(g): Robustness approximations results of LR, LeNet, and CNN against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε		LeNet	CNN	Model C	AND	OR	MV	WMV	Target
1.5		LeNet	CNN	Model C	0.946	0.85	0.946	0.932	0.94

Table 5.19(h): Robustness approximations results of LR, LeNet, and CNN against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε		LeNet	CNN	Model C	AND	OR	MV	WMV	Target
2.0		LeNet	CNN	Model C	0.85	0.62	0.85	0.78	0.79

Table 5.19(i): Robustness approximations results of LR, LeNet, and CNN against ADV Model C

When using LeNet along with CNN as surrogates, the patterns are kind of the same as we saw when using LR with CNN as surrogates, however, LeNet is a convolution neural network and has better transferability than LR and as we move to the results with $\varepsilon = 2.0$, the AND gate, in this case, is over-approximating the target robustness which might be for the fact that LeNet is not as weak as LR against Model C and the adversarial examples crafted for it are transferring more than LR. So, we cannot call LeNet a very weak model against Model C, having 10% transferability, and combined with a stronger model such as CNN, the weighted majority voting gate would approximate the target model robustness well.

5.3.2.4 Experiments 4

Since LR proved to be not much effective in terms of transferability, we omit it in this set of experiments and use Model A instead. The table below shows the measures of transferability for the three surrogates.

	Transferability		
ε	**Model A**	**LeNet**	**CNN**
1.0	0.196	0.078	0.26
1.5	0.2	0.094	0.92
2.0	0.24	0.10	0.82

Table 5.20: Transferability of Model A, LeNet, and CNN to Model C

Looking at the table above, Model A seems better having transferability slightly more than twice the transferability of LeNet. Since now we have models that have better transferability, we can expect the OR gate to give much better approximation results as compared to before.

The tables below show the approximation results.

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	**Model A**	**LeNet**	**CNN**	**Model C**	**AND**	**OR**	**MV**	**WMV**	**Target**
1.0	Model A	LeNet	CNN	Model C	0.98	0.95	0.97	0.97	0.98
1.5	Model A	LeNet	CNN	Model C	0.95	0.85	0.93	0.93	0.94
2.0	Model A	LeNet	CNN	Model C	0.88	0.60	0.80	0.78	0.79

Table 5.21(a): Robustness approximations results of Model A, LeNet, and CNN against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	Model A	LeNet		Model C	AND	OR	MV	WMV	Target
1.0	Model A	LeNet		Model C	0.98	0.95	0.98	0.97	0.98
1.5	Model A	LeNet		Model C	0.94	0.85	0.94	0.93	0.94
2.0	Model A	LeNet		Model C	0.84	0.67	0.84	0.81	0.79

Table 5.21(b): Robustness approximations results of Model A, LeNet, and CNN against ADV Model C

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	Model A		CNN	Model C	AND	OR	MV	WMV	Target
1.0	Model A		CNN	Model C	0.98	0.97	0.98	0.97	0.98
1.5	Model A		CNN	Model C	0.95	0.92	0.95	0.93	0.94
2.0	Model A		CNN	Model C	0.87	0.72	0.87	0.78	0.79

Table 5.21(c): Robustness approximations results of Model A, LeNet, and CNN against ADV Model C

As we were expecting, the results with OR gate seem much better than before and that's for the reason that we have models that have better transferability except LeNet which is not as high. This becomes clear from table 5.21(c) where the results with OR gate seem much improved since we are using two models, one with high transferability and the other one that can be called average and not very low. Since the AND and weighted majority voting gates both are best approximating two results each (AND gives the best result with $\varepsilon = 1.0$, while weighted majority voting gives the best result with $\varepsilon = 2.0$, and in the case of $\varepsilon = 1.5$ both over-approximate and under-approximate by same value so the result can be considered common) Model A needs to have transferability higher than the current transferability to see a shift of result that optimally approximate the target model to the OR gate.

The tables below show when surrogate models one by one are tested against the target model.

ε	Surrogates			Target	Robustness Approximation via Surrogates	Target Robustness
1.0	Model A			Model C	0.976	0.98
1.5	Model A			Model C	0.93	0.94
2.0	Model A			Model C	0.817	0.79

Table 5.21(d): Robustness approximations results of Model A, LeNet, and CNN against ADV Model C

ε	Surrogates			Target	Robustness Approximation via Surrogates	Target Robustness
1.0		LeNet		Model C	0.96	0.98
1.5		LeNet		Model C	0.87	0.94
2.0		LeNet		Model C	0.695	0.79

Table 5.21(e): Robustness approximations results of Model A, LeNet, and CNN against ADV Model C

ε	Surrogates			Target	Robustness Approximation via Surrogates	Target Robustness
1.0			CNN	Model C	0.97	0.98
1.5			CNN	Model C	0.93	0.94
2.0			CNN	Model C	0.77	0.79

Table 5.21(f): Robustness approximations results of Model A, LeNet, and CNN against ADV Model C

From tables, 5.21(d), 5.21(e), 5.21(f), it is clear that CNN alone can approximate the target model robustness since is very much the same in architecture as the target model, however, in practice, it would not be easy to find a surrogate model who is fully transferring or transferring to a very high extant to the target model. Also, still, the results, when using a model along with CNN as a surrogate whose adversarial examples are averagely transferring (i.e., Model A with transferability equal 24% on $\varepsilon = 2.0$), show slightly better approximation when compared to the results of CNN alone against Model C. Moreover, the results of using just CNN against Model C very much coincide with the results of using two models, Model A and CNN as surrogates against Model C. But in practice, it would not be easy

5.3.2.5 Experiments 5

In this group of experiments, the same models are used as used in section *5.3.2.4* but instead of Model C as a target, adversarial-trained LeNet is used as a target, and Model C is moved in place of LeNet as a surrogate. The table below shows transferability measures.

	Transferability		
ε	**Model A**	**Model C**	**CNN**
1.0	0.07	0.07	0.8
1.5	0.16	0.11	0.14
2.0	0.23	0.14	0.15

Table 5.22: Transferability of LR, LeNet, and CNN to Model C

It can be seen in the table above that adversarial samples crafted for Model A and Model C at $\varepsilon = 1.0$ does not transfer well while adversarial examples crafted for CNN at the same perturbation level are transferring very well, about 80%. However, as the perturbation level increases, the transferability of CNN drops while the other two improve. The adversarial examples crafted for all of the three surrogates have almost the same transferability at $\varepsilon = 1.5$ while at $\varepsilon = 2.0$, Model A transferability is highest.

The tables below show the robustness approximation results when three of them are used together against LeNet.

	Surrogates			**Target**	**Robustness Approximation via Surrogates**				**Target Robustness**
ε	**Model A**	**Model C**	**CNN**	**LeNet**	**AND**	**OR**	**MV**	**WMV**	**Target**
1.0	Model A	Model C	CNN	LeNet	0.98	0.96	0.976	0.976	0.98
1.5	Model A	Model C	CNN	LeNet	0.95	0.90	0.93	0.93	0.95
2.0	Model A	Model C	CNN	LeNet	0.88	0.68	0.80	0.80	0.77

Table 5.23(a): Robustness approximations results of Model A, Model C, and CNN against ADV LeNet

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	Model A	Model C		LeNet	AND	OR	MV	WMV	Target
1.0	Model A	Model C		LeNet	0.98	0.96	0.98	0.97	0.98
1.5	Model A	Model C		LeNet	0.95	0.90	0.95	0.93	0.95
2.0	Model A	Model C		LeNet	0.85	0.75	0.85	0.81	0.77

Table 5.23(b): Robustness approximations results of Model A, Model C, and CNN against ADV LeNet

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε	Model A		CNN	LeNet	AND	OR	MV	WMV	Target
1.0	Model A		CNN	LeNet	0.98	0.97	0.98	0.97	0.98
1.5	Model A		CNN	LeNet	0.95	0.92	0.95	0.93	0.95
2.0	Model A		CNN	LeNet	0.87	0.72	0.87	0.82	0.77

Table 5.23(c): Robustness approximations results of Model A, Model C, and CNN against ADV LeNet

	Surrogates			Target	Robustness Approximation via Surrogates				Target Robustness
ε		Model C	CNN	LeNet	AND	OR	MV	WMV	Target
1.0		Model C	CNN	LeNet	0.98	0.97	0.98	0.97	0.98
1.5		Model C	CNN	LeNet	0.95	0.91	0.95	0.92	0.95
2.0		Model C	CNN	LeNet	0.85	0.70	0.85	0.78	0.77

Table 5.23(d): Robustness approximations results of Model A, Model C, and CNN against ADV LeNet

In table 5.23(a), the results show patterns again as we saw before. For $\varepsilon = [1.0, 1.5]$, the AND gate is approximating well because the adversarial examples crafting for all the three models are averagely transferring, however, in the case of $\varepsilon = 2.0$, Model A transferability is higher, and the resulting shift to weighted majority voting gate, but weighted majority voting is over-approximating the target robustness by a small amount. If we take the average of transferability measures in the case of $\varepsilon = 2.0$, we get 17%. In the cases of using two models, if Model A is included, with $\varepsilon = 2.0$, the result shifts to OR gate, and taking the average of transferability for the two we get 19%. If we look at table 5.23(d), with $\varepsilon = 2.0$, the weighted majority voting gate is better approximating the target robustness, and taking the average of the transferability of the models, Model C and CNN, we get 14.5%. That might be a sign that if we can find models whose transferability is greater than 20%, we can safely choose the OR gate. We can choose WMV gate if we have surrogate models whose transferability, with $\varepsilon = 2.0$, each is between 14% and 20%. Furthermore, for the same perturbation level, $\varepsilon = 2.0$, if each of the surrogate models' transferability is lower than 14%, it would be the AND gate that would be best approximating the target robustness.

5.4 Discussion

With reference to the extensive experiments presented above, one thing which is clear is that if the surrogate models do not have sufficient transferability or the transferability is very low, AND gate would be the one to choose because in case of weak surrogate models whose adversarial examples don't transfer well, AND gate will give the best approximation result. If the models have average transferability or the transferability is not very low, the weighted majority gate (WMV) will most probably approximate the target robustness better. However, if the surrogate models have strong transferability, it would not be wrong to select the approximation result with OR gate as the optimal approximation result.

In the case of using models different from the target as surrogates, as we saw in sections *5.3.1.1* and *5.3.2.1*, the results might not be as expected with lower perturbations and with higher perturbation levels, they might very probably under-approximate the target robustness. So, it's better to use models as a surrogate that are of the same kind, i.e., if the target model performs image classification or accepts images as query, then the target model most probably will be a convolution neural network, as they are better with image classification, and we can train convolution neural networks as surrogates to in order to be able to infer optimally the target model robustness.

As for the level of attack perturbation, with $\varepsilon = 2.0$, it can be said that if two of the surrogate models have at least 10% transferability, considering three surrogates, the WMV gate will give the best approximation result. Looking at section *5.3.2.3*, if we consider three models at $\varepsilon = 2.0$, the WMV gate is giving better results since one model, LR has transferability lower than 10%. However, if we look at the table with two surrogates, LR and CNN, the AND gate provides a better

result. Furthermore, in the case of two surrogates with LeNet instead of LR, the best result move to the WMV gate again which proves that at least two models need to have transferability greater than or equal to 10% to choose the result with the WMV gate. For the shift of best result from WMV to OR gate, we have two sets of experiments, section *5.3.1.2*, and what we can get from it at most is that all the models need to have at least 28% transferability in order to choose the result of OR gate as the optimal result. However, if we look at tables 5.23(b) and 5.23(c), tables 5.23(b) show results with surrogates with Model A and Model C who has transferability 23% and 14% respectively with $\varepsilon = 2.0$ and the OR gate is under-approximating the target robustness by a small amount. While table 5.23(c) shows Model A and CNN whose transferability is 23% and 15% respectively but the approximation value drops further a little bit with OR gate despite CNN having a transferability of 1% higher than Model C, which might seem to contradict. However, table 5.23(d) shows a further drop in the robustness approximation with OR gate and the models, Model C and CNN, have transferability of 14% and 15%. So, in simple terms, we can say that if we are using three models or two models as surrogates, if the transferability of any of the two models is at least 10%, we can choose the WMV gate while if any of the two models have transferability at least 23%, we can choose OR gate.

For the perturbation magnitudes $\varepsilon = [1.0, 1.5]$, we can observe slightly different transferability levels for choosing gates. In section *5.3.1.1*, we can observe that if at least two models have 10% or higher transferability, the best approximation result move from AND to WMV gate. Observing the results in section *5.3.2.3* further proves the argument stated before. However, in section *5.3.2.3*, we also observe that if one model has at least 82% transferability, then the result shifts to the WMV gate as well. Nonetheless, we can notice in section *5.3.2.4*, in tables 5.21(a) and 5.21(c) that despite two of the models having transferability higher than 10%, the best result stays at AND gate. For instance, Model A has a transferability of 19.6% and CNN has 26% at $\varepsilon = 1.0$ and the best gate in this case is AND. But if we look at the results with $\varepsilon = 1.5$, Model A has 20% transferability and CNN has 92% transferability and, in this case, the result moves to the WMV gate. This means that we need at least two models with transferability higher than 20% to choose the result of the WMV gate as the best robustness approximation. Now, in section *5.3.1.3*, we can infer that at the minimum, two surrogates need to have at least 20% transferability to observe the best robustness approximation with the OR gate. However, this contradicts the previous statement where it was argued that two surrogates must have 20% or higher transferability to choose WMV gate. This contradiction is cleared by observing the results in section *5.3.1.2* where we can note that for choosing the OR gate, the transferability needs to be at least 28%. Consequently, we can say that at least two models must have transferability higher than 20% to observe the best result at the WMV gate while if the transferability is higher than 28%, then we can safely choose the result given by the OR gate to be the optimal approximation of the target model.

Conclusion

The use of machine learning models has risen immensely recently for their powerful near-human or superhuman performance in some cases, however, they are still struggling with being robust against adversarial attacks that can greatly reduce their performance. Robustness is important for machine learning models to help them resist adversarial attacks. Robustness is even more important in safety-critical systems where the space for error or misclassification is extremely little and a model which is not robust to these adversarial attacks, specifically evasion attacks, can cause havoc, causing harm to critical systems and/or human life.

In this work, we discussed how evasion attacks can mislead the machine learning models into misclassifying instances, why is it important to establish the security of machine learning models and why is it important to certify robustness along with discussing previously proposed approaches to robustness certification as well as why is it challenging to certify robustness in black-box or real-world scenarios. We tried to deal with the problem of robustness certification by proposing a framework that can be used by an analyst who wants to evaluate the robustness of a machine learning model in black-box settings via surrogate models for the reason of robustness certification. As most of the previous research work has tried to certify robustness in white-box settings via verification techniques, giving theoretical bounds followed by experimental guarantees in achieving these bounds, we discussed how our approach is different and challenging (as an analyst who wants to certify robustness or an attacker who want to evaluate a target model for robustness to adversarial attacks).

We presented the experimental evaluations done for the development of this thesis and the experimental assessments show the possibility of effectively approximating the robustness via surrogate models in black-box settings. It is clear from the evaluations presented that the approximation results move from AND to weighted majority voting (WMV) to OR gate with increasing transferability. We have set some initial transferability bounds under or over which the approximation result shift between logical gates, however, there is still the need for more extensive experiments done to refine the bounds.

References

[1] T. M. Mitchell and T. M. Mitchell, *Machine learning*, vol. 1, no. 9. McGraw-hill New York, 1997.

[2] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural networks*, vol. 32, pp. 323–332, 2012.

[3] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[4] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2154–2156.

[5] C. Szegedy *et al.*, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[6] K. Eykholt *et al.*, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1625–1634.

[7] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[8] Y. Shi, S. Wang, and Y. Han, "Curls & whey: Boosting black-box adversarial attacks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6519–6527.

[9] Y. Dong, T. Pang, H. Su, and J. Zhu, "Evading defenses to transferable adversarial examples by translation-invariant attacks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4312–4321.

[10] L. Wu and Z. Zhu, "Towards understanding and improving the transferability of adversarial examples in deep neural networks," in *Asian Conference on Machine Learning*, 2020, pp. 837–850.

[11] A. Demontis *et al.*, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in *28th USENIX security symposium*, 2019, pp. 321–338.

[12] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J Res Dev*, vol. 44, no. 1.2, pp. 206–226, 2000.

[13] N. Cristianini, "The road to artificial intelligence: A case of data over theory," *New Sci (1956)*, vol. 26, 2016.

[14] B. Liu and others, *Web data mining: exploring hyperlinks, contents, and usage data*, vol. 1. Springer, 2011.

[15] S. Haykin and D. J. Thomson, "Signal detection in a nonstationary environment reformulated as an adaptive pattern classification problem," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2325–2344, 1998.

[16] S. B. Kotsiantis, I. Zaharakis, P. Pintelas, and others, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, no. 1, pp. 3–24, 2007.

[17] C. Cortes and V. Vapnik, "Support-vector networks," *Mach Learn*, vol. 20, pp. 273–297, 1995.

[18] O. Maimon and L. Rokach, "Data mining and knowledge discovery handbook," 2005.

[19] P. Branco, L. Torgo, and R. P. Ribeiro, "A survey of predictive modeling on imbalanced domains," *ACM computing surveys (CSUR)*, vol. 49, no. 2, pp. 1–50, 2016.

[20] W. S. Noble, "What is a support vector machine?," *Nat Biotechnol*, vol. 24, no. 12, pp. 1565–1567, 2006.

[21] J. S. Cramer, "The Origins of Logistic Regression: Tinbergen Institute Discussion Papers," 2002.

[22] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[23] B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack," *IEEE Trans Knowl Data Eng*, vol. 26, no. 4, pp. 984–996, 2013.

[24] Z. Kolter and A. Madry, "Adversarial robustness: Theory and practice," *Tutorial at NeurIPS*, p. 3, 2018.

[25] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[26] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*, 2017, pp. 39–57.

[27] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Artificial intelligence safety and security*, Chapman and Hall/CRC, 2018, pp. 99–112.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[29] N. Carlini *et al.*, "On evaluating adversarial robustness," *arXiv preprint arXiv:1902.06705*, 2019.

[30] A. Wieland and C. M. Wallenburg, “Dealing with supply chain risks: Linking risk management practices and strategies to performance,” *International journal of physical distribution & logistics management*, 2012.

[31] S. Calzavara, L. Cazzaro, C. Lucchese, F. Marcuzzi, and S. Orlando, “Beyond robustness: Resilience verification of tree-based classifiers,” *Comput Secur*, vol. 121, p. 102843, 2022.

[32] L. Li, X. Qi, T. Xie, and B. Li, “Sok: Certified robustness for deep neural networks,” *arXiv preprint arXiv:2009.04131*, 2020.

[33] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, “Ai2: Safety and robustness certification of neural networks with abstract interpretation,” in *2018 IEEE symposium on security and privacy (SP)*, 2018, pp. 3–18.

[34] F. Tambon *et al.*, “How to certify machine learning based safety-critical systems? A systematic literature review,” *Automated Software Engineering*, vol. 29, no. 2, p. 38, 2022.

[35] C. Urban and A. Miné, “A review of formal methods applied to machine learning,” *arXiv preprint arXiv:2104.02466*, 2021.

[36] L. Pulina and A. Tacchella, “An abstraction-refinement approach to verification of artificial neural networks,” in *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*, 2010, pp. 243–257.

[37] G. Singh, T. Gehr, M. Püschel, and M. Vechev, “Boosting robustness certification of neural networks,” in *International conference on learning representations*, 2019.

[38] J. Cohen, E. Rosenfeld, and Z. Kolter, “Certified adversarial robustness via randomized smoothing,” in *international conference on machine learning*, 2019, pp. 1310–1320.

[39] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, “Fast and effective robustness certification,” *Adv Neural Inf Process Syst*, vol. 31, 2018.

[40] Z. Wang, C. Huang, and Q. Zhu, “Efficient global robustness certification of neural networks via interleaving twin-network encoding,” in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 1087–1092.

[41] M. Andriushchenko, F. Croce, N. Flammarion, and M. Hein, “Square attack: a query-efficient black-box adversarial attack via random search,” in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII*, 2020, pp. 484–501.

[42] S. Calzavara, L. Cazzaro, and C. Lucchese, “AMEBA: An Adaptive Approach to the Black-Box Evasion of Machine Learning Models,” in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 292–306.

[43] Y. Bai, Y. Wang, Y. Zeng, Y. Jiang, and S.-T. Xia, “Query efficient black-box adversarial attack on deep neural networks,” *Pattern Recognit*, vol. 133, p. 109037, 2023.

[44] M. Melis, A. Demontis, M. Pintor, A. Sotgiu, and B. Biggio, “secml: A python library for secure and explainable machine learning,” *arXiv preprint arXiv:1912.10013*, 2019.

[45] Y. LeCun, “The MNIST database of handwritten digits,” *http://yann. lecun. com/exdb/mnist/*, 1998.

[46] L. Deng, “The mnist database of handwritten digit images for machine learning research [best of the web],” *IEEE Signal Process Mag*, vol. 29, no. 6, pp. 141–142, 2012.

[47] Y. LeCun, Y. Bengio, and others, “Convolutional networks for images, speech, and time series,” *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.