Robustness against Adversarial Attacks on Deep Neural Networks

By Yi-Ling Liu

A thesis submitted to the Department of Computing
Imperial College London
in fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing

Imperial College London

October 2021

ABSTRACT

While deep neural networks have been successfully applied in several different domains, they exhibit vulnerabilities to artificially-crafted perturbations in data. Moreover, these perturbations have been shown to be transferable across different networks where the same perturbations can be transferred between different models. In response to this problem, many robust learning approaches have emerged. Adversarial training is regarded as a mainstream approach to enhance the robustness of deep neural networks with respect to norm-constrained perturbations. However, adversarial training requires a large number of perturbed examples (e.g., over 100,000 examples are required for MNIST dataset) trained on the deep neural networks before robustness can be considerably enhanced. This is problematic due to the large computational cost of obtaining attacks. Developing computationally effective approaches while retaining robustness against norm-constrained perturbations remains a challenge in the literature.

In this research we present two novel robust training algorithms based on Monte-Carlo Tree Search (MCTS) [1] to enhance robustness under norm-constrained perturbations [2,3]. The first algorithm searches potential candidates with Scale Invariant Feature Transform method and makes decisions with Monte-Carlo Tree Search method [2]. The second algorithm adopts Decision Tree Search method (DTS) to accelerate the search process while maintaining efficiency [3]. Our overarching objective is to provide computationally effective approaches that can be deployed to train deep neural networks robust against perturbations in data. We illustrate the robustness with these algorithms by studying the resistances to adversarial examples obtained in the context of the MNIST and CIFAR10 datasets. For MNIST, the results showed an average training efforts saving of 21.1% when compared to Projected Gradient Descent (PGD) and 28.3% when compared to Fast Gradient Sign Methods (FGSM). For CIFAR10, we obtained an average improvement of efficiency of 9.8% compared to PGD and 13.8% compared to FGSM. The results suggest that these two methods here introduced are not only robust to norm-constrained perturbations but also efficient during training.

In regards to transferability of defences, our experiments [4] reveal that across different network architectures, across a variety of attack methods from white-box to black-box

and across various datasets including MNIST and CIFAR10, our algorithms outperform other state-of-the-art methods, e.g., PGD and FGSM. Furthermore, the derived attacks and robust models obtained on our framework are reusable in the sense that the same norm-constrained perturbations can facilitate robust training across different networks. Lastly, we investigate the robustness of intra-technique and cross-technique transferability and the relations with different impact factors from adversarial strength to network capacity. The results suggest that known attacks on the resulting models are less transferable than those models trained by other state-of-the-art attack algorithms.

Our results suggest that exploiting these tree search frameworks can result in significant improvements in the robustness of deep neural networks while saving computational cost on robust training. This paves the way for several future directions, both algorithmic and theoretical, as well as numerous applications to establish the robustness of deep neural networks with increasing trust and safety.

ACKNOWLEDGMENTS

My sincere gratitude goes to my supervisor Alessio Lomuscio providing me the opportunity to perform research in a highly interesting field. Thank you for honing my skills as a researcher and instructing me in deep work as well as rigor that are key ingredients to good science.

Furthermore, I would like to express my warmest thanks to my colleagues in the Verification of Autonomous Systems (VAS) group for their support and their contribution to an inspiring and pleasant atmosphere during these four years.

Also, I would like to thank Computing Department of Imperial College London for the numerous opportunities I have enjoyed throughout my studies.

Then, I say "Thank You!" to Shuang Xia, Chen Chen, Yuliya Gitlina and Haoyang Wang for making London feel more homely.

I would like to send my most sincere thanks to Yao-Chun Fang for his understanding especially during the strenuous moments.

Last, but certainly not least, I would like to say "感恩！", a special thanks to my family and my friends back home. This thesis is dedicated to my family.

DECLARATION OF ORIGINALITY

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

COPYRIGHT

Contents

List of Figures

List of Tables

List of Algorithms

Symbols and Acronyms

Symbols

x_i	the i-th original (clean, unmodified) input data
y_i	the label of i-th input data
x'_i	the i-th adversarial example (modified input data)
y'_i	the label of i-th adversarial example
$\mathbb{R}^D$	the D-dimensional Euclidean space
$f(\cdot)$	the deep learning model for an image classification task ($f(\cdot) \in F : \mathbb{R}^D \rightarrow y$)
θ	the parameters of a deep learning model $f(\cdot)$
$\mathcal{L}(\theta, x, y)$	the loss function (e.g., cross-entropy) of a model $f(\cdot)$
η	the differences between original data x_i and adversarial example x'_i
$\|\cdot\|_p$	the distance metric in p norm on suitable real vector spaces given by the p-th root of the sum of the vector components
$\bigtriangledown f$	the gradient vector of a deep learning model f
sign	the odd mathematical function that extracts the sign of a real number
ϵ	the measurement constraint of perturbations applied to adversarial examples
$\mathcal{Z}$	the softmax function which is used to obtain the probability of the class y

Acronyms

DNN	Deep Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
FFNN	Feedforward Neural Network
ReLU	Rectified Linear Unit
SVM	Support Vector Machine
PCA	Principal Component Analysis
LR	Linear Regression
DT	Decision Tree

kNN	k-Nearest Neighbour
DCN	Deep Contractive Network
DAE	Denoising Autoencoder
FGSM	Fast Gradient Sign Method
BIM	Basic Iterative Method
JSMA	Jacobian Saliency Map Attack
PGD	Projected Gradient Descent
AT	Adversarial Training
MCTS	Monte-Carlo Tree Tearch
SIFT	Scale Invariant Feature Transform
MAT	MCTS-based Adversarial Training
DTS	Decision Tree Search

Chapter 1

Introduction

Work on artificial neural networks, commonly referred to as "neural networks", has been motivated right from its inception by the recognition that the human brain computes in an entirely different way from the conventional digital computer. The brain is a highly complex, nonlinear, and parallel computer (information-processing system). It can organise its structural constituents, known as neurons, to perform certain computations (e.g., pattern recognition, perception, and motor control) many times faster than the fastest digital computer in existence today. Consider, for example, human vision, which is an information-processing task. It is the function of the visual system to provide a representation of the environment around us and, more important, to supply the information we need to interact with the environment. To be specific, the brain routinely accomplishes perceptual recognition tasks (e.g., recognising a familiar face embedded in an unfamiliar scene) in approximately 100-200 ms, whereas tasks of much lesser complexity take a great deal longer on a powerful computer.

The human vision system is surprisingly robust when it comes to changes in object appearance, shape, and pose and reliably perceive and accurately navigate the world around us without realising the difficulty of the task. An open question in computer vision is: can computers understand the world around us with the same accuracy and reliability as humans do? Creating such algorithms is the main exploration in computer vision. Deep neural networks (DNNs) have emerged as one universal representation in

recent years and mimic information processing system in the human brain. In its most general form, a neural network is a machine that is designed to model how the brain performs a particular task or function of interest; the network is usually implemented by using electronic components or is simulated in software on a digital computer. DNNs consist of multiple processing layers that extract a higher-level representation of the data, which is similar to the visual cortex. With recent advances, deep neural networks have surpassed human performance on the image classification task [5], speech recognition [6] and machine translation [7]. DNNs have been applied to different fields from speech recognition to image recognition and play an important role in several applications.

1.1 Research Motivation

The success of deep neural networks in countless applications is undeniable. There are several applications in many different fields, from computer vision [8], natural language processing [9], face recognition in smartphone devices [10], automated image organisation in image directories [11], image classification for websites with large visual databases [12] to visual recognition system in self-driving cars [13]. Although deep neural networks have achieved significant experimental results in image recognition, many existing image classifiers are highly vulnerable to small imperceptible perturbations in the input data. The intriguing result is that the changes required to shift the prediction of DNNs are invisible to the human eye. For example, Fig. 1.1 shows an example of the image that was classified correctly as a "school bus" but was misclassified as a "ostrich" after applying small perturbations. Yet, the perturbed image still looks like a "school bus" for human perceptions. These examples, named adversarial examples, are blind spots or optical illusions for deep neural networks [14].

Adversarial examples are comprised of applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect classification result with high confidence [16,17]. Szegedy et al. first proposed this intriguing discovery about the image misclassification problem with small perturbations on the images and successfully fooled deep neural networks with

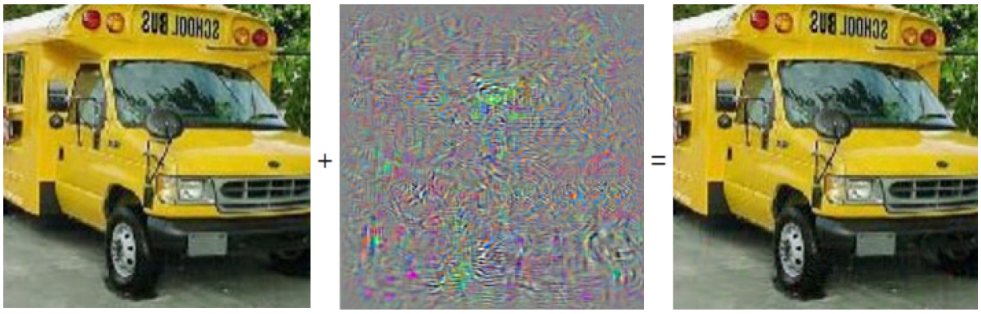


Fig. 1.1: An adversarial example for GoogleNet [15]. GoogleNet correctly classified the image on the left as a "school bus". An adversarial image (shown on the right) was constructed by adding small imperceptible perturbation (shown in the middle) to the original image. The neural network misclassified the adversarial image as a "ostrich". The original and adversarial images are indistinguishable to the human eye, yet, GoogleNet predicted a different label for the adversarial image.

high confidences [17]. In many situations, like image classification, these modifications can be so subtle that a human observer does not even notice the modification, yet the classification outcome is still incorrect. Adversarial examples will raise safety concerns in self-driving cars while traffic signs or road conditions are misclassified due to scratches or changes to camera angle or lighting conditions [18,19], e.g., speed limits changed from 20 to 200 miles. Moreover, it has been shown that the same adversarial examples tend to fool different models with different architectures which are trained on different training sets in similar sequences, which is called transferability. In general, the fact that they can be generated by simple and structured procedures and are common to different models can be used to perform attacks between models [20].

1.2 Research Challenges and Existing Approaches

There are several indicators of image recognition methodologies to evaluate the robustness under adversarial attacks. One is *accuracy* - the image classifier is required to endure some perturbations and still recognises images correctly. Another key indicator is *learning ability* - the model itself could learn from the existing training dataset and the newly generated dataset efficiently. An ideal image recognition methodology usu-

ally contains both high accuracy and self-learning ability. The primary reason for the existence of adversarial examples is the linear behaviour in high-dimensional spaces of DNNs [16,17,18,19,14]. For high dimensional weights of DNNs, we can make many infinitesimal changes to the input that will add up to one large change to the output so even a minor perturbation will make considerable contributions for the results. Therefore, the major challenge in robustness is how to endure perturbations whilst preserving the accuracy of a classifier.

In response to this vulnerability, a growing body of work has focused on improving the robustness of DNNs [21,22,23,24,25,26]. In particular, the literature concerning adversarial robustness has sought to improve robustness to small, imperceptible perturbations of data. There are two categories of existing countermeasures for adversarial examples. One of these is reactive countermeasure which defends after adversaries generate adversarial examples and this usually attempts to detect adversarial examples from inputs after deep neural networks are established, e.g., adversarial detecting [27]. The other category is proactive countermeasure which enhances deep neural networks more robust before adversaries generate adversarial examples, e.g., adversarial re-training [28]. In the next chapter, we will discuss more details about two types of reactive countermeasures (adversarial detecting [27] and input reconstruction [20]) and two kinds of proactive countermeasures (network distillation [21] and adversarial re-training [28]). Although these methodologies improve the robustness of image recognition to some extent, there are still some existing limitations under different attacks.

To this end, the adversarial robustness literature has developed novel robust training algorithms, i.e. adversarial training [29], which typically incorporates norm-constrained perturbations in a robust optimisation formulation where the norm-constrained perturbations are the distances from the input data to the perturbed input. Adversarial training has provided a rigorous framework for understanding, analysing, and improving the robustness of DNNs considering norm-constrained perturbations. However, adversarial training requires learning via a large number of perturbed images (e.g., over 100,000 examples are required for MNIST dataset aforementioned) before robustness is obtained. This is expensive and time-consuming. Therefore, developing computationally effective, robust training approaches is a topic of interest. This raises the key research question

of learning models robust to adversarial examples in a computationally effective manner and is also our main target in this thesis.

1.3 Contributions and Publications

The main contributions of this thesis can be categorised in a threefold way:

1. Conceptual:

   - We introduce two classes of attack methods: *Monte-Carlo Tree Search* (MCTS) attack and *Decision Tree Search* (DTS) attack. The benefit of MCTS is on the analysis of the most promising moves, expanding the search tree based on the search space. Through this method, the computational efforts can reduce during the search of adversarial examples while maintaining robustness enhancement. More details about MCTS will be introduced in Chapter 3.

   - We combine these attack methods and introduce two robustness countermeasures efficiently: *MCTS-based adversarial training* (MAT) and *decision tree search robust optimisation* (DTSRobust).

   - We evaluate the *transferability* with our MCTS-based robustness countermeasure, i.e. whether our method can prevent adversarial examples transferred between different models and improve model generalisation.

2. Methodological:

   - We introduce a novel, systematic methodology of combining MCTS with adversarial training to improve the training efficiency and robustness in deep neural networks. With the adversarial examples generated from the MCTS-based attack method, the model can accelerate the convergence rates.

   - We introduce a novel, systematic methodology of combining DTS with robust optimisation to enhance the training efficiency and robustness in deep neural

networks. The adversarial examples generated from the DTS attack method are reusable and able to apply in different models. For this reason, the computational efforts can save to some level and the model robustness is competitive with the state-of-the-art methods.

- Using the MCTS-based attack method aforementioned, we introduce a black-box adversarial training algorithm and evaluate systematic transferability between different models to verify if our method can provide robustness and reduce transferability.

3. Practical:

- We introduce MCTS-based Adversarial Training (`MAT`), a toolkit implementing the work aforementioned.
- We introduce Decision Tree Search robust optimisation (`DTS`), a toolkit implementing the second work aforementioned.
- We present implementations of transferability evaluations in `MAT` and give results compared with the state-of-the-art methods.

The results presented in this thesis have previously appeared, in a shorter form, in the following papers [2,4,3]:

- Yi-Ling Liu, Alessio Lomuscio. An MCTS-based Adversarial Training Method for Image Recognition. Proceedings of the 32nd International Joint Conference on Neural Networks (IJCNN). Budapest, Hungary. 1-8(2019). IEEE Press. DOI: 10.1109/IJCNN.2019.8852337.
- Yi-Ling Liu, Alessio Lomuscio. Robustness Learning via Decision Tree Search Robust Optimisation. Proceedings of the 32nd British Machine Vision Conference (BMVC). United Kingdom, 2021. BMVA Press.

- Yi-Ling Liu, Alessio Lomuscio. A Method for Robustness against Adversarial Attacks on Deep Neural Networks. Proceedings of the 33rd International Joint Conference on Neural Networks and IEEE World Congress on Computational Intelligence (WCCI). Glasgow, United Kingdom. 1-8(2020). IEEE Press.
  DOI: 10.1109/IJCNN48605.2020.9207354.

This thesis builds based on the above papers by combining the theoretical results into a unified presentation. Further, the individual toolkits presented in the papers are combined to support all the different platforms.

1.4 Thesis Structure Overview

The remainder of this thesis is organised as follows. Chapter 2 first outlines some machine learning basics, introduces how adversarial examples are generated with different approaches and explains two genres of countermeasures with existing limitations. We now provide an overview of the research problems investigated and three major research contributions fulfilled relating to these problems in the thesis.

Chapter 3 describes our MCTS-based robustness method and how we can improve the robustness of image recognition from the scope of a global perspective. Chapter 4 introduces a novel decision tree search robust optimisation method to cope with different adversarial attacks. Chapter 5 describes how we aim to conquer the phenomenon of transferability on DNNs and further improve the robustness. In the following, we give a brief summary in each chapter.

- In Chapter 2, we first define the problem and introduce neural networks with which we experiment. We then present the existing state-of-the-art methods from attack methods to countermeasures toward robustness. For the attack methods, we introduce five genres of attacks and then conclude for these attacks. For the countermeasures against attacks, we separate them into reactive and proactive types with two different methods for each of these types. Lastly, we discuss the existing limitations toward these state-of-the-art methods.

- In Chapter 3, we present an adversarial training algorithm based on Monte-Carlo Tree Search. We illustrate the robustness of the algorithm by studying its resistance to adversarial examples in the context of the MNIST and CIFAR10 datasets. For MNIST, after 2000 epochs the experimental results showed an average improvement of efficiency of 21.1% when compared to PGD. For CIFAR10, after 7000 epochs we obtained an average improvement of efficiency of 9.8% compared to PGD. We further compare the robustness of the algorithm against various attack methods. The results suggest that the adversarial training method here introduced is not only robust against adversarial examples but also efficient during training.

- In Chapter 4, we present a novel method for robustness training for ReLU-based deep neural networks. The method involves decision tree search targeting the worst-case data points to generate adversarial examples. We combine the decision tree search method with robust optimisation to train a robust model while maintaining accuracy at comparably lower computational effort than the state-of-the-art methods. The efficiency is obtained by focusing on small regions centred around the input that have significant potential to generate adversarial samples. We implemented the resulting method in the toolkit DTSRobust, which was evaluated against the state-of-the-art defence methods on MNIST and CIFAR10 datasets. In experiments, DTSRobust achieved a 14.2% gain on efficiency against the state-of-the-art defence methods in MNIST and 10.3% of that in CIFAR10 while maintaining similar accuracy. .

- In Chapter 5, we present a novel black-box adversarial training algorithm to defend against the state-of-the-art attack methods in machine learning. To search for an adversarial attack, the algorithm analyses small regions around the input that are likely to make significant contributions to the generation of adversarial samples. Unlike some of the literature in the area, the proposed method does not require access to the internal layers of the model and is therefore applicable to applications such as security. We report the experimental results obtained on models of different sizes built for the MNIST and CIFAR10 datasets. The results suggest that known attacks on the resulting models are less transferable than those models trained by the state-of-the-art attack algorithms.

- In Chapter 6, we first summarise the achievements for each chapter and then discuss some related works from different perspectives, i.e. perturbation-based adversarial robustness, a broader view of robustness, robustness in generative models and equivariance and invariance to noises in computer vision. Lastly, we summarise the overall contributions and discuss some future works in different directions, e.g., learning a library of different noise models, model-based algorithms and architectures, applications beyond image classification and theoretical foundations.

Before diving into technical details, we review machine learning basics and neural networks in the next chapter. Then, we introduce different genres of literatures relating to this problem and discuss the existing limitations of these approaches.

Chapter 2

Background and Literature Review

Humans can recognise objects and places seamlessly. In their first efforts in the 60s, researchers attempted to construct programmable rules for extracting useful information from images. However, it soon became clear that the complexity of visual data makes the task of designing extraction rules very challenging. The world around us seems just too complex to be described with simple rules. The solution is to enable computers to learn directly from the data. Machine learning algorithms allow computers to learn from experience without being explicitly programmed to perform the task at hand. This chapter provides a basic background summary about machine learning in general. We cover empirical risk minimisation, discuss regularisation methods, and the issue of data overfitting. We then describe the basic principles underlying deep neural network models. Then, we present some recent attacks on deep neural networks and defences against adversarial noises. Lastly, we highlight some limitations that occurred in these state-of-the-art methods.

2.1 Machine Learning Basics

Artificial intelligence (AI) has captivated our imagination for decades. If developed, it might be the greatest humankind invention. The first attempts to program intelligent

machines can be traced back to the 1960s by the British logician and computer pioneer Alan Mathison Turing. After the initial optimism, the researchers soon learned that the complexity of the real world makes it infeasible to program a computer with an explicit set of rules for performing the task. Learning in some form is necessary. While the way a machine learns is different from the way a human learns, machine learning has been largely inspired by the principles of continuous improvement from experience in human learning. Machine learning research can thus potentially shed light on the principles that govern human intelligence.

Machine learning is the study of algorithms that enable computers to learn from data. A commonly cited and more formal definition of machine learning is "A computer program is said to learn from experience `E` with respect to some class of tasks `T` and performance measure `P` if its performance at tasks in `T`, as measured by `P`, improves with experience `E`" [30]. We can describe many learning problems in this framework. As an example, let's consider a basic image classification task:

Task `T`: is to recognise objects in the image.

Performance measure `P`: is a percentage of correctly classified images.

Experience `E`: is a dataset of images with the target labels.

Machine learning algorithms can be divided into three categories according to the amount of information available about the target: supervised learning, unsupervised learning, and reinforcement learning. In this work, we consider supervised learning problems, in particular, image classification problems. For supervised learning, the goal is to learn a mapping from the input to the output for the image classifier $f : x \rightarrow y$. x is an image $x \in \mathbb{R}^D$, where D is the dimensionality of the image. y is the target output for image classifier f with k-labels, where $y \in \{\ell : 1, ..., k\}$. Output is usually encoded using 1-hot coding vector of length k with 1 at position ℓ if $y = \ell$ and 0 at all other positions. The loss function for classifier f is expressed as $\mathcal{L}(x, \theta, y)$ (or $J(x, \theta, y)$) - a function that quantifies and evaluates the quality of fitness between the parameter θ of f and the observations (x, y).

Machine learning models can be categorised into *parametric* and *non-parametric*

models. An example of a *non-parametric* model with no assumption for underlying data distribution is the k-nearest neighbour classifier. In our work, we consider parameter models. A parametric family of models defines a set of functions $\mathcal{F} = \{f_\theta \mid \theta \in \Theta\}$, where $f_\theta = f_\mathcal{F}(x, \theta)$ is a mapping from the input space to the target space; $\theta \in \Theta$ are trainable model parameters, e.g. neural network weights. Additionally, the function family $\mathcal{F}$ can depend on non-trainable parameters or *hyperparameters*, e.g., the type and the number of layers in a neural network. A parametric family restricts the class of functions the model can represent. In this way, we provide an inductive bias to the model for learning about task `T`.

Each member $f_\theta \in \mathcal{F}$ represents a particular instance of a machine learner. The goal of learning is to find the most suitable member f^* of a parametric family $\mathcal{F}$ for a given task `T` and a given performance measure `P`. For parametric models, this is equivalent to finding optimal parameters $\theta^* \in \Theta$. Assume that training data was drawn independently and identically distributed (i.i.d.). If π is data distribution from which data was i.i.d. drawn, we can formalise learning as an optimisation problem with expectation $\mathbb{E}$:

$$f^* = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(x,y)\sim\pi}[\mathcal{L}(f(x, \theta), y)] \tag{2.1}$$

Data distribution π is not available for most problems of interest. Instead, data is sampled from the empirical or training distribution $\mathbb{D}$:

$$\begin{aligned} \hat{f} &= \arg\min_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(x_i, \theta), y_i) \\ &= \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(x,y)\sim\mathbb{D}}[\mathcal{L}(f(x, \theta), y)], \end{aligned} \tag{2.2}$$

where m is the number of training examples.

Does the solution of Equation (2.2) converge to the solution of Equation (2.1)? Fortunately, we can show that under mild conditions on the function family $\mathcal{F}$ empirical estimate $\hat{f}$ convergences to the optimal solution f^* in the limit of the number of examples $\lim_{m\to\inf} \hat{f} = f^*$. This principle is known as the *empirical risk minimisation* principle. Empirical risk minimisation is the foundation of many machine learning algorithms, in-

cluding neural networks. To find a solution to Equation (2.2), we can use any optimisation technique, e.g. gradient descent, Newton's method, or a method tailored for the problem's structure. Next, we introduce neural networks as one example of a parametric machine learning model, which is a particular focus in this thesis.

2.2 Neural Networks

Artificial neural network (ANN) is a parametric model, which has been introduced to machine learning over decades. Their creation and development were loosely inspired by the layered structure of the human brain. Neural networks for pattern recognition do not aim to imitate the human mind intrinsically, yet, many of the neural network terminologies and ideas have been borrowed from neuroscience. Spiking neural networks (SNNs) provide a much more accurate approximation of the computations in the human brain [31]. However, SNNs are difficult to train due to the non-differentiability of the spiking activation function.

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. It consists of neurons, synapses, weights, biases, and functions and can be trained like any other machine learning algorithm. Neural network computation is represented as a directed graph of connected processing units or neurons. Each neuron receives an output from the adjacent neurons as an input and performs some transformation of the data, e.g., weighted sum or output thresholding. An output signal of the neuron is passed to the connected neurons. The neurons that perform related computations are organised in layers. For simplicity, we describe neural networks as a sequence of layer transformations. The neural networks can be categorised into two groups: feedforward neural networks (FFNN) and recurrent neural networks (RNN). Recurrent neural networks (RNNs), in which data can flow in any direction, are used for applications such as language modeling while FFNNs are used in computer vision. Due to the vast number of parameters, FFNNs can be reduced by using prior knowledge about the structure of natural images and be represented as convolutional neural networks (CNNs) used in computer vision. In the section, we first introduce feedforward neural networks

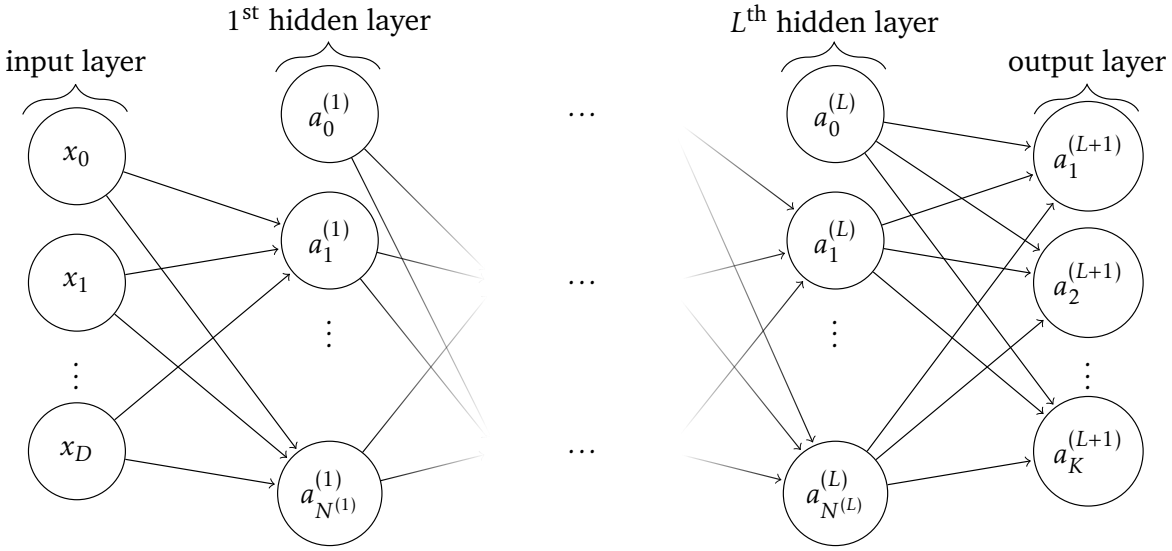


Fig. 2.1: Feedforward neural network with L fully connected layers. Each neuron is connected to all neurons in the previous layer, but the neurons in the same layer do not share connections.

(FFNNs) and then convolutional neural networks (CNNs) for the study of robustness to deep neural networks in computer vision.

2.2.1 Feedforward Neural Networks

We start with an introduction to feedforward neural networks. Each layer of a feedforward neural network transforms an output from the previous layer and passes the transformed output as an input to the next layer. The architecture of an L-layer fully connected feedforward neural network is represented in Fig. 2.1.

For an input vector $x_t \in \mathbb{R}^D$, each hidden layer transforms its input vector from the previous layer to the next layer by applying an affine transform as follows:

$$\begin{aligned} z^0 &= x_{t\ (t:\ 0\ldots D)}, \\ a^{(l+1)}_{i\ (i:\ 1\ldots N^{(l)})} &= \sum_{j=1}^{N^{(l)}} w_{ij}^{(l)} z_j^{(l)} + b_i^{(l)}, \\ z_i^{(l+1)} &= \sigma(a_i^{(l+1)}), \end{aligned} \tag{2.3}$$

where $N^{(l)}$, $W^{(l)}$ and $b^{(l)}$ ($a_0^{(l)}$ unconnected in Fig. 2.1) are the numbers of nodes, a weight

matrix, and a bias vector of the l-th layer ($l = 1$ to L), respectively, and $\sigma(a_i^{(l+1)})$ is a nonlinear ReLU activation function. The Rectified Linear Unit (ReLU) is defined as $ReLU(x_t) = max(0, x_t)$, where the output is the maximum between 0 and the input positive value. In the last layer, the *softmax* function is used to obtain the probability of the class y_t, which is formulated as:

$$softmax(x_t) = \frac{exp(w_{y_t} a^{(L)})}{\sum_{n=1}^{N^{(L)}} exp(w_n a^{(L)})}, \tag{2.4}$$

where w_{y_t} is the weight matrix for class y_t, $N^{(L)}$ is the number of nodes in the last layer and $a^{(L)}$ is the activation values in the last layer. In summary, a DNN model θ is defined by the set of weight matrices W, bias vectors b, and a nonlinear activation function $\sigma(x)$ for an input x as follows:

$$\theta = \{W, b, \sigma(x)\}, \tag{2.5}$$

where $W = \{W^{(1)}, ..., W^{(L)}\}$ and $b = \{b^{(1)}, ..., b^{(L)}\}$ [32,33].

A fully connected layer is a basic component for building a neural network. However, the number of layer parameters limits its applications to vision problems. Consider a fully connected layer with n input and k output units. The number of the layer parameters is scaled as $(n \times k + k) \cdot L$, where L is the number of neural network layers. It is prohibitively large for image data (where $n,\ k \sim 1000$) and leads to overfitting even for simple computer vision tasks. The number of parameters can be reduced by using prior knowledge about the structure of natural images. Next, we introduce convolutional layers that use prior information about images.

2.2.2 Convolutional Neural Networks

Hubel and Wiesel [34] discovered that neurons in the human visual cortex are organised in a complex arrangement of two types of cells: simple and complex cells. Simple cells only respond to the specific edge-like pattern within their receptive field, while complex cells are locally invariant to the pattern's exact location. Discoveries of the simple and complex cells and selective receptive fields in the visual cortex had led to the development of convolutional neural networks [35]. Convolutional neural networks implement the idea of simple and complex cells using convolutional and pooling layers, respectively.

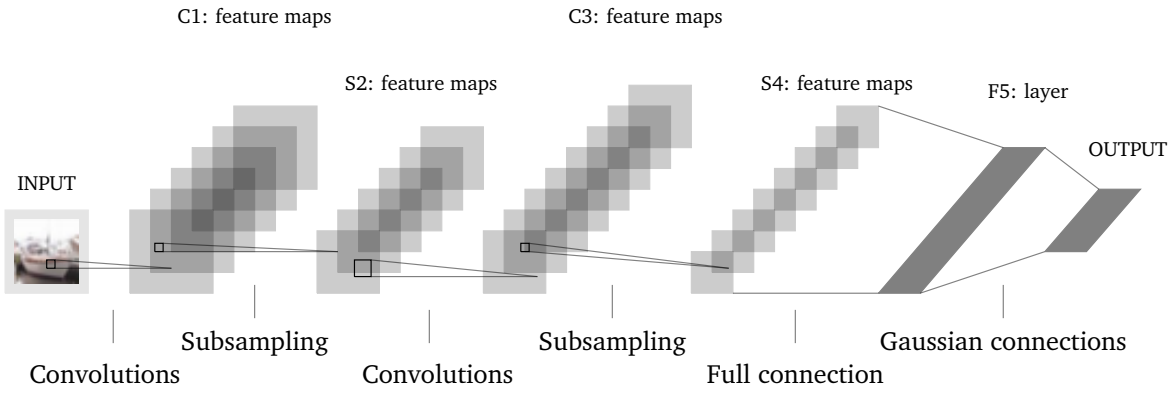


Fig. 2.2: The architecture of a traditional convolutional neural network. The objective of the convolution layers is to extract the high-level features such as edges, from the input image. It need not be limited to only one convolutional layer. Conventionally, the first convolutional layer is responsible for capturing the low-level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well. The pooling layers (subsampling layers) are responsible for reducing the spatial size of the convolved features. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

We describe the operation that the convolutional and polling layer perform next and an example of a convolutional neural network is presented in Fig. 2.2.

A convolutional layer contains a set of learnable filters (or kernels) which have a small receptive field. The height and weight of the filters are smaller than those of the input volume. Each filter is convolved with the input volume to compute an activation map made of neurons. In other words, the filter is slid across the width and height of the input and the dot products between the input and filter are computed at every spatial position. The output volume of the convolutional layer is obtained by stacking the activation maps of all filters along the depth dimension. Since the width and height of each filter are designed to be smaller than the input, each neuron in the activation map is only connected to a small local region of the input volume. In other words, the receptive

field size of each neuron is small and equal to the filter size.

The local connectivity is motivated by the architecture of the animal visual cortex where the receptive fields of the cells are small. The local connectivity of the convolutional layer allows the network to learn filters that maximally respond to a local region of the input, thus exploiting the spatial local correlation of the input (for an input image, a pixel is more correlated to the nearby pixels than to the distant pixels). In addition, as the activation map is obtained by performing convolution between the filter and the input, the filter parameters are shared for all local positions. This sharing reduces the number of parameters for efficiency of expression, efficiency of learning, and good generalisation.

A pooling layer (subsampling layer) is usually incorporated between two successive convolutional layers. The pooling layer reduces the number of parameters and computation by down-sampling the representation. There are two types of pooling: max pooling and average pooling. Max pooling returns the maximum value from the portion of the image covered by the kernel. On the other hand, average pooling returns the average of all the values from the portion of the image covered by the kernel. Max pooling also performs as a noise suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, average pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, max pooling performs better than average pooling.

In Fig. 2.2, we demonstrate the architecture of a traditional convolutional neural network. The architecture of the original convolutional neural network, as introduced by LeCun et al. (1989), alternates between convolutional layers including hyperbolic tangent non-linearities and subsampling layers. In this illustration, the convolutional layers already include non-linearities and, thus, a convolutional layer actually represents two layers. The feature maps of the final subsampling layer are then fed into the actual classifier consisting of an arbitrary number of fully connected layers. The output layer usually uses softmax activation functions (Gaussian connections) in Equation (2.4).

2.3 Attacks on Deep Neural Networks

Adversarial examples on conventional machine learning models have been discussed for years [36,18,19,37,14,38]. Machine learning-based systems with handcrafted features are the main targets, such as spam filters, intrusion detection, biometric authentication, fraud detection, etc. [38]. Biggio et al. [16] first proposed a gradient-based approach to generate adversarial examples against linear classifier, support vector machine (SVM), and neural network. They also reviewed several proactive defences and reactive approaches to improve the security of machine learning models [39]. Compared with adversarial examples in deep neural networks, the difference is that their methods permit greater freedom to modify the data. The digit handwriting MNIST dataset was first evaluated under their attack although a human could easily distinguish from the adversarial digit images. In [17], they proposed a method to generate adversarial attacks for image classifications on deep neural networks with high confidence. This body of work has received particular attention in applications of deep neural networks to vision systems for autonomous vehicles, face recognition, and malware detection [13,10,40]. These developments make the security aspects of deep neural networks increasingly important. In particular, resistance to small perturbations to ensure robustness is now a key property of classifiers.

To evaluate the distance between the adversarial example and input data, the l_p norm and l_∞ norm are commonly used. For $1 \leq p < \infty$, the l_p norm of x_i is defined as $\|x_i\|_p = (\sum_{j=1}^{D} |x_i(j)|^p)^{1/p}$ and the l_∞ norm is defined as $\|x_i\|_\infty = max_j|x_i(j)|$. In this section, we illustrate and review some representative approaches for generating adversarial examples. Although some of these approaches are defeated by some countermeasures which will be introduced in the later section, we first introduce some state-of-the-art attacks to show how these attacks can achieve. These techniques are illustrated in the following sequences. We start from Fast Gradient Sign Method (FGSM) [29] as this method represents a fast method for generating adversarial examples. Jacobian-based Saliency Map Attack (JSMA) [41] is accomplished by searching the input feature of x that makes the most significant changes to the output. Deepfool method [36] follows the rule of searching the closest distance from original input to the decision boundary of adversarial examples. C&W's attack method [42] is effective for most of the existing

adversarial detecting defences so we will evaluate the resistance strength to this attack on our robustness methodology in later chapters. Projected Gradient Descent (PGD) [26] is regarded as a strong attack and competitive with C&W's attack method so we also evaluate our methodologies against it in the following chapters.

2.3.1 Fast Gradient Sign Method

Goodfellow et al. [29] proposed a fast method for generating adversarial examples called Fast Gradient Sign Method (FGSM). They only performed one step gradient update along the direction of the sign of gradient at each pixel. According to Goodfellow et al., given an adversarial input $x' = x + \eta$, where x is original input image and η stands for perturbations, the goal of an adversarial attack is to maximise the second term in the addition expressed as Equation (2.6):

$$\mathcal{L}(x', \theta, y') = \mathcal{L}(x, \theta, y) + (x' - x)\nabla_x \mathcal{L}(x, \theta, y), \tag{2.6}$$

where θ is the hyperparameters of a model, x' is the adversarial example, y' is the adversarial class, and ∇_x is the gradient with respect to x. Through minimising $\mathcal{L}(x', \theta, y')$ subjecting to $\|x' - x\|_\infty \leq \epsilon$, the required perturbation is derived as Equation (2.7), where $\|\cdot\|_\infty$ denotes the distance between two data samples with l_∞ norm and ϵ is the magnitude of the perturbation factoring the sign matrix. Increasing its value increases the likelihood of x' being misclassified by the classifier $f(\cdot)$, but on the contrary, it causes high vulnerable chances detected by humans.

$$\eta = \epsilon \cdot \text{sign}(\nabla_x \mathcal{L}(x, \theta, y)) \tag{2.7}$$

Fig. 2.3 shows an adversarial example on ImageNet image sets. The loss function in the figure is expressed as $J(x, \theta, y)$ which is the same as $\mathcal{L}(x, \theta, y)$ used in Equation (2.7). The original picture of a panda is classified as a gibbon with 99.3% confidence after applying small perturbations [29]. FGSM is imprecise at evaluating the model robustness, but it is extremely fast in practice. Due to its speed, FGSM can be used to improve the robustness of the model without significantly increasing the training time. This weakness of this method is the linear property in the high dimensional domain that can not resist

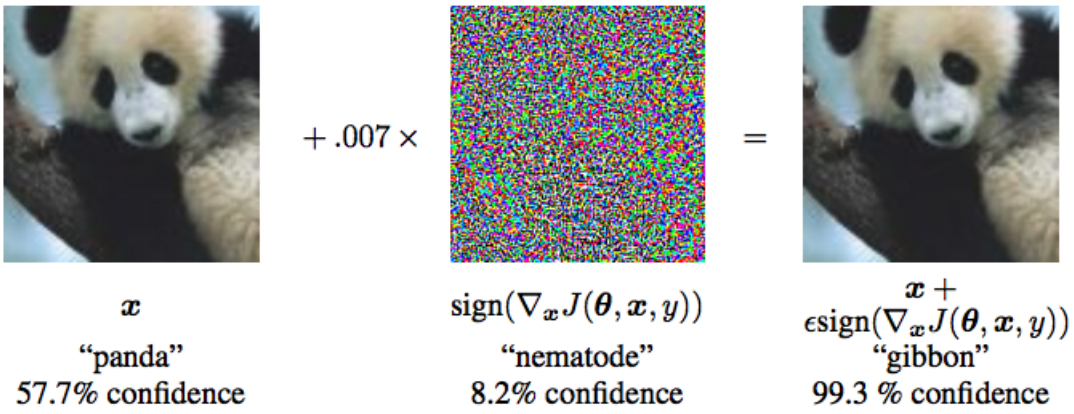


Fig. 2.3: An adversarial example for GoogleNet [15] generated by Fast Gradient Sign method [29]. GoogleNet correctly classified the image on the left as a "panda" class. An adversarial image (shown on the right) was constructed by adding small imperceptible perturbation (shown in the middle) to the original image. The neural network misclassified the adversarial image as a "gibbon". As we can see, the original and adversarial images are visually indistinguishable to the human eye, yet, GoogleNet predicted a different label for the adversarial image.

to adversarial examples, although the linear behaviour accelerates the training process. We review more details about the defence based on FGSM attack in Section 2.5.

2.3.2 Projected Gradient Descent

The Projected Gradient Descent (PGD) [28] attack is widely believed to be one of the most powerful attack methods. It is also competitive with C&W [42] and FGSM [29] attacks and is regarded as a strong attack so we will evaluate our methodologies against this method in the following chapters. This method adopts the multi-step variant of FGSM, i.e., projected gradient descent (PGD) on the negative loss function [28]:

$$x'_{t+1} = \text{Clip}_{x+\eta}\left(x'_t + \alpha \cdot \text{sign}(\nabla_x \mathcal{L}(x', \theta, y))\right), \tag{2.8}$$

where α is the variant step size at step t and the Clip function ensures that the output falls in the valid input value $(0 \ldots 255)$. PGD iteratively re-starts from many points in the l_∞ balls around data points from the respective evaluation sets. As PGD does not explicitly minimise the l_p-norm of the perturbation, to evaluate robustness at N distinct

thresholds ϵ, PGD attack needs to be restarted N times which linearly increases the cost of the attack.

2.3.3 Jacobian-based Saliency Map Attack

Papernot et al. [41] designed a saliency adversarial map called Jacobian-based Saliency Map Attack (JSMA) and this is suitable for targeted misclassification. According to their method, the saliency map was based on forward derivative and this provides the adversary the required information to cause the neural network misclassifying a given sample. More precisely, the adversary attempted to misclassify a sample x such that it is assigned a target class t which is not equal to the original class of x. To achieve this specification, the probability of target class t denoted by $\mathcal{Z}(x)_t$ must be increased while the probabilities $\mathcal{Z}(x)_j$ of all other classes $j \neq t$ decrease, until $t = \arg\max_j \ \mathcal{Z}(x)_j$, the maximum probability among all other classes. The model of saliency map is expressed as Equation (2.9):

$$\begin{aligned} &\text{Target Class:} && \alpha_{pq} = \sum_{i \in \{p,q\}} \frac{\partial \mathcal{Z}(x)_t}{\partial x_i}, \\ &\text{Other Classes:} && \beta_{pq} = \sum_{i \in \{p,q\}} \sum_{j \neq t} \frac{\partial \mathcal{Z}(x)_j}{\partial x_i} - \alpha_{pq}, \\ &&& (p^*,\ q^*) = \arg\max_{(p,q)} \ (-\alpha_{pq} \cdot \beta_{pq}) \cdot (\alpha_{pq} > 0) \cdot (\beta_{pq} < 0), \end{aligned} \tag{2.9}$$

where i is an input feature that stands for a pair of pixels p, q. The derivative of the target class is required to be positive for $\mathcal{Z}(x)_t$ to increase while the derivative of other classes is required to be negative for $\mathcal{Z}(x)_j$ to decrease. Therefore, the negative derivative of the target class and the positive derivative of other classes are then discarded. The product on α and β allows comparisons in all input features for convenience. Similar to JSMA, [43] found that it is possible to change the model's prediction by modifying a single pixel. To generate adversarial examples, they applied a differential evolution (DE) algorithm on the population of vector coordinates that modify a single pixel in the image.

Fig. 2.4 [41] demonstrates a saliency map, where large absolute values represent feature with a significant impact on the output when applied perturbations. This method emphasises features corresponding to larger forward derivative values in given input when

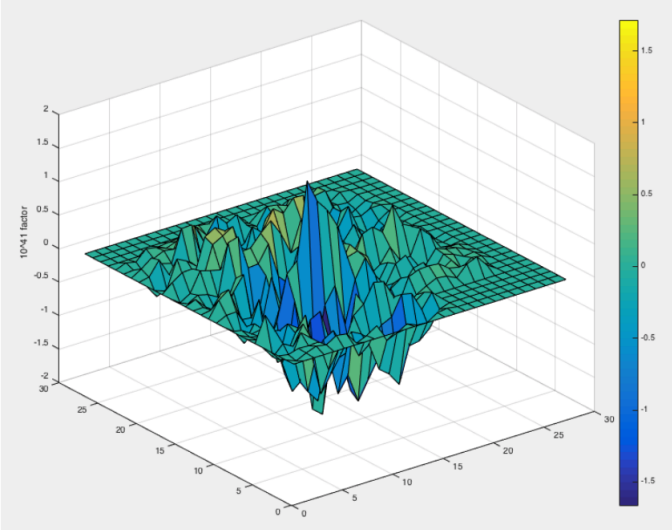


Fig. 2.4: Saliency map of a 784-dimensional input to the LeNet architecture [41].

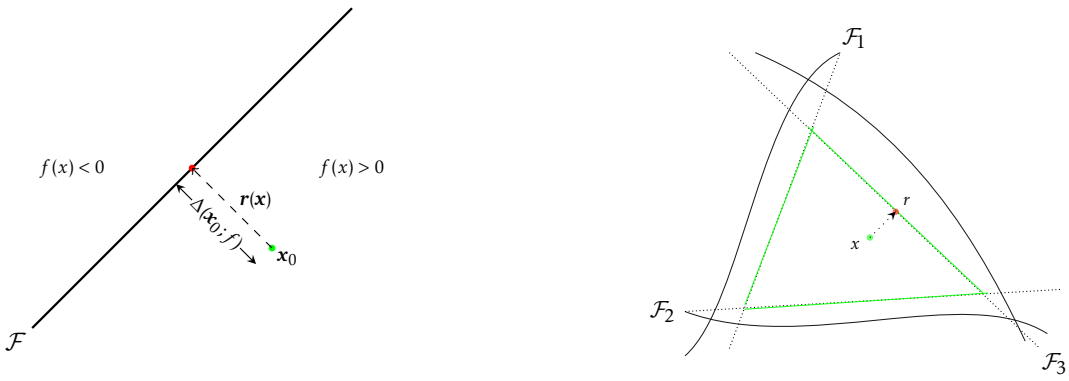
constructing a sample, making their search more efficient and ultimately leading to smaller overall distortions. Although they achieve a 97% adversarial success rate by modifying only 4.02% input features per sample on average, the runtime of this method is inefficient due to the significant computation cost in each pixel of input samples.

2.3.4 Deepfool Attack

DeepFool [36] searches the closest class boundary and takes one step in the direction of the closest decision boundary. The optimisation process stops as soon as adversarial perturbation is found. For a 2-class affine classifier in Fig. 2.5a, the Euclidean distance to the decision hyperplane can be defined exactly as follows: $r(x) = -\ (f(x)/\|w\|_2^2) \cdot w$, where w is the weight vector. For an n-class nonlinear classifier in Fig. 2.5b, Moosavi-Dezfooli et al. [36] approximated the current prediction region of DNNs using polyhedron. The distance to the class y decision boundary for the input x with the label y can be approximated as:

$$r_j(x, y) = \frac{|f_y(x) - f_j(x)|}{\|\nabla f_y(x) - \nabla f_j(x)\|} \cdot (\nabla f_y(x) - \nabla f_j(x)), \tag{2.10}$$

where $\nabla f_j(x)$ is the gradient of the unnormalised output for the class j at the input x. The authors computed the distance to the decision boundary for all classes $j \neq y$, in total $(k-1)$ calculations. Then, the minimal adversarial perturbation is the smallest



(a) Adversarial examples for a linear binary classifier.

(b) One example of polyhedron for x belonging to class 4.

Fig. 2.5: A minimal adversarial perturbation for a 2-class affine classifier and an n-class nonlinear classifier [36]. In Fig. 2.5a, the minimal adversarial perturbation for a 2-class affine classifier is the distance to the decision hyperplane. The region of the current prediction for an n-class nonlinear classifier, such as DNNs, can be approximated using a polyhedron. Then, the adversarial perturbation for an n-class nonlinear classifier is the shortest distance to the facets of the polyhedron in Fig. 2.5b.

distance among all classes: $r = \arg\min_{j \neq y} r_y(x, y)$. They applied the above approximation iteratively until the adversarial perturbation was generated.

Although DeepFool is a relatively fast and accurate method, it has some limitations: 1) it does not explicitly handle box constraints on the input $x + r \in [0, 1]^m$; 2) it does not explicitly minimise the norm of the perturbation. The optimisation process stops as soon as the adversarial perturbation is found. In [44], they introduced fast adaptive boundary attack (FAB), which addresses the limitations of DeepFool attack. FBA solves the box-constrained l_1, l_2, and l_∞-norm projection on the approximated decision hyperplane exactly. In addition, the authors introduced a biased backward step, which reduces the perturbation norm after each step. SparseFool (SF) [45] proposes a geometry inspired ℓ_1-norm attack, which uses DeepFool [36] attack as a subprocedure to estimate the local curvature of the decision boundary. They developed an efficient algorithm to compute ℓ_1-norm projection of the perturbation on the decision boundary subject to image box

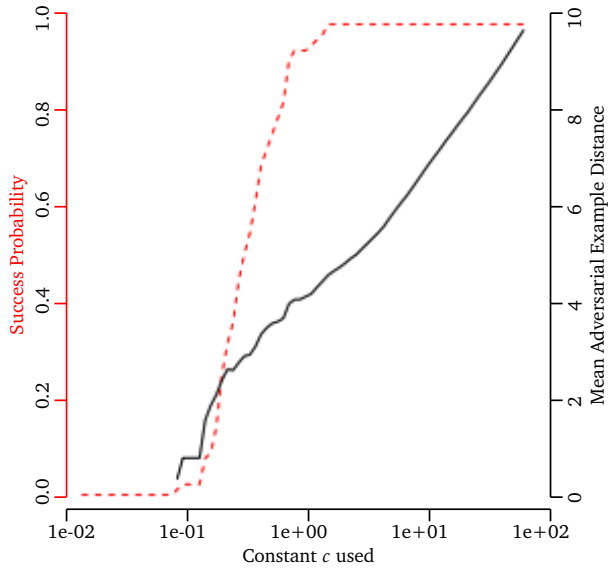


Fig. 2.6: Sensitivity on the constant c [42].

constraints.

2.3.5 Carlini & Wagner Attack

The C&W attack [42] is widely believed to be the most powerful attack which defeats the defence strategy of network distillation that we will illustrate in the later section 2.5. They explored how the choice of the minimiser, the surrogate loss function, and handling of the box constraints affect the attack's optimisation. In their experiments, they found that Adam optimiser [46] with multi-class hinge loss and hyperbolic tangent transformation of variables is the best attack on a set of image datasets. The authors also highlighted the importance of using strong attacks when we evaluate and compare defences based on robustness. In particular, they showed that defensive distillation [42,47] is not robust to adversarial examples. At the present moment, C&W is the recommended attack for the assessment of DNNs robustness to ℓ_2-norm perturbations [48].

In the work [42], Carlini and Wagner considered a wide variety of formulations, but we introduce here the one that performs best according to their evaluation. They define a new objective function g so that it satisfies Equation (2.11), where $g(x') \geq 0$ if and only

if $f(x') = y'$ (label of the adversarial class). From their seven objective functions g, one of the most effective functions evaluated by their experiments is expressed as Equation (2.12), where $\mathcal{Z}$ represents the softmax function and κ is one constant controlling the confidence.

$$\begin{aligned} \min \quad & \| \eta \|_p + c \cdot g(x+\eta) \\ s.t. \quad & x' = x + \eta \in [0,1]^n. \end{aligned} \tag{2.11}$$

$$g(x') = \max(\max_{i \neq y'}(\mathcal{Z}(x')_i) - \mathcal{Z}(x')_{y'} \ , \ -\kappa). \tag{2.12}$$

They also introduced a new variant ω to avoid box constraint satisfying $\eta = \frac{1}{2}(tanh(\omega)+1) - x$ and then explored an optimal c by binary search. However, they discovered that if the gradients of $\|\eta\|_p$ and $g(x+\eta)$ are not in the same scale, it would be hard to obtain an optimal constant c in all of the iterations. Fig. 2.6 demonstrates the sensitivity on the constant c. When $c < 0.1$, the success rate of attacks is close to 0. After $c > 1$, the attack becomes effective with a high success rate.

2.3.6 Summary of Attack Techniques

Goodfellow et al. [29] proposed a fast method for generating adversarial examples called Fast Gradient Sign Method (FGSM). They only performed one step gradient update along the direction of the sign of gradient at each pixel. This weakness of this method is the linear property in the high dimensional domain that can not resist to adversarial examples, although linear behaviour accelerates the training process. Papernot et al. [41] designed a saliency adversarial map called Jacobian-based Saliency Map Attack (JSMA) and this is suitable for targeted misclassification. Although they achieved 97% adversarial success rate by modifying only 4.02% input features per sample on average, the runtime of this method was inefficient due to the significant computation cost in each pixel of input samples. Moosavi-Dezfooli et al. [36] attempted to find the closest distance from original input to the decision boundary of adversarial examples. DeepFool provides fewer perturbations compared to FGSM and JSMA and it also reduces the intensity of perturbations instead of the number of selected features compared to JSMA. The C&W attack [42] is

widely believed to be the most powerful attack in l_2-norm perturbations which defeat the defence strategy of network distillation [21]. The PGD [28] attack plays an important role among popular state-of-the-art methods and is regarded as a powerful attack method in l_{inf}-norm perturbations. For the reason of its significant attack potential, we will evaluate our robustness under this attack method in the following chapters.

2.3.7 Adversarial Attacks in the Physical World

Initially, the threat of adversarial examples was considered frivolous for real-world applications. In [17], they argued that adversarial noise occupies low probability "pockets" in the input space that act as "blind spots" to DNNs. Because of that, it was not clear if adversarial examples are physically realisable. The problem of adversarial noise was not considered to be a serious security issue for real-world systems. However, since then, several methods have been introduced that are effective in producing physically feasible and imperceptible adversarial disturbances. In [18], they showed that it is possible to generate printable adversarial images that are consistently misclassified when shown to the camera of a cell phone. In [49], they generated a special pattern that misleads face recognition and face identification systems when printed on the specs frame. In [50,19], they introduced a road sign attack. They generated a special sticker that fools an object detection system when the sticker was placed on the road sign. [51] argued that adversarial examples are not a concern for objection detection in autonomous vehicles due to the changing physical conditions of a moving car. However, the findings in [50,19,52] suggest that it is not the case.

2.4 Reactive Countermeasures

Many DNNs are generally fragile to seemingly imperceptible changes to their input data [17] as aforementioned. Well-documented examples of such fragility to carefully-designed noise can be found in the context of image detection [53], video analysis [54], and traffic sign misclassification [50]. In response to this vulnerability, a growing body of work has focused on improving the robustness of DNNs [21,22,23,24,25,26]. In particular,

the literature concerning adversarial robustness has sought to improve robustness to small, imperceptible perturbations of data.

The disparity between the error on the test data and the error on the adversarial examples raises several questions regarding the limitations and weaknesses of the existing deep learning models. In this section and the following one, we review some recent defence algorithms for improving DNN robustness to adversarial noise. In particular, we cover reactive and proactive methods.

Countermeasures for adversarial examples contain two main categories of defence strategies that are reactive and proactive. This section introduces two representative reactive countermeasures which defend after attacks and attempt to detect adversarial examples from inputs after deep neural networks are established. The first genre is adversarial detecting [27] and the second one is input reconstruction [20].

2.4.1 Adversarial Detecting

Metzen et al. [27] created a small detector subnetwork for adversarial examples as an auxiliary network of the original neural network. The detector is a small neural network that was trained on the binary classification task to distinguish genuine data from data containing adversarial perturbations, i.e. the probability of the input being adversarial.

Fig. 2.7 demonstrates the structure of a detecting network, where **Conv** denotes a convolution layer, $\mathbf{Res}^{*\mathbf{5}}$ denotes a sequence of 5 residual blocks as introduced by He et al. [55], **GAP** denotes a global-average pooling layer, **Dens** is a fully connected layer and **MP** denotes max-pooling. Metzen et al. demonstrated that adversarial perturbations can be detected surprisingly well even though they are quasi-imperceptible to humans. However, detectability is 85% or more except for the "Iterative" l_2-based adversary in Fig. 2.8. For this adversary, the detector only reaches the chance level. Other choices of the detector's attachment depth, internal structure, or hyperparameters of the optimiser might achieve better results. This failure case emphasises that the detector has to detect very subtle patterns and the optimiser might get stuck in bad local optima or plateaus.

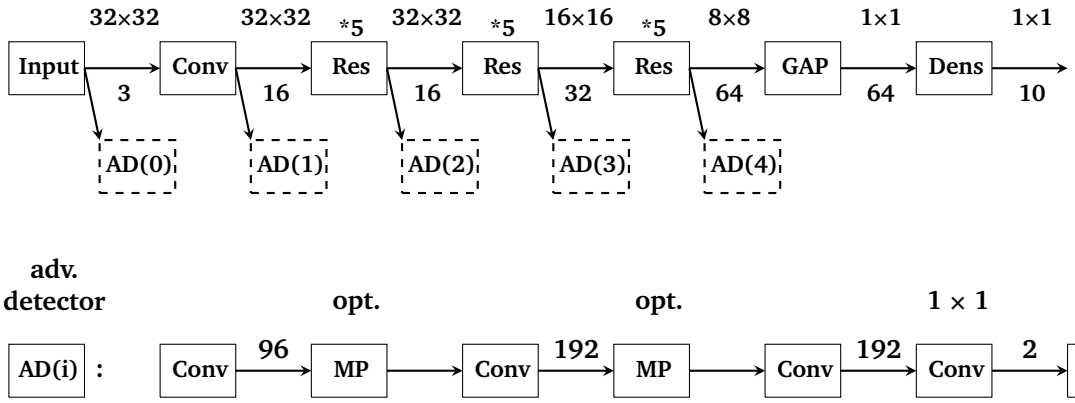


Fig. 2.7: (Top) ResNets used for classification. Numbers below arrows denote the number of feature maps and numbers on top of arrows denote spatial resolutions. **Conv** denotes a convolutional layer, **Res*5** denotes a sequence of 5 residual blocks as introduced by He et al. [55], **GAP** denotes a global-average pooling layer and **Dens** a fully connected layer. Spatial resolutions are decreased by strided convolution and the number of feature maps on the residual' s shortcut is increased by 1×1 convolutions. All convolutional layers have 3×3 receptive fields and are followed by batch normalization and rectified linear units. (Bottom) Topology of detector network, which is attached to one of the **AD(i)** positions. **MP** denotes max-pooling and is optional: for **AD(3)**, the second pooling layer is skipped, and for **AD(4)**, both pooling layers are skipped.

Several researches have also attempted to detect adversarial examples in the testing stage [56,57,58,59,60,61,62]. SafetyNet [56] extracted the binary threshold of each ReLU layer' s output as the features of the adversarial detector and detected adversarial images by RBF-SVM. [60] added an outlier class to the original deep learning model and the model detected the adversarial examples by classifying it as an outlier. Similarly, [62] employed probability divergence (Jensen-Shannon divergence) as one of its detectors. [61] demonstrated that after whitening by Principal Component Analysis (PCA), adversarial examples have different coefficients in low-ranked components. However, Carlini and Wagner summarised most of these adversarial detecting methods [27,57,58,59,60,61] and demonstrated that these methods can not defend against their previous attack, C&W Attack [42], with slight changes of loss function.

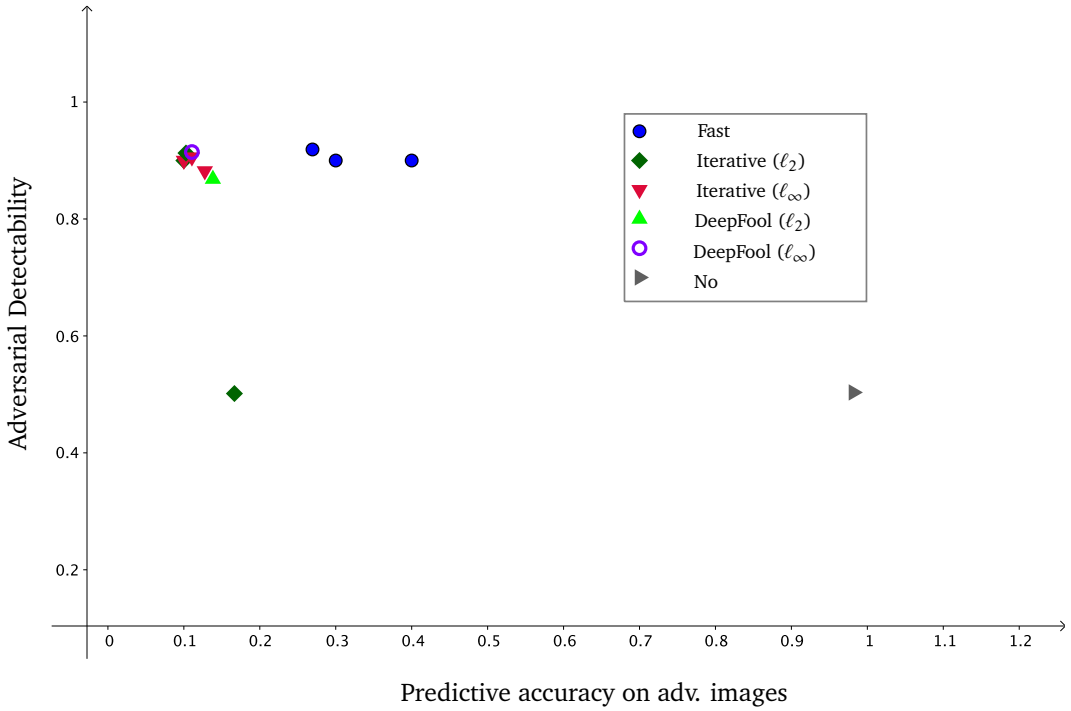


Fig. 2.8: Illustration of detectability of different adversaries and values for ϵ on 10-class ImageNet [27]. The x-axis shows the predictive accuracy of the ImageNet classifier on adversarial examples of the test data for different adversaries. The y-axis shows the corresponding detectability of the adversarial examples, with 0.5 corresponding chance level.

2.4.2 Input Reconstruction

The concept of input reconstruction aims to transform adversarial examples to clean data and to assist deep neural network in predicting correct results. Gu et al. [20] proposed a variant of denoising autoencoder network with a penalty, called deep contractive network (DCN), to increase the robustness of neural networks. They performed various experiments to assess the removability of adversarial examples by corrupting with additional noise and pre-processing with denoising autoencoders (DAEs), which is shown in Fig. 2.9. A denoising autoencoder is a feedforward neural network that learns to denoise adversarial images. By training this neural network to learn interesting features from training images, the neural network can be then employed to extract features from

similar images of the test dataset as well. They discovered that DAEs can remove substantial amounts of the adversarial noise, e.g., 9.1% error rate on ConvNetM with $\sigma = 0.1$ Gaussian noise. However, when stacking the DAE with the original DNN, the resulting network can again be attacked by new adversarial examples with even smaller distortion. As a solution, they proposed DCN, a model with a new end-to-end training procedure that includes a smoothness penalty inspired by the contractive autoencoder (CAE) [63]. This increases the network robustness to adversarial examples without a significant performance penalty. However, such a penalty is computationally expensive for calculating partial derivatives at each layer in the standard back-propagation framework. Although the simplified model respecting to each local layer was employed, this layer-wise contractive penalty objective does not guarantee global optimality for the solution and also limits the capacity of the neural network.

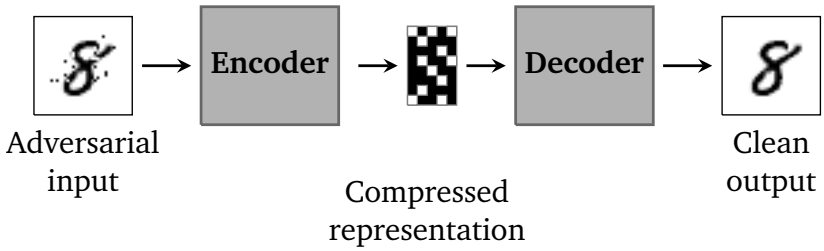


Fig. 2.9: An example of denoising autoencoder (DAE) [20]. The denoising autoencoder processes a noisy image, generating a clean image on the output side.

Further researches via denoising are introduced as follows. In [64], they showed that it's possible to remove some adversarial noise with a small mean filter. Features squeezing [65] reduced the search space available to an adversary, which makes the attack more difficult. PixelDefend [66] used PixelCNN to reconstruct the original image from its adversarial example. In [67], they studied the robustness of DNNs when the input image was preprocessed using JPEG compression, total variation minimisation, and image quilting. These simple input transformations were shown to be effective in countering adversarial examples. In [68], they proposed a defence method by randomisation at inference time, e.g., random resizing and random padding. In [69], they stochastically combined several weak denoising defences, e.g., JPEG compression and non-local means denoising. Fea-

ture denoising [70] introduced special network blocks that denoise hidden features using non-local means.

2.5 Proactive Countermeasures

This section introduces two proactive countermeasures which improve the robustness of deep neural networks before adversaries generate adversarial examples. The first one is network distillation [21] and the second one is adversarial re-training [29].

2.5.1 Network Distillation

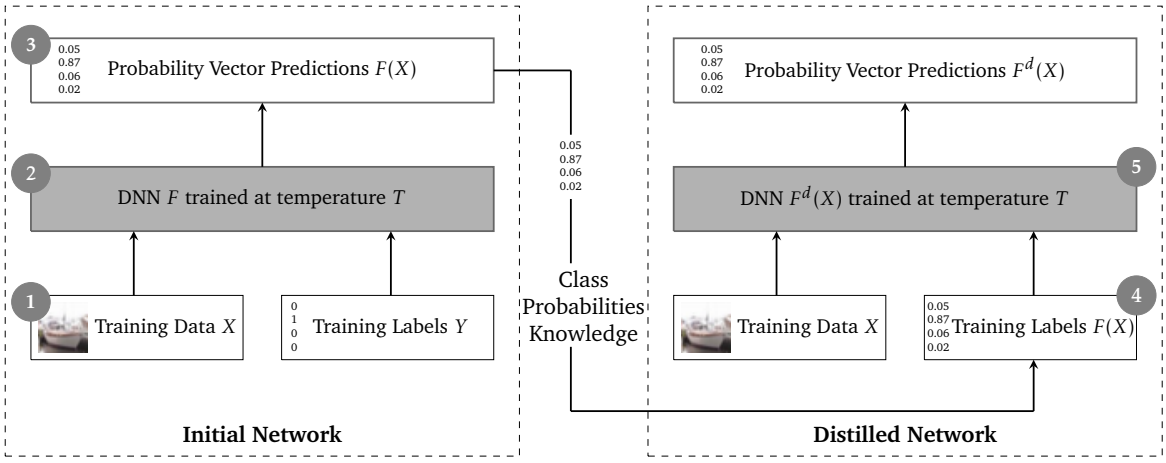


Fig. 2.10: An overview of the defence mechanism based on a transfer of knowledge contained in probability vectors through distillation [21]: The mechanism first trains an initial network F on data X with a **softmax** temperature of T. It then uses the probability vector $F(X)$, which includes additional knowledge about classes compared to a class label, predicted by network F to train a distilled network F^d at temperature T on the same data X.

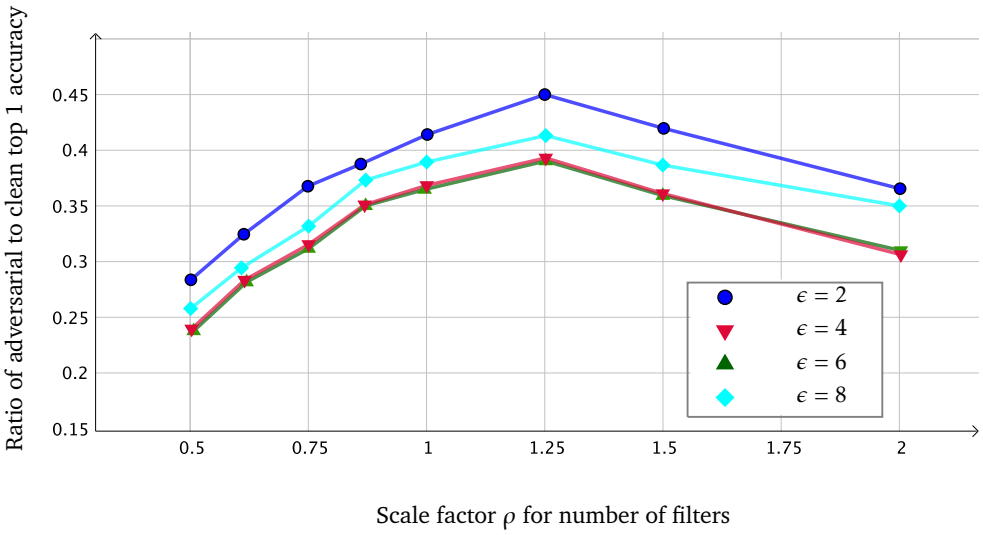
Papernot et al. [21] presented network distillation to defend deep neural networks against adversarial examples. Network distillation was originally designed to reduce the size of deep neural networks by transferring knowledge from a large network to a small one. The knowledge extracted by distillation and transferred in smaller networks can

maintain accuracies comparable with those of larger networks and can also be beneficial to improve the generalisation capability of DNNs outside of the training dataset and therefore enhance resilience to perturbations. In Fig. 2.10, the probability of classes produced by the first DNN is used as inputs to train the second DNN. The framework first trains an initial network F on data X with a softmax temperature of T. It then uses the probability vector $F(X)$, which includes additional knowledge about classes, predicted by network F to train a distilled network F^d at temperature T on the same data X. T is a temperature parameter to control the level of knowledge distillation. The schema of network distillation can be repeated several times and connects several deep neural networks. They demonstrated that attacks primarily target the sensitivity of networks and then proved that using high-temperature softmax reduces the model sensitivity to small perturbations. Network distillation improves the generalisation of neural networks and reduces the success rate of JSMA attack by 0.5% and 5% on MNIST and CIFAR10 respectively. However, the defensive distillation is unable to defend a novel method introduced by Carlini et al. [42] aforementioned. C&W attack can still acquire 100% success attack rate on DNNs trained with defensive distillation.

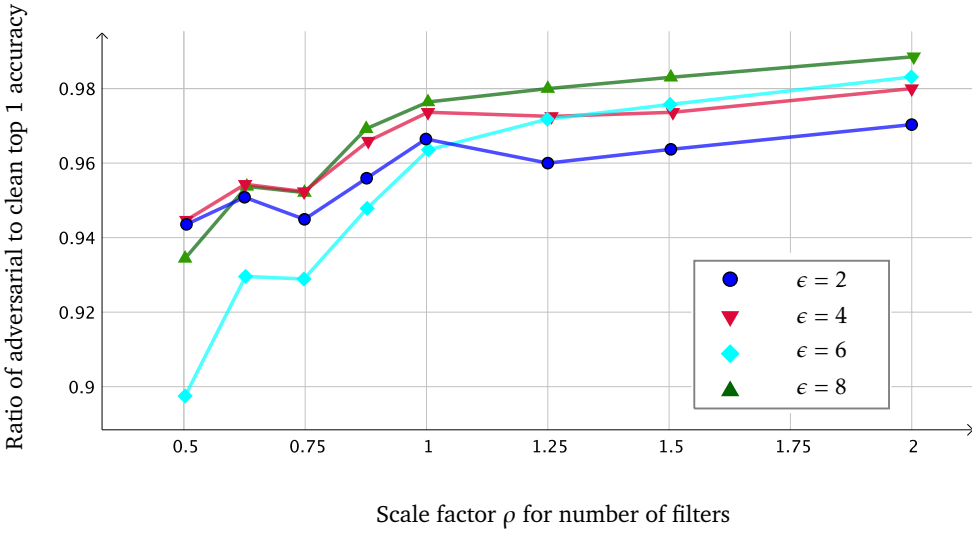
2.5.2 Adversarial Re-training

Training with adversarial examples is currently the mainstream strategy for increasing robustness against several attack strategies. Goodfellow et al. [29] first included adversarial examples which were generated with Fast Gradient Sign Method in every step of the training stage and injected them back into the training set. They demonstrated that adversarial training can improve the robustness of deep neural networks significantly since it can provide regularisation for deep neural networks and improve precisions.

Adversarial re-training with adversarial examples to make neural networks more robust is widely adopted in many works [71,72,22,73]. Although this method improves robustness, the method consumes large numbers of adversarial examples before converging into a steady state during the model training process. This is because perturbations applied on images are distributed evenly based on Fast Gradient Sign Method. This raises the key research question of learning models robust to adversarial examples in a

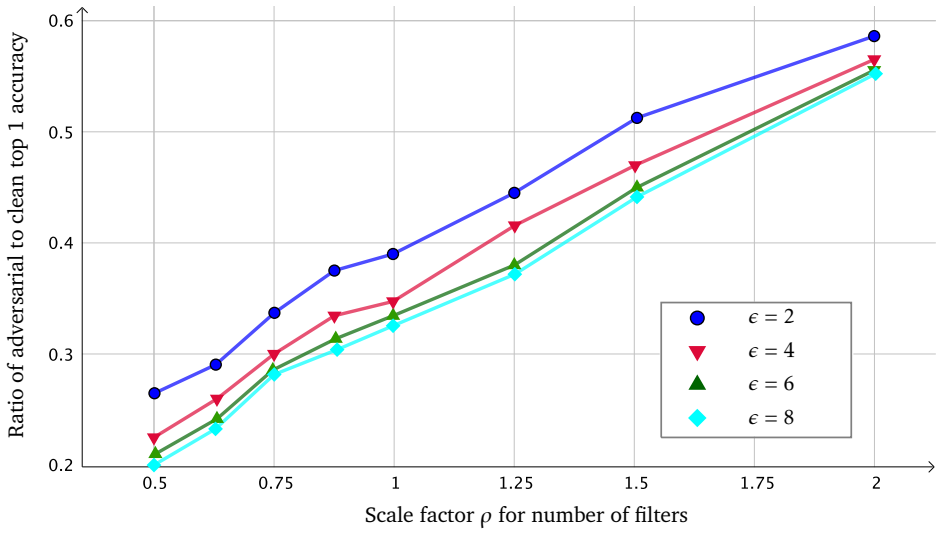


(a) No adversarial training.

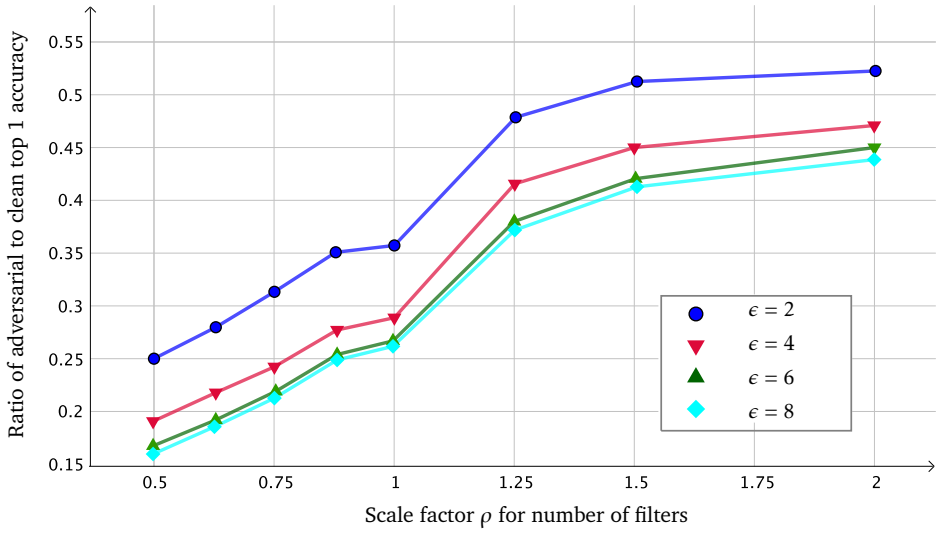


(b) With adversarial training.

Fig. 2.11: One step attack with/without adversarial re-training [28].



(a) No adversarial training.



(b) With adversarial training.

Fig. 2.12: Iterative attack with/without adversarial re-training [28].

computationally effective manner. The algorithm is demonstrated in Algorithm 1.

Algorithm 1: Adversarial training of network N.

Size of the training minibatch is m.

Number of adversarial images in the minibatch is k

function AdversarilTraining (N);

Randomly initialize network N

repeat

  Read minibatch $B = \{X^1, ..., X^m\}$ from training set;

  Generate k adversarial examples $\{X^1_{adv}, ..., X^k_{adv}\}$ from corresponding clean examples $\{X^1, ..., X^k\}$ using current state of the network N;

  Make new minibatch $B' = \{X^1_{adv}, ..., X^k_{adv}, X^{k+1}, ..., X^{k+1+m}\}$;

  Do one training step of network N using minibatch B'

until *training converged*;

In this algorithm, the initial setting for a neural network is random uniform distribution. They started a training loop by choosing a minibatch set B of size m from training set images and then generated adversarial examples of size k from corresponding clean images $\{X^1, ..., X^k\}$ using the current state of network N with gradient sign method. The new minibatch B' is composed of the original minibatch B and the adversarial examples of size k. One training step of network N is trained with the new minibatch B' and the previous steps are repeated until the training loss is converged.

In [29], they only evaluated the MNIST dataset and comprehensive analysis of adversarial training methods on the ImageNet dataset was presented in [28]. They employed half adversarial examples and half origin examples in each step of training stage. From their results, adversarial training increases the robustness of neural networks for one-step attack (e.g., FGSM) and is also adopted for regularisation to avoid overfitting. In Fig. 2.11, the left column is without adversarial training while the right one is adversarial training with different magnitude of perturbations. Although the accuracy increases dramatic for the one-step attack, it does not improve accuracy under basic iterative attacks (BIM) in Fig. 2.12. Much larger models may be necessary or more effective adversarial examples (e.g., C&W attack) for adversarial training are required to achieve better robustness.

2.6 Existing Limitations

We have reviewed some state-of-the-art defence methods so far in Section 2.4 and 2.5. In the first, we placed those commonly referred to as "reactive countermeasures" (such as adversarial detecting [27] and input reconstruction [20]). In the second, we listed those involving proactive countermeasures (such as network distillation [21] and adversarial retraining [28]). Both countermeasures improve the robustness of image classifiers, but they still have limitations.

Metzen et al. [27] created a small detector subnetwork for adversarial examples as an auxiliary network of the original neural network. The detector is a small neural network which was trained on the binary classification task to distinguish genuine data from data containing adversarial perturbations, i.e. the probability of the input being adversarial. Approaches have been put forward to detect adversarial examples at testing stage [56,57,58,59,60,61,62]. SafetyNet [56] extracted the binary threshold of each ReLU layer' s output as the features of the adversarial detector and detected adversarial images by RBF-SVM. In [60], they added an outlier class to the original deep learning model; the resulting model detects the adversarial examples by classifying them as outliers. Similarly, in [62] they employed Jensen-Shannon probability divergence to construct detectors. In [61], they demonstrated that after whitening by Principal Component Analysis (PCA), adversarial examples have different coefficients in low-ranked components. However, Carlini et al. demonstrated that many adversarial detecting methods, including [27,57,58,59,60,61], cannot defend against the C&W attack [42].

Input reconstruction approaches aim to transform adversarial examples to clean data and then applying these to assist neural networks to predict correct results with denoising autoencoders (DAEs) [20]. The DAEs aim to learn a representation for a set of data and to reduce dimensionality by training the networks to ignore signal noise. Input reconstruction has been used to alter adversarial examples to assist deep neural network in predicting correct classifications. Gu et al. [20] proposed a variant of denoising autoencoder network with a penalty, called deep contractive network (DCN), to increase the robustness of neural networks. This increases the network robustness to adversarial examples without

a significant performance penalty. However, calculating partial derivatives at each layer in the back-propagation framework becomes computationally expensive. Also, the proposed layered based approach does not guarantee global optimality.

Papernot et al. [21] presented a method for network distillation to defend deep neural networks against adversarial examples. Network distillation was originally designed to reduce the size of deep neural networks by transferring knowledge from a large network to a small one. The intuition is that knowledge extracted by distillation and transferred in smaller networks can also be beneficial to improve the generalisation capability of DNNs outside the training dataset and therefore enhance resilience to perturbations. They showed that network distillation can improve the generalisation of neural networks and reduce the success rate of JSMA attack by 0.5% and 5% on MNIST and CIFAR10 respectively [41]. However, network distillation was unable to defend against an attack by developed in [42].

Adversarial training [29] concerns explicitly training a model on adversarial examples, in order to make it more robust against attacks or to reduce its test error on clean inputs. Training with adversarial examples is one of the most powerful countermeasures to make networks more robust. It enhances robustness by generating adversarial examples at every step of the training stage and injecting them into the training set. In this way, adversarial training provides regularisation for DNNs and improves precisions through the process. A disadvantage of adversarial training is that it takes more training efforts compared with other methods. We have shown that the method here presented is more lightweight compared to adversarial training while maintaining similar performance.

Some state-of-the-art defence methods have been introduced in the previous sections. However, these methods can not provide comprehensive defences under different attack methods for some reasons. In adversarial detecting strategy, the failure case emphasises that the detector has to detect very subtle patterns and the optimiser may get stuck in bad local optima or plateaus. In input reconstruction strategy, although the simplified model with respect to each local layer is employed, the layer-wise contractive penalty objective does not guarantee global optimality for the solution and also limits the capacity of the neural network. In network distillation strategy, though they demonstrate that attacks

primarily target the sensitivity of networks and then prove that using high-temperature softmax reduces the model sensitivity to small perturbations. However, Carlini et al. [42] introduce a novel method for constructing adversarial examples that breaks the defensive distillation. In adversarial re-training strategy, although the accuracy increases dramatic for one-step attack, it would not help under basic iterative attacks aforementioned. Even though they adopt iterative adversarial examples during training, few benefits are obtained from it. In the meanwhile, it is computationally costly and they are unable to prevent the procedure from reducing the accuracy on clean examples significantly. Much larger models may be necessary or more effective adversarial examples (e.g. C&W attack) for adversarial training are required to achieve better robustness. This also implies that adversarial training with more powerful adversarial examples increases the robustness of deep neural networks indeed. However, this comes at a high cost as the resistance to attacks increases only after large numbers of adversarial examples have been added to the training data. This raises the key research question of learning models that are robust to adversarial examples in a computationally effective manner.

Chapter 3

An MCTS-based Method for Robustness

To enhance the robustness against adversarial examples in a computationally effective manner while maintaining accuracies, we present an adversarial training algorithm based on Monte-Carlo tree search (MCTS). We illustrate the robustness of the algorithm by studying its resistance to adversarial examples in the context of the MNIST [74] and CIFAR10 [75] datasets. In this research, we develop a new adversarial training method based on MCTS-based adversarial training (MAT). We put forward the architecture of the training method and present the experimental results obtained. We show that MAT not only can provide effective adversarial examples for adversarial training, but also robustness against different attack methods, e.g., FGSM, PGD and C&W. To generate adversarial examples, we use an MCTS-based attack method. This enables us to reduce significantly the number of adversarial examples required for adversarial training, which leads us to a computationally attractive approach to generating robust models.

We start with a problem formulation in Section 3.1. Section 3.2 describes how to generate adversarial examples with an MCTS-based attack method. Section 3.3 explains the MCTS-based adversarial training algorithm and how that leads to an improvement of the robustness of image recognition classifiers against different attacks. Section 3.4 reports experimental results obtained on the MNIST and CIFAR-10 datasets; section 3.5

summarises the chapter and compares the results with some related works.

The material presented in this chapter has been heavily adjusted to fit the rest of this thesis, but is loosely based on research first presented in the following paper [2]:

- Y. Liu and A. Lomuscio. An MCTS-based Adversarial Training Method for Image Recognition. Proceedings of the 32nd International Joint Conference on Neural Networks (IJCNN). Budapest, Hungary. 1-8(2019). IEEE Press.

3.1 Problem Formulation & Notation

Before introducing the MAT method, we first recall the definition for adversarial examples. Given a trained neural network $f(\cdot) \in F : \mathbb{R}^D \to y$, an original input data sample $x \in \mathbb{R}^D$ and a label in K classes $y \in \{1, ..., K\}$, generating an adversarial example $x' \in \mathbb{R}^D$ can be defined as a box-constrained optimisation problem, which optimising the cost function with respect to x constrained in a range:

$$\begin{aligned} \min_{x'} \quad & \|x' - x\|_p \\ s.t. \quad & f(x') = y', \\ & f(x) = y, \\ & y \neq y', \\ & x, x' \in [0, 1], \end{aligned} \tag{3.1}$$

where y and y' is the output label of x and x', $\|\cdot\|_p$ denotes the distance between two data samples under some norm p. Let $\eta = x' - x$ be the perturbations applied on x. This optimisation problem aims to minimise the perturbation while misclassifying the prediction with a constraint of input data. Many variants of this optimisation problem have been proposed focusing on different scenarios and assumptions. For instance, some adversaries attempt to constrain the perturbation in a subtle range that is unnoticeable to humans. The optimisation objective function develops to the distance of targeted

prediction score from the original prediction score. This can be formulated by:

$$\begin{aligned} \min_{x'} \quad & \mathcal{L}(f(x'), \theta, y) \\ s.t. \quad & \|x' - x\|_p \leq \epsilon, \\ & f(x') = y', \\ & f(x) = y, \\ & y \neq y', \\ & x, x' \in [0, 1], \end{aligned} \tag{3.2}$$

where $\mathcal{L}(f(x'), \theta, y)$ is the cost function for the distance of targeted prediction score from the original prediction score and ϵ denotes the measurement of the perturbation constraint.

3.2 MCTS-based Attack Method

In this section, we present an MCTS-based attack method with the assistance of scale invariant feature transform (SIFT) algorithm and how this method can be employed to generate effective adversarial examples. In subsection 3.2.1, we introduce SIFT algorithm and how it can work with MCTS algorithm in subsection 3.2.2. Subsection 3.2.3 explains how to generate effective adversarial examples under the perturbation constraint.

3.2.1 Scale Invariant Feature Transform (SIFT)

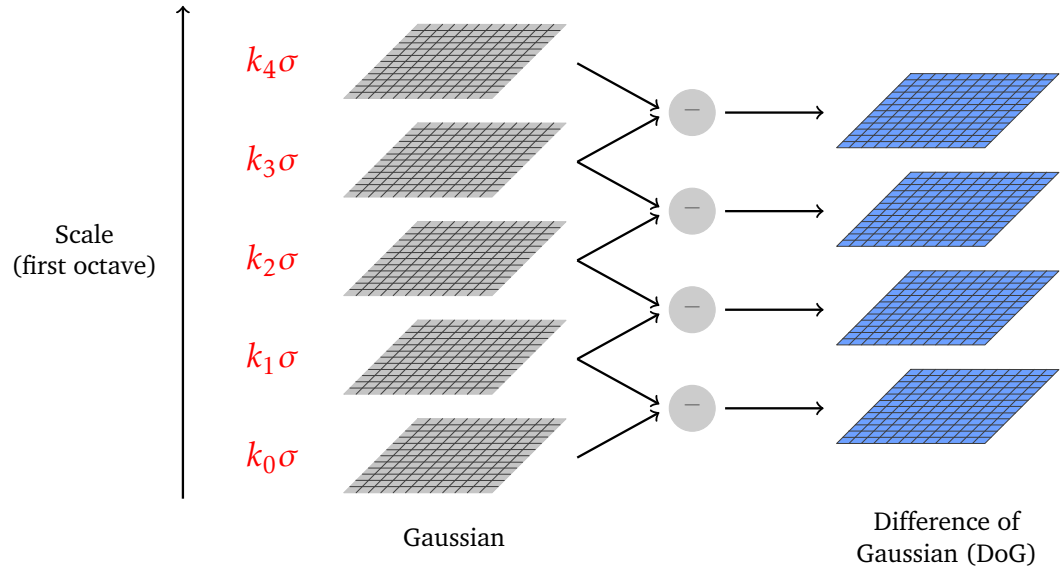
SIFT algorithm can be employed to extract features for any object in an image and provide a feature description of this object. This description can then be used to identify and locate objects in other images. In order to perform accurate recognition, these extracted features should be detectable under variations of image scale, orientation and illumination. These features are normally associate with high-contrast regions of images, such as object edges. The SIFT algorithm was first proposed to extract image features in object recognition systems [76] that these features are invariant to image scaling, translation and rotation. Through the invariant characteristics to image transformation of SIFT, adding perturbations on the locations of these invariant features on images are more likely

to induce an adversarial example. This concept is taken into this research and evaluated on different constraints with MNIST and CIFAR10 datasets. This SIFT algorithm will extracts keypoints and then computes its descriptors which involves four stages that are scale-space extrema detection, keypoint localisation, orientation assignment and keypoint descriptor. We explain more details about SIFT in the following subsections (A to D).

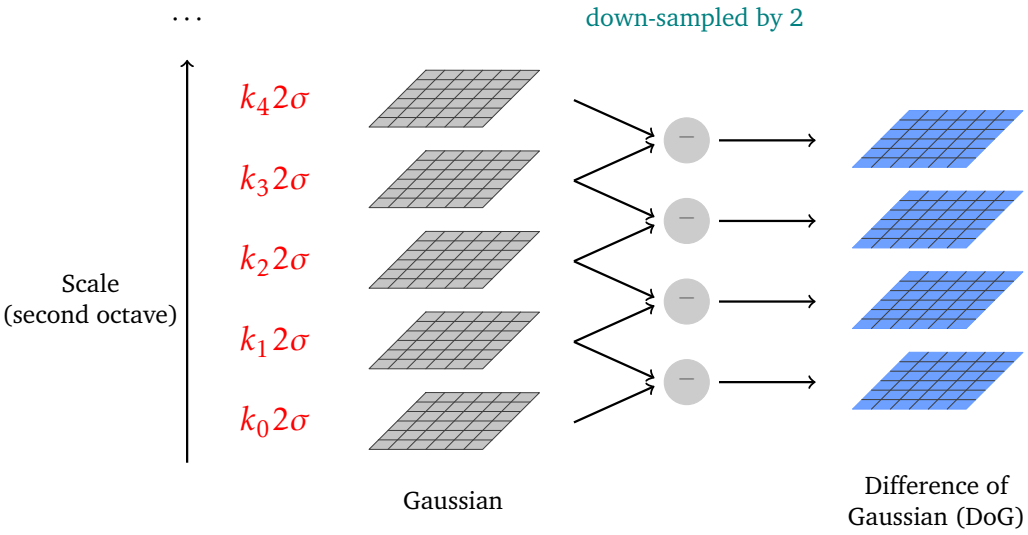
A. Scale-space Extrema Detection

To search for these features, Laplacian Filters are normally used to detect areas of rapid changes (edges) in images. Since derivative filters are very sensitive to noise, it is common to smoothen the image (e.g., using Gaussian-blurred images) before applying the Laplacian operation. This two-step process is called Laplacian of Gaussian (LoG) operation. For the reason of the significant computational cost of LoG operation, the Difference of Gaussians (DoG) is employed for simplification in Fig. 3.1. In SIFT, key locations are defined as maxima of Difference of Gaussian (DoG) that occur at different scales and low-contrast candidate points and edge response points along an edge are then discarded. To maintain invariance to image scaling, this process is implemented under different octaves with different scaling value $n\sigma$ of the image in a Gaussian pyramid structure. This is first convolved with Gaussian-blurs at different scales of $k_i\sigma$. The convolved images are then grouped by octaves (an octave corresponds to a doubling value of σ), and the value of k_i is selected so that a fixed number of convolved images per octave are obtained. The Difference-of-Gaussian images are then obtained from adjacent Gaussian-blurred images per octave. In Fig. 3.1a, the first octave with scale value σ and different k_i is first calculated with Gaussian-blurs and then computes the Difference of Gaussian between each image. In Fig. 3.1b, the second octave with down-sampled size by 2 is calculated using the same scheme and so on.

Once DoG images have been obtained, local extrema of the DoG images over different scales are then identified. This is implemented by comparing each pixel in the DoG images to its eight neighbours at the same scale and nine corresponding neighbouring pixels in each of the neighbouring scales. If a local extremum is identified, this is considered a potential invariant feature. A feature description containing different invariant feature candidates will be obtained in the last step of SIFT algorithm.



(a) The first octave of Gaussian pyramid.



(b) Second octave of Gaussian pyramid and following octaves in smaller scales.

Fig. 3.1: The Gaussian pyramid of one image.

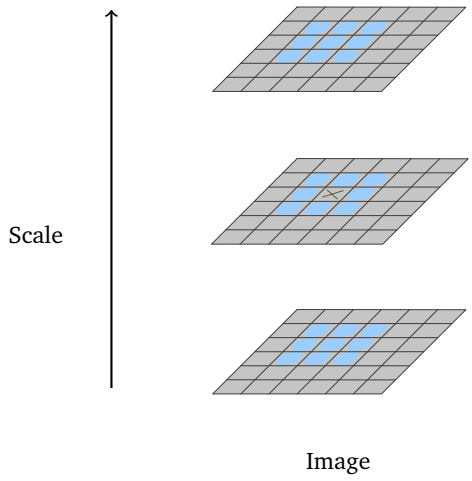


Fig. 3.2: The comparison process of scale-space extrema detection.

B. Keypoint Localisation

As some of the local extrema produced from the scale-space extrema detection stage are unstable, they have to be refined to obtain more accurate keypoints. The keypoint localisation stage is to perform a detailed fit for a more accurate location of extrema. For each potential keypoint, interpolation of nearby points is employed to accurately determine its position. The interpolation is implemented with quadratic Taylor expansion of the DoG scale-space function using the potential keypoint as the origin to compute offset. If the offset from the potential keypoint is less than 0.03, this potential keypoint, which is a low-contrast keypoint, is then discarded [77]. DoG function also has strong responses to edges, so high edge responses are necessary to be eliminated for stability. This is implemented with a second-order Hessian matrix to solve the principal curvature and the eigenvalues of the Hessian matrix are proportional to the principal curvatures. This can avoid explicitly computing the eigenvalues because this is only concerned with the ratio. If the ratio of two eigenvalues of the Hessian matrix exceeds the threshold, which is set as 10 in the paper [77], the keypoint is discarded. Fig. 3.3a demonstrates the potential keypoints from Scale-space Extrema Detection, Fig. 3.3b shows remaining keypoints discarding low-contrast keypoints, and Fig. 3.3c is the result eliminating high edge responses.

(a) Scale-space Extrema. (b) Discarding Low Contrast. (c) Filtering Edge Responses.

Fig. 3.3: Different stages of keypoint localisation

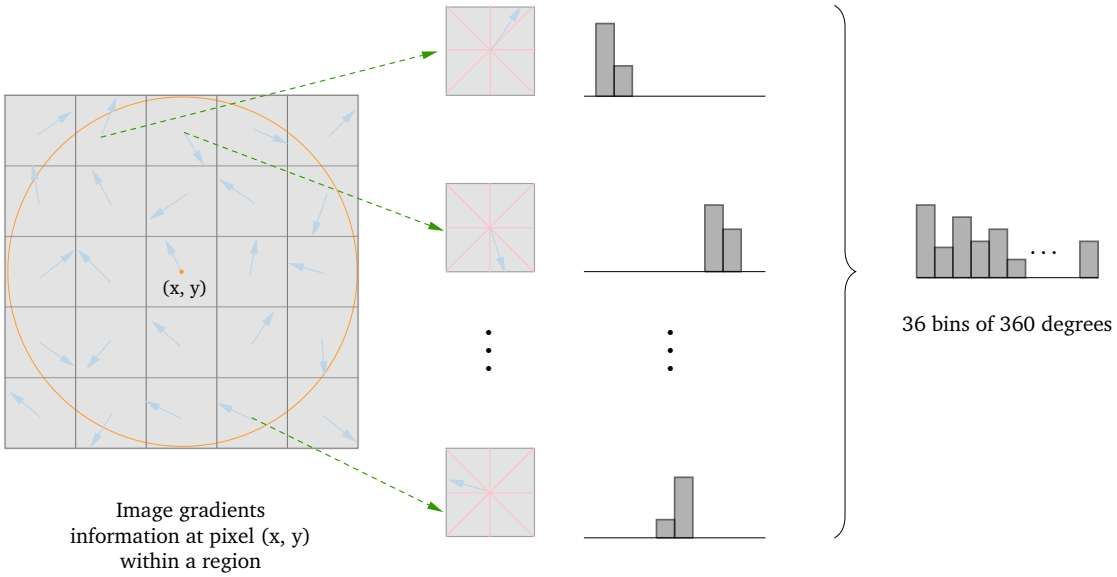


Fig. 3.4: The orientation assignment for some keypoint centred at (x, y) within a region. The orientation histogram has 36 bins covering the 360-degree range of orientations. Increasing the bin numbers will increase the computation efforts.

C. Orientation Assignment

In this stage, an orientation is assigned to each keypoint (x, y) to achieve invariance to image rotation. At each pixel A_{ij}, the image gradient magnitude M_{ij} and orientation

R_{ij} are computed using pixel differences shown in Equation (3.3) and (3.4). An orientation histogram is formed from the gradient orientations of sample points within a region around the keypoint. The orientation histogram has 36 bins covering the 360-degree range of orientations. Each sample added to the histogram is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ that is 1.5 times that of the scale of the keypoint. The highest peak value of the 36 bins in the histogram is taken and any peak value above 80% of it is also considered as a reference orientation. Therefore, for locations with multiple peaks of similar magnitudes, there will be multiple keypoints created at the same location and scale but with different orientations. Only about 15% of points are assigned multiple orientations, but these contribute significantly to the stability of matching. The detailed process of orientation assignment for some keypoint (x, y) is shown in Fig. 3.4.

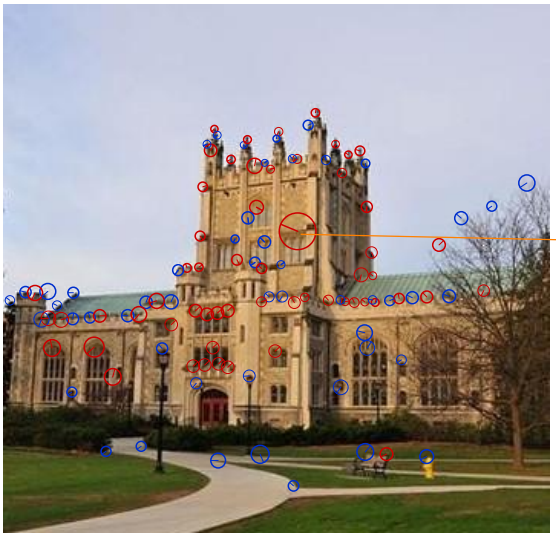
$$M_{ij} = \sqrt{(A_{ij} - A_{i+1,j})^2 + (A_{ij} - A_{i,j+1})^2}, \tag{3.3}$$

$$R_{ij} = tan^{-1} \frac{A_{ij} - A_{i+1,j}}{A_{i,j+1} - A_{ij}}. \tag{3.4}$$

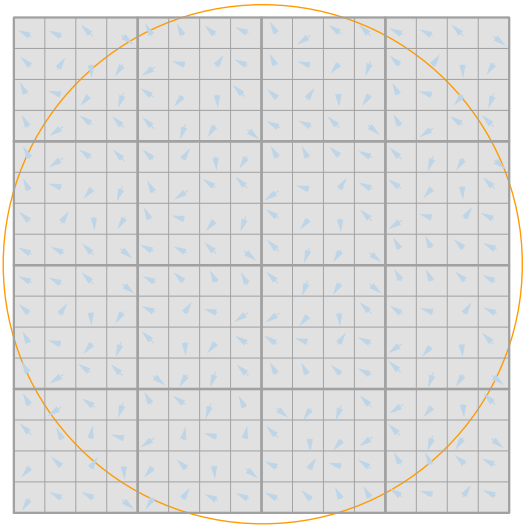
D. Keypoint Descriptor

Previous steps found keypoint locations at particular scales and assigned orientations to them. This ensured invariance to the image location, scale and rotation. This subsection is to compute a descriptor vector for each keypoint such that the descriptor is highly distinctive and partially invariant to the remaining variations such as illumination, 3D viewpoint, etc. This step is performed on the image closest in scale to the keypoint's scale.

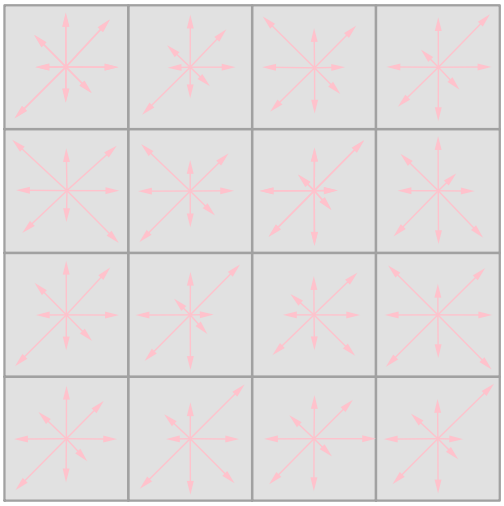
In Fig. 3.5, we explain the process of keypoint descriptor assignment. In Fig. 3.5(a), it shows all the keypoints of some image. First, the image gradient magnitudes and orientations are sampled around the keypoint location, using the scale of the keypoint to select the level of Gaussian blur for the image. In order to achieve orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation. For efficiency, the gradients are precomputed for all levels of the pyramid as described previously. These are illustrated with small arrows at each sample



(a) Keypoints of an image



(b) Image gradients of some keypoint centred in (x, y) within a region



(c) The keypoint descriptor

Fig. 3.5: The process of keypoint descriptor assignment. A keypoint descriptor is created by first computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location, as shown in figure (b). These are weighted by a Gaussian window, indicated by the overlaid circle. These samples are then accumulated into orientation histograms summarising the contents over 4x4 subregions, as shown in figure (c), with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This figure shows a 4×4 descriptor array computed from a 16×16 set of samples.

location in Fig. 3.5(b). A Gaussian weighting function with σ equal to one half the width of the descriptor window is used to assign a weight to the magnitude of each sample point. This is illustrated with a circular window. The purpose of this Gaussian window is to avoid sudden changes in the descriptor with small changes in the position of the window and to give less emphasis to gradients that are far from the centre of the descriptor, as these are most affected by misregistration errors. The histograms are computed in 16×16 regions around the keypoint such that each histogram contains samples from 4×4 subregions of the original neighbourhood region. Finally, these samples are then accumulated into orientation histograms summarising the contents over 4×4 subregions, as shown in Fig. 3.5(c), with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. Since there are $4 \times 4 = 16$ histograms each with 8 bins, the vector has 128 elements. As the best results are achieved with a 4×4 array of histograms with 8 orientation bins in each, the experiments in the paper use a $4 \times 4 \times 8 = 128$ element feature vector for each keypoint. A change in image contrast in which each pixel value is multiplied by a constant will multiply gradients by the same constant, so this contrast change will be canceled by vector normalisation. For this reason, this vector is then normalised to unit length to enhance invariance to affine changes in illumination [77].

3.2.2 Monte Carlo Tree Search (MCTS)

In this subsection, we explain how the Monte Carlo Tree Search [1] algorithm can be employed to assist in making decisions about which candidates should be applied perturbations once the description of invariant feature candidates is obtained from SIFT. Each node in the game tree stands for a candidate from SIFT. MCTS is a heuristic search algorithm for the decision process which is widely adopted in games. The key feature of MCTS is the search of promising candidate moves on the basis of random sampling in search space. Each round of MCTS consists of four steps, that are selection, expansion, simulation and backpropagation, which is illustrated in Fig. 3.6.

In the selection step, a root and successive child nodes are selected until a leaf node is reached according to each confidence value on the node, which is defined in Equation (3.5).

The root is the current game state and a leaf is any node that has a potential child from which no simulation (playout) has yet been initiated. The selection is more about a way of biasing choice of child nodes that letting the game tree expand towards the most promising moves. In the expansion step, unless the leaf node ends the game decisively (e.g., win/loss/draw) for either player, create one (or more) child nodes from the leaf node and choose one node from them. In the simulation step, one random playout from the created node is completed. This step is sometimes also called playout or rollout. A playout may be as simple as choosing uniform random moves until the game is decided, for example in chess, the game is won, lost, or drawn. In the last backpropagation step, the result of the playout is used to update information in the nodes on the path from the newly created node to the root node.

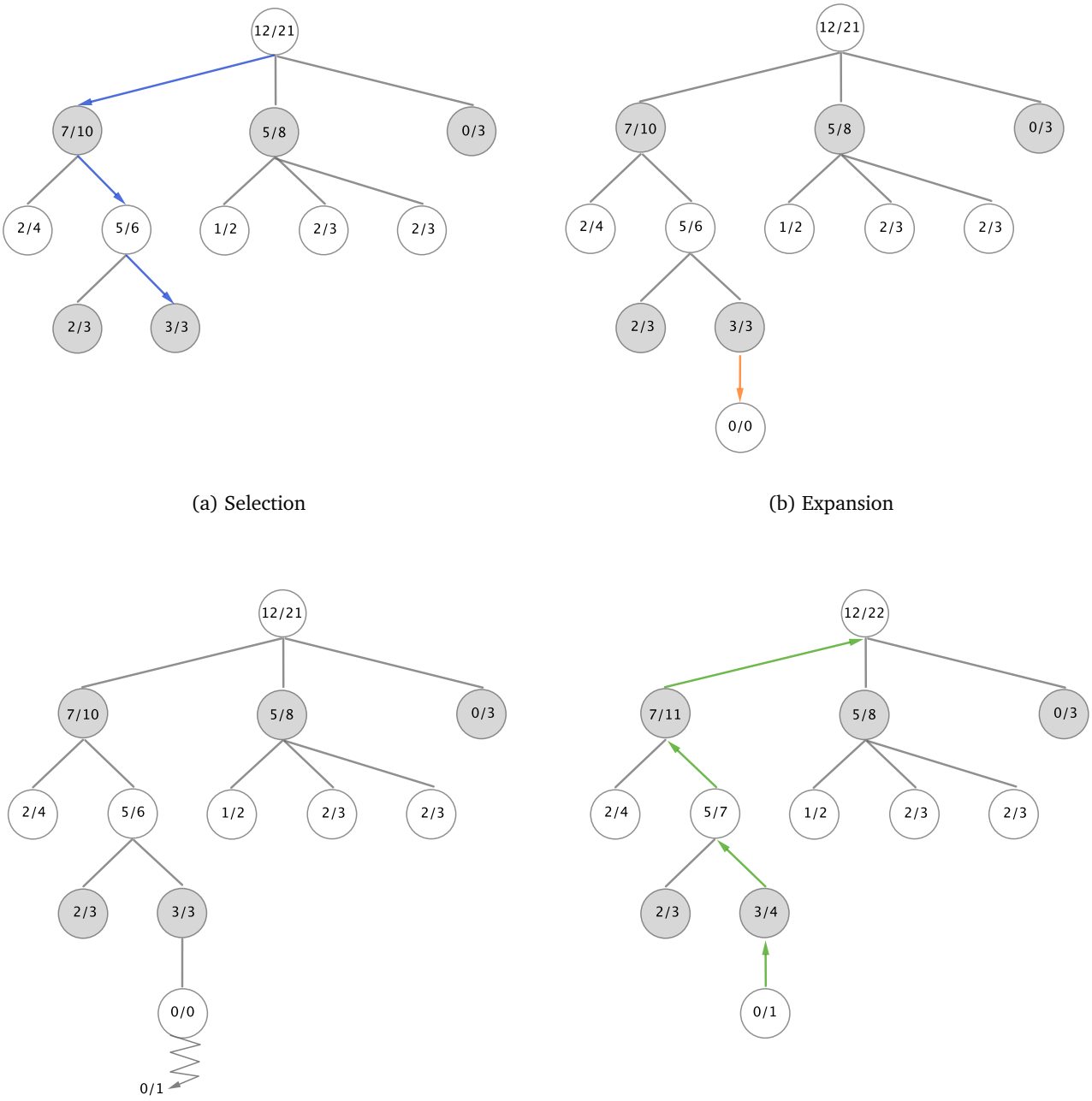
As for the confidence value, according to MCTS, it is recommended to choose the move in each node of the game tree for which the expression in Equation (3.5) has the highest confidence value:

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}. \tag{3.5}$$

where w_i stands for the number of wins for the node considered after the i-th move, n_i stands for the number of simulations for the node considered after the i-th move, N_i stands for the total number of simulations after the i-th move and c is the exploration parameter, theoretically equal to $\sqrt{2}$. The higher the confidence value means the more contributions to search adversarial examples. Conversely, the lower probability to obtain an adversarial example the lower is the confidence value. This confidence measure above is inspired by neural approaches to game playing [78].

Instead, we employ this concept to search adversarial examples from the keypoints obtained from SIFT. In our implementation, w_i stands for the inversion of Euclidean distances for the node considered after the i-th move, n_i stands for the number of visits for the node considered after the i-th move, and N_i stands for the total number of visits after the i-th move.

In the following subsections, we explain how the MCTS algorithm is employed to assist in making decisions about which candidates should be applied perturbations once the description of invariant feature candidates is obtained from SIFT for details.



(a) Selection

(b) Expansion

(c) Simulation

(d) Backpropagation

Fig. 3.6: The process of Monte Carlo Tree Search. Each round of MCTS consists of four steps, that are selection, expansion, simulation and backpropagation. The value in each node stands for the confidence value.

A. Selection

In the selection step, we start from root node R and select successive child nodes down to a leaf node L according to the confidence value on each node as shown in the blue path in Fig. 3.6(a). The step is aiming to choose child nodes that allow the game tree to expand towards the most promising moves, which means the most potential keypoints to apply perturbations for an adversarial example.

B. Expansion

The expansion step creates one or more child nodes and chooses node C from one of them unless L terminates the game with a win/loss result. The win stands for an adversarial example is obtained and loss is the other way around. In Fig. 3.6(b), one node is created from the leaf node L with the orange path.

C. Simulation

We then simulate the game from the chosen node C in the simulation step in Fig. 3.6(c). This step simulates the results if an adversarial example is obtained once a perturbation is applied to the point. 0 stands for failure to an adversarial example and 1 stands for success to get one adversarial example.

D. Backpropagation

The last backpropagation step updates relative information according to the simulation results along the path from node C back to the root node R in a backtracking direction. In Fig. 3.6(d), the failure result is propagated back along the whole path by adding one to each denominator shown in the green path.

Through SIFT method, we obtain the potential keypoints for adversarial examples and via the heuristic characteristics of MCTS, it helps to search the effective adversarial examples according to the confidence values acquired in each simulation iteration. The higher the confidence value means the more contributions to search adversarial examples while perturbed.

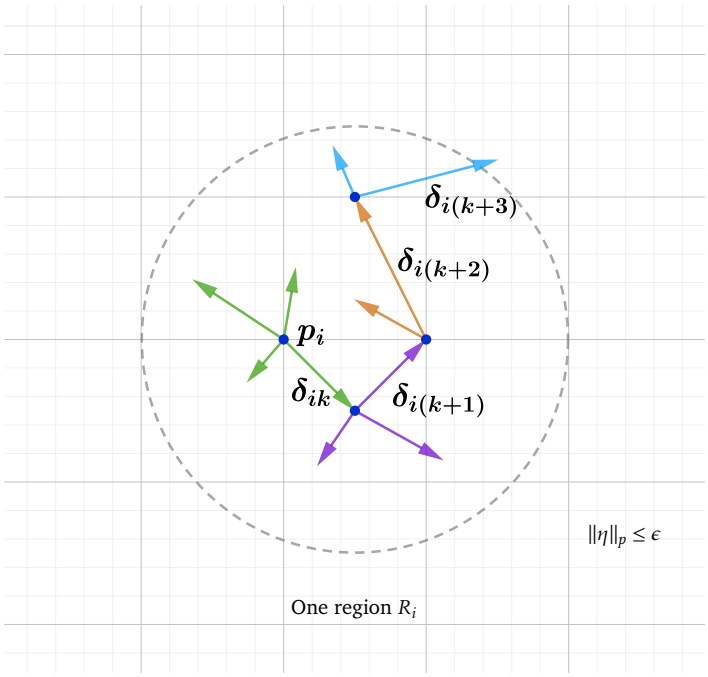


Fig. 3.7: Some game tree starting from point p_i with perturbations δ_{ik} along the path in a region R_i under the constraint of $\| \eta \|_p \leq \epsilon$.

3.2.3 Effective Adversarial Examples

In this subsection, we define an effective adversarial example under the constraint with $\|\eta\|_p \leq \epsilon$ using the MCTS-based attack method. Given an image, it is divided into R regions. For each region R_i in an image, each path of the game tree is illustrated with a ladder-like structure in Fig. 3.7.

Each region R_i contains several nodes p_k of the game tree and a specific perturbation δ_k is applied after one node p_k is selected. The total perturbations applied in a region is then presented as $\sum_{k=0}^{K} \delta_k$, where K is the total number of nodes along a path. An effective adversarial example $\mathcal{A}$ with $||\eta||_p \leq \epsilon$ is formulated as Equation (3.6):

$$\mathcal{A} = \| \sum_{i=0}^{R} \sum_{k=0}^{K} \delta_{ik} \|_p \leq \epsilon, \tag{3.6}$$

where R is the number of regions in an image. Note that the total perturbations applied in a region are possibly zero if there are no differences of the region between an adversarial example and the original image.

3.3 MCTS-based Adversarial Training

We start by introducing the MCTS-based adversarial training framework in subsection 3.3.1. The overall algorithm, including a robust optimisation function, is presented in subsection 3.3.2.

3.3.1 MCTS-based Adversarial Training Framework

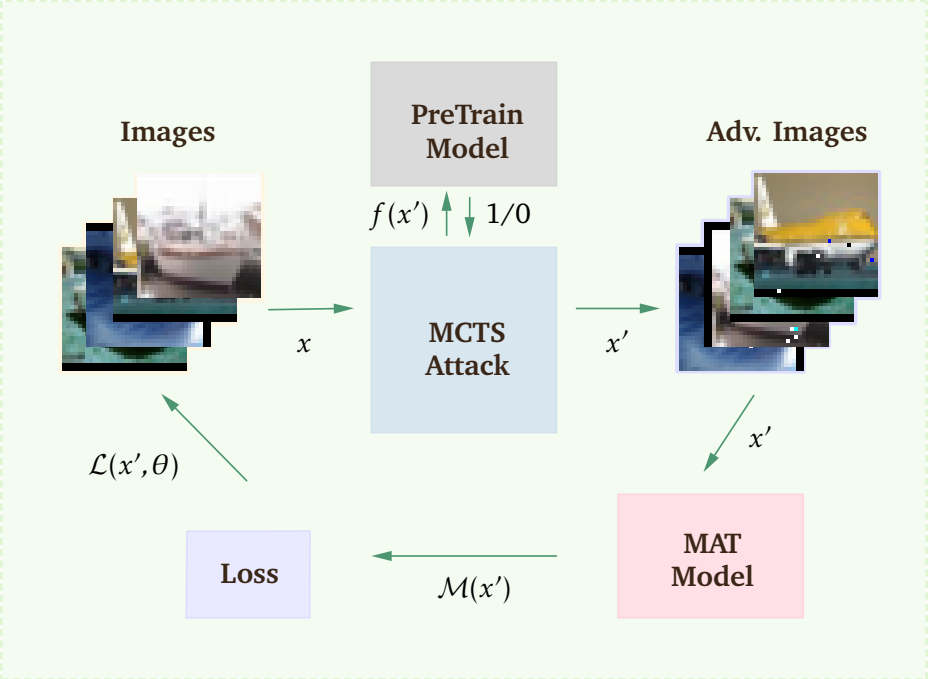


Fig. 3.8: MCTS-based Adversarial Training Framework.

We propose the training framework illustrated in Fig. 3.8. This is divided into two steps: the first consists of an attack generation step; the second is an adversarial training step. For the attack generation part, first, a model $f(\cdot)$ is trained as an assistant classifier from a training dataset of images x. Then, the procedure follows a training loop. In each training loop, corresponding to a training epoch, a minibatch of size m of images is randomly selected from the training dataset x. This minibatch is then analysed by the MCTS attack block to generate adversarial examples x'. Whether or not the images

generated by MCTS attack block are proper adversarial examples is determined by a test on the assistant classifier $f(\cdot)$. The resulting adversarial examples x' form the basis for training the adversarial training model $\mathcal{M}(\cdot)$, which is initially untrained. The loss value $\mathcal{L}$ is updated according to the softmax result of adversarial training model $\mathcal{M}$. The training loop continues for as many epochs as required until the required accuracy is converged.

Algorithm 2: MCTS-based Adversarial Training

Deep neural network $\mathcal{M}$

Size of the training minibatch is m

function MCTS-based_Adversarial_Training ($\mathcal{M}$);
Setup Pre_Training Model $f(\cdot)$
Randomly initialize neural network $\mathcal{M}$
while *training not converged* **do**
    Read minibatch $B = \{x_1, ..., x_m\}$ from training set;
    Generate m adversarial examples $B_{adv} = \{x'_1, ..., x'_m\}$ from corresponding clean examples $\{x_1, ..., x_m\}$ with MCTS attack method (Section 3.2);
    Do one training step of network $\mathcal{M}$ using minibatch B';
    Update model loss with min_max optimisation:
    $\min_\theta \mathcal{L} = \min_\theta \sum_{i=1}^{m} \max_\eta \mathcal{L}(x'_i, \theta, y_i)$;
end

3.3.2 MCTS-based Adversarial Training Algorithm

We now introduce the MCTS-based adversarial training (MAT) algorithm in Algorithm 2, including the robust optimisation function applied in MAT model $\mathcal{M}$. Following the framework mentioned above, the MAT algorithm randomly selects a minibatch in each epoch until the loss of the MAT model converges to the desired value. The convergence criteria ensure that the resulting model M is robust in small neighbourhoods of every training point around x. We call these neighbourhoods the perturbations η and we represent them as $x' = x + \eta$. The overall process can be regarded as a solution to the robust optimisation problem against adversarial examples formulated as:

$$\min_{\theta} \mathcal{L} = \min_{\theta} \sum_{i=1}^{m} \max_{\eta \leq \|\epsilon\|} \mathcal{L}(x_i', \theta, y_i), \tag{3.7}$$

where η is the uncertainty set under the constraint ϵ corresponding to the adversarial example x_i'. This involves optimising the model parameter θ with respect to a worst-case data (x_i', y_i), rather than against the original training data; the i-th worst-case data point is selected from the uncertainty set η. These uncertainty sets are determined by the problem at hand; adversarial training [17] can be understood as one such problem.

3.4 Experimental Results

In this section, the experimental results obtained from the MCTS-based adversarial training algorithm presented in the previous section are reported. To evaluate robustness, we consider a network to be reliable against adversarial attacks if the predictions of the network are accurate under different attack methods (FGSM or C&W) with different amounts of perturbations. As previously discussed, the FGSM is extremely fast in practice and is widely adopted to generate attacks. The C&W is regarded as one of the most powerful attack methods, so is considered in our experiments. The PGD method is believed to be the state-of-the-art attack method and is largely applied to develop a robust model. For these reasons, we compare our results with PGD under the attacks of FGSM and C&W.

We use two datasets: MNIST [74] and CIFAR-10 [75]. The resulting accuracy, considering different attacks, is defined in subsection 3.4.1 and the experimental setup is reported in subsection 3.4.2. The experimental results for our method with adversarial accuracies are shown in subsections 3.4.3 and 3.4.4.

3.4.1 Adversarial Accuracy

In this paper, we consider a network to be reliable against adversarial attacks if the predictions of the network are accurate under different attack methods (FGSM or C&W) and different amounts of perturbations. This is defined as the adversarial accuracy and

is formulated as the fraction of test dataset for which the model is robust to all allowed perturbations under different metrics: l_0, l_1, l_2 or l_∞. In what follows we most commonly consider the l_∞ metric.

3.4.2 Experimental Setup

The MNIST database [74] of handwritten digits contains a training set of 60,000 examples and a test set of 10,000 examples in size of 28×28. The digits were size-normalised and centred in a fixed-size image. We implemented MAT under the perturbation constraints of size $\epsilon = 0.01, 0.02, 0.03$ in the l_∞ norm and evaluated the adversarial accuracy under different attacks of FGSM [29] and C&W [42] methods. Note that the values of input images were normalised in the interval $[0, 1]$. The training network consists of two convolution layers with 32 and 64 filters, respectively. Each convolutional layer is followed by a 2×2 max-pooling layer and a fully connected layer of size 1024. This network achieves 99.2% accuracy on the test set when evaluated on clean images.

The CIFAR10 dataset [75] is a labelled subset of the 80 million tiny image dataset. It contains a training set of 50,000 examples and a test set of 10,000 examples of 32×32 colour images in 10 different classes. These 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. As before, we implemented MAT under the perturbation constraints of size $\epsilon = 0.01, 0.02$ in the l_∞ norm and evaluated the adversarial accuracy under different attacks of FGSM [29] and C&W [42] methods. The values of input images were also normalised in the interval $[0, 1]$. The training network Resnet [55] was applied with 5 residual units with (16, 160, 320, 640) filters each on the CIFAR10 dataset; the resulting classifier achieved 95.2% accuracy on the test set when evaluated on clean images.

In the followings, we focus on small perturbation constraints which are sometimes not detectable by human eyes, so we do not consider large perturbations which might lose image resolutions. Also, these small perturbations are the most difficult ones to generate. With our MAT algorithm, we search for those critical points in the images for generating adversarial examples.

Dataset	Epsilon	**Original Acc.**		**FGSM Adv. Acc.**		l_0	l_1	l_2	Diff.
		PGD (%)	MAT (%)	PGD (%)	MAT (%)				
MNIST	$\epsilon = 0.01$	96.38	98.87	96.13	96.35	12.60	12.20	3.36	0.01
	$\epsilon = 0.02$	96.15	98.32	95.89	96.21	18.00	17.50	4.02	0.02
	$\epsilon = 0.03$	96.02	97.84	93.65	95.02	26.30	25.70	4.76	0.03
CIFAR10	$\epsilon = 0.01$	70.83	78.36	68.27	73.36	11.30	11.85	3.27	0.01
	$\epsilon = 0.02$	68.27	72.19	65.39	69.82	17.50	16.80	3.97	0.02

Table 3.1: The adversarial accuracy comparisons of PGD and MAT against FGSM attack method with MNIST and CIFAR10 datasets.

Dataset	Epsilon	**Original Acc.**		**C&W Adv. Acc.**		l_0	l_1	l_2	Diff.
		PGD (%)	MAT (%)	PGD (%)	MAT (%)				
MNIST	$\epsilon = 0.01$	96.38	98.87	93.24	94.23	12.60	12.20	3.36	0.01
	$\epsilon = 0.02$	96.15	98.32	93.12	94.19	18.00	17.50	4.02	0.02
	$\epsilon = 0.03$	96.02	97.84	91.32	92.85	26.30	25.70	4.76	0.03
CIFAR10	$\epsilon = 0.01$	70.83	78.36	64.84	70.12	11.30	11.85	3.27	0.01
	$\epsilon = 0.02$	68.27	72.19	62.93	67.98	17.50	16.80	3.97	0.02

Table 3.2: The adversarial accuracy comparisons of PGD and MAT against C&W attack method with MNIST and CIFAR10 datasets.

3.4.3 MNIST

We report some of the adversarial examples obtained with the MCTS-based attack method on MNIST dataset in Fig. 3.9. The images on the first row are the original clean images in 10 different classes; the second row is the corresponding adversarial examples using the MCTS-based attack. Through this method, only minor perturbations, where

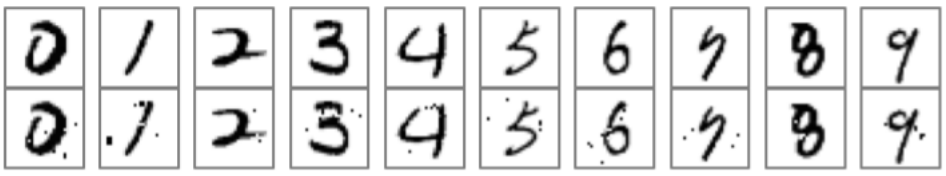
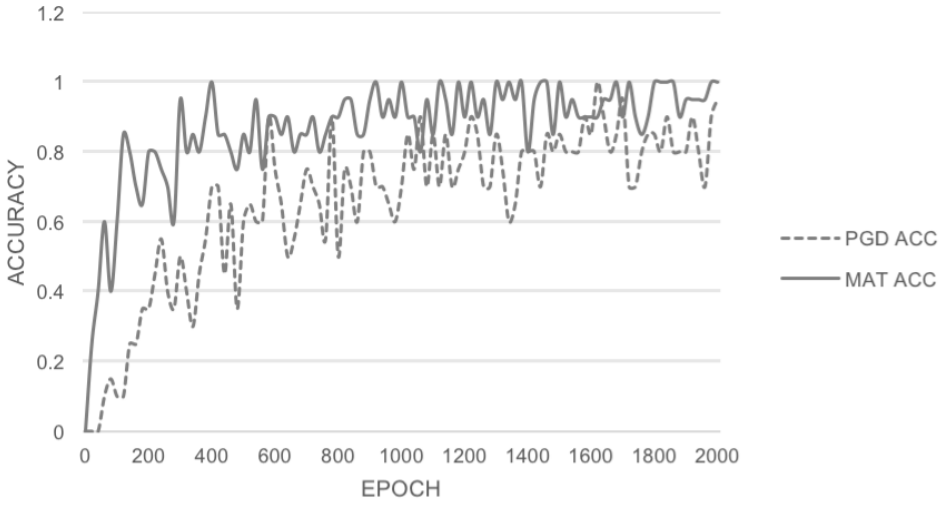
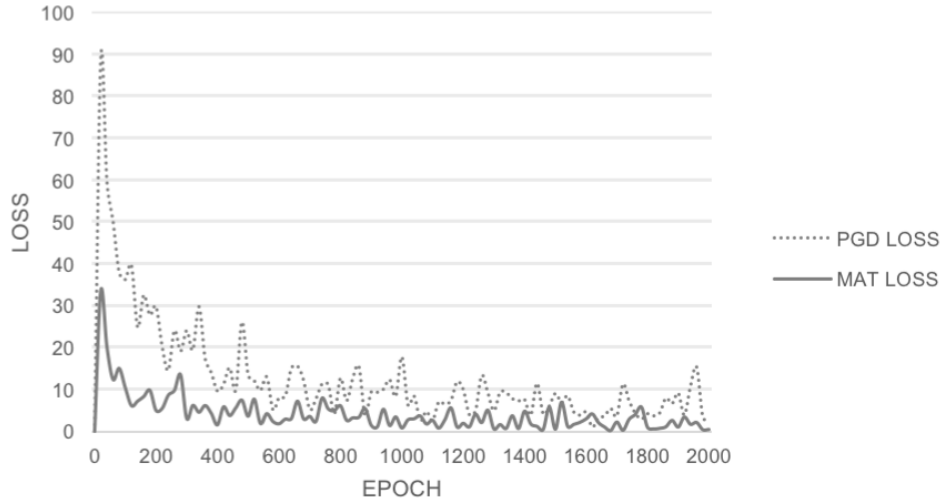


Fig. 3.9: Effective adversarial examples on MNIST dataset with $\epsilon = 0.01$.



(a) Adversarial Accuracy.



(b) Adversarial Loss.

Fig. 3.10: Adversarial accuracy and loss value on MNIST dataset with $\epsilon = 0.01$.

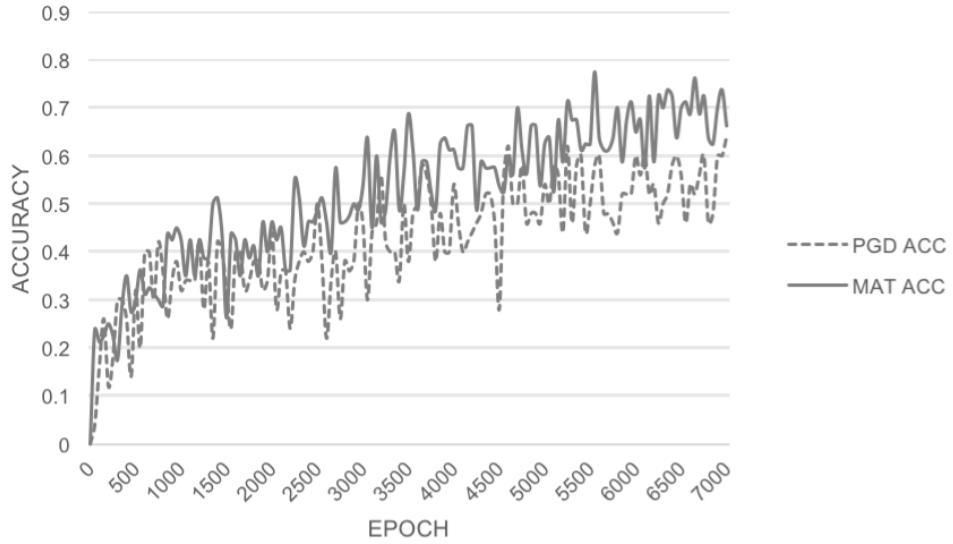
Fig. 3.11: Effective adversarial examples on CIFAR-10 dataset with $\epsilon = 0.01$.

$\epsilon = 0.01$, are required to generate attacks. These attacks are hard to detect by humans; see for example the one reported for class 8. In addition, these attacks are effective when used for adversarial training. As we show in Fig. 3.10, the resulting training process becomes efficient with the same training epochs. The time for the pre-training model in MAT is not considered in the training progress as the training time for it is small compared with the time of adversarial training. In the figure, the comparison is made against the results obtained by using Projected Gradient Descent (PGD) [26], one of the state-of-the-art methods in adversarial training. The figure reports an average improvement of efficiency of 21.1% over the first 2,000 epochs to achieve the same level of accuracy; the improved average loss value obtained is 8.63. The average memory usage for PGD and MAT are 1.5 and 1.8 GB respectively with a GPU of NVIDIA TITAN XP.

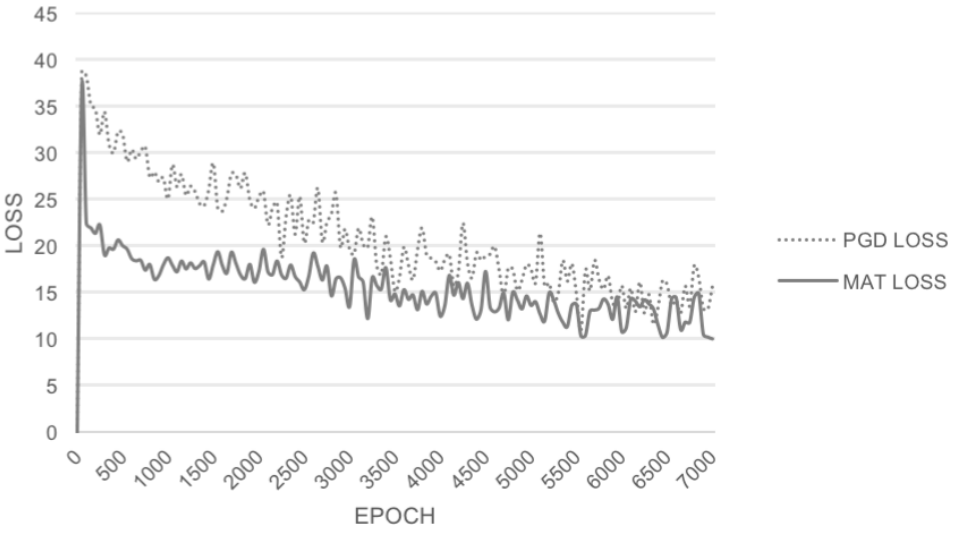
Table 3.1 summarises the experimental results on the MNIST dataset against FGSM attack method [29] under the perturbation constraints of size $\epsilon = 0.01, 0.02, 0.03$ in the l_∞ norm. Table 3.2 summarises the experimental results on the MNIST dataset against C&W attack method [42] also under the perturbation constraints of size $\epsilon = 0.01, 0.02, 0.03$ in the l_∞ norm. The results obtained for the average l_0, l_1, l_2 norms and the differences between clean images and adversarial examples are also presented in the table. Under the same training epoch of 2,000, MAT achieved better accuracy and robustness; this can be of benefit for both training efficiency and computational cost. The adversarial accuracy can be higher if more training epochs are adopted.

3.4.4 CIFAR10

In this subsection, we report the experiments obtained with the MCTS-based attack method on the CIFAR10 dataset (see Fig. 3.11). The images on the first row are the



(a) Adversarial Accuracy.



(b) Adversarial Loss.

Fig. 3.12: Adversarial accuracy and loss value on CIFAR10 dataset with $\epsilon = 0.01$.

original clean images from 10 different classes; the second row reports the corresponding adversarial examples obtained by using our MCTS-based attack method. Only minor perturbations, with $\epsilon = 0.01$, were required to generate adversary symbols. As before some attacks are difficult to discover by humans (see, e.g., the one for the deer class). In our experiments, we found the same results we obtained in MNIST. The overall training process was reduced significantly, as it is shown in Fig. 3.12. When comparing the results against PGD, we found an average improvement of efficiency of 9.8% when training up

to 7,000 epochs to achieve the same level of accuracy and an improved average loss value is 5.43. The average memory usage for PGD and MAT are 2.2 and 2.3 GB respectively with a GPU of NVIDIA TITAN XP.

The last two rows of Table 3.1 summarises the experimental results on the CIFAR10 dataset against FGSM attack method [29] under the perturbation constraints of size $\epsilon = 0.01, 0.02$ in the l_∞ norm. Table 3.2 summarises the experimental results on the CIFAR10 dataset against C&W attack method [42] also under the perturbation constraints of size $\epsilon = 0.01, 0.02$ in the l_∞ norm. The average l_0, l_1, l_2 norms and the differences between clean images and adversarial examples are also presented in the table. The evaluation confirms the results obtained with MNIST. Comparing with PGD under a training epoch of 7,000, MAT obtains higher accuracy with less training efforts and computational cost. In the same way with MNIST, the adversarial accuracy can be better if more training epochs are implemented. In summary, the MNIST and CIFAR experimental results show that the proposed method is effective and general against both single and multiple colour channels.

3.5 Summary

In this chapter, we proposed an MCTS-based adversarial training framework and presented the experimental results obtained on MNIST and CIFAR10 datasets. For MNIST, after 2000 epochs the experimental results showed an average improvement of efficiency of 21.1% when compared to PGD. For CIFAR10, after 7000 epochs we obtained an average improvement of efficiency of 9.8% compared to PGD. We further compared the robustness of the algorithm against previous work against C&W attack method. The results suggested that the adversarial training method accompanied with MCTS is not only robust with respect to adversarial examples but also efficient during training.

We especially focused the evaluations on small perturbations since pre-processing components like denoising elements are normally included prior to a neural network model in real applications. The contributions of this work show: i) that the method is computationally attractive, ii) it maintains competitive accuracies compared with PGD adversarial

training against the FGSM and C&W attack, and iii) does not appear to be susceptible to local optima.

Chapter 4

A Decision Tree Search Robustness Method

Adversarial training provides a rigorous framework for understanding, analysing, and improving the robustness of DNNs by considering norm-constrained perturbations. However, adversarial training requires learning via a large number of perturbed examples before resistance is obtained. This is expensive and time-consuming. Therefore, developing computationally effective and robust training approaches is a topic of interest.

In this chapter, we develop a robust learning method based on decision tree search and robust optimisation. The method involves decision tree search targeting the worst-case data points to generate adversarial examples. The decision tree search method is combined with robust optimisation to training a robust model while maintaining accuracy at comparably lower computational effort than SoA methods. Given an arbitrary input to a DNN, our algorithm searches in small regions centred around the input that have significant contributions to generate adversarial samples. This method doesn't need to access the internal layers of DNNs and thus falls in the realm of black-box adversarial attack. As we show, the method results to be more robust against different adversarial attacks and is more competitive compared with Fast Gradient Sign Method (FGSM) [29] and Projected Gradient Descent (PGD) [26], the state-of-the-art method in adversarial training. Our method also reduces significantly the number of adversarial examples re-

quired for adversarial training, which leads us to computationally attractive advantages when generating robust models.

The remainder of this chapter is organised as follows. In Section 4.1 we describe a novel method to generate adversarial examples with decision tree search attack. Section 4.2 introduces how adversarial examples are employed for adversarial training via robust optimisation. Section 4.3 reports experimental results obtained on the MNIST and CIFAR-10 datasets and a brief summary concludes the chapter.

The material presented in this chapter is presently under review and is loosely based on research first presented in the following paper [3]:

- Y. Liu and A. Lomuscio. Robustness Learning via Decision Tree Search Robust Optimisation. Proceedings of the 32nd British Machine Vision Conference (BMVC). United Kingdom, 2021. BMVA Press.

4.1 Decision Tree Search Attack

In this section, we present a decision tree search adversarial attack DTSAttack to generate effective adversarial examples. The method is composed of four steps, i.e. initialise spanning tree, tree traversal, sampling nodes, and back propagation, which are summarised in Algorithm 3. In subsection 4.1.1, we introduce how to initialise spanning tree for tree traversal; in subsection 4.1.2, we explain how to traverse tree according to a confidence value and initialise explorable nodes. We then sample values for each node on the tree from the explorable nodes in subsection 4.1.3 and finally update the confidence information in the back propagation step in subsection 4.1.4. The iteration will continue until the termination conditions are satisfied.

4.1.1 Initialise Spanning Tree

For each image $x \in \mathbb{R}^d$ in d dimensions, x can be separated into m sub-regions with the number of pixels j in each sub-region. To search the most potential pixels with a

Algorithm 3: Decision Tree Search Adversarial Attack: DTSAttack

```
function DTSAdversarialAttack ;
  Input  : Clean image dataset x
           Initialise perturbation constraint setting for ϵ
           Initialise search trees T
  Output: Effective adversarial examples x'
Initialise Spanning Tree to obtain maximal distance values;
while not terminalNode and Pert ≤ ϵ and time < TimeOut do
    while iterationTime < stepSearchTime do
        Tree Traversal: Traverse the spanning tree according to the confidence value for
         each node i;
        Initialise Exploration Nodes: Explore available expanding nodes N_e;
        for each exploration nodes N_e do
            Sampling Nodes: Randomly choose one region from available sub-regions;
            Back Propagation: Update associated information for each node along the
             path;
        end
    end
    Choose Best Child Node: Choose one of the best path from the root node ;
    Make One Move: Make one move based on the best exploration node as child node
     and update new root node;
end
```

higher probability of generating an adversarial example, we first compute each pixel by the distance between the average value $x_a = \frac{\sum_{i=0}^{d} x_i}{d}$ and each pixel value as $\mathcal{D} = \|x_i - x_a\|$, where x_i is the value of each pixel i, and the matrix $\mathcal{D}$ is the distance of each pixel x_i from the average value. The larger value in $\mathcal{D}$, it stands for the more potential pixel intuitively. We then sort $\mathcal{D}$ by the value with descent order as $\mathcal{D}_\mathcal{S} = \mathit{Sort}(\mathcal{D})$ and divide $\mathcal{D}_\mathcal{S}$ into m sub-regions N_m with the number of points j in each sub-region. This initialisation step establishes a starting search step for the root node N_r of the spanning tree, which will be the start point of the next tree traversal step.

4.1.2 Tree Traversal

We now present how to expand nodes from the initialised spanning tree and what criteria are used to make decisions of choosing nodes in the tree traversal step. In this step, we consider the most promising moves according to the confidence value of each node in the spanning tree. Each node of the spanning tree is regarded as a sub-region N_m from the previous step. In each move k, we choose a child node N_c with the highest confidence value down to a leaf node N_l. The confidence value $\mathcal{C}_k$ of associated information in each node N_c of the spanning tree is formulated as Equation (4.1), which is on the basis of Monte Carlo Tree Search [1] algorithm, a decision search algorithm for decision process, and revised accordingly:

$$\mathcal{C}_k = \frac{w_k}{n_k} + c\sqrt{\frac{\ln N_k}{n_k}}, \tag{4.1}$$

where w_k stands for the Euclidean distances for the node considered after the k-th move, n_k stands for the number of visits for the node considered after the k-th move, and N_k stands for the total number of visits after the k-th move. The exploration parameter c is theoretically equal to $\sqrt{2}$. The higher the confidence value means the more contributions to search adversarial examples.

4.1.3 Sampling Nodes

From the leaf node N_l in the previous step, we then choose among the exploration nodes N_e for the sampling step. The explorable nodes consist of the remaining sub-regions excluding the ancestors of a node N_a in the tree. For example, the explorable nodes of the root node N_r are the remaining sub-regions from the sub-regions in the root node. We first sample nodes from these explorable nodes and simulate whether an adversarial example is generated. This step continues by choosing from the remaining sub-regions randomly and applying perturbations accordingly on the datapoints j in each sub-region until the end of the search time. We then simulate based on these newly perturbed points and examine whether an adversarial example is found. The randomly choosing process is formulated as Equation (4.2):

$$\mathcal{RC} = random(\mathcal{S}_{avai}(N_e) - \mathcal{S}_{used}(N_a)), \tag{4.2}$$

where *random* choose one region from the remaining sub-regions randomly, $\mathcal{S}_{avai} = \{N|N \in N_e, \text{exploration nodes}\}$ is the available set of explorable sub-regions and $\mathcal{S}_{used} = \{N|N \in N_a, \text{ancestor nodes}\}$ is the sub-regions in the ancestors of a node.

4.1.4 Back Propagation

We now present how to update the information for the newly explored nodes under the constraint with $\|\delta\|_p \leq \epsilon$. We select one path from the previous sampling step with the maximal distance between the newly perturbed and the clean points. We then back propagate and update the corresponding distance value and number of visits for the confidence values of nodes along the expanding path.The perturbation amount $\mathcal{P}$ in each iteration is constrained with $\|\delta\|_p \leq \epsilon$, which is formulated as Equation (4.3):

$$\mathcal{P} = \| \sum_{t=0}^{T} \sum_{r=0}^{R} \Delta_{tr} \|_p \leq \epsilon, \tag{4.3}$$

where $\sum_{r=0}^{R} \Delta_{tr}$ stands for the number of perturbations in each node, and T is the total number of nodes along the same path.

We explain more details for the whole process in Fig. 4.1a-4.1d. In Fig. 4.1a, the image is divided into m sub-regions and the root node N_1 is selected as the start point in the spanning tree. In Fig. 4.1b, the tree is expanded to N_2 and N_3. These two nodes are then sampled and the updated confidence values are back-propagated accordingly. In Fig. 4.1c, the path with highest confident values of nodes (N_1, N_2) is selected. The expanded nodes N_4 and N_5 are then sampled to check if an adversarial example is acquired. The confidence values are updated for each node of the whole path accordingly. In Fig. 4.1d, the nodes (N_1, N_2, N_5) are selected and the nodes (N_6, N_7) are expanded from N_5. The simulation and back-propagation steps are applied as previously mentioned. These iterations will continue until the termination conditions are satisfied. The Δ_{tr} stands for the number of perturbations in each node. For example, Δ_{1r} in the node N_1 is the perturbations applied

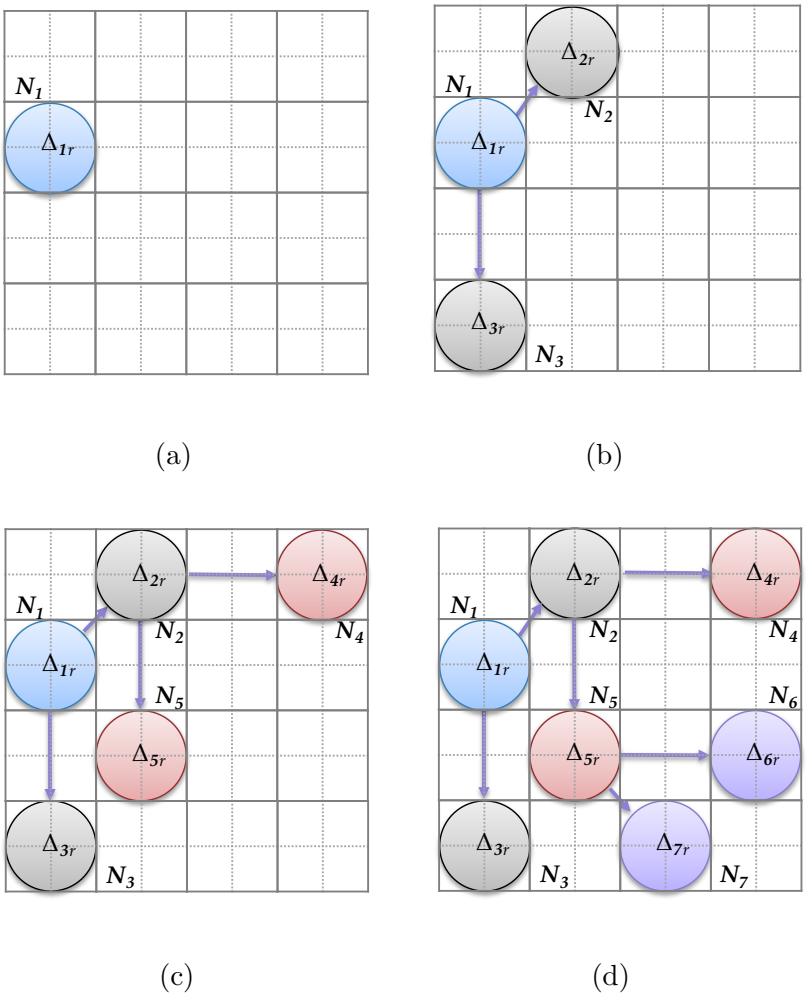


Fig. 4.1: The process of the decision tree search attack.

when N_1 is selected. The overall perturbations applied in an adversarial example are the summation of perturbations of the nodes along the whole path.

The decision tree search attack is summarised in Algorithm 3. The algorithm starts from the step of initialising the spanning tree (line 2). In the search iteration, the spanning tree is traversed according to the confidence value (line 5) and available expanding nodes are then explored (line 6). For each exploration node, one region is randomly chosen from available sub-regions for sampling (line 8). Lastly, the associated information is updated for the nodes along the search path (line 9). These iterations (line 4-13) will continue until the termination conditions are satisfied.

4.2 The DTS Robust Tool

In this section, we introduce a robust optimisation method in subsection 4.2.1, which interacts with DTSAttack from the previous section to form the basis of the DTSRobust toolkit described in subsection 4.2.2. The overall algorithm, including the robust optimisation method, is presented in Algorithm 4. To be more specific, we combine robust optimisation with DTSAttack thereby obtaining a method that is evaluated against other attack methods, which also include robust optimisation.

4.2.1 Robust Optimisation

The robust optimisation method [79] described below aims to obtain stable solutions under uncertainty of the data. The uncertainty has a deterministic and worst-case nature; perturbations to the data are drawn from uncertainty sets $\mathcal{U}$. The objective in robust optimisation is to obtain solutions, which are feasible and well-behaved under any realisations of the uncertainty from $\mathcal{U}$. An optimal solution among feasible solutions is the one that has minimal cost given the worst-case realisation from $\mathcal{U}$. Robust optimisation thus normally have a min-max formulation, where the objective function is minimised with respect to a worst-case realisation of perturbations. For example, consider a linear programming problem in Equation (4.4):

$$\min_{x}\{c^T x : Ax \leq b\}, \tag{4.4}$$

where the given data is (A, b, c) and the objective is to search a solution x, which is robust to perturbations in the data. No solution can be well-behaved if the perturbations of the data are arbitrary. Hence, we restrict to only allowing the perturbations existing in the uncertainty set $\mathcal{U}$. The corresponding robust optimisation is formulated as:

$$\min_{x} \sup_{(A,b,c)\in\mathcal{U}} \{c^T x : Ax \leq b\}, \tag{4.5}$$

where the objective is to select an x that can work well with all instances of the problem parameters within the uncertainty set $\mathcal{U}$.

Given this, the problem can be formulated as the search for a stable solution in a small neighbourhood around every training point x_i. This neighbourhood corresponds to the uncertainty set $\mathcal{U}_i$. For example, we may set $\mathcal{U}_i = R_p(x_i, r)$, a region with radius r around x_i with respect to some norm p. To do so, we select from the neighbourhood a representative $x_i' = x_i + \delta_{x_i}$, the point on which the network output will induce the greatest loss. The network output on x_i' is required to be y_i, the target output for x_i. Assuming that many test points are close to training points from the same class, we expect that the training algorithm will have a regularisation effect and consequently will improve the network performance on test data. Moreover, we expect this approach to increase the robustness of the network output to adversarial example. Hence, the training network is optimised with a minimisation-maximisation approach:

$$\min_{\theta} \mathcal{L} = \min_{\theta} \sum_{i=1}^{m} \max_{\delta \leq \|\epsilon\|} \mathcal{L}(x_i', \theta, y_i), \tag{4.6}$$

where δ is the uncertainty set under the constraint ϵ corresponding to the adversarial example x_i'. This involves optimising the model parameter θ with respect to a worst-case data (x_i', y_i), rather than against the original training data; the i-th worst-case data point is selected from the uncertainty set δ. These uncertainty sets are determined by the problem at hand; adversarial training [29] can be understood as one such problem.

4.2.2 DTS Implementation Framework

We now introduce the decision tree search robust optimisation algorithm in Algorithm 4. We first initialise a non-trained neural network $\mathcal{M}$ and a pre-trained model $f(\cdot)$. We then choose a minibatch of size m of images from input dataset x and generate size m of corresponding adversarial examples with our DTSAttack. These corresponding adversarial examples are then applied to model $\mathcal{M}$ for robust optimisation. Lastly, the related model loss is updated according to the values from the minibatch.

A more detailed explanation is shown in Fig. 4.2, and is divided into two steps:

Algorithm 4: DTS Robust Optimisation: DTSRobust

Size of the training minibatch is m

function DTS_Robust_Optimisation ($\mathcal{M}$);
Setup Pre_Training Model $f(\cdot)$
Randomly initialise neural network $\mathcal{M}$
while *training not converged* **do**
    Read minibatch $B = \{x_1, ..., x_m\}$ from training set;
    Generate m adversarial examples $B_{adv} = \{x'_1, ..., x'_m\}$ from corresponding clean examples B with DTSAttack;
    Do one training step of network $\mathcal{M}$ using B_{adv};
    Update model loss with min_max optimisation: $\min_\theta \mathcal{L} = \min_\theta \sum_{i=1}^{m} \max_\eta \mathcal{L}(x'_i, \theta, y_i)$;
end

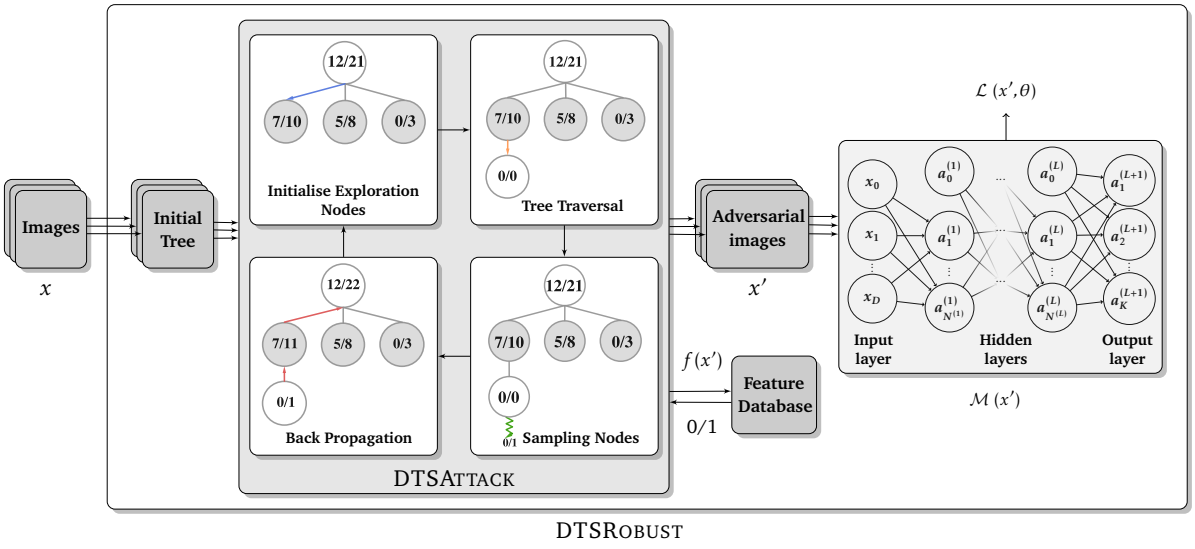


Fig. 4.2: The DTSRobust toolkit training framework. The framework is divided into an attack generation step DTSAttack and a robust training model $\mathcal{M}$.

the first consists of an attack generation step with DTSAttack; the second is a robust optimisation step. For the attack generation part, first, a model $f(\cdot)$ is trained as an assistant classifier from a training dataset of images x. Then, the procedure follows a training loop. In each training loop, corresponding to a training epoch, a minibatch of size m of images is randomly selected from the training dataset x. This minibatch is then analysed by the DTSAttack to generate adversarial examples x'. Whether or not the images generated by DTSAttack are proper adversarial examples is determined by a test on the assistant classifier $f(\cdot)$. The resulting adversarial examples x' form the basis for training the robust model $\mathcal{M}(\cdot)$, which is initially untrained. The loss value $\mathcal{L}$ is updated according to the softmax result of the robust model $\mathcal{M}$. The training loop continues for as many epochs as required until the required accuracy is converged.

Following the framework mentioned above, the DTSRobust randomly selects a minibatch in each epoch until the loss of the robust model converges to the desired value. The convergence criteria ensure that the resulting model $\mathcal{M}$ is robust in small neighbourhoods of every training point around x. We call these neighbourhoods the perturbations δ and we represent them as $x' = x + \delta$. The overall process can be regarded as a solution to the robust optimisation problem against adversarial examples.

4.3 Experimental Results

In this section, we evaluate the DTS robust optimisation algorithm presented in the previous section and report the results using MNIST [74] and CIFAR10 [75] datasets. We evaluate the robustness obtained against different attack methods, namely FGSM, PGD, and DTS. The experimental setup and results are shown from subsection 4.3.1 to 4.3.5.

4.3.1 Experimental Setup

The MNIST database of handwritten digits contains a training set of 60,000 examples and a test set of 10,000 examples. The digits were size-normalised and centred in a fixed-size image of 28×28. We generated adversarial examples under the perturbation

Dataset	**Network Name**	**Network Architecture**	**Test Error**
MNIST [74]	Simple Network (S. (MDL Training)	Conv1($\begin{bmatrix} 5, & 5, & 1, & 32 \end{bmatrix}$, 2×2) Conv2($\begin{bmatrix} 5, & 5, & 32, & 64 \end{bmatrix}$, 2×2) Full($\begin{bmatrix} 7 \times 7 \times 64, & 1024 \end{bmatrix}$) Avg-pool($\begin{bmatrix} -1, & 7 \times 7 \times 64 \end{bmatrix}$)	2.3
	Wide Network (W. (MDL Training)	Conv1($\begin{bmatrix} 5, & 5, & 1, & 64 \end{bmatrix}$, 2×2) Conv2($\begin{bmatrix} 5, & 5, & 64, & 128 \end{bmatrix}$, 2×2) Full($\begin{bmatrix} 7 \times 7 \times 128, & 1024 \end{bmatrix}$) Avg-pool($\begin{bmatrix} -1, & 7 \times 7 \times 128 \end{bmatrix}$)	1.8
CIFAR10 [75]	Simple Network (S. (MDL Training)	Conv1($\begin{bmatrix} 3 \times 3, & 16 \end{bmatrix}$) Conv2($\begin{bmatrix} 3 \times 3 & 16 \times \mathrm{k} \\ 3 \times 3 & 16 \times \mathrm{k} \end{bmatrix} \times$ N, k:1 N:5) Conv3($\begin{bmatrix} 3 \times 3 & 32 \times \mathrm{k} \\ 3 \times 3 & 32 \times \mathrm{k} \end{bmatrix} \times$ N) Conv4($\begin{bmatrix} 3 \times 3 & 64 \times \mathrm{k} \\ 3 \times 3 & 64 \times \mathrm{k} \end{bmatrix} \times$ N) Avg-pool($\begin{bmatrix} 8 \times 8 \end{bmatrix}$)	4.9
	Wide Network (W. (MDL Training)	Conv1($\begin{bmatrix} 3 \times 3, & 16 \end{bmatrix}$) Conv2($\begin{bmatrix} 3 \times 3 & 16 \times \mathrm{k} \\ 3 \times 3 & 16 \times \mathrm{k} \end{bmatrix} \times$ N, k:10 N:5) Conv3($\begin{bmatrix} 3 \times 3 & 32 \times \mathrm{k} \\ 3 \times 3 & 32 \times \mathrm{k} \end{bmatrix} \times$ N) Conv4($\begin{bmatrix} 3 \times 3 & 64 \times \mathrm{k} \\ 3 \times 3 & 64 \times \mathrm{k} \end{bmatrix} \times$ N) Avg-pool($\begin{bmatrix} 8 \times 8 \end{bmatrix}$)	4.2

Table 4.1: The full network architectures and parameters for MNIST and CIFAR10.

constraints of size $\epsilon = 0.02$ and 0.03 respectively in the l_∞ norm. To investigate model capacity, we consider two training networks of simple and wide architectures, respectively. The simple network consists of two convolution layers of sizes 32 and 64 filters, and a fully connected layer of size 1024. The wide network consists of two convolution layers of sizes 64 and 128 filters, and also a fully connected layer of size 1024. Both networks are

adversarially trained with FGSM, PGD and DTS methods.

The CIFAR10 dataset contains a training set of 50,000 examples and a test set of 10,000 examples of 32×32 colour images in 10 different classes. The values of input images were also normalised in the interval $[0, 1]$. As before, we generated adversarial examples under the perturbation constraints of size $\epsilon = 0.02$ and 0.03 in the l_∞ norm. For the CIFAR10 dataset, we used the Resnet model [55] as the simple network and modify the network via using wider layers by a factor of 10, resulting in a network with 5 residual units with (16, 160, 320, 640) filters each. We also performed adversarial training with FGSM, PGD and DTS methods on these two architectures and investigated the resulting accuracies against white-box attack adversaries on different adversarial trained models.

4.3.2 Network Architecture

To investigate model capacity, we consider two training networks of simple and wide architectures for MNIST dataset [74], respectively. The simple network consists of two convolution layers of sizes 32 and 64 filters, and a fully connected layer of size 1024. The wide network consists of two convolution layers of sizes 64 and 128 filters, and also a fully connected layer of size 1024. Both networks are adversarially trained with FGSM, PGD and DTS methods. The test errors for these two network architectures are 2.3% and 1.8% respectively, and the detailed parameters for each layer are summarised in the first two rows of Table 4.1. The MDL in the table is represented as one of the FGSM, PGD and DTS methods.

For the CIFAR10 dataset [75], we used the Resnet model [55] with different filters as the simple and wide networks. The network width is determined by factor k and the groups of convolutions are shown in brackets where N is the number of blocks in a group. For the simple network, the factor k is set as 1 while the number of blocks N is set as 5. For the wide network, a modified one is implemented via wider layers by a factor, resulting in a network of (16, 160, 320, 640) filters with 5 residual units in each convolution, where the factor k is set as 10. The network width is determined by factor k and the groups of convolutions are shown in the table where N is a number of blocks in

Target Model \ Adversary	Nature	FGSM	FGSM^{R}	PGD^{2}	PGD^{5}	PGD^{20}	DTS^{1}	DTS^{2}	l_2	ϵ_{DIFF}
S. (FGSM Training)	96.2	95.1	94.8	92.7	91.4	89.7	90.8	89.6	3.87	0.02
S. (PGD Training)	95.8	95.2	94.2	93.6	92.7	90.6	91.5	90.8	3.71	
S. (DTS Training)	**97.7**	**96.3**	**95.1**	**93.8**	**93.3**	**91.9**	**91.9**	**91.6**	3.27	
W. (FGSM Training)	97.4	96.7	96.4	94.6	94.1	92.7	92.8	92.3	3.95	
W. (PGD Training)	96.9	95.7	95.3	95.1	94.8	93.4	93.5	93.1	3.86	
W. (DTS Training)	**97.8**	**96.8**	**96.3**	**95.8**	**95.2**	**94.1**	**93.9**	**93.6**	3.79	
S. (PGD Training)	94.8	92.7	91.9	90.7	89.9	88.7	89.5	88.6	4.17	0.03
S. (DTS Training)	**96.5**	**94.1**	**93.2**	**92.7**	**92.3**	**90.6**	**90.5**	**90.1**	4.04	
W. (PGD Training)	96.1	92.8	92.1	91.3	90.3	89.7	90.1	89.3	4.28	
W. (DTS Training)	**97.3**	**94.3**	**93.7**	**93.1**	**92.5**	**91.7**	**91.8**	**91.2**	4.21	

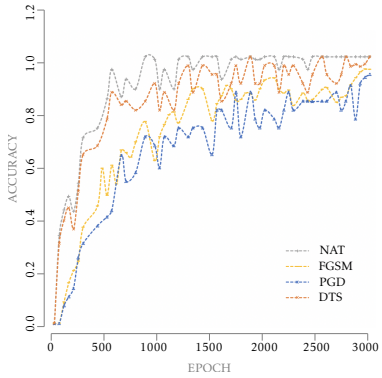
Table 4.2: The resulting accuracy of nature training, FGSM, PGD and DTS methods against white-box adversarial attacks with $\epsilon_{\text{DIFF}} = 0.02$ and 0.03 on MNIST dataset.

group [55]. The resulting ResNet28-10 model [80] is adopted by an error rate of 4.17% and with 36.5M parameters at most. We also perform adversarial training with FGSM, PGD and DTS methods on these two architectures and investigated the resulting accuracies against white-box attack adversaries on different adversarial trained models. The details are enumerated in the last two rows of Table 4.1.

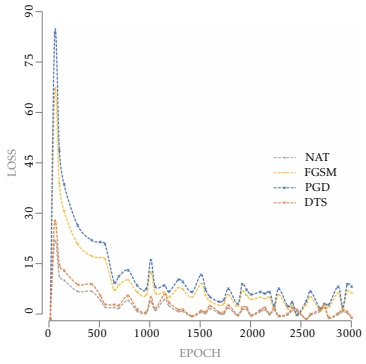
In all our experiments we use Stochastic Gradient Descent (SGD) with momentum and cross-entropy loss. The initial learning rate is set to 0.1, weight decay to 0.0005, dampening to 0, momentum to 0.9 and minibatch size to 128. On CIFAR learning rate dropped by 0.1 at 10,000, 15,000 and 20,000 epochs and we train for total 30,000 epochs. We select the ResNet28-10 model [80] as it outperforms the original ResNet [55] by 0.92% (with the same minibatch size during training) on CIFAR10.

4.3.3 MNIST

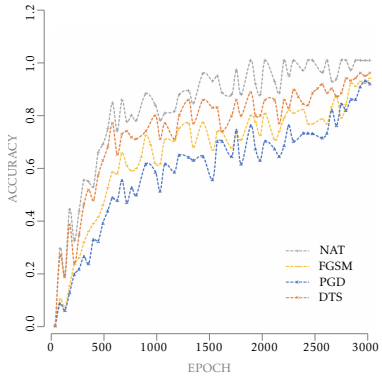
Table 4.2 summarises the resulting accuracies obtained on the MNIST dataset. We generated adversarial examples using the white-box attack method of FGSM, PGD and



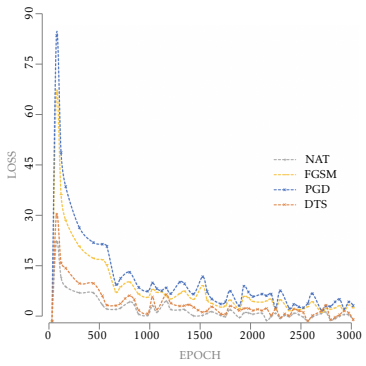
(a) Adversarial accuracy comparisons of nature, FGSM, PGD and DTS with $\epsilon_{\text{DIFF}} = 0.02$.



(b) Adversarial loss comparisons of nature, FGSM, PGD and DTS with $\epsilon_{\text{DIFF}} = 0.02$.



(c) Adversarial accuracy comparisons of nature, FGSM, PGD and DTS with $\epsilon_{\text{DIFF}} = 0.03$.



(d) Adversarial loss comparisons of nature, FGSM, PGD and DTS with $\epsilon_{\text{DIFF}} = 0.03$.

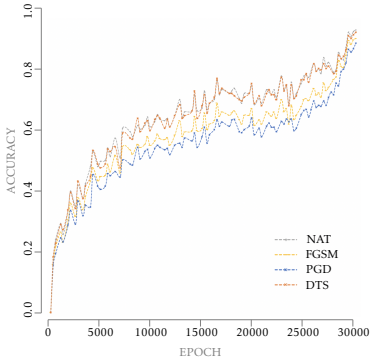
Fig. 4.3: Accuracy and loss comparisons on MNIST dataset with $\epsilon_{\text{DIFF}} = 0.02, 0.03$ in the l_∞ norm.

DTS with $\epsilon_{\text{DIFF}} = 0.02$ and 0.03, and then evaluated them on a target network, that was adversarially trained independently, with different methods. The target models consist of two different architectures, which are simple and wide networks. For example, S.(FGSM Training) means the target model trained with FGSM adversarial training using a simple network. For each target model, we choose randomly and fairly from the full set of training set for 3,000 epochs with minibatch size of 128. The results would be similar even running for several times since each time the number of trained images is millions of images, e.g., MNIST with 0.4M of images and this is fairly massive for MNIST with

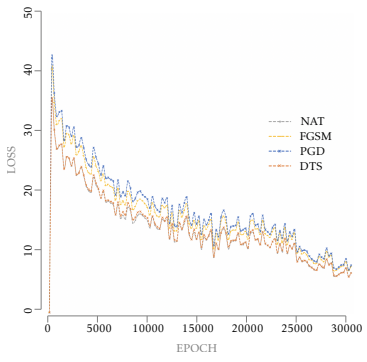
Target Model \ Adversary	Nature	FGSM	FGSM^{R}	PGD^{2}	PGD^{5}	DTS^{1}	DTS^{2}	l_0	l_1	l_2	ϵ_{DIFF}
S. (FGSM Training)	93.8	93.0	92.5	89.5	89.1	87.9	87.0	14.5	13.6	3.93	0.02
S. (PGD Training)	93.3	92.4	92.0	91.6	91.2	90.1	88.9	13.8	12.9	3.82	
S. (DTS Training)	**94.8**	**93.2**	**92.8**	**91.9**	**90.7**	**90.5**	**89.8**	12.3	11.3	3.74	
W. (FGSM Training)	94.2	93.9	93.3	91.4	90.8	90.3	89.5	14.2	13.9	3.91	
W. (PGD Training)	94.0	93.7	93.4	92.9	92.3	91.6	91.0	13.9	13.5	3.84	
W. (DTS Training)	**94.9**	**94.1**	**93.7**	**93.2**	**92.5**	**91.7**	**91.5**	12.4	11.9	3.79	
S. (PGD Training)	92.4	89.8	88.4	87.6	86.9	86.4	85.9	20.8	18.9	4.22	0.03
S. (DTS Training)	**93.2**	**92.5**	**91.8**	**90.7**	**89.7**	**88.6**	**87.4**	19.2	18.3	4.04	
W. (PGD Training)	93.4	90.1	88.9	88.1	87.4	87.1	86.2	20.3	19.2	4.34	
W. (DTS Training)	**93.8**	**92.5**	**92.1**	**91.4**	**90.2**	**89.3**	**88.6**	19.5	18.6	4.21	

Table 4.3: The resulting accuracy of nature training, FGSM, PGD and DTS methods against white-box adversarial attacks with $\epsilon_{\text{DIFF}} = 0.02$ and 0.03 on CIFAR10 dataset.

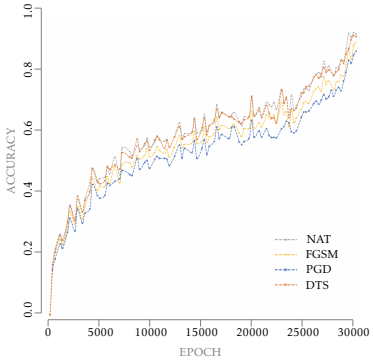
$\epsilon_{\text{DIFF}} = 0.02$ and 0.03. For this reason, the confidence interval results are not shown in the experiments since we can approximate the distribution with a Gaussian with large sample sizes. This is expressed as $interval = z \cdot \sqrt{(accuracy \cdot (1 - accuracy))/n}$, where *interval* is the radius of the confidence interval, *accuracy* is classification accuracy, n is the size of the sample and z is the number of standard deviations from the Gaussian distribution. The first column (Nature) stands for the accuracy of each adversarially trained target network without attacks. The FGSM random attack (FGSM^{R}) was implemented according to [22], whereby small random perturbations are performed before applying FGSM. The PGD attack considered 10 random restarts uniformly distributed under ϵ per input and settings of 2, 5 and 20 steps with step size 0.01. The search time for DTS attack is constrained in 1 second and 2 seconds respectively; once an adversarial example is found, the process will be terminated. From the results, the attack strength from strong to weak is DTS, PGD and then FGSM as the accuracies against DTS attack (DTS^{X} columns) are lower than PGD and FGSM columns. This means that DTSAttack is a stronger attack, and thus the resulting accuracies among different trained networks are lower. In addition, the results show that DTS contributes to improved accuracies against different adversaries,



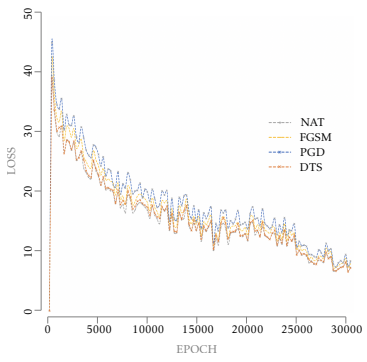
(a) Adversarial accuracy comparisons of nature, FGSM, PGD and DTS with $\epsilon_{\text{DIFF}} = 0.02$.



(b) Adversarial loss comparisons of nature, FGSM, PGD and DTS with $\epsilon_{\text{DIFF}} = 0.02$.



(c) Adversarial accuracy comparisons of nature, FGSM, PGD and DTS with $\epsilon_{\text{DIFF}} = 0.03$.



(d) Adversarial loss comparisons of nature, FGSM, PGD and DTS with $\epsilon_{\text{DIFF}} = 0.03$.

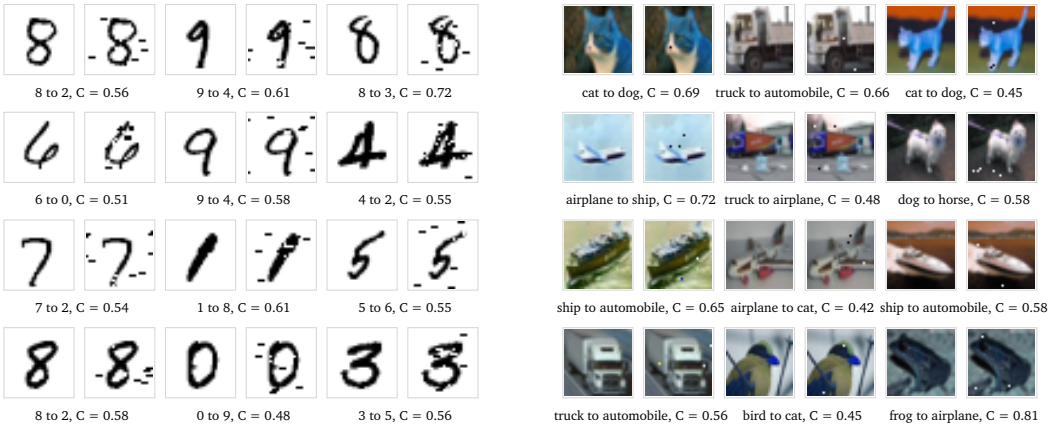
Fig. 4.4: Accuracy and loss comparisons on CIFAR10 dataset with $\epsilon_{\text{DIFF}} = 0.02, 0.03$ in the l_∞ norm.

and maintain robustness even under the DTS attack itself. For example, the accuracy of S. (DTS training networks) is 93.8, which is higher than other training networks 92.7 and 93.6 under the PGD attack (column PGD^2). Moreover, changing the architecture from simple to wide networks also contributes to accuracies overall. The accuracies decrease only few percentages even increasing ϵ_{DIFF} to 0.03 and the W. (DTS training) is more robust than W (PGD Training) in general. The average distances required for DTS are smaller than the others as DTS searches mainly the most potential features. Figure 4.3 reports the accuracy and loss trends for different adversarial training methods over the

first 3,000 epochs with $\epsilon_{\text{DIFF}} = 0.02$ and 0.03. The average memory usage for FGSM, PGD and DTS are 1.5, 1.7, and 1.8GB respectively. We find that the accuracies of robust optimisation with DTS converges faster than the other two state-of-the-art methods with 14.2% on average when compared with PGD method. The improvement is computed with the accuracy differences between DTS and PGD by epochs during training. The reason for this gain is that we only search important features as opposed to adding perturbations with randomly distributed methods. With these advantages, DTS does save training efforts and contribute to robustness during adversarial training.

4.3.4 CIFAR10

Table 4.3 summarises the resulting accuracies obtained on the CIFAR10 dataset with similar setting as MNIST. For each target model, we also choose randomly and fairly from the full set of training set for 30,000 epochs with minibatch size of 128. The results would be similar even running for several times since each time the number of trained images is millions of images, e.g., CIFAR10 with 3.8M of images and this is fairly massive for CIFAR10 with $\epsilon_{\text{DIFF}} = 0.02$ and 0.03. The results obtained also demonstrate that a strong adversary can help to improve model accuracies. In addition, DTS contributes to improving the accuracies against different adversaries while retaining robustness against the DTS attack itself. For example, the accuracy of W. (DTS training network) under the DTS attack itself (column DTS^2) is 91.5, which is higher than other networks with 91.0 or 89.5 (e.g., W. (PGD training network) or W. (FGSM training network)) with $\epsilon_{\text{DIFF}} = 0.02$. Comparing the results against different network architectures reveals that changing the architecture from simple to wide networks can contribute to accuracies generally. The average distances required for DTS are smaller than FGSM and PGD as DTS only searches for the most potential features. Figure 4.4 reports the accuracy and loss trends for different adversarial training methods over the first 30,000 epochs with $\epsilon_{\text{DIFF}} = 0.02$ and 0.03. The average memory usage for FGSM, PGD and DTS are 1.9, 2.2, and 2.3GB respectively. The data obtained support the conclusion that the adversarial training with DTS converges faster with 10.3% on average than the other state-of-the-art methods.



(a) MNIST adversarial images with $\epsilon_{=0.02}$. (b) CIFAR10 adversarial images with $\epsilon_{=0.02}$.

Fig. 4.5: Some adversarial examples for MNIST and CIFAR10 under $\epsilon_{\text{DIFF}} = 0.02$.

4.3.5 Adversarial Examples with DTS

We present some adversarial examples obtained for MNIST and CIFAR10 in Figure 4.5 with different confidence values under $\epsilon_{\text{DIFF}} = 0.02$. Through DTSAttack, only minor perturbations, where $\epsilon_{\text{DIFF}} = 0.02$, are required to generate attacks. The odd columns are the original images with correct classes and the even columns are their corresponding adversarial examples. In the figure, most confidence values are over 0.5 while generating adversarial examples. From the results, the adversarial ones are still clear to distinguish from the correct classes. Some of these attacks are not easy to detect by humans; see for example the ones reported for class cat to class dog with confidence values of 0.69 and 0.45 in CIFAR10.

4.4 Summary

In summary, we proposed a decision tree search robust optimisation framework and presented the experimental results obtained on MNIST and CIFAR10 datasets with different perturbation settings of $\epsilon = 0.02$ and 0.03. Also, we compared the impacts on robustness on different network architectures. As aforementioned about adversarial training [29], which concerns explicitly training a model on adversarial examples, in order to

make it more robust against attacks or to reduce its test error on clean inputs, training with adversarial examples is one of most powerful countermeasures to make networks more robust. It enhances robustness by generating adversarial examples at every step of the training stage and injecting them into the training set. In this way, adversarial training provides regularisation for DNNs and improves precisions through the process although it takes more training efforts compared with other methods. Conquering the training efforts during adversarial training, we have shown that the method here presented is more lightweight compared to adversarial training, while maintaining similar performance.

Though we evaluated robustness in different sizes of models, we do not compare under different model types, e.g., from ResNets to VGGNets. As we select model types following the principles of choosing the most accurate model for some specific dataset, we do not evaluate under various types of model architectures. The proposed approach was evaluated against small perturbations since pre-processing components like denoising elements are normally included in real applications. The results we obtained show that: i) that the method is computationally attractive compared to the present SoA, ii) differently from other approaches, it can defend against the FGSM and PGD attacks, and iii) it achieves global optima while maintaining robustness. From the benefits aforementioned, this method can be applied in more possibilities of exploration for robustness on different aspects of applications.

In summary, this work improves on the state of the art by providing a technique that is computationally more attractive, can defend against two types of state-of-the-art attacks and does not appear to be susceptible to local optima. Another stream of research focuses on robustness against black-box attacks and transferability [81,82]; these are not directly comparable, since the present paper focuses on white-box attacks. For a more detailed discussion about robustness against black-box attacks and the transferability phenomenon, we are going to explore and evaluate these faces in the next chapter.

Chapter 5

MRobust: Transferability for DNNs

Conventional machine learning models are known to be vulnerable to adversarial examples [38,18,36,14]. A gradient-based approach to generate adversarial examples for linear classifiers, support vector machines (SVM), and neural networks in the context of MNIST models, was first developed in [16]. This was then extended to proactive and reactive defences to improve the security of machine learning models [39]. It has also been observed that adversarial examples may have imperceptible differences compared to the original input [17]. [29] suggested that adversarial examples are inherently caused by the linear behaviour of deep neural networks (DNNs) when operating in high-dimensional spaces. The topology and geometry of adversarial examples were analysed in [83], while the local intrinsic dimensionality of adversarial regions for adversarial examples was characterised in [84]. More recently, it has been observed that adversarial examples are transferable, i.e. an example, which is adversarial for a DNN can often be used to mislead the prediction of other DNNs [81,17,85]. Moreover, adversarial examples could be universal in the sense that a single example may be used against several different models created from the same dataset [86].

In this chapter, we first develop a black-box adversarial attack method and then a black-box adversarial training algorithm MRobust to improve transferability and defend against state-of-the-art attack methods. Before diving into our black-box adversarial training algorithm, we first introduce some preliminaries about white-box and black-box

attack methods relating to transferability and different adversarial specificities. Given an arbitrary input to a DNN, the proposed attack method analyses small regions around the input that are likely to make significant contributions generating adversarial samples. Furthermore, the attack method does not require access to the internal layers of the model and thus falls in the realms of black-box attacks.

Moreover, we demonstrate the experimental results obtained on models with different sizes on MNIST and CIFAR10 datasets. The results suggest that known attacks on the resulting models are less transferable than those models trained by state-of-the-art attack algorithms, i.e. FGSM [29] and PGD [26]. From the comparisons, our results further show that the resulting DNNs synthesised via our method are less susceptible to transferability of attacks. We also show that the method reduces significantly the number of adversarial examples required for adversarial training.

The remainder of this chapter is organised as follows. As previously mentioned, Section 5.1 starts with introducing the transferability occurring between deep neural networks and Section 5.2 discusses different attack methods considering transferability and different adversarial specificities. Section 5.3 then describes the main black-box algorithm here proposed aimed at generating adversarial examples and how adversarial examples are employed for adversarial training. Lastly, Section 5.4 reports quantitative results on the MNIST and CIFAR10 datasets and a brief summary concludes the chapter.

The material presented in this chapter has appeared, in a shorter form, in the following paper [4]:

- Y. Liu and A. Lomuscio. A Method for Robustness against Adversarial Attacks on Deep Neural Networks. Proceedings of the 33rd International Joint Conference on Neural Networks and IEEE World Congress on Computational Intelligence (WCCI). Glasgow, United Kingdom. 1-8(2020). IEEE Press.

5.1 Defining Transferability

Many machine learning models have been shown to be vulnerable to adversarial examples, which are specially crafted to cause a machine learning model to produce an incorrect output. Adversarial examples that affect one model can often affect another model, even if the two models have different architectures or were trained on different training sets. This also happened in both models, which were trained to perform the same task. An attacker may therefore train their substitute model, craft adversarial examples against the substitute, and transfer them to a victim model, with very little information about the victim [81]. Recent work has further developed a technique that uses the victim model as an oracle to label a synthetic training set for the substitute, so the attacker does not need to know the input training information. In this section, we introduce the transferability methodology which transfers between previously unexplored (substitute, victim) pairs of machine learning model classes, called cross-technique transferability. We also introduce another transferability method that lies between different training datasets but the same training technique, named intra-technique transferability.

5.1.1 Cross-technique Transferability

Adversarial sample transferability has been found across the machine learning space and these samples can transfer well across models trained with different techniques or ensembles. For example, a support vector machine and decision tree respectively misclassify 91.43% and 87.42% of adversarial samples crafted for a logistic regression model [81]. Previous work on adversarial example transferability has primarily studied the case where at least one of the models involved in the transfer is a neural network [17,29,87]. However, the adversarial sample is capable to transfer in a more generally characterised manner between a diverse set of models chosen to capture most of the space of machine learning algorithms.

We discuss five machine learning techniques: Deep Neural Networks (DNNs), Linear Regression (LR), Support Vector Machines (SVMs), Decision Trees (DTs), and k-

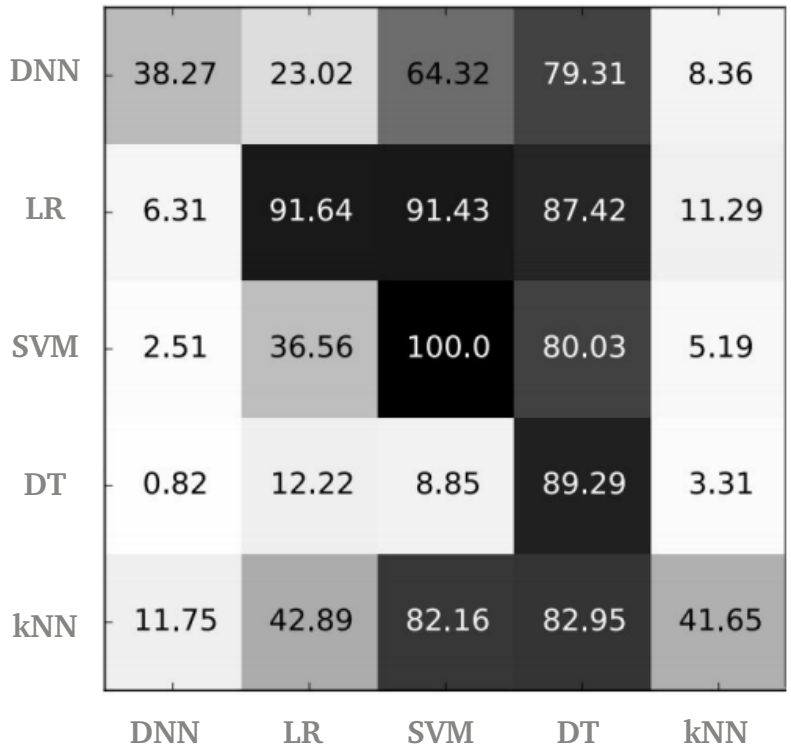


Fig. 5.1: The cross-technique transferability matrix: cell (i, j) is the percentage of adversarial samples crafted to mislead a classifier learned using machine learning technique i that are misclassified by a classifier trained with technique j [81].

Nearest Neighbours (kNNs). According to [81], DNNs were chosen for their state-of-the-art performance, LR for its simplicity, SVMs for their potential robustness stemming from the margin constraints when choosing decision boundaries at training, DTs for their non-differentiability, and kNNs for being lazy-classification models. Theano [88] and Lasagne [89] were used while training DNN, LR, and kNN models. In this study, the DNN is made up of a hierarchy of 2 convolutional layers of 32 3×3 kernels, 2 convolutional layers of 64 3×3 kernels, 2 rectified linear layers of 100 units, and a softmax layer of 10 units. It was trained during 10 epochs with learning, momentum, and dropout rates of respectively 10^{-2}, 0.9, and 0.5 decayed by 0.5 after 5 epochs. The LR was performed using a softmax regression on the inputs. It was trained during 15 epochs at a learning rate of 10^{-2} with a momentum rate of 0.9 both decayed by 0.5 after 10 epochs. The linear SVM and DT were trained with scikit-Learn.

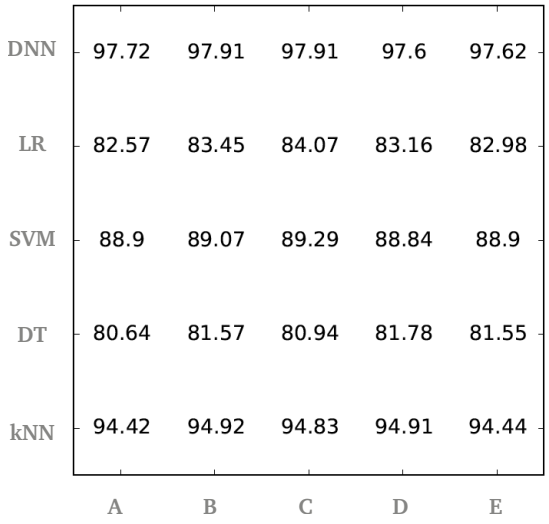
The cross-technique transferability is defined between models i and j, trained using different machine learning techniques, as the proportion of adversarial samples produced to be misclassified by model i that are also misclassified by model j. Hence, this is a more complex phenomenon than intra-technique transferability because it involves models learned using possibly very different techniques like DNNs and DTs. Yet, cross-technique transferability is surprisingly a strong phenomenon to which techniques like LR, SVM, DT, and ensembles are vulnerable, making it easy for adversaries to craft adversarial samples misclassified by models trained using diverse machine learning techniques.

From Fig. 5.1, a cross-technique transferability matrix is presented, where each cell (i, j) holds the percentage of adversarial samples produced for classifier i that are misclassified by classifier j. In other words, rows indicate the machine learning technique that trained the model against which adversarial samples were crafted. Columns indicate the underlying technique of the classifier making predictions on adversarial samples. This matrix shows that cross-technique transferability is a strong but heterogeneous phenomenon. The most vulnerable model is the decision tree (DT) with misclassification rates ranging from 79.31% to 89.29% while the most resilient is the deep neural network (DNN) with misclassification rates between 0.82% and 38.27%.

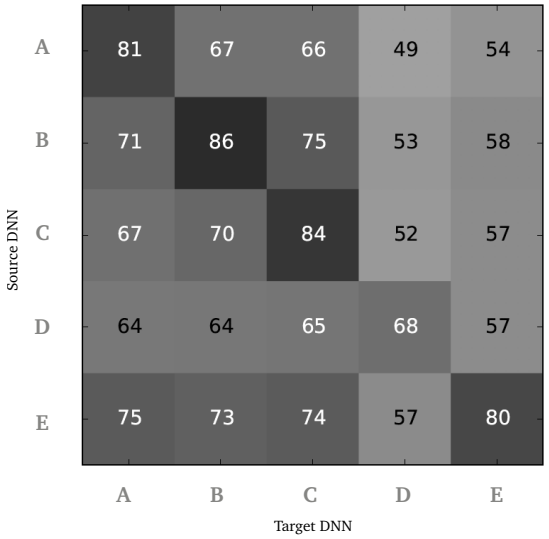
We conclude that all machine learning techniques studied are vulnerable to two types of adversarial sample transferability with different impacts. The most surprising results in adversarial samples being misclassified across multiple models learned with different machine learning techniques. This cross-technique transferability greatly reduces the minimum knowledge that adversaries must possess to force a machine learning model misclassifying inputs that they crafted.

5.1.2 Intra-technique Transferability

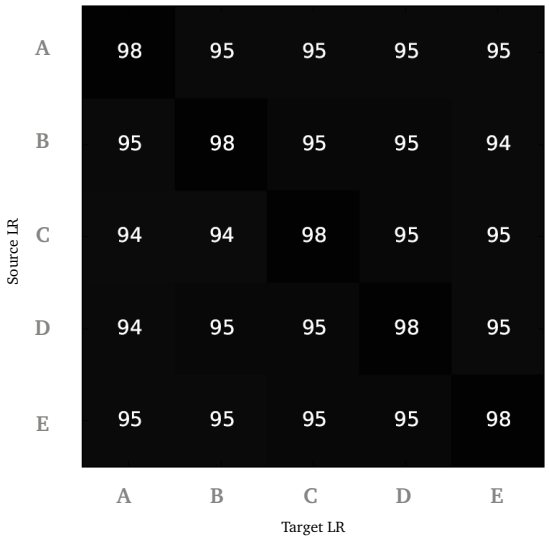
The intra-technique transferability is defined across models trained with the same machine learning technique but different parameter initialisations or datasets (e.g., f and f' are both neural networks or both decision trees). The differentiable models like DNNs and LR have been shown to be more vulnerable to intra-technique transferability



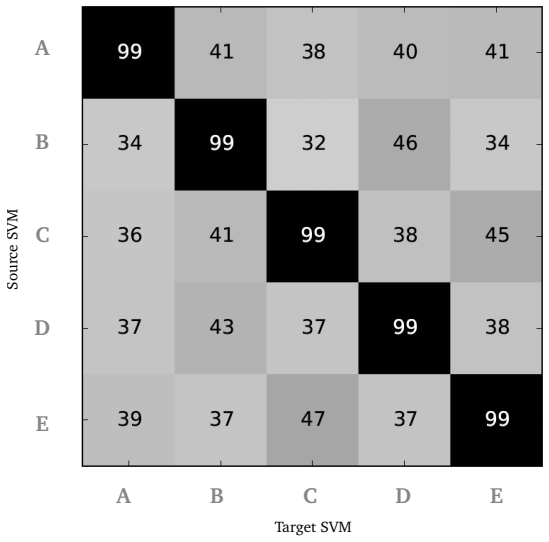
(a) Model Accuracies.



(b) DNN models.



(c) LR models.



(d) SVM models.

Fig. 5.2: Intra-technique transferability for different techniques. Fig. 5.2a reports the accuracy rates of the 25 models used, computed on the MNIST test set. Fig. 5.2b to Fig. 5.2d in cell (i, j) report the intra-technique transferability between models i and j using the same method with different parts of the dataset, i.e. the percentage of adversarial samples produced using model i misclassified by model j.

than non-differentiable models like SVMs, DTs, and kNNs according to [81]. The intra-technique transferability is measured between models i and j, both learned using the same machine learning technique. The measurement is based on the proportion of adversarial samples produced to be misclassified by model i that are misclassified by model j. To train different models using the same machine learning technique, the training set was split into disjoint subsets A,B,C,D,E of 10,000 samples each, in order of increasing indices. For each of the machine learning techniques (DNN, LR, SVM, DT, kNN), five different models were learned referred to as A,B,C,D,E. Model accuracies, i.e. the proportion of labels correctly predicted by the model for the testing data, are reported in Fig. 5.2a. For each of the 25 models, the suitable adversarial sample was crafted with suitable algorithms described previously from 10,000 samples in the test set, which was unused during training. The parameters of different algorithms were fine-tuned to achieve a quasi-complete misclassification of the 10,000 adversarial samples by the model on which they are crafted. Upon empirically exploring the input variation parameter space, the ϵ is set to 0.3 for the fast gradient sign method algorithm and $\epsilon = 1.5$ for the SVM algorithm.

Fig. 5.2b to Fig. 5.2d report intra-technique transferability rates for each of the DNN, LR, and SVM learning techniques. Rates (i, i) on the diagonals indicate the proportion of adversarial samples misclassified precisely by the same model i on which they were crafted. Off-diagonal rates (i, j) indicate the proportion of adversarial samples misclassified by a model j different from the model i on which they were crafted. From these figures, all models are vulnerable to intra-technique transferability in a non-negligible manner. LR models are most vulnerable as adversarial samples transfer across models at rates larger than 94%. DNN models display similarly important transferability, with rates of at least 49%. On the SVM matrix, the diagonal stands out more, indicating that this technique is to some extent more robust to the phenomenon. In the case of SVMs, this could be explained by the explicit constraint during training on the choice of hyperplane decision boundaries that maximise the margins (i.e. support vectors).

In conclusion, all models are found vulnerable to intra-technique adversarial sample transferability-misclassification of samples by different models trained using the same machine learning technique, the phenomenon is stronger for differentiable models like DNNs and LR than for non-differentiable models like SVMs.

5.2 White-box and Black-box Attack Methods

Adversarial attacks can be categorised based on the attacker's knowledge about the model (white-box and black-box attacks), the attack's specificity (targeted and non-targeted attacks), and the perturbation measurement (ℓ_∞-, ℓ_2-, ℓ_1-, and ℓ_0-norm attacks). White-box attacks have full knowledge of the neural network model, including the training data, the architecture, the weights, and the hyperparameters of the model, whilst black-box attacks have access only to the output of the model, e.g., the final decision or the score.

Targeted attacks aim to produce a targeted misclassification, whereas the adversarial label for an un-targeted attack can be arbitrary except the original one. A targeted attack searches for the perturbation that changes the network prediction to the specific target $f(x + \delta) = y_t$. Targeted attacks usually occur in the multi-class classification problem. For example, an adversary fools an image classifier to predict all adversarial examples as one class. In a face recognition/biometric system, an adversary tries to disguise a face as an authorised user (Impersonation) [47]. Targeted attacks usually can be realised by maximising the probability of the targeted adversarial class.

In contrast to targeted attacks, non-targeted attacks do not assign a specific class to the neural network output. The adversarial class of output can be arbitrary except the original one, which is $f(x + \delta) \neq f(x)$. For example, an adversary makes his/her face misidentified as an arbitrary face in a face recognition system to evade detection (dodging) [47]. Non-targeted attacks are easier to implement compared to targeted attacks since it has more options and space to redirect the output. Non-targeted adversarial examples are usually generated in two ways: 1) running several targeted attacks and taking the one with the smallest perturbation from the results; 2) minimising the probability of the correct class. Some generation approaches (e.g., extended BIM, ZOO) can be applied to both targeted and non-targeted attacks. For binary classification, targeted attacks are equivalent to non-targeted attacks.

In the previous section, we introduced the transferability among different machine learning models and discussed robustness in these models. This section will review the

landscape of the research on different adversarial attacks, especially for deep neural networks. A myriad of attacks on DNNs has been proposed since the discovery of adversarial examples and has been categorised into two genres, white-box and black-box attacks. Intuitively, a white-box adversary should always be stronger than any black-box adversary because it has complete information about the attack's target model. However, in a special case, when the defence obfuscates or shatters the gradients, white-box adversaries tend to overestimate model robustness [90]. Athalye et al. [91] suggested evaluating defence on white-box and black-box adversaries. If the model is more robust to white-box adversaries, then the model might mask the gradients. In the following subsections, we review some white-box and black-box adversarial attacks. A more in-depth review and comparison of various attack methods can be found in [92,93].

5.2.1 White-box Attack Model

White-box attacks assume the adversary knows everything about the trained neural network models, including training data, model architectures, hyper-parameters, numbers of layers, activation functions, model weights. Realising this information, the adversary can generate adversarial examples via calculating model gradients easily. We discussed some attack methods in Section 2.3, including the followings:

- **Fast Gradient Sign Method (FGSM)**: This method performs one step gradient update along the direction of the sign of gradient at each pixel.
- **Projected Gradient Descent (PGD)**: This method adopts the multi-step variant of FGSM.
- **Jacobian-based Saliency Map Attack (JSMA)**: This method uses the saliency map based on forward derivative and provides the adversary with the required information to cause the neural network misclassifying a given sample.
- **Deepfool Attack**: This method finds the closest class boundary and takes a step in the direction of the closest decision boundary.

- **Carlini & Wagner Attack**: Carlini and Wagner consider a wide variety of formulations with a set of techniques to improve the speed and accuracy of the gradient-based attack. They summarise from one of them that performs best according to their evaluation.

5.2.2 Black-box Attack Model

Black-box attacks assume the adversary has no access to the trained neural network model. The adversary, acting as a standard user, only knows the output of the model (label or confidence score). This assumption is common for attacking online Machine Learning services (e.g., Machine Learning on AWS1, Google Cloud AI2, BigML3, Clarifai4, Microsoft Azure5, IBM, Bluemix6, and Face++7). Most adversarial methods are white-box attacks. However, they can be transferred to attack black-box services due to the transferability of adversarial examples which we discussed in Section 5.1.

Attacks under the black-box threat model are more difficult to perform than white-box settings because the gradients of the model are not known. Black-box, or gradient-free attacks, could be divided into two categories: gradient estimation and transfer-based attacks. In gradient estimation, the gradients of the black-box model are estimated, which may require many queries for accurate approximation [94,95]. Transfer-based attacks [90] rely on the fact that adversarial examples transfer between models. However, the efficiency of the transfer-based attacks largely depends on the quality of the substitute network.

In [90], a practical black-box adversarial attack was introduced based on the property that adversarial examples often transfer between models. A substitute model was first trained based on the model's task. Then, adversarial examples generated for the substitute model can be used to attack the target model. Brendel et al. [96] introduced a decision-based attack which estimates the decision boundary using rejection sampling. Starting at some adversarial image, they randomly draw a random perturbation from the candidate distribution and minimise the distance to the original image. In [94], a random perturbation was sampled from an orthonormal basis of discrete cosine transform (DCT), which significantly improves query-efficiency of the decision-based attack. Gradient-based

attacks should be almost always stronger than gradient-free attacks. However, gradient masking [90] can fool gradient-based attacks and give a false sense of security [91]. If the defence obfuscates the gradients, gradient-free attacks often perform better than white-box attacks. In [48], it was suggested that defences should be tested on both white-box and black-box adversaries. If the model is more robust to white-box adversaries, then the model probably masks the gradients.

Some defence methods have been explored to enhance robustness against the black-box attacks. DeepDGA [97] used generative adversarial network (GAN) to generate adversarial domain names to evade detection of domain generation algorithms. MalGan also adopted a GAN-based algorithm to generate malware examples and evade black-box detection [98].They used a substitute detector to simulate the real detector and leveraged the transferability of adversarial examples to attack the real detector. MalGan was evaluated by 180K programs with API features. However, it required the knowledge of features used in the model. Another line of works used a large number of features (2,350) to cover the required feature space of portable executable (PE) files [99]. The features included PE header metadata, section metadata, and import & export table metadata. They also defined several modifications to generate malware evading deep learning detection. The solution was trained by reinforcement learning, where the evasion rate is considered as a reward.

5.3 The MRobust Defence Method

After some introductions about the transferability techniques and different attack methods, in this section, we present a black-box adversarial attack method relying on the Scale Invariant Feature Transform (SIFT) algorithm [76] aforementioned in Chapter 3. This method is categorised into the genre of the non-targeted attacks. We present how this algorithm uses Monte-Carlo tree search (MCTS) [1] to generate effective adversarial examples in Subsection 5.3.1. In Subsection 5.3.2 we summarise the full black-box adversarial training algorithm, called MRobust, including a robust optimisation function.

Algorithm 5: Black-box Adversarial Attack: MAttack

1 **function** Black-boxAdversarialAttack ;

Input : Clean image dataset x

Initialise perturbation constraint setting for ϵ

Initialise search trees T

Output: Effective adversarial examples x'

2 Execute Scale Invariant Feature Transform to obtain potential candidate descriptor;

3 **while** *effective adversarial example not found* **do**

4 Pick up one most promising point as root node N_R from the descriptor;

5 Trace from the root node;

6 Selection: select nodes with greatest confidence value and trace till the leaf node;

7 Expansion: expand one node from leaf node;

8 Simulation: do simulation with the expanded node and check win or loss;

9 Backpropagation: update associated information for each node along the traversing path back to the root node

10 **end**

5.3.1 Black-box Adversarial Attack Method

In this subsection we introduce Scale Invariant Feature Transform [76] to search potential invariant feature candidates and Monte Carlo Tree Search [1] to find an adversarial example based on the outcomes of Scale Invariant Feature Transform. We summarise the full algorithm in Algorithm 5.

Scale Invariant Feature Transform [76]. The algorithm begins by executing SIFT [76] on the clean image, given a perturbation ϵ, received as input (line 2). SIFT was first proposed to extract image features in object recognition systems [76]. It can be employed to extract features for any object in an image and provide a feature description of this object. This description can then be used to identify and locate objects in other images. In order to perform accurate recognition, these extracted features should be detectable under variations of image scale, orientation and illumination, and these extracted features are also invariant to image scaling, translation and rotation.

For the purpose of searching for features, Laplacian of Gaussian (LoG) are normally used to detect areas of rapid changes (edges) in images. Due to the significant computation cost of LoG operations, the Difference of Gaussians (DoG) is instead employed for simplification. Once the DoG is obtained, each pixel in the DoG is compared with its eight neighbours at the same scale and the process is then repeated at different scales. If a local maximum is identified, this is considered a potential invariant feature. The points that are low-contrast and those that are edge response points are discarded. By discarding these points we are left with potential candidates for feature invariance over the parameters above. These points are collected into a descriptor also containing the location and orientation of these points.

Monte Carlo Tree Search (MCTS) [1]. We now explain how Monte Carlo Tree Search [1] can be employed to identify potential adversary symbols by searching in the neighbourhood of the candidates obtained by SIFT as above in Algorithm 5 (line 3-10). MCTS is a heuristic search algorithm for decision making; its key feature consists of selecting the most promising moves on the basis of random sampling of the search space. In each iteration, MCTS consists of four steps, that are selection (line 6), expansion (line 7), simulation (line 8) and backpropagation (line 9). We consider each node of the tree as a specific pixel in an image. We first traverse from the root node N_R and choose a child node N_C with the highest confidence value down to a leaf node N_L. Then we expand one or more children nodes N_E and simulate from one of them to get a win or loss. A win of the game represents the fact that an adversarial example can be found once a perturbation is applied to this pixel. In the last step, we update the tree structure with the new confidence information for each node according to the simulation result whether the search was a win or a loss.

We use the confidence value C_i, representing the associated information in each node N_C of the game tree, given by Equation (3.5) aforementioned.

Intuitively the higher probability to derive an adversarial symbol have the higher confidence value for each node. On the contrary, the lower probability to obtain an adversarial one will have a lower confidence value. The confidence measure above is inspired by the game of Go [78] where is widely used to make decisions in different games

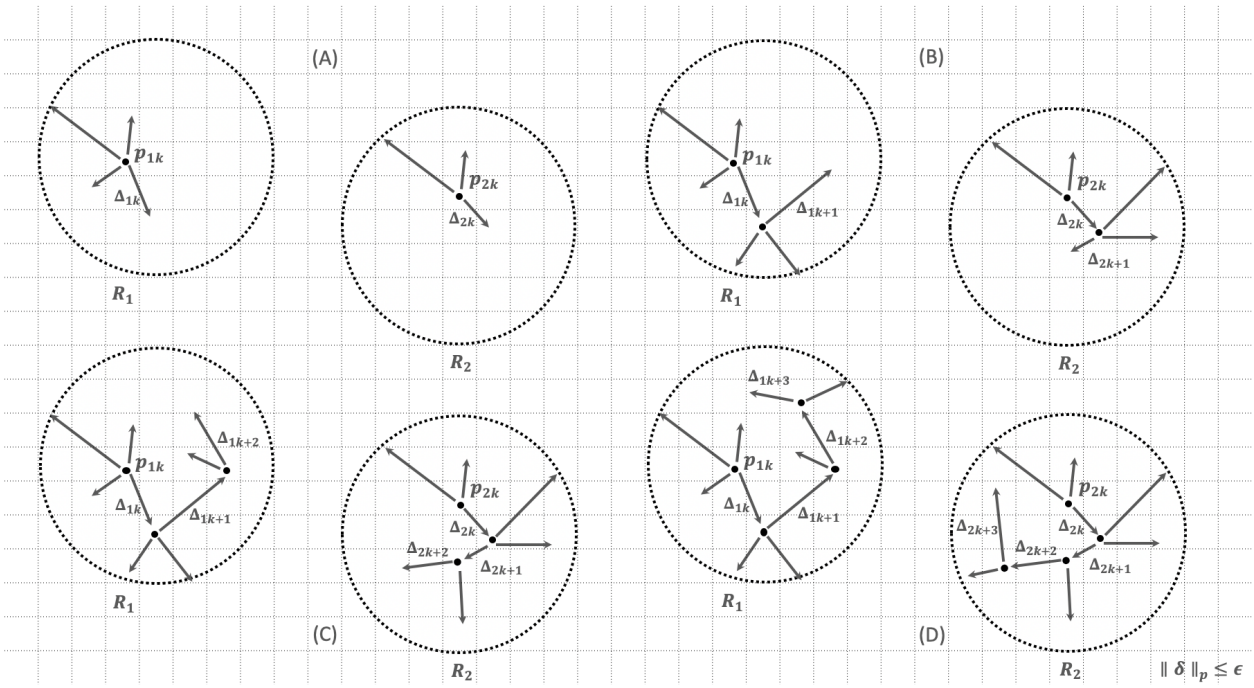


Fig. 5.3: An Effective Adversarial example in a region R_i with $\|\delta\|_p \leq \epsilon$.

and applications for deep neural networks.

Given the above, we select one effective adversary symbol from the candidates above by imposing the constraint $\|\delta\|_p \leq \epsilon$. For each region R_i in an image, each path of the game tree can be considered as a ladder-like structure. Each region R_i contains several nodes $N_{C_{ik}}$ of the game tree and a specific perturbation Δ_{ik} is applied after a node $N_{C_{ik}}$ is selected. The total perturbations applied to a region is then presented as $\sum_{k=0}^{K} \Delta_{ik}$, where K is the total number of nodes along a path. An effective adversarial example $\mathcal{E}_A$ with $\|\delta\|_p \leq \epsilon$ is then formulated as Equation (5.1):

$$\mathcal{E}_A = \| \sum_{i=0}^{R} \sum_{k=0}^{K} \Delta_{ik} \|_p \leq \epsilon, \tag{5.1}$$

where R is the total number of regions in an image.

We explain more details in Fig. 5.3 (A-D). In this figure, there are two regions $R_{i=1..2}$ and each region R_i contains several nodes $N_{C_{ik}}$ of the game tree. For region R_1, a specific perturbation Δ_{1k} is applied after one node $N_{C_{1k}}$ is selected. The total perturbations applied in a region R_1 is then presented as $\sum_{k=0}^{K} \Delta_{1k}$. An effective adversarial example

Algorithm 6: Black-box Adversarial Training: MRobust

Deep neural network M

Size of the training minibatch is m

function Black-boxAdversarialTraining ;

Input : Deep neural network M

Training dataset x

Size of the training minibatch is m

Output: Deep neural network MRobust

Adversarial training accuracy and loss values

Randomly initialize deep neural network M

while *training not converged* **do**

  Read minibatch $B = \{x_1, ..., x_m\}$ from training dataset;

  Generate m adversarial examples $B_{adv} = \{x'_1, ..., x'_m\}$ from corresponding clean examples $\{x_1, ..., x_m\}$ with black-box adversarial attack method in Algorithm 1:

  function Black-boxAdversarialAttack;

  Do one training step of network M using minibatch B';

  Update model loss with robust optimisation: $\min_\theta \mathcal{L} = \min_\theta \sum_{i=1}^{m} \max_\delta \mathcal{L}(x'_i, \theta, y_i)$;

end

$\mathcal{E}_A$ with constraint $\|\delta\|_p \leq \epsilon$ is obtained once all perturbations in these two regions satisfied with the constraint.

5.3.2 Black-box Adversarial Training Algorithm: MRobust

In the previous subsection, we obtained effective adversarial examples by using the black-box adversarial attack method. In this subsection, we introduce a black-box adversarial training algorithm relying on these effective adversarial examples. We summarise the full algorithm in Algorithm 6, including the robust optimisation step.

We now explain Algorithm 6. We first randomly initialised the neural network M. The loss value $\mathcal{L}$ of the neural network M is updated according to the softmax result of each adversarial training iteration. The training loop continues for as many epochs as required until the required accuracy is converged. During each training loop, we randomly

select a minibatch B of size m from the training dataset and generate corresponding minibatch B_{adv} consisted of size m adversarial examples using the black-box adversarial attack method. The minibatch B_{adv} is then applied to the robust model M for adversarial training. The convergence criteria ensure that the resulting model M is robust in small neighbourhoods of every training point around x. We call these neighbourhoods the perturbations δ and we represent them as $x' = x+\delta$. The overall process can be regarded as a solution to the robust optimisation problem against adversarial examples and formulated as:

$$\min_{\theta} \mathcal{L} = \min_{\theta} \sum_{i=1}^{m} \max_{\|\delta\| \leq \epsilon} \mathcal{L}(x'_i, \theta, y_i), \tag{5.2}$$

where δ is the perturbation set under the constraint ϵ corresponding to the adversarial example x'_i. This involves optimising the model parameter θ with respect to a worst-case data (x'_i, y_i), rather than against the original training data, which is related to the black-box attack method previously; the i-th worst-case data point is selected from the perturbation set δ.

5.4 Experimental Results

In this section, we evaluate the robustness of the black-box adversarial training algorithm (MRobust) presented in the previous section and report the results obtained on the MNIST [74] and CIFAR-10 [75] datasets. We evaluate the transferability between different adversarial training models using FGSM, PGD, and MRobust methods. The evaluation basis for transferability is defined in Subsection 5.4.1 and the experimental setup and results are shown in Subsections 5.4.2, 5.4.3 and 5.4.4.

5.4.1 Robustness Transferability

Attacks transferability is problematic in applications [81] as attacks identified in one domain may be easily transferable to another. Transferability can be analysed in terms of intra-technique and cross-technique transferability. Intra-technique transferability con-

Target \ Source	S. (Nat. Train)	S. (FGSM Train)	S. (PGD Train)	S. (M Train)	W. (Nat. Train)	W. (FGSM Train)	W. (PGD Train)	W. (M Train)
S. (Nature Train)	12.3	85.6	85.8	86.2	6.4	78.4	86.6	87.3
S. (FGSM Train)	78.3	64.3	68.4	74.7	76.4	64.8	76.2	82.1
S. (PGD Train)	80.2	78.4	81.4	80.1	79.2	76.5	80.8	82.7
S. (M Train)	83.8	84.2	84.7	74.9	83.2	84.5	86.8	79.7
W. (Nature Train)	15.7	89.7	88.6	90.1	5.2	77.2	85.3	90.7
W. (FGSM Train)	79.2	72.7	79.6	82.6	70.5	64.2	81.2	84.8
W. (PGD Train)	81.4	80.1	82.7	83.5	81.4	79.5	82.4	82.9
W. (M Train)	85.5	86.3	86.2	78.6	85.1	85.7	88.4	80.6

Table 5.1: The robustness transferability comparison of nature training, FGSM, PGD and MRobust methods using black-box adversarial attack from the source network on MNIST.

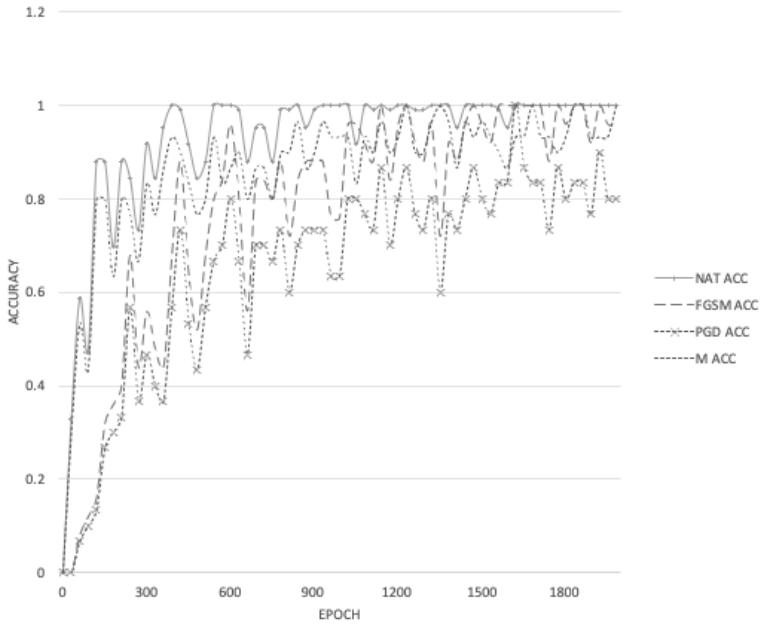
cerns the misclassifications (caused by a set of attacks) on different models trained on the same learning method. Cross-technique transferability concerns misclassifications (caused by a set of attacks) on models trained on different learning methods. Here we focus on cross-technique transferability against attacks on models built with FGSM [29], PGD [26], and MRobust methods. Specifically, we use the black box adversarial attack method of the previous section as an adversary and evaluate the robustness of different adversarially trained models against this adversary. We also study the robustness of different model architectures and evaluate how the capacity of the network impacts transferability.

In the following, we focus on robustness transferability (RT), defined as $RT = 1 - AttackSuccessRate$, which measures the percentage of adversarial samples produced using the black-box attack adversary that do not cause a misclassification on the trained model. In other words, a higher robustness transferability represents a situation in which the trained model is more robust under the attack of the black-box attack adversary.

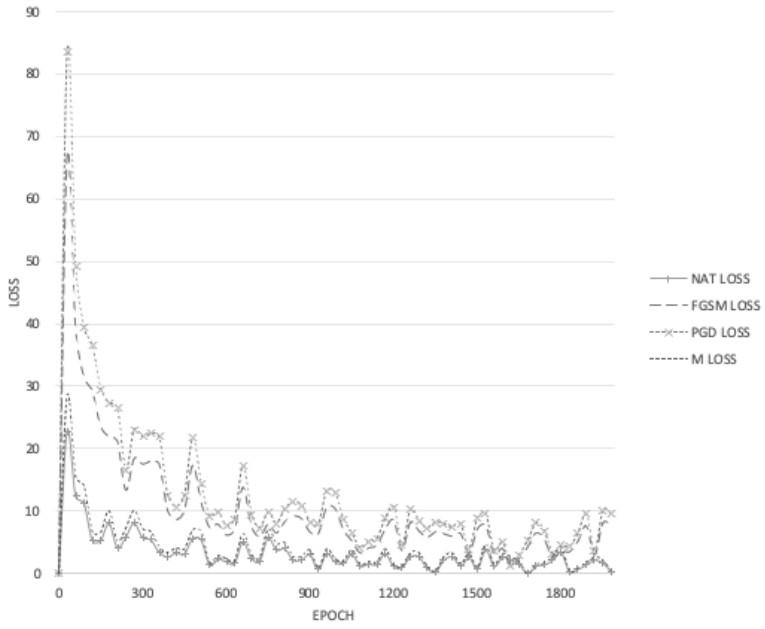
5.4.2 Experimental Setup

We evaluated the method on the MNIST and CIFAR10 datasets. The MNIST database of handwritten digits contains a training set of 60,000 examples and a test set of 10,000 examples aforementioned. The digits were size-normalised and centred in a fixed-size image of 28×28. We generated adversarial examples under the perturbation constraints of size $\epsilon = 0.1$ in the l_∞ norm. To investigate model capacity, we considered two training networks of simple and wide architectures, respectively. The simple network consisted of two convolution layers of sizes 32 and 64 filters and a fully connected layer of size 1024. The wide network consisted of two convolution layers of sizes 64 and 128 filters, and also a fully connected layer of size 1024. Both networks were adversarially trained with FGSM, PGD and MRobust methods. The robustness transferability for black-box attack adversary between different adversarial trained methods and architectures are shown in Table 5.1. More explanations about the experimental results are described in Subsection 5.4.3.

The CIFAR10 dataset contains a training set of 50,000 examples, and a test set of 10,000 examples of 32×32 colour images in 10 different classes aforementioned. The value of each pixel in the input image was normalised in the interval $[0, 1]$. As before, we generated adversarial examples under the perturbation constraints of size $\epsilon = 0.1$ in the l_∞ norm. For the CIFAR10 dataset, we used Resnet model [55] as the baseline model and constructed a variant with layers wider by a factor of 10, resulting in a wide network with 5 residual units with (16, 160, 320, 640) filters each. This network achieved up to 95.2% accuracy on the clean test dataset. We also performed adversarial training with FGSM, PGD and MRobust methods on these two networks and investigated the robustness transferability for black-box attack adversary between different adversarially trained methods. The resulting robustness transferability measures are shown in Table 5.2. We will explain more experimental results in Subsection 5.4.4.



(a) Adversarial Accuracy on MNIST dataset with $\epsilon = 0.1$.



(b) Adversarial Loss on MNIST dataset with $\epsilon = 0.1$.

Fig. 5.4: Accuracy and loss comparisons on MNIST dataset with $\epsilon = 0.1$ in the l_∞ norm.

5.4.3 MNIST

Table 5.1 summarises the results for robustness transferability obtained for the MNIST dataset. We first trained DNNs with FGSM, PGD and MRobust methods as the source and target models and generated attacks with a MAttack adversary based on the query result from each of the trained source models. The generated attack was then transferred into each of the target trained models and a check was carried to determine whether the target trained models can successfully defend against this attack. The results obtained show that a strong adversary generally reduces transferability and increases robustness transferability. For example, the robustness transferability with source simple trained model S. (M Train) paired with target S. (FGSM Train) is 74.7; this is higher than 68.4, which is the value for the same target but source S. (PGD Train). In addition, the MRobust method generally augments robustness transferability for different models, except for those that have MRobust itself as a source. For instance, the robustness transferability with source simple trained model S. (M Train) paired with target W. (PGD Train) is 83.5; this is higher than 82.7, which is the value for the same target but source S. (PGD Train). This happens in most cases. On the contrary, the robustness transferability with source simple trained model S. (M Train) paired with target W. (M Train) itself is 78.6; this is lower than 86.2, which is the value for the same target but source S. (PGD Train). The reason is that we generate attacks based on the source model using the similar method. Moreover, changing the architecture from simple to wide networks generally has a positive effect on robustness transferability; so the value with source wide model W. (M Train) paired with target W. (M Train) is higher than source S. (M Train) paired with target W. (M Train).

We report the accuracy and loss for different adversarial training methods over the first 2,000 epochs in Figure 5.4a and Figure 5.4b. The results show that the convergence is highest for nature training with no adversarial examples, followed by MRobust, then PGD, and finally FGSM. The results show that MRobust converges faster than the competing methods with 13.2 % because it searches for only potential candidates of the perturbations for adversarial examples. This can also help to reduce the required adversarial examples for adversarial training and thus save the training efforts. In summary,

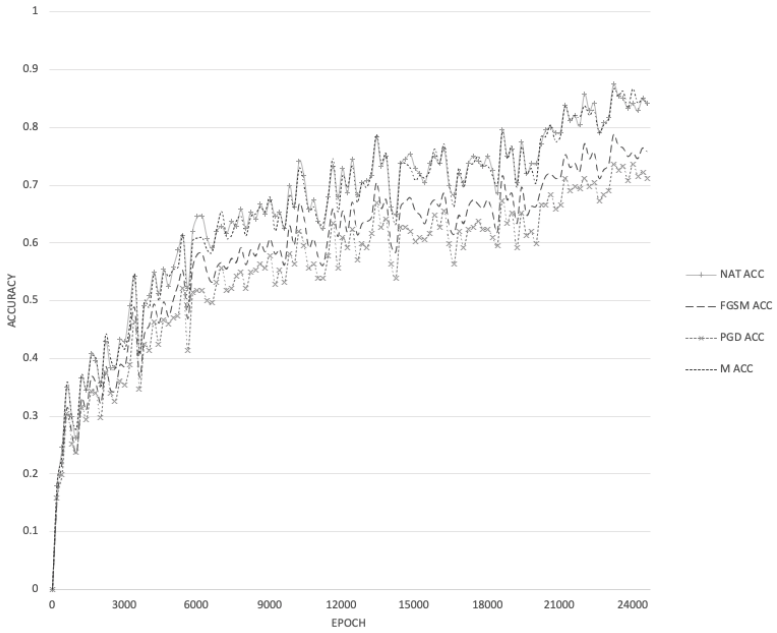
Target \ Source	S. (Nat. Train)	S. (FGSM Train)	S. (PGD Train)	S. (M Train)	W. (Nat. Train)	W. (FGSM Train)	W. (PGD Train)	W. (M Train)
S. (Nature Train)	9.1	75.4	76.4	76.9	3.4	71.9	77.6	78.3
S. (FGSM Train)	68.3	53.4	58.2	64.9	66.1	55.5	66.9	72.5
S. (PGD Train)	69.6	68.1	70.7	69.4	69.3	65.9	70.1	73.2
S. (M Train)	73.3	74.1	75.2	64.5	72.1	74.9	76.4	69.3
W. (Nature Train)	10.3	79.1	78.3	79.6	8.3	67.6	75.9	79.8
W. (FGSM Train)	69.1	63.2	69.7	71.5	69.4	54.9	70.1	74.6
W. (PGD Train)	70.6	69.8	71.3	72.9	70.8	69.2	71.9	72.8
W. (M Train)	75.7	77.1	76.8	69.4	75.2	76.6	79.2	68.3

Table 5.2: The robustness transferability comparison between nature training, FGSM, PGD and MRobust methods using black-box adversarial attack from the source network on CIFAR10.

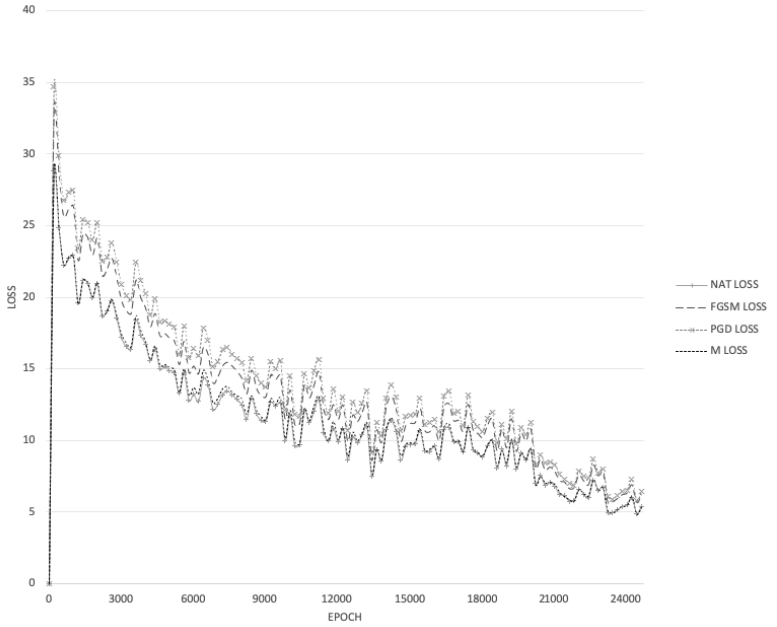
the adversarial training with MRobust converges faster than the other two methods.

5.4.4 CIFAR10

Table 5.2 summarises the robustness transferability of the CIFAR10 dataset. As above the results show that a strong adversary reduces transferability and helps the robustness transferability. For instance, the robustness transferability with source simple trained model S. (M Train) paired with target W. (FGSM Train) is 71.5; this is higher than 69.7, which is the value for the same target but source S. (PGD Train). Furthermore, the MRobust contributes to robustness transferability for different models, except for those with a source MRobust themselves; see, e.g., source S. (M Train)/target W. (M Train) is lower than source S. (PGD Train)/target W. (M Train), whilst the value of source S. (M Train)/target W. (PGD Train) is higher than source S. (PGD Train)/target W. (PGD Train). Moreover, changing the architecture from simple to wide networks improves robustness transferability on average, e.g., source W. (M Train)/target W. (M Train) is higher than source S. (M Train)/target W. (M Train).



(a) Adversarial Accuracy on CIFAR10 dataset with $\epsilon = 0.1$.



(b) Adversarial Loss on CIFAR10 dataset with $\epsilon = 0.1$.

Fig. 5.5: Accuracy and loss comparisons on CIFAR10 dataset with $\epsilon = 0.1$ in the l_∞ norm.

We plot the accuracy and loss for different adversarial training methods over the first 25,000 epochs in Figure 5.5a and Figure 5.5b. As above we found that the nature training method converged faster than all, followed by MRobust, PGD and FGSM. The results also show that MRobust requires fewer adversarial examples during adversarial training and thus improves the training efforts by 5.2%. As in the MNIST case, we can see from Figure 5.5a and Figure 5.5b, the adversarial training with MRobust converges more efficient than the other two methods.

5.5 Summary

In this chapter, we first introduced some preliminaries about white-box and black-box attack methods relating to transferability and different adversarial specificities before diving into our black-box adversarial training algorithm. We then developed a black-box adversarial attack method and a black-box adversarial training algorithm MRobust to improve transferability and defend against state-of-the-art attack methods. The attack method does not require access to the internal layers of the model and is therefore applicable to applications such as security.

Moreover, we demonstrated the experimental results obtained from models with different sizes on MNIST and CIFAR10 datasets. The results suggested that known attacks on the resulting models are less transferable than those models trained by state-of-the-art attack algorithms, i.e. FGSM [29] and PGD [26]. From the comparisons, our results further showed that the resulting DNNs synthesised via our method are less susceptible to the transferability of attacks. We also showed that the method reduces significantly the number of adversarial examples required for adversarial training.

In summary, we proposed the MRobust defence method with MCTS and evaluated the robustness transferability results on MNIST and CIFAR10 datasets in this work. We focused on small perturbations from potential candidates that are capable to generate adversarial examples as this can save time complexity for adversarial training and increase the robustness against adversarial attacks. We used the black-box adversarial training framework to learn robustness under different constraint metrics. We do not explore

different noise models, e.g., image rotations or different light conditions as we emphasise non-augmented data here.

In real applications some pre-processing components like denoising elements are normally included prior to neural network models; so we here only focus on potential perturbations. Summarising for this chapter, the results show: i) that the method is computationally attractive as we adversarially train potential perturbations, and save more computational efforts, ii) it can defend against the FGSM and PGD attacks and sustain a competitive robustness transferability, and iii) does not appear to be susceptible to local optima compared with the SoA. In future work, we will continue to improve robustness and computational efforts applying on different datasets against different attacks.

Chapter 6

Conclusions

In this chapter, we assess the advancements of this thesis and draw systematic views of related work. We conclude with possible future directions. Section 6.1 summarises the main contributions of each chapter. Then, Section 6.2 evaluates these via drawing some systematic views of related work compared with those presented in Chapter 2. In light of this analysis, Section 6.3 underlines the main overall achievements of the thesis. Section 6.5 finalises with possible future directions and further developments of this line of work.

6.1 Summary of Thesis Achievements

In Chapter 3, we presented an adversarial training algorithm based on Monte Carlo Tree Search. We illustrated the robustness of the algorithm by studying its resistance to adversarial examples in the context of the MNIST and CIFAR10 datasets. For MNIST, after 2000 epochs the experimental results showed an average improvement of efficiency of 21.1% when compared to PGD. For CIFAR10, after 7000 epochs we obtained an average improvement of efficiency of 9.8% compared to PGD. We further compared the robustness of the algorithm against previous work against various attack methods. The results showed that the adversarial training method introduced is not only robust with respect

to adversarial examples but also efficient during training.

In Chapter 4, we presented a novel method for robustness training for ReLU-based deep neural networks. The method involved decision tree search targeting the worst-case data points to generate adversarial examples. We combined the decision tree search method with robust optimisation to train a robust model while maintaining accuracy at comparably lower computational effort than the state-of-the-art methods. The efficiency was obtained by focusing on small regions centred around the input that have significant potential to generate adversarial samples. We implemented the resulting method in the toolkit DTSRobust, which was evaluated against state-of-the-art defence methods on MNIST and CIFAR10 datasets. In experiments DTSRobust achieved a 14.2% gain on efficiency against the state-of-the-art defence methods in MNIST and 10.3% of that in CIFAR10 while maintaining similar accuracy.

Chapter 5 presented a novel black-box adversarial training algorithm to defend against the state-of-the-art attack methods in machine learning. In order to search for an adversarial attack, the algorithm analysed small regions around the input that are likely to make significant contributions to the generation of adversarial samples. Unlike some of the literature in the area, the proposed method does not require access to the internal layers of the model and is therefore applicable to applications for security, e.g., obfuscating malware code within network packets or misleading signature detection; attacks in biometric recognition where fake biometric traits may be exploited to impersonate a legitimate user. We reported the experimental results obtained on models of different sizes built for the MNIST and CIFAR10 datasets. The results demonstrated that known attacks on the resulting models are less transferable than those models trained by the state-of-the-art attack algorithms.

6.2 Comparisons in Related Work

In this section, we draw a comparison between the results in this thesis and the existing state-of-the-art methods. See also the discussion at the end of each chapter. This section is systematisation and an extension of those presentations. The motivations and

contributions of this thesis give the main criteria to be considered in this comparison. These criteria are categorised as follows: 1) perturbation-based adversarial robustness; 2) a broader view of robustness in DNNs; 3) robustness in generative models; 4) equivariance and invariance to noises in computer vision.

6.2.1 Perturbation-based Adversarial Robustness

A rapidly growing body of work has addressed the adversarial robustness of deep neural networks with respect to small norm-bounded perturbations. This problem has motivated various related researches about adversarial attacks and defences within the scope of norm-bounded adversaries [91,100]. While some defences have resisted against a variety of strong adversaries [26], it remains an open question as to how well to defend against such attacks.

Several notable works formulate adversarial training algorithms which proposed methods to defend against adversarial attacks, the goal of which is to defend neural networks against worst-case perturbations [29,36,101]. Some of the most successful works take a robust optimisation perspective, in which the goal is to find the worst-case adversarial perturbations of data by solving a min-max problem [26,102]. In a different yet related line of work, optimisation-based methods have been proposed to provide certifiable guarantees on the robustness of neural networks against small perturbations [103,104,105]. Another line of researches has studied how adapting network architectures can be used to defend against adversarial examples [106,62].

As adversarial training methods have become more sophisticated, computational efficiency is regarded as a criterion during the training process. Our objective was to provide an efficient training algorithm that is not only robust against different norm-bounded perturbations but also generates effective adversarial examples which are perceptually similar to a given input image such as [43,107]. Based on these purposes, we proposed an MCTS-based Robustness Method in Chapter 3 and a Decision Tree Search Robustness Method in Chapter 4. Both of these methods provide generalisations for the robustness of DNNs while maintaining competitive computation efforts from the results. Besides, these

methods also provide adversarial examples only considering potential features which are possible to generate successful attacks. Benefits from these advances, our methods are efficient during adversarial training while retaining robustness against a variety of attacks, e.g., the results from Section 3.4 and 4.3.

6.2.2 A Broader View of Robustness in DNNs

From a broader view of robustness in DNNs, defence methods can be classified into two variants. The first variant can be seen as reactive defence methods, which defend after the attacks are generated and attempt to detect adversarial examples from inputs after DNNs are built. Adversarial detecting [27] and input reconstruction [20] fall in the realm of reactive ones. The second variant can be seen as proactive defence methods, which defend before adversaries generate adversarial examples. The representative researches of proactive defence methods are network distillation [21] and adversarial re-training [28]). While these defence methods have helped to motivate new notions of robustness to some extent, these approaches proposed defences against the nuisances are limited in the sense that they do not generalise well to a learning paradigm. For the adversarial detecting methods, Carlini et al. demonstrated that many adversarial detecting methods [27,57,58,59,60,61] cannot defend against the C&W attack [42] to varying degrees. As for the input reconstruction [20], calculating partial derivatives at each layer in the back-propagation framework becomes computationally expensive. Also, the proposed layered based approach does not guarantee global optimality. Speaking of the proactive defence methods such as network distillation [21], it has been shown failing to defend against an attack developed in [42]. Although the adversarial re-training [28] can defend against most series of attacks, the computational efforts are regularly concerned.

The works discussed above do not provide comprehensive defences under different attack methods on the whole. Our method improves on the state of the art by providing a defence method that is comparably computationally attractive, can defend against the C&W attack and does not appear to be susceptible to local optima from the results.

More recently, a different line of work has considered the robustness of deep neural

networks against transformations that are more likely to be encountered in applications. Nuisances that have recently received attention from the adversarial robustness community include adversarial quilting [67], adversarial patches and clothing [108], geometric transformations [109,110,111], distortions [52], deformations and occlusions [112], and nuisances encountered by unmanned aerial vehicles [113]. While these progresses have helped to motivate new notions of robustness, the approaches that propose defences against these nuisances are limited in the sense that they do not generalise to a learning model which could be applied across different forms of natural variation. In response to these works and motivated by myriad safety-critical applications, a defence method that is plausible to be leveraged for the robustness of different applications is considered. From the motivation behind our method, which can provide general robust training algorithms on deep neural networks across a variety of scenarios and applications.

6.2.3 Robustness in Generative Models

Another line of work that focuses on attack and defence strategies against adversarial examples use generative models in the loop of training. In [114], the authors proposed attack strategies that use the generator from a generative adversarial network (GAN) to generate additive perturbations that can be used to attack a classifier. The authors of [115] and [116] use the generator from a GAN to generate adversarial examples that obey norm-based constraints. Alternatively, [117] use GANs to construct adversarial patterns that can be used to transfer adversarial examples from one domain to another. Similarly, [118] generate unrestricted adversarial examples via a generative model.

On the other hand, a framework called DefenseGAN, which uses a Wasserstein GAN to "de-noise" adversarial examples [119], has been proposed to defend against perturbation-based attacks. This defence method was later broken by the Robust Manifold Defense [120], which searches over the parameterised manifold induced by a generative model to find worst-case perturbations of data. The min-max formulation used in this work is analogous to the projected gradient descent (PGD) defence [26].

In contrast to the generative adversarial models, we describe in this thesis are works

that aim to learn and model the natural variability within worst-case perturbations of data via robust optimisation. Our methods are shown to be resistant against attacks such as PGD and C&W compared with the aforementioned works. Furthermore, it is still unknown in most of these aforementioned works whether they are robust to PGD and C&W. Also, the generated samples from our methods are perceptually realistic ones, which can be applied to another hierarchy of networks for robust training.

6.2.4 Equivariance and Invariance to Noises in Computer Vision

Parallel to the progress made toward training deep neural networks to be robust against small, norm-bounded adversarially chosen perturbations, a related line of work in the computer vision community has sought to design equivariant DNNs. In the context of adversarial robustness, if P is a function that perturbs an input by a small amount, DNNs are often trained to provide the same prediction for $f(P(x))$ and $f(x)$ [121].

More generally, several more recent works have sought to provide robustness or invariance against noise-based attacks; such works have included [122,123], which used an information-theoretic approach to edit the noise content of images to create perceptually similar data that caused misclassification. Similarly, another line of work has sought to use differentiable renderers to produce "semantic adversarial examples" [124]. In this line of work, mechanisms are often used to edit noise factors such as rotation or scaling in images by creating perturbations in a given semantic latent space [125].

The progress toward equivariant and invariant neural networks in computer vision has largely focused on designing new network topologies to combat a given transformation or a set of related transformations. Our robust training framework differs fundamentally from the above approaches in the following aspects. Rather than changing the topology of the neural network, we propose to change the robust training procedure according to the model of variation. In the case where the model is known (e.g. white-box attacks) we can use it during training to provide worst-case examples to train the neural network. In more challenging and natural cases where the model is unknown (and hence cannot be

used to alter the topology) we proposed to learn the model in advance and then use it for training. Our robust training paradigm could provide an intellectual bridge between robust deep learning and exploiting invariances in computer vision.

6.3 Overall Contributions

The contributions of our thesis can be summarised as follows:

- **Norm-based robust deep neural networks.** We propose a paradigm for different norm-bounded adversarial robustness with robust learning, wherein models of natural variation express changes due to a variety of applications.
- **Robust optimisation formulation.** We formulate the novel problem of model-based robust training by constructing novel general robust optimisation procedures that search for challenging norm-based variations of data.
- **Learned models of natural variation.** For many different forms of natural variation commonly encountered in safety-critical applications, we show that our robust models can be used to learn models of natural variations that are consistent with realistic conditions.
- **Model-based robust training algorithms.** We propose a family of novel robust training algorithms that exploit models of natural variations in order to improve the robustness of deep neural networks against different metrics of worst-case natural variations.
- **Broad applicability and robustness improvements.** We show empirically that models of natural variations can be used in our formulation to provide significant improvements in the robustness of deep neural networks for several datasets commonly used in deep neural networks. We reported improvements as large as 10-20 percentage points in the test efficiency compared to state-of-the-art adversarially robust classifiers on tasks involving challenging conditions.

- **Reusability and modularity of models of natural variation.** We show that models of variations can be reused on multiple new and different datasets without retraining to provide high levels of robustness against naturally varying conditions. Further, we show that models of different variations can be easily composed to provide robustness against multiple forms of variations.

- **Out-of-distribution robustness.** We show that our norm-based paradigm can be used to provide robustness to unseen and out-of-distribution data that has been subjected to higher levels of variations than the data that is seen during training. We conclude robust methods varying against both white-box and black-box attacks while maintaining competitive efficiency.

While the experiments in this thesis focus on image classification tasks subject to challenging conditions, our norm-based robust learning paradigms are much broader and can, in principle, be applied to many other deep learning domains as long as one can obtain accurate models of how the data can vary in a useful manner. Before diving into numerous directions for future research, we list some limitations in which this thesis does not address in the next section.

6.4 Thesis Limitations

We enumerate the uncovered scopes from four aspects.

Different Noise Models. In this thesis, we used the robust-learning framework to learn robustness under different constraint metrics. We did not explore different noise models, e.g., image rotations or different light conditions. This is left for future work.

Different Model-based Architectures. Though we evaluated the robustness in different sizes of models, we do not compare under different model types, e.g., from ResNets to VGGNets. As we select model types following the principles of choosing the most accurate model for some specific dataset, we do not evaluate under various types of model architectures.

Different Applications beyond Image Classification. In this thesis, we focus on researches in image classifications but robustness against adversarial attacks exists in different domains, e.g., natural language processing (NLP), malware detection, or network attack. As a constantly increasing number of real-world applications and systems have been powered by deep learning, robustness in these applications becomes more and more important.

Theoretical Foundations. While this thesis emphasises improvement for robustness against different noises from a physical perspective, a different perspective from a theoretical aspect is not a central objective here. Some theoretical questions can be considered in this field. For example, how do we develop a faster algorithm from a geometric transformation such as rotation or scaling as well as a statistical perspective aforementioned?

6.5 Future Work

In this thesis, we investigated the problem of ensuring the robustness of deep neural networks with respect to small changes in the input. Motivated by perceptible perturbations in computer vision, such as lighting changes, we proposed novel algorithms based on robust training for deep neural networks that provides robustness with respect to different variations. Our notion of robustness differs from the notion of adversarial training with respect to norm-bounded perturbations. Our optimisation-based formulation for model-based training results in a family of training algorithms that we refer to as model-based robust training. These algorithms exploit either known or previously learned models of natural variations using both robust and adversarial approaches. Given a model of natural variation P that models naturally occurring perturbations, the main idea across these algorithms is to use P to perform model-based data augmentation or model-based adversarial training to produce samples with varying noises. In the case of unknown noises, by blending models P with adversarial training, we empirically find that our model-based paradigms provide significant robustness improvements for numerous physically meaningful noises across various datasets. Our model-based paradigm is naturally compositional,

leverages models across datasets, and shows improved robustness as datasets become more challenging. In what follows, we briefly highlight several of these broad directions from the aforementioned perspectives.

Learning a Library of Different Noise Models. First, the problem of how to best learn a model of natural variations to perform model-based training is an open and interesting problem. In this thesis, we used the robust-learning framework, but other existing architectures may be better suited for specific noises or datasets. Indeed, a more rigorous statistical analysis of generative models may lead to the discovery of new architectures designed specifically for model-based training. To this end, recent work in learning equivariances in computer vision may provide insight into learning physically meaningful models.

Model-based Algorithms and Architectures. Another important direction involves the development of new algorithms for solving the min-max formulation. In this thesis, we presented three algorithms which can be used to approximately solve robustness problem. In particular, adapting Monte-Carlo methods to search globally over the manifold induced by learned models in a latent space of variability may provide more efficient, scalable, or robust results. Do we need to decouple offline learning of a model of natural variation or it is possible to think of a new architecture in which the model and the classifier can be learned simultaneously? Another interesting direction is to rethink deep network architectures in a model-based manner by taking inspiration from how equivariance is exploited in deep network architectures used in computer vision.

Applications beyond Image Classification. Throughout the thesis, we have focused on empirical demonstrations of our approaches in numerous image classification tasks. But our model-based paradigm could be broadly applied in numerous applications within computer vision as well as outside computer vision. Within computer vision, one can consider other tasks, such as segmentation, in the presence of challenging physical noises. Outside computer vision, one exciting area is to exploit physical models of robot dynamics with deep reinforcement learning for applications such as walking in unknown terrains. In any domain where one has access to good models, our approaches allow domain experts to leverage these models in order to make deep neural networks more

robust.

Theoretical Foundations. Finally, we believe that there are many interesting open questions with respect to the theoretical aspects of model-based robust training. What type of models provides significant robustness gain in our paradigms? How accurate does a model need to be to produce neural networks that are robust to natural variations? We would like to address such theoretical questions from a geometric, physical as well as a statistical perspective with an eye toward developing faster algorithms that are both more sample-efficient as well as more robust. A deeper theoretical understanding of our robust learning paradigm could result in new approaches that blend model-based and data-based methods and algorithms.

Bibliography

[1] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *Proceedings of the 5th International Conference on Computers and Games*, 2006, pp. 72–83.

[2] Y. Liu and A. Lomuscio, "An mcts-based adversarial training method for image recognition," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2019.

[3] Y. Liu and A. Lomuscio, "Robustness learning via decision tree search robust optimisation," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2021.

[4] Y. Liu and A. Lomuscio, "A method for robustness against adversarial attacks on deep neural networks," in *Proceedings of the International Joint Conference on Neural Networks and IEEE World Congress on Computational Intelligence (WCCI)*, 2020.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[6] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[7] I. Sutskever, O. Vinyals, and Q. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27*, 2014, pp. 3104–3112.

[8] S. Sabour, N. Frosst, and G. Hinton, "Dynamic routing between capsules," in *Advances in neural information processing systems*, 2017, pp. 3856–3866.

[9] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *arXiv:1810.04805*, 2018.

[10] I. Apple, "Face id security https://images.apple.com/business/docs/FaceID_Security_Guide.pdf," 2017.

[11] H. Tamura and N. Yokoya, "Image database systems: A survey," *Pattern Recognition*, vol. 17, no. 1, pp. 29–43, 1984.

[12] S. Vijayanarasimhan and P. Natsev., "large visual databases https://research.googleblog.com/2016/09/announcing-youtube-8m-large-and-diverse.html," 2016.

[13] M. Bojarski, D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," in *arXiv: 1604.07316*, 2016.

[14] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015, pp. 427–436.

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[16] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," *ECML PKDD*, vol. 8190, no. 3, pp. 387–402, 2013.

[17] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[18] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *arXiv: 1607.02533*, 2016.

[19] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song, "Robust physical-world attacks on deep learning models," in *arXiv:1707.08945*, 2017.

[20] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[21] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proceedings of the 37th IEEE Symposium on Security and Privacy*. IEEE, 2016, pp. 582–597.

[22] F. Tramer, A. Kurakin, N. Papernot, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," in *arXiv:1705.07204*, 2017.

[23] A. Rozsa, M. Gunther, and T. Boult, "Towards robust deep neural networks with bang," in *arXiv:1612.00138*, 2016.

[24] W. He, J. Wei, X. Chen, N. Carlini, and D. Song, "Adversarial example defenses: Ensembles of weak defenses are not strong," in *arXiv:1706.04701*, 2017.

[25] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," in *arXiv:1704.01155*, 2017.

[26] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[27] V. F. J.H. Metzen, T. Genewein and B. Bischoff, "On detecting adversarial perturbations," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[28] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[29] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[30] T. Mitchell, *Machine Learning.* New York: McGraw-Hill, 1997.

[31] N. Kasabov, *Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence*, 1st ed. Springer, 2018.

[32] C. Bishop, *Pattern Recognition and Machine Learning.* Berlin, Heidelberg: Springer-Verlag, 2006.

[33] S. Haykin, *Neural networks and learning machines*, 3rd ed. Upper Saddle River, NJ: Pearson Education, 2009.

[34] D. Hubel and T. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat' s visual cortex," *The Journal of Physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[35] K. Fukushima, "Neocognitron: Self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 1980.

[36] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, 2016, pp. 2574–2582.

[37] W. Knight, "The dark secret at the heart of ai," in *MIT Technology Review*, 2017.

[38] M. Barreno, B. Nelson, A. Joseph, and J. Tygar, "The security of machine learning," *Machine Learning*, vol. 81, no. 2, pp. 121–148, 2010.

[39] F. Roli, B. Biggio, and G. Fumera, "Pattern recognition systems under attack," *of the 18th Iberoamerican Congress on Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, vol. 8258, pp. 1–8, 2013.

[40] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proceedings of 10th International Conference on Malicious and Unwanted Software (MALWARE)*, 2015, pp. 11–20.

[41] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proceedings of the 37th IEEE Symposium on Security and Privacy*. IEEE, 2016, pp. 372–387.

[42] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proceedings of the 38th IEEE Symposium on Security and Privacy*. IEEE, 2017, pp. 39–57.

[43] J. Su, D. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019.

[44] F. Croce and M. Hein, "Minimally distorted adversarial examples with a fast adaptive boundary attack," in *arXiv: 1604.07316*, 2019.

[45] A. Modas, S. Moosavi-Dezfooli, and P. Frossard, "Sparsefool: A few pixels make a big difference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 9087–9096.

[46] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations, ICLR*, 2015.

[47] N. Carlini and D. Wagner, "Defensive distillation is not robust to adversarial examples," *CoRR*, 07 2016.

[48] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin, "On evaluating adversarial robustness," *CoRR*, vol. abs/1902.06705, 2019.

[49] M. Sharif, S. Bhagavatula, L. Bauer, and M. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 152–1540.

[50] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "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.

[51] J. Lu, H. Sibai, E. Fabry, and D. Forsyth, "NO need to worry about adversarial examples in object detection in autonomous vehicles," *CoRR*, 2017.

[52] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, "Synthesizing robust adversarial examples," *CoRR*, 2017.

[53] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *arXiv:1903.12261*, 2019.

[54] X. Wei, J. Zhu, S. Yuan, and H. Su, "Sparse adversarial perturbations for videos," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 8973–8980.

[55] 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 (CVPR)*. IEEE, 2016, pp. 770–778.

[56] J. Lu, T. Issaranon, and D. Forsyth, "Safetynet: Detecting and rejecting adversarial examples robustly," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 446–454.

[57] A. Bhagoji, D. Cullina, and P. Mittal, "Dimensionality reduction as a defense against evasion attacks on machine learning classifiers," in *arXiv:1704.02654*, 2017.

[58] R. Feinman, R. Curtin, S. Shintre, and A. Gardner, "Detecting adversarial samples from artifacts," in *arXiv:1703.00410*, 2017.

[59] Z. Gong, W. Wang, and W. Ku, "Adversarial and clean data are not twins," in *arXiv:1704.04960*, 2017.

[60] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," in *arXiv:1702.06280*, 2017.

[61] D. Hendrycks and K. Gimpel, "Early methods for detecting adversarial images," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[62] D. Meng and H. Chen, "Magnet: a two-pronged defense against adversarial examples," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 135–147.

[63] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proceedings of the International Conference on Machine Learning (ICML)*. Omnipress, 2011, pp. 833–840.

[64] X. Li and F. Li, "Adversarial examples detection in deep networks with convolutional filter statistics," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 5764–5772.

[65] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," in *Annual Network and Distributed System Security Symposium (NDSS)*, 2018, pp. 1–15.

[66] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman, "Pixeldefend: Leveraging generative models to understand and defend against adversarial examples," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[67] C. Guo, M. Rana, M. Cisse, and L. Maaten, "Countering adversarial images using input transformations," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[68] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille, "Mitigating adversarial effects through randomization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[69] E. Raff, J. Sylvester, S. Forsyth, and M. McLean, "Barrage of random transforms for adversarially robust defense," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 6521–6530.

[70] C. Xie, Y. Wu, L. Maaten, A. Yuille, and K. He, "Feature denoising for improving adversarial robustness," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 501–509.

[71] Y. Wu, D. Bamman, and S. Russell, "Adversarial training for relation extraction," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1779–1784.

[72] Y. Dong, H. Su, J. Zhu, and F. Bao, "Towards interpretable deep neural networks by leveraging adversarial examples," in *arXiv:1708.05493*, 2017.

[73] R. Huang, B. Xu, D. Schuurmans, and C. Szepesvari, "Learning with a strong adversary," in *arXiv:1511.03034*, 2015.

[74] Y. LeCun and C. Cortes, "Mnist handwritten digit database http://yann.lecun.com/exdb/mnist/," 1998.

[75] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10 https://www.cs.toronto.edu/~kriz/cifar.html," 2010.

[76] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 1999, pp. 1150–1157.

[77] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, 2004.

[78] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G.Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[79] A. Bental, L. E. Ghaoui, and A. Nemirovski, *Robust Optimization*, ser. Princeton Series in Applied Mathematics. Princeton University Press, October 2009.

[80] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2016, pp. 87.1–87.12.

[81] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," in *arXiv:1605.07277*, 2016.

[82] Y. Liu and A. Lomuscio, "Mrobust: A method for robustness against adversarial attacks on deep neural networks," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

[83] A. Fawzi, S. Moosavi-Dezfooli, P. Frossard, and S. Soatto, "Empirical study of the topology and geometry of deep networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018, pp. 3762–3770.

[84] X. Ma, B. Li, Y. Wang, S. Erfani, S. Wijewickrema, G. Schoenebeck, D. Song, M. Houle, and J. Bailey, "Characterizing adversarial subspaces using local intrinsic dimensionality," in *arXiv:1801.02613*, 2018.

[85] F. Tramr, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "The space of transferable adversarial examples," in *arXiv:1704.03453*, 2017.

[86] S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 1765–1773.

[87] D. Warde-Farley and I. Goodfellow, Adversarial perturbations of deep neural networks. In T. Hazan, G. Papandreou, and D. Tarlow, editors, *Perturbations, Optimization, and Statistics*. MIT Press, 2016.

[88] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math compiler in python," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010.

[89] E. Battenberg, S. Dieleman, and al, "Lasagne: Lightweight library to build and train neural networks in theano," 2015.

[90] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. 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, p. 506—519.

[91] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," in *Proceedings of the 35th International Conference on Machine Learning*, 2018, pp. 274–283.

[92] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[93] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14 410–14 430, 2018.

[94] C. Guo, J. Gardner, Y. You, A. Wilson, and K. Weinberger, "Simple black-box adversarial attacks," *CoRR*, 2019.

[95] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," *CoRR*, 2018.

[96] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," 2018.

[97] H. Anderson, J. Woodbridge, and B. Filar, "Deepdga: Adversarially-tuned domain generation and detection," 2016.

[98] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan," 2017.

[99] H. Anderson, A. Kharkar, B. Filar, and P. Roth, "Evading machine learning malware detection," 2017.

[100] F. Tramer, N. Carlini, W. Brendel, and A. Madry, "On adaptive attacks to adversarial example defenses," 2020.

[101] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. Ghaoui, and M. Jordan, "Theoretically principled trade-off between robustness and accuracy," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019, pp. 7472–7482.

[102] J. Kolter and E. Wong, "Provable defenses against adversarial examples via the convex outer adversarial polytope," *CoRR*, 2017.

[103] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," *CoRR*, 2018.

[104] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas, "Efficient and accurate estimation of lipschitz constants for deep neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 427–11 438.

[105] M. Fazlyab, M. Morari, and G. Pappas, "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming," 2020.

[106] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier, "Parseval networks: Improving robustness to adversarial examples," in *Proceedings of the 34th International Conference on Machine Learning*, 2017, pp. 854–863.

[107] Y. Dong, F. Liao, T. Pang, X. Hu, and J. Zhu, "Discovering adversarial examples with momentum," 2017.

[108] Z. Wu, S. Lim, L. Davis, and T. Goldstein, "Making an invisibility cloak: Real world adversarial attacks on object detectors," 2020.

[109] C. Kanbak, S. Moosavi-Dezfooli, and P. Frossard, "Geometric robustness of deep networks: analysis and improvement," 2017.

[110] M. Balunovic, M. Baader, G. Singh, T. Gehr, and M. Vechev, "Certifying geometric robustness of neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 313–15 323.

[111] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, "Exploring the landscape of spatial robustness," 2019.

[112] X. Wang, A. Shrivastava, and A. Gupta, "A-fast-rcnn: Hard positive generation via adversary for object detection," in *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 3039–3048.

[113] Z. Wu, K. Suresh, P. Narayanan, H. Xu, H. Kwon, and Z. Wang, "Delving into robust object detection from unmanned aerial vehicles: A deep nuisance disentanglement approach," in *2019 IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 1201–1210.

[114] C. Xiao, B. Li, J. Zhu, W. He, M. Liu, and D. Song, "Generating adversarial examples with adversarial networks," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 2018, pp. 3905–3911.

[115] L. Schott, J. Rauber, M. Bethge, and W. Brendel, "Towards the first adversarially robust neural network model on mnist," 2018.

[116] Z. Zhao, D. Dua, and S. Singh, "Generating natural adversarial examples," 2018.

[117] M. Naseer, S. Khan, H. Khan, F. Khan, and F. Porikli, "Cross-domain transferability of adversarial perturbations," 2019.

[118] I. Dunn, H. Pouget, T. Melham, and D. Kroening, "Adaptive generation of unrestricted adversarial inputs," 2019.

[119] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-gan: Protecting classifiers against adversarial attacks using generative models," 2018.

[120] A. Jalal, A. Ilyas, C. Daskalakis, and A. Dimakis, "The robust manifold defense: Adversarial training using generative models," 2019.

[121] J. Cohen, E. Rosenfeld, and Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *Proceedings of the 36th International Conference on Machine Learning*, 2019, pp. 1310–1320.

[122] J. Jacobsen, J. Behrmann, R. Zemel, and M. Bethge, "Excessive invariance causes adversarial vulnerability," 2020.

[123] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener, "Efficient verification of neural networks via dependency analysis," in *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, 2020, pp. 3291–3299.

[124] T. Dreossi, S. Jha, and S. Seshia, "Semantic adversarial deep learning," 2018.

[125] L. Jain, S. Chen, W. Wu, U. Jang, V. Chandrasekaran, S. Seshia, and S. Jha, "Generating semantic adversarial examples with differentiable rendering," 2020.