KU LEUVEN

FACULTY OF
ENGINEERING SCIENCE

Efficient and Evasive Distributed Adversarial Attacks using Particle Swarm Optimization

Sander Prenen

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
computerwetenschappen, hoofdoptie
Artificiële intelligentie

Promotoren:
Prof. dr. ir. W. Joosen
Dr. ir. D. Preuveneers

Assessoren:
Prof. dr. D. Devriese
I. Tsingenopoulos

Begeleiders:
I. Tsingenopoulos
V. Rimmer

Academiejaar 2021 – 2022

© Copyright KU Leuven

Without written permission of the thesis supervisors and the author it is forbidden to reproduce or adapt in any form or by any means any part of this publication. Requests for obtaining the right to reproduce or utilize parts of this publication should be addressed to the Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 or by email `info@cs.kuleuven.be`.

A written permission of the thesis supervisors is also required to use the methods, products, schematics and programmes described in this work for industrial or commercial use, and for submitting this publication in scientific contests.

Zonder voorafgaande schriftelijke toestemming van zowel de promotoren als de auteur is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot het Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 of via e-mail `info@cs.kuleuven.be`.

Voorafgaande schriftelijke toestemming van de promotoren is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Preface

In front of you lies the thesis: "Efficient and Evasive Distributed Adversarial Attacks using Particle Swarm Optimization". The text is the result of a year of hard work and I am very proud of the final result.

The subject seemed very interesting when I first read about it in May of 2021. Having worked on it for the past year, I can guarantee you that it did not only seem an interesting topic, it proved to be one as well. I hope you agree with me once you have read this text and I wish you a lot of enjoyment in reading it.

This work will be the crown jewel of my academic career. After five years of blood, sweat and tears, albeit not literally, I can hear the fat lady singing in the distance. Before it is completely over, I want to thank everyone that helped me over the course of these five years. I want to thank my family and girlfriend for supporting me on this journey and proofreading all reports I have written, even though the contents of some of them was very difficult to grasp. I want to thank my friends for the fun times and the company during the lessons, some of which seemed to last for days. I want to thank the assessors and members of the jury for taking their time to read this text and triggering my brain one last time with their interesting remarks and questions. Last but not least, I want to thank my supervisor, co-supervisor and assistant-supervisors for the guidance throughout the year and their feedback on my intermediate work.

Elvis has left the building!

Sander Prenen

Contents

Abstract

Adversarial attacks craft small perturbations that can be added to inputs, so that they are classified incorrectly by a classifier. Current defensive schemes try to flag these attacks based on the similarity between successive submitted queries. This work aims to implement a new adversarial attack that is able to bypass this detection mechanism. This attack is called PSO-BBA after the two constituent algorithms Particle Swarm Optimization and Biased Boundary Attack [1]. Other improvements, such as distributing the query submission over multiple nodes or grouping attacks together have been explored as well. The final attack is more evasive than current state-of-the-art attacks, but is slightly less efficient. The attacker has to make a trade-off depending on the primary goal of the attack.

Samenvatting

Adversarial aanvallen creëren kleine perturbaties die aan afbeeldingen kunnen worden toegevoegd. Door deze toevoeging, worden deze afbeeldingen niet langer juist geclassificeerd door een kunstmatige intelligentie klasseerder. Moderne verdedigingsmechanismen proberen om zo een aanval te deteteren op basis van gelijkenissen tussen opeenvolgend verstuurde afbeeldingen. Dit werk implementeert een nieuwe *adversarial* aanval genaamd PSO-BBA. Deze aanval is genoemd naar de twee onderdelen, namelijk *Particle Swarm Optimization* en *Biased Boundary Attack* [1]. Het doel van de nieuwe aanval is het omzeilen van het verdediginsmechanisme. Verschillende verbeteringen worden voorgesteld en aangebracht aan het algoritme. Enkele van deze verbeteringen zijn: een gedistribueerde manier voor het versturen van afbeeldingen over meerdere machines en het groeperen van aanvallen. De uiteindelijke aanval wordt minder gedetecteerd dan de modernste aanvallen, maar hij is minder performant. De aanvaller moet dus een afweging maken afhankelijk van het belangrijkste doel, detectie omzeilen of hoge performantie.

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence. 1, 2, 3

ANN Artificial Neural Network. 3, 4, 6

API Application Programming Interface. 7, 24, 25, 33, 65

BA Boundary Attack. 15, 16, 17, 18, 44

BBA Biased Boundary Attack. 16, 18, 21, 24, 29, 30, 31, 32, 33, 36, 38, 41, 42, 44, 46, 47, 50, 51, 53, 54, 55, 60, 61

CNN Convolutional Neural Network. 3, 4

DB Distance-Based. 34, 36, 37, 60

DDoS Distributed Denial of Service. 21

DkNN Deep k-Nearest Neighbors. 10

DNN Deep Neural Network. 3

EA Evolutionary Algorithm. 10, 20

EDB Embedded-Distance-Based. 34, 36, 37, 38, 39, 40, 41, 60

FEDB Flushing-Embedded-Distance-Based. 37, 38

FGSM Fast Gradient Sign Method. 7, 53

HSJA HopSkipJumpAttack. 17, 18, 36, 44, 46, 47, 50, 51, 53, 54

MGRR Multi-Group with Random Redistribution. 12, 13

ML Machine Learning. 1, 2

MRR Modified Round-Robin. 33, 34, 36, 60

PSO Particle Swarm Optimization. 2, 10, 11, 12, 13, 21, 24, 29, 30, 31, 32, 33, 36, 38, 41, 42, 44, 46, 47, 50, 51, 53, 54, 55, 59, 62

ReLU Rectified Linear Unit. 3, 5, 28

RNN Recurrent Neural Network. 3

RR Round-Robin. 33, 34, 35, 36, 42, 46, 60

SVM Support Vector Machine. 1

Chapter 1

Introduction

One of the most basic tasks in Artificial Intelligence (AI) or more specifically Machine Learning (ML) is the classification of data. This task is usually performed by a classifier or model based on features present in this data. Some examples of classification tasks are: character recognition [2], spam detection [3] and face recognition [4]. These tasks are generally very daunting for humans causing a great interest in improving the accuracies of these classifiers.

The first classifiers made use of algorithms such as nearest neighbors [5], decision trees [6] and Support Vector Machines (SVMs) [7]. More recent classifiers are based on neural networks, which in turn are inspired by the human brain. These networks come in a great variety of forms and show near-human performance on some very specific tasks [8]. This improvement in performance caused even more interest in the applications of AI and ML. Forbes [9] recently predicted that in ten years AI will be present in all areas of our society. The next decade is deemed as the *"most promising era in technology innovation"* [9].

The prevalence of ML in the near future does not only bring opportunities. It also poses some threats that may not be obvious at first glance. A spam email passing through an AI-based spam filter might not be the end of the world, while mispredicting some characters from an image can have a bigger impact depending on the context. False predictions in the medical domain might be even more severe, but they could be overturned by a doctor. Self-driving cars not detecting traffic signs could put several lives at risk. Depending on the situation in which the classifier is deployed, some threats can be more dangerous than others.

All previous examples of threats are due to the inaccuracies of the classifier itself. There are also some threats caused by the inherent nature of neural networks. In 2014, it was shown that neural networks are prone to adversarial examples [10]. An adversarial example consists of a correctly classified data instance and a small perturbation. This perturbation causes the neural network to misclassify the instance. This allows malicious users to create images that are seemingly identical, while being

classified differently. This poses an even greater threat to the security of AI than the inaccuracies of the classifiers.

The process of creating an adversarial example is called an adversarial attack. Different types of attacks exist depending on the information available to the attacker. White-box attacks require complete knowledge of the classifier under attack, while black-box attacks only require the output(s) of the model. All attacks require sending one or more queries to the classifier.

Ever since the discovery of adversarial examples, an arms race between adversarial attack and defense researchers has been taken place. In the current state of affairs in the adversarial ML domain, the stateful defensive mechanism [11] is considered the state-of-the-art. Unlike previous defensive schemes, this scheme holds state of previously submitted queries. This allows the scheme to make a decision based on a series of queries instead of a single query.

The stateful defense mechanism makes the assumption that there is no collaboration between different users of the model and that every submitted query can be traced back to the submitting user. This is not necessarily the case in real scenarios, since users are free to work together. It is even possible for one user to set up multiple accounts and essentially collaborate with itself. This work aims to exploit the assumption made by the defensive mechanism and create adversarial examples while triggering as few detections as possible.

This goal is achieved by first altering an existing adversarial attack using Particle Swarm Optimization (PSO) in order to bypass the stateful defense mechanism. PSO is an optimization framework inspired by the flocking of birds. Afterwards multiple ideas are explored in an effort to improve the efficiency or evasiveness of the altered attack. Some ideas are specific for the created attack, while others are generally applicable to all other adversarial attacks.

The work is structured as follows: Chapter 2 discusses the necessary background needed in order to comprehend the rest of the work. Chapter 3 gives an overview of work related to this subject. Some specific adversarial attacks are discussed as is the stateful defense mechanism. Chapter 4 introduces the main ideas used in the development of the altered attack. This chapter also poses some interesting research questions. Chapter 5 proposes a novel attack and iteratively refines this attack in order to increase the efficiency or evasiveness. At the end of every section of this chapter, some conclusions are drawn and discussed. Chapter 6 contains a discussion about the work as a whole. Some final remarks are given as well as some pointers for future work. Finally, Chapter 7 concludes this work by answering the research questions posed in Chapter 4.

Chapter 2

Background

2.1 Neural networks

Ever since the invention of computer systems, it has always been a goal of scientists and engineers to create AI. Current state-of-the-art approaches are mimicking the human brain, more specifically the neurons inside the brain. Already in the fifties, Rosenblatt introduced his perceptron [12]. The perceptron is a single neuron able to learn binary linearly separable patterns. It does so by finding a hyperplane that separates the two classes. This hyperplane is called the decision surface or decision boundary and the perceptron itself is called a classifier. Geometric regions separated by a decision boundary are called decision regions. The concept of linear separability is explained in Figure 2.1 in two dimensions. In Figure 2.2, the decision boundaries and decision regions are explained visually.

Unfortunately not all patterns are linearly separable. To overcome this problem, the neurons can be layered, creating an Artificial Neural Network (ANN) in the process. Layering neurons sequentially is essentially a linear combination of neurons. This in itself does not create non-linear decision surfaces. Non-linear activation functions are added for the ANN to be able to learn more complex decision boundaries. Some commonly used activation functions are Rectified Linear Unit (ReLU) [13], Heaviside step function [14] and softmax [15] (or sigmoid when used on scalars). In Figure 2.3 the plots of some activation functions can be found.

The neurons can be combined in different ways to create different ANN architectures. Each architecture has its own strengths and weaknesses. Convolutional Neural Networks (CNNs) excel in classifying visual data [4, 16, 17], whilst Recurrent Neural Networks (RNNs) are widely used when there exist dependencies inside the data, such as in speech recognition [18, 19] or time series prediction [20]. More recent research focuses on Deep Neural Networks (DNNs), due to the ever increasing computational power available. DNN approaches are able to compare to and even surpass human performance on very specific tasks [8, 21].

Most ANNs are not built from individual neurons, but rather from layers of neurons. An ANN architecture describes the different layers of which the network consists. The most basic type of layer is the fully connected or dense layer. This layer, as the name suggests, consists of neurons that are connected to all neurons of the previous layer. Every neuron in this layer outputs a value based on the perceptron rule:

$$y = b + W \cdot X$$

Here y is the output of the neuron, b is the bias, W is the weight matrix and X is the vector consisting of the outputs of the previous layer. The values of the bias and weights are learned based on the training data. $W \cdot X$ is the dot product $\sum_{i=1}^{n} w_i x_i$, where n is the number of neurons in the previous layer [22].

Activation layers apply activation functions to their inputs. They can be standalone layers, but more frequently they are integrated in other layers.

Convolution layers [2] are the main building blocks of CNNs. Convolution layers map inputs to so called feature maps. A filter W slides over the input X from left to right top to bottom. At every position, the dot product $W \cdot X$ of the frame and the underlying values of the input are computed. This value is placed at the corresponding position of the feature map. Higher values of the dot product (for a fixed value of $\|W\|$) mean that W and X are more similar at this position of X. The feature map therefore indicates where in the input X a certain pattern W can be found. This process is translation-invariant, meaning that a certain pattern can be detected at any location of X. Scale and rotation invariance can be achieved by repeating the process for scaled and rotated filters. This invariance is the main reason why CNNs are widely used for visual data. An example of a convolution layer is shown in Figure 2.4.

The use of convolution layers tends to cause an explosion in the dimensionality of the data. One filter will produce a feature map that is smaller than the original input. But a convolution layer typically consists of many filters, causing the explosion in the number of connections and weights. Pooling or subsampling layers [2] alleviate this problem by aggregating nearby points. The aggregation happens using a sliding window similar to a convolution layer. This approach of reducing the dimensionality has the added benefit that the output of the model is less sensitive to the exact location of a pattern. Some frequently used pooling layers are the MaxPooling and AveragePooling layers. They aggregate nearby points using the maximum and average value respectively. Convolution and pooling layers are often used in combination with each other and are repeated multiple times. This allows the CNN to detect patterns at different scales.

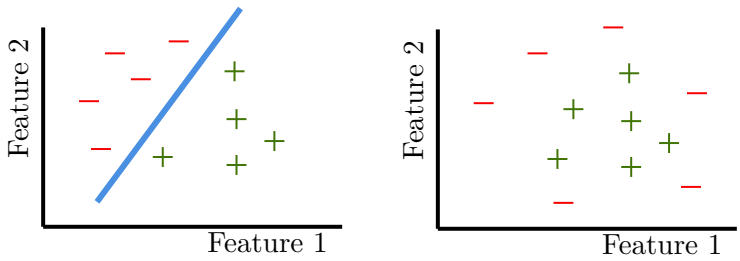


FIGURE 2.1: Linearly separable classes on the left and non-linearly separable classes on the right. Two classes are linearly separable if there exists a hyperplane for which all examples of one class are on the same side of this hyperplane, whilst all examples of the other class are on the other side of the hyperplane. In two dimensions, the hyperplane is a straight line.

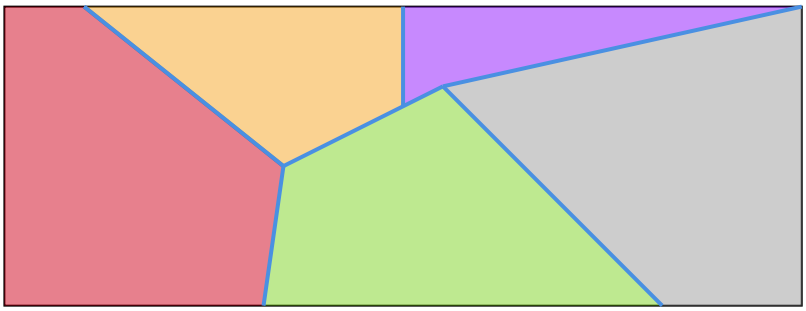


FIGURE 2.2: Decision boundaries (blue lines) separate different decision regions (colored regions).

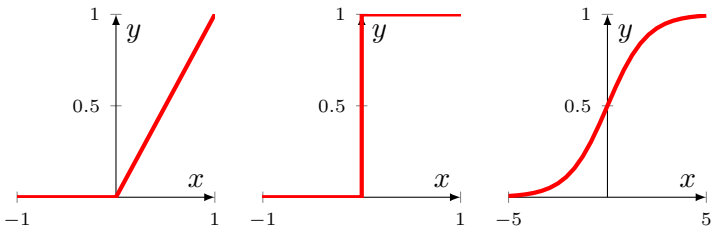


FIGURE 2.3: Plots of different activation functions. From left to right: ReLU, Heaviside step and sigmoid.

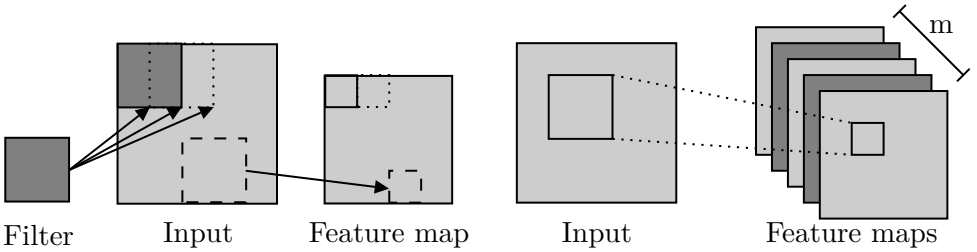


Figure 2.4: Example of a convolution layer. The filter slides over the input from left to right top to bottom. At every position, the dot product of the filter and input image is computed. This value is added to the corresponding position of the feature map. This process is repeated for m different filters leading to m feature maps. Inspired by [23].

2.2 Adversarial attacks

The expressiveness of ANNs is a double-edged sword. It is the cause for the near-human performance on some tasks, but also for counter-intuitive properties. As studied by Szegedy et al [10], one of these properties is the presence of discontinuous decision boundaries. This might cause seemingly identically images to be classified differently. They were the first to define adversarial examples as *"imperceptibly small perturbations to a correctly classified input image, so that it is no longer classified correctly"* [10]. This property of ANNs might not seem important at first glance, but it can be quite worrisome from a security point-of-view. Malicious users could craft images to bypass face recognition software [24] or attack the camera of a self-driving car to misclassify traffic signs [25]. These images would seem identical to correctly classified images for humans. Other fields where adversarial examples are of interest include malware detection [26], natural language processing [27] and industrial control systems [28]. Adversarial attacks are algorithms used to craft such adversarial examples.

All adversarial attacks are evaluated against a threat model. A threat model is *"a structured representation of all the information that affects the security of an application"* [29]. This information consists of the goals, knowledge and capabilities of the attacker, the accessibility of the model under attack and the costs of (un)successful attacks.

Most research on adversarial attacks is done using images. Researchers have the most freedom in this domain, since a slightly altered image is still an image with roughly the same contents. Slightly modifying an industrial control system however, might cause complete disruption of the system. Research in other domains is mostly conducted by altering existing image algorithms to the specific use case. For this reason this work focuses on adversarial attacks on images only.

2.2.1 Adversarial attacks terminology

Adversarial attacks are generally divided in two categories, white-box attacks and black-box attacks. In a white-box attack, the attacker has complete knowledge of the classifier under attack. This knowledge consists of the architecture, parameters and thus their gradients and all output of the classifier. Examples of white box attacks are the Fast Gradient Sign Method (FGSM) [30] or the Carlini & Wagner attack [31].

In black-box attacks, the only piece of information the attacker has access to is the output of the model. Depending on the literature, this output consists of the final decision or class label(s) only (decision-based attack) or the class labels and the corresponding confidence scores (score-based attacks). Black-box attacks are more relevant in real scenarios, since most attacks are performed on a third-party Application Programming Interface (API). These APIs generally do not reveal the underlying model.

Transfer attacks [32] try to overcome this hurdle by creating a surrogate model. This is an undefended model similar to the model under attack. This idea is based on the observation that adversarial perturbations often are transferable to other models than the one they were designed for [30]. Attacks can leverage information (such as gradients) from the surrogate model to breach the black-box model. Due to the transferability of adversarial examples, it is also possible to perform so called zero-query attacks. Zero-query attacks are performed entirely on the surrogate model and the resulting adversarial example is forwarded to the black-box model.

Both white-box and black-box attacks can be divided into targeted and untargeted attacks depending on their goal. In a targeted attack, the goal of the attacker is to create an adversarial example with a specific target class. In an untargeted attack the target class can be any class. Untargeted variants of attacks generally enjoy much more freedom and are therefore able to craft adversarial examples that are closer to the original. Figure 2.5 visually explains the difference between the two types of attack.

What does it mean for images to be close to each other? This is easy to visualize in two dimensions as in Figure 2.5, but in higher dimensions, this is more difficult. Images reside in d-dimensional space, where d is the amount of pixels of the image[1]. Two commonly used distances in higher dimensions are the L_2-distance and the L_∞-distance. They are defined as follows:

[1] For color images in RGB-space, there are actually three times the amount of pixels. A complete set for each color channel.

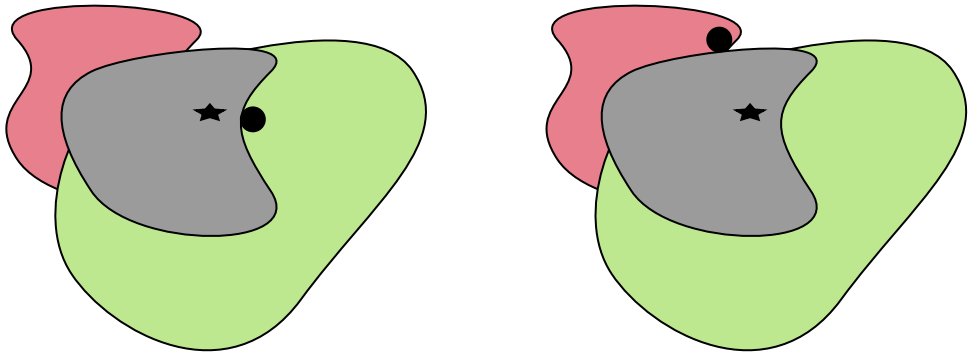


Figure 2.5: Different decision regions are shown in different colors. An adversarial example is being created starting from the original image (black star). On the left an untargeted attack is performed. The adversarial example is the image closest to the original image, that is classified differently (black circle). On the right a targeted attack is performed with the red decision region being the target class. The adversarial example is the image closest to the original image that is in the red decision region.

$$
\begin{aligned}
L_2(X,Y) &= \sqrt{\sum_{i=1}^{d} |x_i - y_i|^2} \\
L_\infty(X,Y) &= \lim_{p\to\infty} \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p} \\
&= \max(|x_1 - y_1|, |x_2 - y_2|, \ldots, |x_d - y_d|)
\end{aligned}
$$

In both distances X and Y represent the images and $(x_1, x_2, \ldots, x_i, \ldots, x_d)$ and $(y_1, y_2, \ldots, y_i, \ldots, y_d)$ are the pixel values of X and Y respectively. The L_2-distance is also known as the Euclidean distance, which is a generalization of the Pythagorean theorem in more than two dimensions. It takes the pairwise distances between all pixels into account. The L_∞-distance is also called the Chebyshev distance. This distance only depends on the maximal pairwise distance between the two images. By minimizing the L_∞-distance, the maximal pixelwise difference is minimized [33]. Sometimes the norm notation $\|\cdot\|$ is used instead of the distance notation. A distance is induced by a norm by $d(X,Y) = \|X - Y\|$. The other direction, a distance inducing a norm, is not as trivial. In order for a distance to induce a norm, the distance has to satisfy the following condition: $d(\alpha X, \alpha Y) = |\alpha| d(X,Y)$. This condition is satisfied in most commonly used distances, but this is not always the case. The

discrete distance [34] is an example of such a distance and is defined as follows:

$$d(X, Y) = \begin{cases} 1, & \text{if } X \neq Y \\ 0, & \text{else} \end{cases}$$

Multiplying both X and Y with a fixed value α does not change the outcome of the distance function.

2.2.2 Adversarial defenses

The existence of adversarial attacks naturally gave rise to adversarial defenses. These defenses can be categorized based on their objective. They can be either proactive or reactive. The goal of a proactive defense consists of making the models under attack more robust, while reactive defenses aim to identify attacks before they reach the model [35]. Different defensive countermeasures can be taken in each category. The remainder of this section will discuss some commonly used techniques [36].

Gradient masking techniques hinder optimization-based attacks by having gradients "*that are not useful*" [37]. They are also sometimes referred to as obfuscated gradients [38]. Three types of obfuscated gradients can be identified. Shattered gradients introduce incorrect or non-existent gradients. Stochastic gradients are caused by random effects in the defense and exploding or vanishing gradients are primarily caused by chaining neural network evaluations. Besides intentionally introducing gradient masking in neural networks, they can also be introduced unintentionally due to the design of the network.

Defensive distillation [39] is a technique that can be classified as gradient masking. The goal of defensive distillation is to smooth the gradients of the model, making it more resilient to small input perturbations. The distillation procedure is done in two steps. First the model is trained on the original data and labels. This step produces a probabilistic output for each input due to the softmax activation function. Then the network is retrained using the original data and the probabilistic outputs as labels. The probabilistic labels contain additional knowledge that can be exploited to increase generalizabilty of the model.

Adversarial training [30] techniques inject adversarial examples in the training dataset and retrain the model in order to create a more resilient model. This is essentially a brute force method to correctly classify some adversarial examples. However this new model is still susceptible to new adversarial attacks, since the decision boundary has only been moved slightly. This can be seen in Figure 2.6.

Preprocessing techniques work on the inputs of a model. Different preprocessing techniques, such as denoising [40], dimensionality reduction [41] and image transformations [42] can be used to defend the model under attack. The goal of all techniques

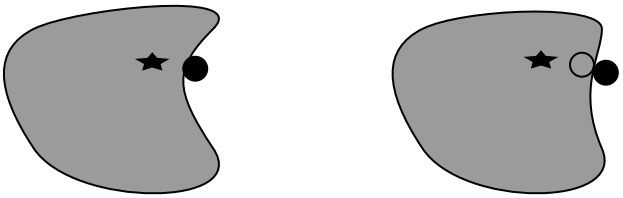


Figure 2.6: Decision boundaries before (left) and after (right) adversarial training. Before adversarial training, an adversarial example can be created (black circle). This example is added to the training set and the network is retrained. A new decision surface is created in the process. This surface classifies the previous adversarial example correctly (empty circle), but there is a new opportunity for an adversarial attack (black circle).

boils down to giving the attacker less control over the exact input that is being fed to the model.

Some defenses rely on **proximity measurements** between the input and the model. An example of this countermeasure is Deep k-Nearest Neighbors (DkNN) by Papernot and McDaniel [43]. DkNN computes support for a decision from a network based on a nearest neighbors search in the training data. Another example is region-based classification [44], where a prediction is made for a given input based on the proximity of training examples.

2.3 Particle swarm optimization

PSO [45] is an optimization framework part of the Evolutionary Algorithm (EA) family. In EAs, populations of candidate solutions evolve based on mechanisms inspired by the field of biology, such as ant colonies [46], mutation and recombination [47]. The mechanism that inspired PSO is the behaviour of flocks of birds. The framework has been applied to numerous problems such as routing problems [48, 49], diagnosing diseases from imaging [50] and calculating heat transfer coefficients [51].

In PSO, different particles x move through the search space based on a set of rules. Their new position x_t is determined by their previous position x_{t-1} and a velocity v_t. The velocity depends on the distance to best position of the swarm g and the best known position of the particle p. The distances can be weighted by acceleration coefficients c to put more emphasis on exploration or exploitation. These values are multiplied by random parameters r uniformly distributed in $[0, 1]$. The velocity is also dependent on the previous velocity with a corresponding weight w. The best position is determined using a fitness function. This function states how 'fit' or good a certain position is with respect to the goal of the optimization problem.

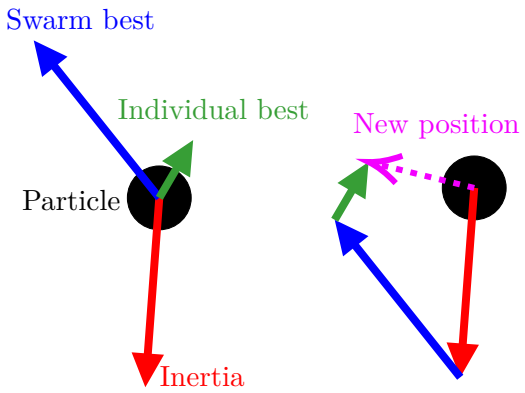


FIGURE 2.7: Particles move based on a step towards their best known position, the swarm's best known position and a step in the direction of the movement of the previous iteration. The different steps are combined to determine the new position of the particle.

Equations 2.1 and 2.2 correspond to the update rules. Each particle i has its own position $x_{t,i}$ and velocity $v_{t,i}$ at time step t, but the i index has been omitted for readability. In Figure 2.7, the steps are graphically represented for a single particle.

$$v_t = \underbrace{w v_{t-1}}_{\text{Inertia}} + \underbrace{c_p r_p (p_{t-1} - x_{t-1})}_{\text{Individual best}} + \underbrace{c_g r_g (g_{t-1} - x_{t-1})}_{\text{Swarm best}} \tag{2.1}$$

$$x_t = x_{t-1} + v_t \tag{2.2}$$

Ever since the first mention of PSO in 1995, efforts have been made in order to improve the framework. The rest of this section will discuss some improvements.

The first version of PSO used a fixed value for the inertia weight w. Later it has empirically been shown that a linearly decaying weight improves performance [52]. This allows the algorithm to focus on exploration in the early iterations, while shifting its focus to exploitation later on. The rate of decay depends on the high start value w_{start}, the lower end value w_{end} and the maximum number of iterations t_{max}. The weight in iteration t is calculated as follows:

$$w_t = w_{end} + (w_{start} - w_{end}) \left(1 - \frac{t}{t_{max}}\right) \tag{2.3}$$

In a large search space, particles can be far apart, which can cause the velocities to explode. By limiting the value of velocities to v_{max}, the swarm can be stabilized and the probability of finding an optimum is increased [45].

Clerc and Kennedy [53] studied PSO from a dynamic systems point of view. They provided a theoretically backed solution to the problem of the exploding velocities.

Instead of limiting the value of the velocity, something of which only empirical evidence has been given, they proposed the use of a constriction factor χ. The constriction factor is able to prevent the velocity explosion, whilst avoiding premature convergence to local optima. The velocity update of equation 2.1 is altered as follows:

$$v_t = \chi(v_{t-1} + c_p r_p(p_{t-1} - x_{t-1}) + c_g r_g(g_{t-1} - x_{t-1}))$$

The inertia weight w is dropped and the entire sum is multiplied by χ. The value of χ is calculated as follows:

$$\chi = \begin{cases} \sqrt{\frac{2\kappa}{C-2+\sqrt{C^2-4C}}}, & C > 4 \\ \sqrt{\kappa}, & \text{else} \end{cases}$$

Here C is the sum of the acceleration coefficients c_p and c_g and κ is a user defined value to determine the rate of convergence. Increasing κ slows down convergence, but causes a more thorough search to take place. The value of κ should be contained in the interval $]0, 1[$.

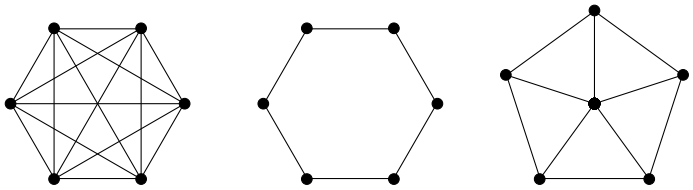
Another improvement on vanilla[2] PSO is Multi-Group with Random Redistribution (MGRR) PSO [55]. The swarm is split into two groups with opposite acceleration coefficients. The first group, with a larger c_p, is more attracted to the local optima, while the second group, with larger c_g, moves to the global optimum. The combination of the two groups accounts for more variation during the search process. This in turn aids in the convergence to the global optimum.

Even with the multi-group improvement, the particles can get trapped in local optima, as is the case in vanilla PSO. Whenever the swarm is stuck in a local optimum, half of the particles are randomly redistributed. The particles that will be redistributed will change approximately half of their values. The redistribution gives the algorithm a chance to escape the local optimum.

Experimental results show that MGRR-PSO has better performance than the vanilla variant [55]. The convergence is also faster and less dependent on the starting positions of the particles. Even the variant without the random redistribution outperforms vanilla PSO.

PSO is often combined with other algorithms in so-called hybridization techniques. The goal of this combination is to create an algorithm that contains the beneficial properties of both its constituents. Some commonly used algorithms combined with PSO are genetic algorithms [47], differential evolution [56] and ant colony optimization [46]. The hybridization is most commonly done in three different ways. In the first way, one algorithm is used before the other. The first algorithm performs optimization on the population used in the second algorithm. The second way divides the entire population into two groups and every group is optimized with a

[2]Vanilla is used to refer to software or algorithms not altered from their original form [54].



(a) Fully connected (b) Ring (c) Pyramid

FIGURE 2.8: Different communication topologies for the PSO framework. The particles are represented by the nodes and the edges represent the communication channels. Inspired by [58].

constituent algorithm. Some inter-group communication is required for this approach to work. The idea behind this technique is similar to the MGRR approach discussed earlier. The third way incorporates specific parts of an algorithm as a local search operator in the other. Thangaraj et al. [57] studied 64 hybridization algorithms containing PSO as one of the constituents. They found that the combination of differential evolution and PSO edges out the other hybrid approaches in terms of average error. However, all techniques outperform vanilla PSO using the error metric.

Another approach to prevent the particles of the swarm to converge to local optima is to restrict the communication inside the swarm. In vanilla PSO all particles have knowledge about the best position of the entire swarm. All particles will take a step towards this position in the next iteration of the algorithm. This causes the particles to converge to each other and limits the exploration of the search space. In order to restrict this knowledge, different communication topologies can be used. Some topologies are depicted in Figure 2.8. Here the nodes represent the different particles and the edges show the possibility of communication between these particles. The fully connected topology is the standard topology used in PSO. A ring topology limits communication of the best position to two neighbors per particle. The pyramid topology is similar to the ring topology, but there exists one particle that can communicate to all other particles. Neighbors in the topologies can be determined based on pre-defined indices or on the position of the particles in search space. The latter requires more computations. It has been shown that the best topology is highly problem dependent and that the size of the swarm plays a significant role in the choice of the best topology [58]. Combinations between the fully connected and ring topology are also possible and show promising results [59].

Chapter 3

Related work

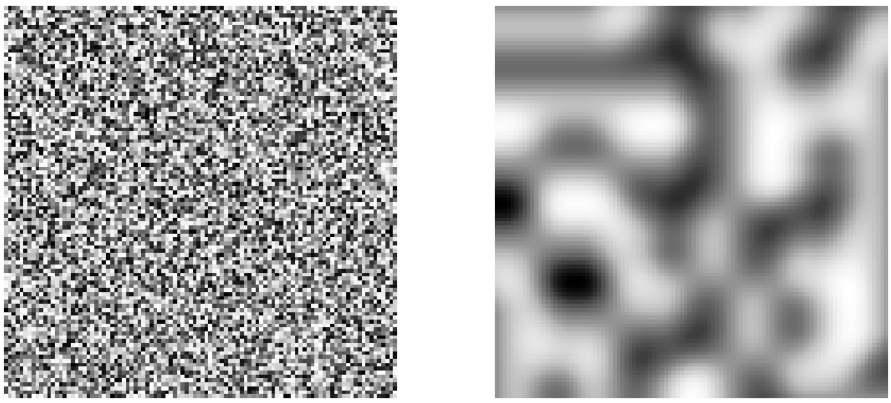
3.1 Boundary attack

Boundary Attack (BA) [60] is a decision-based adversarial attack. The basic intuition of BA differs from traditional adversarial attacks. Unlike these traditional adversarial attacks, where the original image is moved through search space in order to become adversarial, BA starts from an input that is already adversarial. This input is then moved closer to the original image, while staying adversarial.

The attack has to be initialized with an already adversarial input. Two different approaches can be taken depending on the attack setting. In the untargeted case, the input can be sampled from a maximum entropy distribution given the valid domain of this input. Samples that are not adversarial are rejected. An example of such a starting position can be seen in Figure 3.1a. In the case of a targeted attack, the input is a sample from the dataset that is classified as the target class by the model under attack.

BA iteratively updates the adversarial image by performing a step orthogonal to the original image and a step towards this image. In iteration k, a perturbation η_k is sampled from a uniform distribution. This perturbation is rescaled and added to the adversarial image. From this new position in search space, the step towards the original image is taken. This way the path of the attack follows the decision boundary, hence the name of the attack. The intuition of the BA is shown in Figure 3.3. The attack can only follow the boundary if the adversarial image is already near the boundary. The starting image is projected onto the boundary using binary search to ensure that the adversarial image is in the vicinity of the boundary.

The step sizes are adjusted according to local geometry of the boundary. The orthogonal step size δ is adjusted so that approximately half of the orthogonal perturbations is still adversarial. This approach is based on trust region methods [61]. The step size towards the original image ϵ is adjusted using the same principle, but here a user specified threshold is used. The decision boundary tends to become flatter, the closer to the original image the attack gets [62]. Therefore the algorithm converges



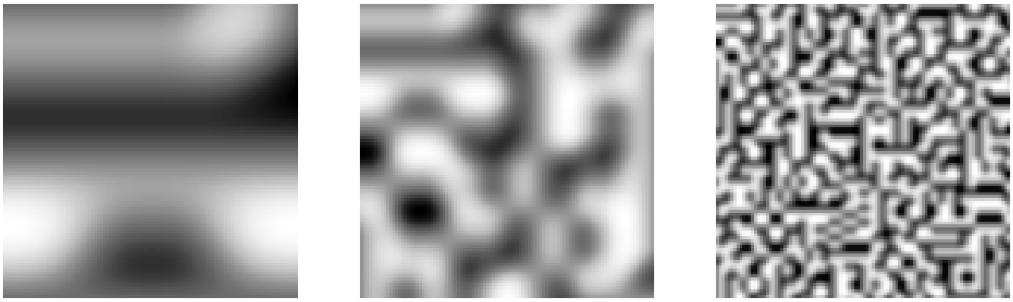
(a) Uniform noise (b) Perlin noise

Figure 3.1: Difference between noise patterns.

when ϵ converges to zero.

Biased Boundary Attack (BBA) [1] (previously known as *Boundary Attack++*) is an improvement on the original BA in three different ways. All three improvements will be discussed in order of the strength of their effect. The first improvement is a biased sampling technique. The key idea behind this strategy is that most previous attacks yield adversarial examples with high frequencies in the image. By sampling the perturbations in the first step of the BA from a low frequency distribution, the frequency of the created adversarial example will be lowered as well. BBA does this by sampling from a Perlin noise [63] distribution instead of a uniform distribution. Lower frequency images yield more natural results and can more easily bypass simple preprocessing defense schemes. The difference between the two noise patterns can be seen in Figure 3.1. The noise patterns can be influenced by a frequency value. This value can be tuned depending on the size of the images at hand. Higher frequency values yield less smooth noise patterns. Figure 3.2 visually shows the influence of the frequency values.

The second improvement is to use a regional mask. The original BA applies a perturbation to the images as a whole. Every pixel will be perturbed with the same magnitude. This magnitude can be altered on a per-pixel basis when using a mask. Pixels that are further away from the target image will receive a larger perturbation than pixels that are already close to the corresponding pixel in the target. The mask m is constructed according to equation 3.1 based on the original image x_{orig} and the adversarial image x_{adv}. It is then pixel-wise applied to the sampled perturbation in equation 3.2. The masked perturbation is normalized afterwards.



(a) Frequency 2 (b) Frequency 5 (c) Frequency 20

FIGURE 3.2: Influence of frequency on Perlin noise patterns.

$$m = |x_{adv} - x_{orig}| \tag{3.1}$$

$$\eta_k = m \odot \eta_k; \eta_k = \frac{\eta_k}{\|\eta_k\|} \tag{3.2}$$

This technique improves efficiency since the search space is significantly reduced. It is also possible to engineer masks for specific examples in order to incorporate other knowledge in the attack.

The final improvement is based on the idea of transfer attacks. A surrogate model is trained and will be used to calculate adversarial gradients. These gradients will then be used to bias the sampling direction for the orthogonal step. If the surrogate model does not closely resemble the defender, then the gradients will only hamper the speed of convergence of the attack instead of causing the attack to fail.

3.2 HopSkipJumpAttack

HopSkipJumpAttack (HSJA) [64], like BA, is a decision-based adversarial attack that starts from an adversarial input. The initial input is obtained in an identical manner as in BA. HSJA is an iterative algorithm that consists of three steps.

The first step is a projection onto the decision boundary of the model under attack. This projection is carried out using a binary search. The second step is to estimate the direction of the gradient at the boundary. Different directions are sampled from a uniform distribution over a d-dimensional sphere, where d is the input dimension. This random direction is added to the boundary point, generating a new query for the model. The results of these queries are combined to a gradient estimation $\widetilde{\nabla S}$ using the Monte Carlo estimate of equation 3.3. In this equation u_b are the random directions and x_t is the boundary position. B is the number of random directions that needs to be sampled. This number increases based on the current iteration of

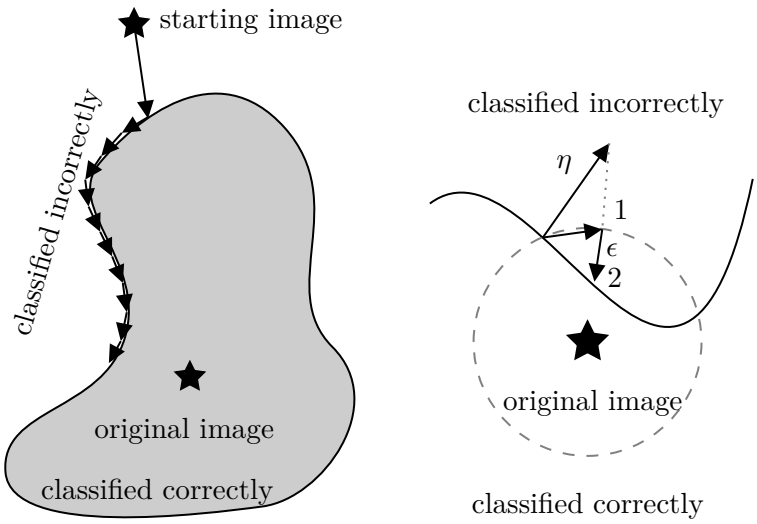


Figure 3.3: Intuition behind the Boundary Attack. On the left the path of the attack is shown. The first step is a projection onto the boundary, afterwards it follows the decision boundary of the class of the original image. Each arrow represents one iteration of the attack. On the right, the two different steps of each iteration can be seen. In the first step, a random direction is sampled and projected onto a sphere around the original image. The second step is to take a step towards the original image from this new position. Image inspired by [60].

the attack to reduce the variance of the estimate. The function ϕ_{x^*} returns 1 if the new position is adversarial and -1 if it is not adversarial. δ is a positive parameter determining the size of the d-dimensional sphere.

$$\widetilde{\nabla S}(x_t, \delta) := \frac{1}{B} \sum_{b=1}^{B} \phi_{x^*}(x_t + \delta u_b) u_b \tag{3.3}$$

Once the gradient has been estimated, the third and final operation is to take a step along this gradient. The step size is determined using a geometric progression scheme. These steps are iteratively repeated until the pre-set stopping criterion is met. Figure 3.4 represents the intuition behind HSJA in a graphical manner.

HSJA eclipses BA and BBA both on median distance against queries and attack success rates using a limited amount of queries. The untargeted version of HSJA is able to compete with white box attacks on the ImageNet dataset [65]. It also performs similar or superior to white box attacks such as the C&W attack [31] when evaluated against defensive mechanisms such as defensive distillation [39], region-based classification [44] and adversarial training [30].

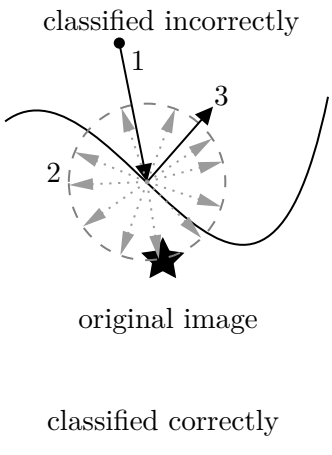


FIGURE 3.4: Intuition behind the HopSkipJumpAttack. Each iteration consists of three steps. The first step is a projection onto the boundary. The second step is the estimation of the gradient at this point. This is done by sampling directions from a uniform distribution and querying the model under attack from this new position (grey arrows). The results are combined via the Monte Carlo estimate. The third and final step is to take a step along the estimated gradient. Image inspired by [64].

3.3 Stateful defense

The defensive schemes discussed in section 2.2.2 all operate on the query level. They try to detect and flag possible attacks based on a single query without taking other context into account. The stateful detection mechanism by Chen, Carlini and Wagner [11] is different in this aspect. As the name suggests, it holds state of previously submitted queries. It is similar to the defenses that use proximity measurements, but the measurement is between queries instead of between the query and training data.

All queries submitted to the model equipped with a stateful detection mechanism are stored in a history buffer. Each user of the model has a distinct history buffer, where its queries are stored. These buffers can be bounded by time or number of queries depending on the resources available and the use case of the model. Each time a query is submitted to the model, the average distance to its k nearest neighbors is calculated and if this distance is lower than a certain threshold, then the user gets flagged by the mechanism. Appropriate actions such as banning the account can be taken.

The distance metric is not calculated in input space. Each query is encoded by a deep similarity encoder [66] to an encoded space, typically of a lower dimension. In this encoded space, images which represent perceptually similar objects are clustered together. A visual of the idea behind the similarity encoder is shown in Figure 3.5. The advantage of the encoded space is twofold. Firstly, the dimension of the encoded

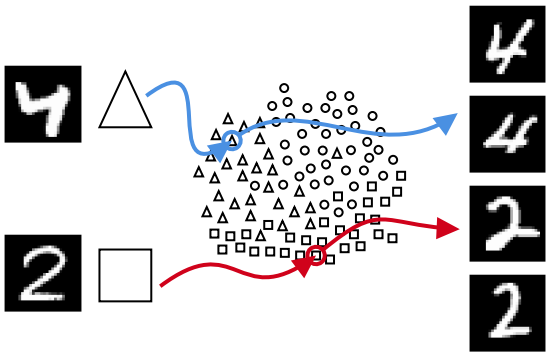


Figure 3.5: Visual representation of a deep similarity encoder. Input images (on the left) are mapped to an embedded space (of lower dimension). Images representing the same concept (squares, circles and triangles) are mapped close to each other in clusters. A nearest neighbor search around the embedding of an image will therefore yield similar images. The example images are taken from the MNIST dataset [2]. Image inspired by [66].

space is smaller than the dimension of the input space. Therefore less space is needed to store the history buffers. Secondly, simpler distance metrics such as L_2-distance in input space can easily be evaded by an attacker. For example the L_2-distance can be significantly increased by simply rotating or shifting the input image.

The parameter k, the number of neighbors to consider is picked as follows. As the training data of the model consists of only benign queries, no attacks should be flagged when feeding the stateful detection mechanism with this data. To allow for some more leniency, a false positive rate of 0.1% is still acceptable. For each value of k, a different threshold will be required to maintain the selected false positive rate. Larger values have the benefit of larger thresholds causing the defense to be more resilient, since attackers' images need to be more diverse. But k is also the number of queries needed before an attack can be flagged. Therefore too large values for k are disadvantageous. Smaller values also reduce computational cost. Chen, Carlini and Wagner set the value of k to 50 for the CIFAR-10 dataset [67], since the thresholds increased sharply up to this value. Other datasets might require different values for k.

3.4 PSO and distributed attacks

There have been several attempts to craft adversarial examples using an EA. Previous attempts tried to reduce the number of queries needed to create a successful adversarial example by utilizing EAs [68, 69, 70, 71, 72, 73].

GenAttack [68] and the similar efficient attack by Dong et al. [69] use genetic

algorithms in order to minimize the number of queries to the model. Both algorithms reduce the dimension of the search space to improve the efficiency of the attack. Once a promising perturbation is found in this lower dimensional space, it is upscaled using a bilinear transformation. By reducing the search space, the number of individuals in the genetic algorithm can be lowered, which in turn lowers the total amount of queries. GenAttack also uses annealing schemes to adaptively scale the parameters of the algorithm. This allows it to escape local optima and improve the adversarial example further.

AdversarialPSO [70] and the similar attack from [71] use PSO as optimization routine on images and audio fragments respectively. Each particle represents a possible adversarial example. Both attacks use the standard rules of PSO as specified by [45] improved with a linearly decaying inertia weight [74]. The former attack also uses a constriction factor to avoid premature convergence [53]. While the latter solves this problem by generating new particles using a genetic algorithm when premature convergence is detected. Both algorithms rely on confidence scores to assign fitness values to certain positions in the search space. PSO-BBA [72], is similar to AdversarialPSO, but only relies on distances to determine fitness values. This attack can therefore also be used in decision-based settings instead of solely in score-based settings.

The idea behind the multi-group PSO attack [73] is to use multiple PSO swarms to escape local optima. The intuition behind it is inspired by the Distributed Denial of Service (DDoS) attack [75]. The swarm is split into multiple smaller groups and each group is placed on a single node. The groups perform the standard PSO algorithm. The best position over all groups is communicated using a dedicated server. Each group submits its queries from its own node, tricking the defensive mechanism into thinking that multiple users are submitting queries. The authors state that this will ultimately result in less detections, but they have not evaluated this against a defensive scheme.

Chapter 4

Approach

The research concerning adversarial attacks and defenses is predominantly driven by a game of cat and mouse. Whenever a new attack is proposed, a defensive mechanism countering the novel attack is developed and vice versa. This work aims to create a new family of algorithms that can be used to perform both targeted and untargeted attacks. The goal of these algorithms is to craft adversarial examples comparable to state of the art approaches while remaining undetected by the stateful detection mechanism [11] described in section 3.3.

4.1 Distribution

The stateful detection mechanism [11] makes the assumption that queries can be traced back to their adversary and that there is no cooperation between different adversaries. This assumption can be problematic as N collaborating adversaries can theoretically reduce the number of submitted queries per adversary by a factor $1/N$. Even a single adversary could set up multiple accounts and submit queries on each account until it is banned. Due to the reduced number of submitted queries, less attacks will be detected since each buffer of the defense mechanism only holds a fraction of all queries.

This work will aim to evade the detection mechanism by distributing the query submissions over multiple nodes. Each node will represent a different user of the model under attack. The users could theoretically be all different persons or they could be different accounts of the same person.

As described in section 3.4, several attempts have been made to distribute adversarial attacks [72, 73]. However, none of these attacks have been evaluated against the stateful detection mechanism, since the goal of the distribution was to make the attack more efficient in terms of the distance between the original image and the resulting adversarial example. This work will distribute the query submission over multiple nodes in order to avoid detection.

4.2 Optimization

As previously mentioned, reducing the number of submitted queries per adversary by a factor $1/N$, where N is the number of collaborators, is straightforward. Adversaries can gain knowledge about the search space by cooperating with other adversaries. They can leverage this knowledge in order to reduce the number of submitted queries even more. This idea has been utilized by multiple algorithms that were mentioned in section 3.4. These algorithms used some form of PSO to optimize the final adversarial example. However all but the PSO-BBA algorithm by Xiang et al [72] rely on the confidence score of the model for the fitness value calculation. All attacks discussed use PSO as an attack in itself. This work will combine the benefits of state of the art black box attacks and PSO.

4.3 Approach

The remaining sections of this work will propose a new family of adversarial attacks. First a threat model is defined in section 4.4. All remaining experiments will be performed with this threat model in mind. Afterwards the novel adversarial algorithm is proposed in section 5.3. This algorithm is iteratively improved based on the results of the experiments. Finally, the final algorithm will be compared with state of the art decision-based attacks.

During the process of creating, improving and optimizing the attack, this work tries to answer the following research questions.

- What are the (dis)advantages of using PSO in relation to vanilla adversarial attacks?
- How can PSO be combined with state of the art adversarial attacks?
- What are the (dis)advantages of distributing an adversarial attack?
- How can adversarial attacks be made more evasive?

This work concludes with an answer on these questions in chapter 7.

4.4 Threat model

This section will describe the threat model that will be used for the remainder of this work. The first and most impactful assumption is that the model under attack will only output labels for the input. There will be no confidence scores or model parameters available. The proposed attack will have to be a decision-based attack. Most real world APIs will only expose the final decision to the user, causing decision-based attacks to be the only viable option. Decision-based attacks are the most restricted type of attack. Therefore they can also be applied to score-based

and white-box models.

The proposed attack will be a targeted attack, as it is the most relevant type from a security point of view. A targeted attack implementation can also easily be ported to an untargeted one. This is done by running a targeted attack for every possible class and selecting the one closest to the original image.

The model will be defended by by a stateful detection mechanism [11] as described in section 3.3. It will have a query bounded buffer for each account. Once an query has been flagged as potential attack, the account that submitted the query will be banned. The user will have to set up a fresh account in order to submit queries again.

There are two main fees associated with the model. The first fee is related to setting up an account. The assumption is made that setting up an account requires a valid credit card or phone number. Whenever an account is banned, the credit card or phone number is invalidated in the system, requiring the attacker to sign up for a new credit card or register for a new phone number. This cost is identical to the assumption made by Chen et al [11]. The second fee is related to the amount of queries submitted to the model. The more queries an attacker submits, the higher the total cost will be. Chen et al [11] did not incorporate a cost per query, but most vision APIs, such as Google Cloud Vision [76] and Amazon Rekognition [77] use this type of fee.

These fees ensure that attackers need to be both efficient and evasive in order to be successful. The evasiveness of the attack is correlated to the number of detections by the stateful defense mechanism. Each detection essentially means that the attacker has to set up a new account and pay the associated cost. To compare the efficiency of different attacks, the following strategy will be used. Every run of an attack has a budget of queries. The attack that has created an adversarial example closest to the original image inside this budget of queries, is the most efficient. The evasiveness of an attack is the primary concern, since the cost of setting up a new account is significantly higher than the cost per query.

Chapter 5

Evaluation

This chapter will discuss the experiments performed, as well as the ideas and intuitions behind them. The adversarial algorithm will be refined throughout the different subsections and the results of the refinement will be discussed at the end of each subsection.

5.1 Evaluation protocol

This subsection describes the evaluation protocol that will be followed for all experiments performed. Experiments will be performed with the MNIST [2] and CIFAR [67] datasets. Some examples of these datasets are visualised in Figures 5.1 and 5.2 respectively. A black box model is trained using the training data of the respective dataset. The architectures of the models are identical to the ones used in [31, 39]. A summary of the two architectures can be found in Table 5.1. The models are implemented using the Keras library [78]. The training parameters are also identical to the ones used in [31, 39]. These models remain unchanged for all experiments.

The trained models are then used to classify all instances in the test set of their respective dataset. The incorrectly classified examples are filtered out of this set as they are essentially already adversarial. A list of experiments is generated based on the remaining examples. An experiment consist s of an original image, a target label and starting position(s). All future refinements will therefore perform the same set of experiments in order to make the comparison more fair. All random effects present in the algorithm are seeded for the same purpose.

The stateful defense mechanism [11] will use a query bounded detector buffer of 1000 queries per user and the value of k will be set to 50, as suggested by the authors of the paper. A threshold is determined in order to achieve a 0.1% false positive detection rate on the training data. The detector buffer will be cleared after each detection to simulate the creation of a new account. The mechanism will have a similarity encoder with an output dimension of 256. The detailed architecture of the

Figure 5.1: Some examples of the MNIST [2] dataset.

Figure 5.2: Some examples of the CIFAR-10 [67] dataset.

Table 5.1: Model architectures for the MNIST and CIFAR models. The architectures are identical to [31, 39].

Layer type	MNIST Model	CIFAR Model
Convolution + ReLU	$3 \times 3 \times 32$	$3 \times 3 \times 64$
Convolution + ReLU	$3 \times 3 \times 32$	$3 \times 3 \times 64$
Max Pooling	2×2	2×2
Convolution + ReLU	$3 \times 3 \times 64$	$3 \times 3 \times 128$
Convolution + ReLU	$3 \times 3 \times 64$	$3 \times 3 \times 128$
Max Pooling	2×2	2×2
Fully Connected + ReLU	200	256
Fully Connected + ReLU	200	256
Softmax	10	10

Table 5.2: Model architectures for the MNIST and CIFAR similarity encoders. Both datasets use the same architecture.

Layer type	Size
Convolution + ReLU	$3 \times 3 \times 32$
Convolution + ReLU	$3 \times 3 \times 32$
Max Pooling	2×2
Convolution + ReLU	$3 \times 3 \times 64$
Convolution + ReLU	$3 \times 3 \times 64$
Max Pooling	2×2
Fully Connected + ReLU	512
Fully Connected + linear	256

TABLE 5.3: Results for the BBA baseline approach on MNIST and CIFAR-10 dataset.

	MNIST		CIFAR	
Attack	Distance	Detections	Distance	Detections
Baseline BBA	2.807	413	1.306	474

encoder is shown in Table 5.2 and is identical for both datasets.

The algorithm receives a budget of 25 000 queries to create an adversarial example. The final adversarial example will be evaluated on the L_2-distance to the original image and the total number of detections by the stateful defense mechanism.

5.2 Determining the baseline

The goal of this subsection is to determine a baseline to which all future algorithms will be compared. The baseline will be a vanilla BBA using the same hyperparameters as suggested in [1]. The orthogonal step will be set to 0.05 and the source step to 0.002. Both the Perlin noise improvement and the regional masking improvement will be used. No gradients of surrogate models will be calculated as the authors of the paper already mentioned that the improvement was marginal.

Table 5.3 reports the average L_2-distance and the average number of detections for the different datasets.

5.3 Applying PSO to BBA

As mentioned in section 4.2, previous attempts to craft adversarial examples using PSO used PSO as a standalone algorithm to guide adversarial examples through the search space closer to the original image. This work aims to combine the benefits of PSO and an existing decision-based attack such as BBA.

Vanilla BBA has the disadvantage that, depending on the starting position, it might get trapped in a local optimum. By using PSO in combination with BBA, multiple starting positions can be explored and the probability of getting trapped is lowered. The intuition behind this idea is shown in Figure 5.3. The more starting positions there are, the higher the probability of finding the global optimum. There is however a clear trade-off in terms of efficiency. By having n different starting positions, the query budget is essentially reduced by a factor n for each starting position.

The efficiency reduction does not have to pose a problem due to the implicit communication in the swarm. Particles can move to promising regions in the search space based on the information of their peers. The promising regions are therefore more

queried.

The proposed PSO-BBA algorithm works as follows. Particles will perform a more aggressive version of BBA. The initial source step ϵ is set significantly higher than in the vanilla version. The increased source step might cause the particle to end up in a non-adversarial decision region. Once this happens, the standard PSO equations (2.1 and 2.2) are used to guide the particle back to the adversarial region. Later refinements of this algorithms will deal with attack iterations. An attack iteration is defined as a specified budget of the aggressive BBA (50 queries in all experiments) or a single query submission using the PSO rules.

The inertia weight of the PSO equations is determined by the linearly decaying scheme of equation 2.3, with the weight decaying from 1 to 0. The acceleration coefficients c are set based on an idea of multi-group PSO. Two groups with opposite acceleration coefficients are created. This approach helps escape local optima [55]. The equations for both c_p and c_g are:

$$c_p = \begin{cases} \max(A1, A2), & \text{if } i \bmod 2 = 0 \\ \min(A1, A2), & \text{else} \end{cases} \tag{5.1}$$

$$c_g = \begin{cases} \min(A1, A2), & \text{if } i \bmod 2 = 0 \\ \max(A1, A2), & \text{else} \end{cases} \tag{5.2}$$

Here i is the index of the particle in its swarm. The values of $A1$ and $A2$ are 1 and 2 respectively. These values are suggested by the authors of [73].

The source step ϵ will be changed after every iteration of the attack. Two separate multipliers are used to respectively increase and decrease the value of this parameter. The value of ϵ is slightly increased if the new position is still adversarial. Likewise it is decreased if the position is no longer adversarial.

By using PSO in combination with BBA, the advantage of multiple starting points, as explained in Figure 5.3, can be exploited, without having a less efficient attack as a whole. Whenever particles end up in non-adversarial regions, they will move closer to the best known position in the swarm due to the PSO equations. At then end of the attack, most particles will be in the same area of the search space, allowing for more exploitation in this specific area.

The PSO framework requires a fitness function to quantify the fitness of a position for the problem at hand. The authors of AdversarialPSO [70] suggested the following fitness function f:

$$f(x) = |p_x - p_{x'}| - \frac{c}{n}\|x - x'\|_2 \tag{5.3}$$

Where x is the position of the particle, x' is the original image, p_x and $p_{x'}$ are the confidence scores of the model in predicting the label of x and x' respectively and c is a constant to weight the penalty. However, as discussed in section 4.4, the confidence scores are not available for this specific attack. The fitness function in equation 5.3 also assumes that the position x will always be adversarial. This will not be the case in the proposed algorithm. The fitness function will therefore be altered to the following:

$$f(x) = \begin{cases} \|x - x'\|_2, & \text{if } x \text{ is adversarial} \\ +\infty, & \text{else} \end{cases} \tag{5.4}$$

The infinite value for the fitness function inside non-adversarial decision regions acts as a penalty, causing the particles to quickly diverge from these regions.

The same set of experiments as in section 5.2 has been performed. The experiments have been done using attacks with both five and ten particles. The initial step sizes have been set to 0.25 and 0.20 for MNIST and CIFAR respectively. The values for the increasing and decreasing multiplier have been set to 1.05 and 0.99 for both datasets. These values have been selected at random and were tested on a small subset of the experiments in order to confirm their effectiveness. Other values will be tested in section 5.6. The results of the experiments can be found in Table 5.4.

The five particle PSO-BBA algorithm outperforms the baseline in terms of distance to the original image on both datasets. This is not the case for the ten particle version. The high number of particles requires sufficient queries in the beginning of the attack in order to discover promising regions in the high dimensional search space of CIFAR. This exploration requires more queries than the query budget allows. The number of detections is lower for all variants of PSO-BBA compared to the baseline. The different starting points have the added advantage that initial queries are more spread out over the search space. These queries are therefore less similar and the detector will not flag as much attacks. This effect can be seen in Figure 5.4 for the MNIST experiments.

The number of detections drops as the number of particles increases. This was to be expected from the intuition behind PSO-BBA. However, the average L_2-distance to the original image is higher for ten particles compared to five. To reduce the number of different parameters in future refinements, only swarms with five particles will be considered from here on.

5.4 Towards distribution of the attack

The stateful defense mechanism [11] makes the assumption that there is no collaboration between adversaries. Based on this assumption, each account or user will have its own detector buffer. This subsection aims to exploit this assumption by

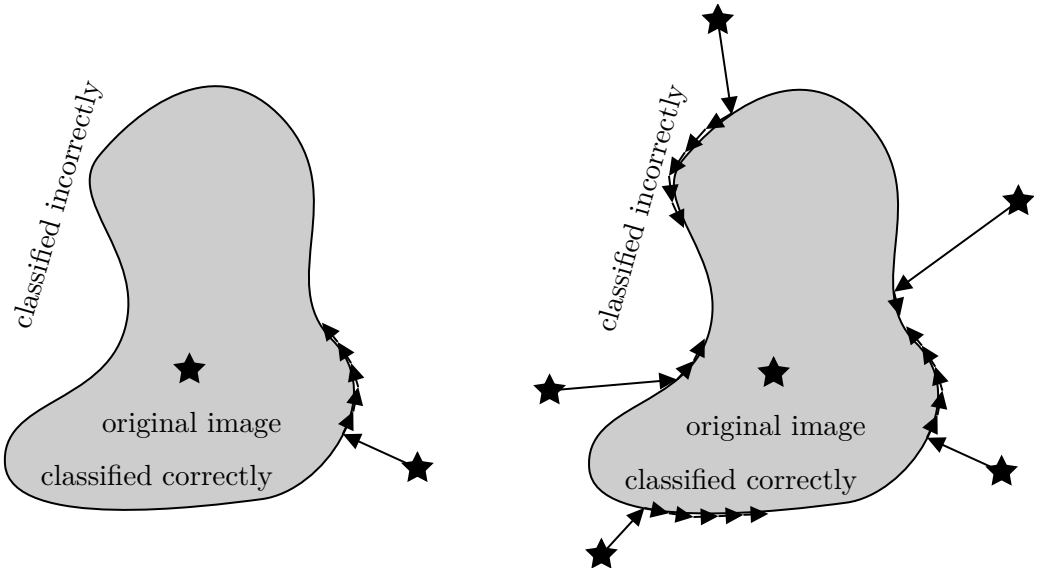


Figure 5.3: The vanilla version of BBA might get stuck in a local optimum depending on the starting point (left plot). By starting from multiple positions, the probability that BBA gets stuck in a local optimum is reduced (right plot). The multiple starting points are particles in a PSO swarm. Image inspired by [60].

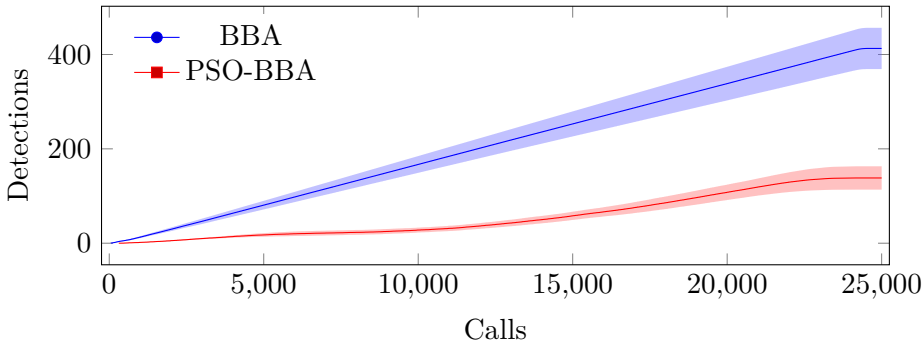


Figure 5.4: The cumulative number of detections for different attack algorithms for the MNIST experiments. The number of detections of the BBA algorithm steadily increases with the number of calls. The number of detections of the PSO-BBA algorithm increases more slowly in the beginning due to the dispersed nature of the attack. The increase is more sharp near the end of the attack when the particles converge. The plot shows the 95% confidence regions around the graphs.

TABLE 5.4: Results for the PSO-BBA approach on MNIST and CIFAR-10 dataset.

	MNIST		CIFAR	
Attack	Distance	Detections	Distance	Detections
Baseline BBA	2.807	413	1.306	474
PSO-BBA (5 particles)	**2.712**	138	**1.239**	257
PSO-BBA (10 particles)	3.157	**44**	2.290	**184**

distributing the query submission over multiple accounts.

By distributing the query submissions over multiple accounts, the number of total detection should be reduced compared to BBA due to two reasons. The first reason is obvious. If a budget of 25 000 queries is distributed over N accounts, then every account will submit 25 000 / N queries. The less queries there are submitted, the less potential attacks will be detected. The second reason is due to PSO. As stated before, the different particles reside in different parts of the search space. This means that a detector buffer of a specific account will contain queries from all over the search space, causing the inter query distances to be larger. The latter reason was also present in the algorithm described in section 5.3.

It should be noted that there is only distribution at the query submission level. The algorithm itself will be executed on one machine without the need for distribution. Even the query submission can be done from one machine by constantly changing the API key or by logging in and out of a specific account whenever it is needed. This approach makes the assumption that the API under attack does not track the IP address of the machine that submitted the query. Multiple machines might therefore be more convenient in a real attack setting.

This work simulates the multiple machines setting by having separate node (or machine) objects on the same machine. Instead of having a detector buffer for each node at the location of the model, the responsibility is moved upstream to the nodes themselves. Each node will have a buffer to which the query will be added. All queries will be passed to the model afterwards. In Figure 5.5, the distribution is shown schematically.

The queries will be distributed over the nodes based on a distribution scheme. Three different distribution schemes will be used in the experiments. The first two schemes are inspired by the Round-Robin (RR) scheduling algorithm [79] and will be called the RR and Modified Round-Robin (MRR) distribution schemes. The third distribution scheme will be built on the assumption that the inter query distance in the detector buffers needs to be maximized.

The RR distribution scheme is fairly straightforward. The outgoing queries of the

algorithm will be evenly distributed over the nodes in circular order. This scheme does not have a notion of the underlying attack. This can cause successive queries forwarded to the same node to be relatively close together by chance.

The MRR distribution scheme maps a particle to a node for the entire duration of an attack iteration. The queries submitted by a single particle during one attack iteration are relatively close together, but this approach ensures that all particles will submit queries to all nodes. This might ultimately cause less detections. The MRR distribution scheme maps a number of particles on a number of nodes. If the amount of particles is equal to the number of nodes then the mapping is straightforward. If the number of nodes is less than the number of particles then dummy nodes are introduced in the mapping. The dummy nodes forward their received queries to a random existing node. The random node changes every rotation. If the number of particles is less than the number of nodes then dummy particles that do not submit queries, are introduced. After every attack iteration, the mapping rotates. The mapping process is shown in Figure 5.6.

The ultimate goal of the attack is to evade the stateful detection mechanism that defends the model under attack. Potential attacks are flagged based on the inter query distance in a detector buffer. The distance-based distribution scheme tries to exploit the defense. Instead of distributing queries over nodes in a RR-fashion, queries are submitted to nodes based on the distance to the previously submitted queries to this node. The node with the highest average distance to the previously submitted queries is selected. For this purpose, every node will hold an internal buffer of queries. The size of this buffer can be chosen by the attacker, as well as the distance metric. Standard L_2-distance can be chosen, but this has the same pitfalls as discussed in section 3.3. It is also possible to train another similarity encoder and use the L_2-distance in the embedded space in order to resemble the defense more closely. The distribution schemes will be called Distance-Based (DB) and Embedded-Distance-Based (EDB) distribution schemes respectively.

The experiments performed in this section will use a buffer size at each node of 20 for both the DB and EDB distribution schemes. The similarity encoder in the case of the EDB scheme will be trained on the test data of the respective dataset and has an output dimension of 128.

The results shown in Table 5.5 reveal that the different distribution schemes have similar performance in terms of the number of detections. The distance is identical for all distribution schemes since the underlying algorithm is the same. They only differ in the way that the queries are distributed over the nodes. The ten node distribution is always better than the five node version. An attacker has to make a trade-off between the number of nodes and the number of detections (and consequently accounts). More nodes require trigger less detection and therefore require less accounts. Depending on the cost of nodes and accounts, the optimal amounts might differ.

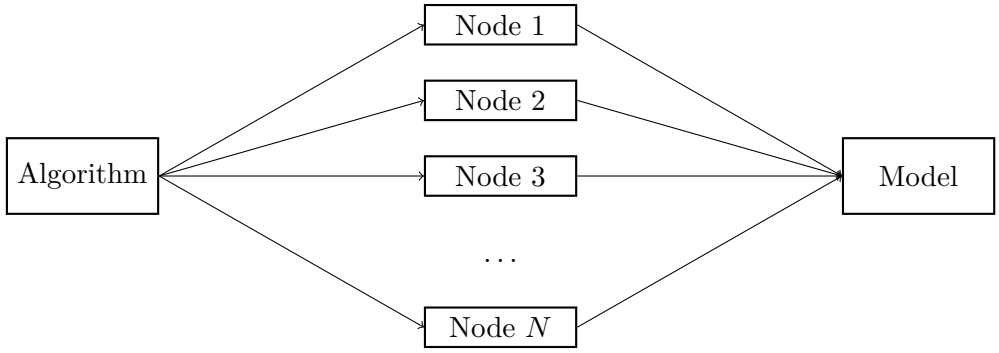


FIGURE 5.5: Schematic overview of the query submission distribution. Each arrow represents a query. The algorithm distributes the queries over the different nodes. Every node forwards its received queries to the model using the corresponding account. In a real setting, the model will have a detector buffer for every account. In this work the buffers are located on the nodes themselves.

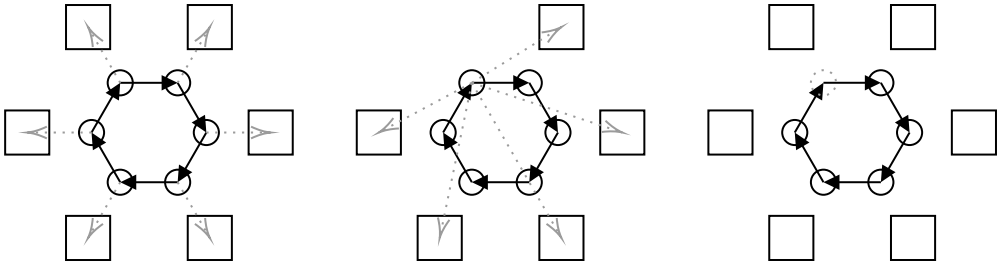


FIGURE 5.6: Overview of the modified round-robin based distribution scheme. The particles are represented by the small circles and the squares represent the different nodes. If the number of particles is equal to the number of nodes (left), then particles submit their queries to the corresponding node. If the number of nodes is less than the number of particles (middle), then the particles without nodes will submit their queries to a random node. This random node will change every attack iteration. If the number of particles is less than the number of nodes (right), then dummy particles are introduced. These dummy particles will not submit any queries, therefore some nodes will not receive queries. After every attack iteration the particle mapping will rotate.

As can be concluded from Table 5.5, there is no real gain in having a complicated distribution scheme. The RR distribution scheme will consequently be used for the remaining experiments since the distance based schemes have more computational overhead. These experiments will also be performed using ten nodes, unless mentioned otherwise.

Table 5.5: Results for the distributed PSO-BBA approaches on MNIST and CIFAR-10 dataset. There is no real advantage of one distribution scheme over the others. Increasing the number of nodes however, does have a clear advantage. The distribution over ten nodes is flagged less than the five node distribution for all schemes. The DB scheme performs best for the MNIST dataset, while the RR scheme is the best performer for the CIFAR dataset. Their respectie values are shown in bold in the table.

		MNIST		CIFAR	
Attack		Distance	Detections	Distance	Detections
Baseline BBA		2.807	413	1.306	474
PSO-BBA		2.712	138	1.239	257
RR-PSO-BBA	(5 nodes)	2.712	124	1.239	224
MRR-PSO-BBA	(5 nodes)	2.712	110	1.239	227
DB-PSO-BBA	(5 nodes)	2.712	107	1.239	230
EDB-PSO-BBA	(5 nodes)	2.712	108	1.239	229
RR-PSO-BBA	(10 nodes)	2.712	104	1.239	**202**
MRR-PSO-BBA	(10 nodes)	2.712	93	1.239	206
DB-PSO-BBA	(10 nodes)	2.712	**87**	1.239	207
EDB-PSO-BBA	(10 nodes)	2.712	88	1.239	205

5.4.1 Adding more nodes to the equation

It is obvious that increasing the number of nodes makes the attack more evasive without an added efficiency cost. In the most extreme scenario, every query will be submitted using its own designated node. This ultimately causes no detections to occur. In fact, this scenario is a complete overkill. A detection is based on the average distance to the k nearest neighbors in the detector buffer associated with the account submitting the query. This means that the defense has to build up a history of k queries before any detection can occur. Only the $(k+1)$th and subsequent queries can therefore be flagged[1]. Every adversarial attack that is distributed in the same manner, therefore only requires $\lceil budget/k \rceil$ nodes in order to remain undetected, where *budget* represents the query budget.

The combination of PSO and BBA as described in section 5.3 requires less nodes in order to be completely evasive. On the MNIST dataset, 25% of the adversarial attacks does not trigger any detections when distributed over 25 nodes. For 50 nodes, already 60% of the attacks are completely evasive and all attacks are completely evasive when using 100 nodes. The CIFAR-10 dataset requires more nodes to be

[1]This limitation should not have a big impact on the defense against current state of the art methods, as HSJA is only 70% successful using 1000 queries [64] and the proposed value for k is 50 in [11]. In [11], being successful is defined as a maximum of 8/255 difference in L_∞-distance between the original and the adversarial example.

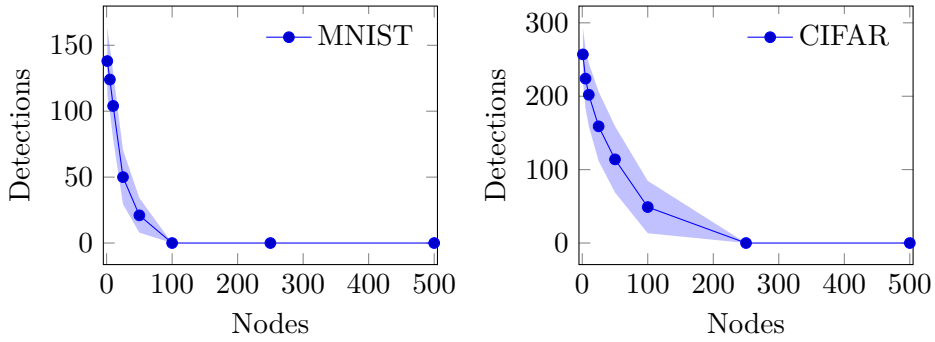


FIGURE 5.7: The number of detections for both datasets, when distributed over a given number of nodes. The number of detections decreases with increasing number of nodes. No attacks are flagged at 100 and 250 nodes for the MNIST and CIFAR dataset respectively. At 500 nodes, no detections can be flagged due to the theoretical limitation of the stateful defensive mechanism. The 95% confidence regions are shown around the curves. Note that the y axes do not line up.

completely evasive. When using 100 nodes, 60% of the attacks do not trigger any detections. If the node count is increased to 250 then no detections occur. Figure 5.7 shows the detection count in function of the number of nodes for both datasets. When the attack is distributed over 500 nodes, no detections are reported due to the limitations of the stateful defense mechanism.

5.4.2 Modifications to the distance based schemes

An improvement on the DB and EDB schemes can be made based on the assumption that the attacking user has knowledge about the status of its account. If the user receives a notification whenever its account is banned, then this knowledge can be incorporated in the distribution scheme. The standard DB and EDB schemes distribute queries based on the distance to the most recently submitted queries from this node. If an attack is flagged based on a submitted query, then this means that the corresponding account is banned and a new account has to be set up on the node. The new account will have a fresh detector buffer, therefore the previous queries submitted from this node are no longer relevant. By flushing the list of previously submitted queries whenever an account is banned, the distribution can happen more effectively. This distribution scheme will be called the Flushing-Embedded-Distance-Based (FEDB) distribution scheme. The results compared to the EDB scheme are shown in Table 5.6.

The FEDB scheme is more evasive than its non-flushing counterpart, both for five and ten nodes. However the gain is marginal. It is also based on the assumption that the attacker is notified of its account being banned. This might not always be the case. Users might be shadow banned [80] momentarily or even permanently, which

Table 5.6: Results for the FEDB distribution scheme compared to EDB on the MNIST and CIFAR-10 dataset. The FEDB scheme outperforms its non-flushing counterpart by a slight margin.

		MNIST		CIFAR	
Attack		Distance	Detections	Distance	Detections
Baseline BBA		2.807	413	1.306	474
PSO-BBA		2.712	138	1.239	257
EDB-PSO-BBA	(5 nodes)	2.712	108	1.239	229
FEDB-PSO-BBA	(5 nodes)	2.712	105	1.239	224
EDB-PSO-BBA	(10 nodes)	2.712	88	1.239	205
FEDB-PSO-BBA	(10 nodes)	2.712	**84**	1.239	**197**

Table 5.7: Results for the FEDB distribution scheme for different buffer sizes on the MNIST and CIFAR-10 dataset. All experiments are performed using ten nodes. Increasing the buffer size has a slight advantage in terms of detections.

		MNIST		CIFAR	
Attack		Distance	Detections	Distance	Detections
Baseline BBA		2.807	413	1.306	474
PSO-BBA		2.712	138	1.239	257
FEDB-PSO-BBA	(buffer size 1)	2.712	91	1.239	208
FEDB-PSO-BBA	(buffer size 20)	2.712	84	1.239	197
FEDB-PSO-BBA	(buffer size 50)	2.712	**82**	1.239	**194**

violates the assumption[2].

The distance based distribution schemes can be greatly impacted by the length of the history they take into account. Longer buffers require more computational overhead, but the best node can be chosen with higher accuracy. Table 5.7 shows the results for different buffer lengths for distribution over ten nodes. Increasing the buffer, lowers the number of detections. However, the extra computations almost triple the run-time of the attack using a buffer size of 50 compared to a buffer of length 20. A buffer size of 20 is therefore an ideal middle ground between evasiveness and computational cost.

[2] A shadow ban detection algorithm should be implemented at the attack's side in case of a permanent shadow ban.

5.5 Throwing the defense off the scent

As previously mentioned, the two goals of the proposed attack are conflicting. In order for the attack to be efficient, subsequent queries will be clumped together in search space, causing the evasiveness of the attack to drop. This is an important trade-off that has to be made depending on the type of attack, cost of detection, efficiency thresholds and many other criteria. In this subsection changes will be made to the algorithm to be more evasive at the cost of efficiency.

Remember that the model under attack holds a query-bounded detector buffer per account. The buffer contains a history of the last submitted queries by a specific account. Every time a new query is inserted in the buffer, the distances to their nearest neighbors is determined and averaged. In the case of an attempted attack, the nearest neighbors will be relatively close together, causing the average distance to fall below a set threshold. Appropriate actions, such as banning the account, can now be taken.

The key idea behind the approach to increase the evasiveness is the following. Benign queries can be submitted by the attacker in order to flood the buffer with noise. The insertion of the benign queries will cause the average inter query distance to be raised, which in turn will lower the number of detections.

The approach raises two questions that will need to be answered: what sort of noise should be inserted and when should it be inserted. The former question will be answered first. Five types of noise are considered. The first one being uniform noise, as has been depicted in Figure 3.1a. The second and third options are Perlin noise with a fixed frequency value and a random frequency value respectively. Perlin noise has been shown in Figure 3.1b, while Figure 3.2 shows the influence of the frequency on the Perlin patterns. The fourth option will be random samples from the test data of the dataset on which the attack is being performed. The fifth and final option is a combination of the previous four strategies. Every time a benign query has to be inserted, one of the four options is chosen at random.

A similarity encoder will encode the submitted queries in an embedded space. The detector buffer will calculate the distances in this embedded space. Therefore it makes sense to determine the strategy with the highest inter query distance in the embedded space. The following experiment has been set up in order to establish the best strategy. The similarity encoder used in the EDB distribution scheme is reused to encode the benign queries. A fixed amount of queries (20 000) is encoded and added to a detector buffer of size 1000. Every time a query is added, the nearest neighbors are determined and the average distance to these neighbors is calculated. These distances are finally averaged over all submitted queries. The results of this experiment can be found in Table 5.8.

The most promising noise strategy is the test data strategy. This makes sense as the

Table 5.8: The average inter query distance (L_2-distance in embedded space) for different noise strategies. Higher values correspond to better strategies as a detector is less likely to flag a potential attack.

	MNIST	CIFAR
Uniform noise	0.140	0.023
Perlin noise (fixed frequency)	0.110	0.004
Perlin noise	0.108	0.004
Test data	0.212	**0.265**
Mixed noise	**0.238**	0.198

similarity encoder is trained to maximize the distance between images representing different concepts. The uniform and perlin noise patterns are probably very similar to the encoder, although they might be further apart in terms of L_2-distance. This causes these patterns to reside in the same regions of the embedded space. The mixed strategy is another promising candidate. It even outperforms the test data strategy on the MNIST dataset. However this is likely due to a large distance between the patterns and test data clusters in the embedded space as the mixed strategy triggers more detections (3056) compared to the test data strategy (0). For this reason, test data will be chosen as the source for the noise queries.

Now all that remains is answering the second question: when should the noise be inserted? Three strategies were considered. The first strategy is insertion of noise queries every nth query. The second strategy is inspired by the conclusion drawn from Figure 5.4. Detections are more common near the end of the attack, when the different particles have converged to the same region in search space. Therefore the second approach makes use of a geometric progression scheme [81] in order to submit more noise queries later in the attack progress. The third and final strategy does not have a fixed interval between the submission of noise queries. The submission is based on the distance to the previous queries. Using the EDB distribution scheme, a node is selected based on the average distance to the queries previously submitted from this node. If this average distance falls below a certain threshold then it is likely that an attack will be flagged by this query. Therefore noise queries are inserted before submitting the query used to progress the attack.

The first strategy does not seem a very good fit on paper. There is no good value for the submission interval n that works for the entire duration of the attack. Smaller values will result in noise being inserted too often, especially in the beginning of the attack when detections occur less frequently. Larger values will not help towards the evasiveness of the attack as the noise queries are inserted too infrequently. The second strategy relies on two hyperparameters, the starting value of the noise submission interval n and the decay rate. The optimal values of these parameters are very problem specific. For these reasons, the first two approaches are dropped from the evaluation.

TABLE 5.9: Results for the EDB distribution scheme with noise insertion for threshold values on the MNIST and CIFAR-10 dataset. All experiments are performed using ten nodes and a buffer size of 20.

		MNIST		CIFAR	
Attack	Threshold	Distance	Detections	Distance	Detections
Baseline BBA	/	2.807	413	1.306	474
PSO-BBA	/	2.712	138	**1.239**	257
EDB-PSO-BBA	/	2.712	88	**1.239**	205
EDB-PSO-BBA	0.10	**2.710**	87	1.537	208
EDB-PSO-BBA	0.25	2.719	87	1.622	**202**
EDB-PSO-BBA	1.00	2.720	**83**	1.595	221
EDB-PSO-BBA	2.00	2.775	89	1.592	241

Experiments will be performed using the third strategy. Noise queries will be inserted based on the mean distance of an incoming query to the previously submitted queries. Whenever this mean distance drops below a set threshold, a noise query will be inserted. The results of the experiments can be found in Table 5.9 for different threshold values. The experiments are performed using a EDB distribution scheme over ten nodes with a buffer size of 20. Lower values for the threshold cause less noise queries to be inserted, while higher value have the opposite effect. When only taking the number of detections into account, there seems to be an optimal dataset-dependent value. However, the noise insertion hampers the efficiency of the attack. Only one datapoint is more efficient than the algorithm without noise insertion due to random effects. All other datapoints are less efficient. For the CIFAR dataset, the attack with insertion performs worse than the vanilla BBA for all threshold values. The small gain in evasiveness is not worth the loss in efficiency, therefore no further experiments have been performed.

The queries inserted in the experiments performed in the previous paragraph are merely noise queries. They have no purpose in progressing the attack, only to evade the detection algorithm. What if the queries that are inserted to evade detection could also serve towards progressing the attack? The attacker would not have to worry about the query budget since all submitted queries would be useful. The idea behind this is based on the assumption that an adversary is often interested in multiple adversarial examples instead of just the one. If this is the case, multiple attacks can be performed concurrently and their respective queries can be submitted in an interleaved manner.

The same experiments are performed again. This time experiments are grouped together. Experiments in the same group are executed concurrently and their queries that have be be submitted are interleaved. The query budget per experiment is

respected, meaning that a group of size s will submit s times the query budget. The interleaved queries are distributed over different nodes as if they came from a single attack.

Figure 5.8 shows the average number of detections per experiment. This means that the total number of detections of a group of size s is divided by s, which allows for an easier comparison to the baseline. Increasing the group size s yields a lower number of detections per experiment. The only exception to this conclusion is a group size of two for the CIFAR dataset. This effect is likely due to some unlucky combinations in the experiment pairings. Experiments with the same original label or same target label can cause more detections due to the attacks exploring the same regions in the search space. This effect is still present in the higher group sizes, but less prominent since the probability of all attacks exploring the same regions is reduced. The average L_2-distance of the resulting adversarial examples is again equal to the results obtained by the PSO-BBA attack, since the only difference is the order in which the queries are submitted. All experiments performed to produce this plot used the RR distribution scheme over ten nodes.

In a real scenario, the attacker would benefit from interleaving the query submission of all planned adversarial attacks. There is an added benefit when the group size s is high enough. Since the detector buffer of the defensive scheme can only hold a certain number of queries, it should be possible that every individual attack only has a few of its queries at a time in this buffer. If this number is below the number of considered neighbors k, then no attack should be flagged even using a single node or account. This might not be possible with the MNIST or CIFAR datasets due the limited number of target classes, causing the individual attacks to help detect each other. This effect can be seen on the CIFAR dataset with group size two in Figure 5.8. But when performing attacks on datasets such as ImageNet [65], where the number of classes is much higher, this scenario is not completely unthinkable.

5.6 Optimizing the attack

There are numerous parameters that influence both the efficiency and evasiveness of the attack. Some parameters are assigned values based on suggestions by the original authors of the work, some receive values based on experiments performed in this work and others get their respective value based on an educated guess. All parameters influence each other, making tuning of a single parameter pointless.

This section will focus on optimization of a small parameter set. Due to the complexity of this task, the goal is not to find the optimal value for all parameters in this set. But rather to find any obviously incompatible combinations of parameters if these are present. For the sake of completeness a list of all possible parameters and their respective default values is given in Table A.1 of appendix A. Table A.2

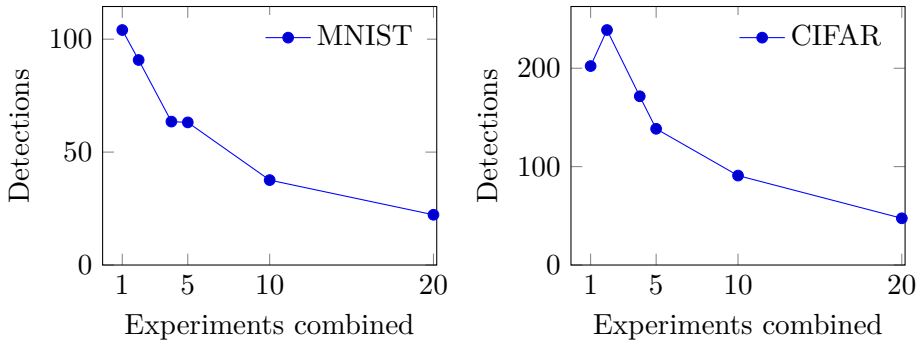


FIGURE 5.8: The average number of detections for both datasets, when experiments are combined and distributed over ten nodes. Experiments are run individually but their queries have to be submitted are interleaved. Increasing the number of experiments that are combined, decreases the average number of detections per experiment. The only data point inconsistent with this conclusion is two combined experiments for the CIFAR dataset. This is probably due to some unlucky combinations of experiments (same original label or same target label). Note that the y axes do not line up.

of the same appendix shows the list of all parameters of the defensive scheme. The attacker has no knowledge of these values but the optimal values of the attacking parameters will depend on them.

The following parameters are chosen to perform the incompatibility search on: the number of particles, the source step ϵ, the acceleration coefficients c_p and c_g and the starting inertia weight w_{start}. The number of nodes and maximal number of queries are not changed as these are more of a physical limitation rather than attack parameters. Other parameters are deemed less important or influential and are therefore also kept constant.

Different combinations for the parameter values are tested on a subset of the experiments performed throughout this section. The considered ranges per parameter are shown in Table 5.10. Values are drawn from these ranges and the subset of experiments is performed using these values. The whole process is done using the Optuna framework [82].

Figure 5.9 shows contour plots for the 200 trials performed in the incompatibility study. Every plot shows contours for two parameters used in the study, where the contours are drawn based on the average L_2-distance to the original image. Blue regions correspond to a lower distance and indicate a synergy between the parameter values. Red regions indicate the opposite. Some insights that can be drawn from this Figure are the following: The number of particles seems to synergize with all other considered parameters for a value of five. The starting inertia weight w_{start}

Table 5.10: Considered ranges for all parameters part of the incompatibility search.

Parameter	Considered range
Number of particles	[1, 20]
Source step ϵ	[10^{-6}, 1]
Acceleration coefficient c_p	[0.5, 3.0]
Acceleration coefficient c_g	[0.5, 3.0]
Starting inertia weight w_{start}	[0.5, 2.0]

should be relatively low, meaning that the particles can move more freely from the beginning of the search. There is no real optimal combination for the acceleration coefficients c_p and c_g. This is to be expected due to the multi-group approach.

However, the same trade-off that had to be made for the duration of this work, should also be considered here. The synergies between parameters might cause a lower final distance to the original but this also triggers more detections. This can clearly be seen in Figure 5.10. All trials were performed on the CIFAR dataset.

5.7 Comparing to state of the art

As stated in section 3.2, HSJA [64] eclipsed BA and BBA in terms of efficiency. This section aims to compare HSJA to the attack proposed in this work both in terms of efficiency as evasiveness. The idea of query submission distribution is applicable to all adversarial attacks. In order to make the comparison more fair, this idea has been applied to HSJA as well.

Table 5.11 shows the results of the comparison. The original images and target labels are identical between both attacks and the results are averaged over all experiments. As expected, HSJA is more efficient than vanilla BBA and the spin-off PSO-BBA attack. For the CIFAR dataset, PSO-BBA performs more than four times worse when only taking the distance into account. PSO-BBA shows its potential when comparing the evasiveness of the attacks. This attack triggers less detections than HSJA both in the undistributed and distributed version. The increased evasiveness is partly due to the inherent distributed nature of the attack, but can also be attributed to the lower efficiency. The larger distance to the original image requires the attack to be less precise in order to remain adversarial. This in turn causes less detections. The larger distance to the original image does not have to pose a problem as the human eye is not very perceptible to such a small difference in L_2-distance. This can be concluded from the visual comparison in Figures 5.11 and 5.12. The difference is notable for some examples, such as the airplane in Figure 5.12, but for others it is not visible, i.e. the frog in the same figure.

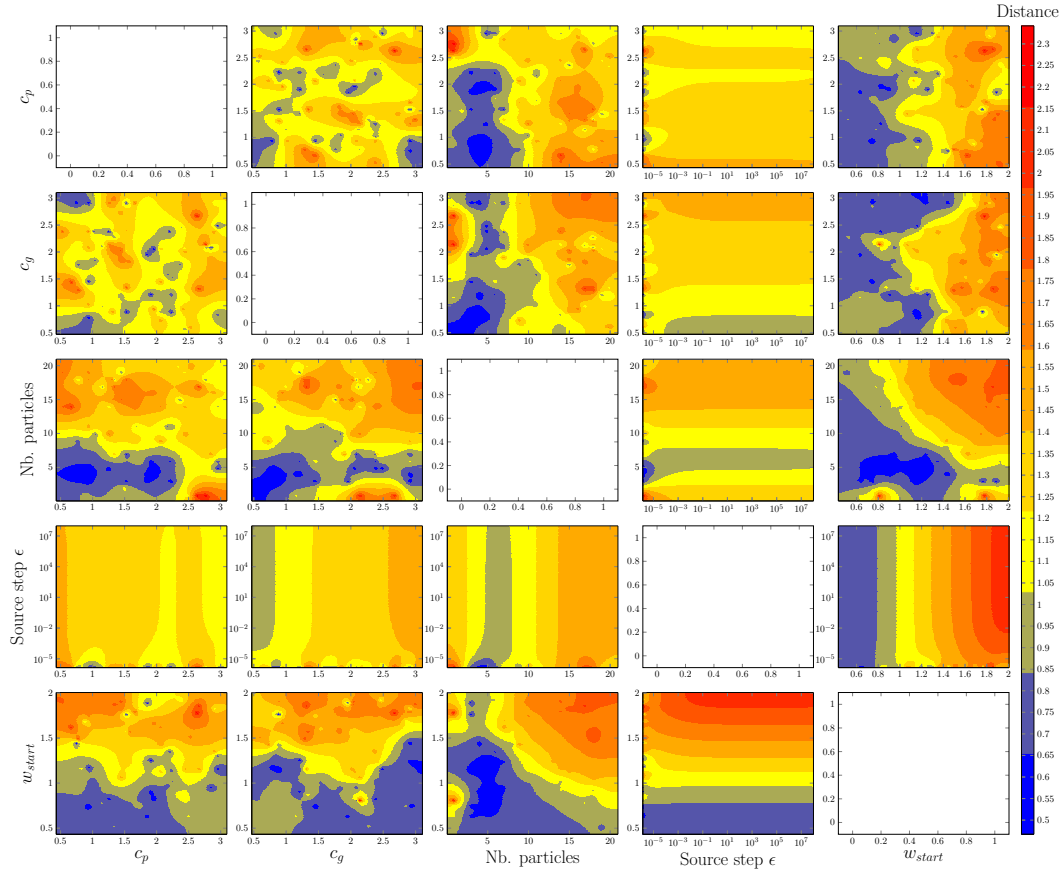


FIGURE 5.9: Contour plots for combinations of parameters used in the incompatibility study. The colorbar indicates the average L_2-distance to the original image for a combination of parameters. Blue regions (corresponding to low distance) indicate a synergy between the respective parameter values, while red regions indicate the opposite. The study has been performed on the CIFAR dataset.

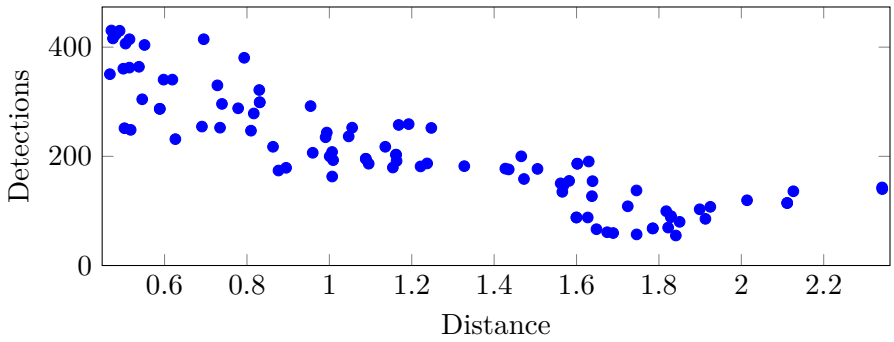


FIGURE 5.10: The plot shows the distance to the original image and the number of detections triggered for all trials in the incompatibility study. In general a larger distance yields less detections.

Table 5.11: Results for the PSO-BBA algorithm proposed in this work and the HSJA from [64]. HSJA is the more efficient attack, but triggers much more detections than PSO-BBA. Increasing the number of nodes causes a drop in the number of detection for both attacks.

		MNIST		CIFAR	
Attack	Nodes	Distance	Detections	Distance	Detections
Baseline BBA	1	2.807	413	1.306	474
PSO-BBA	1	2.712	138	1.239	257
HSJA	1	2.056	481	0.307	487
RR-PSO-BBA	10	2.712	**104**	1.239	**202**
HSJA	10	**2.056**	417	**0.307**	456

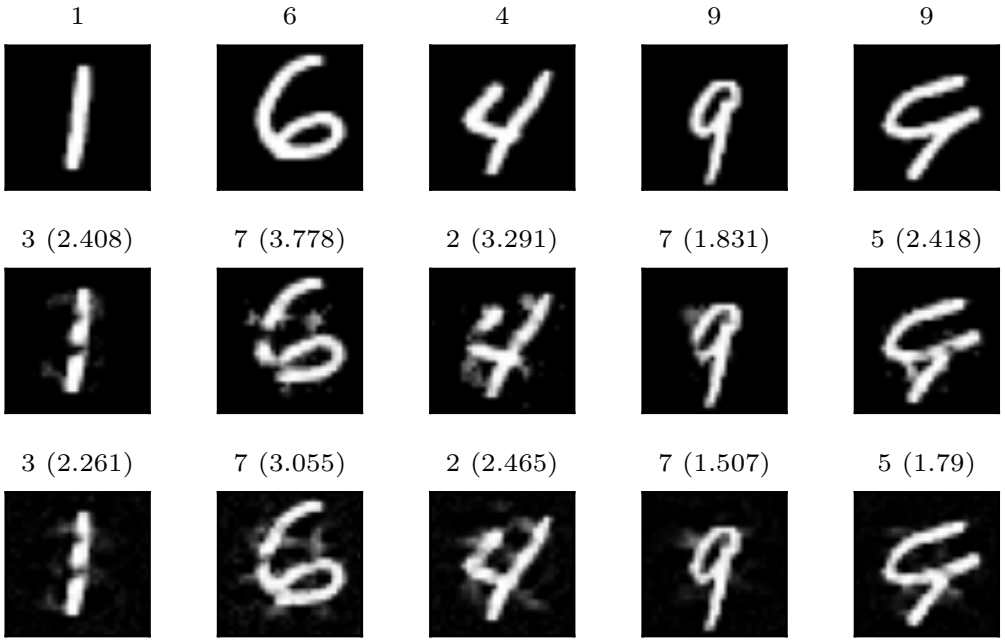


Figure 5.11: The examples of Figure 5.1 are repeated with the corresponding adversarial examples. The first row are the original images with their respective labels above them. The second row is the result of the PSO-BBA attack with the target label specified above the resulting adversarial example. The L_2-distance is shown after the target label in brackets. The setup of the third row is similar to the second row, but the adversarial examples originate from HSJA.

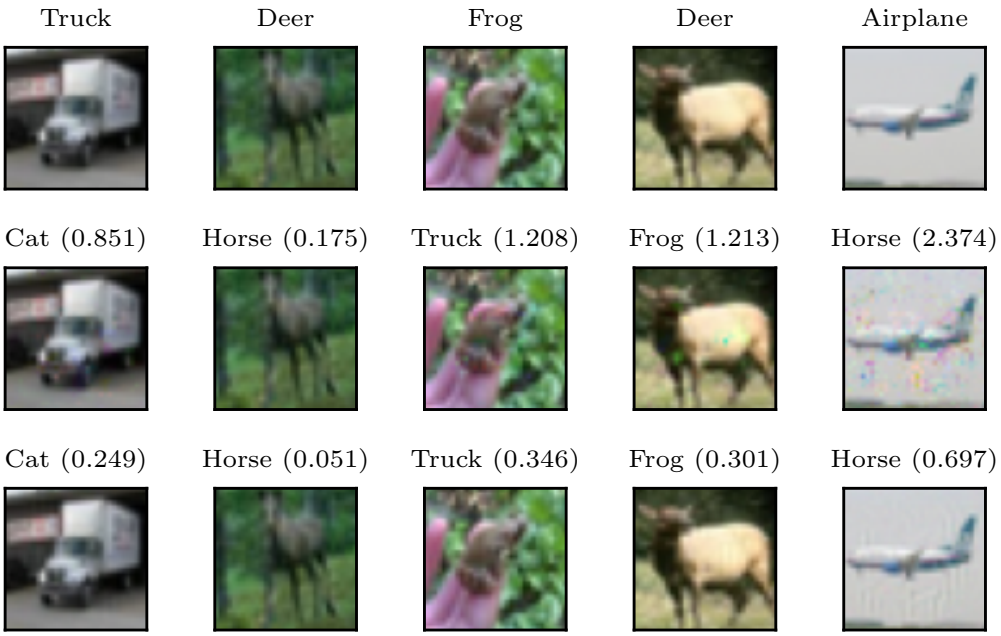


FIGURE 5.12: The examples of Figure 5.2 are repeated with the corresponding adversarial examples. The first row are the original images with their respective labels above them. The second row is the result of the PSO-BBA attack with the target label specified above the resulting adversarial example. The L_2-distance is shown after the target label in brackets. The setup of the third row is similar to the second row, but the adversarial examples originate from HSJA.

Chapter 6

Discussion

The goal of this work was to design and implement an adversarial attack that was both efficient and evasive. Efficiency is measured as the L_2-distance between the original image and the resulting adversarial example. Evasiveness is determined by the number of detections triggered by a stateful defense mechanism [11]. As shown throughout this work, the goals are conflicting. Lowering the distance to the original image usually implies more detections as a certain region in search space is queried more.

The key idea in order to avoid detection was to exploit the assumption made by the stateful defense mechanism. The scheme assumes that there is no cooperation between attackers and that all submitted queries from one user could be traced back to this user. By distributing the submission of queries over multiple nodes, and thus multiple accounts, it is impossible for the defensive mechanism to trace back the origin of the queries to a single user.

The distribution of the query submission lowers the number of detections without altering the efficiency of the attack. It could be naively assumed that distributing the submission over N nodes would reduce the number of detections by a factor N. This is not the case as has been shown in section 5.4. This effect can be explained intuitively using the following example: assume that identical queries are being submitted to a model and that this model uses a stateful defensive scheme with k equal to 50. This would cause every 51^{st} query to be flagged, since a buffer of 50 queries should be constructed first. If this query is submitted 102 times to a node, then two detections will be flagged. If the submission is distributed over two nodes, then every node will receive 51 queries, causing one detection to occur per node. This results in a total of two detections. The real improvement of the distribution lies in the fact that the queries submitted to a single node are further apart in the attack process. The attack has time to move to different areas of the search space during this time, causing the queries to be less similar. Future work could focus on better distribution schemes as the suggested schemes in this work were not very effective.

The best method to avoid detection is to submit queries that are as dissimilar as possible. Several techniques have been explored in this regard. The first idea was specific to the BBA. By selecting multiple starting points for the attack, it was possible to improve both the efficiency and evasiveness. By changing the number of starting positions, it was also possible to make a trade-off between efficiency and evasiveness depending on the costs associated with submitting a query or getting detected.

The second and third technique are essentially applicable to all adversarial attacks. The second technique consisted of inserting noise queries which did not contribute to the attack itself. This proved to be too costly in terms of efficiency for a small gain in evasiveness. However, there is still a lot of potential in this idea. Future work could implement some more sophisticated insertion schemes with better hyperparameter choices. It should be possible to perform any attack using a single account with a near perfect insertion scheme. This would be beneficial if the cost associated with creating a new account is very high.

The third technique was to run attacks concurrently and interleave the query submission of this group of attacks. This causes submitted queries to be more spread out similar to the noise insertion. The additional advantage of this technique over noise insertion is that the submitted queries are no longer unpurposed. Every query contributes to an attack of the group. Future work could optimize the selection of the attacks in the group, since attacks with the same starting or target label tend to help the defense to trigger more detections. A better distribution scheme, specifically designed for this technique, could also be an improvement.

The final algorithm was less efficient than the current state of the art HSJA [64]. However, the attack proposed in this work was more evasive. This makes the algorithm a viable candidate for adversarial attacks, especially when evasion is the primary goal. There are most definitely improvements possible by optimizing the hyperparameters of the attack. These parameters could again be tweaked depending on the needs of the attacker. It should also be noted that the parameters of HSJA are not optimized for the distributed setting. Tuning these parameters using on optimization framework such as PSO or Optuna [82] might make the attack more evasive in this setting.

Some other ideas were considered during the creation of this work, but have not been explored further. These ideas could be promising starting points for future work. The first idea was to have a dynamic number of particles in the PSO swarm. As shown by Figure 5.4, detections occur more frequently by the end of the attack process. This is due to the particles converging to a single region in search space, causing the resulting queries to be more similar. It could be beneficial to reduce the number of particles if this is the case. Depending on the hyperparameter values of the attack, this might cause less detections if the movements of a particle are large enough. It could also cause a gain in efficiency since more queries can be assigned to

a single particle, giving the particle more time to find a good adversarial position.

A second idea was to use the PSO-BBA algorithm as a initialization for another, more efficient attack. Efficient attacks such as HSJA tend to trigger more detections, since they use a lot of queries in order to find the best direction to move towards. This already causes a lot of detection from the start of the process. By assigning part of the query budget to the PSO-BBA algorithm, adversarial examples can be generated without triggering much detections. These examples can be used as starting points for the more efficient attack, reducing the amount of attacks flagged before arriving at this point.

Instead of optimizing the attack proposed in this work, future research might focus on a defensive scheme able to detect this attack. Different approaches can be taken to achieve this goal. A first approach is to relax the assumption of the stateful detection scheme that there is no cooperation possible between different users. Instead of using a buffer per user, a single larger buffer can be used for the entire model. All queries submitted to the model will be added to this buffer. This allows for detection of attacks where the query submission is spread out over multiple accounts. However, it might also cause a lot of false positive detections.

A second approach could be to link buffers that might belong to the same attack together. This could potentially be done by calculating the average image inside the buffer and using a similarity encoder to find similar buffers. This approach has the advantage over the first approach, that benign users' queries are not necessarily compared to all other queries. This should cause a lower false positive rate, albeit still higher than the original stateful defensive scheme.

A third approach is to use a similarity encoder directly for buffers. This allows for another way of linking buffers together based on similarity and could therefore serve as an alternative to the second approach. It could also be used to detect attacks in itself. By training the similarity encoder with benign and adversarial buffers, buffers could be classified as adversarial based the distance to other adversarial buffers. This would require training buffers which are attack specific. Different parameter values might change some aspects of the buffer, meaning that training data has to be available for different parameter settings.

Chapter 7

Conclusion

This chapter aims to conclude this work by answering the research questions posed in section 4.3:

What are the (dis)advantages of using PSO in relation to vanilla adversarial attacks?
PSO is an optimization framework where particles move through search space where every position is mapped to a fitness value. The framework tries to guide the particles to regions with good fitness values. PSO requires no notion of the underlying problem it is trying to solve. It only requires a fitness function to be defined for the entire search space. This allows for the optimization of problems that could not be solved analytically.

Adversarial attacks can be seen as such a problem. It requires finding a position (or image) in search space that is as close as possible to an original image, while receiving a different classification than the original image by the model under attack. Depending on the information available about the model, white-box and black-box attacks can be distinguished. White-box attacks have complete knowledge about the model. This knowledge includes the architecture, parameters, weights and gradients. Especially the gradients are very useful in order to optimize the adversarial examples generated by an attack. Methods such as FGSM [30] use these gradients to quickly improve the adversarial position. Whenever this information is available, PSO is not strong candidate for the optimization process as it does not use this information.

However, most real models operate without providing this information to its users. Black-box attacks such as BBA [1] and HSJA [64] are able to work around this lack of knowledge. HSJA in particular is able to closely match the performance of some white-box attacks, but it requires more queries in doing so. These extra queries allow for more detections by a stateful defense [11], since successive queries tend to be similar in appearance.

PSO is less prone to detections due to its multiple starting points. The queries

submitted to the model originate from different regions of the search space, making them less similar to each other. This is one of the main advantages of PSO as was discussed in section 5.3. It was also shown that PSO could lead to a more efficient algorithm. Vanilla adversarial attacks might terminate the attack process in local optima. By starting from multiple positions, the probability of ending in a local optimum is lowered, which in turn explains the higher efficiency.

The main disadvantage of PSO is the number of hyperparameters present in the framework. Incorrectly tuning the values of these parameters can lead adversarial attacks that are less efficient than their vanilla counterparts. The tuning process is also very time and energy consuming. The optimal values are dependent on the settings of the defensive scheme. This information is not available in a black-box setting. The combination of these factors causes PSO to be tricky to get right, but promising when done right.

How can PSO be combined with state of the art adversarial attacks?
This work proposes the PSO-BBA attack, a combination of PSO and BBA. The vanilla BBA iteratively updates the position of an image in search space in order to move closer to the original image. The PSO framework has particles moving through search space in order to find positions with a good fitness value. Both PSO and BBA guide candidate solutions trough search space which makes them straightforward to combine. The proposed adversarial attack is more efficient and evasive than vanilla BBA.

Combining PSO and HSJA is less straightforward. As discussed in chapter 6, PSO could be used to optimize the hyperparameters of HSJA with evasiveness as the primary goal. PSO could also be used as an initialization for other adversarial attacks. The less efficient PSO-BBA algorithm triggers fewer detections compared to HSJA. The first iterations of HSJA could be replaced by PSO-BBA in order to obtain better adversarial positions to start HSJA from.

What are the (dis)advantages of distributing an adversarial attack?
The main advantage of distributing an adversarial attack is evasiveness. By distributing the query submission over multiple nodes, successive queries submitted to the same node are further apart in the attack process. These queries are therefore less similar and trigger less detection by a stateful defensive mechanism. Section 5.4 discussed the results of distributing PSO-BBA over multiple nodes.

This approach requires that the attacker has access to a large number of accounts and nodes. The nodes are only used to submit queries, making it that they do not require much computational power. Creating a new account might be a time consuming process, causing the attacker to create accounts in bulk. The attacker does not know beforehand how many accounts are needed in order to perform a successful attack. This may cause the attacker to create spare accounts that can be used when another account is banned. However, it can be argued that these spare

accounts can already be used from the beginning of the attack in order to be even more evasive. Using this logic, all accounts should be utilized from the beginning of the attack. Therefore, every detection means that the attacker has to create a new account or that the number of accounts is reduced by one.

How can adversarial attacks be made more evasive?
As mentioned in the answer of the previous research question, distributing the query submission can make an attack more evasive. As shown in Figure 5.7, when the number of nodes is high enough, the attack can be performed without triggering any detections. Several distribution schemes have been proposed, but these proved to be very similar in performance.

It is not always possible to increase the number of nodes. For this reason, other techniques that increase the evasiveness of an attack have been proposed. The PSO-BBA algorithm uses multiple starting positions to increase the evasiveness. However, this technique is not applicable to all adversarial attacks.

Another approach to increase the evasiveness of an attack without increasing the number of nodes is to insert noise queries in order to flush the buffer of the defensive scheme. Different types of noise queries and different insertion schemes were tested, but they caused the attack to be too inefficient for the gain in evasiveness. Instead of inserting noise queries, purposeful queries could also be inserted. This is done by running multiple attacks concurrently and interleaving the query submission of all these attacks. As shown in Figure 5.8, by increasing the number of concurrent attacks, the average number of detections drop. This approach is extremely useful if the attacker is interested in multiple adversarial examples.

Appendices

Appendix A

Parameter table

This appendix serves as an overview of the different parameters present in the attack (and defense).

Table A.1: Parameter table

Parameter	Default value	Usage
Number of particles	5	Controls the number of particles in the PSO swarm. Increasing the number of particles decreases the evasiveness of the algorithm but decreases the efficiency as seen in section 5.3. Decreasing the value decreases both the evasiveness and efficiency.
Number of nodes N	10	Controls the number of nodes over which the query submission can be distributed. Increasing the number of nodes helps the algorithm to be more evasive as seen in section 5.4.1. Every added node comes with a certain cost.
Query budget	25000	Controls the maximum number of queries that can be submitted by the attack. Increasing the number can make the attack more efficient, but the later queries tend to trigger more detections, causing the evasiveness to drop.

Parameter	Default value	Usage
Group size s	1	Controls the number of experiments that are grouped together and are executed concurrently. The queries of the attacks in the same group are submitted in an interleaved fashion. Increasing this value aids towards the evasiveness of the attack without sacrificing efficiency. Experiments have been performed with this parameter in section 5.5.
Distribution scheme	RR distribution	Controls how the queries are distributed over the different nodes. As discussed in section 5.4, the distribution scheme does not heavily influence the number of detections. Other options for this parameter are the MRR, DB and EDB distribution schemes.
History length	20	Controls the size of the query submission history that is taken into account in the DB and EDB distribution schemes. As seen in section 5.4.2, increasing this value helps the algorithm remain more evasive at an added computational cost. The chosen value is a middle ground between this computational cost and the evasiveness.
Source step ϵ	0.25 (MNIST) 0.20 (CIFAR)	Controls the step towards the original image in the BBA. This parameter is also shown in Figure 3.3 as ϵ. Increasing the value causes the queries to be more spread out over the search space, ultimately lowering detections. However, increasing the value too much will result in slow convergence of the attack.
Spherical step η	0.05	Controls the size of the random direction of the BBA algorithm. This parameter is also shown in Figure 3.3 as η. The value of this parameter is chosen based on the results of [1]. Increasing the value, adds more randomness into the algorithm, but can hamper convergence.

Parameter	Default value	Usage
Source step multiplier up	1.05	Controls the speed at which the source step increases when the new position remains adversarial. Setting this value to 1 (together with source step multiplier down) fixes the value of the source step for the entire duration of the attack. The increasing value of the source step helps the attack to be more evasive due to the submitted queries being more spread out.
Source step multiplier down	0.99	Controls the speed at which the source step decreases when the new position remains non-adversarial. Setting this value to 1 (together with source step multiplier up) fixes the value of the source step for the entire duration of the attack. The decreasing value of the source step helps the attack to be more efficient, since the smaller steps aid convergence.
Recalculate mask every	50	Controls after how many iterations the mask used in the BBA should be recalculated. This parameter has no big effect on the efficiency of the attack, only on the run time. After a sufficient amount of iterations the masks tends to change only slightly.
Particle acceleration coefficient c_p	2[1]	Controls the rate of attraction towards the personal best position of a particle. The different steps of the particle are weighted according to equation 2.2. Increasing this value creates a stronger attraction to the personal best position of the particle.

[1] Due to the multi-group approach, these values are actually set based on equations 5.1 and 5.2 with $A1$ and $A2$ being 1 and 2 respectively.

Parameter	Default value	Usage
Global acceleration coefficient c_g	1[1]	Controls the rate of attraction towards the best position of the swarm. The different steps of the particle are weighted according to equation 2.2. Increasing this value creates a stronger attraction to the best position of the swarm.
Maximum velocity v_{max}	0.5	Controls the maximum velocity of a particle. The velocity values of equation 2.1 are clipped by this value in order to avoid the problem of exploding velocities. Decreasing this value causes the algorithm to take smaller steps in every iteration, which in turn reduces the evasiveness.
w_{start}	1	Controls the value of the inertia weight of equation 2.3. Decreasing the value will cause the particle to change its direction more easily in the beginning of the attack, allowing for more exploitation of the search space.
w_{end}	0	Controls the value of the inertia weight of equation 2.3. Increasing the value will cause the particle to change its direction more easily at the end of the attack, allowing for more exploitation of the search space.
κ	0.8	Controls the rate of convergence of the PSO algorithm when using a constriction factor χ. Increasing this value results in slower convergence, but a more thorough search. Decreasing this value has the opposite effect. The value should be between 0 and 1.

Parameter	Default value	Usage
Insert type	Test data	Controls the type of noise queries that are inserted in order to flush the buffer of the detector. The other options are uniforms noise, Perlin noise with a fixed frequency, Perlin noise with a variable frequency and a mix of all other approaches. Other approaches have a smaller inter query distance as shown in Table 5.8. Therefore train data noise is the best option in terms of evasiveness.
Insert time	Distance based	Controls when noise queries should be inserted to flush the buffer of the detector. Other options are fixed interval and geometric decay.
Insert count	1	Controls the number of noise queries that are inserted every time the 'Insert time' decides that noise should be inserted. Inserting more noise queries can improve evasiveness, but the efficiency is lowered since less queries are used to progress the attack.
Insert threshold	/	Controls the threshold used in the distance based insertion scheme. Whenever the distance between an impending query and the previously submitted queries falls below this value, noise queries will be inserted. Increasing this value will result in more noise being inserted, causing the evasiveness to increase while the efficiency drops. Increasing the value too high will result in more detections due to the noise queries being flagged as an attack.
Insert decay rate	/	Controls the decay rate of the geometric decay insertion scheme. The number should be smaller than 1 in order to insert more queries near the end of the attack. The closer this number is to zero, the faster noise queries will be inserted.

Table A.2: Parameter table of the defense[2]

Parameter	Default value	Usage
Buffer type	Query bounded	Controls how queries will be removed from the buffer. A query bounded buffer will hold a certain amount of queries. If a new query has to be added when the capacity is reached then queries are removed based on the first in, first out method [83]. The other option is a time bounded buffer where queries are removed once they reach a certain age. This approach requires more computations as checks need to happen regularly. Therefore the query bounded buffer is chosen in line with [11].
Buffer size	1000	Controls the number of queries the buffer holds before queries are removed again. Increasing the value results in more attack being possibly flagged but requires more storage. The value is chosen in line with [11].
k	50	Controls the number of neighbors that are considered when computing the average distance to a new incoming query. Increasing the value flags attacks with more certainty. However, this value is also equal to the number of queries that have to be inserted before any attack can be detected. Increasing the value too far will therefore result in no detections. The value is chosen in line with [11].

[2]Note that the attacker has no control over (and knowledge of) these parameters. However, the optimal parameters of the attack heavily depend on the defense parameters.

Parameter	Default value	Usage
Threshold	0.009 (MNIST) 0.021 (CIFAR)	Controls the threshold used to flag attacks. If the mean distance to the nearest neighbors of an incoming query falls below this value then the query is flagged as an attack in progress. Appropriate action can be taken against the account that submitted this query. The thresholds are determined based on the procedure discussed in [11]. A false positive rate of 0.1% will be achieved when adding all benign queries of the training set of the respective dataset.
Flush buffer after detection	True	Controls whether or not the buffer of the detector should be cleared after a detection happens. The value is set to true in order to simulate the user having to set up a new account after the previous account has been banned. If milder actions, such a temporary ban, are taken then the buffer should not be cleared.
Shadow ban	False	Controls whether or not the user is notified of a ban. If the value is false then the user is notified. This seems the most probable setting in real scenarios as there is no straightforward way to shadow ban a user while using an API. The attacker can use the knowledge of bans to its advantage as discussed in section 5.4.2.

Bibliography

[1] Thomas Brunner, Frederik Diehl, Michael Truong Le, and Alois Knoll. Guessing Smart: Biased Sampling for Efficient Black-Box Adversarial Attacks. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4957–4965, October 2019. arXiv: 1812.09803.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[3] Dheeru Dua and Casey Graff. Spambase Data Set. http://archive.ics.uci.edu/ml/datasets/spambase, 2017. Accessed: 2022-05-31.

[4] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *ACM Comput. Surv.*, 35(4):399–458, dec 2003.

[5] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[6] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

[7] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

[8] Jon Russell. Google's alphago ai wins three-match series against the world's best go player. https://techcrunch.com/2017/05/24/alphago-beats-planets-best-human-go-player-ke-jie/amp/?guccounter=1, 05 2017. Accessed: 2021-12-08.

[9] Gaurav Tewari. The Future Of AI: 5 Things To Expect In The Next 10 Years. https://www.forbes.com/sites/gradsoflife/2021/12/03/4-effective-dei-strategies-business-leaders-should-measure/?sh=6d68298d578b, 05 2022. Accessed: 2022-05-31.

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

[11] Steven Chen, Nicholas Carlini, and David Wagner. Stateful Detection of Black-Box Adversarial Attacks. *arXiv:1907.05587 [cs]*, July 2019. arXiv: 1907.05587.

[12] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. 65(6):386–408.

[13] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[14] Wikipedia. Heaviside step function. https://en.wikipedia.org/wiki/Heaviside_step_function, 04 2022. Accessed: 2022-04-30.

[15] Wikipedia. Softmax function. https://en.wikipedia.org/wiki/Softmax_function, 03 2022. Accessed: 2022-04-30.

[16] M.V. Valueva, N.N. Nagornov, P.A. Lyakhov, G.V. Valuev, and N.I. Chervyakov. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, 177:232–243, 2020.

[17] Steve Lawrence, C. Lee Giles, Ah Chung Tsoi, and Andrew D. Back. Face recognition: A convolutional neural network approach. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 8(1):98–113, 1997.

[18] Xiangang Li and Xihong Wu. Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. *CoRR*, abs/1410.4281, 2014.

[19] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[20] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. Recurrent neural networks for time series forecasting: Current status and future directions. *CoRR*, abs/1909.00590, 2019.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[22] Wikipedia. Perceptron. https://en.wikipedia.org/wiki/Perceptron, 2022 03. Accessed: 2022-04-16.

[23] Jason Brownlee. How Do Convolutional Layers Work in Deep Learning Neural Networks? https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/, 07 2020. Accessed: 2022-05-31.

[24] Fatemeh Vakhshiteh, Ahmad Nickabadi, and Raghavendra Ramachandra. Adversarial attacks against face recognition: A comprehensive study, 2021.

[25] Abhiram Gnanasambandam, Alex M. Sherman, and Stanley H. Chan. Optical adversarial attack, 2021.

[26] Octavian Suciu, Scott E. Coull, and Jeffrey Johns. Exploring adversarial examples in malware detection, 2019.

[27] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text, 2018.

[28] Giulio Zizzo, Chris Hankin, Sergio Maffeis, and Kevin Jones. Invited: Adversarial machine learning beyond the image domain. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–4, 2019.

[29] Victoria Drake. Threat modeling. https://owasp.org/www-community/Threat_Modeling#:~:text=A%20threat%20model%20is%20a,through%20the%20lens%20of%20security., 09 2021. Accessed: 2022-03-31.

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

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

[32] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations*, 2018.

[33] Wikipedia. Distance. https://en.wikipedia.org/wiki/Distance, 11 2021. Accessed: 2021-12-09.

[34] Wikipedia. Discrete space. https://en.wikipedia.org/wiki/Discrete_space, 04 2022. Accessed: 2022-04-30.

[35] Gabriel Resende Machado, Eugênio Silva, and Ronaldo Ribeiro Goldschmidt. Adversarial machine learning in image classification: A survey toward the defender's perspective. *ACM Comput. Surv.*, 55(1), nov 2021.

[36] Gabriel Resende Machado, Eugênio Silva, and Ronaldo Ribeiro Goldschmidt. Adversarial machine learning in image classification: A survey towards the defender's perspective. *CoRR*, abs/2009.03728, 2020.

[37] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, page 506–519, New York, NY, USA, 2017. Association for Computing Machinery.

[38] Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *CoRR*, abs/1802.00420, 2018.

[39] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *CoRR*, abs/1511.04508, 2015.

[40] Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples, 2015.

[41] Dan Hendrycks and Kevin Gimpel. Visible progress on adversarial images and a new saliency map. *CoRR*, abs/1608.00530, 2016.

[42] Chuan Guo, Mayank Rana, Moustapha Cissé, and Laurens van der Maaten. Countering adversarial images using input transformations. *CoRR*, abs/1711.00117, 2017.

[43] Nicolas Papernot and Patrick D. McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *CoRR*, abs/1803.04765, 2018.

[44] Xiaoyu Cao and Neil Zhenqiang Gong. Mitigating evasion attacks to deep neural networks via region-based classification. *CoRR*, abs/1709.05583, 2017.

[45] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[46] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: Optimization by a colony of cooperating agents. ieee trans syst man cybernetics - part b. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 26:29–41, 02 1996.

[47] John H. Holland. *Genetic Algorithms and Adaptation*, pages 317–333. Springer US, Boston, MA, 1984.

[48] Rami Abousleiman and Osamah Rawashdeh. Electric vehicle modelling and energy-efficient routing using particle swarm optimisation. *IET Intelligent Transport Systems*, 10(2):65–72, 2016.

[49] Tadashi Yamada and Zukhruf Febri. Freight transport network design using particle swarm optimisation in supply chain–transport supernetwork equilibrium. *Transportation Research Part E: Logistics and Transportation Review*, 75:164–187, 2015.

[50] Bruno Almeida and Victor Coppo Leite. *Particle Swarm Optimization: A Powerful Technique for Solving Engineering Problems*. 12 2019.

[51] Romeela Mohee and Ackmez Mudhoo. Analysis of the physical properties of an in-vessel composting matrix. *Powder Technology*, 155(1):92–99, 2005.

[52] Y. Shi and R.C. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1945–1950 Vol. 3, 1999.

[53] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[54] Wikipedia. Vanilla software. https://en.wikipedia.org/wiki/Vanilla_software, 04 2022. Accessed: 2022-05-12.

[55] Naufal Suryanto, Chihiro Ikuta, and Dadet Pramadihanto. Multi-group particle swarm optimization with random redistribution. In *2017 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC)*, pages 1–5, 2017.

[56] Rainer Storn and Kenneth Price. Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization*, 23, 01 1995.

[57] Radha Thangaraj, Millie Pant, Ajith Abraham, and Pascal Bouvry. Particle swarm optimization: Hybridization perspectives and experimental illustrations. *Applied Mathematics and Computation*, 217(12):5208–5226, 2011.

[58] Eid Albalawi, Fujie Chen, Ruppa K. Thulasiram, and Parimala Thulasiraman. Effect of pso communication topologies on task matching in grid computing. *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 426–433, 2019.

[59] Takahiro Tsujimoto, Takuya Shindo, Takayuki Kimura, and Kenya Jin'no. A relationship between network topology and search performance of pso. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–6, 2012.

[60] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models. *arXiv:1712.04248 [cs, stat]*, February 2018. arXiv: 1712.04248.

[61] Ye Wenhe. Trust-region methods. https://optimization.mccormick.northwestern.edu/index.php/Trust-region_methods, 06 2014. Accessed: 2021-12-09.

[62] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. Robustness of classifiers: from adversarial to random noise. *CoRR*, abs/1608.08967, 2016.

[63] Ken Perlin. An image synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, page 287–296, New York, NY, USA, 1985. Association for Computing Machinery.

[64] Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright. HopSkipJumpAttack: A Query-Efficient Decision-Based Attack. *arXiv:1904.02144 [cs, math, stat]*, April 2020. arXiv: 1904.02144.

[65] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[66] Sean Bell and Kavita Bala. Learning visual similarity for product design with convolutional neural networks. *ACM Trans. Graph.*, 34(4), jul 2015.

[67] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009.

[68] Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, and Mani Srivastava. GenAttack: Practical Black-box Attacks with Gradient-Free Optimization. *arXiv:1805.11090 [cs]*, June 2019. arXiv: 1805.11090.

[69] Yinpeng Dong, Hang Su, Baoyuan Wu, Zhifeng Li, Wei Liu, Tong Zhang, and Jun Zhu. Efficient decision-based black-box adversarial attacks on face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7714–7722, 2019.

[70] Rayan Mosli, Matthew Wright, Bo Yuan, and Yin Pan. They might not be giants: crafting black-box adversarial examples with fewer queries using particle swarm optimization. *arXiv preprint arXiv:1909.07490*, 2019.

[71] Hyunjun Mun, Sunggwan Seo, Baehoon Son, and Joobeom Yun. Black-box audio adversarial attack using particle swarm optimization. *IEEE Access*, 10:23532–23544, 2022.

[72] Fengtao Xiang, Jiahui Xu, Wanpeng Zhang, and Weidong Wang. A distributed biased boundary attack method in black-box attack. *Applied Sciences*, 11(21), 2021.

[73] Naufal Suryanto, Hyoeun Kang, Yongsu Kim, Youngyeo Yun, Harashta Tatimma Larasati, and Howon Kim. A distributed black-box adversarial attack based on multi-group particle swarm optimization. *Sensors*, 20(24), 2020.

[74] Y. Shi and R.C. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1945–1950 Vol. 3, 1999.

[75] Kaspersky. What is a ddos attack? - ddos meaning. www.usa.kaspersky.com/resource-center/threats/ddos-attacks, 01 2021. Accessed: 2022-03-18.

[76] Google. Cloud Vision pricing. https://cloud.google.com/vision/pricing#prices, 04 2022. Accessed: 2022-04-08.

[77] Amazon. Amazon Rekognition pricing. https://aws.amazon.com/rekognition/pricing/, 04 2022. Accessed: 2022-04-08.

[78] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[79] Wikipedia. Round-robin scheduling. https://en.wikipedia.org/wiki/Round-robin_scheduling, 04 2022. Accessed: 2022-04-18.

[80] Wikipedia. Shadow banning. https://en.wikipedia.org/wiki/Shadow_banning, 04 2022. Accessed: 2022-05-11.

[81] Wikipedia. Geometric progression. https://en.wikipedia.org/wiki/Geometric_progression, 03 2022. Accessed: 2022-05-15.

[82] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[83] Wikipedia. FIFO (computing and electronics). https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics), 05 2022. Accessed: 2022-05-18.