A study on the limits of adversarial examples in machine learning

by

Bogdan-Alexandru Serban
Student ID: 1856482

A thesis submitted to the University of Birmingham for the degree of
MASTER OF ADVANCED COMPUTER SCIENCE

Supervisor: Mark Ryan

School of Computer Science
University of Birmingham
September 2018

Abstract

Neural networks have become the most popular machine learning algorithms by being able to achieve performance levels comparable to humans in solving various tasks. This led to them being integrated into many systems nowadays, in some of them actually managing security-critical tasks. The fact that they are vulnerable to carefully perturbed inputs, known as adversarial examples makes research in the area of neural networks security to become critical.

In this paper, we are approaching the context of adversarial machine learning from two different perspectives. First, from an attacker perspective, we are studying the properties of adversarial examples when evaluated on the models they were generated on, and on different models. We also test their robustness when applying transformations to them for a better understanding of how they perform.

In the second part, we switch to the perspective of a defender. Thus, we propose a new architecture, based on "synthetic" features, which are pseudo-features that separate the classes into equal groups. Through this architecture we try to generate models that are able to detect adversarial examples from clean images, thus increasing their robustness to adversarial attacks. We have evaluated these models using a modified set of attacks. We have discovered that our models achieve high rates in detecting adversarial examples when they are generated in the untargeted scenario, and also prove to have some robustness in when the attacks are performed in the targeted scenario.

Acknowledgements

I would like to thank my family for providing me with support and continuous encouragement throughout this year. Also, I would like to thank my project supervisor, Mark Ryan, who has offered me guidance and help in the process of researching and writing this thesis.

Contents

List of Figures

List of Tables

1. Introduction

Machine learning is becoming more and more integrated with our everyday lives. The growth of computing power and the capacity of storing large amounts of data have driven research in this area to a completely new landscape. This has allowed machine learning algorithms to achieve high performance in solving various tasks, in some cases, better performance even than the human.

The most powerful class of machine learning algorithms are artificial neural networks. Combined with the availability of large training datasets, these algorithms are able to outperform other machine learning approaches in various tasks such as image classification [23], digit recognition [14], object detection [39], speech recognition [18], natural language processing [10] and even playing games [42].

Apart from the academic environment, neural networks are being more and more integrated into systems that humans interact with on daily basis. Filtering spam from regular emails in mailing services [9], filtering and removing inadequate content from being displayed in social network feeds [32], automated malware detection [41] and fraud detection [21] are only a couple of examples in that matter. As it can be seen, machine learning algorithms are trusted to perform tasks without our direct supervision. In some cases, these tasks require high responsibility and may have negative consequences if they are not performed as intended. Therefore, these algorithms must prove to be secure against malicious individuals which may want to interfere with them in order to gain some advantages over the system.

First studies on vulnerabilities of neural networks have been performed by Szegedy et al. [44]. In their research, they have discovered that neural networks are susceptible to attacks by feeding them input data that is altered in such a way that it appears to be unaltered to humans, but it is seen as completely different by the network. This altered input data is referred to as an "adversarial example" or "adversarial sample" [44]. This discovery has been the basis of a whole new area of research, focused on exploiting these vulnerabilities of deep learning models to change their behaviour in malicious ways. In this matter, various attacks have been proposed in the past years. On the opposite side, research has also been conducted to determine ways to defend against these attacks. Both attacks and defence methods will be discussed in the following sections.

The purpose of this project is to perform a study of neural networks in an adversarial environment. To be more precise, the paper will focus on networks designed to perform image classification tasks. Therefore, the concept of adversarial example refers to an image, which has a carefully crafted perturbation added such that it appears similar to the original image when

evaluated by humans, but it is classified to a completely different class by the neural network.

The project will be composed of two parts. The first part will be focusing on getting a better understanding of the limits of adversarial examples. In other words, we will study the properties of adversarial examples generated with a set of attacks before and after different transformations (rotation, translation, etc.) of various intensities are applied to them. Also in this part, we will study how these adversarial examples preserve their properties when evaluated on a different model than the one they were generated on.

The second part will consist of an attempt to improve the neural network's robustness to adversarial examples by proposing a new architecture based on what we call "synthetic features". The architecture will consist in two major components it will be explained in **section 3.1**. First, a set of small individual neural networks working in parallel will each extract exactly one synthetic feature from the raw input. Then, a deterministic classifier is added which will analyse the extracted features and will assign the input to one of the classes from the set of possible classes or to the "incorrect" class, an extra class that we add with our architecture.

Our intuition is that by using individual networks to extract the features will prove harder for an attack to generate an adversarial example that will be able to fool all the networks at once and generate a consistent output. Thus our network will be capable of detecting adversarial examples from clean images and label them as "incorrect" with high probability.

This paper is structured as follows. In **section 2**, all the necessary background required for the topic of this paper will be provided. **Section 3** will focus on the technical details of the design and implementation of the proposed solution. **Section 4** will give a description of the experimental setup in which the experiments were conducted. In **section 5**, we will show the results we have obtained from running the experiments. In **section 6** an analysis of the results will be performed and pointers towards future work will be given. **Section 7** will conclude our study.

2. Background

2.1 Deep learning

Machine learning is providing solutions for solving various tasks in today's society. From systems that give recommendations on online stores, filtering the content that is being uploaded on social networks, fraud detection and security, it is becoming more and more popular.

The downside of conventional machine learning algorithms is the fact that their capability of processing raw data is rather limited. To implement such algorithms and integrate them in their systems, engineers require extra domain knowledge that will allow them to extract relevant features from the raw data before feeding it into the algorithms [1, 17, 28].

Deep-learning methods, such as neural networks have overcome this by the fact that they are able to discover these features automatically, therefore being able to work directly with raw input data. They are included in a much broader category of methods known as representation learning [3]. These methods have a layered internal structure, gaining a complex understanding of the data by combining multiple simple modules and adding some type of non-linearity. The modules perform successive transformations at each level starting from the raw input and gradually find a more abstract representation that will highlight the relevant features for the task to be performed. The advantage of this approach is that these representations are not created by engineers, but are learned automatically by the system, which in return simplifies to some extent the complexity of building it.

As it was mentioned in the introduction section, due to increases in computational power and in the volume of training data, deep learning methods have achieved performance levels comparable, or even better than humans at performing tasks in various areas of research such as computer vision [14, 23, 39], natural language processing [10, 18], security [9, 21, 41], biology [29], physics [8] and others [32, 42].

A big part of the deep learning approaches is used for solving tasks in the context of supervised learning. Recently, research into applying these methods on solving unsupervised tasks is beginning to gain more importance, as it is being considered to become more important in the long run compared to supervised learning [28]. As this study is focused on models performing image classification, from this point forward we will be referring only to the context of supervised learning.

A neural network is a function organized in a manner similar to biological neural networks found in the brains of animals [48], which performs a non-linear mapping $f : \mathbb{R}^n \to \mathbb{R}^m$, $f(x) = y$ from a specific input space, $\mathbb{R}^n$, to a specific output space, $\mathbb{R}^m$. Networks are organized into

layers, which are stacked on top of each other, and the output from each layer represents the input for the next one. This type of organization allows the network to have its own internal representation of the input data, by learning how to extract and use the information that is relevant for the task to be performed.

As it can be seen in **figure 2.1**, a regular neural network is made up of three types of layers.

The **input layer** is the first layer of the network, which directly receives the raw input data and passes it to the next layers called **hidden layers**.

Each of these layers performs some transformations on its input to create a different representation, which is more selective and has higher invariance [28]. This representation is then passed to the next hidden layer which repeats the process. The number of hidden layers in a neural network can vary from one layer to any number, the only limit being the computational resources available.

In theory, according to Universal Approximation Theorem [11], a neural network with one sufficiently wide enough hidden layer can learn to approximate any continuous function. In practice, it has been proven that while this is true, it may prove infeasible due to the computational cost of running a very wide. Thus, increasing the number of hidden layers has been proven to allow reducing the width of the network, but preserving its properties. A neural network which has more than one hidden layers is referred to as a **deep neural network**.

The last layer of a neural network, called the **output layer**, is the layer that gives the output of the network, which varies based on the task to be performed. In the case of image classification, for example, the output layer will provide a probability distribution over all the possible classes, where the i^{th} element represents the probability of the input image to belong to class i.

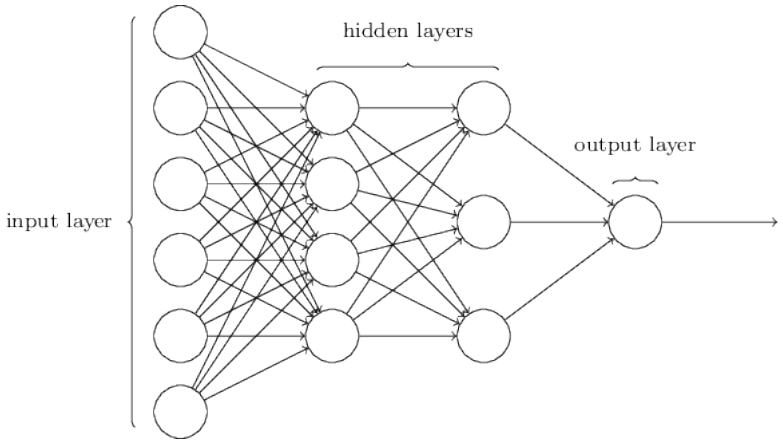

Figure 2.1: Overview of a neural network [1]

Each layer in a neural network is made up of **neurons**, also referred to as **perceptrons** [40]. They represent the most basic unit of a neural network. As seen in **Figure 2.2**, a neuron performs a weighted sum with weights w_i of the output from the previous layer, a_i, followed by

[1] **Source:** Neural Networks and Deep Learning

adding a bias b_i.

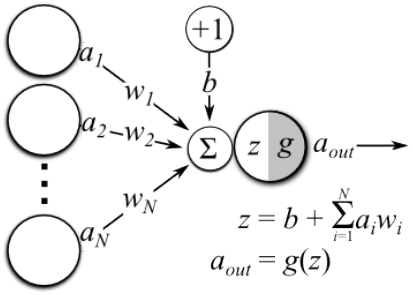

Figure 2.2: Close-up on a neuron[2]

The output of this operation is then fed into a **activation function** which has the role to add non-linearity to the model and to "decide" if that neuron has triggered or not [47], mimicking to some extent the working mechanism of biological neurons. The output of the activation function is then sent to the next layer. The most commonly used ones and the ones that are going to be used in this study are Sigmoid, ReLu and Softmax. They are described in the following paragraphs, where $z_t(x)$ is the value of the biased weighted sum performed by the t^{th} neuron as shown in **Figure 2.2**, and x represents the output of the previous layer.

Sigmoid activation functions have been designed to act as a "smooth step function" [47], which means that it will act as a binary activation (outputs values between 0 and 1) and also has a smooth gradient.

$$\sigma_t(x) = \frac{1}{1 + e^{-z_t(x)}} \tag{2.1}$$

A **Rectified Linear Unit (ReLu)** as defined in **Equation 2.2**, is designed to speed up the training process by behaving similarly to linear functions, while keeping non-linear properties [15].

$$\text{ReLu}_t(x) = \max(0, z_t(x)) \tag{2.2}$$

The **Softmax** activation is most commonly used on the output layer. It is very popular for neural networks performing classification tasks. Its purpose is to turn the output of the neural network into a probability distribution over the output space. A Softmax activation acts as follows:

$$\text{Softmax}_t(x) = \frac{e^{z_t(x)}}{\sum_{j=1}^{C} e^{z_j(x)}} \tag{2.3}$$

Apart from the basic, fully connected layers, where all the neurons in one layer are connected to all the neurons in the previous layer, based on the task to be solved there have been developed other types of layer architectures. In the context of this paper it is worth mentioning convolutional and pooling layers, which have defined a whole new architecture model for neural networks, namely **convolutional neural networks**. The most important aspect of this architecture is the fact that it can process raw data that comes in the form of multi-dimensional

[2]**Source:** The Clever Machine Blog

arrays, such as colour images, which consist in three two-dimensional arrays of pixel values, one for each colour channel.

Convolutional layers work by using small-sized filters which are convoluted with the input image, resulting in a set of feature maps. Each map is connected only to a small region of the input coming from the previous layer. Usually, convolution layers are followed by a pooling layer, which is used to merge similar features detected by the convolutional layers into one. The can be considered as a downsampling operation while preserving the maximum amount of information. This has the effect of reducing the size of the internal representation and maintaining some invariance to small variations in the image [28].

Using convolutional and pooling layers has several advantages over fully connected ones. Firstly, the total number of parameters is greatly reduced, which implies that increasing the depth of the network will not result in a great increase in the number of parameters. Secondly, using convolutions ensures regional invariance. In other words, the feature maps are invariant of the regions of the input image (e.g. an image of a dog will still be classified as a dog even if it is rotated). Nowadays, convolutional and pooling layers are used in almost any model that deals with solving computer vision problems [24, 28].

The weights w_l and biases b_l associated to each layer "l" represent the parameters of the neural network on which the mapping function defined above depends. We shall refer to the set containing the weights and biases of a neural network by using the Greek letter theta (θ).

In order to train a neural network, one must first define a loss function. In the context of supervised learning, it measures how different the output of the neural network is from the true output. If we consider the case of image classification, let (x_i, y_i) be a pair of training data, where $x_i \in \mathbb{R}^{w*h*c}$ is an image of width w and height h, with c being the number of colour channels (3 in the case of RGB images) and $y_i \in \mathbb{R}^C$ (where C is the number of possible classes) is a vector representing a probability distribution over the space of all possible class labels. Therefore, the loss function, denoted as $L(\theta, x, y)$, will measure how different the predicted probability distribution $\hat{y_i}$ is from the true probability distribution y_i(where y_{ij} is 1 if the input image belongs to class j and 0 otherwise) for a specific set of network parameters θ. This is referred to as the **cross entropy loss** and it is the most commonly used loss in the case of classification tasks.

Training a neural network consists of fine-tuning the parameters θ to minimize the value of the loss function. In other words, in the context of image classification, training aims to bring the probability distribution outputted by the network as close as possible to the true probability distribution. This fine tuning is performed through an optimisation method called **Stochastic Gradient Descent**. This consists in feeding a small part of the inputs, called a **batch** to the network, computing the average gradient of the loss function with respect to the network parameters evaluated for these inputs, and then adjusting the parameters accordingly [28].

The equations for updating the weights and biases of the network at time $t+1$ using gradient descent are presented below. The hyperparameter α, called **learning rate** specifies the degree to which we are adjusting the parameters of the network each step, based on the gradient of the

loss function.

$$w_{t+1} \leftarrow w_t + \alpha \cdot \nabla_{w_t}(L(\theta, x, y)) \tag{2.4a}$$
$$b_{t+1} \leftarrow b_t + \alpha \cdot \nabla_{b_t}(L(\theta, x, y)) \tag{2.4b}$$

The gradient vectors are computed using the **backpropagation** algorithm, which uses the chain rule of differentiation to compute the gradients of the cost function with respect to the weights and biases. The gradients are computed starting from the output layer and then propagating them towards the input layer using the chain rule. Repeating this process iteratively will ensure the convergence of the function towards a point of a local minimum, which gives a close approximation of the optimal solution to the given task.

2.2 Distance Metrics

Distance metrics are functions used to measure the similarity between two inputs. In adversarial machine learning literature, the most frequently used distance metrics are the L_p norms. The L_p norm between two inputs x and x' is usually written as $\|x - x'\|_p$ and is computed as:

$$\left\|x - x'\right\|_p = \left(\sum_{i=1}^{n} |x_i - x'_i|^p\right)^{1/p} \tag{2.5}$$

In the context of adversarial machine learning, most commonly used L_p norms are L_0, L_1, L_2 and L_∞ norms.

The L_0 distance represents the number of features that are different in the two inputs. In the context of image classification, the L_0 distance between two images represents the number of pixels that differ from one image to the other.

The L_1 distance measures the total amplitude of the differences between the features of the two inputs. In image classification, this distance measures the total amplitude of the pixel value differences between the two images. This norm was not very popular in the context of adversarial machine learning until it was integrated into the Elastic-Net attack, which is described in more detail in **section 2.3**.

The L_2 distance represents the Euclidian distance between the two inputs. This norm is used by attacks which produce low perturbations in images, such as L-BFGS [44], Carlini-Wagner [5], or DeepFool [31], as many small perturbations in pixel values will yield in a small value for the L_2 distance.

The L_∞ accounts for the maximum difference between every two corresponding features of the two inputs. In image classification, the L_∞ norm represents the maximum change between the values of two corresponding pixels. This metric is commonly used with attacks based on the gradient of the loss function with respect to the input, such as Fast Gradient Sign Method and its related methods, as it can be seen in **section 2.3**.

2.3 Adversarial examples

Recent progress in the field of computer vision systems powered by neural networks introduced them to many areas which have high-security risks, such as self-driving cars [4] or malware detection [41]. This has become a strong motivation for research in assessing the vulnerabilities of these systems against input which was adversarially malformed by an attacker.

The first study on this topic was performed by Szegedy et al. in [44]. In their paper, they have discovered that neural networks have some non-intuitive characteristics and intrinsic blind spots generated by the training algorithms. They have proved that exploiting these characteristics of the neural networks, one can carefully modify the input fed to the network in such a way that the output will be completely different from the correct output. In their paper, Szegedy et al. refer to these inputs as "**adversarial examples**" or "**adversarial samples**". We call a method of computing these adversarial examples an **adversarial attack**.

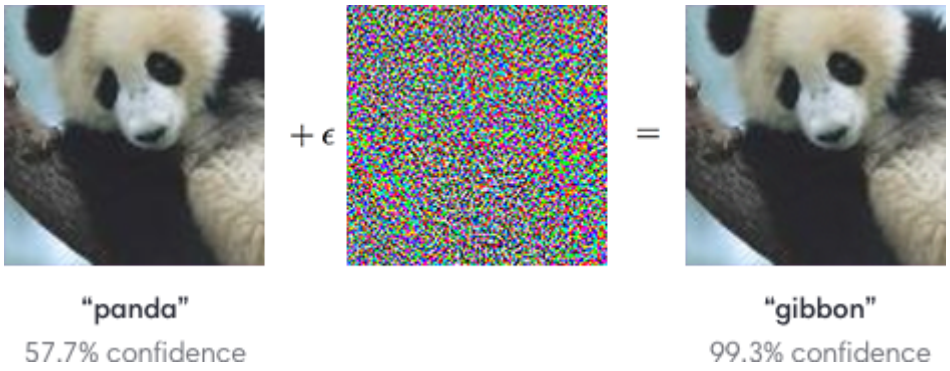

Figure 2.3: Example of how a carefully crafted perturbation can change an image from being classified as a "panda" to being classified as a "gibbon"[4]

In the context of image classification, let $(x, y) \in \mathbb{R}^{w*h*c} \times \mathbb{R}^C$ be a pair consisting of an image x and its associated label y. An adversarial example represents an image $x' \in \mathbb{R}^{w*h*c}$ which is obtained by taking the original image x, and applying a non-random perturbation δ such that the loss $L(\theta, x', y)$ will be maximized [44]. Thus, the adversarial example is computed as $x' \leftarrow x + \delta$. This is depicted in **Figure 2.3**, whereby adding only a small perturbation can make the image to be classified as something totally different and even with higher confidence than the original image.

Adversarial attacks can be performed in two ways: **targeted** and **untargeted**. When performing a targeted attack, the attacker chooses a target class $t \neq y$ and generates an adversarial example x' such that x' is classified by the neural network as the target class t instead of its correct class y. An untargeted attack is a less powerful version of the targeted one, where the attacker tries to generate an adversarial example that will force the neural network to misclassify it.

When considering the model to be attacked, adversarial attacks are split into **white-box** and **black-box** attacks. We refer to a white-box attack when the attacker has access to all

[4]**Source:** Goodfellow et al. [15]

the information about the neural network such as its architecture and the weights and biases We refer to a black-box attack when the attacker does not have access to that information and can use the neural network only as a callable oracle. In this project, we are considering only white-box attacks.

In order to compute the perturbation, Szegedy et al. have used a box-constrained L-BFGS optimization method to minimize the following loss function [44]:

$$\omega \cdot |\delta| + L(\theta, x', y), \text{ subject to } x + \delta \in [0, 1]^{w*h*c} \tag{2.6}$$

Note that the images are considered to have the pixel values scaled between 0 and 1. In this paper, we are making the same assumption on all the images that we are using. We perform this scaling by dividing each pixel by the maximum possible value, namely 255.

The results they have obtained showed that it is possible to use adversarial examples to reduce the accuracy of several networks to 0% while applying perturbations of less than 10%, measured with the L_2 distance metric. Also, they have proved that adversarial examples generated on one network keep their properties as they are being evaluated on other models, even if the models were trained on disjoint datasets and have different architectures. This property is called **transferability** of the adversarial examples.

This discovery was the starting point for research on this topic. Many other methods of generating adversarial examples have been proposed in the literature up to the present day. In the remainder of this subsection, we will describe the ones that will be evaluated in the following sections.

Fast Gradient Sign Method

Starting with the discovery of Szegedy et al. [44], Goodfellow et al. [15] have brought a different theory to justify the existence of adversarial examples, namely the linear nature of the neural networks.

Fast Gradient Sign Method generates an adversarial example by either maximizing the value of the loss function for the correct class in the case of an untargeted attack or by minimizing the value of the loss function for the target class, if the attack is targeted. This is achieved by computing the gradient of the loss function with respect to the input, then adding or subtracting a small portion of it to the input. This method is designed to perform optimal under L_∞ norm restrictions.

For a specific pair of inputs (x, y), where x is the raw input, and y is the correct output assigned to it, an untargeted adversarial example x' is computed as shown in **equation 2.7** and a targeted one is computed as shown in **equation 2.8** [15]:

$$x' \leftarrow x + \epsilon \cdot \text{sign}(\nabla_x L(\theta, x, y)), \tag{2.7}$$

$$x' \leftarrow x - \epsilon \cdot \text{sign}(\nabla_x L(\theta, x, t)), \tag{2.8}$$

where ϵ represents the maximum perturbation allowed, measured with the L_∞ norm.

When assessing the performance of the adversarial samples generated with this method, Goodfellow et al. have been able to obtain a 99.9% error rate while attacking an MNIST classifier while $\epsilon = 0.3$ and an error rate of 89.4% with $\epsilon = 0.1$ [15].

The advantage of this method is that it is very fast. This allows adversarial examples to be used during the training procedure in order to increase the robustness of the network to adversarial input. This will be discussed in more detail in the **section 2.4**.

Basic Iterative Method

Basic Iterative Method has been proposed as an extension to Fast Gradient Method. The motivation behind this method is that better adversarial examples can be generated when iteratively updating the perturbation by a small amount at each step and clipping the results to be in the neighbourhood defined by the maximum perturbation allowed, measured with the L_∞ norm [25].

For generating an adversarial example, Basic Iterative Method starts from the original image, then at each step a small fraction of Fast Gradient Sign Method is applied, then the pixel values are clipped so they remain in the neighbourhood of the original image, defined by ϵ and measured with the L_∞ norm. This is described in **equation 2.9** for the untargeted attack and **equation 2.10** for the targeted one with target class t:

$$x_0^* = x,\ x_{n+1}^* = \text{Clip}_{x,\epsilon}\left\{x_n^* + \alpha \cdot \text{sign}(\nabla_x L(\theta, x_n^*, y))\right\}, \tag{2.9}$$

$$x_0^* = x,\ x_{n+1}^* = \text{Clip}_{x,\epsilon}\left\{x_n^* - \alpha \cdot \text{sign}(\nabla_x L(\theta, x_n^*, t))\right\}, \tag{2.10}$$

where α is the update applied at each step, also referred to as the step size, ϵ is the maximum allowed perturbation to the original image, defined in this case through the L_∞ distance, and $\text{Clip}_{x,\epsilon}$ is a function that keeps the values of each pixel of the image inside the neighbourhood defined by the L_∞ norm and the ϵ hyperparameter. Clipping is performed as follows [25]:

$$\text{Clip}_{x,\epsilon}\left\{x^*\right\}(i,j,c) = \min\left\{1,\ x(i,j,c) + \epsilon,\ \max\left\{0,\ x(i,j,c) - \epsilon,\ x^*(i,j,c)\right\}\right\}, \tag{2.11}$$

where $x(i,j,c)$ is the value of the pixel from the c^{th} colour channel, on the i^{th} line and j^{th} column of the image x.

Results have shown that this method produces smoother and more efficient perturbations than FGSM, which have proven to work even in the case when the adversarial examples were printed and then fed to the neural network using the camera of a mobile device [25].

Momentum Iterative Method

Momentum Iterative Method is an improvement to the Basic Iterative Method by using momentum [13]. Momentum is a method of accelerating gradient-based optimization methods by accumulating the velocity vector of the gradients over the iterations. Thus, the steps for each iteration are smoother and poor local optima points are avoided [38].

At each iteration, the perturbation is computed based on the accumulated velocity vector of the gradients. This vector is updated by incorporating the previous gradients with a decay factor, μ, set as an hyperparameter, as shown in **equation 2.12**.

$$g_{t+1} = \mu \cdot g_t + \frac{\nabla_x J(x_t^*, y)}{\|\nabla_x J(x_t^*, y)\|_1} \tag{2.12}$$

Before being incorporated, the gradients are normalized using the L_1 (or any other) norm of themselves due to the fact that their magnitude is different at each iteration. The adversarial example at iteration $t+1$ is computed as follows:

$$x^*_{t+1} = x^*_t + \alpha \cdot sign(g_{t+1}), \tag{2.13}$$

where α is the step size, as in Basic Iterative Method.

Momentum makes the update steps smoother, which makes the algorithm to avoid ending up in points of poor optima and thus having a better performance and transferability across different models compared to Basic Iterative Fast Gradient Sign Method [13].

Projected Gradient Descent Method

Proposed in [30], this method is thought of as a unified view of all the attacks related to Fast Gradient Sign Method. Madry et al. have designed this method in their attempt to find a universal way of generating adversarial attacks, and also to improve the robustness of deep learning models through adversarially aware training (this technique is described in **section 2.4**).

In their work, Madry et al. state that attacks which iteratively apply Fast Gradient Sign Method can be considered as performing "projected gradient descent on the negative of the loss function" [30]. Their attack is stated as an optimization problem which, if we are to consider the context of image classification, aims to find a perturbation which will increase the value of the loss function for the correct class label if the attack is untargeted or decrease it for the target label for a targeted attack, while not exceeding a fixed threshold.

Therefore, Projected Gradient Descent Method starts with a random perturbation, then performs gradient descent for a fixed number of steps to optimize it such that it increases/decreases the value of the loss function for the original/target class, while not exceeding the maximum perturbation allowed.

Elastic Net Method

Proposed in [6], Elastic-Net attack on deep neural networks was inspired from the Carlini-Wagner attack [5], which was the only attack to still be effective on neural networks that were using defensive distillation [37] as a defence mechanism against adversarial attacks. This method is motivated by the fact that all the other attacks are either based on L_2 or on L_∞ norms, but none consider the L_1 norm, which can improve the quality of the generated adversarial examples [6].

This method uses the same loss function as the one defined in the Carlini-Wagner attack [5]. Let (x, y) be a image-class pair, $t \neq y$ a target class and x' be the adversarial image that is to be classified as the target class t. The loss function for targeted attacks is defined in **equation 2.14** and for untargeted attacks in **equation 2.15**:

$$f(x,t) = \max\left\{\max_{j \neq t}[\mathbf{Logit}(x)]_j - [\mathbf{Logit}(x)]_t, -k\right\} \tag{2.14}$$

$$f(x) = \max\left\{[\mathbf{Logit}(x)]_y - \max_{j \neq y}[\mathbf{Logit}(x)]_j, -k\right\} \tag{2.15}$$

In both cases, $\mathbf{Logit}(x) = [[\mathbf{Logit}(x)]_i, ..., [\mathbf{Logit}(x)]_c] \in \mathbb{R}^C$ is the layer right before applying the Softmax layer. This layer is also called the **logits** layer. The parameter $k \geq 0$ is a parameter specifying the confidence level, namely the gap between the value of the logit for the target class and the value of the logit corresponding to any other class than the target label with the maximum value [6].

The effect of this loss function is as follows. In the case of a targeted attack, the loss function aims to maximize the value of the logit corresponding to the target class, while decreasing the value of the logit corresponding to the correct class. In the case of an untargeted attack, the loss function aims to minimize the value of the logit corresponding to the correct class and maximize the value of the largest logit form the remaining ones.

Having defined the loss function, Elastic-Net Attack generates adversarial examples by solving the following optimization problem:

$$\begin{aligned} \text{minimize}_x \; & c \cdot f(x,t) + \beta \cdot \left\|x' - x\right\|_1 + \left\|x' - x\right\|_2^2 \\ & \text{subject to } x \in [0,\ 1]^p, \end{aligned} \tag{2.16}$$

where $f(x,t)$ is defined as above, $c,\ \beta \geq 0$ are parameters for the regularization of the loss function and the L_1 norm and the box constraint $x \in [0,\ 1]^p$ ensures that the values of the pixels are kept in the limits of the values of a scaled image. The optimal value for the constant c is found through performing a binary search. Note that in the optimization problem there are two more terms apart from the loss function. These two terms are part of the Elastic-Net regularization technique, which is commonly used in problems that require high-dimensional feature selection [52], and implies a linear combination of two penalty functions for L_1, respectively L_2 norms.

This optimization problem is solved by using the Interactive Shrinkage-Thresholding Algorithm (ISTA) [2]. At each iteration of this algorithm, an adversarial example is computed as follows:

$$x'^{(k+1)} = S_\beta(x'^{(k)} - \alpha_k \nabla g(x'^{(k)})), \tag{2.17}$$

where $g(x)$ is the function to be optimized as described in **equation 2.16**, α_k is the step size for the k^{th} iteration and S_β is the projected shrinkage-thresholding function [6].

After generating the adversarial examples for each iteration, the best one is selected based on the smallest distortion with respect to the original image x defined either by the elastic-net regularization metric, or by the L_1 norm.

Based on the experimental results, this method is considered to be the state of the art in generating adversarial examples for attacking neural networks, achieving both high rates of success of targeted attacks, even on ones using defensive distillation as a defence [6].

Jacobian Saliency Map Method

In his paper, Papernot et al. [34] propose a new class of methods to generate adversarial examples, different from the methods derived from Fast Gradient Sign Method. Their approach is based on understanding the relationship between the inputs and the outputs of a neural network. In other words, they are using the derivative of the function defined by the network with respect

to the input to define a matrix called a saliency map. Each element of this matrix shows how changing the corresponding element in the input affects the output of the neural network.

Let $f : \mathbb{R}^n \to \mathbb{R}^m$, $f(x) = y$ be the mapping function defined by the neural network. The saliency map is defined as follows [34]

$$\nabla f(X) = \frac{\partial f(X)}{\partial X} = \left[\frac{\partial f_j(X)}{\partial x_i}\right]_{i \in 1...n, j \in 1...m} \tag{2.18}$$

The forward derivatives are computed using a method similar to the one applied in back-propagation, starting from the input layers and propagating the gradients froward towards the output layer using the chain rule. Thus, the $(i, j)^{th}$ element of the saliency map shows how the j^{th} element of the output changes if the i^{th} element of the input increases. In the context of image classification, the $(i, j)^{th}$ element in the saliency map shows how the probability of the j^{th} class changes by increasing the value of the i^{th} pixel.

Let (X, y) be a pair of image - true label from the test set and let $t \neq y$ be the target label. The features to be increased by an adversary in order to generate an adversarial example are selected based on the scores computed in the following *adversarial saliency map* [34]:

$$S(X,t)[i] = \begin{cases} \left(\frac{\partial f_t(X)}{\partial X_i}\right) \left| \sum_{j \neq t} \frac{\partial f_j(X)}{\partial X_i} \right|, \; if \; \frac{\partial f_t(X)}{\partial X_i} > 0 \text{ and } \sum_{j \neq t} \frac{\partial f_j(X)}{\partial X_i} < 0 \\ 0, \text{ otherwise} \end{cases} \tag{2.19}$$

This method also allows input features to be decreased to generate adversarial examples that will be classified as a target class t. The pixels to be changed are selected based on the scores from an adversarial saliency map which is defined similarly to the one in **equation 2.19**. Decreasing input features has proven to be less effective as shown in [35].

For efficiency reasons, the implementation of this method selects pairs of pixels that give the maximum change in the probability of the target class and increases/decreases their values. This process is repeated until either the adversarial sample is classified as the target class, the maximum number of iterations is reached, or the maximum allowed perturbation is applied to the image [34]. To measure this perturbation, the authors are using the L_0 distance, wherein the case of RGB images, it is considered to be 1, if the value of a pixel corresponding to one channel is changed. Therefore, if for one pixel the values for all the channels have been changed, then there will be a L_0 perturbation of 3.

DeepFool

DeepFool [31] is an exclusively untargeted attack using the L_2 norm as a similarity measure. It aims to find the smallest possible perturbation that will disrupt the classifier into misclassifying an input image.

In order to generate adversarial examples using this method, the authors are starting from a simplified problem, assuming that the neural network acts as a linear classifier, which uses a hyperplane to separate classes. The adversarial example is then generated in an iterative manner. First, the solution to the simplified problem of the linear classifier is derived in an analytical manner, then the adversarial example at that current step is slightly moved towards

the solution by applying a small perturbation. Then, the whole process is repeated until the adversarial example is able to force the network to misclassify, or the maximum number of iterations is reached.

The adversarial examples generated with this approach only differ slightly from the original images, with perturbations similar to the ones generated by L-BFGS method described at the beginning of this subsection.

2.4 Defence mechanisms

Another area of adversarial machine learning which has got a big amount of attention in the recent years is the study of defence mechanisms against adversarial attacks. This section is intended to give a brief description of the most important defence methods that have been proposed in the literature.

Adversarial Training

Adversarial training is the most common method for increasing the robustness of the neural networks to adversarial attacks. This is the first method of defence against adversarial attacks that was proposed in the literature.

Adversarial training implies using a mixture of the original images and correctly labelled adversarial examples to train the network to learn a more robust decision boundary. This method was first proposed in [44] but was considered as infeasible due to the fact that the L-BFGS attack was slow. With the proposal of Fast Gradient Sign Method [15], using adversarial examples in the training set has become feasible because they could be generated in a short time.

The effectiveness of adversarial training has increased over the years as different methods to generate adversarial examples have been created. Methods like Projected Gradient Descent [30] or Momentum Iterative Method [13] have been created to increase the quality of the adversarial examples, and thus to improve the robustness of the networks when using adversarially enhanced training datasets.

While adversarial samples generated with Projected Gradient Descent improve neural network's robustness against white-box attacks, they are still vulnerable to black-box attacks. Recent experimental results [13] have shown that adversarial examples generated using Momentum Iterative Method are able to avoid points of poor local maxima of the loss function, which allows adversarially trained networks with examples generated through this method not only to resist white-box attacks but also black-box ones. The disadvantage of these two methods is that they do not scale properly to tackling models trained on more complex datasets, for example on ImageNet [12], as it has been shown in [26].

The state of the art in adversarial training is using adversarial examples generated on other models than the one we would like to improve its robustness. Proposed in [45], this method is called "Ensemble adversarial training" and implies generating adversarial examples on various other neural networks models using various of the existing attacks and using them to enhance the training set. Experimental results have shown that this method does scale to complex

models trained on InageNet [12] dataset, and generates neural networks that are resistant both to white-box and black-box attacks.

Defensive Distillation

Distillation [19] was introduced as a technique to reduce the dimensions of a network while preserving its accuracy so it can be used in environments with limited computational resources. To achieve that, two neural networks are used. The first one, which is more complex, or even an ensemble of models is used to label the training set using its output as "soft labels" [19]. Then, a simpler neural network is trained to match these soft labels, instead of matching the hard ones as in regular training.

One particular aspect that allows distillation to work is the fact that it uses a slightly modified Softmax layer, by adding a hyperparameter referred to as "distillation temperature" [19]. The new Softmax output is defined as follows:

$$\text{Softmax}(x, T)_t = \frac{e^{z_t(x)/T}}{\sum_{j=1}^{C} e^{z_j(x)/T}}, \tag{2.20}$$

where the temperature is denoted with T and $z_t(x)$ represents the value of the t^{th} logit.

Distillation requires both networks to be trained with high values for the temperature parameter, but at test time, the temperature should be set to 1. This allows the simpler model to reach accuracy scores comparable with the ones of more complex ones and also have a faster learning time [19].

Papernot et al. [36] have used a slightly modified version of this method to increase the robustness of the neural networks to adversarial attacks. The main difference between their approach and the original distillation is that the models share the architecture, as their purpose is to achieve high adversarial robustness, not to compress the model. This, combined with using high values for the distillation temperature have proven to have negative effects on the success rate of the existing adversarial attacks at that moment, while still preserving the accuracy of the original non-distilled model. Also, by using defensively distilled models, it becomes harder for an attacker to generate adversarial examples. Their experiments have shown that an attacker would have to perturb 500% more input features to successfully generate one [36].

Although it provided good robustness to the existing adversarial attacks at that time, defensive distillation is still vulnerable to more recent attacks like Carlini-Wagner or Elastic-Net [5,6].

Sample Regularisation

Sample regularisation is a defence method against adversarial attacks which has a different approach. Instead of trying to increase the robustness of the network as described in the previous two methods, this approach aims at detecting and removing the adversarial perturbation from the images prior to feeding them to the neural network.

There are several approaches to do so. Gu et al., in [16] try to remove the effect of adversarial perturbations by preprocessing the input images. They try noise addition, using autoencoders and using denoising autoencoders [49]. By applying noise, they manage to reduce the error

rate of the adversarial attacks, but not to completely remove their effects, as it will also be shown in **section 4.1** of this study. Their second approach was training an autoencoder to encode the images into embedded vectors and then to decode them back into images, in order to remove the adversarial perturbations. This proved to be working, but only to some extent: if they stacked the autoencoder on top of the neural network, then attacks could be performed on the whole ensemble. Adversarial perturbations generated from this ensemble have proven to be more subtle than the ones generated directly on the network [16].

A similar approach to this was to use a denoising autoencoder. This means training an autoencoder to remove noise that does not belong to the distribution of the original images. Although it had good performance, this approach also was vulnerable to an end-to-end attack, the same way as the autoencoder was, generating more subtle perturbations.

A more recent approach for sample regularization is "Feature squeezing", proposed in [51]. The authors propose a framework for detecting adversarial examples with low computational cost. Their approach is motivated by the fact that usually, the feature input space is larger than required and this provides an adversary with the necessary freedom to generate adversarial examples. To counter this, their method works through "squeezing out the unnecessary features" [51]. In other words, they are reducing the input of the network to a smaller feature space which contains only the important features. Then, the "squeezed" input is fed to the neural network and the output is compared to the output of the network for the original "non-squeezed" input, by using a distance metric. Based on this distance, the framework decides if an input is considered or not an adversarial input.

Experimental results have shown that this framework is able to successfully detect adversarial examples on all the three datasets that were tested and it proves to be a cheap and promising method for increasing the robustness of neural networks to adversarial examples [51].

In the following section, we will give a formal description of our approach for using synthetic features to increase the robustness of neural networks to adversarial examples and also we will provide a motivation for our design choices.

3. Design

3.1 Synthetic features

The second part of this study is focused on experimenting with a new architecture based on synthetic features to improve the robustness of neural networks to adversarial attacks. We refer strictly to the context of neural networks being used for classification tasks This section aims to give an in-depth explanation of the motivation behind this method and also a description of the solution that has been implemented.

Firstly we begin by giving a definition of what a synthetic feature represents. We consider a **synthetic feature** a pseudo-feature of the input of a neural network that separates the possible classes into two equal groups. If we are to consider the task of digit classification, which has 10 possible classes, let the two groups be {0, 2, 3, 8, 9} and {1, 4, 5, 6, 7} and the synthetic feature be "upper loop", we can say that the digits in the first group do have this feature, while the ones in the second group do not have it.

In our approach, the features that we are considering are not required to have meaning for humans, like the one given as an example earlier. Thus, if we consider 10 possible classes, there will be ${}^{10}C_5 = 252$ such features.

The method that we propose is to have a neural network for each of these synthetic features, which will be able to detect the presence or the absence of that feature in the input image. Thus, for the scenario of digit recognition, we will have 252 individual neural networks that are extracting the features from the input image. Using these extracted features, we deterministically find the class that the input is most likely to belong to. Also, we add an extra output class, namely the "incorrect input" class which will specify if the network considers an input as not belonging to any of the possible classes.

The motivation behind this approach is that having individual neural networks that are independent of one another will increase the difficulty for an adversary to successfully generate an adversarial example. In our hypothesis, this is due mainly to the fact that there will be difficult for the adversary to find a perturbation such that it will fool all the individual networks at once. Thus, the adversarial example will generate inconsistency in the synthetic features that are found by the networks, and it will be classified as "incorrect", or as we will refer to for the context of digit recognition: "NaN" (Not a Number).

The architecture of our system is made up of two major components. The first component is the feature extractor which has the role to detect the presence or absence of a synthetic feature in the input that is fed to it. As described earlier, each feature is extracted by an individual

neural network, thus, our feature extractor component is made up of multiple neural networks, one for each feature to be extracted. We are going to refer to these small networks as "mini-networks". The architecture of the mini-networks that was used in the evaluation of our system will be described in **section 4.2**.

At runtime, each of these networks receives a copy of the input data, does some processing in its hidden layers and outputs a continuous value between 0 and 1. We have called these values "s-values" and we can interpret them as the confidence of a mini-network with which it considers that the input has the synthetic feature it is looking for. Referring to **figure 3.1**, a value for s_i close to 0 suggests the absence of the i^{th} synthetic feature from the input, while a value close to 1 suggests the exact opposite. We will refer to the layer constructed by all these s-values as the "S-layer".

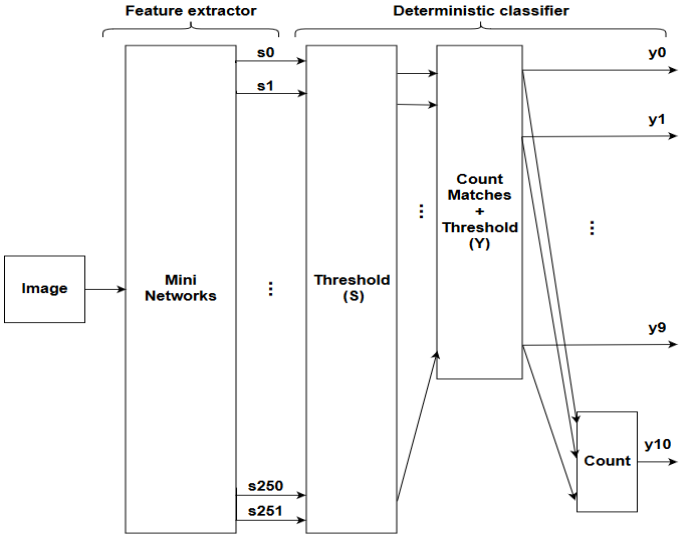

Figure 3.1: Architecture of our synthetic network

The second major component of our architecture is the deterministic classifier. Its role is to use the features extracted by the first part of the network described above and determine how consistent they are with the true features of each output class. Also, at this step we are adding an extra output class, the "incorrect" class, or how we refer to it in the context of digit classification, the "NaN" class. The role of this class is to be triggered when the deterministic classifier is not able to find sufficient consistency in the features that have been extracted by the mini-networks.

To give a more detailed description of how the deterministic classifier works, we will start with the workflow from the S-layer described above. The first step that we do is to apply a threshold on the s-values using our S-threshold, which will turn them from soft to hard values (0 or 1). By using this threshold, we have a finer control on what we are considering as being an existing feature and what not. Should we consider an example, let S-threshold be 0.6, then we are considering a feature s_i being present in the input, only if the neural network that is in charge of detecting it is more than 60% confident in its detection.

The second step of our workflow is to determine which is the class the input belongs to. For this, we are counting the number of matches between the detected synthetic features and the true expected output for each of the possible classes. We then scale it down to values between 0 and 1 by dividing with the maximum number of possible matches. We consider these values to be likelihoods of the input belonging to a specific class. Then, we apply a second threshold, namely Y-threshold to these likelihoods in order to obtain the $y_i,\ i \in 0, ..., C-1$ values, where C is the number of possible classes. This threshold allows us to have control over the proportion of features that have to match in order to consider an input as belonging to a specific class.

The final step is to determine if there is the case to output the "incorrect" class for the given input. We do this in the following way:

$$y_C = \begin{cases} 0, \text{ if exactly one } y_i = 1,\ i = 0, \ldots, C-1 \\ 1, \text{ otherwise} \end{cases} \tag{3.1}$$

In other words, we consider an input as being an incorrect input if either the extracted synthetic features do not match sufficiently enough with the true features of any of the possible classes, or that they match sufficiently with the features of more than one class. Thus, by varying the value of the Y-threshold, we can control how sensitive is the network as classifying inputs as "incorrect". Note that when we decide to output a value of 1 for the "incorrect" class, the values of other $y_i,\ i \neq C$ is set to 0.

The motivation of this approach is that if an adversarial example is fed to the neural network, it will either generate inconsistency in the features detected, which in return will not lead to any of the $y_i's,\ i \neq C$ to be 1, or will generate consistency with the features of more than one class, which will yield in more than one $y_i's,\ i \neq C$ to be 1. In either of the cases, in principle, the adversarial example will be classified as "incorrect", instead of the intended target class.

Apart from the much finer control that they offer on the output of the network, the two thresholding operations, with S and Y are also used to introduce a discontinuity in our workflow. This results in the impossibility of an attack to be performed directly on the output layer of our system as gradients cannot be computed using the chain rule. This does not imply that an attack on our system is impossible. Attacks are possible by targeting the S-layer instead as it can be seen in **section 4.2**, where we will evaluate our method against a set of such modified attacks.

The training procedure for our system is slightly different from a normal neural network. This is due to the fact that the thresholding operations that are performed in the deterministic classifier component. Therefore, as the only trainable parts of the system are the feature extractor mini-networks, then we first train them to reach high accuracy in feature detection and then we stack the second component over it. For efficiency, we are training all the networks in parallel. To achieve that, we had to re-label our training and testing datasets such that each label consists in an array where the i^{th} value is a 1 if the input has the feature i or 0 if not. The labels have been generated by first generating, in a deterministic way, all the possible ways to combine the classes in two equal groups, and then marking the inputs based on their class as follows: 1 if the class is in the i^{th} group, else 0.

In the following section, we will give a detailed description of our experimental setup for both parts of the study.

4. Experimental setup

In this section, we will give a detailed description of the experiments that have been performed in both parts of this study. Neural network models were built and trained using the **Keras 2.0** framework, with a **Tensorflow 1.8** backed. All the implementations were done using **Python 3.5**. The computation for the experiments has been performed using **Google Cloud Platform**, on an instance equipped with an 8 core CPU, Nvidia K80 GPU and 30GB of RAM.

4.1 Adversarial Robustness

As mentioned in the introduction, the first part of this study consists in evaluating how adversarial examples are affected when different transformations are applied to them, and how well they perform when evaluated on a model different than the one that they were generated on.

Our approach is the following. First, we select the neural network models on which we are going to perform the experiments. Then, we select the test images, generate adversarial examples for them using the set of attacks that we are studying and evaluate their performance using a series of metrics that will be described later in this section. Finally, a set of transformations is applied to the adversarial examples and then they are evaluated based on the same metrics. Next, we will give a more detailed description of each of these steps.

Datasets and Test Images

For evaluating the robustness of the adversarial attacks we have decided to use the most popular datasets from the area of image classification, namely MNIST [27], CIFAR10 [22], and ImageNet [23].

In our experiments, we are using subsets of these three datasets, which were selected as follows. For MNIST and CIFAR10, we have selected 1000 images from the test sets (100 for each class) that are classified correctly by our models. For ImageNet, we have randomly selected 100 images that are correctly classified by our model.

In this study, we have opted to use smaller datasets for evaluation due to the fact that some of the attacks, such as Elastic-Net, are really time-consuming to run at their optimal parameter settings and thus, quality of the adversarial examples has been chosen over quantity.

Neural network models

For our experiments, we have trained our own classifier neural networks for MNIST and CIFAR10 datasets and we have used pre-trained models for the ImageNet dataset. For each dataset, two models were required: the main one, on which the adversarial examples are generated, and a secondary one, which is used to evaluate their transferability.

As mentioned, for MNIST and CIFAR10 we have built and trained our own models. The architectures used for these models are described in **Table 4.1** where the main classifiers are referred to as "MNIST", and "CIFAR10", and the secondary classifiers are referred to as "MNIST_s" and "CIFAR10_s".

Table 4.1: Model Architectures (Conv=Convolution Layer, Dense=Fully Connected Layer)

Layer	**MNIST**	**MNIST_s**	**CIFAR10**	**CIFAR10_s**
ReLu Conv.	32 filters (3x3)	32 filters (3x3)	64 filters (3x3)	64 filters (3x3)
ReLu Conv.	32 filters (3x3)	32 filters (3x3)	64 filters (3x3)	64 filters (3x3)
ReLu Conv.	-	32 filters (3x3)	-	64 filters (3x3)
Max Pool.	2x2	2x2	2x2	2x2
ReLu Conv.	64 filters (3x3)	64 filters (3x3)	128 filters (3x3)	128 filters (3x3)
ReLu Conv.	64 filters (3x3)	64 filters (3x3)	128 filters (3x3)	128 filters (3x3)
ReLu Conv.	-	64 filters (3x3)	-	128 filters (3x3)
Max Pool.	2x2	2x2	2x2	2x2
ReLu Dense	256 units	256 units	256 units	256 units
Relu Dense	256 units	256 units	256 units	256 units
Softmax Dense	10 units	10 units	10 units	10 units
Accuracy	99.66%	99.67%	83%	84.5%

It is noticeable that the difference between the main and the secondary models in both cases is the fact that there is an extra convolution layer in each block of convolutions-pooling operations. We have chosen to have a more complex model to test the transferability of the adversarial attacks as it is a more likely scenario for a black-box attack: adversarial examples are generated on a simpler model and then they are used on a more complex one.

All the models were trained using an Adam optimizer [20], with a learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$. As regularization methods, we have used data augmentation, weight decay with $\alpha = 0.00001$ and dropouts with probability 0.5.

In the case of ImageNet, we have used a pretrained InceptionV3 [43] which achieves 78.8% top-1 respectively 94.4% top-5 accuracy as our main model and a pretrained Xception [7] which achieves 79% top-1 respectively 94.5% top-5 accuracy as our secondary model. Both models were included in the Keras framework.

Attacks

The attacks that are being evaluated in this paper are the ones that have been described in **section 2.3**, namely: Fast Gradient Sign Method (FGSM), Momentum Iterative Method (MIM),

Projected Gradient Descent Method (PGDM), Jacobian Saliency Map Method (JSMM), Elastic-Net Method (ENM) and DeepFool (DF). The adversarial attacks were implemented using the **cleverhans v2.1.0** [33] framework, which provides standardized versions of the attack algorithms for experimental use.

Most of the parameters for the attacks are used with the default values that have been defined in the cleverhans framework. For FGMS, PGDM and MIM, we have used a value for the maximum applied perturbation (eps) of 0.3 for all the models. PGDM and MIM have set to run each for 100 iterations on MNIST and CIFAR10 and 30 iterations on ImageNet with a step size on each iteration of 0.01.

JSMM was run only on MNIST and CIFAR10, as it has proven infeasible to run for a classifier trained on ImageNet due to computational limitations. We have used a value of 0.15 for the maximum percentage of perturbed features.

For EMN, we have set a batch size of 500 for MNIST and CIFAR10 and 100 for ImageNet, a learning rate of 0.01 for MNIST and CIFAR10 and 0.5 for ImageNet, and it was configured to run 6 binary search steps for all the models.

DeepFool has been used with the default parameter configuration that is defined in the framework.

In our experiments, we chose to deploy all the attacks in a targeted scenario, except for the DeepFool attack, which is exclusively untargeted. We have used the following methodology. For MNIST and CIFAR10, for each image in the test set, an adversarial example targeted at each of the other 9 incorrect classes has been generated using each attack. This means that 9000 adversarial images have been generated for each attack and for each dataset. For ImageNet, we have randomly selected 10 "incorrect" target classes, different than any of the classes of the images in the test dataset. We have then generated adversarial examples targeted to each of these 10 classes for each attack, which yields a total of 10000 adversarial examples for each attack.

Metrics

The generated adversarial examples have been evaluated based on a series of metrics, which we will briefly describe in the following paragraphs.

The **error rate (ER)** represents the percentage of the adversarial examples that are misclassified by the neural networks. This measures how successful was the attack in reducing the accuracy of the model.

The **attack success rate (ASR)** represents the percentage of the adversarial examples that are classified as the target label. This metric measures how successful is the attack in generating targeted adversarial examples.

The **prediction confidence (PC)** represents the average confidence with which the models classify the adversarial examples. In other words, this metric measures how much an adversarial example fools the neural network into misclassifying it.

The L_0 **perturbation** represents the average percentage of perturbations that have been applied to the original images, measured using the L_0 norm. In other words, this metric measures the average number of pixels that have been changed in the original image to generate the adversarial example.

The L_2 **perturbation** represents the average percentage of perturbations that have been applied to the original images, measured using the L_2 norm. In other words, this metric measures the average Euclidean distance between the original images and the adversarial examples that have been generated from them.

It is to be noticed that the L_0 and L_2 perturbations have been measured only for the plain adversarial examples (the adversarial images without applying any transformation on them).

Transformations

The purpose of this part of the study is to evaluate how the adversarial examples generated by the attacks described above behave when transformations are applied to them. We have decided to experiment with 4 types of transformations, as it will be described in the following paragraphs.

Rotations. The adversarial images are rotated by a fixed angle before being evaluated again. The angle values ranged between $-15°$ and $15°$ with an increase step of $5°$.

Translations. The adversarial images are translated on both the X and Y axis by an amount specified as a fraction of their width respectively height. For example, considering an image from the MNIST dataset, having a size of 28×28 pixels, and X translation of 0.1, then we will shift the image to the right for 3 pixels ($28 \cdot 0.1 = 2.8 \approx 3$). We have used translations of ± 0.05, ± 0.1 and ± 0.15 in different combinations for both X and Y axis.

Gaussian noise. The adversarial examples are modified by adding noise sampled from a Gaussian probability distribution of mean 0 and various standard deviations to each individual pixel. The values for the standard deviation are: 0.05, 0.1, 0.15.

Gaussian blur. The adversarial examples are modified by applying a Gaussian blur kernel of two different sizes 3x3 and 5x5. For each pixel, the Gaussian blur filter computes a weighted average of the pixels in its neighbourhood defined by the kernel, with weights sampled from a Gaussian distribution.

It is to be noted that in the case of rotations and translations, points outside the boundaries of the images may become visible, generating gaps. We use the following strategies for filling those gaps. For MNIST, the gaps are filled with a constant value of 0, which turns them into black pixels, matching the background of the images. For CIFAR10 and ImageNet, the gaps are filled by replicating the values of the nearest pixels to them.

4.2 Synthetic features

In the second part of this study we are experimenting with a new architecture for neural networks in order to increase their robustness to adversarial examples, as described in **section 3.1**. In this section, we will give a description of the setup used for evaluating this method.

Mini Network Architecture

As it was described in **section 3.1**, our system uses a series of neural networks, which we refer to as mini-networks and act as independent extractors of synthetic features. When choosing the architecture for our mini-networks, we have experimented with various complexities for the

model. The architecture of one mini-network that we have concluded to use in our evaluation is described in **Table 4.2**.

Table 4.2: Architecture of a mini-network

Layer	**Size**
ReLu Convolution	8 filters (3x3)
Max Pooling	2x2
ReLu Fully Connected	10 units
Sigmoid Fully Connected	1 unit

As it can be seen, each mini-network has a fairly simple architecture. We have chosen to do so because we wanted to avoid our networks to be overconfident in their predictions of the features. As we have observed in our iterations with different architectures, having a more complex architecture for the mini-networks has proven to make them too confident in their predictions of the features, even when dealing with images consisting of randomly generated pixels. Thus, we have opted for a relatively simple architecture which achieves fairly high accuracy when evaluated on clean, unaltered images.

Test Images

For evaluating our system, we have decided to use the test set of the MNIST dataset for image classification. The test set is made up of 10000 images from 10 classes, representing the digits 0 to 9. Thus, the total number of different equal-sized groups in which these classes can be separated is ${}^{10}C_5 = 252$. Therefore, we have 252 synthetic features that we can use to classify the images.

Number of Mini Networks and Thresholds

In our study, we have decided to experiment with two configurations of mini-networks, namely 252 and 126. The motivation behind using 126 networks is given in the following paragraphs. Let us consider our 10 output classes. There are 252 possible different groups of 5 classes out of the total of 10. We have observed that the groups are complementary to each other, in the sense that the first half of them is the opposite of the second half.

To give a better understanding of this complementarity, let us consider that the i^{th} synthetic feature separates our digits into two groups, namely {0, 1, 2, 3, 4} and {5, 6, 7, 8, 9}. For a specific input, the value of i^{th} synthetic feature will be 1 if the digit represented in the input belongs to the first group, or 0, if it is not, which implicitly means that it belongs to the second group. On the other side, there will also be a j^{th} synthetic feature which does the exact opposite, namely has a value of 1 if the digit present in the input belongs to the second group and 0 if not and implicitly belongs to the first one. Therefore we can consider these two features complementary to each other and use this information to reduce to half the number of features that we are considering, which has a high impact on the width of our model and on the number of parameters.

In the experiments, a value of 0.5 has been used for the S-threshold and a value of 0.9 for the Y-threshold. The total number of parameters and test set accuracy of the two approaches using these thresholds are shown in **Table 4.3**.

Table 4.3: Number of parameters and test set accuracy for the two models

Model	**126 Mini-nets**	**252 Mini-nets**
Number of parameters	≈1.7 million	≈3.4 million
Test set accuracy	95.63%	95.59%

Attacks

It was mentioned in **section 3.1** that using the thresholding layers introduces discontinuity in the system and thus an attack targeting the output layer is not possible. In order to properly evaluate the robustness of the two models, the implementations of the attacks provided in cleverhans were modified to target the S-layers instead of the output layer.

For our evaluations, we have used the following attacks: Fast Gradient Sign Method (FGSM), Basic Iterative Method (BIM), Momentum Iterative Method (MIM) and Projected Gradient Descent (PGDM). All the attacks were evaluated for two values of the maximum added perturbation (eps), namely 0.3 and 1. For the iterative methods (BIM, MIM, PGDM), a step size of 0.05 has been used and the attacks were run for 10 iterations each.

All the attacks are performed both in an untargeted and targeted manner, with targets chosen randomly from the set of 9 incorrect classes.

Metrics

For evaluating the two models, a series of metrics have been measured, which are described in the following paragraphs.

The percentage of adversarial examples labelled as "incorrect" (**NaN**) measures how well does our system perform at detecting if the image that is fed to it is an adversarial example or is a correct image.

The percentage of misclassifications (**Misc**) represents the percentage of the adversarial images that have been fed to the models and are classified as belonging to a class differently than the correct one, and also have not been classified as "Nan". Thus, this metric will show how many adversarial examples still remain undetected to our system.

The **attack success rate (ASR)** represents the percentage of the adversarial examples that are classified as the target label. This metric measures how successful is the attack in generating adversarial examples that are targeted for the S-layer.

The **accuracy (Acc)** measures how many adversarial examples are classified as the correct class of the original image. This measures if our models are able to bypass the perturbation applied to the images and classify them correctly.

In the following section, we will show the results that we have obtained by performing the experiments described in the current section.

5. Results

In this section, we will show the results of the experiments that were described in **section 4**. First, we will evaluate the properties of adversarial examples and how they change when transformations are applied to them and then we will study their transferability and how it is affected by transformations.

Secondly, we will evaluate the performance of our proposed architecture based on synthetic features by generating adversarial examples with modified cleverhans attacks targeted at the S-layers.

5.1 Adversarial robustness

In the first part of this study, we are evaluating the properties of adversarial examples before and after transformations have been applied to them. A detailed description of the methodology used for these experiments is given in **section 4.1**. We will first show the results of evaluating the adversarial examples on the models that were generated on. Then we will evaluate the same adversarial examples on different models than the ones they were generated on to study their transferability.

5.1.1 Main models

We will first analyse the results obtained after evaluating the attacks without any transformation being applied to them. The values of the metrics are given in **Table 5.1**. For each attack individually, we will discuss these values and how they change when transformations are applied to the adversarial examples. Note that when discussing the effects of the transformations, we compare them with the metrics obtained without any transformation.

Fast Gradient Sign Method(FGSM)

FGSM performs poorly in the targeted scenario. The adversarial examples generated with it are achieving a maximum ASR of 14% on the MNIST dataset and a minimum of 1% on ImageNet with CIFAR10 in between. On the other hand, it can be seen that the adversarial images perform fairly well in disrupting the classifier, reaching error rates close to 90% for MNIST and CIFAR10 and relatively smaller for ImageNet, of only 70%.

It can be seen that the classifier trained with CIFAR10 is predicting the class of the adversarial examples with high confidence of 96%, while the one trained on MNIST is less confident in

Table 5.1: Metrics for the adversarial attacks evaluated on the models they were generated on without transformations being applied

Model	Metric	FGSM	MIM	PGDM	JSMM	ENM	DF
MNIST	**ER**	86%	100%	100%	99%	100%	100%
	ASR	14%	100%	100%	97%	100%	-
	PC	70%	99%	99%	54%	53%	55%
	L0	62%	61%	74%	21%	41%	62%
	L2	25%	20%	19%	19%	7%	8%
CIFAR10	**ER**	89%	100%	100%	100%	100%	100%
	ASR	10%	100%	100%	99%	100%	-
	PC	96%	99%	99%	70%	54%	52%
	L0	99%	99%	99%	99%	99%	99%
	L2	28%	10%	7%	8%	1%	1%
ImageNet	**ER**	70%	100%	100%	-	98%	100%
	ASR	1%	100%	99%	-	98%	-
	PC	45%	70%	98%	-	21%	38%
	L0	98%	97%	99%	-	41%	99%
	L2	28%	21%	17%	-	1%	1%

its predictions, having an average confidence of 70%. The classifier trained on ImageNet is the least confident when evaluating adversarial examples, having an average confidence in its predictions of 45%. In the case of MNIST, the adversarial examples have a lower L_0 perturbation, of only 62%, while for CIFAR10 and ImageNet almost all the pixels are modified. The adversarial examples generated on all three datasets have similar values for the L_2 perturbations, of around 25%.

Adversarial examples generated with FGSM on all three datasets appear to be very robust when rotations are applied to them. ASR is slightly reduced, by maximum 3% on the MNIST dataset when the value of the angle is larger than 10° and decreases slower for the other two. ER is more sensitive to rotations, as increasing the angle results in bigger differences, with a decrease by 15% for angles bigger than 10° in the case of MNIST, and smaller decreases in the case of the other two datasets. The prediction confidence is slightly affected in the case of all three datasets.

The same effect appears when adversarial examples are translated over the X and Y axis. We recorded only slight decreases in both ASR and ER for all the datasets, while the prediction confidence actually increases by small amounts. The direction and the amplitude of translation do not seem to have any influence on the amount with which ER or ASR decrease.

Further perturbing adversarial examples by adding Gaussian noise does not appear to have any significant effect for any of the datasets and for any value of the standard deviation. The only notable difference is that when adding noise with a standard deviation of 0.15, the error rate and the prediction accuracy in the case of MNIST slightly increase.

Applying Gaussian blur to the adversarial examples yields some contradictory results. While for MNIST and ImageNet, both ASR and ER are reduced, with considerable amounts for a

bigger kernel size, for CIFAR10, the exact opposite is happening. Both ASR and ER increase by approximately 10% each. In all three cases, applying Gaussian blur reduces the confidence of the prediction.

Overall, we can say that the quality of the adversarial attacks generated with FGSM is average, but their effect is preserved even when transformations are applied to them.

Momentum Iterative Method(MIM) and Projected Gradient Descent Method(PGDM)

We will analyse MIM and PGDM together as the results are very similar. Adversarial examples generated with both attacks manage to increase the error rates of the classifiers to 100% for all the datasets and to be classified as the target labels in all the cases for MNIST and CIFAR10, and only in 99% of the cases for ImageNet. Also, it can be observed that both attacks produce fairly similar perturbations, both measured as L_0 norm, where they record values of around 98% for CIFAR10 and ImageNet, while for MNIST the values differ by a bigger amount with 61% for MIM and 74% for PGDM.

The L_2 perturbations differ only by a maximum of 4% and are situated around 20% for MNIST and ImageNet and 10% for CIFAR10. The most interesting aspect of the adversarial examples is the fact that they are classified with high confidence by the classifiers. For MNIST and CIFAR10, the classifiers are around 99% confident in their predictions, while for ImageNet adversarial examples generated with MIM are classified with less confidence of only 70%. On the other hand, adversarial examples generated with PGDM for ImageNet are classified with a high confidence of 98%.

Results have shown that adversarial images generated with both attacks are robust to rotations for MNIST and CIFAR10 datasets. For these, ER and ASR record slight decreases only for angles bigger than 10°, both metrics being reduced by maximum 25%. On the other side, for ImageNet, the adversarial examples behave differently. While the ones generated with MIM, have a higher robustness for small angle values of less than 5°, still keeping an ASR of above 75% and an ER of over 90%. The ones generated with PGDM are less robust, with ASR decreasing to around 50% and ER to around 70%.

Further increasing the rotation angle shows negative effects on ASR and ER for both attacks, more prominent in the case of PGDM. For MIM, ASR and ER are decreased to a minimum of 15% respectively 70% while for PGDM, they are decreased to near 0% ASR and around 35% error rate.

Applying Gaussian noise to the adversarial examples further increases the differences between the two attacks. Examples generated with MIM have proven to be more robust when disrupted by adding Gaussian noise to them, keeping values above 95% for both ASR and ER on all datasets when the noise with smaller variance (0.05, 0.1). When more disruptive noise is added, the adversarial examples are less successful to be classified as the target class, but we still have an ASR of 70% for MNIST and ImageNet and a smaller value of only 46% for CIFAR10. The error rates are above 90% in all the cases and the classifier is still highly confident in its predictions.

Adversarial examples generated with PGDM appear to be more sensitive to Gaussian noise being added to them. For MNIST and CIFAR10, noise applied in small amounts still does not manage to lower ASR below 97% respectively 83% while the error rates are still above 90%. For ImageNet, the real difference between MIM and PGDM is visible, as for adversarial examples

generated with the latter both ASR and ER are reduced significantly even for noise with small standard deviation, starting from 72% ASR and 80% ER and decreasing to 4% ASR and 40% ER. For PGDM, the confidence of the predictions is also reduced for all the datasets. Thus adversarial examples generated with PGDM are less robust to Gaussian noise than the ones generated with MIM, especially in the case of ImageNet.

Gaussian blur has proven to be the most efficient to decrease the effects of the adversarial examples generated with both attacks. Again MIM has proven to generate more robust adversarial examples than PGDM, recording the largest drops in ASR and ER for CIFAR10, reaching 26% respectively 71% for a larger filter size, while the other two datasets keep an ASR above 45% and an ER over 60%.

For PGDM, we notice that both adversarial examples generated on CIFAR10 and ImageNet are sensitive to Gaussian blur, with ASR below 50% and ER below 70%, while for MNIST the adversarial examples are slightly more robust, keeping an ASR between 41% and 84% and a similar ER.

The prediction confidence appears to be significantly affected by Gaussian blur for both attacks, being reduced to values as low as 30%.

Overall we can state that both attacks generate highly efficient adversarial examples, but the ones generated with PGDM are slightly better. On the other hand, when applying transformations to them, the adversarial examples generated with MIM have proven to be superior.

Jacobian Saliency Map Method(JSMM)

JSMM was run only for MNIST and CIFAR10 due to computational limitations as it was explained in **section 4.1**. The results show that adversarial examples generated with this method perform similar for both datasets, achieving near 100% error rates and being classified as the target label in over 97% of the cases.

The first difference in the results that we can see comparing with the ones of the attacks already analysed is the fact that the classifiers are less confident in their predictions, with an average confidence of 54% in the case of MNIST and 70% in the case of CIFAR10. On the other side, the attack generates smaller perturbations for MNIST, affecting only 21% of the pixels with a L_2 perturbation of 19%. For CIFAR10, it appears that all the pixels are affected but with a smaller L_2 perturbation.

Rotations have a high efficiency in disrupting the adversarial examples generated with JSMA, affecting the ones generated on CIFAR10 more than on MNIST. For MNIST, depending on the amplitude of the rotation, ASR and ER are reduced by amounts starting from 20%, respectively 10% for small angles, and up to 80% and 50% respectively, for angles larger than 10°. For CIFAR10, ASR and ER drop, even more, being reduced by 40% respectively 20% for small angles, and up to more than 90% and 50% respectively for larger rotation angles. On the other hand, as the rotation angle increases, the confidence of the predictions also increases.

Translation highly reduces the effectiveness of adversarial examples generated with JSMM. Again, we observe that the amount with which the effectiveness is reduced is again invariant of the direction or intensity of the translation. In all cases, for MNIST, ASR is reduced to values around 30% and error rates are decreased to approximately 70%, while for CIFAR10, ASR is

reduced to lower values, respectively 11% and error rates to approximately 55%. Again, we observe that the confidence of the predictions increases considerably for both datasets.

Adding Gaussian noise to the adversarial examples generated with JSMM, reduces the ASR, for MNIST to values between 50% and 20% and for CIFAR10 between 25% and 12%. On the other side, the error rate is still kept at around 85% for all the angles and both datasets, therefore adding noise does make the adversarial examples to not be classified as the target classes, but they are still able to force the network to misclassify.

One interesting aspect about adding Gaussian noise to JSMM adversarial examples is that the confidence of the prediction drops when the noise is sampled with a small standard deviation, then it increases above the value obtained for non-altered adversarial samples when the noise is sampled with higher standard deviation.

Applying a Gaussian blur filter on the adversarial images decreases the ASR to values below 10% for both datasets and for both filter sizes. The error rate is decreased also, but not with the same rate, reaching values around 30% for MNIST and 60% for CIFAR10. We observe that the confidence of the predictions increases when a Gaussian blur filter is applied to the adversarial images generated with JSMM.

Overall, JSMM generates adversarial examples that perform well, but are classified with lower confidence than the ones generated with MIM or PGDM. When further transformations are applied to them, the adversarial images lose a significant part of their effects on the neural networks.

Elastic-Net Method(ENM)

Attacks generated using ENM have high performance, being classified as the target labels in all the cases for MNIST and CIFAR10 and in almost all the cases for ImageNet, thus achieving high error rates for all the datasets. The examples are predicted with lower confidence than the ones generated with previous attacks that have been analysed by now, with a confidence of around 50% for MNIST and CIFAR10 and much lower, of only 21%, for ImageNet.

On the other side, the perturbations generated by this attack are more subtle, both measured with L_0 and L_2 norms. It can be seen that less than half of the pixels are affected for MNIST and ImageNet, while almost all the pixels are affected for CIFAR10, but in all three cases, the L_2 perturbation is very small. Therefore perturbations generated by ENM change the values of the pixels but only slightly.

When rotations are applied to adversarial examples generated with ENM, their effect is reduced considerably for all the datasets. On ImageNet, ASR is reduced to near 0% and the error rate drops to values between 25% and 18%. For CIFAR10 an MNIST, ASR is reduced to values near 0% only for angles greater than 5°, while still having adversarial images that are classified as their target labels when they are rotated by smaller angles. The error rates follow the same rule, with values between 65-75% for small angles and decreasing to values between 25-35% for larger rotations. The prediction confidence for all the datasets is increased as the rotation is applied to them.

Applying translation along the X and Y axis on the adversarial examples generated with the ENM attack appears to be invariant of the amplitude or the direction of the translation. In other words, for all the datasets we have obtained similar decreases in the ASR and ER for all

the translations that have been applied. ASR is reduced to values around 20% for MNIST, 10% for CIFAR10 and 1% for ImageNet. Error rates are also considerably reduced to values around 50% for MNIST and CIFAR10 and 20% for ImageNet. Again, as transformations are applied to the adversarial examples, the prediction confidence increases.

Perturbing the adversarial examples using Gaussian noise has proven to affect them considerably, especially on CIFAR10 and ImageNet, with ASR reduced to around 15% for CIFAR10, and near 1% for ImageNet and between 25-40% on MNIST. Error rates are reduced considerably on ImageNet, with only approximately 20% of the examples being misclassified, while for MNIST and CIFAR10 still being kept at around 70%. Again, we observe that adding Gaussian noise increases the prediction confidence by considerable amounts.

Adding Gaussian blur to the adversarial examples appears to completely eliminate their effects on the neural network models, with both ASR and ER being reduced below 10%. On CIFAR10 and ImageNet, while the ASR is reduced to values near 0%, the error rates are higher, around 50%, respectively 25%. The prediction confidence increased considerably for all the datasets when Gaussian blur was added.

Overall, adversarial examples generated by ENM achieve their goals with high success while having smaller perturbations. This in return implies that the adversarial examples have low robustness against transformations being applied to them.

DeepFool

Adversarial examples generated with the DeepFool method are successfully forcing the networks to misclassify all the inputs for all the datasets. The examples are classified with smaller confidence, similar to the ones generated with ENM. Also similar to ENM, DeepFool generates adversarial examples that have small perturbations, more specifically measured as the L_2 norm which both are optimized for.

Adversarial examples generated with DeepFool have proven to have the extremely low robustness to all types of transformations that we have evaluated. For both MNIST and CIFAR10, applying rotations to the adversarial examples yields in a decrease of the error rate to values between 20% and 40%, based on the amplitude of the rotation. For translation, the error rates on MNIST are higher than on CIFAR10, with values between 30% and 50% for the former and between 15% and 30% for the latter. Gaussian noise is able to drive down the percentage of misclassifications only by approximately 30-40% for both datasets. Gaussian blur completely removes the effects of the perturbations for MNIST, while on CIFAR10 around 50% of the adversarial examples are still misclassified. For ImageNet, the adversarial examples are not robust at all, with error rates below 10% for all the transformations. The only values greater than 10% were obtained in the case of Gaussian blur, where error rates were situated around 15%. In all the cases and all the datasets, the prediction confidence increases considerably.

Overall, adversarial attacks generated with DeepFool accomplish their goal with high success while using only small perturbations to do so. Their downside, as in the case of the ones generated with ENM is that they have low robustness to all types of transformations.

5.1.2 Transferability

We evaluate the transferability of the adversarial attacks by using the ones that have been generated for the experiments in **section 5.1.1**, and evaluating them on the secondary models, based on the same metrics. The results of the experiments are shown in **table 5.2**.

We will briefly describe the behaviour of these adversarial examples when the same transformations are applied to them before being evaluated on the secondary models.

Table 5.2: Metrics for the adversarial attacks evaluated on the secondary models without transformations being applied

Model	Metric	FGSM	MIM	PGDM	JSMM	ENM	DF
MNIST	**ER**	86%	90%	88%	67%	35%	29%
	ASR	12%	79%	81%	23%	13%	-
	PC	79%	88%	90%	76%	77%	82%
CIFAR10	**ER**	89%	88%	87%	35%	21%	10%
	ASR	11%	61%	68%	5%	3%	-
	PC	94%	89%	91%	85%	88%	94%
ImageNet	**ER**	61%	56%	19%	-	9%	4%
	ASR	0%	2%	0%	-	0%	-
	PC	46%	46%	69%	-	77%	80%

FGSM

When evaluated on the secondary models, adversarial examples generated with FGSM appear to behave similarly to when evaluated on the models they were generated on. The only notable difference is the fact that for ImageNet, the error rate is slightly lower, of 61%, compared to 70% on the main model. Also, for MNIST, the model is more confident in its predictions. The rest of the metrics have values that differ only by less than 5%.

When evaluating the adversarial examples after the transformations have been applied to them, we have observed that they behave similarly when evaluated on the secondary model, as when evaluated on the one they were generated on.

MIM and PGDM

Adversarial examples from both MIM and PGDM behave fairly similar to each other when fed to the secondary models. For MNIST, both attacks generate adversarial examples that maintain a high error rate of approximately 90%, and a relatively high ASR, of around 80% while having high prediction confidence. For CIFAR10, error rates are still high, situated at around 87-88% but the ASR drops to between 61-68%, which is a considerable decrease compared to when evaluated on the main models. The model predicts the classes for the adversarial examples generated by both attacks with a fairly high confidence of around 90%.

On ImageNet, the performance of the adversarial examples decreases significantly. The ones generated with MIM appear to still be misclassified in 56% of the cases, while the ones generated

with PGDM are misclassified only in 19%. Also, the ASR for both is near 0% which is a major drop from an ASR of 100% which was recorded on the main models. The prediction confidences for ImageNet are also lower than on the main model.

When transformations are applied, the adversarial examples generated with both attacks perform similar on CIFAR10 and MNIST, while on ImageNet the ones from PGDM perform slightly worse than the ones from MIM. On MNIST, the examples keep an ASR of above 55% in almost all cases with a couple of exceptions for Gaussian noise with high standard deviation and Gaussian blur with a larger filter. The error rates are above 80% in all cases, except Gaussian blur, with lower error rates.

On CIFAR10, the ASR varies more, with values between 45% and 65% for rotations and translations, and fairly smaller for Gaussian noise and blur. The examples generated with PGDM appear to be less resistant to transformations.

On ImageNet, the adversarial examples fail to be classified as target labels on any transformation that was applied to them. Adversarial examples generated with MIM manage to keep an error rate of around 50% in all the cases, while ones generated with PGDM have lower error rates between 15% and 25%.

JSMM

When evaluated on the secondary models, adversarial examples obtained through JSMM have rather poor effects. Only 23% of the examples are classified as the intended classes for MNIST, while on CIFAR10 only 5% manage to achieve that. The error rates are also reduced, with only 67% of the examples being misclassified for MNIST and 35% for CIFAR10. On the other side, the adversarial examples seem to be classified with higher confidence by the secondary models, than by the ones they were generated on.

Applying transformations on the adversarial examples obtained through JSMM further reduces ASR for both datasets, on MNIST varying slightly around 20% while on CIFAR10 being reduced near 0%. An interesting aspect that we have observed here is that when applying transformations, the error rates actually increase, by small amounts on MNIST, but with higher values for CIFAR10.

ENM

Evaluated on a different model than the one they were generated on, adversarial examples obtained through ENM appear to perform poorly on all three datasets. On MNIST, only 13% of the images are classified as their target labels, and only 35% manage to force the neural network to misclassify them. The average prediction confidence is higher than the one recorded on the main model.

On CIFAR10, the values for the ER and ASR decrease even more, to 21% of the adversarial examples being misclassified and only 3% being classified as their target class. The average prediction confidence is again above the one of the main model.

On ImageNet, the adversarial examples do not seem to have almost any effect, with only 9% being misclassified and none of them being classified as their target labels. Again, the average prediction confidence is higher than for the main model.

Applying transformations to the images affects the ASR only slightly on MNIST and CIFAR10, while on ImageNet, the ASR is 0% always. Intriguingly, the number of misclassified adversarial examples increases when combined with transformations. For example, applying Gaussian noise considerably increases the error rates for all the datasets.

DeepFool

Adversarial examples generated with DeepFool do not appear to be highly transferable between models. The maximum number of misclassifications is achieved on MNIST, with 29% error rate. On CIFAR10, only 10% of the images are misclassified, while on ImageNet 4% of them are able to force the neural network to misclassify. The average prediction confidence increases considerably compared to the main model.

Applying transformations to the adversarial examples generated with DeepFool does not appear to influence their performance in any considerable way. The most notable difference is obtained when applying a Gaussian noise and blur. Noise with high variance increases the error rates to considerable values for both MNIST and CIFAR10. Also, for CIFAR10, Gaussian blur increases the error rates to a value almost double than the original one.

5.2 Synthetic features

In this part of the study, we are evaluating our proposed architecture based on synthetic features. For a more detailed description of it, please refer to **section 3.1**. The experimental setup for the evaluation is described in **section 4.2**. The results of the experiments are shown in **Table 5.3** for untargeted attacks and in **Table 5.4** for targeted attacks.

We will refer to our model as being robust to adversarial examples if it either classifies them as their correct label, or as "NaN". In other words, a model is robust if the percentage of misclassifications is low.

Table 5.3: **Results for untargeted attacks on the S-layers.** Attacks are performed using both the recommended amount of perturbation added (**eps**) and also the maximum amount.

		126 Mini-Nets			**252 Mini-Nets**		
Attack	**EPS**	**Acc**	**NaN**	**Misc**	**Acc**	**NaN**	**Misc**
FGSM	**0.3**	0%	99.98%	0.02%	0%	99.99%	0.01%
	1.0	0%	100%	0%	0%	100%	0%
BIM	**0.3**	0%	97.96%	2.04%	0%	98.12%	1.88%
	1.0	0%	98.79%	1.21%	0%	98.53%	1.47%
MIM	**0.3**	0%	98.66%	1.34%	0%	98.19%	1.81%
	1.0	0%	98.79%	1.21%	0%	98.28%	1.72%
PGDM	**0.3**	0%	99.78%	0.22%	0%	99.87%	0.13%
	1.0	0%	100%	0%	0%	100%	0%

Analysing the results, we have observed that our models behave differently based on the scenario the attack is performed in. When adversarial examples are generated using untargeted

Table 5.4: **Results for targeted attacks on the S-layer.** The attacks are performed using both the recommended amount of perturbation added (**eps**) and also the maximum amount. The target classes are chosen randomly.

		126 Mini-Nets				**252 Mini-Nets**			
Attack	**EPS**	**Acc**	**NaN**	**ASR**	**Misc**	**Acc**	**NaN**	**ASR**	**Misc**
FGSM	**0.3**	0.02%	99.95%	003%	0.03%	0.15%	99.82%	0.03%	0.03%
	1.0	0%	100%	0%	0%	0%	100%	0%	0%
BIM	**0.3**	0.05%	30.15%	69.8%	69.8%	0%	17.12%	82.88%	82.88%
	1.0	0.05%	25.09%	74.86%	74.86%	0%	11.35%	88.47%	88.47%
MIM	**0.3**	0.02%	54.13%	45.85%	45.85%	0.02%	38.41%	61.57%	61.57%
	1.0	0%	46.12%	53.88%	53.88%	0.01%	29.82%	70.17%	70.17%
PGDM	**0.3**	0.06%	55.79%	44.15%	44.15%	0%	29.09%	70.91%	70.91%
	1.0	0%	99.68%	0.32%	0.32%	0%	84.05%	15.95%	15.95%

attacks, the results in **Table 5.3** show that both our models are able to label more than 98% of them as "NaN" for all the attacks that have been studied. Also, we have observed that none of the adversarial examples is classified as belonging to the correct class, as both models have accuracy levels of 0%.

In the case of targeted attacks, our models behave differently based on the attack that is used. It appears that both the models are very robust to adversarial examples produced with FGSM and slightly less robust for examples generated with PGDM. On the other hand, images perturbed with MIM and BIM appear to be more successful in forcing our models to misclassify. We can see that for all the attacks, the adversarial examples generated are classified as the correct label in fewer than 1% of the cases. Also, we have observed that if adversarial examples are not classified as their correct label, or as "NaN", then they are classified as the target label in all the cases. Next, we will discuss the results of each attack individually.

FGSM

Adversarial examples generated using FGSM in both configurations are successfully classified by our models as "NaN" in more than 99% of the cases for the recommended amount of perturbation added, and in all cases when all the perturbation is added. We suspect that this is due to the fact that FGSM is not able to perform well as a targeted attack, as it was shown in **section 4.1**.

Thus, FGSM is not able to successfully perturb the adversarial examples it generates such that they will present all the synthetic features of the target class while removing the ones from the original class. Therefore, the adversarial examples are not able to fool all our mini-networks in order to produce an output consistent enough with the target class, or with any other class.

BIM

BIM appears to be perturbing the original images in such a way that they are able to fool our feature detector mini-networks to produce an output which is consistent either with the

target class or with any other class. It can be observed that while both our models have a low robustness to this attack, the model with 126 mini-networks outperforms the more complex one.

It can be seen that the difference between limiting the maximum perturbation that can be applied or applying all of it does not yield in a highly different rate of misclassifications or adversarial examples labelled as incorrect. We have recorded a decrease of only 5-6% in the rate of adversarial examples labelled as "NaN" when we have applied the maximum allowed perturbation. For the 126 mini-networks model, we have approximately 30.15% of the adversarial examples classified as "NaN", when the perturbation is limited, and 25.09% of them, when the perturbation is not limited. Only 0.05% of the adversarial examples are not able to fool the models, while the rest are classified as the target classes.

For the model with 252 mini-networks, there are fewer adversarial examples classified as "NaN", only 17.12% when a limited perturbation is added and 11.35% when all the perturbation is added. The others are misclassified as the intended target class.

MIM

A similar behaviour as described for the previous attack can be observed in the case of MIM. Again this method appears to be generating perturbations that can fool our mini-network into believing that the features of the target classes are found in the adversarial examples, while the original synthetic features are not. This yields in a smaller percentage of adversarial examples labelled as "NaN" than in the case of FGSM, but almost twice as many when compared to the ones generated with BIM.

Thus, for 30% maximum perturbation applied, we have approximately 54.13% of the images classified as "NaN" in the case of the model with 126 mini-nets and a lower 38.41% for the one with 252. When all the perturbation is applied to the images (eps=1.0), we record a slightly smaller number of adversarial examples being labelled as incorrect, with only 46.12% for the model with 126 mini-nets and 29.82% for the one with 252.

In all cases, the adversarial examples that are not classified as "NaN" are being classified as the intended target label.

PGDM

When evaluating our models to adversarial examples generated with PGDM, we have observed that there is a more prominent difference between our two models. In other words, the model with fewer mini-networks appears to label almost twice as many adversarial examples as "NaN" than the more complex one when the maximum perturbation applied is only of 30%. As it can be seen in **table 5.4**, we have 55.79% of the adversarial examples labelled as "NaN" for the first model and 29.09% for the later.

When the maximum perturbation is applied to the adversarial images, both models classify them as "NaN" with higher rates, but still, the one with 126 networks outperforms the more complex one with 99.68% of the images classified as "NaN" for the former and fewer, 84.05% for the later.

In the following section, we will summarize the experimental results and we will perform a critical analysis of our system.

6. Discussion

In this section, we will give a summary of the findings that we have obtained from this study. Also, we will discuss the downsides of our approaches, and we will try to provide future improvements to overcome these disadvantages.

6.1 Adversarial robustness

The first part of the project was focused around studying the properties of adversarial examples and how these change when they are evaluated on a different model from the one they were generated on and when transformations are applied to them.

Firstly, our findings have shown that all the attacks that have been studied, except Fast Gradient Sign Method, are able to generate adversarial examples that force our neural network models to misclassify in almost 100% of the cases. These adversarial examples also succeed in being classified as their target label in almost all the cases.

Secondly, we have discovered that the Projected Gradient Descent Method(PGDM) and Momentum Iterative Method(MIM) generate adversarial examples with the highest transferability amongst the ones that have been studied. Out of these, the ones generated with MIM has proven to be preserving its properties on the second model even in the case of the ImageNet dataset. These results confirm the purpose MIM was designed for, which is to generate adversarial examples that perform well as black-box attacks [13].

Also, we have discovered that some the adversarial examples generated from some attacks, like PGDM and MIM are classified more confidently than others, such as Elastic Net Method(ENM) or Jacobian Saliency Map Method(JSMM). We suspect that this is happening due to the fact that the formers are optimized under the L_∞ distance metric and produce more rough and visible perturbations, while the latter are optimized under the L_2 metric and produce more fine and subtle perturbations.

Evaluating the adversarial examples after transformations have been applied to them has given some interesting results. Firstly, we have discovered that the most efficient way to reduce the effectiveness of the adversarial examples on neural networks out of the ones studied is using Gaussian noise, or Gaussian blur, the latter being more effective. These results are also supported by the fact that in [16] the authors also manage to reduce the effects of adversarial examples by using these two methods.

Again attacks optimized under L_∞ distance metric have proven to generate perturbations that are more robust than the ones generated from attacks optimized under L_0 or L_2 distances

when further transformations are applied to them. In other words, the cost of producing adversarial examples with finer perturbations, using attacks like ENM or JSMM is that they are classified with less confidence, have low transferability and perform poorly when transformations are applied to them.

Overall we have observed that some attacks produce adversarial examples that are more robust to transformations than others and have high transferability between models. One interesting aspect that we have observed was that adding Gaussian noise with high variance usually increased the error rate. One may argue that this is happening because the noise perturbs the image such that it becomes unrecognisable to the network even without the adversarial perturbation added to it.

In order to answer this question, we have evaluated the original test set images without any adversarial perturbations while applying the same transformations on them. While it was not the case for ImageNet and MNIST classifiers, for CIFAR10, the error rate increases more when images perturbed by Gaussian noise or blur are fed to them.

Thus, while for MNIST and ImageNet, the increase in error rates for the adversarial examples when applied Gaussian noise or blur cannot be motivated by the fact that the models are not robust to these types of perturbations, for CIFAR10 it proves to be the exact opposite. We consider this as being a flaw in our approach, which affects to some extent the results of our experiments. For further work, we consider improving the quality of our CIFAR10 classifiers and evaluating the attacks again.

As of directions for extending the results of this part of the study, we consider evaluating the adversarial attacks when multiple transformations are applied at once. Also, we consider extending the range of attacks that are being studied, as the adversarial examples generated with each of them appears to have different properties, therefore, we cannot generalize the ones that we have discovered in this study to all the attacks.

We also consider improving our study of the attacks being performed in the black-box scenario. Our current approach has performed only a shallow evaluation of how the adversarial examples perform when evaluated on different models than the ones they were generated on. We find this a good direction for future work, as black-box attacks are the most common scenario in which the attacks will be performed, therefore it is an important topic to study.

6.2 Synthetic features

The second part of this project was focused on increasing the robustness of neural networks to adversarial examples by giving them a more human-like approach to the classification task.

Overall, the results of the experiments do seem to prove our motivation for this method to be correct. In other words, it can be seen that some attacks, such as FGSM and PGDM find it more difficult to generate perturbations that fool all the feature extractors at once. Thus, this generates inconsistency in the extracted features and the number of matches with the true synthetic features do not exceed the Y-threshold for any of the classes. On the other hand, it appears that methods which apply the perturbation in an iterative manner such as BIM and MIM do manage to fool the feature extractors. We suspect that this is the case due to their iterative nature of computing the gradients at each iteration and adding a small fraction of it to

the adversarial example. This yields into features of the target label being added and the ones of the original label being removed in a gradual approach.

One aspect that we have observed while performing the experiments on our models was the fact that in the context of an untargeted attack, none of the attacks presented above has been able to find perturbations that would force the networks to misclassify with high rates. We remind that in the context of our architecture, we refer to an adversarial example being misclassified if it is classified to a class different than its correct class and it is not labelled as incorrect.

When the attacks are performed in the untargeted scenario, their approach is to generate an adversarial example that increases the value of the loss function for the correct class. In our case, we have the output in the form of multiple synthetic features, but only a small number of combinations of these features translates into our classes afterwards. Therefore, when untargeted attacks are performed, in most of the cases the adversarial examples will produce an output of features which is different from the correct one. This will indeed yield in a high value for the loss function, but the combination of synthetic features extracted by the mini-networks will not match the true combination for any of the possible classes in most of the cases. In our intuition, this is due to the fact that there are too many possible combinations of these features, and only a small number of them actually are translated into our classes.

To give a better understanding, let us consider our 252 features which are translated into 10 classes. There are 2^{252} possible combinations of these features, which is a very large number, but only a really small part of these combinations actually have a meaning and can be translated as our 10 classes.

Also, it is worth mentioning that during our experiments in finding the best architecture for the mini-networks we have discovered that there is a trade-off between adversarial robustness and accuracy. As it can be seen in **section 4.2**, our models achieve accuracy levels of approximately 95% on the MNIST test sets. Although this is not a relatively low accuracy, it is considerably smaller than the state of the art for this dataset of 99.79% [50].

Our point is that the accuracy of the models can be improved by increasing the complexity of the mini-networks and by reducing the value of the Y-threshold, but this comes with the price of the robustness to adversarial examples. Our finding is confirmed in [46] which states that there is always a trade-off between adversarial robustness and standard accuracy. Therefore, one possible direction for future work is to experiment with different architectures for the mini-networks to get a better understanding of how this tradeoff between adversarial robustness and accuracy behaves in the case of our architecture.

In our experiments, we have evaluated our models only on the MNIST dataset. We consider this a flaw in our approach because MNIST is a dataset composed only of grayscale images, which have only a single colour channel. Therefore, they are easier to be processed by our networks in order to extract the synthetic features. In future work, we consider evaluating our architecture on more complex datasets, preferably colour images such as CIFAR10, to see if the results we have obtained in this study generalize to other datasets.

The main downside of our method is its scalability. In other words, having that many neural networks working in parallel implies a great increase in the number of parameters to be optimized in the training process. As we have seen even using simple architectures for the mini-networks

and using only 126 of them still yielded in at approximately 1.7 million parameters for a classifier working on the MNIST dataset, which is quite a large number compared to the conventional networks that are also able to achieve higher accuracy.

Also, if we are to consider more complex datasets, with more than 10 classes, our approach is less likely to work, as, with the increase in the possible number of classes, we will have a large increase in the number of synthetic features to consider. When considering more complex datasets, like CIFAR10 for example which has colour images, the simple architecture for a mini-network that worked on MNIST will not give good results on CIFAR10. Therefore we need to consider increasing the complexity of the mini-networks which will yield an increase in the number of parameters of the model.

We, therefore, identify another direction for future work for our methods, namely decreasing the width of the network, by reducing the number of features we are considering. The problem with this is that we need a way to generate the labels for training the feature extractors. Our current approach, as described in **section 3.1**, is to deterministically generate all the possible groups of the output classes and use them to label our training images. Thus, we are constricted to use a number of synthetic features which is given by the operation "**n choose k**", where **n** is the number of possible classes, and **k** is the number of elements we want to have in a group(in our case, $n/2$).

We are considering a method inspired by the process of distillation [19] to do so. In other words, we could train a regular neural network classifier which has a layer with sigmoid activation functions before the output one. We could then train this model to reach high accuracy in performing classification, then we could remove the output layer from it which will turn it into a feature extractor. Hopefully, the values for the sigmoid activations will be consistent for each class.

We could, therefore, use this feature extractor to label our training and testing datasets with the synthetic features, in a similar manner to which in distillation, the first model is used to label the datasets with "soft" labels [19]. Having our labelled datasets, we can proceed with building our model using the architecture proposed in this study. The advantage would be that we have a higher control over the number of features that are being considered, and therefore we can decrease the number of mini-networks.

Experiments are required to verify if this method would work or not, but theoretically, it appears as a good way to increase the scalability of our architecture. This can be considered the starting point for future research.

7. Conclusions

Deep learning is becoming more popular every day and it is being integrated into increasingly more systems. This makes the existence of adversarial examples to be a serious aspect that has gotten increasing attention over the last years.

In this project, we have studied the limits of adversarial machine learning from two opposite perspectives.

Firstly, we have evaluated how adversarial examples generated by various attacks behave when evaluated both on the models they were generated on and on different models. The results from this parts have given us a better understanding of the properties of the perturbations these attacks are applying on the original images. Also, we have studied how robust the perturbations are when the images are further changed by applying transformations to them. We have discovered that perturbations optimized under L_∞ distance appear to have more robustness to transformations and still keep their effects on the neural networks even when evaluated on a different model than the one they were created on.

Also, we have discovered that perturbations optimized under L_2 distance, although they are more subtle and harder to detect, are less robust when transformations are applied to them, and actually lose a great part of their effects. We have concluded that there is a tradeoff between the subtlety and the robustness of the perturbations introduced by the attacks, as for more resistant adversarial examples, an attack would have to introduce stronger perturbations, which in return will prove easier to detect by humans.

Secondly, we have proposed a new architecture of a neural network that is based on detecting pseudo-features in the input data that can be used to detect and label accordingly adversarial examples that are fed to it. We have proven on the MNIST dataset that this architecture does achieve high rates of detecting adversarial examples generated with Fast Gradient Sign Method and Projected Gradient Descent Method, but are less successful in detecting adversarial examples generated with Basic Iterative Method or Momentum Iterative Method.

These results show that our intuition behind this architecture is correct and through further developments, the rate on which adversarial examples are detected can be improved. We also point towards the downside of this architecture, namely scalability, and theoretically propose a method to increase that in a future work.

Overall, this project has given us good insights on how deep learning algorithms behave in an adversarial context, both from the perspective of the attacker and from the one of the defender.

Bibliography

[1] Mark Andrew. Hall. Correlation-based feature selection for machine learning. 19, 06 2000.

[2] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sciences*, 2:183–202, 2009.

[3] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, Aug 2013.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[5] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017.

[6] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: Elastic-net attacks to deep neural networks via adversarial examples. *CoRR*, abs/1709.04114, 2018.

[7] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[8] T Ciodaro, D Deva, J M de Seixas, and D Damazio. Online particle detection with neural networks based on topological calorimetry information. *Journal of Physics: Conference Series*, 368(1):012030, 2012.

[9] J. Clark, I. Koprinska, and J. Poon. A neural network based approach to automated e-mail classification. In *Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)*, pages 702–705, Oct 2003.

[10] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM.

[11] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

[12] J. Deng, W. Dong, R. Socher, L. 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, June 2009.

[13] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Xiaolin Hu, and Jun Zhu. Boosting adversarial examples with momentum. *CoRR*, abs/1710.06081, 2017.

[14] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay D. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *CoRR*, abs/1312.6082, 2013.

[15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2014.

[16] Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. *CoRR*, abs/1412.5068, 2014.

[17] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, March 2003.

[18] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov 2012.

[19] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. 03 2015.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[21] Eric Knorr. How paypal beats the bad guys with machine learning. *https://www.infoworld.com/article/2907877/machine-learning/how-paypal-reduces-fraud-with-machine-learning.html*, 2015.

[22] Alex Krizhevsky. Learning multiple layers of features from tiny images. 05 2012.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[24] 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 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[25] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016.

[26] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *CoRR*, abs/1611.01236, 2016.

[27] Y. LECUN. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

[28] Yann LeCun, Y Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[29] Junshui Ma, Robert P. Sheridan, Andy Liaw, George E. Dahl, and Vladimir Svetnik. Deep neural nets as a method for quantitative structure-activity relationships. *Journal of chemical information and modeling*, 55 2:263–74, 2015.

[30] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[31] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *CoRR*, abs/1511.04599, 2015.

[32] Dat Tien Nguyen, Firoj Alam, Ferda Ofli, and Muhammad Imran. Automatic image filtering on social networks using deep learning and perceptual hashing during crises. *CoRR*, abs/1704.02602, 2017.

[33] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018.

[34] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *CoRR*, abs/1511.07528, 2015.

[35] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *CoRR*, abs/1511.07528, 2015.

[36] 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.

[37] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597, 2016.

[38] Boris Polyak. Some methods of speeding up the convergence of iteration methods. 4:1–17, 12 1964.

[39] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[40] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[41] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. *CoRR*, abs/1508.03096, 2015.

[42] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016.

[43] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[44] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

[45] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Dan Boneh, and Patrick D. McDaniel. Ensemble adversarial training: Attacks and defenses. *CoRR*, abs/1705.07204, 2017.

[46] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. There is no free lunch in adversarial robustness (but there are unexpected benefits). 05 2018.

[47] Avinash Sharma V. Understanding activation functions in neural networks. *https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0*, 2017.

[48] Marcel van Gerven and Sander Bohte. Editorial: Artificial neural networks as models of neural information processing. *Frontiers in Computational Neuroscience*, 11:114, 2017.

[49] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.

[50] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[51] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *CoRR*, abs/1704.01155, 2017.

[52] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

A. Appendix

Git Repository: `https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2017/bas782`

Project File Structure
The code written in this project is made up of jupyter notebook files and plain python files. We will describe the contents of each folder in the following paragraphs.

In **adversarial_examples** the files related to testing attacks, generating the adversarial examples, and performing the experiments are found. There is one subfolder, namely **adv_metrics**, which contains the results of the experiments stored in .csv files.

In **adversarial_framework** we have python files that contain the helper functions used throughout the project, the custom keras layers and the modified attacks used in the second part of the project. Note that the implementation of the attacks is a modified version of the implementation provided in cleverhans. In **data_processing**, we find jupyter notebook files that were used to select our testing datasets. In **exp_datasets**, we find the datasets that were used in our experiments for the first part of the project. In **models**, we find the models that we have used in the project, both for the first part, which are found in two sub-folders, namely **mnist**, and **cifar10**, and for the second part, which are found in the sub-folder called **synth_nets**.

In **synthetic_features**, we find the jupyter notebook files that were used to train and evaluate the models based on synthetic features, and also files used to experiment with various aspects of the architecture. In **train_models** we find the jupyter notebook files that were used to train the classifiers for MNIST and CIFAR10. In **requirements**, the dependencies for running the code are found.

Running the code
In order to run any of the Jupiter notebooks, first, the root directory must be added to the PYTHONPATH environment variable. This can be done in the following way:

```
For Windows (in command line):
    set PYTHONPATH=%PYTHONPATH%;path_to_the_root_directory
For Linux (in terminal):
    PYTHONPATH="${PYTHONPATH}:path_to_the_root_directory"
    export PYTHONPATH
```

The package requirements are provided in the GitLab repository, in files for fast installation. More information on running the code can be found in **Readme.md** of the GitLab Repository.