Politecnico di Torino

MASTER'S DEGREE IN ELECTRONIC ENGINEERING

Master's Degree Thesis

Analysis of robust neural network classifiers against gradient-based attacks

Supervisors:

Prof. Enrico MAGLI
Prof. Tiziano BIANCHI

Candidate:

Federico MICELLI

Academic Year 2020-2021

Abstract

During the last few years, deep learning has been applied to a huge number of applications in the field of multimedia, scientific research, and industrial processes. Over an increasing number of specific visual classification problems, the performance of these algorithms has reached greater accuracy levels than human capabilities. However, the ever-growing employment of neural networks in our society raises serious warnings in the matter of security, as they can be targeted by malevolent adversaries. Many barriers affect the use of deep neural networks in applications where security is of key importance, such as medical diagnostics and autonomous driving. One of the most severe flaws of deep learning is represented by adversarial attacks, a collection of methods that are designed to interfere with neural networks input data to produce undesired outputs or, in general, to cause algorithm malfunctions and classification accuracy reduction. This happens in the face of perturbations that are very difficult to detect. Indeed, the adversarial examples generated by these attacks are often undetectable by human eyes.
This thesis aims at investigating the adversarial robustness of the **Gaussian Class Conditional Simplex** (GCCS) method, a novel defense against adversarial examples designed at Politecnico di Torino. In particular, after providing insights into the fields of neural networks, deep learning, and adversarial attacks, this thesis reports a thorough experimental evaluation that illustrates the greater robustness of the GCCS method against a multitude of attacks with respect to competing state-of-the-art techniques. First, I carried out experiments with **Adversarial Training** (AT), a common approach based on providing adversarial examples at training time. I show how robustness is greatly improved thanks to AT. Further, I have obtained the most robust model by combining GCCS and other state-of-the-art techniques. Such a model has then been employed in a series of tests devoted to demonstrating that in most cases GCCS classification method outperforms other techniques independently on the considered attack. These results have also been confirmed by plotting features distributions in the latent space: in the case of GCCS, these are well-separable so that high inter-class separation is ensured. I also show how other state-of-the-art techniques tend to mix features belonging to different classes with the subsequent misclassification.

To my family and friends.
A special feeling of gratitude to my parents
for their love, endless support and encouragement.

Acknowledgments

This thesis is part of a research work carried out by the Electronics and Telecommunications Department at Politecnico di Torino in collaboration with Sony R&D Center Europe.
I am grateful to Prof. Enrico Magli and Prof. Tiziano Bianchi for giving me the opportunity of working on such an interesting and relevant field, deepening both neural networks and robustness against adversarial examples fields.
I also thank Research Assistant Andrea Migliorati for assisting and supporting me during this work period.

Table of contents

List of tables

List of figures

Chapter 1

Introduction

1.1 Computer vision and security concerns

During the last years, computers have become more and more important in our lives. They are used to perform several complex tasks that humans would complete in a long time. However, in addition to that, computers have also been placed in the foreground of innovation for particular fields, in order to develop algorithms that could tackle an even wider spectrum of problems. One of the most challenging topics over the years has been Computer Vision (CV), which investigates the capacity of deriving meaningful information from digital images or videos and taking actions based on that information [5].
Computer vision runs analyses of data over and over until it discerns distinctions and ultimately recognizes images. For example, to train a computer to recognize automobiles, it needs to be fed with vast quantities of car images and car-related items to learn the differences and recognize such a vehicle.

An essential technology is used to accomplish this: **Convolutional Neural Networks** (CNN). In particular, CNNs characterized by many layers, named **deep neural networks**, are used.
Machine learning uses algorithmic models that enable a computer to teach itself about the context of visualized data. If enough data is fed through the model, the computer will look at the data and teach itself to tell what an image represents. These algorithms enable the machine to learn by itself, rather than someone programming it to recognize the content of an image.
A CNN, instead, helps a machine learning or deep learning model look by breaking images down into pixels that are given tags or labels. It uses the labels to perform mathematical operations and makes predictions about what it is seeing. The neural

network runs these operations and checks the accuracy of its predictions in a series of iterations until the predictions start to come true. It is then recognizing or seeing images similarly to humans.

The problems related to computer vision are often classification problems. They consist of classifying data into a given set of possible classes, assigning a label to each data entry with certain confidence or probability. Common applications for this approach are algorithms able to distinguish and recognize objects into images or to guarantee security during the authentication phase of a service. As anticipated, neural networks may be employed for many applications: for instance, another kind of problem is named regression. This one is used to estimate numerical values or trends and it has several applications, such as estimation of house prices, stock values, or weather forecasting. However, this thesis will cover only classification problems.
Machine learning is massively used due to the key feature of these algorithms: indeed, after completing training on a given dataset, they can generalize well over previously unseen data. In addition to that, neural networks have reached accuracy levels well over human possibilities over very specific tasks such as cancer diagnostics or video surveillance.

1.2 A relevant flaw of neural networks: adversarial examples

Neural networks could potentially be manipulated and forced to produce an unwanted output, i.e. misclassify a provided input feature. For this reason, the excessive employment of these networks pones some security warnings, especially in fields that involve people's safety and security like medical uses or digital security ones. For instance, in this last case, one can think of an adversary as a malevolent actor who interferes with the neural network of a firewall system to misclassify possible hazards and malware.

In particular, it has been proven that by applying an infinitesimal non-random perturbation to an input, it is possible to arbitrarily change the network output prediction. The perturbed inputs are known as **adversarial examples** [6]. Notice that these modified versions of the input features are so difficult to intercept that they might even be indistinguishable by the human eye. An example and its effect on the network is shown in Figure 1.1.

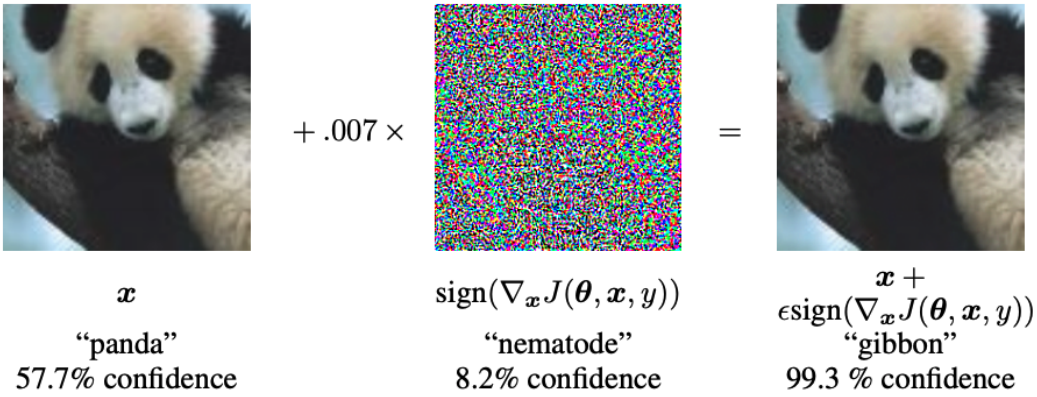


Figure 1.1: Adversarial example generated and applied to misclassify network output prediction. The original input feature is classified as "panda" with a confidence level of 57.7%. After the imperceptible adversarial perturbation is added to the input, the network classifies the input as "gibbon" with very high confidence, despite the perturbed image being indistinguishable from the original one to the naked eye. [1]

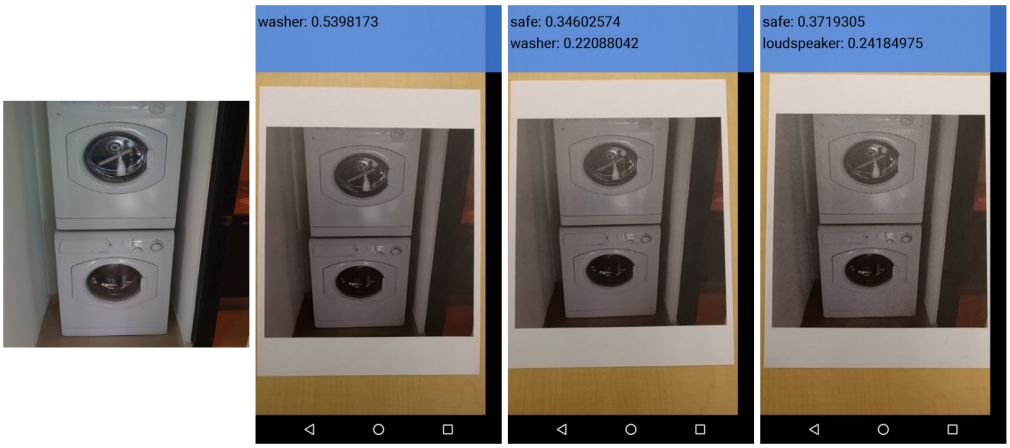
1.2.1 Adversarial examples in the real world

Usually, one can think of these perturbed inputs as features that are directly fed into the machine learning classifier digitally. However, this is not always the case for systems that operate in the physical world, like the ones that receive inputs from cameras or other kinds of sensors.

In this context, adversarial examples are not digital data anymore, but they can be printed images or sounds. For instance, an adversarial example for the voice command domain would consist of a recording that seems to be innocuous to a human observer, such as a song, but contains voice commands recognized by a machine learning algorithm. Furthermore, even an unintelligible sound that a human perceives just as the noise could cause an intentional misclassification.

Moreover, in the field of secure authentication, face recognition systems based on photos are vulnerable to these attacks, i.e. a previously captured photo of an authorized user's face is presented to the camera instead of an actual face to get access. Even more, an adversarial example for this domain might consist of imperceptible markings applied to a person's face, so that a human would recognize their identity correctly, but a machine learning system would not give them access as they are recognized as a different person [2].

As visible in Figure 1.2, a printed image representing a washer is captured by a camera that embeds a machine learning algorithm for classification. It is correctly classified when no perturbation is applied. However, increasing the strength of the attack (perturbation) leads to a degradation of the confidence with the subsequent misclassification.



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

Figure 1.2: Adversarial example belonging to the physical world. ***A*** *- image taken from the dataset;* ***B*** *- clean image printed with correct classification;* ***C*** *- small perturbation applied to the printed image and degradation of the accuracy;* ***D*** *- strong perturbation applied to the printed image and misclassification.[2]*

1.3 Thesis map

This thesis aims at evaluating adversarial robustness on ResNet, an n-dimensional classifier designed at DET Polito in collaboration with Sony R&D Center Europe. This classifier is based on **Gaussian Class Conditional Simplex** (GCCS) method, a novel defense approach that pushes classification accuracy well beyond other state-of-the-art competing methods, especially under attacks.

In chapter 2 I provide technical background on neural networks, introducing them from the elementary node to entire layers. Also, several mathematical functions and operations used in such a context will be explained.

Chapter 3 aims at presenting the GCCS method mentioned before. Insights into the employed encoder, designed loss function, and the criterion to be used as a decision rule will be provided.

In chapter 4 I will focus on many implemented attacks, classifying them and describing technical instructions to develop different attacks. After that, I report several useful suggestions to perform correct defense evaluation, also explaining the importance of stating a threat model.

Chapter 5 is intended to group all the launched experiments, providing a brief description of the test context and commenting on the obtained results. These experiments have the purpose of testing GCCS and other state-of-the-art methods robustness against gradient-based attacks under different conditions.

Chapter 2

Background on Neural Networks

Finding correspondences between image regions (also known as patches) is a key factor in many computer vision applications: due to this fact, various descriptors for patch matching have been proposed to improve accuracy and robustness against variations in perspective and illumination conditions of the input image. Nowadays, the most studied technology when it comes to object classification is the so-called deep learning, which is based on a family of layered, non-linear functions called Artificial Neural Networks (ANNs). ANNs fall in the realm of machine learning and they present several layers and learnable parameters that can hugely vary depending on the specific application. The goal of this new machine learning paradigm, which is inspired by the biological brain, is to learn hierarchical sets of features directly from data and therefore without relying on any previous knowledge on the input data distribution. As said, ANNs architecture is divided into an arbitrary amount of layers. Each layer is formed by many elementary blocks, called artificial neurons. In this chapter, I introduce ANNs from elementary nodes to the structure of entire layers.

2.1 Functionality of an artificial neuron

An artificial neuron is a logical unit able to perform basic mathematical operations, such as additions and multiplications. As visible in Figure 2.1, its scheme recalls the biological neuron. The artificial neuron receives several inputs on the dendrites and produces just one output on the axon.

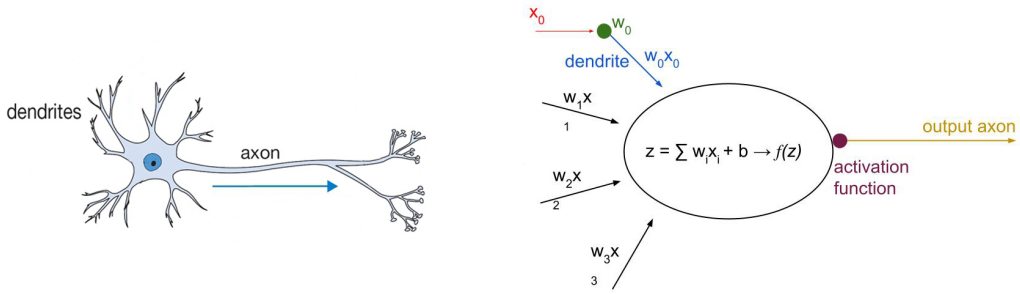


Figure 2.1: Biological neuron represented on the left: dendrites and axon of the cell are indicated. The schematic of the corresponding artificial version is drawn on the right. In particular, inputs, weights, bias, and activation function are highlighted.

Inputs $\mathbf{x_i}$, coming from different dendrites may have different importance so they are weighted with corresponding parameters $\mathbf{w_i}$, where i refers to the i^{th} dendrite. Moreover, another important parameter, called bias (in the figure: b), characterizes the neuron [7]. The weighted signals from all dendrites are summed up together with the bias of the neuron to form the intermediate variable $\mathbf{z}$, which is passed as input to the activation function.

$$z = \sum_{i=0}^{N-1} w_i \cdot x_i + b \tag{2.1}$$

The output of the activation function is propagated on the axon as general output produced by the neuron.

$$o = f(z) \tag{2.2}$$

Activation functions are used to determine the output of the neural network. There are a lot of different functions that can be employed but they all have to be **differentiable** and **monotonic**. The most common ones are listed below.

- Sigmoid
- Hyperbolic tangent (tanh)
- Rectified linear unit (ReLU)

2.1.1 Sigmoid function

The sigmoid activation function is a logistic curve that has the following expression:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The function behaves as in the following.

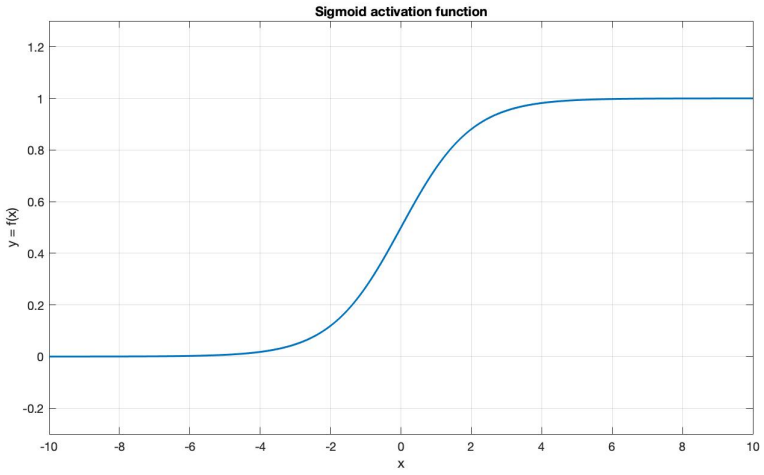


Figure 2.2: Sigmoid activation function.

As it can be noticed by the plot, this function is defined in all $\mathbb{R}$ but only exists in the range (0, 1) and asymptotically reaches the edges of such interval. The reason why it is commonly used is that it models quite well probability distributions since also probability distributions are defined over the same range. Generally, the sigmoid activation is chosen for binary classification problems.

The sigmoid function expresses just one probability distribution. The softmax function extends the concept, as it is more generalistic. It is chosen whenever a multiclass classification is required. In particular, it is expressed as:

$$\sigma(z)_{\mathrm{i}} = \frac{e^{\mathrm{z_i}}}{\sum_{j=1}^{K} e^{\mathrm{z_j}}}$$

where K is the number of output classes so, for each class of the set, it returns the probability associated with it.

2.1.2 Hyperbolic tangent function

Just like the sigmoid function, also the hyperbolic tangent is a commonly employed S-shaped logistic function. However, it enlarges the output range to (-1, 1), so negative outputs are also available.

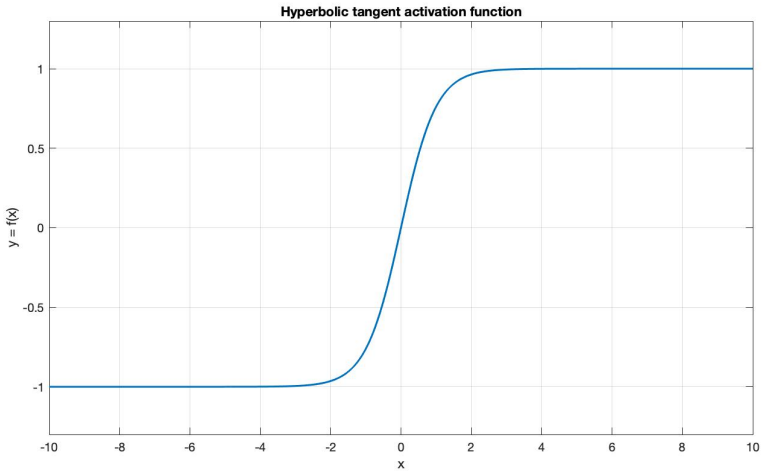


Figure 2.3: Hyperbolic tangent activation function.

As visible in Figure 2.3, the hyperbolic tangent function is 0 centered. For this reason, negative inputs are mapped as strongly negative while infinitesimal inputs are mapped as nearly null. This is a step forward if compared to the sigmoid function that, instead, is 0.5 centered, so negative inputs are mapped close to the 0 region.

2.1.3 Rectified linear unit function

The main limitation of the previously mentioned activation functions is that they saturate. Indeed, for all of them, both large positive and negative inputs tend to an asymptote, causing the so-called **vanishing gradient** issue [8], in which the network gets stuck at a local minimum and does not train efficiently. Rectified Linear Unit (ReLU) solves this problem. This activation function is one of the most used in machine learning. It is described by the following expression:

$$f(z) = max(0,z)$$

As visible in Figure 2.4, two main regions can be distinguished: the rectified one, related to negative inputs, and the linear one, associated with inputs greater or equal to 0. In particular, all negative inputs are mapped as 0, while positive ones are mapped with their value; this makes this function produce output values in the range $[0, +\infty)$.

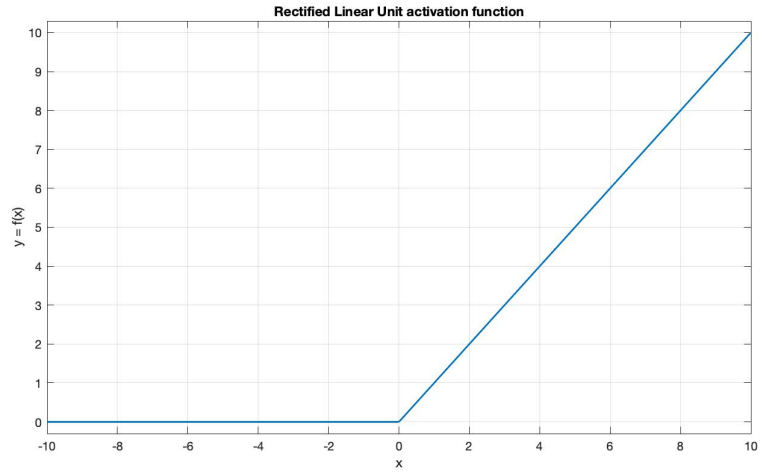


Figure 2.4: Rectified linear unit activation function.

The only problem is that negative values immediately become null, which decreases the ability of the model to fit or train properly. This means that any negative input, given to the ReLU activation function, produces a zero output immediately in the graph, which in turn affects the resulting graph by not mapping the negative values appropriately.
For this reason, leaky ReLU has been introduced. This last function is quite similar to the previous one, but leakage is introduced for negative inputs. It is described by the expression reported below, where a is the leakage factor. The behavior of the leaky ReLU is shown in Figure 2.5.

$$f(z) = \begin{cases} z & \text{if z} \geq 0 \\ az & \text{if z} < 0 \end{cases}$$

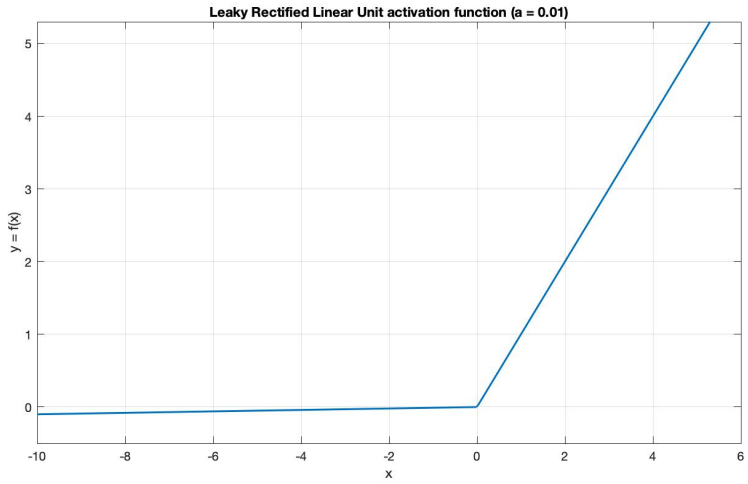


Figure 2.5: Leaky ReLU activation function.

2.2 Layers and architecture of the network

Neural networks are groups of artificial neurons, each one based on the block diagram presented before, combined in specific ways. In particular, they are divided into layers, as shown in Figure 2.6.
The layers at the boundaries are called input layer and output layer respectively. The former is in charge of receiving data as inputs while the latter is used to produce output results of the network. For this reason, they both are also known as interfacing layers. Intermediate layers are called hidden layers: we commonly refer to **deep learning** when dealing with a large number of hidden layers.

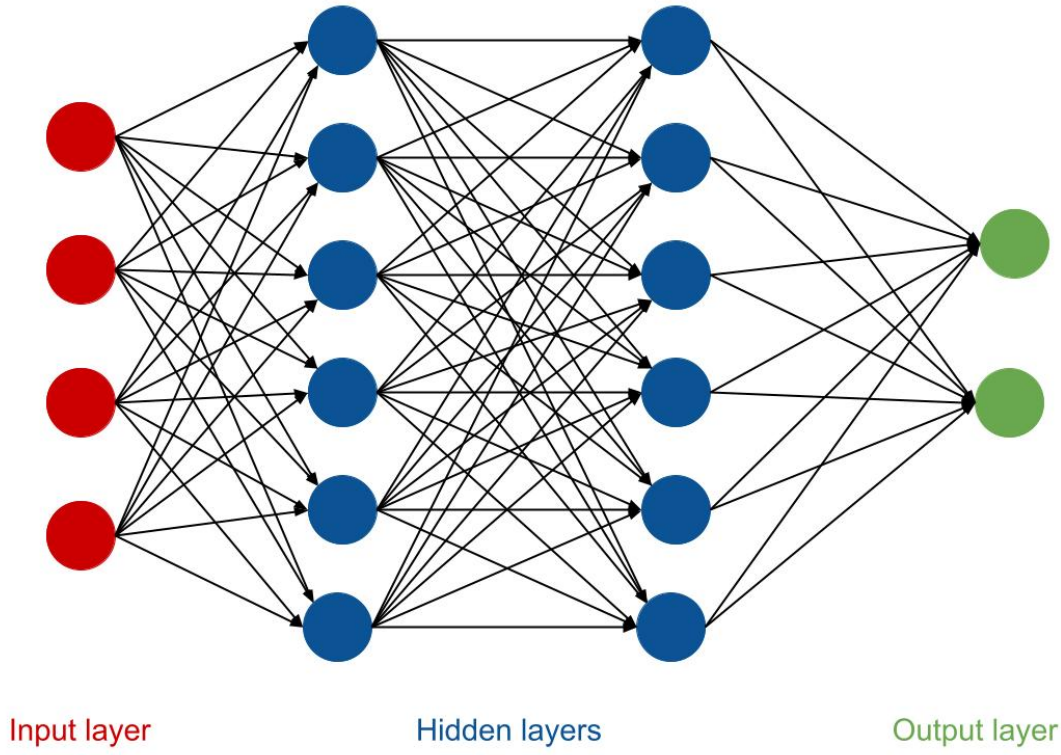


Figure 2.6: Architecture of a neural network considering input, hidden and output layers.

Different layers may have different functions according to how neurons are organized and how they process incoming data. The most important layers are listed in the following.

- Dense layer
- Convolutional layer
- Pooling layer

The first category is characterized by a high density of connections between neurons: indeed, every node is bound to each neuron of the previous layer and each one of the next layer. Just like in a biological brain, this allows creating a lot of synapses, so that every single node contributes to the computation of the final output. The scheme of this architecture is reported in Figure 2.7.

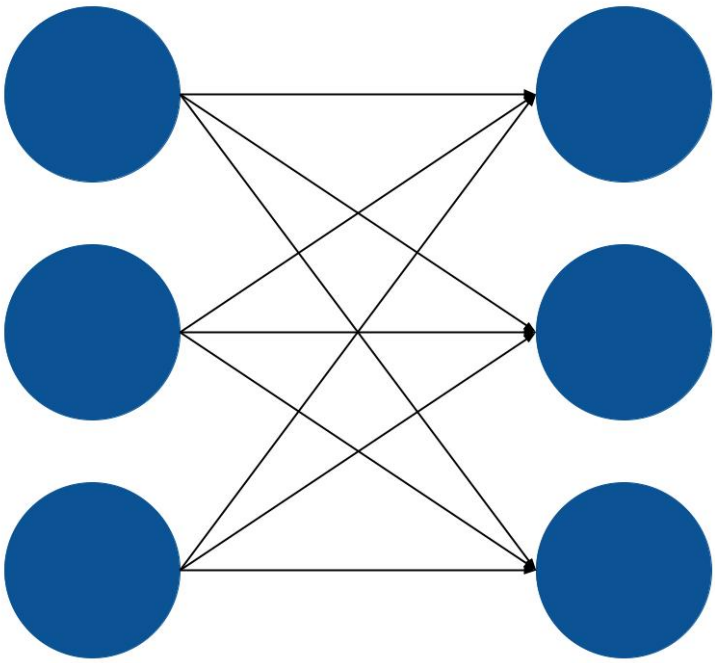


Figure 2.7: Dense layer: all neurons of the layer are connected to each node of the next one.

Convolutional layers, instead, are mostly employed in problems that need to process complex data such as images, in which spatial information is of key importance as it carries semantic meaning. The operation involves a convolution between a part of the image, called the **receptive field**, and a filter (or kernel) with a chosen dimension [9]. The convolution operation consists of multiplying each pixel of the receptive field with the kernel value at the corresponding position of the filter. After that, all partial products are summed up together to get the final result, which will amount to the pixel values of the new convoluted image. The principle behind this mathematical operation is shown in Figure 2.8 below.

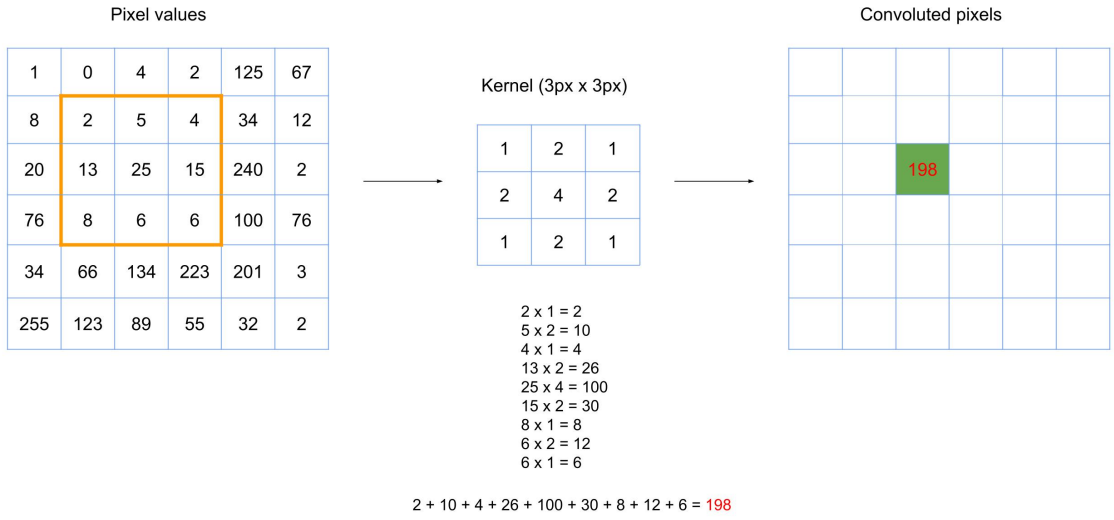


Figure 2.8: Convolution operation between the original pixel values and a 3x3 kernel. The partial products are summed up together to get the output.

The convolution operation becomes more challenging when dealing with color images. Indeed, differently from grey-scale ones, colors are obtained combining three distinct channels: red, green, and blue, leading to RGB images. In this case, it is requested to perform a convolution operation for each channel before evaluating the final result.

Finally, pooling layers are inserted into the network to reduce the dimensions of the feature maps (outputs of the discussed operation applied to the previous layers) [9], allowing a reduction of the parameters to be learned and, consequently, decreasing the requested computational cost. It is a way to undersample data.
With the purpose explained above, these layers apply a certain pooling operation to a specific area of the image. The most common operations are the max-pooling and the average-pooling. In the former, the entire area of the image under analysis is replaced by a single pixel having a value equal to the largest of the considered ones; the latter consists of averaging values of the original zone to get a mean value. As said, pooling is an undersampling operation since both cases reduce an entire area of the image to a single pixel with a computed value. An example of the max-pooling operation is shown in Figure 2.9.

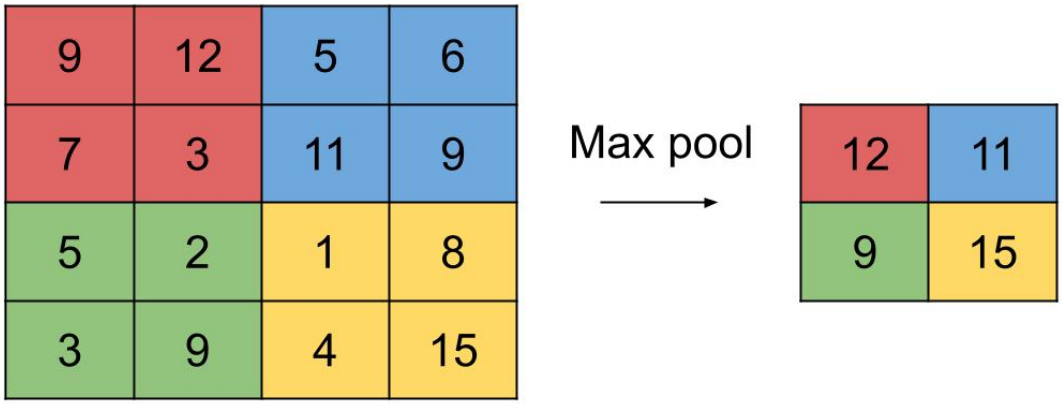


Figure 2.9: Max pooling operation: data are undersampled so that dimensions will reduce. In the scheme, the input has sizes 4 x 4 and the output 2 x 2.

2.3 Training of the network

To correctly set and tune the weights and the bias of each node of the neural network, a training phase has to be performed so that the network can learn all its parameters. The training relies on a dataset, a collection of inputs associated with the correct output label. In other words, during the training, the network learns how to map input-output pairs according to the examples provided in the dataset.

Generally, the learning procedure consists of iteratively feeding the network with inputs taken from a chosen dataset and comparing the produced output to the correct label embedded in the dataset entry. After the comparison, the error between them is computed according to a chosen loss function. Neural networks aim at minimizing the loss function to improve classification accuracy. The network can then fine-tune its parameters to improve performances during the next iteration, thanks to the backpropagation method that is explained in the following.

There are different ways of performing the training of the network: unsupervised learning and supervised learning. In the former, the network is able to self-assign bonus or malus points according to the produced output. This encourages correct learning while errors are strongly discouraged. The latter, instead, is the most common approach: it consists of evaluating the error between the produced output and the true output label according to a chosen loss function, which has to be differentiable. The most common loss functions will be analyzed in the following.
After that, the sensitivities of the computed error versus every parameter are evaluated and used to fine-tune all the variables of the model. This thesis work will rely only on supervised learning, as the employed datasets embed a lot of input-output pairs.

2.3.1 Backpropagation and Stochastic Gradient Descent (SGD)

Section 2.1 explains how neurons, the elementary part of each layer of neural networks work. Their functionality is based on several parameters described before, such as weights and biases.

However, it has not been said yet how these parameters can be tuned to improve the performances of the network. For the sake of simplicity, let's consider a two-individual layer neural network, represented in Figure 2.10.

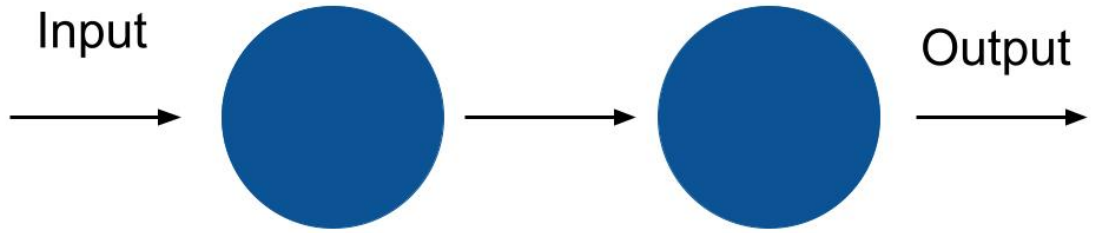


Figure 2.10: Two layer neural network considered in the following example.

The output of each node can be computed by means of the formula:

$$o_j = f(x_j \cdot w_j + b_j) \tag{2.3}$$

where f is a generic activation function and pedix j refers to the j^{th} layer of the network. Neurons parameters w_j and b_j are commonly initialized with random values.
The output produced by the entire network, $\hat{y}$, is computed as:

$$\hat{y} = f(f(x_1 \cdot w_1 + b_1) \cdot w_2 + b_2) \tag{2.4}$$

It is now possible to determine the error, or the cost, between the true output and the predicted one, according to a chosen loss function L. In expression 2.5, every parameter is known except for the weights and biases of each neuron, which are the independent variables.

$$L(y,\hat{y}) = L(y,f(f(x_1 \cdot w_1 + b_1) \cdot w_2 + b_2)) \tag{2.5}$$

The error can be backpropagated [10] to every variable by evaluating the gradient of the network, i.e. the derivative of the loss function with respect to each independent parameter.

$$\frac{\partial L}{\partial w_j} \qquad \frac{\partial L}{\partial b_j} \tag{2.6}$$

Before providing the next entry of the dataset, parameters may be updated and tuned in an iterative way considering the committed error.

$$w_{\text{j new}} = w_j - \frac{\partial L}{\partial w_j} \qquad b_{\text{j new}} = b_j - \frac{\partial L}{\partial b_j} \tag{2.7}$$

This iterative algorithm is known as **Stochastic Gradient Descent** (SGD)[11]. In particular, it aims at minimizing the loss function, leading to a maximization of the classification accuracy over previously unseen data.

Referring to Figure 2.11 below, representing the loss as a function of the weight w_1 in a generic way, it is possible to consider P1 as a starting point, which is obtained by randomly initializing weights and biases of the nodes.
After the first iteration and the first update, the working point of the network becomes P2, which is characterized by the updated value of the network parameter, leading to a reduction of the loss.

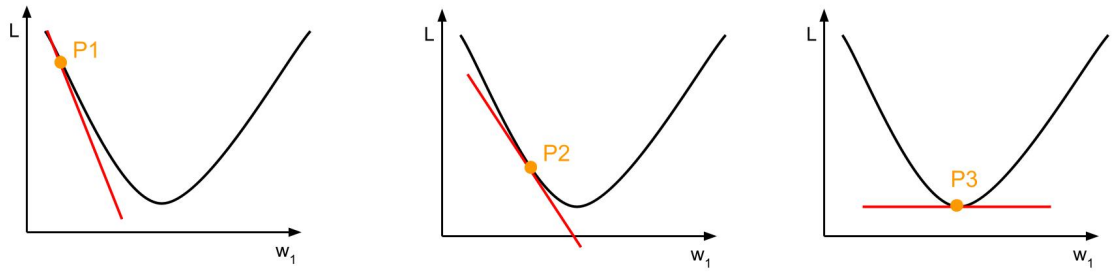


Figure 2.11: SGD algorithm: ***left****: P1 with randomly initialized parameters;* ***center****: P2 reached after an iteration of SGD algorithm with consequent loss reduction;* ***right****: P3, the minimum of the loss function is reached.*

The process is repeated iteratively until the condition in the expression 2.8 is reached, meaning that the derivatives of the loss function with respect to the weights and biases are equal to 0, or the gradient of the network is null. For this reason, updates are not needed anymore. In other words, it means that a minimum of the loss function has been found, giving high accuracy. Generally, the best achievable accuracy is obtained only whenever a global minimum of the loss function is reached.

Indeed, if the reached critical point was a local minimum, performances could be further improved.

$$\frac{\partial L}{\partial w_j} = 0 \qquad \frac{\partial L}{\partial b_j} = 0 \tag{2.8}$$

Actually, updating parameters by computing all the derivatives in the expression 2.6 would have a very high computational cost, thus much time-consuming.
In reality, dataset entries are grouped in sets, called **batches**, so that the loss is computed and averaged over multiple input-output label pairs. This allows reducing the number of needed backpropagation iterations.

For instance, considering a dataset having 50000 entries, the improvement introduced thanks to the organization in batches is shown in the following, when a batch size of 32 is considered.

$$No\ batches\ (batch\ size = 1) : \frac{50000\ entries}{batch\ size} = \frac{50000\ entries}{1} = 50000\ iterations$$

$$Batches\ of\ 32\ elements : \frac{50000\ entries}{batch\ size} = \frac{50000\ entries}{32} = 1563\ iterations$$

Moreover, the duration of the training is determined by the concept of epochs, which has to be defined. A training **epoch** is defined as the number of passes of the entire training dataset the machine learning algorithm has completed.

In addition to that, a condition named **overfitting** has to be avoided. Overfitting refers to a model that models the training data too well while lower classification accuracies are obtained on the test data. Overfitting happens when a model learns the details and noise in the training data: this impacts the performance of the model on previously unseen data. Two common approaches to reduce the risk of overfitting are:

- **Data augmentation**: operation in which train input images are artificially modified by means of graphical operations, such as translations, rotations, or zooms. This allows extending the dataset, increasing the number of available examples.

- **Dropout**: an option that consists of disabling several artificial neurons in different layers of the network for certain input features. In this way, network nodes are randomly turned on and off for different iterations, avoiding a stall condition in the learning phase of the parameters. The dropped-out neurons do not contribute to forward pass and do not participate in backpropagation [12]. Dropout improves the generalization skills of the model because, in this way, neurons do not depend on each other anymore.

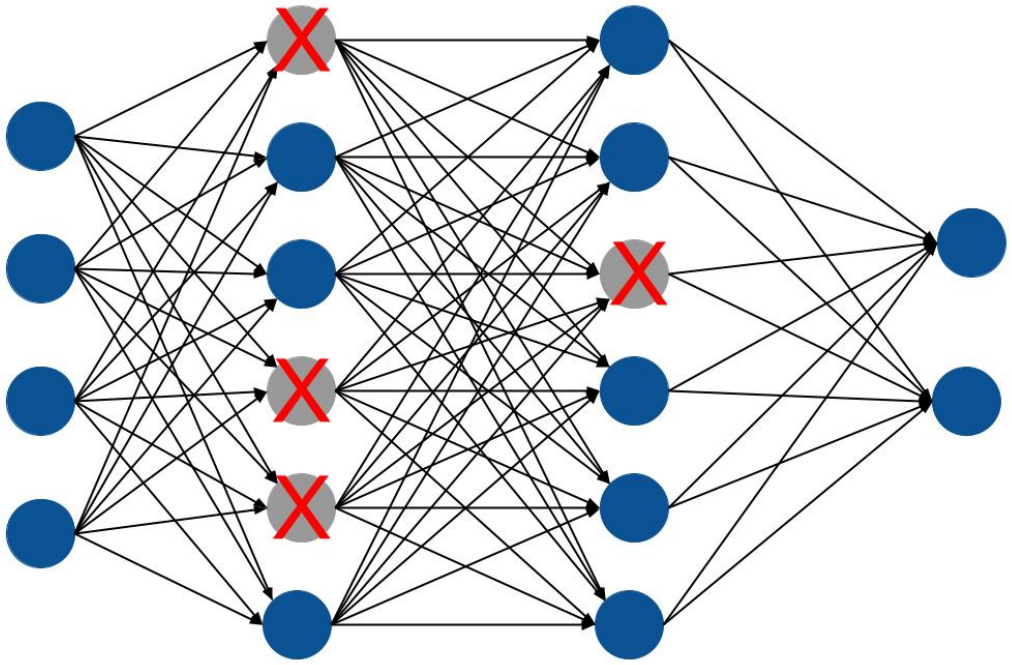


Figure 2.12: Dropout method consists of disabling randomly chosen neurons to avoid overfitting: in the image neurons disabled are represented in grey color with a red cross.

In the following part of the section, loss functions used for different purposes will be analyzed: in particular, mean squared error is employed whenever the problem involves regression, while cross-entropy is mostly used for classification.

Finally, a focus on the employed datasets will be proposed. The chosen datasets are often used to train a variety of different models as they embed a huge amount of input-output pairs.

2.3.2 Mean squared error loss function

The most common loss function for regression is the mean squared error. It represents the mean overseen squared differences between true and predicted values and it's evaluated using the formula:

$$L(y,\hat{y}) = \frac{1}{N}\sum_{i=0}^{N}(y - \hat{y}_{\mathrm{i}})^2 \tag{2.9}$$

where y is the true output, $\hat{y}$ is the output produced by the network, and N is the input data batch size, the number of input features considered for an iteration. Notice that if no organization in batches is employed, then $N = 1$ so the backpropagation takes place after each sample. Mean squared error is very sensitive towards outliers: indeed, given several examples with the same input feature values, the optimal prediction will be their mean target value. Thus, it works well when input data are normally distributed around a mean value and when it is important to penalize outliers.

2.3.3 Cross-entropy loss function

Cross-entropy is the most used loss function for classification problems [13]. It is a measure of the difference between two probability distributions, such as the produced output and the correct label, for a given random variable or set of events.

To better understand how it works, it is useful to recall several information theory concepts, in particular how information quantifies the number of bits required to encode and transmit an event. Lower probability events have more information while higher probability events contain less information. Information $h(x)$ can be calculated for an event x, given the probability of occurrence of the event $P(x)$ as follows:

$$h(x) = -log(P(x)) \tag{2.10}$$

Entropy is the number of bits required to transmit a randomly selected event from a probability distribution. A skewed distribution has low entropy, whereas a distribution where events have equal probability has a larger entropy. A skewed probability

distribution is characterized by a low entropy because likely events commonly happen. On the other hand, balanced distribution turn to have higher entropy because events are equally likely. Entropy $H(x)$ can be calculated for a discrete random variable x with a set of states and their probability $P(x)$ as follows:

$$H(X) = -\sum_{x \in X} P(x) \cdot log(P(x)) \tag{2.11}$$

The cross-entropy loss function builds upon the idea of entropy from information theory and calculates the number of bits required to represent or transmit an average event from one distribution compared to another distribution.
Consider a target probability distribution P and an approximation of the target distribution Q, then the cross-entropy of between Q and P is the number of additional bits to represent an event using Q instead of P. The cross-entropy between two probability distributions, such as Q and P, can be stated formally as $H(P,Q)$, where $H()$ is the cross-entropy function, P may be the target distribution, i.e. the true output, and Q represents the approximation of the target distribution, i.e. the predicted output.
Cross-entropy can be calculated using the probabilities of the events from P and Q, as follows:

$$H(P,Q) = -\sum_{x \in X} P(x) \cdot \log_2(Q(x)) \tag{2.12}$$

Where $P(x)$ and $Q(x)$ are the probability distributions of the event x and $\log_2$ is the base-2 logarithm, meaning that the results are in bits.

Having computed the entropy for discrete probability distributions, we can now move to continuous ones. In such a case, cross-entropy is computed using the integral across the events instead of the sum. Therefore, equations 2.11 and 2.12 change as follows:

$$H(X) = -\int_X P(x) \cdot log(P(x))\, dx \tag{2.13}$$

$$H(P,Q) = -\int_X P(x) \cdot log(Q(x))\, dx \tag{2.14}$$

In the case of machine learning, the cross-entropy loss function is expressed as:

$$L(y,\hat{y}) = -\sum_{i=0}^{N} y \cdot log(\hat{y}_{\mathrm{i}}) \tag{2.15}$$

where y is the true output, $\hat{y}$ is the evaluated output probability and N is the output size or the number of classes.

2.4 Introduction to common datasets

In this thesis work, I have considered three image datasets: MNIST, SVHN, and CIFAR10. Each dataset is composed of two parts: train data and test data. The former is the actual group of input-output pairs used to train the model, so the model sees and learns from this data. The latter, instead, is the sample of data used to provide an unbiased evaluation of the final model fit on the training dataset [14].

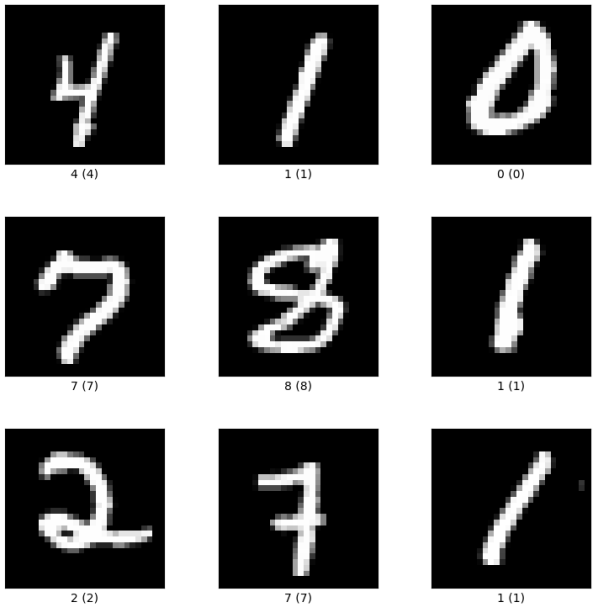


Figure 2.13: Examples of MNIST input features. [Taken from Tensorflow Resources].

The first considered dataset, the **Modified National Institute of Standards and Technology**(MNIST) database, is a collection of 70000 grey-scale images representing handwritten numbers, as shown in Figure 2.13. In particular, they are divided into 60000 training images and 10000 test ones. Each image has sizes 28x28 pixels, leading to a number of input pixels that equals 784. The dataset has ten possible output classes as the represented handwritten numbers go from 0 to 9.

The **Street View House Numbers** (SVHN) dataset is a more challenging dataset than MNIST. Just as before, images represent numbers. However, this time images are colored, so information is contained on three different channels (Red, Green, and Blue), and collected from the Street View feature embedded in Google Maps application: for this reason, a variable amount of digits can be included in the input feature. Some examples of this dataset entries are shown below in Figure 2.14. This dataset has ten possible output classes, one for each digit. It embeds 73257 entries for training and 26032 entries for testing, each one having sizes 32x32x3.

Figure 2.14: Examples of SVHN input features. [3]

The last dataset is also the most challenging one. The **Canadian Institute For Advanced Research 10** (CIFAR10) dataset, is a collection of 60000 colored images, divided into 50000 train images and 10000 test ones. Each feature has dimensions 32x32x3 since they are squared-colored images. The images represent a subject of the ten possible output classes, which are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. Examples of CIFAR10 input features are shown in Figure 2.15.

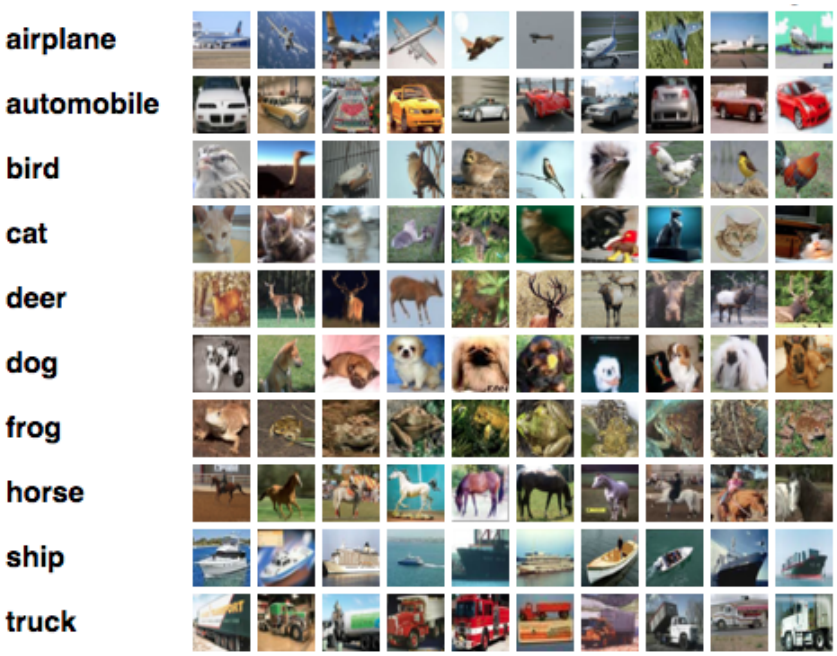


Figure 2.15: Examples of CIFAR10 input features. [4]

Chapter 3

Gaussian class conditional simplex method

The **Gaussian Class Conditional Simplex** (GCCS) [15] method is designed to be employed in a generic classifier with an arbitrary number of classes.
In particular, the network employs the GCCS loss function preceded by a specific encoder, based on an arbitrary amount of residual layers [16]. This last one is a feature extractor that maps input data onto a latent space that has as many dimensions as the number of possible output classes, i.e. the output size. To each dimension corresponds a class conditional statistical distribution.

3.1 Feature extractor

To correctly map data, the feature extractor requires two important parameters: $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. The former defines the mean value of the statistical distribution, while the latter represents the variance. As visible in Figure 3.1, Gaussian distributions are characterized by these parameters. The higher the mean value, μ, the farther the distribution will be from the origin of the axes; the higher the variance, Σ, the lower and larger the distribution will be, meaning that data will not be very close to the mean value and outliers may be present, as represented by the red line. On the other hand, considering a lower Σ means that data are close to each other and to the mean value, as shown with the blue curve.
It is more common to use σ, a parameter called standard deviation, instead of Σ. There is a quadratic relation between them, so $\Sigma = \sigma^2$.

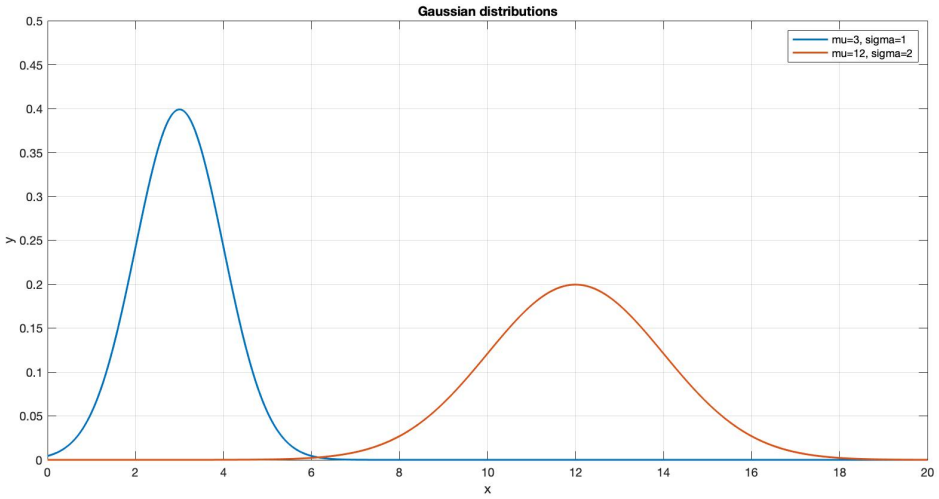


Figure 3.1: Gaussian distributions with different mean and variance values. The blue curve has $\mu = 3$ and $\Sigma = \sigma^2 = 1$; the red line has $\mu = 12$ and $\Sigma = \sigma^2 = 4$.

Since the feature extractor maps data into a D-dimensional latent space, where D is, as said, the number of available classes, the generated statistical distributions will be centered on the vertices of a simplex having $D-1$ dimensions. Assuming the simplex to be regular, all classes are expected to be equidistant between each other [15].

By properly setting the previously described parameters, this method allows reaching a high inter-class separation and a low intra-class dispersion. For this reason, data belonging to the same class will be grouped as the classifier recognizes them as very similar, while data of different classes are well separated, so improving the robustness of the network. A requirement is to set parameters such that:

$$\frac{\mu_T}{\sigma_T} > \sqrt{2D} \tag{3.1}$$

For the sake of clarity, a visual representation of the simplex-based class separation is reported in Figure 3.2.

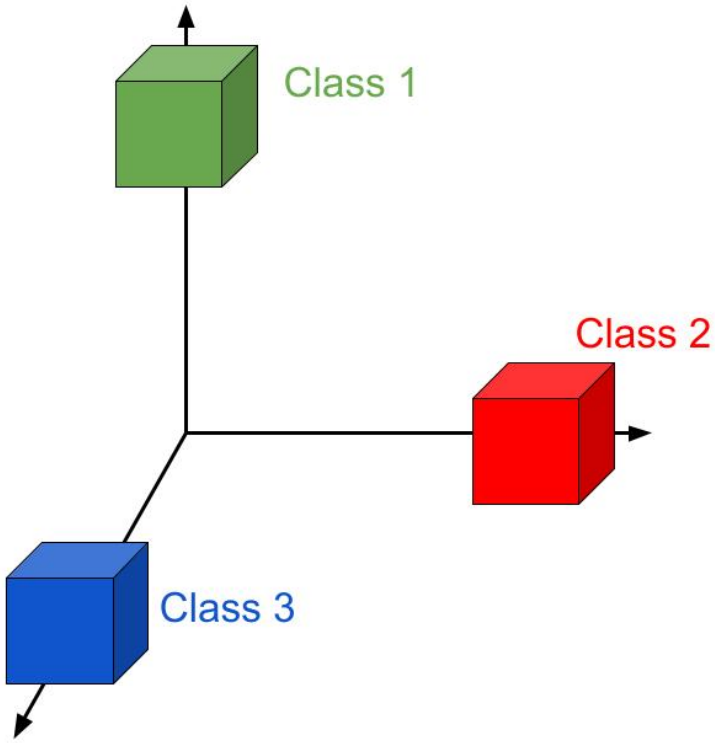


Figure 3.2: Simplex based class distribution adopted in GCCS method: data belonging to different classes are centered on the vertices of a simplex, obtaining high inter-class separation and low intra-class dispersion.

The GCCS framework has also another important benefit. Since distributions are gaussian, the optimal decision boundaries are simple hyperplanes. This property is in contrast to the typical behavior of neural networks, which learn very complex decision boundaries. This design also leads to better accuracy and greater robustness than other state-of-the-art methods.

3.2 Loss function

In order to tune the network parameters and minimize the distance between predicted output distributions and target ones, a particular loss function is employed. The method estimates, for each class, first and second order statistics of the predicted output distributions, μ_O and Σ_O. To evaluate how distant they are from target ones, the Kullback-Leibler divergence is used. This last one can be calculated with respect to gaussian target distributions as reported in the expression 3.2, where pedix i refers to the i$^{\text{th}}$ class, O to the predicted output class, and T to the target class.

$$\boldsymbol{L}_i = log\frac{|\boldsymbol{\Sigma}_T|}{|\boldsymbol{\Sigma}_{\text{Oi}}|} - D + tr(\boldsymbol{\Sigma}_T{}^{-1}\boldsymbol{\Sigma}_{\text{Oi}}) + (\boldsymbol{\mu}_{\text{Ti}} - \boldsymbol{\mu}_{\text{Oi}})^T\boldsymbol{\Sigma}_T{}^{-1}(\boldsymbol{\mu}_{\text{Ti}} - \boldsymbol{\mu}_{\text{Oi}}) \tag{3.2}$$

So, extending the concept, the cumulative total loss that considers all classes is computed as follows.

$$\boldsymbol{L} = \sum_{i=1}^{D} \boldsymbol{L}_i \tag{3.3}$$

In particular, L reaches its minimum value when the predicted statistics of the D distributions exactly match the target ones. Since, as previously said, data are divided into batches, if the batch size is too small it becomes difficult to control the tails of the gaussian distributions. To solve this issue, Kurtosis, defined in the expression 3.4, is also employed. In particular, the notation represents the j$^{\text{th}}$ component of the i$^{\text{th}}$ target distribution.

$$\boldsymbol{K}_{\text{i,j}} = \left(\frac{z_{\text{i,j}} - \boldsymbol{\mu}_{\text{O i,j}}}{\sigma_{\text{O i,j}}}\right)^4 \tag{3.4}$$

So the total loss function becomes:

$$\boldsymbol{L}_{\text{GCCS}} = \sum_{i=1}^{D}[\boldsymbol{L}_i + \lambda(\boldsymbol{K}_i - 3)] \tag{3.5}$$

where λ is a parameter that balances the effect of Kurtosis compared to the Kullback-Leibler divergence and 3 is the found target Kurtosis for each class.

3.3 Decision rule

If the GCCS method converges properly, it is possible to define the optimal decision boundaries in the learned latent space.
In particular, for the chosen target distributions, optimal boundaries are defined by partitioning the D-dimensional latent space, in which the simplex is located, into Voronoi regions.
Considering a set of points S, a Voronoi space splitting is a mathematical concept that associates a region $V(p) \quad \forall p \in S$ such that all points belonging to $V(p)$ are closer to p than any other point in S. An example of Voronoi regions is reported in Figure 3.3 [17].
In this way, all the points are closer to their region centroid, i.e. the mean of their distribution, than any other in the $D-1$ simplex.
For this reason, the resulting decision rule requires the computation of the distance between output feature points and all centers of the regions, and consequently classify the sample as belonging to the class at the minimum distance.

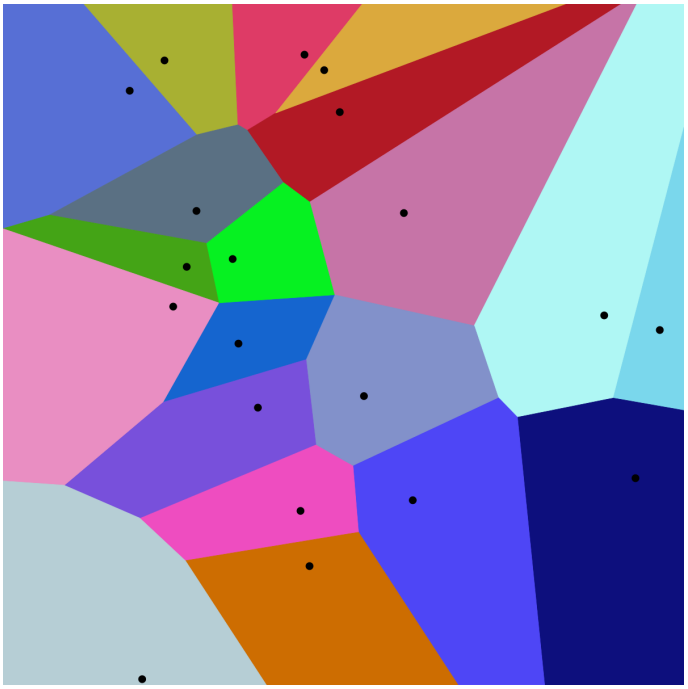


Figure 3.3: Example of Voronoi space division with all regions highlighted. [Taken from Wikipedia].

Chapter 4

Evaluating adversarial robustness

The ever-growing demand for machine learning algorithms across various applications in our daily lives pones some security warnings. In particular, many barriers affect the use of deep neural networks in applications where security is of key importance, such as medical diagnostics and autonomous driving. One of the most severe flaws of deep learning is represented by adversarial examples, a collection of methods that are designed to interfere with neural networks input data to produce undesired outputs or cause algorithm malfunctions and classification accuracy reduction. This happens in the face of modifications that are very difficult to detect. Indeed, these adversarial examples are often undetectable by human eyes. Whenever the model is fed with them, so that network is under attack, the reached security level can be measured through proper metrics, like the classification accuracy. More specifically, classification accuracy is evaluated as a function of a tunable parameter ϵ that indicates how strong is the applied attack, such that $\frac{||\boldsymbol{n}||_\infty}{||\boldsymbol{x}||_\infty} \leq \epsilon$, where n is the added noise vector, and x is the input signal.

Attacks are divided into two main categories: **untargeted attacks** and **targeted attacks**. The goal of the former ones is to cause a general misclassification in labeling the input, such that the predicted class does not correspond to the true output label. The latter ones, instead, are generated to be misclassified to the desired output class, which is the target of the attack.

Attacks are also classified considering the adversary's knowledge of the model. In particular, all the implemented attacks belong to the white-box category, thus the attacker is supposed to have access to the network gradients. That means that the attacker has a copy of the model's weights. This threat model gives the attackers much more power than black-box attacks as they can specifically craft their examples to fool the model without having to rely on attacks that often result in human-visible perturbations.

In the following part of the chapter, I introduce the implemented and considered attacks. After that, more details on how a correct defense evaluation should be carried out will be provided.

4.1 Non targeted attacks

4.1.1 Fast Gradient Sign Method (FGSM)

The first implemented attack is called the **Fast Gradient Sign Method** (FGSM)[6]. It is a single-step attack that consists of summing up a non-random perturbation to the original input data. Usually, the generated adversarial examples are quite similar to the original features, such that human eyes cannot detect any variation but an error is induced when classifying with machine learning algorithms. In particular, the added perturbation is smaller than the precision of the input data.
According to the FGSM technique, adversarial examples are crafted as [1] :

$$x_{\text{adv}} = x + \epsilon \cdot sign(\nabla_x L(\theta,x,y)) \tag{4.1}$$

where ϵ is the noise power (i.e. the magnitude of the added perturbation), L is the chosen loss function, θ represents the model parameters, x and y are the input features and output label respectively.

4.1.2 Projected Gradient Descent (PGD)

The **Projected Gradient Descent** (PGD) is an iterative version of the previously mentioned FGSM attack [15]. With this method, noise is added over multiple iterations, resulting in the strongest adversarial attack that exploits first-order information about the trained model.
In particular, PGD tries to find the perturbation that maximizes the loss of a model on a specific input while keeping the size of the perturbation smaller than a specified amount ϵ.

The PGD attack can be summarized with the instructions below, even though the attacker is free to apply further optimization improvements [18].

1. Start from a random perturbation
2. Take a gradient step in the direction of the greatest loss
3. Project computed perturbation on the input feature
4. Repeat points 2 and 3 until the desired number of iterations is reached

The algorithm can be described with the following pseudo-code.

Algorithm 1 Projected Gradient Descent

$x_{adv} \leftarrow x$
$i \leftarrow 0$
while $i < K$ **do**
    $noise \leftarrow sign(\nabla_x L(\theta,x,y))$
    $\epsilon_{\%} \leftarrow \frac{\epsilon}{100}$
    $x_{adv} \leftarrow x_{adv} + \epsilon_{\%} \cdot noise$

Where X_{adv} is the generated adversarial example, ϵ is the noise power and K is the chosen amount of iterations, which plays an important role in defining the strength of the attack and the time requested to generate the corresponding adversarial examples. The greater K the stronger the attack with the subsequent longer time needed.

4.1.3 DeepFool

DeepFool is an iterative untargeted attack that can optimally show the flaws of a neural network. In particular, it aims at minimizing the euclidean distance between perturbed samples and original ones. Decision boundaries between classes are estimated and, according to them, perturbations are added iteratively.
In the following, the algorithm to generate adversarial examples according to this method will be presented for both binary and multiclass classifiers [19].

In Figure 4.1 below, it can be noticed that the robustness of the binary classifier f for an input x_0 is simply the distance between x_0 and the hyperplane, which is used as a decision rule to label outputs. Minimal perturbation to fool the classifier's decision, $\hat{r}$, corresponds to the orthogonal projection of x_0 onto the hyperparameter plane, given by:

$$-\frac{f(x_0)}{||\nabla f(x_0)||_2^2} \cdot \nabla f(x_0) \tag{4.2}$$

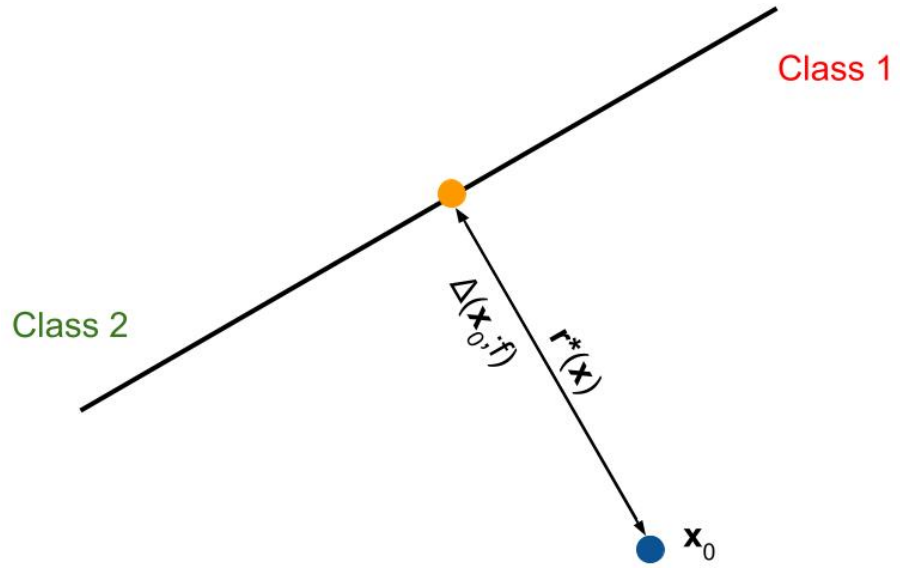


Figure 4.1: Hyperplane for binary classifier, distance between x_0 and hyperplane and purturbation $\hat{r}$ are highlighted.

The DeepFool algorithm for binary classifiers is reported below.

Algorithm 2 DeepFool for binary classifiers

$\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$

$i \leftarrow 0$

while $sign(f(\boldsymbol{x}_i)) = sign(f(\boldsymbol{x}_0))$ **do**

  $\boldsymbol{r}_i \leftarrow -\frac{f(\boldsymbol{x}_i)}{||\nabla f(x_i)||_2^2} \cdot \nabla f(\boldsymbol{x}_i)$

  $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$

  $i \leftarrow i + 1$

return $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$

In particular, the procedure takes as inputs the classifier f and the input x_0 and returns as output the minimal perturbation $\hat{r}$ to fool the model into misclassifying the input. The method consists of iterating the following steps until the misclassification is obtained:

1. Calculate the projection of the input onto the closest hyperplane
2. Add that perturbation to the input and re-test

Considering a multiclass classifier and assuming the input to be x, a hyperplane is used as a decision rule for each class. Looking at the place in the space where x lies, it is classified into a certain output class. The algorithm finds the closest hyperplane, projects x onto that hyperplane, and pushes it a bit beyond, thus misclassifying it with the minimal perturbation possible. A visual representation of the space containing all hyperplanes is shown in Figure 4.2, while in the following the generalized algorithm is reported [19].

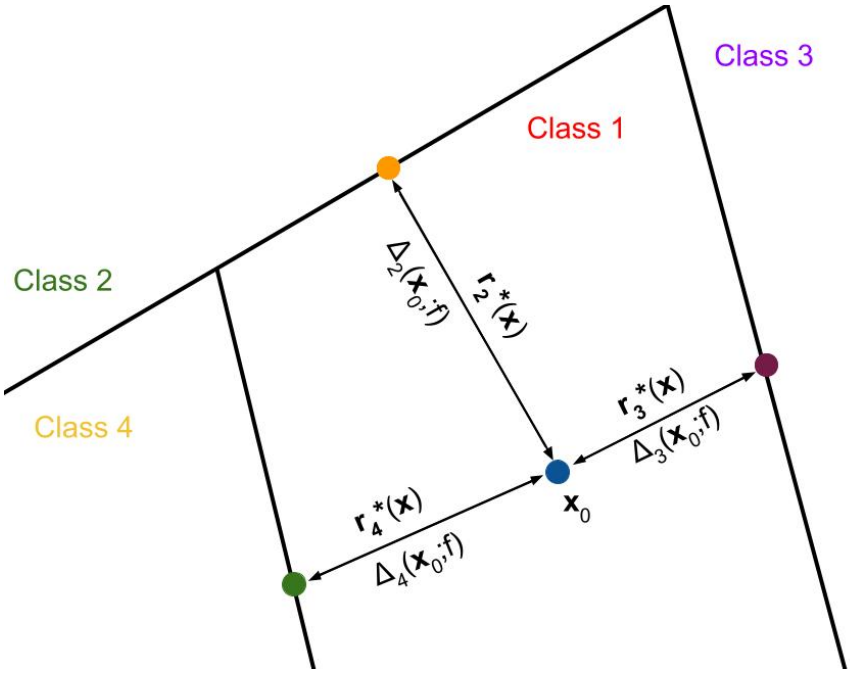


Figure 4.2: Hyperplanes for a multiclass classifier, distances between x_0 and all hyperplanes and perturbations $\hat{r}_i$ are highlighted.

Algorithm 3 DeepFool for multiclass classifiers

$\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$
$i \leftarrow 0$
while $\hat{k}(\boldsymbol{x}_i) = \hat{k}(\boldsymbol{x}_0)$ **do**
  for $k \neq \hat{k}(\boldsymbol{x}_0)$ **do**
    $\boldsymbol{w}_k' \leftarrow \nabla f_k(\boldsymbol{x}_i) - \nabla f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
    $f_k' \leftarrow f_k(\boldsymbol{x}_i) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
  $\hat{l} \leftarrow \arg\min_{k \neq \hat{k}(\boldsymbol{x}_0)} \frac{|f_k'|}{||\boldsymbol{w}_k'||_2}$
  $\boldsymbol{r}_i \leftarrow -\frac{|f_{\hat{l}}'|}{||\boldsymbol{w}_{\hat{l}}'||_2^2} \cdot \boldsymbol{w}_{\hat{l}}'$
  $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$
  $i \leftarrow i + 1$
return $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$

Just as in the binary case, the method takes as inputs the multiclass classifier f and the input x and gives the minimal required perturbation $\hat{r}$ as output. The following sequence of instructions is repeated until a misclassification is reached, meaning that the original label and the perturbed label are not equal. In particular, for each iteration, the minimum difference between the original gradients and the gradients of each of the n classes, w_k, and the difference in the labels, f_k, are stored.

1. The inner loop stores the minimum w_k and f_k. Using them it is possible to calculate the closest hyperplane for the input x by means of the expression 4.3

2. The minimal vector that projects x onto the closest hyperplane is evaluated thanks to the formula 4.2

3. The minimal perturbation is added to the image and the misclassification is tested

As mentioned, the closest hyperplane is computed using the expression 4.3 reported below.

$$\hat{l}(\boldsymbol{x}_0) = \underset{k \neq \hat{k}(\boldsymbol{x}_0)}{\arg\min} \frac{f_k(\boldsymbol{x}_0) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_0)}{||\boldsymbol{w}_k - \boldsymbol{w}_{\hat{k}(\boldsymbol{x}_0)}||_2} \tag{4.3}$$

Where variables starting with f are the class labels, variables starting with w are the gradients, variables with k as subscript represent the classes with the most probability after the true class, and variables with subscript $\hat{k}(\boldsymbol{x}_0)$ are for the true class.
Expression 4.3 leads to a minimal perturbation to be added to the input features computed as:

$$r_*(\boldsymbol{x}_0) = \frac{f_{\hat{l}(\boldsymbol{x}_0)}(\boldsymbol{x}_0) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_0)}{||\boldsymbol{w}_{\hat{l}(\boldsymbol{x}_0)} - \boldsymbol{w}_{\hat{k}(\boldsymbol{x}_0)}||_2^2} (\boldsymbol{w}_{\hat{l}(\boldsymbol{x}_0)} - \boldsymbol{w}_{\hat{k}(\boldsymbol{x}_0)}) \tag{4.4}$$

As explained, this algorithm iteratively tries to find the minimum perturbation to be applied to input such that misclassification is obtained. Therefore, the robustness of the tested models cannot be plotted like the reached accuracy as a function of noise power, since this approach returns the minimum ϵ instead.

As described [19], robustness is now evaluated through the ρ_{adv} parameter, defined as:

$$\rho_{adv} = \frac{1}{|batch|} \cdot \sum_{\boldsymbol{x} \in batch} \frac{||\hat{\boldsymbol{r}}(\boldsymbol{x})||_2}{||\boldsymbol{x}||_2} \tag{4.5}$$

where $|batch|$ represents the batch size, x is the input feature and $\hat{r}$ is the projection of the input feature onto the closest hyperplane, thus the minimum perturbation to be applied. In other words, ρ_{adv} averages on the whole batch the minimum perturbation to be applied, normalized to the input data. Notice that, as reported in the expression 4.5, this metric is based on norm 2. The higher the parameter ρ_{adv} the greater the robustness ensured by the considered model since the requested perturbation to get a misclassification has a larger norm. On the other hand, a small ρ_{adv} means that a small perturbation is sufficient to misclassify the input image onto the closest hyperplane used as a decision rule.

4.2 Targeted attacks

4.2.1 Targeted Gradient Sign Method (TGSM)

The **Targeted Gradient Sign Method** (TGSM) is a simple modification of the FGSM attack. Indeed, this approach is quite similar to the corresponding untargeted version, where a perturbation ϵ was summed up to the original input feature. In this case, instead, adversarial examples are obtained by subtracting the same amount of noise, ϵ, according to the following formula:

$$x_{adv} = x - \epsilon \cdot sign(\nabla_x L(\theta,x,y)) \tag{4.6}$$

Notice that noise is evaluated as already presented when describing untargeted attacks. The only difference with respect to the FGSM method is the addition that has become a subtraction.

The reason for such variation is because in the untargeted approach the purpose is to inject noise, thus projecting the gradient towards the direction characterized by a greater loss associated with the true label, resulting in a simpler prediction of a wrong class with the consequent misclassification. In the targeted attack case,

instead, the goal is to reduce noise related to the target class. It is done by projecting the gradient of the network towards a direction that minimizes the loss when labeling the target class.

Notice that the attack under analysis can be implemented either with a single step or with an iterative approach. The iterative approach results in a stronger attack with higher computational cost, thus more time-consuming. In particular, the greater the number of wanted iterations K the stronger the attack.
The K-steps iterative algorithm used to craft these samples is similar to the one presented for the PGD attack and it is reported below.

Algorithm 4 Targeted Gradient Sign Method

$x_{adv} \leftarrow x$
$i \leftarrow 0$
while $i < K$ **do**
    $noise \leftarrow sign(\nabla_x L(\theta,x,y))$
    $\epsilon_{\%} \leftarrow \frac{\epsilon}{100}$
    $x_{adv} \leftarrow x_{adv} - \epsilon_{\%} \cdot noise$

4.2.2 Jacobian Saliency Map Attack (JSMA)

Jacobian-based Saliency Map Attack (JSMA) is the last considered targeted attack. It consists of iteratively computing the jacobian matrix of the loss function with each class label with respect to every component of the input, i.e. the jacobian matrix to extract the sensitivity direction. This process aims at forming a saliency map, which is a map that shows the sensitivities of an image to get labeled into a certain target class to some pixels. In other words, the map shows the pixels that mostly influence the network to predict a certain output class [15] [20].
Indeed, this saliency map is used at every iteration to choose which pixels should be tampered, so that the likelihood of changing the output class towards the target one is increased.

It is important to notice that the major difference between JSMA and FGSM is that JSMA reduces the number of needed perturbations, making the adversarial examples far less detectable. The drawback of this attack is that it comes at an expense of a higher computational cost.

4.3 State of the art of the defense methods

It is fundamental to discover how to resist these adversarial examples because the more machine learning systems get built into our daily lives the more adversarial attacks could become a problem.
To prevent the classifier under analysis to misclassify input features when fed with adversarial examples, several methods have been proposed in scientific literature. A brief introduction to these methods is given in the following. However, only the adversarial training technique will be used for this thesis.

- Defense against a specific attack
- Gradient masking
- Adversarial training

The first approach has to be considered as a starting point as it is very basic. As suggested, indeed, it consists of making the classifier robust against a specific considered attack. However, if the adversary iterates the attack or tries different ones, several security concerns may appear.

4.3.1 Gradient masking

Gradient masking is a technique, sometimes used unintentionally, that consists of modifying the network weights to project the gradient of the network towards a decided direction, which is not the most vulnerable.
In this case, it is possible to have a false sense of security in defenses against adversarial examples [21], while the network is not robust as it seems. In particular, when following the gradient direction does not optimize the loss, iterative-based attacks cannot succeed.

Three different situations can be recognized:

- **Shattered gradient**: the situation in which the network is not differentiable. For this reason, a numeric instability condition may occur or the gradient might be incorrect
- **Stochastic gradient**: the gradient of the network is computed considering random factors associated either to the defensive network or to the input feature
- **Exploding gradient**: the network is extremely deep with a large number of hidden layers. For this reason, the gradient may explode when evaluated

Gradient masking can be easily detected if particular conditions are verified. It can be circumvented using specific algorithms to break the defense and let the attacks succeed.

4.3.2 Adversarial Training

The last method is the one that will be employed in this thesis work. **Adversarial Training** (AT) consists of crafting adversarial examples according to a selected attack, i.e. a sequence of carefully selected instructions, and providing them to the defensive neural network during the training phase.
Adversarial Training (AT) is a technique that extends the data augmentation method, with the goal of ensuring greater robustness of a model against various attacks. This is possible since the model learns how to behave when fed with adversarial examples that aim at causing a misclassification of the true label. In particular, the considered dataset is enlarged by crafting new entries, which are the generated adversarial examples. These new inputs are created according to a specifically chosen attack that can be either targeted or untargeted. Actually, targeted examples are unlikely to be employed during adversarial training because of their intrinsic characteristic. A model trained with targeted adversarial examples would be very robust against the tested target class but it could be easily fooled whenever a different target is chosen by the adversary. For this reason, adversarial training is always preferred to be implemented with untargeted attacks. In this way, the network learns how

to resist general misclassification with any of the possible classes, thus having each class robust like all other ones.

Adversarial training is different from classical data augmentation, which usually embeds operations such as rotations, translations, and zooms. Indeed, data augmentation allows the network learning transformations that are supposed and expected to happen in a real case scenario; during adversarial training, instead, data are augmented to form examples that are unlikely to occur in a classical scenario, but that can expose flaws of the network.

4.4 Defenses Evaluation Framework

While attack research has flourished during the last years, progress on defense research has been comparatively slow. Indeed, most proposed defenses quickly show to have been evaluated incorrectly or incompletely, thus giving a false sense of security. For this reason, estimating the robustness of a model against adversarial examples is a very complex task that should rely on a specific list of operations.
In the following, a set of principles for performing defense evaluations will be reported [22].

4.4.1 State a threat model

First of all, to perform a correct evaluation of the defense, it is necessary to consider a **threat model.** In fact, without this one, defense proposals are often either not falsifiable or trivially falsifiable. The threat model includes a set of assumptions such as the adversary's goals, knowledge, and capabilities.
The goal of the adversary is for sure to cause a misclassification but, as explained in the previous part of the chapter, it can be a general misclassification (i.e. untargeted attack) or a misclassification with the desired output (i.e. targeted attack).
It is also important to restrict the adversary's capabilities. Most defenses typically limit the adversary to making small changes to inputs, so that these examples cannot be detected by humans. Moreover, a common assumption is that the adversary has direct access to the model's input features, however, this is not always true, for

this reason, different hypotheses on the capabilities of the adversary may impact significantly the evaluation of defense effectiveness.
As said, the threat model is expected to express how an adversary knows the network to attack. As anticipated when discussing different attack algorithms, typical supposition involves either a complete knowledge of the model and its parameters, calling it a **white-box attack**, or no knowledge of the model at all, referring to it as a **black-box attack** [22].

4.4.2 Principles for rigorous evaluation

It is crucial to actively attempt to defeat the defense being proposed. In particular, this should be done by including a range of sufficiently different attacks with carefully tuned hyperparameters. In all these scenarios, one should assume the existence of an adversary who will spend whatever time is required to develop the optimal attack to get a misclassification. This means that it is important to focus on the strongest attack for the threat model and on the defense that are taken into account. Often, the strongest attack corresponds to an iterative attack with a considerable perturbation budget. Indeed, one should verify that, in general, iterative attacks perform better than single-step ones. In particular, for the former ones, it is important to ensure that increasing the number of iteration does not decrease the classification accuracy of the model.
It's worth testing both targeted and untargeted attacks. In theory, an untargeted attack is strictly easier than a targeted one. However, in practice, there can be cases where targeting any of the $N - 1$ classes will be stronger than performing one untargeted attack. Moreover, it is fundamental to verify that increasing the perturbation budget strictly impacts the classification accuracy. Attacks that allow more distortion are strictly stronger than attacks that allow less distortion, leading to more probable errors in the defense.
Finally, obtained results should be expressed using proper metrics. A possible metric is the classification accuracy of the model as a function of the noise power. For iterative attacks, another useful curve is the one representing classification accuracy versus the number of attack iterations.

Chapter 5

Experiments on GCCS networks

5.1 Adversarially trained models results

In this section, I will investigate how AT improves the robustness of a model. I will compare standardly trained models with adversarially trained ones.
Several experiments have been performed to test different models against the attacks mentioned in Sections 4.1 and 4.2. To obtain these results, the two datasets MNIST and CIFAR10 have been used. ResNet-18, a network that employs an encoder characterized by 18 residual layers has been chosen. The network has been trained with different loss functions and different trainings so that various models can be compared.

These experiments aim at showing the benefits of adversarial training versus standard training. Adversarial training has been adopted in two distinct versions: the former generates adversarially perturbated inputs by using the FGSM attack (AT-FGSM), thus noise is injected in a single step. The other one, instead, is a more sophisticated approach that adds noise during multiple steps, according to PGD attack (AT-PGD). In particular, the chosen number of steps equals 5. Training with the three different settings has been carried out for both cross-entropy, presented in Section 2.3.3, and GCCS, explained in Chapter 3 [15], loss functions. Two different results are expected: first of all, considering already obtained experimental results, GCCS outperforms cross-entropy; in addition to that, adversarial training, especially the PGD version, is expected to guarantee stronger robustness to the trained network.
Each model has been trained with a batch size of 200 entries for a total of 100 epochs on both MNIST and CIFAR10 datasets. As described, the GCCS approach also requires choosing the mean and variance of the statistical distributions: these

parameters have been set to $\mu = 70$ and $\sigma^2 = 1$. Moreover, the effect of Kurtosis is balanced by the factor λ that has been fixed to $\lambda = 0.2$.
A table that summarizes the considered models with the set hyperparameters is reported below.

Model	**Adversarial training/Standard training**	**Loss function**
1	AT: PGD	GCCS ($\mu = 70; \sigma^2 = 1$)
2	AT: FGSM	GCCS ($\mu = 70; \sigma^2 = 1$)
3	STD training	GCCS ($\mu = 70; \sigma^2 = 1$)
4	AT: PGD	cross-entropy
5	AT: FGSM	cross-entropy
6	STD training	cross-entropy

Table 5.1: Tested models to verify benefits of adversarial training versus standard training.

The results are reported in the following. Notice that, in every graph, a solid line with highlighted points means that the model has been trained with GCCS loss function. On the other hand, a dashed line represents a cross-entropy-based model. The tests have been performed with a maximum noise power ϵ that equals 0.1 for the MNIST dataset, while on CIFAR10 the maximum power budget has been reduced to 0.02. Due to the high computational cost and the consequent time requested, the JSMA attack on CIFAR10 has been analyzed considering a maximum noise power equal to $\epsilon = 0.006$ [15].

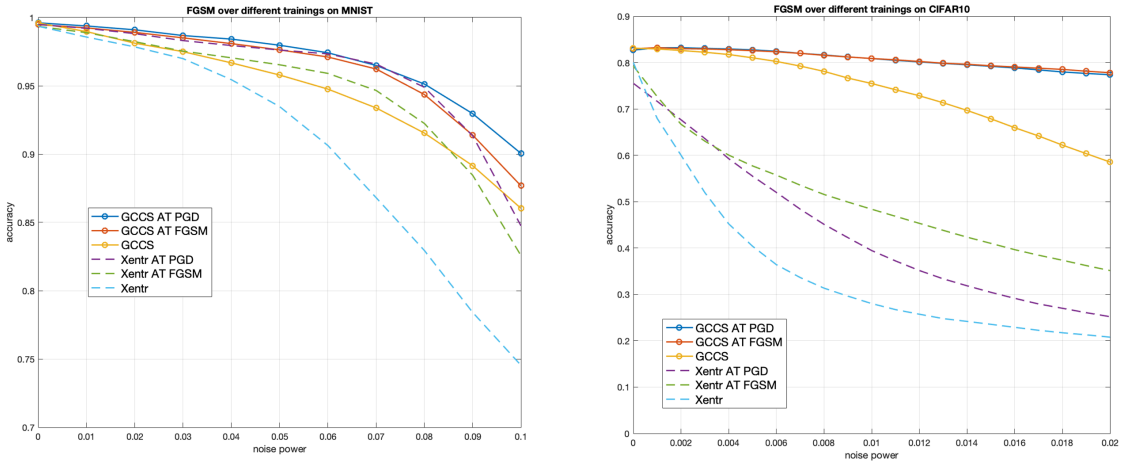


*Figure 5.1: Classification accuracy on the test set under the FGSM attack, as a function of the perturbation budget ϵ for MNIST (**left**) and CIFAR10 (**right**).*

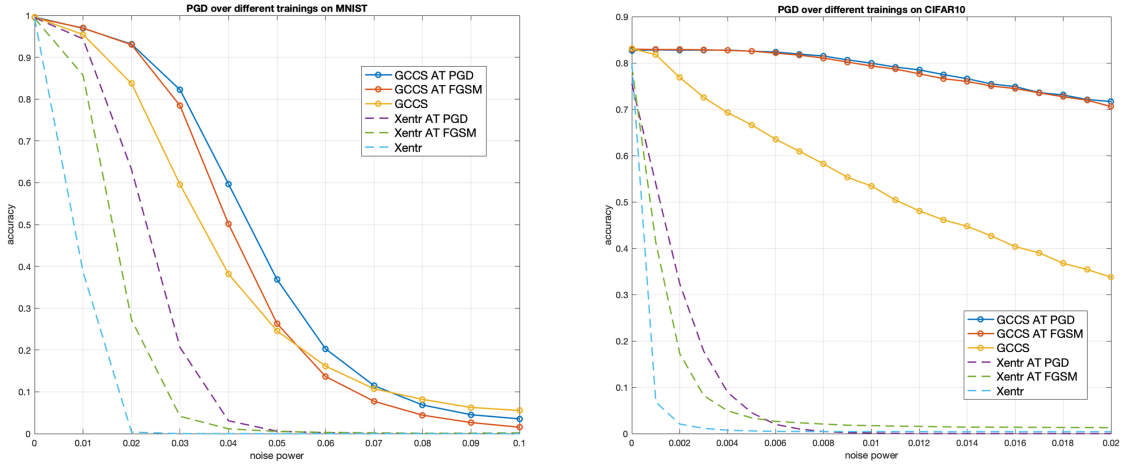


*Figure 5.2: Classification accuracy on the test set under the PGD 5-steps attack, as a function of the perturbation budget ϵ for MNIST (**left**) and CIFAR10 (**right**).*

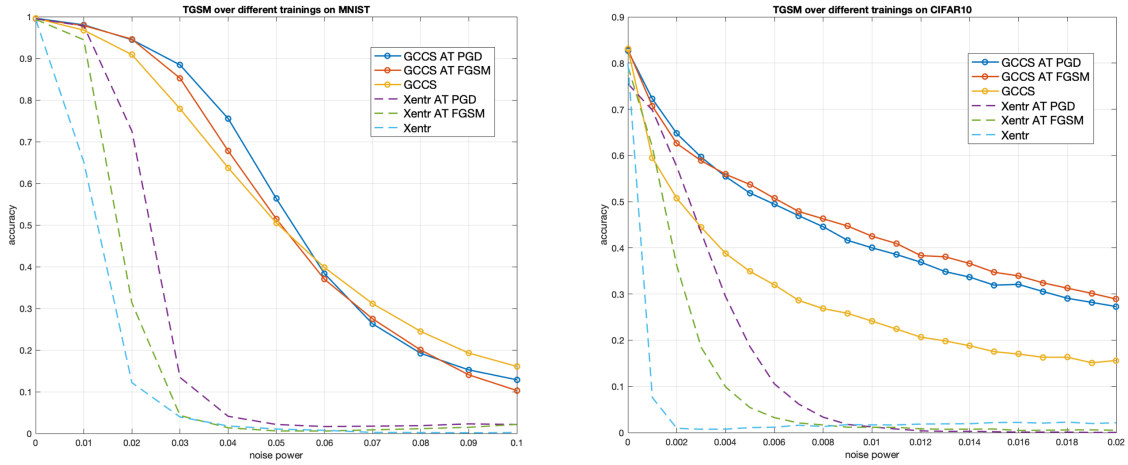


*Figure 5.3: Classification accuracy on the test set under the TGSM 5-steps attack, as a function of the perturbation budget ϵ for MNIST (**left**) and CIFAR10 (**right**).*

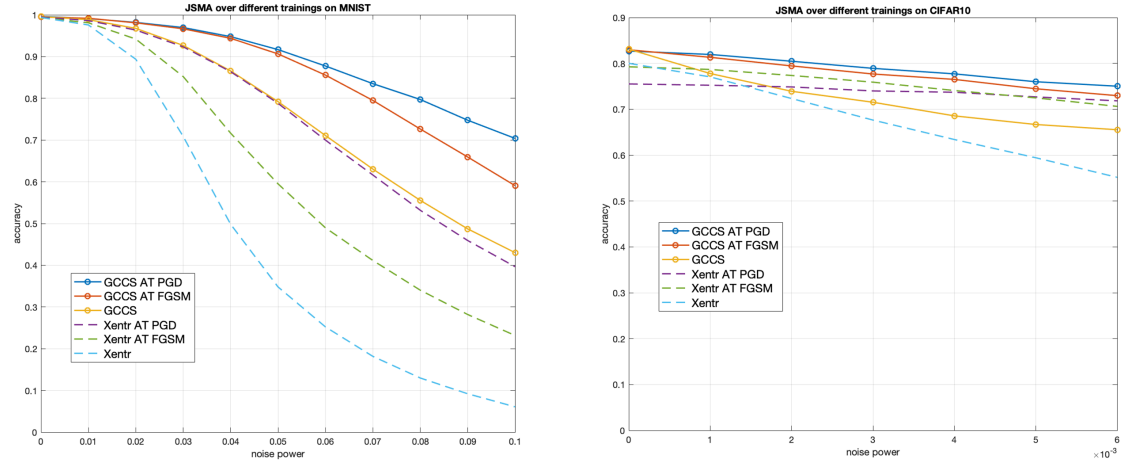


*Figure 5.4: Classification accuracy on the test set under the JSMA 200-steps 1-pixel attack, as a function of the perturbation budget ϵ for MNIST (**left**) and CIFAR10 (**right**).*

As shown in Figures 5.1 and 5.2 reporting results related to untargeted attacks, the GCCS loss function always leads to better performances with respect to cross-entropy. In these cases, also standard training combined with GCCS guarantees greater robustness than any model based on the cross-entropy loss function. As expected, if adversarial training is combined with the GCCS method the model is more robust than the other methods against the implemented attacks, if compared to the

other configurations. No great difference is visible between the model trained with FGSM adversarial training and the PGD one, especially on the CIFAR10 dataset. In particular, referring to the more challenging dataset, the GCCS approach reaches 83% accuracy in the case of no attack. The benefit of adversarial training becomes clear when analyzing a high noise power budget, in particular in Figure 5.1 with the case of the FGSM attack: in this scenario, adversarially trained models degrade to 78% classification accuracy while standardly trained model worsens to 59% accuracy. This difference is much larger in the case of the PGD attack, where the standard training only leads to 34% accuracy.
Cross-entropy loss function, instead, is more vulnerable even when a simple FGSM attack is issued. Indeed, it starts from an accuracy of 78% when examples are not perturbed and the metric gets worse when noise power increases. The trend is particularly visible when issuing an iterative PGD attack: in this case, all models that rely on cross-entropy drop their classification accuracy by just $\epsilon = 0.004$. Although standard training always performs worse, cross-entropy shows an interesting behavior when trained with adversarial examples: in Figure 5.1 the FGSM-trained model seems to be more robust but the scenario is overturned in Figure 5.2, where PGD results are reported.
Similar conclusions can be made for tests done on the MNIST dataset: also in these scenarios, GCCS performs better than cross-entropy, especially when combined with adversarial training. In these experiments, AT-PGD always seems to be slightly better than AT-FGSM for both GCCS and cross-entropy loss functions.

Analyzing now targeted attacks reported in Figures 5.3 and 5.4, similar conclusions can be made. Also in this test, the benefits of training with adversarial examples are clearly visible as both GCCS and cross-entropy perform better if compared to standardly trained models. Just as in the PGD attack, GCCS seems to be more robust than cross-entropy, especially when it comes to considering a larger power budget ϵ. Also in this context, AT-FGSM and AT-PGD lead to quite similar results when GCCS loss is employed, while the latter is preferable when the cross-entropy loss is chosen since it slightly improves the robustness.

Referring to the MNIST dataset, the advantages of GCCS over cross-entropy are emblematic when looking at the TGSM attack in Figure 5.3. Indeed, with $\epsilon = 0.03$ the former ensures 85% accuracy while the latter drops to nearly 10%.

Different conclusions can be drawn for the JSMA attack on CIFAR10: Figure 5.4 shows that any of the adversarially trained models is more accurate than models obtained with standard training. Indeed, both cross-entropy models trained with adversarial examples perform better than the standard GCCS model. However, just as in the other cases, when the GCCS method is combined with AT, especially considering the AT-PGD, higher performances are ensured. Instead, considering cross-entropy loss function, AT-FGSM has to be preferred until $\epsilon = 0.004$: above this noise power budget, AT-PGD performs better.
A different scenario can be noticed when dealing with the MNIST dataset: in this case, standard GCCS equals PGD adversarial training on cross-entropy. As expected, combining GCCS loss function with PGD adversarial training guarantees the highest performances with 70% accuracy in the case of $\epsilon = 0.1$. On the other hand, the standardly trained cross-entropy model seems to be the less robust among the tested ones.

5.2 Deepen PGD adversarial training

As presented in Section 5.1, PGD is often an optimal method to craft adversarial examples during training. Indeed, it leads to comparable or even higher robustness of the model if compared to AT-FGSM. For this reason, AT-PGD has been chosen as default adversarial training during the next experiments: this means that all GCCS and cross-entropy models that will be presented in the following are obtained with AT-PGD.
In particular, the next task consists of finding the most robust model, investigating various combinations between cross-entropy or GCCS loss functions, and standard or adversarial training. There are four relevant configurations, which are listed below since either cross-entropy or GCCS can be used as starting models. Fine tunings could be applied to further improve the performances.

- Model 1: GCCS adversarially trained
- Model 2: Cross-entropy adversarially trained + GCCS adversarially trained
- Model 3: GCCS standardly trained + GCCS adversarially trained
- Model 4: Cross-entropy standardly trained + GCCS standardly trained + GCCS adversarially trained

The focus is on GCCS models since they perform better than cross-entropy: it could be used as starting model or even applied to an already trained model, tuning the network parameters. However, it would be convenient if Model 2 and Model 4 ensured good results since advanced cross-entropy ready-to-use models are available on the internet and they could be easily fine-tuned with GCCS. Despite that, Model 1 and Model 3, which are based on the GCCS method, are expected to ensure stronger robustness.

5.2.1 Different confgurations comparison

The considered configurations are summarized in the following tables. They have been tested on CIFAR10 and SVHN datasets with the attacks presented in Sections 4.1 and 4.2. Whenever relying on GCCS loss, the selectable parameters have been set to: mean $\mu = 70$, variance $\sigma^2 = 1$, and Kurtosis balancing factor $\lambda = 0.2$.

Model	Base model	1st fine tuning	2nd fine tuning
1	GCCS AT (400 epochs)	-	-
2	Cross-entropy AT (400 epochs)	GCCS AT (60 epochs)	-
3	GCCS (400 epochs)	GCCS AT (60 epochs)	-
4	Cross-entropy (400 epochs)	GCCS (60 epochs)	GCCS AT (60 epochs)

Table 5.2: Tested models configurations on CIFAR10 dataset.

Model	Base model	1st fine tuning	2nd fine tuning
1	GCCS AT (300 epochs)	-	-
2	Cross-entropy AT (300 epochs)	GCCS AT (40 epochs)	-
3	GCCS (300 epochs)	GCCS AT (40 epochs)	-
4	Cross-entropy (300 epochs)	GCCS (40 epochs)	GCCS AT (40 epochs)

Table 5.3: Tested models configurations on SVHN dataset.

Results obtained deploying ResNet with 18 residual layers are shown in the following figures, where also reference lines are plotted. These last ones are GCCS standardly trained used as the base model for Model 3, cross-entropy standardly trained employed as the base model for Model 4, and cross-entropy adversarially trained used as starting point for Model 2. As before, solid lines represent GCCS-based models while dashed lines stand for cross-entropy-based models. For both

CIFAR10 and SVHN datasets, the white box attacks have been launched considering a noise power budget between $\epsilon = 0$ and $\epsilon = 0.02$. Due to the high computational cost, a restricted range has been decided for the JSMA attack, setting the maximum noise power to $\epsilon = 0.006$.

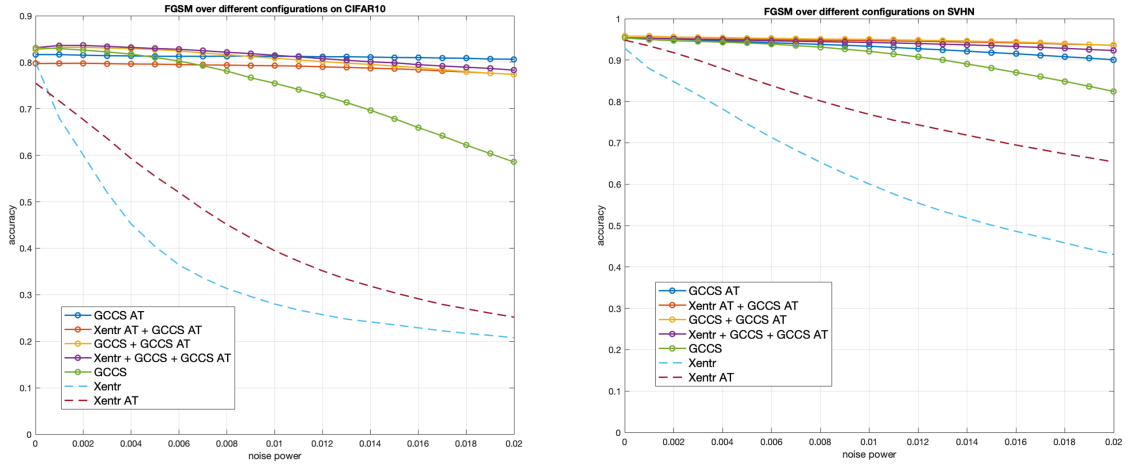


*Figure 5.5: Classification accuracy on the test set for different configurations under the FGSM attack, as a function of the perturbation budget ϵ for CIFAR10 (**left**) and SVHN (**right**).*

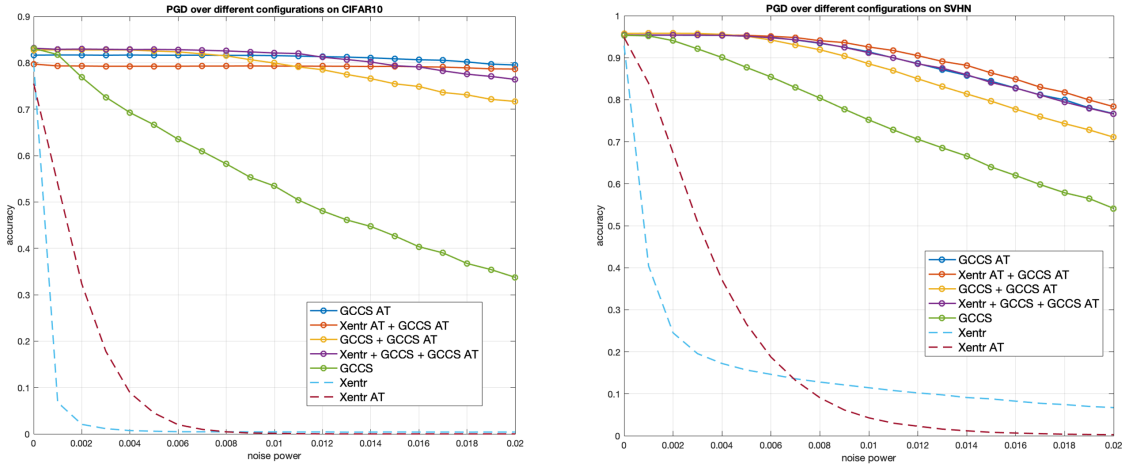


*Figure 5.6: Classification accuracy on the test set for different configurations under the PGD 5-steps attack, as a function of the perturbation budget ϵ for CIFAR10 (**left**) and SVHN (**right**).*

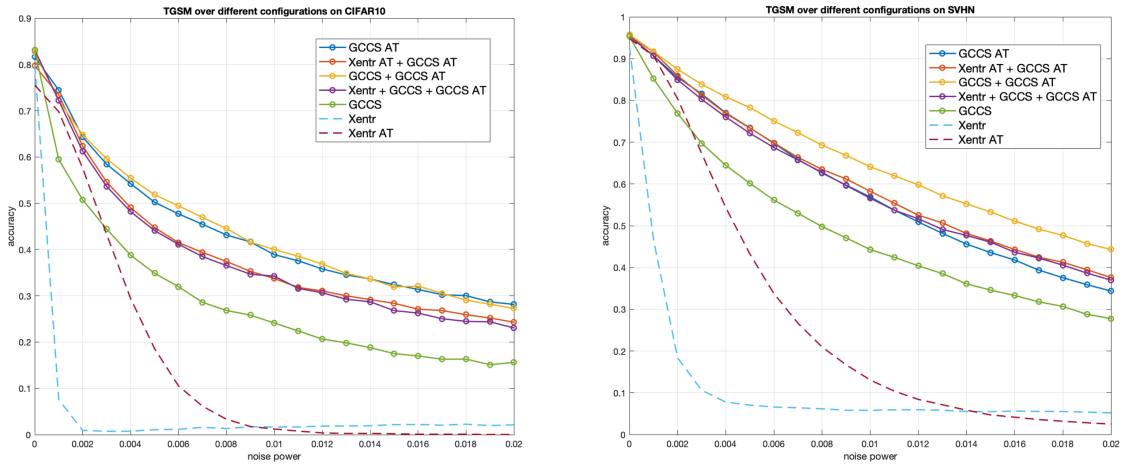


*Figure 5.7: Classification accuracy on the test set for different configurations under the TGSM 5-steps attack, as a function of the perturbation budget ϵ for CIFAR10 (**left**) and SVHN (**right**).*

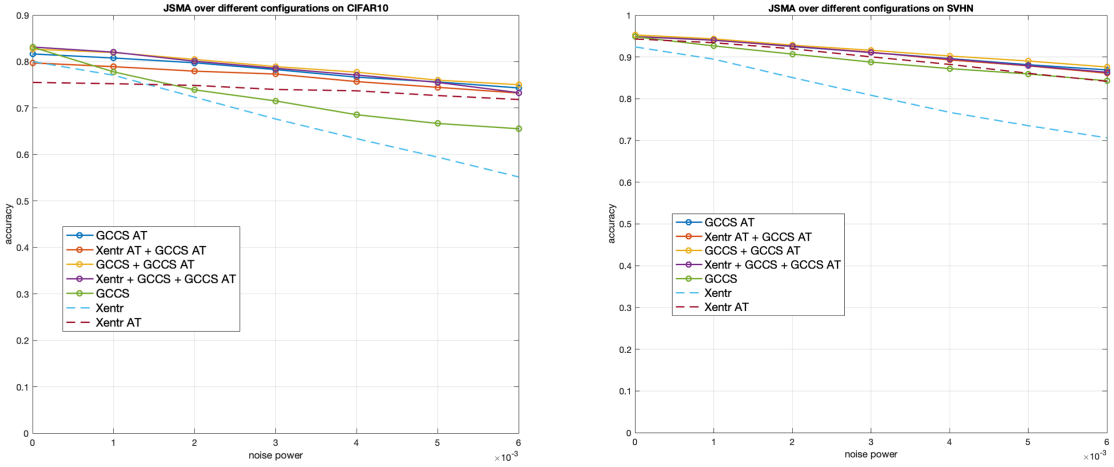


*Figure 5.8: Classification accuracy on the test set for different configurations under the JSMA 200 steps 1-pixel attack, as a function of the perturbation budget ϵ for CIFAR10 (**left**) and SVHN (**right**).*

As expected, all tested configurations perform better than the previously mentioned reference lines, as they guarantee higher robustness. They all behave similarly since the curves can often be superimposed on each other. For this reason, it is difficult to state which is the most robust model.
Considering the FGSM attack in Figure 5.5, all configurations show comparable behaviors. Referring to CIFAR10 results, they all guarantee similar accuracies, always

higher than 80% for every considered ϵ; on SVHN they begin to deviate from each other with large ϵ, having Model 3 as the best model and Model 1 as the worst one. When attacking the networks with PGD attack, instead, Model 3 appears to perform slightly worse when compared to the other configurations, reducing the accuracy by 5% in the case of $\epsilon = 0.02$ on both datasets. In this case Model 1 and Model 2 perform quite well.
Considering now targeted attacks, TGSM in Figure 5.7 leads to the conclusion that Model 3 outperforms all other tested configurations. On CIFAR10 Model 3 is comparable to Model 1 and they both show higher robustness if compared to cross-entropy-based models. On SVHN, instead, Model 3 is the only configuration well above the others. Finally, the JSMA attack leads to quite similar results for both datasets: in these cases, curves related to all configurations are superimposed, thus comparable classification accuracies are ensured. Also in these cases, Model 3 is slightly higher than other curves when looking at the maximum tested ϵ.

Generally, by observing the experimental results, one can suppose that Model 3 (GCCS + GCCS AT) is often the optimal choice. Indeed, as described, Model 3 behaves in an optimal way showing higher classification accuracy than the other configurations in all performed attacks on both datasets, except for the PGD attack. In addition to that, one can notice that looking at $\epsilon = 0$ points, i.e. when no attack is employed, Model 3 always reaches the highest working point for both datasets, ensuring a great accuracy in normal conditions.
However, we observed certain situations where Model 2 and Model 4 are not the least robust methods: in particular, they both ensure great robustness for FGSM and PGD attacks on the SVHN dataset, shown in Figures 5.5 and 5.6.

For this reason, I further investigate the behavior of models trained with the different configurations by also employing the DeepFool attack. Recalling Section 4.1.3, results obtained by the DeepFool attack are evaluated using the ρ_{adv} metric expressed in formula 4.5. For this reason, the robustness of the tested models cannot be plotted like the reached accuracy as a function of noise power anymore, since this approach returns a numerical value instead.

The DeepFool attack has been launched and the following results have been obtained.

Model	**Description**	**CIFAR10**	**SVHN**
1	GCCS AT	2.1922	3.2764
2	Cross-entropy AT + GCCS AT	2.0799	2.8011
3	GCCS + GCCS AT	2.3735	3.1166
4	Cross-entropy + GCCS + GCCS AT	2.1523	2.6179

Table 5.4: DeepFool attack results, expressed by ρ_{adv} parameter, for the tested configurations on CIFAR10 and SVHN.

Data reported in Table 5.4 confirm the previously described results. As initially supposed, models based on GCCS, i.e. Model 1 and Model 3, perform better than models based on the cross-entropy loss function. In particular, dataset Model 3 seems to be the most robust configuration on the CIFAR10 while Model 1 is slightly better on the SVHN dataset.

For the sake of completeness, the DeepFool attack has been also issued on the previously reported reference models. Results are shown in Table 5.7.

Description	CIFAR10	SVHN
GCCS	2.5669	2.9177
Cross-entropy	1.3869	1.4735
Cross-entropy AT	0.2027	0.4632

Table 5.5: DeepFool attack results, expressed by ρ_{adv} parameter, for reference models on CIFAR10 and SVHN.

Also these data show that GCCS always ensures greater robustness than cross-entropy. In addition to that, one can notice that standardly trained models look stronger than the corresponding adversarially trained ones when the DeepFool attack is issued. In particular, on CIFAR10, standard GCCS reaches $\rho_{adv} = 2.5669$ while the most robust adversarially trained configuration (Model 3 with: GCCS + GCCS AT) only equals $\rho_{adv} = 2.3735$. The same behavior can be noticed considering cross-entropy: in this case, the adversarially trained model is almost 7 times weaker than the standardly trained model.

5.3 Latent space analysis when applying PGD and TGSM

As said when describing the implemented attacks, TGSM is a variation of the PGD attack with the fundamental difference of targeting a specific class instead of causing a general misclassification. In principle, an untargeted attack should be easier than a targeted one as it is necessary to take a gradient step in the direction of any hyperplane used as a decision rule; for targeted attacks, instead, this step should have a direction that points the target class. Despite that, figures 5.6 and 5.7 show an interesting behavior: while cross-entropy-based models confirm this hypothesis, GCCS ones seem to behave oppositely. Indeed, it can be noticed how GCCS better resists PGD rather than TGSM attack. With the purpose of better understanding the mentioned graphs, the learned distributions in the latent space have been studied in the case of both loss functions for the mentioned attacks on the CIFAR10 dataset.

More specifically, Model 3 (GCCS + GCCS AT) and reference cross-entropy adversarially trained models have been compared. The distributions have been plotted first in the case of no issued attack, thus $\epsilon = 0$.

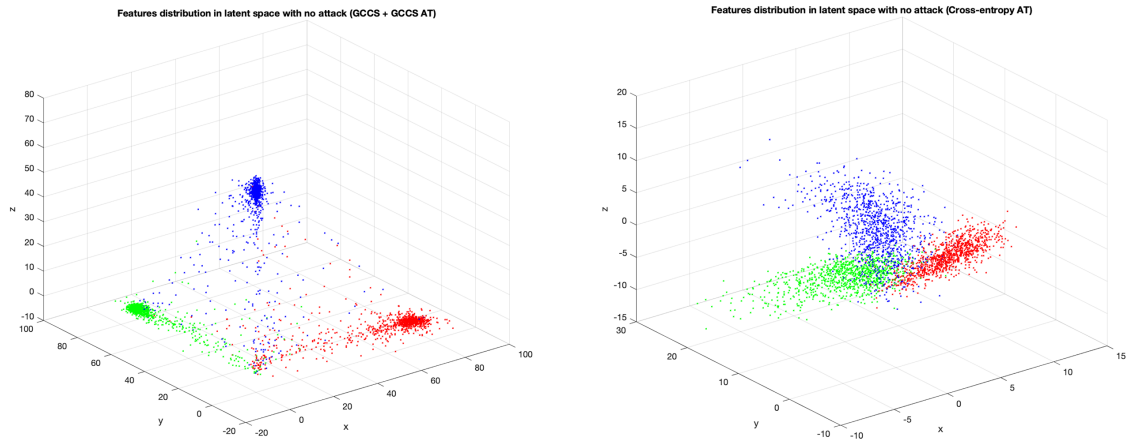


*Figure 5.9: Feature distributions in the latent space with no attack for Model 3 (GCCS + GCCS AT) (**left**) and cross-entropy adversarially trained (**right**).*

Then the visual representation has been analyzed considering different strengths of the attacks, various noise powers such as $\epsilon = 0.005$, $\epsilon = 0.01$, and $\epsilon = 0.02$ have been selected.

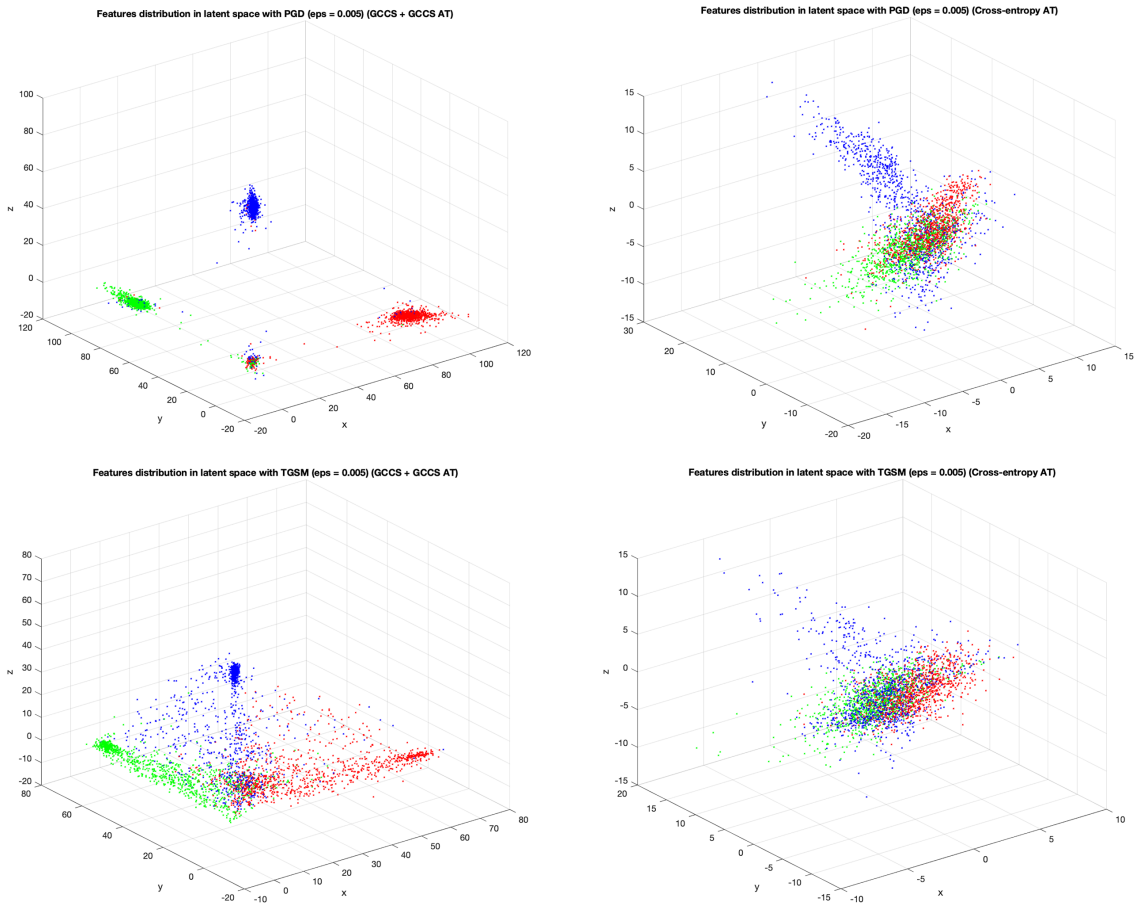


*Figure 5.10: Feature distributions in the latent space with PGD and TGSM attacks having $\epsilon = 0.005$ for Model 3 (GCCS + GCCS AT) (**left**) and cross-entropy adversarially trained (**right**).*

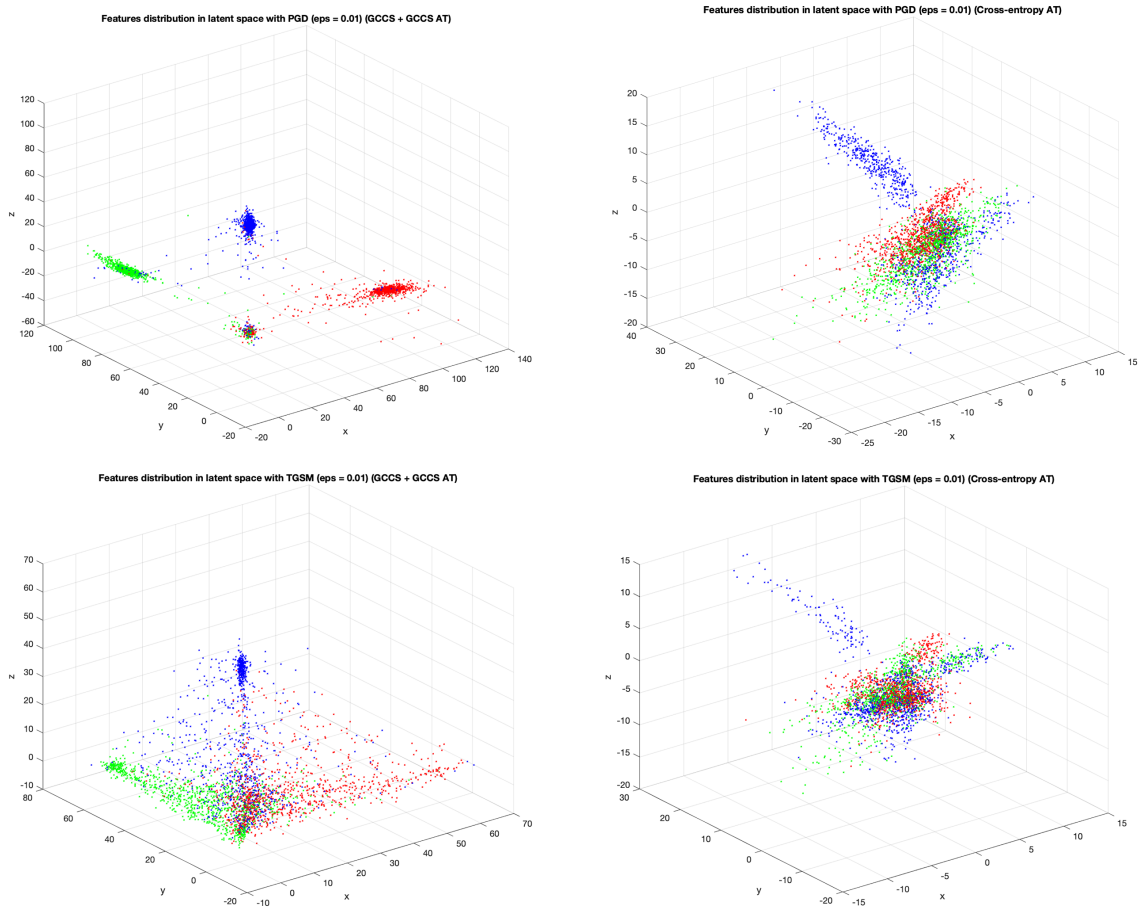


Figure 5.11: Feature distributions in the latent space with PGD and TGSM having $\epsilon = 0.01$ *for Model 3 (GCCS + GCCS AT) (**left**) and cross-entropy adversarially trained (**right**).*

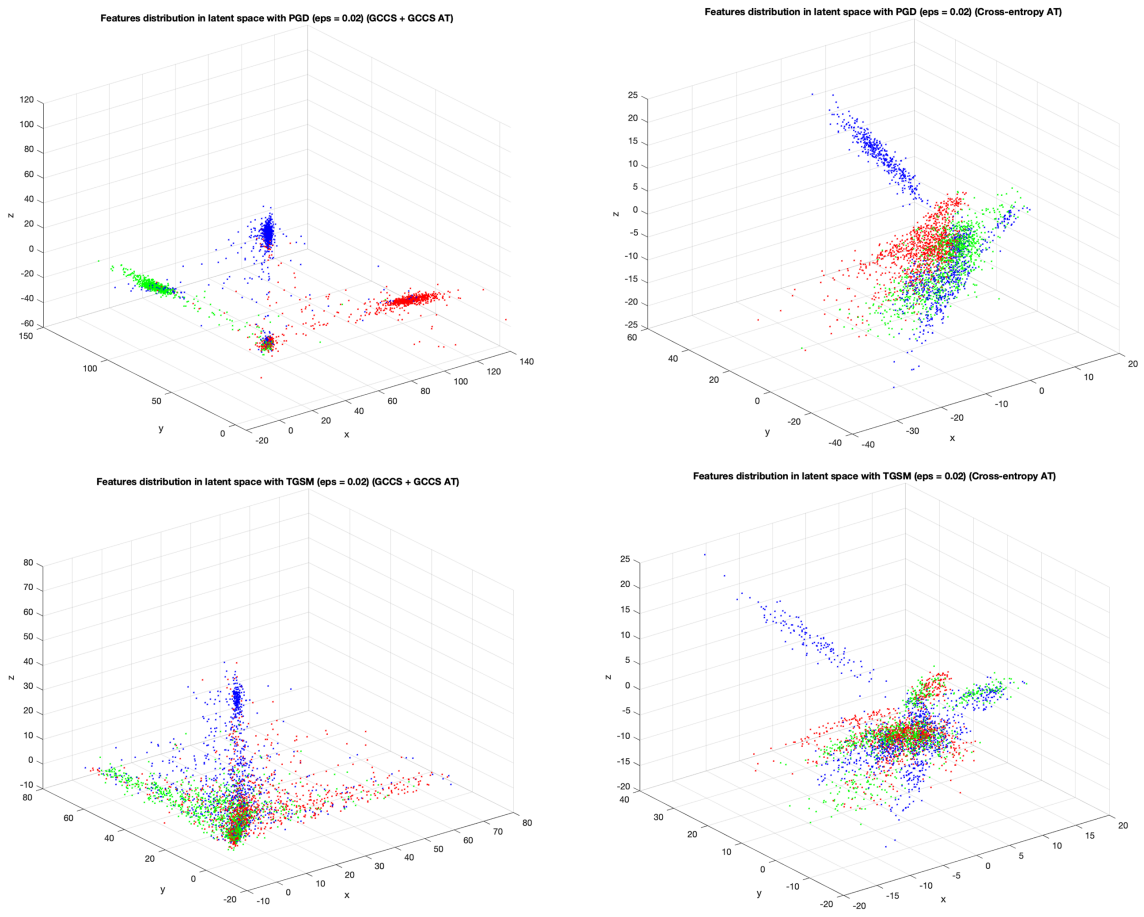


*Figure 5.12: Feature distributions in the latent space with PGD and TGSM attacks having $\epsilon = 0.02$ for Model 3 (GCCS + GCCS AT) (**left**) and cross-entropy adversarially trained (**right**).*

Figures 5.10, 5.11, and 5.12 show that, in the case of the PGD attack, when GCCS is employed classes are always well-separable, even considering large noise powers. When applying TGSM, instead, classes tend to mix near the origin of the axes. This trend is much more clear when increasing the noise power ϵ, as shown in Figure 5.12. In the case of the cross-entropy loss function, features belonging to different classes are separable just in the case of no applied attack. Indeed, either with PGD or TGSM attack they mix even when small noise powers ϵ are considered. This condition leads to very low classification accuracy as the learned distributions are not well-separable anymore.

5.4 Additional experiments

5.4.1 Insights into iterative attacks

In this section, the previously considered PGD and TGSM iterative attacks have been furtherly studied. In particular, the experiments described in previous sections aimed at evaluating the classification accuracy of the model versus the attack noise power for a chosen number of iterations. Now the tests have the purpose of obtaining data related to the classification accuracy as a function of the iterations considered in the employed attack for a fixed ϵ. In other words, previous experiments used to increase the power budget while now we are increasing the number of algorithm steps keeping constant the noise power.

As done in the previous experiments, the models already presented in Section 5.1 have been considered to investigate the benefits related to adversarial training when it comes to facing an increasing number of attacks iterations. The experiment has been carried out considering $\epsilon = 0.01$ and the maximum number of attack iteration $K = 30$. PGD and TGSM have been launched on all the mentioned models and the following results have been obtained.

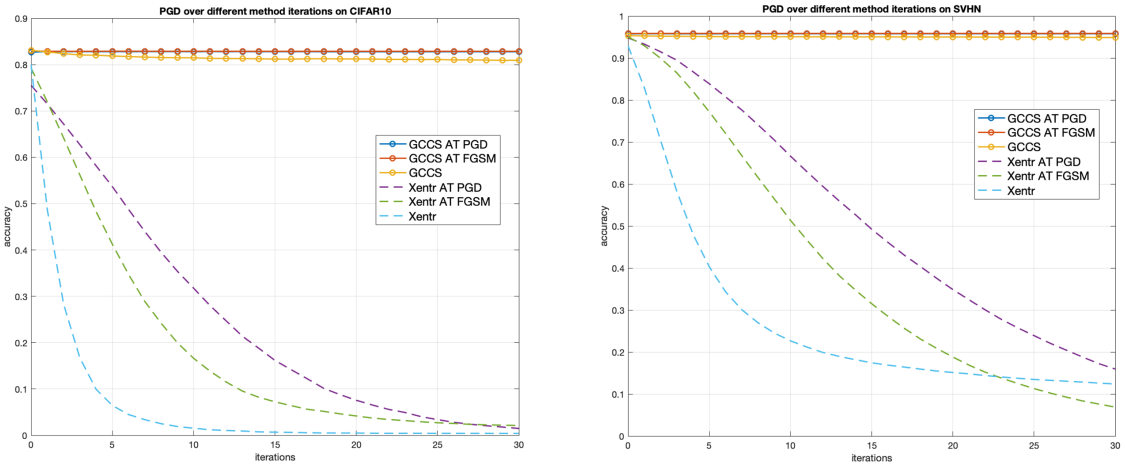


*Figure 5.13: Classification accuracy on the test set under the PGD attack ($\epsilon = 0.01$), as a function of the number of algorithm iterations for CIFAR10 (**left**) and SVHN (**right**). The figure reports models obtained with different trainings (standard training, AT-FGSM, and AT-PGD).*

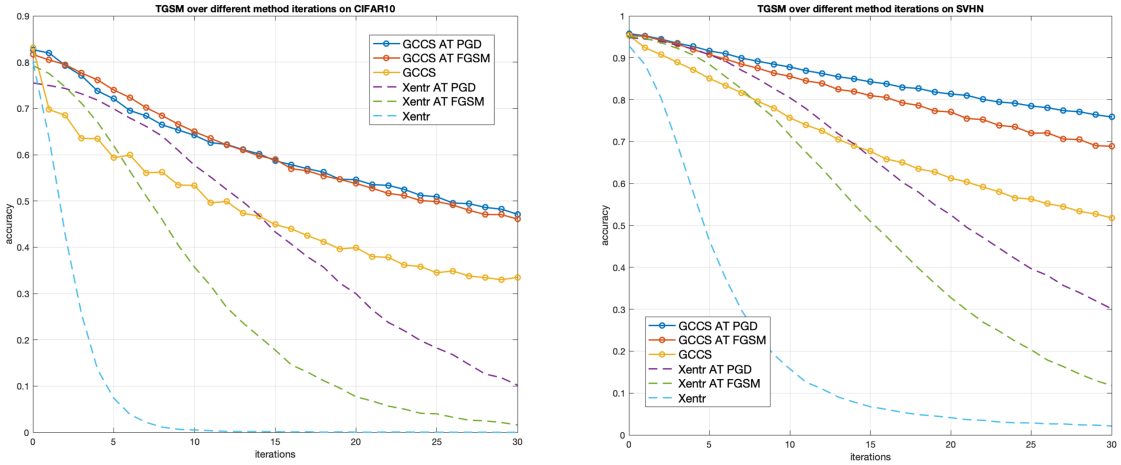


*Figure 5.14: Classification accuracy on the test set under the TGSM attack ($\epsilon = 0.01$), as a function of the number of algorithm iterations for CIFAR10 (**left**) and SVHN (**right**). The figure reports models obtained with different trainings (standard training, AT-FGSM, and AT-PGD).*

First, one can notice that GCCS always outperforms cross-entropy. After that, in the case of the PGD attack, all models based on GCCS seem to be comparable as the curves can be superimposed on each other on both datasets. Cross-entropy-based models, instead, lead to the conclusion that adversarial training improves the robustness of the model and, to further improve classification accuracy, one should apply AT-PGD instead of AT-FGSM.
Considering now TGSM attack, instead, both GCCS and cross-entropy present the same pattern with GCCS performing better than cross-entropy: AT-PGD has to be the preferred choice as it always leads to the best classification accuracy. Follows AT-FGSM and finally standardly trained model is the weakest choice among the tested ones.

In addition to the previous results, PGD and TGSM attacks have been issued to test all the previously presented configurations and reference lines with a fixed noise power $\epsilon = 0.01$. Iterations have been cycled from 0 to a maximum value of 30. The obtained results are reported in Figures 5.15 and 5.16.

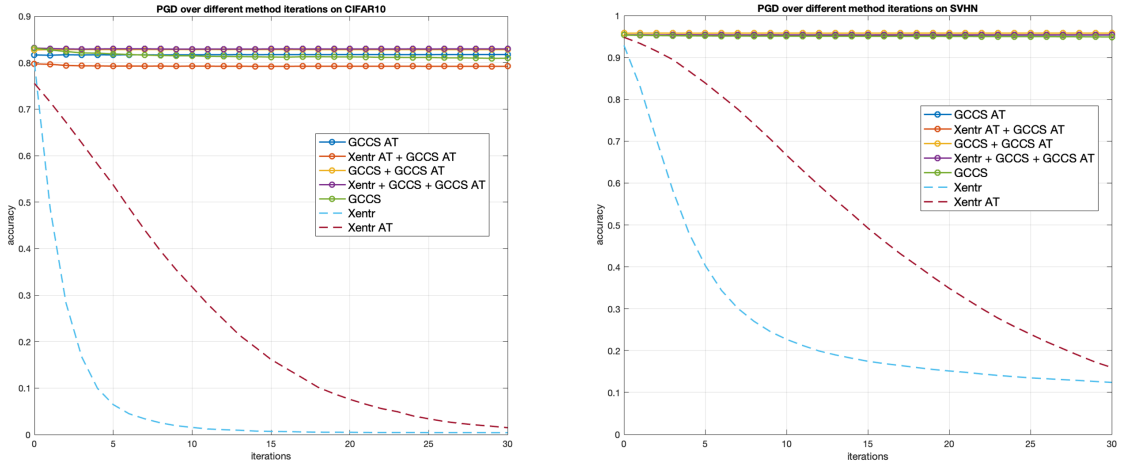


*Figure 5.15: Classification accuracy on the test set for different configurations under the PGD attack ($\epsilon = 0.01$), as a function of the number of algorithm iterations for CIFAR10 (**left**) and SVHN (**right**). The focus is on the four previously tested configurations.*

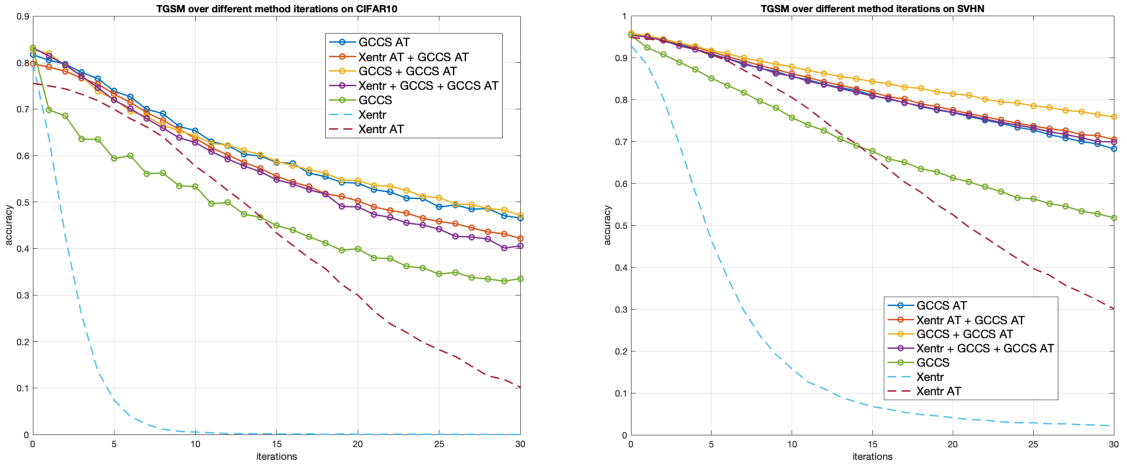


*Figure 5.16: Classification accuracy on the test set for different configurations under the TGSM attack ($\epsilon = 0.01$), as a function of the number of algorithm iterations for CIFAR10 (**left**) and SVHN (**right**). The focus is on the four previously tested configurations.*

As shown in the reported plots, also in these experiments one can clearly notice the benefits brought by the GCCS method. Indeed, looking at CIFAR10 results, curves related to GCCS combined with AT-PGD are well above the others. In particular, looking at Figure 5.15 where the PGD attack is under analysis, all GCCS

models, including also the standardly trained one, do not suffer the increment of algorithm iterations, as they remain constant to their initial value. Referring to cross-entropy models, instead, a significant performance improvement is ensured when training the network with adversarial examples. The same conclusions can be made for the SVHN dataset as the obtained characteristics are very similar to the previous cases. Finally, one can state that Model 3 leads to the best classification accuracies also in these cases.

Considering now TGSM attack on CIFAR10 reported in Figure 5.16, all AT-PGD GCCS models are more robust than reference curves, especially for high iteration values. In particular, also in these tests Model 3 seems to be the less weak model, laying above the other curves. However, GCCS standardly trained shows worse performances than adversarially trained cross-entropy for a number of iterations lower than 15. Above this value, the two curves swap with the GCCS model leading to greater classification accuracy. Again, the same conclusion can be stated for the SVHN dataset.

5.4.2 Target class assignment

Focusing on targeted adversarial attacks, one can further investigate the method employed to assign the target class. Indeed, if just a single approach is chosen, possible biases may impact the results. For this purpose, multiple algorithms have been tested aiming at deepening this behavior. Attacks issued until this section used to assign the target class according to the following expression:

$$target = (true + 1) \mod C \tag{5.1}$$

where C represents the output size, i.e. the number of possible output labels. For instance, considering MNIST, SVHN, or CIFAR10, the presented expression becomes:

$$target = (true + 1) \mod 10 \tag{5.2}$$

as they all present ten possible output classes.

Since class labels are expressed by means of integer numbers from 0 to 9, the mentioned formula simply assigns an adversarial label that follows in sequence the true label. Assignments are performed according to the following Table 5.6.

True label	**Assigned target label**
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	0

Table 5.6: Target class assignments according to expression 5.1.

To exclude the possibility of having biases related to the target class, a random target class assignment over the batch has been employed. The algorithm simply reads the true class and iteratively tries to assign a different target, always in the range between 0 and 9, for each iteration of the tested attack. The pseudo-code of this assignment is reported in the following.

Algorithm 5 Randomly assigned target class

```
1: while i < batch size do
2:     target ← random(0; 9)
3:     while target == true do
4:         target ← random(0; 9)
5:     i ← i + 1
```

Both the implemented versions have been issued for TGSM and JSMA attacks on CIFAR10 and SVHN datasets. This experiment has been performed on the most robust configuration (GCCS + GCCS AT) and all the reference lines, i.e. GCCS, cross-entropy, and cross-entropy adversarially trained models. As previously discussed, I have considered the TGSM attack for a noise power $\epsilon \in [0\ 0.02]$ while, due to the high computational cost, JSMA has been launched on a restricted range with a maximum ϵ that equals 0.006. The obtained results are shown in the following Figures 5.17 and 5.18.

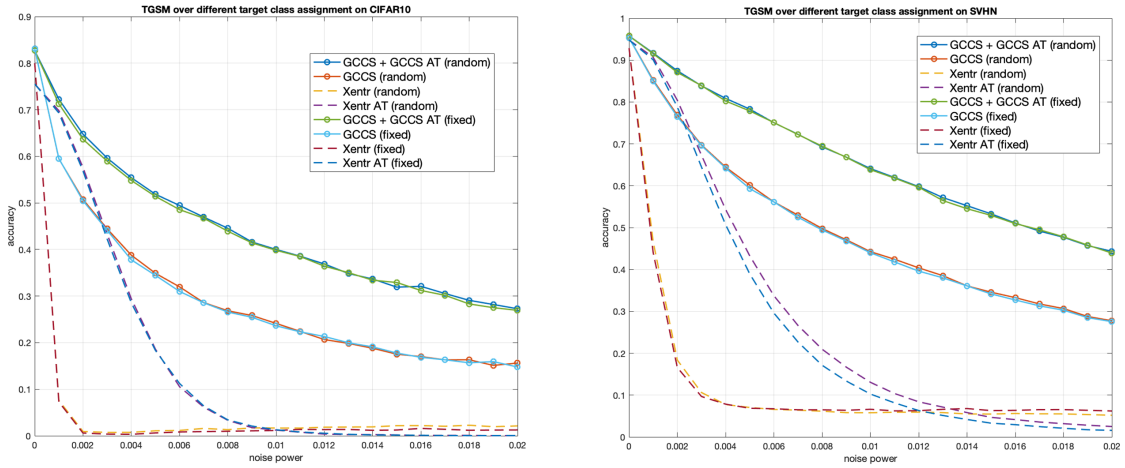


*Figure 5.17: Classification accuracy on the test set with various target class assignments under the TGSM 5-steps attack, as a function of the perturbation budget ϵ for CIFAR10 (**left**) and SVHN (**right**).*

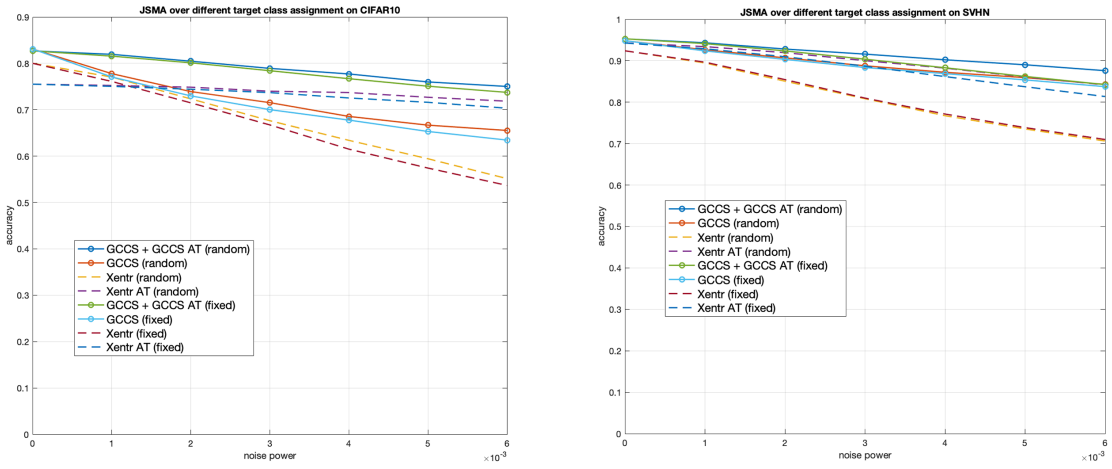


*Figure 5.18: Classification accuracy on the test set with various target class assignments under the JSMA 200-steps 1-pixel attack, as a function of the perturbation budget ϵ for CIFAR10 (**left**) and SVHN (**right**).*

As shown in Figures 5.17 and 5.18, the accuracy on the test set while under the two considered targeted attacks does not correlate to the target label assignment. Indeed, curves with different assignments related to TGSM and JSMA on both considered datasets are comparable and superimposed. This behavior is shown for all tested models even with different trainings.

5.4.3 Target distribution parameters

Finally, I investigate the role of the chosen mean value for features distributions in the GCCS approach. The previously found most robust configuration, Model 3 (GCCS + GCCS AT), has been trained multiple times setting different values for the mean parameter μ required by the GCCS method. In particular, three values have been chosen: $\mu = 40$, $\mu = 70$, $\mu = 100$; the other requested parameters have been fixed to the previous values: variance $\sigma^2 = 1$ and Kurtosis balancing factor $\lambda = 0.2$. FGSM, PGD, and TGSM attacks have been issued on CIFAR10 and SVHN, providing the results reported in the following figures.

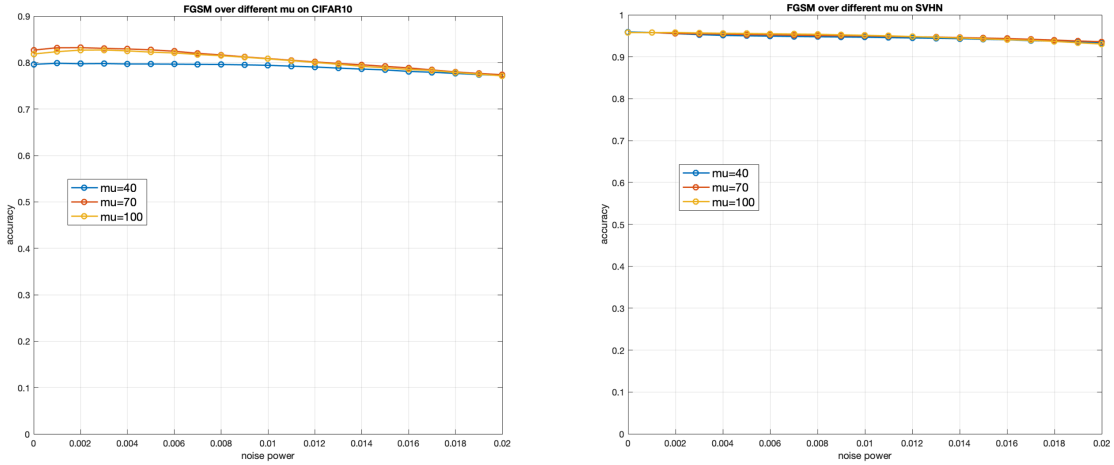


*Figure 5.19: Classification accuracy on the test set with mean values under the FGSM attack, as a function of the perturbation budget ϵ for CIFAR10 (**left**) and SVHN (**right**).*

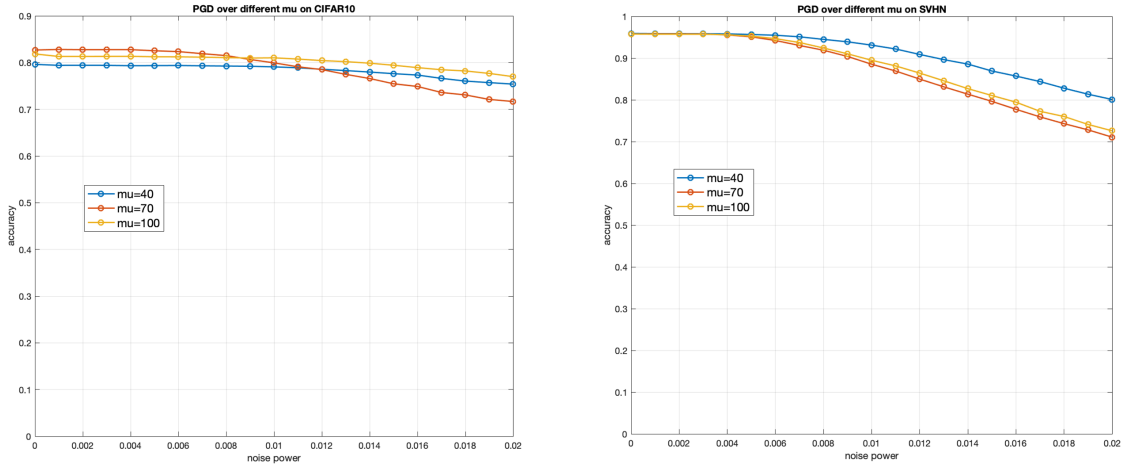


*Figure 5.20: Classification accuracy on the test set with mean values under the PGD 5-steps attack, as a function of the perturbation budget ϵ for CIFAR10 (**left**) and SVHN (**right**).*

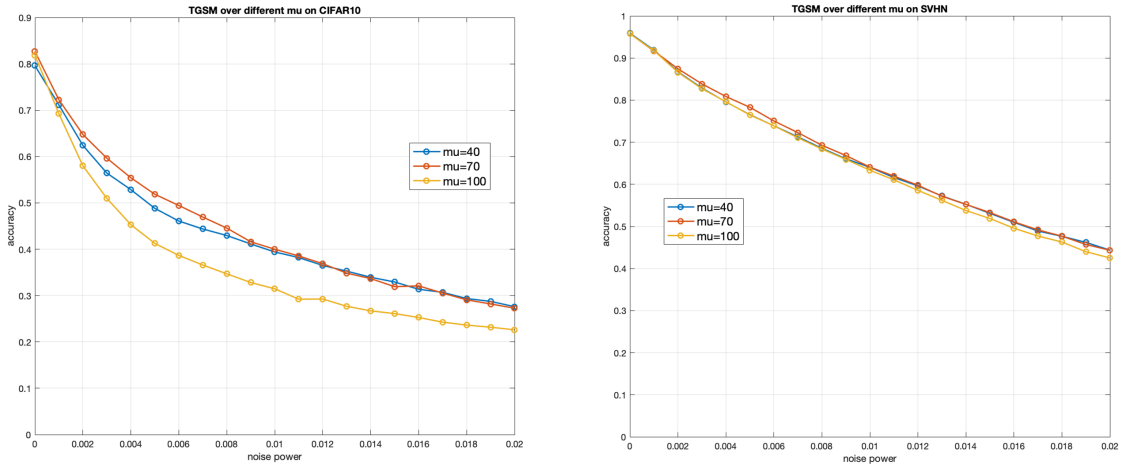


*Figure 5.21: Classification accuracy on the test set with mean values under the TGSM 5-steps attack, as a function of the perturbation budget ϵ for CIFAR10 (**left**) and SVHN (**right**).*

As shown in the reported figures, models characterized by different mean values generally reach similar classification accuracies. This behavior is particularly enhanced in the FGSM attack shown in Figure 5.19, where curves related to various μ are superimposed on each other on both datasets, especially for large noise powers. Referring to the PGD attack and trading off the results, the lowest mean considered, equal to $\mu = 40$, seems to be the preferred choice. Finally, looking at the TGSM attack, the model trained with the highest mean value is the weakest among the considered ones. One can expect that the greater the mean value the more robust the model. However, considering very large μ values beyond a certain bound, the model may not converge properly. Such a result is related to the functionality of the employed encoder. Indeed, large mean values may make the training unstable, with the subsequent reduction of the classification accuracy. The ratio between μ and σ parameters of the target distributions has to be carefully fixed considering the number of the latent space dimensions. In the particular scenario that I studied, where the presented classification problems have a total of 10 output classes, the $\mu = 40$ seems to be the optimal choice. As said, such a parameter may be set to different target values depending on the considered context. Finally, the performances associated with the three tested configurations are comparable when analyzing the TGSM attack on the SVHN dataset.

In addition to that, the same experiment has also been performed considering the DeepFool attack. The obtained ρ_{adv} are reported in the following table.

Mean value	**CIFAR10**	**SVHN**
$\mu = 40$	1.4742	1.9171
$\mu = 70$	2.3735	3.1166
$\mu = 100$	2.7953	3.9131

Table 5.7: DeepFool attack results, expressed by ρ_{adv} parameter, for different mean values on CIFAR10 and SVHN.

The results highlight an expected trend: the higher the mean value of the feature distributions the more robust the model. Indeed, the higher the mean value the farther the distributions will be between each other, allowing a larger inter-class separation. Therefore, the average required perturbation to get a misclassification onto the closest hyperplane becomes greater. Notice that this result seems to be in contrast with the previously reported plots. However, as already explained, they cannot be compared since the classification accuracy is based on norm ∞ while ρ_{adv} is obtained computing norm 2. Moreover, ρ_{adv} averages the required energy to misclassify an input image while the previously reported plots are worst-case results: they show the minimum classification accuracy for a given perturbation budget ϵ.

Chapter 6

Conclusions and future works

The experiments I carried out show the improvement of performance brought by GCCS when compared with other competing methods. In particular, models trained with the GCCS method are more robust to adversarial examples, i.e. they ensure greater classification accuracy than competing methods when attacks with the same noise power are employed. Such a condition can be reached thanks to the basic principle of the GCCS method, which enables the learning of feature distributions placed on the vertices of a simplex laying in a multi-dimensional latent space. In this way, features belonging to the same class are grouped close to each other and placed at a great distance with respect to the other classes, thus ensuring high inter-class separation and low intra-class dispersion.

In addition to that, the experiments also show that combining GCCS with adversarial training guarantees higher robustness than other state-of-the-art methods. During the training phase, different approaches have been tested to craft adversarial examples: AT-FGSM and AT-PGD. Although on GCCS models they performed similarly, generally, the latter has to be chosen since it leads to slightly better results. This condition is particularly enhanced when considering other state-of-the-art methods, such as cross-entropy models.
Different models have been subjected to various attacks. The most robust configuration among the tested ones, listed in Table 5.2 is obtained performing a first standard training with GCCS loss function and fine-tuning the obtained model with AT-PGD always with GCCS method. The resulting model (GCCS + GCCS AT) shows excellent classification accuracy in the nominal case and high robustness when attacked.

In contrast to other approaches, GCCS also shows great performances when increasing the number of steps of an iterative attack. In particular, GCCS-based models show great classification accuracy even with a very large number of iterations of a PGD attack. Considering cross-entropy models such behavior cannot be observed, as classification accuracy decreases with just 10 iterations.

To further compare the proposed GCCS framework with other defenses, the evaluation may be extended considering other attacks. The focus of this thesis has been placed on the evaluation of the GCCS method versus gradient-based generation techniques for adversarial examples. Future experiments may include also gradient-free or hard-label attacks. The former ones are effective even when gradient-based attacks fail, meaning that the gradient of the network is somehow masked and the defense can be easily bypassed. An example of an implementable gradient-free attack is **SPSA** [23]. The latter ones, instead, only require access to the most confident output label. They are much slower than other methods as they make many more queries, but defenses can do less to accidentally prevent from these attacks. **The Boundary Attack** is an example of a hard-label attack [24].

Bibliography

[1] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[2] Alexey Kurakin, Ian Goodfellow, Samy Bengio, et al. Adversarial examples in the physical world, 2016.

[3] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[4] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[5] Ilija Mihajlovic. Everything you ever wanted to know about computer vision. `https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e`. Updated: 2019-04-25.

[6] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[7] Andrej Krenker, Janez Bešter, and Andrej Kos. Introduction to the artificial neural networks. *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech*, pages 1–18, 2011.

[8] Zheng Hu, Jiaojiao Zhang, and Yun Ge. Handling vanishing gradient problem using artificial derivative. *IEEE Access*, 9:22371–22377, 2021.

[9] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[10] Convolutional neural networks for visual recognition course (backpropagation, intuitions) - stanford university. `https://cs231n.github.io/optimization-2/`. Updated: 2021-03-15.

[11] Convolutional neural networks for visual recognition course (optimization: Stochastic gradient descent) - stanford university. `https://cs231n.github.io/optimization-1/`. Updated: 2021-03-15.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[13] Kiprono Elijah Koech. Cross-entropy loss function. `https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e`. Published: 2020-10-02.

[14] Training and test sets: Splitting data. `https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data`. Updated: 2020-02-10.

[15] Arslan Ali, Andrea Migliorati, Tiziano Bianchi, and Enrico Magli. Beyond cross-entropy: learning highly separable feature distributions for robust and accurate classification. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 9711–9718. IEEE, 2021.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] Voronoi diagram. `https://en.wikipedia.org/wiki/Voronoi_diagram`. Updated: 2021-09-17.

[18] Oscar Knagg. Know your enemy. how you can create and defend against adversarial attacks. `https://github.com/oscarknagg/adversarial/blob/master/notebooks/Creating_And_Defending_From_Adversarial_Examples.ipynb/`. Published: 2019-01-06.

[19] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[20] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[21] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pages 274–283. PMLR, 2018.

[22] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*, 2019.

[23] Jonathan Uesato, Brendan O'donoghue, Pushmeet Kohli, and Aaron Oord. Adversarial risk and the dangers of evaluating against weak attacks. In *International Conference on Machine Learning*, pages 5025–5034. PMLR, 2018.

[24] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017.