UCLouvain

École polytechnique de Louvain

Binarized Neural Networks: Adversarial Image Generation and Robustness

Author: **Benoît Ronval**
Supervisor: **Siegfried Nijssen**
Readers: **Lucile Dierckx, Michel Verleysen**
Academic year 2022–2023
Master [120] in Computer Science and Engineering

Abstract

In this master thesis, we design and test a new approach called the robust training loop, used to perform adversarial training of binarized neural networks. Even if deep learning models can obtain good scores for various tasks, they are easily fooled by small modifications of the data given as input. Preventing such unwanted behavior may turn out to be a complex task because of the difficulty to interpret how the model really works. State-of-the-art solutions for real-valued neural networks generally use information from the gradient to create the modification. They then use the disrupted samples in a specific training algorithm, along with a modification of the loss function. However, each kind of modification is linked to a specific training algorithm and uses directly the gradient from the model, which is not usable with all types of deep learning models.
In this thesis, we focus on binarized neural networks since they can be exactly verified, which means we can create the modification by understanding how they perform their task. With these modified samples, we propose an extension of the classical training algorithm, named the robust training loop, in order to improve the resistance of such models against modification of their inputs. We believe this algorithm can be easily extended to other methods of sample modification than the one used in this work. Through extensive experiments, we show that our approach can indeed improve robustness according to the considered criteria.

Acknowledgements

I want to thank all the people who have supported and helped me during the realization of this work.

I would like to give a huge thank you to my advisor, Professor Siegfried Nijssen, for his availability and guidance during all this year. Without him and his help, this work would not be as it is now. His precious advice for the thesis and the PhD proposal really pushed me to give the best of myself.

I would also like to thank Lucile Dierckx and Professor Michel Verleysen for agreeing to be part of the jury as a reader.

Finally, I would not be there without the support of my parents and friends. I would like to thank all of them for helping me to progress during all these years, and for encouraging, supporting and reassuring me when it was necessary.

List of abbreviations

Adv	Adversarial
AI	Artificial Intelligence
ANN	Artificial Neural Network
BN	Bacth Normalization
BNN	Binarized Neural Network
CDCL	Conflict-Driven Clause Learning
CNF	Conjunctive Normal Form
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
DPLL	Davis-Putnam-Logemann-Lovelan
EEV	Efficient and Exact Verification
EEVBNN	Efficient and Exact Verification of Binarized Neural Networks
eps	epsilon, ϵ
FGS, FGSM	Fast Gradient Sign, Fast Gradient Sign Method
FGV	Fast Gradient Value
GAN	Generative Adversarial Network
ILP	Integer Linear Programming
MILP	Mixed Integer Linear Programming
ML	Machine Learning
MLP	Multi-Layer Perceptron
PGD	Projected Gradient Descent
SAT	Satisfiable
SBN	Shift-based Batch Normalization
SGD	Stochastic Gradient Descent
SMT	Satisfiability Modulo Theory
TLE	Time Limit Exceeded
TN	True Negative
TP	True Positive
UNSAT	Unsatisfiable

Contents

Chapter 1

Introduction

Artificial Intelligence, Machine Learning and Deep Learning. For more than a decade, these domains of Computer Science have been growing very fast with impressive results for many different tasks. Their use in everyday life has helped to improve important domains such as healthcare or security.

For a long time, high performance for a model on a given task was enough. It appears now that, with good results put aside, deep learning presents important flaws. Their tendency to overfit the data, their high power consumption, the difficulty to understand why it outputs a certain result or the possibility to fool them easily are all examples of current concerns with this type of model. The last criterion, corresponding to the robustness of a model, is at the center of several recent publications. The challenge is to prevent the deep learning models to be tricked by simple modification of the data given as input. The reason behind the importance of this property is to ensure that the model will be able to behave correctly when deployed into the real world.

Since these problems are known and important, many researchers are studying them and proposing solutions. Different tools can be used to reduce their impact depending on the situation, but nothing can (and probably will never be able to) solve entirely these situations.
An example of such tool is the Binarized Neural Networks. The use of binary values inside the network is able to reduce power consumption and allows us to use them on embedded devices, such as mobile phones.
Concerning the robustness, there is a distinction between the attack and the actual improvement of the property. The objective of the attack is to create a sample with small perturbation, an adversarial sample, that can fool the model. Many different attacks exist, all of them using different techniques and having strengths and weaknesses when compared to the others. On the other hand, one could have special

attention to this property during the training of the model. We call this an adversarial training and different methods to perform it exist for classical neural networks.

Our goal in this work is twofold. We study the use of a specific attack against binarized neural networks, Efficient and Exact Verification (EEV) [1], and we propose an adversarial training we call the robust training loop. Through our experiments on images, we will see how the adversarial samples evolve and their properties. We will also assess the evolution of the robustness with our specific training. To do this, we will explore different ways to evaluate such characteristic but also compare different approaches and different architectures of BNNs.
Along these tests, we will see if properties of classical neural networks concerning robustness can be found back with their binarized version.

This manuscript is separated into eight chapters. We begin with basic theoretical notions that are mandatory to understand the other chapters. Next, we will talk about robustness in more detail, explaining what it is and what is the state-of-the-art for the attacks and adversarial training with classical neural networks. It will also cover the theory behind the tool we will use for our experiments. The fourth chapter will present our proposed approach. We detail there the algorithm for the robust training loop and its variants. The following chapter concerns the practical experimental setup used to perform the various tests. The sixth and main chapter talks about our results. We will first present the adversarial images obtained and explore some of their properties. The important part will be the evaluation of the robust training loop through many comparisons. Finally, we will end with a small discussion before reaching the conclusion.

Chapter 2

Theoretical background

In this first chapter, we go over theoretical notions that are important to understand all the work realized here. We will present first the principle of deep neural networks, with well-known models and how to train them. Then, we will present the theory behind binarized neural networks, a quantized version of the classical artificial neural networks. Finally, we will talk about SAT solvers as they will play an important role later in this work.

2.1 Deep Neural Networks

For the last ten years, deep learning has been evolving very fast in different domains of application. From simple image classification to image generation or from spam detectors to large language models, neural networks have been improved and diversified by many researchers. However, the mathematical theory behind this concept is nothing new, as we will see in this chapter.
Most of the following explanations are inspired by the very well-written book "Deep Learning" [2] and the course given by Pr. Verleysen and Pr. Lee at EPL [3].

2.1.1 Basic unit and formulae

With many different names, deep learning (DL), deep neural networks (DNNs) or even artificial neural networks (ANNs), this field of machine learning relies mainly on one specific component: the perceptron [4]. Starting first as a simple linear operation, it was later improved with a non-linear activation function in order to compute non-linear results. This whole structure is often referred to as single-layer non-linear regression. Figure 2.1 shows its typical architecture.

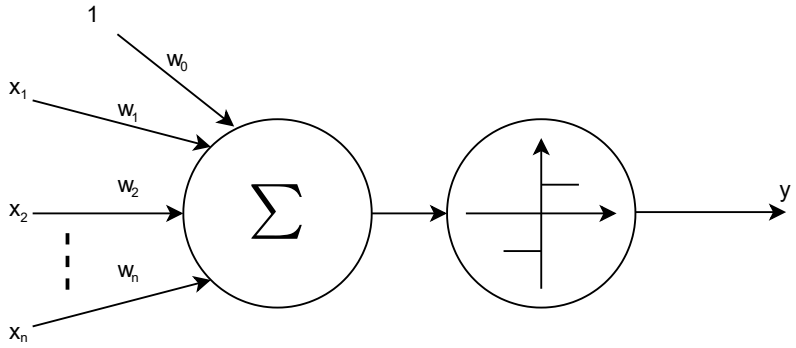


Figure 2.1: Diagram of a single-layer non-linear regression

We consider here the inputs X, a vector of n values, and the vector of weights, W. It is quite common to consider a bias, noted w_0. In the equations, it can be separated from the other weights or not, in which case the first element of X will be set to 1.
The activation function, represented in figure 2.1 by the second circle, is referred to as σ. One could use any differentiable function but we (almost) always prefer non-linear functions as they can not be represented elsewhere in the network with the linear operations. The value obtained as output from the function is called the activation value. In this simple case, the activation value is the output of the regression, y.
We can cite many different activation functions but the most used ones are the hyperbolic tangent (tanh), the sigmoid, the softmax or even the Rectified Linear Unit (ReLU) function and many variants derived from them.

This leads to the following formula at the core of deep learning:

$$y = \sigma\left(\sum_{i=0}^{n} x_i \cdot w_i\right) \tag{2.1}$$

2.1.2 Multi-layer perceptron and backpropagation

Once the single-layer non-linear regression is available, we can combine multiple of them to obtain what we call a Multi-Layer Perceptron (MLP) as shown in figure 2.2. Note that, in this case, we stick to the common approach of considering the sum operation and the activation function directly together.

The size of each component may of course vary with the current application or from arbitrary choices. The size of the inputs and outputs will be imposed by the

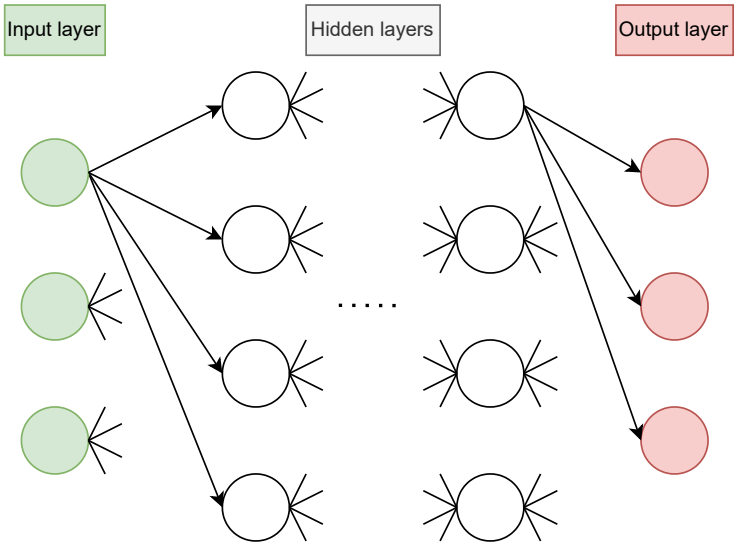


Figure 2.2: Schematic architecture of a Multi-Layer Perceptron.

data and the result we aim to obtain, while we have more liberty when it comes to the hidden layers. Indeed, we may change the number of hidden layers and the number of units inside each one of them. In its most basic aspect, it is fully connected which means all the components of the layer i are connected to all the components of the layer $i+1$.

The important values that matter here are the same as in the simple non-linear regression equation 2.1, but generalized for this bigger network. Each link between the layers has a weight value and each component of each layer has an activation value obtained from an activation function, which is often the same for the whole neural network. Therefore, to obtain the output y from an input x we compute the so-called "forward pass" with

$$y_k(x) = \sigma\left(\sum_{i=0}^{l} w_{ki}\sigma\left(\sum_{j=0}^{m} w_{ij}\sigma\left(\cdots\sigma\left(\sum_{a=0}^{n} w_{ba}x_a\right)\right)\right)\right) \tag{2.2}$$

where l, m, n are the sizes of the hidden layers and b is the index used to refer to the first hidden layer. In simple words, the input of the layer $i+1$ is the output of the layer i.

However, in order to obtain this value, we first need the weights between each layer, w_{ij}. The process of computing those elements is referred to as the training of the neural network. We use the algorithm of the backpropagation with a loss function (also called cost function, error function or even objective function) that we want

to optimize, combined with the Stochastic Gradient Descent (SGD).

The backpropagation algorithm with SGD is, in short, used to send information from the loss function applied to the output obtained from the given input. It is based on the computation of the gradients of the activation and loss functions, hence the need for them to be differentiable. The weights, initialized randomly, are then updated with the values computed from the derivatives.
We mention that many improvements of this algorithm exist to improve the computation speed, the memory space of the fully trained network or even the wanted performance. However, we will not discuss them here since they are out of the scope of this work.

The choice of the loss function has to be made during the design of the architecture of the deep network. Following the final application, one could prefer a certain function over another one. For example, if we consider a simple binary classifier (male or female, number or letter, ...), we can use the mean square error. It may also change with the type of output we expect, either a regression or a classification. In the case of a classification with multiple labels as output, which will be the case for our experiments, we tend to use the cross-entropy loss function defined as

$$H(x) = -\sum_{i=0}^{n} t_i \, log(y_i) \tag{2.3}$$

where t_i represents the true label of the input x_i and y_i its output from the network.

2.1.3 Convolutional Neural Networks

We have seen in the previous paragraph what is the most standard structure in deep learning. This expanding field of Artificial Intelligence (AI) is of course not limited to the MLP, which is why we briefly present the Convolutional Neural Network (CNN) [5].

Its main interest resides in image processing as it is able to find and exploit patterns in the given images. The main new component that performs this operation is the convolution layer. It scans through the images with a kernel and performs a convolution to extract a value that is then used to build a new image with the corresponding information.

Some additional layers are often used in this type of neural network. We can cite the pooling layers, either using the maximal or mean value, that aims to reduce the size of the images while keeping as most useful information as possible. It is often followed by some fully connected layers, like those present in the MLP.

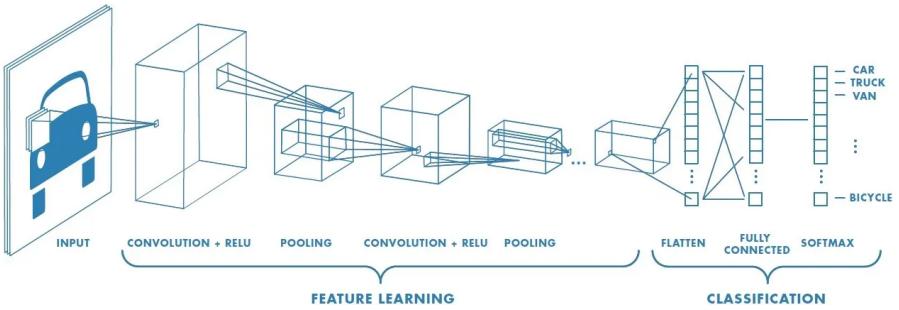


Figure 2.3: Example of Convolutional Neural Network, from `https://indabaxmorocco.github.io/materials/hajji_slides.pdf`

2.2 Binarized Neural Networks

Deep learning, as presented in the previous section, has proven to give impressive results and is still improving. As the research on this topic continues, many new architectures, tricks and optimizations emerge to solve the known problems of those powerful AI models. One of these other approaches concerns the values inside the network. It is referred to as the Binarized Neural Networks (BNNs) [6].

2.2.1 Principle

The principle of BNNs lies in the constraint of imposing a binary value to all the weights and activations instead of the traditional real values. Those binary values can either be -1 and 1 or 0 and 1 depending on the implementation we consider. However, the inputs and outputs of the model are the only elements that are not binarized. Indeed, inputs are given to the BNN as any other model and outputs correspond to what we expect, either labels or real values depending on the application.

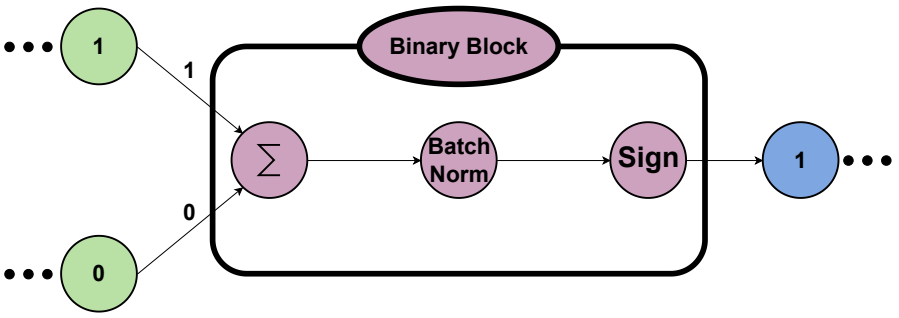


Figure 2.4: Architecture of one binary block forming the binarized neural networks

This specific type of architecture is achieved with a combination of layers that we describe below and which is illustrated in figure 2.4.
As mentioned before, the values present in the model are binary values, in this case 0 and 1. The values are then sent to a linear layout that computes the sum of the values multiplied by their weight. Therefore, it simply counts the number of neurons that are activated i.e. that have an activation value of 1.
The next step is a batch normalization (BN) layer [7]. This layer is also used in other deep learning models but plays an important role in BNNs. Its goal is to improve both the training speed and the performance of the network with trainable parameters that perform a standardization on the current batch of data.
As described in its original publication, it uses the following computation:

$$\begin{aligned} \mu_{\mathcal{B}} &= \frac{1}{m}\sum_{i=0}^{m} x_i \\ \sigma^2_{\mathcal{B}} &= \frac{1}{m}\sum_{i=0}^{m}(x_i - \mu_{\mathcal{B}})^2 \\ \hat{x}_i &= \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \\ y_i &= \gamma\,\hat{x}_i + \beta \end{aligned} \tag{2.4}$$

for the batch of data $\mathcal{B}$ which contains m data, $x_{1...m}$ and outputs the value y_i from this layer.

In BNNs, batch normalization is used mainly for the same reasons as in standard neural networks. Matthieu Courbariaux and the other original authors of the first BNN publication [6] claim that its use for this type of architecture seems to reduce the impact of the scale of the weights. However, as shown in equation 2.4, BN requires some multiplication to compute the output. To mitigate the impact of this operation in the case where the number of neurons becomes very large (typically in CNN cases), one could use a shift-based batch normalization (SBN). Its main interest is that it approximates the multiplication operation and therefore speeds up the training part. The authors claim that it does not impact the final performance of the model. SBN is defined by the following operations:

$$
\begin{aligned}
\mu_{\mathcal{B}} &= \frac{1}{m}\sum_{i=0}^{m} x_i \\
C(x_i) &= x_i - \mu_{\mathcal{B}} \\
\sigma^2_{\mathcal{B}} &= \frac{1}{m}\sum_{i=0}^{m} (C(x_i) \ll\gg AP2(C(x_i))) \\
\hat{x}_i &= C(x_i) \ll\gg AP2((\sqrt{\sigma_{\mathcal{B}^2} + \epsilon})^{-1}) \\
y_i &= AP2(\gamma) \ll\gg \hat{x}_i
\end{aligned}
\tag{2.5}
$$

where $\ll\gg$ represents both left and right binary shift and $AP2$ is the approximate power of 2.

The last part of this essential binary block is the activation function. To respect the constraint of binary values, it is mandatory that we use the sign function, defined by

$$
Sign(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ -1 \text{ or } 0 & \text{otherwise} \end{cases}
\tag{2.6}
$$

The attentive reader will note that the derivative of this function is 0 almost everywhere and therefore not really suited for the backpropagation with stochastic gradient descent. Therefore, we usually consider an approximation in order to perform the training. Many choices are available [8], we show here the most simple and straightforward one which is the approximate sign function defined by

$$
Sign_{approx}(x) = \begin{cases} x & \text{if } x \geq -1 \text{ and } x \leq 1, \\ -1 & \text{if } x < -1, \\ 1 & \text{otherwise} \end{cases}
\tag{2.7}
$$

Note that this function considers -1 and 1 binary values. The case with 0 instead of -1 is a simple adaptation from it.

We insist on the fact that this approximation is only used during the backward pass, which will therefore use non-binary values to learn all the parameters. During the forward pass, it is the original sign function that is used, enabling the binarized values in the whole structure.

2.2.2 Quantization

Quantization is the process of reducing the number of different values to a set of possible elements. Binarized neural networks are in fact an extreme example of a

quantization of a deep network by allowing only binary values for the weights and activations.
It has many applications, not limited to deep learning only. Quantization leads to famous algorithms like K-Nearest-Neighbours with Voronoi zones or even a reduction of the size of the datasets.

According to what we have said before, the inputs passed to the BNNs are real values. However, to temper the effect of the binarization and to facilitate a later verification of the model, we usually use a quantization step on the input [1].

$$x^q = \left\lfloor \frac{x}{s} \right\rceil \cdot s \tag{2.8}$$

where $x \in \mathbb{R}^n_{[0,1]}$ is the real value input, x^q represents the quantized input with $s \in [0, 1]$ the quantization step.

2.2.3 Comparison with classical neural networks

We know that classical neural network models perform very well for many tasks. They are able to reach incredible performance, especially when the inputs have some sort of structure, like an image or text. However, they often contain a huge amount of parameters, making the training and the inference (i.e. the backward and forward pass) computationally intensive, both on time and power resources.

Binarized neural networks mitigate all these problems of standard deep networks. Thanks to the binary values, the memory needed to store the model is theoretically 32 times lower than in the common case. The time and power consumption are also drastically reduced with this type of network [9]. These effects can be observed on everyday computers but especially on some dedicated hardware, such as FPGA chips [10]. Moreover, all these advantages do not prevent BNNs to reach competitive accuracies on common datasets [11].
What will interest us in the next chapters is that BNNs can be easier to verify. Indeed, their use of binary values instead of floating point numbers allows other techniques of verification as we will describe later.

However, even with these very interesting advantages, BNNs are not ready to replace classical neural networks yet. They suffer from even worse scalability problems than the other deep learning models. On a more practical aspect, they also require a full re-implementation as we can not simply impose the binary condition on the layers commonly used for deep learning. However, in the two or three past years, libraries started to appear to help developers create easily their own BNN.

2.2.4 Common implementations

As mentioned in the previous subsection, researchers first had to develop their own binarized architecture in order to work with BNNs. Nowadays, the most commonly used implementation is Larq [12], which is based on PyTorch. They propose the implementation of classical deep learning layers as well as state-of-the-art models with Larq Zoo.

The BNN implementation that will interest us comes from the publication *Efficient and Exact Verification of Binarized Neural Networks* (EEVBNN) [1]. The authors have implemented their own linear and convolution layers for binary weights and activations. They did so in order to easily run verification on the trained models. This part will be more detailed in the next chapters.

2.3 SAT solvers

2.3.1 Concept

A SAT solver, for a boolean satisfiability solver, is a computer program whose goal is to find an assignment of boolean values such that a logic formula is true.
A logic formula is made of boolean variables, either True or False, and operations connecting them through the use of logic symbols. Such formula is satisfiable (SAT) if at least one set of values makes the formula true. In the other case, it is said unsatisfiable (UNSAT).

Most solvers require that the logic formulae we aim to solve are in conjunctive normal form (CNF). This form simply consists of several clauses connected to the others with the only use of the logical "and" operator, $\wedge$. Inside a clause, we can have several boolean variables but linked only with the logical "or" operator, $\vee$. It is also allowed to use the logical negation, $\neg$, to inverse the value of the variable.
Of course, not all logic formulae are directly in CNF. Fortunately, we can transform any formula into a CNF by following precise steps. We will not detail this procedure in this work as it deviates too much from the main topic.

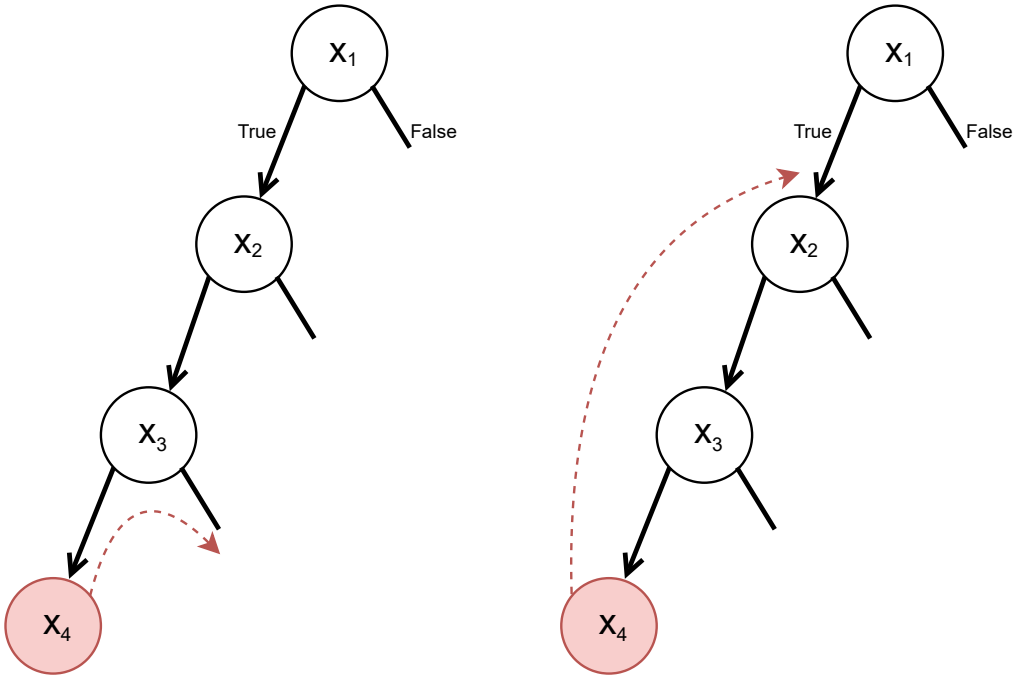
2.3.2 CDCL solver

There are different strategies that can be used inside a SAT solver. We describe the Conflit-Driven Clause Learning (CDCL) [13] as it is the one used in our main scientific paper [1].
CDCL solver is based on Davis-Putnam-Logemann-Loveland (DPLL) solver. Basically, it consists in a backtracking algorithm that explores the possible boolean

assignation for the provided logic formula. It stops in three cases: 1) it finds a satisfiable solution somewhere during the search (the formula is SAT), 2) it finished the search without finding any valid set of values (the formula is UNSAT) or 3) it reaches an imposed time limit.

The main advantage of a CDCL solver over other algorithms of this type is the creation, during the search, of new clauses derived from the encountered conflicts. Typically, after a choice of value for a variable and a propagation to simplify the clauses, it may find a conflict that forces the output to be UNSAT. There, the solver creates a new clause to prevent future computations that would lead to this same conflict, resulting in an acceleration of the search.

The other strength of the CDCL solver, and where it really differs from the DPLL solvers, concerns the backtracking procedure. Instead of simply going back one step to the previous arbitrary value assignation, it will go back to the latest assignment of the variable involved in the conflict. This behavior is called non-chronological backtracking, in opposition to classical, chronological, backtracking.



(a) Chronological backtracking (b) Non-chronological backtracking

Figure 2.5: Illustration of the backtracking types

Chapter 3

Robustness

Now that the basic theoretical notions have been presented, we go more in-depth regarding the robustness. We start by detailing what it is and why it is necessary for neural networks. We will also talk about the adversarial images and their properties, a basic tool to evaluate how robust a model can be. The big part of this chapter will be about the attacks we can perform against the classical deep networks to generate such images and how to modify the training algorithm in order to improve the robustness of the models. Next, we will discuss the attacks but for binarized neural networks as we will see that we can not simply use the same tools.

3.1 Definition and example

As mentioned before, deep learning is able to reach impressive performance, either on classical accuracy or on other metrics. Those results may comfort some people that neural networks are indeed the next big step in the domain of Artificial Intelligence. However, we should not hide recurring problems of this type of architecture. Among many of them, we can cite the lack of explainability, the idea of understanding why a model outputs such value, that makes the neural networks appear like "black boxes". Here, we talk in more detail about another known problem: the robustness, sometimes called the consistency, of the neural networks.

3.1.1 Definition

The robustness of a neural network model can be defined as its ability to resist perturbations applied to the inputs. In this context, resisting means that, for a given input, the final output of the model will stay the same, with the condition

that the disturbance is not too large.

When talking about robustness, most of the literature focuses on treating images, either with the task of assigning a predefined label to it or recognizing patterns inside it. Indeed, we can, in these cases, directly assess if the neural network model has been fooled by the disturbance or if it was able to ignore it and process the image correctly to provide the correct classification.
On the other hand, the case of regression is not so clearly defined. By design, when there is a range of possible output values, it seems logical that a quantification of the error produced by a perturbation of the input is more difficult to obtain.

3.1.2 Application of the robustness

This concept of robustness may not have been obvious directly. Most of the time, the perturbations applied to the input may practically never happen in a real-world context. However, as the concerns about the correctness of neural networks grow with their deployment on everyday tasks, programmers have to ensure their model is sufficiently robust.

A typical concrete example of the need for robust neural networks is the case of autonomous cars [14]. In order to operate correctly in a real environment, those cars need sensors to scan their surroundings. They are then able to process the information and make a decision from the current situation. This decision should only be affected by the important elements such as the pedestrians, the other cars on the road, the traffic signs, ... Concerning this last category, it will certainly be not surprising to learn that those signs do not keep their original aspect, either because of the time and the weather or because of tags made by people. These elements, which we call perturbations to match the vocabulary previously introduced, may fool the system of the autonomous car, leading it to make a wrong decision. One could think of many situations where the outcomes of this situation would be dangerous, like for example a stop sign labeled as a 50 km/h sign.

This basic example has little chance to occur in a real situation thanks to the many precautions taken by the developers behind this type of car. Nevertheless, we believe it is a good way to illustrate and motivate the work presented here.

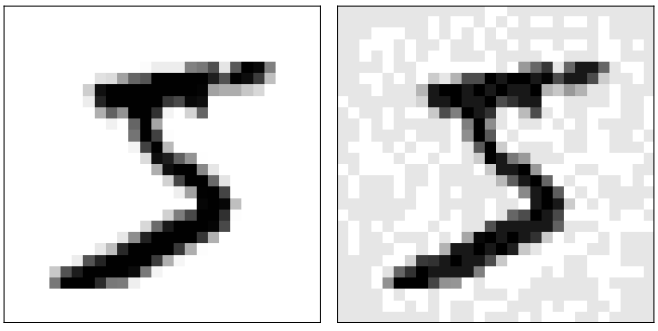
3.2 Adversarial images

3.2.1 Definition

A key component when talking about the robustness of neural networks is the notion of adversarial image (also called adversarial examples or adversaries) [15]. From an original image x, an adversarial image x' is an image close to x with some perturbations that make the deep learning model fail at its classification task.

Ideally, these perturbations should not be visible to the human eye or, at least, a human should still be able to classify correctly the adversarial samples in the case where the noise is visible. Indeed, the noise added may be more visible on black-and-white images than on images with RGB channels. In the usual implementations, the values of the pixels are comprised in the range $[0, 1]$. The additive perturbation we apply to create an adversarial image, commonly called ϵ, has, therefore, the same range. Note however that the addition of this value can not make a pixel get a value higher than 1.
This parameter value is not a constant applied to some pixels of the image but more of a maximal applicable perturbation. For example, if we set $\epsilon = 0.2$, a pixel value may be increased by a maximum of 0.2 but could also be increased only by 0.1 or even not change at all.

In other words, the adversarial image is an image that is tailor-made to fool the model such that it receives an incorrect label. In the literature, researchers sometimes perform a first check to ensure the original image is correctly classified before trying to build an adversarial example from it.



(a) Original image (b) Adversarial image

Figure 3.1: Example of an adversarial image on the MNIST dataset (with $\epsilon = 0.1$)

(a) Original image (b) Adversarial image

Figure 3.2: Example of an adversarial image on the CIFAR10 dataset (with $\epsilon = 0.03$)

We insist on the fact that the process of creating adversarial samples is more than simply adding random noise to an image. From a deep learning model and an image from a selected dataset on which the model has been trained, a dedicated algorithm will search for a precise change of pixel values. These types of algorithms are generally called an attack against the model as their goal is to make the model fail at its classification task.

Of course, using a high perturbation (i.e. using a high ϵ value) will certainly result in a misclassification, but recall that a human should still be able to perform well at this labeling task.

3.2.2 Transferability

Adversarial examples are made with two key elements: an original image and a deep learning model. In general, the perturbation will therefore be different for each image but also, and more importantly, for each possible model architecture. It often goes further than that as it appears that adversarial samples also vary with the training of the model we consider.

Fortunately, adversarial examples show a property of transferability, up to a certain point [16]. Transferability is the ability of an adversarial sample made for one model to remain adversarial when passed to another model.
In general, it is simply measured with the percentage of images that remain adversarial when transferred.

This property may show to be useful when we want to use adversarial examples with many different deep learning models. If these images were specific to the model for which they have been created, one would need to create samples for

each of the models. Thanks to the transferability, we can only generate a certain amount of elements for one model and use them for the other networks as well.

An important point to keep in mind when evaluating the transferability of a set of adversarial images is the value used for the perturbation we apply. Indeed, the higher the ϵ is, the more the images will be transferable to other models due to the increasing difficulty to classify them. If we set a value close to 1 for this parameter, there is a high chance that the image will remain adversarial for all the models. It is also very likely that the time to create it will be very short. However, an image that is too disrupted has no interest in the classification task.
Therefore, one has to find the right balance between the perturbation value and the expected transferability to avoid the generation of adversarial images for each possible model.

3.3 Evaluation

We are now interested in the existing possibilities to evaluate the robustness of a neural network.
In machine learning in general, we often want to obtain some insights into the performance of the model we are training. We can simply look at its accuracy on a test set or other metrics of the same kind (recall, loss, ...) to obtain such information.
When it comes to robustness, it seems clear in the literature that there is no consensus on how to perform such a measure.

3.3.1 Adversarial accuracy

The most straightforward metric, used in most of the publications that aim to assess the robustness of their deep learning model, is the adversarial accuracy [17]. As its name indicates, we evaluate the model based on its accuracy on adversarial images, separated from the training set to avoid overfitting.

$$\text{adversarial accuracy} = \frac{TP + TN}{n_{adv}} \tag{3.1}$$

where n_{adv} represents the number of adversarial images we use in the adversarial test set and TP, TN are respectively the number of true positive and true negative samples classified by the model.
This method of evaluation is actually quite convenient as it allows a direct comparison with the original accuracy i.e. the accuracy on unmodified images.

3.3.2 Attacks and other evaluations

Another way to obtain an evaluation of the robustness of a model is to attack the network. In other words, researchers seek techniques that are able to create images to make the model fail. Therefore, the numerical value for the robustness directly depends on the number of adversarial images found with the considered attack. One of the most important notions when creating adversarial images is the L_p distance:

$$||x - x'||_p \tag{3.2}$$

where $||\cdot||_p$ is the p-norm defined here:

$$||x||_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}} \tag{3.3}$$

Many attacks that aim to create an adversarial example exist [18]. In the literature, a difference is often made between targeted and non-targeted attacks. The first type is an attack that aims to produce an adversarial example with a specific label when classified by the model. On the opposite, a non-targeted attack simply produces an adversarial image whose label, when given to a model, is different from its original one.

Chronologically, one of the first attacks that generates such images is the L-BFGS [15]. The idea here is to find an image x' close to the original image $x \in \mathbb{R}^m$ but which is classified differently (i.e. an adversarial image). The results should satisfy the following problem (using the formulation from Carlini et al. [18])

$$\begin{aligned} &\text{minimize } c\,||x' - x||_2 + loss_{F,l}(x') \\ &\text{such that } x' \in [0,1]^m \end{aligned} \tag{3.4}$$

where $loss_{F,l}$ is a function that maps an image to a positive real number, such as the cross-entropy. We solve this problem for different values of $c > 0$, using bisection search to update it, in order to find an adversarial image x' with a minimum distance from the original image.

The most popular attack against networks is probably the Fast Gradient Sign Method (FGSM or simply FGS), first presented by Goodfellow, Shlens and Szegedy [19]. The idea here is to use direct information from the gradient, and more precisely from its sign, to obtain a perturbation that will allow us to create adversarial samples.

Basically, when given an image x with label y during the training of the model, the backpropagation algorithm will compute the gradient of the loss function $J(\theta, x, y)$ where θ represents the current parameters of the model. From the gradient value, we can obtain:

$$\eta = \epsilon \, \text{sign}(\nabla_x J(\theta, x, y)) \tag{3.5}$$

where ϵ has similar behavior to the cases previously discussed, acting on the perturbation value.
Finally, this η value can be added to the original image x in order to produce an adversarial example.

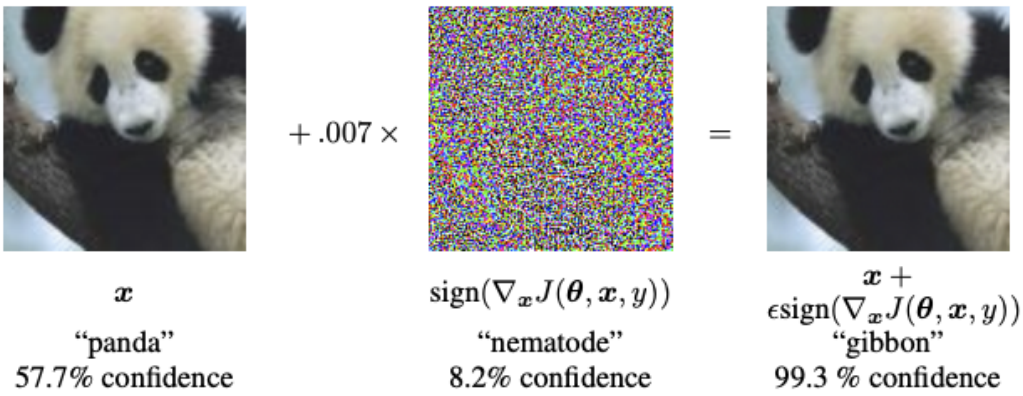


Figure 3.3: Example of the use of FGS to generate an adversarial image from the dataset ImageNet. The ϵ used in this specific case is 0.007. The image is taken from the original publication presenting the FGS method [19]

FGS is called a single-step attack. This denomination comes from the fact that we only compute the perturbation once and then add it to the original image.

Another approach, based on L-BFGS, is DeepFool [20]. It also uses the L_2 norm to find an image close to the original. Its main advantage is to consider any classifier as a linear classifier, with the help of approximations to define linear boundaries in the domain space when needed. According to their results, this approach leads to adversarial images in less time and with fewer perturbations than in the original case of L-BFGS.

Many other attacks against deep networks exist but listing and explaining them all would require work entirely dedicated to this task.

Furthermore, the attacks we have described here have direct access to the deep network. They are able to understand the structure of the model as well as read its weights. It is said that they are therefore working in a white-box manner. Black-box attacks also exist but rely more on transferability instead of constructing an adversarial example. However, approaches in this direction are also developed with competitive results compared to white-box attacks [21, 22].

Given all those different attacks, another possible way to evaluate the robustness of a model is to assess how well it can resist them. For example, a model that can resist an attack up to a certain amount of perturbation could be considered partially robust. Another model could be said to be more robust than this previous one when it is submitted to the same kind of attack but is able to resist higher perturbation values.

3.4 Adversarial training for standard neural networks

In the last chapter, we have seen how a deep neural network can learn a task with the help of backpropagation. We have also just discussed the importance of the robustness of those networks and talked about algorithms that can create images able to fool them. The next important question is then: how can we train a deep learning model while ensuring that it is robust to adversarial inputs?

We do not expect a model to be totally resistant to all types of attacks. The true goal of adversarial training, which means training a model in order to make it more robust, is to reduce the number of adversarial examples or, at least, prevent the attacks from finding them easily.
We will present here different ways to perform such training for classical neural networks, all of them having pros and cons as this task represents a big challenge in deep learning.

3.4.1 Important factors

First of all, any deep learning model should be able to become more robust when adversarial training is applied. However, Rozsa et al. [16] showed that better results for this type of training will be obtained when the basic model already performs well in terms of basic accuracy.
They evaluated the robustness not as a direct value but as a relative measure between their tested models. A model that requires a higher perturbation in terms

of L_2 or L_∞ norms to be fooled is considered more robust to adversarial examples than another model with a lower perturbation value. Here, the norms were used to create adversarial examples with FGS and Fast Gradient Value (FGV). This last attack is a simple extension of FGS, where the algorithm uses a scaled version of the gradient of the loss to produce the adversarial example instead of a step defined by the sign of this gradient. As those norms may not match the human perception (which is why L_1 is preferred in general), they also use the PASS score, a metric used to assess how adversarial an image is, to evaluate their results.

Their conclusion was that best-performing models for the accuracy can reach much better robustness. They also noticed that adversarial examples produced with these networks showed better transferability than the ones based on worse models.

Another important factor when it comes to improving robustness is the architecture of the models, as shown by the work of Madry et al. [17]. It seems logical that the model capacity, which functions it is able to represent, is highly linked to its robustness and its ability to handle correctly adversarial samples. Indeed, the more a network can adapt during training (adversarial or not), the more out-of-distribution samples it may handle.

To verify this property, the authors selected a convolutional neural network and doubled its size (the number of convolutional filters and the size of the fully connected layers next) while testing it against an FGS attack. In addition to the capacity conclusion, they also mention that small models may fail to adapt to adversarial images or has to choose between adversarial or original accuracy.
However, as in the work of Rozsa, the transferability is also impacted. Better results on high-capacity networks lead to adversarial images that are less transferable to other models.

3.4.2 Training modification

As we will see, many of the adversarial training algorithms use adversarial image generation at some point. They are therefore close to the attacks we have previously talked about but, in this case, the produced data is directly used in the training.

A noticeable element when it comes to the adversarial training part is the loss landscape corresponding to the optimization problem [17]. Indeed, many solutions implementing a type of this specific way of learning algorithm will define their own loss function in order to take into account the presence of adversarial examples. They also consider only one type of attack, choosing in general the FGS as it is

considered the best option in terms of results and time consumption.

For most of the loss functions, their landscapes show a very interesting property: most of the local optimal points have very close values, even if they are widely spread in the domain. This conclusion has been made by training a model against adversaries created with FGS or Projected Gradient Descent (PGD), which is a multi-step variant of FGS.

Another remark that may appear to be obvious to many is that learning adversarial examples changes a lot the original decision boundary from a model. Even if the universal approximation theorem states that a neural network with only one hidden layer could model any function, the relation to find during adversarial training may be too complicated for smaller models. It correlates with the importance of the architecture of the model that we talked about in the last section.

The most classical way to do an adversarial training is to incorporate adversarial examples into the training set and use a modified loss function [19]:

$$\tilde{J}(\theta, x, y) = \alpha\, J(\theta, x, y) + (1 - \alpha)\, J(\theta, x + \epsilon\, \text{sign}(\nabla_x J(\theta, x, y)), y) \qquad (3.6)$$

The use of a modified loss function allows us to indicate to the model that there are adversarial examples in the dataset. This particular function is crafted to be used with FGS, as the second term indicates it. The parameter $\alpha \in [0, 1]$ has to be set by the user to prefer the original or the adversarial examples. As an indication, the authors of the original publication used 0.5.
When using this basic mixture approach, adversarial images are constantly generated during the training. It ensures that those images are adversarial to the current version of the model, as it may change very quickly during this learning phase, making the previously generated examples obsolete.

In all machine learning training, there is always a risk of overfitting. The adversarial training is not an exception, as explained by Tramèr et al. [23]. According to them, using a single-step attack such as FGS for this type of learning phase presents degenerate optima leading to overfitting.

To solve this issue, a new type of adversarial training is suggested: ensemble adversarial training. The idea behind it is that the overfitting appears because the adversaries are adapted to the current state of the model during its learning phase. The approach solves this problem by decoupling the generation from the trained model by using another pre-trained network to generate those examples. In other words, adversarial examples are generated with another model and then

transferred to the model of interest during its training phase.

Another failure point is identified as "catastrophic overfitting" by Wong et al. [24]. This kind of overfitting is characterized by a huge decrease of the adversarial accuracy (down to 0%) when using a PGD attack. A simple way to avoid this problem is to use some early stopping of the learning phase according to the current performance.

Training a model with a mixture of original and adversarial examples to make it robust is the simplest but most efficient approach. In this direction, researchers have tried to develop new techniques to improve both the training time needed and the quality of the produced adversaries.
Free adversarial training [25] is one of the most efficient ways to perform such learning phase. Adversaries are directly produced during the training of the model based on a PGD attack. However, in order to speed up the generation and to produce strong adversarial examples, the model is trained on the minibatch of data m times in a row. This major change of the learning algorithm also allows us to use the perturbation obtained from the previous minibatch as an initialization for the current one. Note that in order to keep the same number of total iterations, the authors have divided the number of epochs by m.

The free adversarial training is significantly faster than training with PGD but still not as fast as we want. Therefore, Wong et al. [24] proposed a return to the classical FGS with some modifications known as fast adversarial training.

Algorithm 1 Fast adversarial training with FGS for T epochs, given a radius of perturbation ϵ, step size α, a loss function J and a dataset of size M for a model f_θ. Taken from the previously cited publication [24].

for $t = 1...T$ **do**
  for $i = 1...M$ **do**
    // perform FGS attack
    $\delta = \text{Uniform}(-\epsilon, \epsilon)$
    $\delta = \delta + \alpha \cdot \text{sign}(\nabla_\delta J(\theta, x_i + \delta, y_i))$
    $\delta = \max(\min(\delta, \epsilon), -\epsilon)$
    // update weight of the model
    $\theta = \theta - \nabla_\theta J(\theta, x_i + \delta, y_i)$
  end for
end for

Their idea is to use FGS with random initialization for the perturbation. Compared to the free approach, it does not need to train m times in a row on the same

minibatch but only to compute two times the gradient, once for the perturbation and once for the classical update of the model parameters.

We have seen different ways to perform an adversarial training. However, comparison between them is often a difficult task as they are not trained with the same learning algorithm, against the same attack or not even with the same loss function.
In the publication, the most used metrics to compare the approaches are mainly the time required to perform the training and the adversarial accuracy obtained in the end. However, as all the approaches do not target this specific metric, the results may not always be that easily compared.

3.4.3 Alternatives to adversarial training

In all the algorithms and modifications of classical training of deep learning models we have talked about, adversarial images are a mandatory element in order to improve robustness.
However, obtaining such images through the attacks we have described before (and many others) can be very time-consuming. Even if approaches tend to lower this factor such as the free or fast adversarial training, we still have to produce the images at a given time based on information on the model.

To address this problem, scientists have also explored other ways to reach robustness. A first approach from researchers of the Alibaba group [26] is to get inspiration from the Generative Adversarial Networks (GANs) [27]. We have here two networks: the victim and the attacker. For a fixed victim network, the goal of the attacker will be to generate, based on an original image, a sample that can fool the victim network.

In this particular case, the adversarial example is therefore not built from the final model but instead sampled from a latent distribution that is learned by another model. Unfortunately, even if this approach can save a certain amount of time for the generation part, it may prove to be inefficient for complex datasets, such as ImageNet. Shafahi et al. [25] suggest that this phenomenon is due to the difficulty of GANs to cover the whole domain of such datasets.

Another way to avoid computationally intensive work for adversarial image generation is to regularize the training by using label smoothing or logit squeezing, as suggested by Shafahi and his team [28]. Their goal is to mimic the effects of adversarial training with the help of those two elements.

Label smoothing works by transforming one-hot label vector, a 1 for the correct label and 0 for all the others, into one-warm vector following the formula

$$y_{warm} = y_{hot} - \alpha \left(y_{hot} - \frac{1}{N_c} \right) \tag{3.7}$$

where $\alpha \in [0, 1]$ is the smoothing parameter to be set by the user and N_c is the number of possible classes. Such a tool is also used in classical training algorithms to reduce the risks of overfitting. Concerning the robustness, it allows us to obtain smaller logit gaps, where the logit is the name used for the output value of a neural network on which we may still apply an activation function.
In addition, we can apply logit squeezing which simply consists of crushing the logits by penalizing too large output value during the training.

These two examples of processing are used to reach similar results to the ones obtained with adversarial training, according to the experiments realized by the authors. However, they have not yet been tested on large and complex problems. In theory, nothing prevents us to use them in addition to adversarial training, even if the observed logits should be similar in the end.

3.5 Binarized model verification

We have explained how attacks and adversarial training work for classical neural networks. Even if we are more interested in the binarized version of such models, most of the work was done for these classical architectures which is why we first focused on them.
In this section, we talk about what exists for the BNNs and how we can perform attacks against them.

3.5.1 Classical attacks

Model verification is the assessment of well how the model performs against an attack, in our case an adversarial attack. Therefore, in this section, we are only interested in the methods to attack models and not the training part.

As with any other deep learning model, the necessity of computing the gradients during the learning phase allows us to use its value to create attacks. Therefore, the FGS method explained in section 3.3.2 could be used with BNNs as well.
The improved, but more costly version, the projected gradient descent, could also be applied since the only difference is that it uses FGS several times to compute better adversarial examples [29, 30].

We will see later however that these techniques may not be as efficient as with classical networks.

Another attack that we did not talk about before and that is used against BNNs in Galloway's work [29] is the Carlini-Walter attack [18]. They specifically used the L_2 attack which is an iterative process using an optimizer such as Adam in order to produce strong adversarial images.
This attack works directly on the logits produced by the network. It also has the possibility to tune a parameter to encourage the attack to produce an adversary with a specific label, similar to a targeted attack but without the strict condition that the image must have the desired label.
Its other strength is that it can be restarted several times from different starting points in order to avoid getting stuck in global minima.

3.5.2 Problems of classical attacks

These attacks have proven to be quite effective against the standard neural networks, with floating point values for the weights and well-known activation functions.

However, we recall one of the specificities of the binarized neural networks: it uses a sign activation function for its hidden layers in order to produce binarized weights. This function is a mandatory part of the BNNs and can therefore not be replaced by any other one at inference time.
The problem arising with this function is that any gradient-based attack will be unsuitable for the BNNs, as stated by Khalil and his team [31]. Indeed, we have discussed in section 2.2.1 the need for an estimator for the activation function at training time. The attacks will then use the information from this estimator in order to produce adversarial examples.

In their publication, the authors have shown that this gradient used by FGS, PGD or other attacks may not give the correct ascent direction, sometimes even going in the opposite direction, thus leading to incorrect adversarial examples.
This problem motivates the need for another approach, more specific to binarized neural networks and that does not use the gradient directly.

3.5.3 BNN encoding

We have said before that BNNs present several advantages when compared to classical neural networks such as the energy consumption or the ability to run on edge devices such as smartphones.

When it comes to the model verification part, binarized neural networks present another major advantage due to their specific architecture: the ability to be encoded in a discrete modeling language, such as a set of formulae. Once the model is encoded in formulae, a solver for this type of encoding can be used in order to verify a property. Typically, we can use it to obtain adversarial examples.

A first idea, which comes from classical neural networks, is to use the Satisfiability Modula Theory (SMT) [32]. Here, each node value of the network is considered as a variable and the verification query, i.e. the query to create an adversarial example, will act as a set of constraints on these variables.
We distinguish two kinds of constraints: the linear constraints, which encode the lower and upper bounds for the variables and the equations that represent the model, and the piecewise-linear constraints, used to represent the activation function.

The next approach that is used in the literature to encode BNNs is Mixed Integer Linear Programming (MILP) [31, 33]. Each component of the blocks forming the BNNs, the weighted sum, the batch normalization and the sign activation function, are translated into Boolean formula separately. The output layer has its own encoding as we also need to encode the notion of getting the label corresponding to the highest prediction. This encoding works with a set of linear constraints that define the BNN we give as input.

MILP encoding can also be used for classical neural networks as it still uses real-valued variables. The Integer Linear Programming (ILP) approach aims to reduce the usage of such values by using the properties of BNNs. As we know that the weights and activation values may only take two different values, the ILP encoding will start from the MILP, remove variables and simplify constraints by getting rid of the real-valued variables.

The ILP encoding contains therefore fewer elements to solve than the MILP one. However, both approaches take a significant amount of time to be solved. Since the main task here is to solve these encodings with the help of an adequate solver in order to produce adversarial images, we look at a final idea with SAT encoding.

This type of encoding starts from the ILP formulation and transforms every node of the network into a constraint, along with its input. Such constraint is usually called the cardinality constraint.
To encode the binary blocks (weighted sum, batch normalization and binarization) of the network, we first have to notice that all the constraints will be of the forms

$$\langle a_i, x_k \rangle \geq C_i \Rightarrow v_i = 1 \tag{3.8}$$

where a_i represents the activation values, x_k is the input to the binary block and C_i a constant. Note that both of them are therefore binary values. In the case where the inequality is not respected, the resulting Boolean variable v_i will be set to 0 or -1, depending on the selected BNN implementation. Here the usage of 0 instead of the classical -1 becomes interesting as it allows a direct translation from the value of the variable to the SAT formula.
All of this encoding has been presented for the first time in the work of Narodytska and her team [33].

Jia and Rinard started their work from this SAT encoding and synthesized it into the reified cardinality constraint [1]:

$$y_i = \sum_{j=0}^{m} l_{ij}(x_j) \gtreqless \left[b_i(k^{BN}, W, b^{BN}) \right] \tag{3.9}$$

where x_j is the input to the binary block and y_i is its output. The variables $l_{ij}(x)$ take the value $0, x$, or $\neg x$ depending on the binarized weights W_{ij}. The constant $b_i(k^{BN}, W, b^{BN})$ is derived from the binarized weights and from k^{BN} and b^{BN}, the constants defining the batch normalization layer from this binary block. Concerning the symbols, $[\cdot]$ is a flooring or ceiling operation and the $\gtreqless$ operation is either a $\geq$ or $\leq$. It outputs either a 0 or a 1 as their implementation uses directly 0 and not -1. Both of these symbols depend on the sign of k^{BN}.
Note that the variables are still the same as before, i.e. one variable for each node of the network. The reified cardinality constraints only encode the relations between them.

This formula allows an exact encoding of the BNN. In other words, properties could be searched in the deep learning domain or in the SAT domain equivalently. In this case, it is highly interesting as it enables the use of powerful and efficient SAT solvers.

3.6 MiniSatCS

In the last section, we have presented several encodings that can be used with BNNs. From now on, we will focus on the last one which uses the reified cardinality constraints and comes from the work of Kai Jia and Martin Rinard [1].
The next step is to use this SAT encoding with a SAT solver to verify the model on a given image, i.e. generating an adversarial image.

In our case, we will use the MiniSatCS solver introduced in the publication *Efficient and Exact Verification of Binarized Neural Networks* [1].
At its core, it is a CDCL-based SAT solver, explained in the section 2.3.2. It extends the well-known MiniSat solver and, according to the experiments made by the authors, it has proven to be much more efficient than other SAT solvers. Its first task here is to convert the reified cardinality constraints into disjunctive clauses to obtain a CNF form of the SAT formula.

Now, given an image as input and a binarized model, the solver encodes the model and uses the given image to search for perturbation leading to misclassification. Each pixel of the input is encoded as a Boolean variable on which we can add a value, up to a certain maximum ϵ. In practice, we actually encode each possible interval of value the input can take. It is done thanks to the quantization of the input. The solver has three different outputs:

- SAT: the formula is satisfiable which means it was able to find an adversarial example with the input. We can therefore retrieve the perturbation found to build the adversary.
- UNSAT: the formula is unsatisfiable which means that, after all the possible Boolean assignments, none of them lead to a misclassification by the model. In this case, we say that the model is robust for the input image.
- TLE: for Time Limit Exceeded. The solver did not find any adversarial example in the allowed amount of time. Whether it can be considered robust or not for this input is left to the user.

3.7 Conclusion

We have seen that the robustness can be defined and evaluated in several ways. Most of the works use adversarial images as the basic tool for this characteristic. We have seen the most known attacks for classical neural networks such as FGS or PGD and the different methods to encode BNNs and verify them afterwards. Both of them can give us adversarial images that we can use for adversarial training. However, one of the main messages to remember is that standard attacks from real-valued models may not be adapted to BNNs, hence the need for encoding and verification.

Chapter 4

Proposed approach

Previously, we have explained the concepts behind binarized neural networks and their possible encoding into sets of formulae to perform verification. We have also talked about the concept of adversarial training that aims to improve the robustness of deep learning models. In this chapter, we start by detailing the methods we will consider to evaluate the robustness. Next, we give the parameters we will use during the experiences with the MiniSatCS solver. Finally, we explain our proposed algorithm called the robust training loop, with the motivation behind it and its possible variants.

4.1 Robustness evaluation

As our work will be focused on how to improve the robustness of binarized neural networks, we need tools to measure it. The most common one is the adversarial accuracy, presented in section 3.3.1.
Its strengths are that it is both simple to implement but also easy to interpret and to compare with the accuracy on original images. Most of the publications mentioned before use this metric when it comes to the evaluation of the attack or adversarial training.

We introduce in this section three new and original metrics that we have created during the realization of this work.

4.1.1 Absolute robust score

The idea behind the absolute robust score is to use the probability values p at the output of the model instead of the classification label alone. Those probabilities are

obtained by applying a softmax function on the logits we obtain from the network. The metric is defined as:

$$\text{absolute robust score } = \frac{1}{n} \sum_{i=0}^{n} \Big(1 - |p_i(l_i^{truth}) - \max(p_i)|\Big) \tag{4.1}$$

where n is the number of samples we have to classify and $p_i(l_i^{truth})$ is the probability related to the true label of the sample i. Therefore, $max(p_i)$ is the highest probability given by the model for the input sample i, i.e. the probability that is used to decide the predicted label.

4.1.2 Robust ratio evaluation

As a follow-up on the idea to use the probabilities, we propose now the robust ratio evaluation that we define as:

$$\text{robust ratio } = \frac{1}{n} \sum_{i=0}^{n} \frac{\max(p_i)}{1 - p_i(l_i^{truth})} \tag{4.2}$$

The goal behind this metric is to assess two things: how the logits are impacted by adversarial training but also how the probability related to the truth label evolves when compared to the probability used for the prediction. The complementary is used here to increase the impact of the errors. If the prediction is wrong, then $max(p_i) > p_i(l_i^{truth})$. Our goal is to obtain the largest value possible for the model.

4.1.3 Inverse robust ratio evaluation

The next metric we designed is directly inspired by the robust ratio evaluation, as indicated by its name. The inverse robust ratio is defined by the formula:

$$\text{reverse robust ratio } = \frac{1}{n} \sum_{i=0}^{n} \frac{p_i(l_i^{truth})}{1 - \max(p_i)} \tag{4.3}$$

The idea here is the same as for the previous metric. We wanted to evaluate both cases to be rigorous.

4.1.4 Notable comments

We expect that the two last metrics behave similarly. Indeed, if the robustness is improved, the predictions made by the model should become more and more accurate. The resulting effect is that we have an increasing number of cases where $p_i(l_i^{truth}) = \max(p_i)$. Therefore, both metrics will tend to be the same when the

model is robust enough.

The interesting aspect is also that they can all be applied to original images. As for the accuracy, we will then be able to compare the original score and the adversarial score the model obtains.

An important point on which we insist is that those metrics are simply different ways to evaluate the output of the model. They do not modify the training in any case.
We proposed them here as only a few alternatives to adversarial accuracy are proposed in the literature.

4.2 Parameters for MiniSatCS

We have presented the MiniSatCS solver in section 3.6. In this work, it will be our main tool to generate the adversarial examples that are needed to evaluate and improve the robustness of our binarized models.

We mentioned earlier that this solver uses a timer to avoid an extremely long search for the adversary. In our case, we will use a time-out value of 90 seconds when performing the search for an image and a model.
The other important parameter is the maximal perturbation value the solver can use: $\epsilon \in [0, 1]$. This will always be indicated in the results of our experiments.

We have presented the three cases it can output: SAT, UNSAT and TLE. Throughout the remaining parts of this work, we will mainly focus on the number of adversarial images found, i.e. the SAT case. This means that we implicitly consider the TLE cases as robust for the model under study.

4.3 Robust training loop

We describe now the important aspect of this work and our main contribution to the domain of robustness and adversarial training concerning binarized neural networks.

4.3.1 Motivations

The goal behind the robust training loop is to propose and explore another kind of adversarial training, one where we do not generate the data during the computation

of the gradient such as FGS or PGD. We believe this may allow other people to generalize it to other attacks against fully trained binarized or classical neural networks.

This generalization would be possible as the core of the training algorithm is not modified. Indeed, as we will describe in the next subsection, it mainly consists in adding an external loop to the training algorithm. This presents several advantages such as being able to run it for models for which we do not have the implementation or even for different attacks and not simply the one we used in our work.

We also wanted to develop an approach that would allow us to assess the iterative improvements of the model regarding robustness. As we are adding more and more data, we think watching the impact of the augmentation is also an important element to assess. This iteration process with an evaluation of the robustness at each epoch also allows the user to stop the training according to a certain threshold they may define. For example, we could stop as soon as we have reached 70% of adversarial accuracy, saving a large amount of time by not performing additional epochs. Of course, the usage of this algorithm should not alter too much the accuracy obtained with the original data.

One could question the use of BNNs in this algorithm. They are of course not mandatory for its realization but their faster training time and convergence help to reduce the total amount of time needed. Obviously, it is also related to the attack we are studying here which uses the MiniSatCS solver that works only for this type of architecture.

4.3.2 Principle

The robust training loop is a simple modification of the classical training loop. Its aim is to improve the robustness of a given binarized neural network.
Instead of generating adversarial examples during the training itself, we train our model classically on the training set with an evaluation using the usual test set. Then, once this usual learning part is done, we use the obtained model to generate adversaries on it and use them to augment the training set. An adversarial testing set, generated at the same time as the training one or taken from an external source, is used to evaluate the performance of the model with a given metric.

At this point, we have a fully trained BNN, a dataset containing a mixture of original and adversarial samples and an evaluation of the model performance regarding the robustness. The idea is to relaunch the training of the model, with or without recreating it, with this augmented train set. The process is similar to the multiple epochs we use in the classical training algorithm but focused on data

generation with the specific goal of improving the consistency of the model being trained.

Algorithm 2 Robust training loop algorithm. The *train_set* and *test_set* are obtained from publicly available datasets. *adv_test_set* (for adversarial test set) is loaded from a previous generation of adversarial data corresponding to the *test_set*.

for $i = 0...n_{robust}$ **do**
  $model \leftarrow model\_init()$
  for $j = 0...n_{epochs}$ **do**
    $train(model)$
    $test(model)$
  end for
  $adv\_score \leftarrow evaluate\_robustness(model)$
  $adv\_images \leftarrow generate\_adv\_images(train\_set, \epsilon)$
  $train\_set \leftarrow train\_set + adv\_images$
end for

This algorithm contains elements that require more explanation. The n_{epochs} is the usual number of epochs used to train a deep learning model while n_{robust} is the number of times we run the robust training loop. We will refer to them as robust epochs in opposition to the classical training epochs.

The function $evaluate\_robustness(model)$ symbolizes the use of one of the previously introduced metrics to have an idea of the robustness of the model. Thanks to the use of a loop, we will be able to visualize its evolution through robust epochs.

Before augmenting the training set, represented here by the last line of the algorithm, we first need to generate adversarial images. It is important that the data here are generated on the train set only and not on the test set to avoid passing any information to the model about the images used to evaluate it.
As explained before, we need to set a maximal perturbation value for the attack, represented here as before by ϵ.
When we add these new data to the image set, we shuffle all the train set in order to avoid any unwanted effect during the training. Note that we do not replace the previously generated adversarial data with the new ones. At each robust epoch, we add a certain amount of new adversaries to the mixture of original and adversarial elements composing the train set.

Concerning the test set, we can generate the corresponding images at this moment. However, in order to save time, we prefer to use a previously generated adversarial test set obtained from the same attack.

4.3.3 Description of the different approaches

Throughout all our experiments, we are going to use two distinct approaches for the robust training loop: the adversarial sampling approach and the generation approach.

They both act on the *generate_adv_images* operation. The generation approach sticks to the behavior we have described before. Given the model we have just trained, it uses the MiniSatCS solver to encode the BNN and each image we give it to generate adversarial images. It has the advantage to be specific to the model we are currently training but it may take a large amount of time as it should become more and more difficult to generate adversarial examples through the robust epochs.

On the other hand, the adversarial sampling approach is developed to be an approximation of the generation variant. Its goal is to drastically reduce the time needed for the robust training loop by using precomputed adversarial examples. Thus, instead of using the MiniSatCS solver in the algorithm, we use it beforehand to obtain a large set of adversarial images that can be sampled to feed the training set of the algorithm.
Note that, since we used pre-generated images, we could directly use all the adversarial images in the training if we recreate the model at each robust epoch. However, in order to stick to the approximation idea, we will use it constantly like the generation approach which means adding a set of adversarial images at each robust epoch.

This second approach is actually highly based on the transferability property of adversarial examples. With a higher value of ϵ, we have more chances to obtain results close to the generation approach as the images will be more transferable. Another advantage is that we avoid the risk of overfitting that we discussed in section 3.4.2.

There is actually one major difference between these two approaches that need to be mentioned. The more robust epochs we have, the less the adversarial images produced by the generation approach will look like the ones used in the adversarial sampling approach. Even if the differences may not be visible to the human eye, these images are not built on the same information of the model. In one case, the adversarial sampling one, all the images are generated on the same model with no adversarial training. In the other case, the images are produced on models that are more and more robust.
The differences should not be too notable to directly reject one of the two approaches. Note however that both approaches will be evaluated on a test set generated the same way as for the images used in the adversarial sampling approach.

4.3.4 Extensions

The robust training loop described before is the basic version of the algorithm. During this work, we will consider improvements that are described here.

First of all, the MiniSatCS has the possibility to handle multiple models at the same time. It means that if we give it three different models and an image as input, the obtained adversarial image is adversarial for all of the three models. It allows to produce stronger adversaries, which is especially useful for transferability and therefore for the adversarial sampling approach.

Since we can use several models in the verification, we can also train multiple models to perform classification. We will then consider three cases: one where we only use a single model, one where we use five models and compute the mean performance and a final case where we use majority voting with five models. The setup with the mean is used here to reduce random and unwanted effects that could appear during the training of the single model.

Chapter 5

Experimental setup

In this chapter, we present the different elements that will be used to perform our experiments. We introduce first well-known datasets that we use to assess the performance and robustness of our binarized neural networks.
Then, we detail the BNN architectures we wrote and which are used throughout all our experiments. This point is important as we will refer to them as "model of type x" in the later chapters.
Finally, we mention briefly the dedicated platform we use to run our tests.

5.1 Datasets

To train, test and perform adversarial attacks against our BNNs, we use different well-known datasets: MNIST [34], Fashion-MNIST [35] and CIFAR-10 [36].

In the experiments, we will mainly use MNIST to perform various tests. The other datasets will be used to validate the results on images with the same shape (Fashion-MNIST) and colored images (CIFAR-10). Examples of images taken from these datasets are available in the appendix A.

5.1.1 MNIST dataset

The MNIST dataset is one of the most used in machine learning tasks. It is composed of $60,000$ images for training and $10,000$ images for testing. These images have a shape of 28x28 pixels. They are typically represented in black and white as they contain only one channel.

The dataset represents the ten digits and therefore contains ten classification labels, from 0 to 9. The digits are handwritten and their distribution among the train

and test sets is uniform.

5.1.2 Fashion-MNIST dataset

The Fashion-MNIST is very close to the MNIST dataset. It has the same image format, 28x28 pixels with one channel, as its goal is to be a direct replacement for the dataset with digits. It is composed of $60,000$ train images and $10,000$ test images, also distributed uniformly for both of them.

In this case, the data represent various clothes, with the following possible labels: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot.

5.1.3 CIFAR-10 dataset

The final dataset we consider is the CIFAR-10 set. This last one largely varies from the two others as its images have a shape of 32x32 pixels with three channels, red, blue and green, which means the images have colors.
Another difference with the MNIST variations is that it contains $50,000$ training samples and $10,000$ testing samples. Once again, the images are distributed uniformly in both sets.

The images represent different and various things in common life. The classes are: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship and Truck.

5.2 Binarized Neural Network models

During our experiments, we will use four different architectures of binarized neural networks that we describe here. Each of them is represented by a number, from 1 to 4, and we will use it to refer to the model in the next chapters.

We will mostly use the models 1 and 4 as they are the two most interesting models due to their size. Models 2 and 3 are here to assess how the fixed architecture can act on the robustness measures we will consider.

We recall that the BNN architecture requires what we call a binary block. It is made of a linear layer followed by a batch normalization layer and a sign activation function. In the following descriptions, we use the term of a binary block of size x. It simply refers to a binary block with a linear layer made of x units followed by the two other layers we have just mentioned.

5.2.1 BNN type 1

The first model we consider is a multi-layer perceptron with the following layers:

- An input quantizer with a quantization value of 0.1 and a flatten operation
- 2 binary blocks of size 512
- 1 binary block of size 100
- A linear output layer with 10 units followed by a batch normalization layer

The output layer will always be made of 10 units as all the datasets we consider have 10 different classes. The final batch normalization is a simple addition from Jia's work [1], which is supposed to improve performance.

This type of model has $1,433,763$ parameters for MNIST and Fashion-MNIST and $3,776,675$ parameters for CIFAR-10.

5.2.2 BNN type 2

The second model is also a MLP, but much smaller:

- An input quantizer with a quantization value of 0.1 and a flatten operation
- 2 binary blocks of size 50
- A linear output layer with 10 units followed by a batch normalization layer

This type of model has $84,611$ parameters for MNIST and Fashion-MNIST and $313,411$ parameters for CIFAR-10. It is the smallest model we will consider.

5.2.3 BNN type 3

The third model is a MLP of medium size, between the types 1 and 2. It is made of:

- An input quantizer with a quantization value of 0.1 and a flatten operation
- 3 binary blocks with sizes $50, 100, 200$, in this order
- A linear output layer with 10 units followed by a batch normalization layer

This type of model has $267,711$ parameters for MNIST and Fashion-MNIST and $496,511$ parameters for CIFAR-10.

5.2.4 BNN type 4

The fourth and final model is a CNN. Here, the information for the binary blocks refers to the parameters of the convolutional layer. The model is made of:

- An input quantizer with a quantization value of 0.1
- 1 binary block which outputs 32 channels and uses a kernel size of 4, a stride of 2 and a padding of 1
- 1 binary block which outputs 64 channels and uses the same parameter values as the one before
- A flatten operation
- 2 binary blocks with linear layers of size 512 and 256 respectively
- A linear output layer with 10 units followed by a batch normalization layer

This type of model has $3,546,867$ parameters for MNIST and Fashion-MNIST and $4,531,915$ parameters for CIFAR-10. It is the largest model we will consider.

Concerning the parameters, the kernel size represents the size, in pixels, of the window we apply to the image. The stride is the number of pixels the kernel is moved by for each of its computations. The padding is the number of additional pixels we consider around the image.

5.3 Usage of dedicated platform

To run our experiments, we will use the "Consortium des Equipements de Calcul Intensif" (CECI). It allows us to run multiple tests at the same time with similar hardware. We mainly used the Dragon1 cluster with 4 CPUs and 6GB of memory.

Chapter 6

Results and analysis

With all the previous explanations in mind, we share now the results we have obtained through different experiments. The first part concerns the basic scores of our models. Are the BNNs able to reach high accuracy for the original samples? Next, we will talk about the perturbation applied to create adversarial images. Does it follow a pattern? Is it the same as random noise? Concerning the adversarial images, is there a significant amount of them that is transferable to the other models?
An important focus will be made on the comparison between the adversarial sampling and the generation approaches for the robust training loop. Is one of them better than the other to improve the robustness of a BNN? Is a combination of models more interesting than a single model? We will also perform smaller experiments on the other robustness metrics to see if one of them can provide interesting information. Additionally, how do the results compare to a case with noisy images instead of adversarial ones? Can the robust training loop be applied to a non-binarized neural network and still provide similar performance? Finally, we will also investigate another way to assess the robustness of the models trained with the robust training loop.

6.1 Accuracy of Binarized Neural Networks

In this first section of our results, we answer the first question about the performance of the model on the original images. The interesting part is also to assess how the binarized neural networks compare to the standard version in terms of accuracy.

To answer it, we consider all the model types as described previously in section 5.2. For each of these architectures, we use two different cases: a first one with a single

model (single) and a second one with a majority voting model (maj.) based on five networks with the same types.
The table 6.1 reports the accuracies obtained for the considered models.

	Type 1		**Type 2**		**Type 3**		**Type 4**	
Dataset	single	maj.	single	maj.	single	maj.	single	maj.
MNIST	97.27	97.87	90.51	91.24	94.36	96.17	97.29	98.65
FASHION	86.69	88.18	70.85	78.19	85.21	87.36	89.36	90.89
CIFAR	47.76	52.78	38.36	43.03	45.14	49.77	55.38	64.99

Table 6.1: Performance (in %) of BNNs on the test sets of the indicated data. Each model has been trained with 25 training epochs (no adversarial training is used here) and tested on the corresponding test set.

The observed values are aligned with the expectations. The largest models, types 1 and 4, show better results than the two smallest ones. Moreover, the majority voting strategy allows one to gain a few percentages when compared to the single model.

When compared to the state-of-the-art of classical neural networks, our binarized models perform quite well, especially on MNIST. Indeed, the best accuracy we found for a standard neural network[1] is 99.87%, only 1% more than our best score with BNNs. The two other datasets show less impressive results but recall that our goal here was not to fine tune BNNs to obtain the best accuracies. The state-of-the-art scores for Fashion-MNIST[2] and CIFAR-10[3] are respectively 96.91% and 99.5%.

Compared to other BNNs from the literature, our models show competitive results. In Jia's work [1], they obtained 97.15% on MNIST and 55.22% on CIFAR-10. For Fashion-MNIST, Khalil et al. got between 80% and 90% of accuracy.

In the next parts, we will focus on a majority voting model made with type 1 networks as they present good scores for each dataset. They are also much faster to train than the type 4 models, due to the higher number of parameters in the CNN. We claimed in the state-of-the-art part that networks that have a high original accuracy are more easily made robust. We can therefore expect good results for the MNIST dataset with the selected model. However, the case with CIFAR-10 may turn out to be not convincing because of its low accuracy.

[1] `https://paperswithcode.com/sota/image-classification-on-mnist`
[2] `https://paperswithcode.com/sota/image-classification-on-fashion-mnist`
[3] `https://paperswithcode.com/sota/image-classification-on-cifar-10`

6.2 Adversarial images for BNNs

Before trying to improve the robustness of our model, we have a first look at the data we will be using. More specifically, we wonder how the adversarial images evolve with the value of epsilon. Is the perturbation applied on the same pixels? Does it follow a pattern? Is it the same as an image with a simple random noise?

Next, we look at the transferability of the generated images. We will try to see if there is a significant amount of images that remain adversarial when passed to another model. Finally, we list all the adversarial datasets that we have generated and the approach used for each of them.

6.2.1 Evolution of the perturbation

We show below one example of the evolution of the perturbation according to the ϵ value for each dataset.
As stated in the Proposed approach chapter, we used the MiniSatCS solver to verify (i.e. attack) a model with a specific input. The resulting images are the adversarial images shown here.
Each of these images has been obtained on a model of type 1 that has been trained with 25 training epochs.

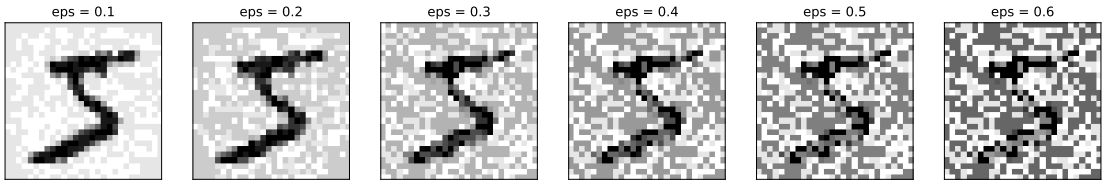


Figure 6.1: Adversarial images illustrating the evolution of the perturbed image for the MNIST dataset, according to the value of ϵ

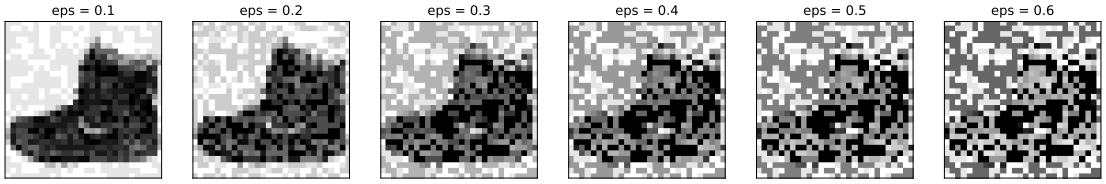


Figure 6.2: Adversarial images illustrating the evolution of the perturbed image for the Fashion-MNIST dataset, according to the value of ϵ

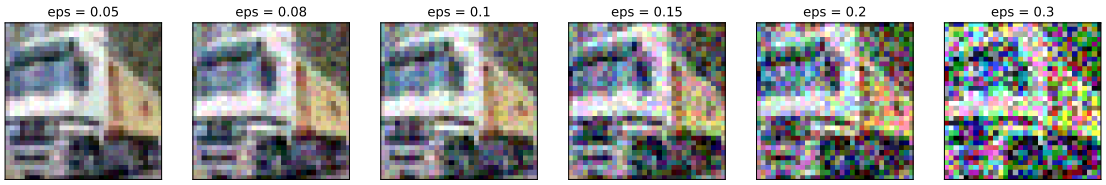


Figure 6.3: Adversarial images illustrating the evolution of the perturbed image for the CIFAR-10 dataset, according to the value of ϵ

If we observe the images, especially the ones for MNIST and Fashion-MNIST, we can see an interesting behavior. For low values of ϵ, no pattern seems to appear. However, once it becomes high enough (depending on the dataset), the same pixels of the image are affected by the perturbation. We can also see, thanks to the two last images of figures 6.1 and 6.2, that the applied disturbances can be the same for different datasets.

We can explain such observation with the working of the CDCL solver, described in section 2.3.2. It tries many different variable assignments until it is able to find an adversarial image for the given model. In case of failure, it simply backtracks and makes other decisions.
It also seems fair to suppose that adversaries are easier to find when the perturbation is high. In this case, one of the first variables assignment may provide the wanted result, which explains the apparition of the same pattern with the highest values of ϵ.

Note however that the images shown here are just one example. Many other perturbation patterns are present in the datasets. In other words, the adversarial training on these images is not limited to a training phase with a single new disturbance.

As stated before, it is much easier to find adversaries with CIFAR-10 due to the RGB channels. We note also that it is irrelevant to test the same ϵ values as the other datasets since we will quickly encounter multi-color nonsense instead of an interesting adversarial image.

We can also verify how many pixels are impacted by the attack. For this task, we compute the mean of pixels whose value has been changed between their original and adversarial version with 10 images. We do this for all of our binarized models, considering only a single network each time.

Dataset	Type 1	Type 2	Type 3	Type 4
MNIST ($\epsilon = 0.5$)	76.33	78.57	78.15	78.62
FASHION ($\epsilon = 0.5$)	85.84	86.42	86.42	86.2
CIFAR ($\epsilon = 0.1$)	99.06	99.01	98.95	98.93

Table 6.2: Mean percentage (%) of modified pixels over 10 images with a single model

It appears very clearly that the architecture of the model has no impact on the scope of the perturbation. This may be due to the MiniSatCS solver that tries directly to change many pixels in order to solve instances in less time. Further investigations with the type 1 model on MNIST have shown that the value of ϵ may impact a bit these results. With 10 images and $\epsilon = 0.1$, the percentage of modification, in this case, is 65.05%. The same test with $\epsilon = 0.2$ gives 74.14% of pixel modification. Thus, even if the perturbation value can have some effects on the number of modified pixels, its impact remains small as the case with $\epsilon = 0.2$ shows already close results to the one presented just before with $\epsilon = 0.5$.

6.2.2 Comparison with random noise

The next question we seek to answer is how different an adversarial image is from an image with random noise.
To investigate this, we simply select an image and add a random value to each pixel. It is important to note that the value of the pixel will always be in the range $[0, 1]$. We distinguish two cases: a classical noise, with values between 0 and 0.5, and a high noise, with a range from 0.3 to 0.5. The maximal value is selected here to match the value of ϵ that we will use the most often in the next results.

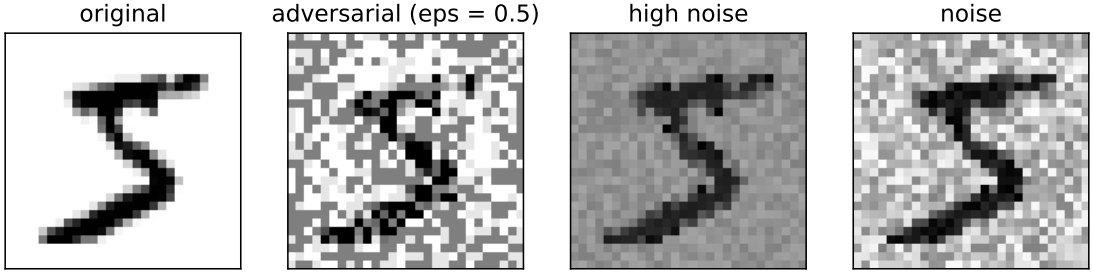


Figure 6.4: Illustration with a selected image of the adversarial, noise and high noise cases

As expected, the adversarial image appears to be very different from the simple addition of random noise. Fewer pixels are affected with a large perturbation value and the attack is able to target parts of the element of interest on the figure, as we

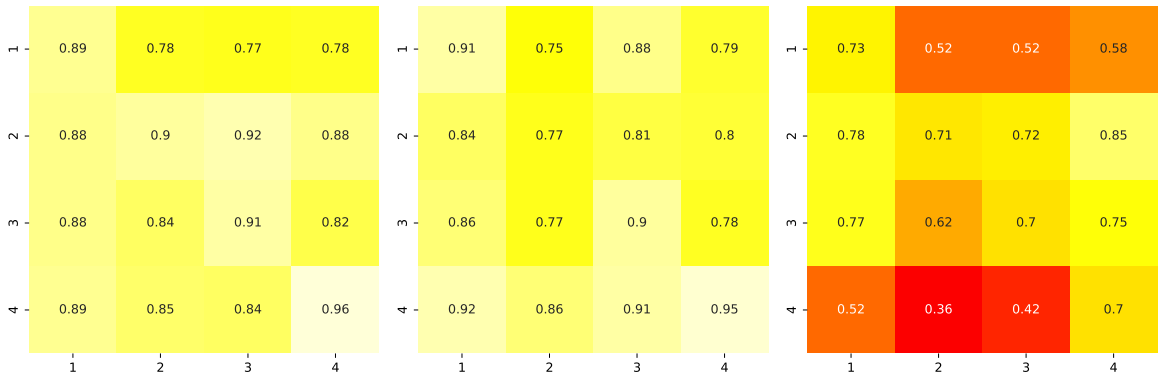
can see on the top of the 5.

This behavior is of course expected and illustrates once again the use of a solver to generate such images. Since each pixel is a variable encoded in the SAT formula, it is able to decide, up to a certain extent, which parts of the image are important for the classification by the model.

6.2.3 Transferability analysis

In order to increase the robustness of our model with the adversarial sampling approach, we first have to ensure that a reasonable amount of adversarial images keep this property when transferred to another model.

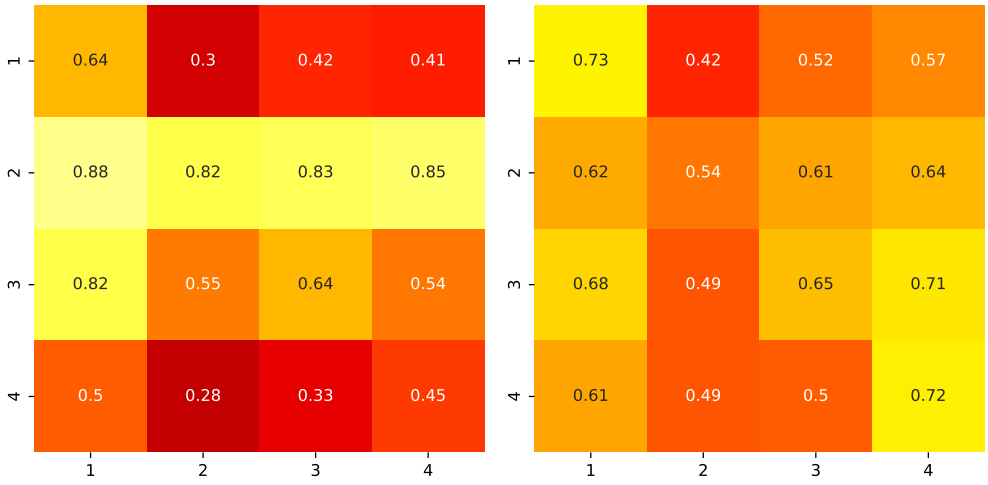
In this case, we distinguish two sets of models: the models on which we generate the adversarial images and the models on which we transfer the results. For the first category, we consider five models for each type, all of the same type, i.e. five models of type 1 when generating the images for the type 1 case. An image is therefore adversarial only when it fools all five models. Each of the five models is trained with 25 epochs.
For the second category, we only use one model of the given type for the classification. The results might therefore be higher than if we had used majority voting but still reflects this more complex model quite well.



(a) MNIST, $\epsilon = 0.5$ (b) Fashion-MNIST, $\epsilon = 0.5$ (c) CIFAR-10, $\epsilon = 0.1$

Figure 6.5: Transferability matrices between model types for each dataset. The square (i, j) represents the proportion of generated images from type j that remains adversarial when tested on type i.

From these numbers, we can conclude that, with a high enough value of ϵ, most



(a) MNIST, $\epsilon = 0.2$ (b) Fashion-MNIST, $\epsilon = 0.2$

Figure 6.6: Transferability matrices with a lower value of perturbation

of the images keep their adversarial property. This result is encouraging for the adversarial sampling approach of the robust training loop as we will then be able to generate such samples with this perturbation value and use them with many different model architectures.

Interesting observations can also be derived concerning the models if we consider a lower perturbation value, displayed in figure 6.6.
Recall first that the types 1 and 4 are the largest models we have. We can see that the images produced for these models, the first and last column of each matrix, are much stronger adversaries as they tend to keep their property more when passed to smaller models, types 2 and 3.
It proves here the importance of the model architecture, both in the generation and inference part. Larger models will be more difficult to attack but, when the attack succeeds, the result can be transferred to more models.

Also, we can notice that, in this case, the adversaries produced by the attack are much less transferable. Indeed, if we take for example the case of the type 1 model corresponding to the first line, we see that the percentage of transferable images has drastically dropped compared to the previous case with $\epsilon = 0.5$. This pushes us to use a higher value of disturbance, especially for the adversarial sampling approach.

6.2.4 List of generated images

Now that we have had our first look at the adversarial data, we state here all the adversarial sets we have created. They can be useful in our case for adversarial training with the adversarial sampling approach but can obviously be used in other contexts and for other works about robustness.

The data have been generated on one model of type 1 trained with 25 epochs on all the training data each time. We only used one model in this case in order to save time as the generation in itself takes already a large amount of time.

When we mention that x data from a set have been translated to adversarial examples, it simply means that we have used the x first data from the set. It does not mean that we have obtained x adversaries as some models are already robust for a very small part of the images.

For the MNIST dataset, we have many different sets. We have translated all the training data for $\epsilon = 0.5$, the $40,000$ first images for $\epsilon = 0.2$ and $10,000$ first samples for $\epsilon = 0.1$. In each case, the full test set has been translated to adversarial examples.

For the Fashion-MNIST dataset, we have used the $15,000$ first images with $\epsilon = 0.5$ and $10,000$ for $\epsilon = 0.2$. Once again, all the data from the test sets have been passed to adversarial.

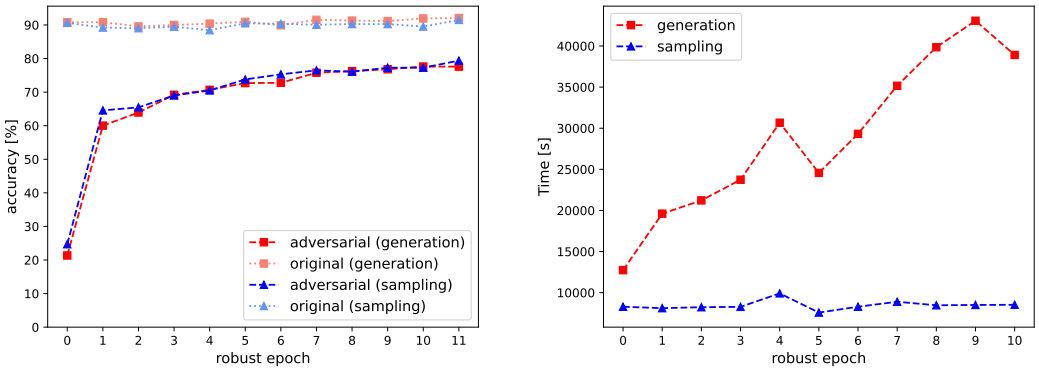
Finally, for the CIFAR-10 dataset, we have the first $15,000$ training samples and all the test set with $\epsilon = 0.1$ that have been translated to adversarial examples. The reason behind the smaller number of data, in this case, is due to the fact that these images are more than three times bigger than the other datasets, resulting in a much higher generation time.

6.3 Robust training loop experiments

This section deals with our proposed adversarial training: the robust training loop. We will perform diverse experiments to compare the generation and the adversarial sampling approaches. We will also compare different models to see which one can lead to the best improvement of the robustness. We will discuss how to evaluate the robustness according to the previously defined metrics. To go further, we will also use the algorithm to compare the adversarial training with random noise images and with a classical neural network. As a final test, we will use another method to evaluate the robustness of the fully trained models.

6.3.1 Adversarial sampling and generation approaches comparison

We start our comparison with one important question that we want to answer: is one of the approaches, adversarial sampling or generation, better than the other to improve the robustness of a BNN?



(a) Accuracy

(b) Time to generate 500 adversarial images

Figure 6.7: Comparison between the adversarial sampling ("sampling" in the graphs) and the generation approaches for the robust training loop on MNIST with $\epsilon = 0.5$. The model used the majority voting principle with five types 1 BNN, trained with 12 robust epochs and 5 training epochs. 500 different adversarial images are added at each robust epoch and we use $10,000$ original samples.

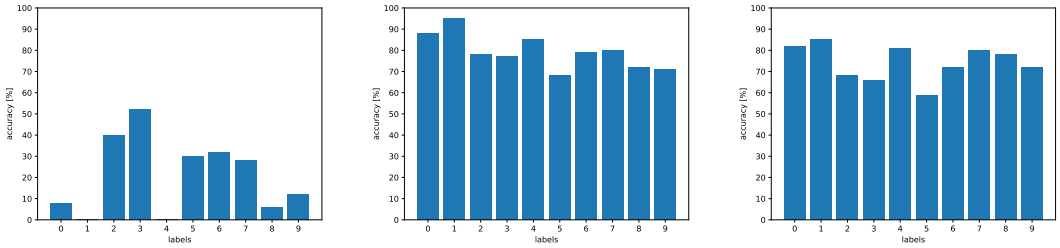
The first and obvious step to find the answer is to execute the algorithm. The results for the MNIST set are shown in figure 6.7. The "adversarial" line represents the adversarial accuracy while the "original" one is for the accuracy of the original images. All the values reported here are obtained on the evaluation of the test set.

The first thing we can observe is that adding adversarial data to the training set at each robust epoch is able to increase the adversarial accuracy without too large modification of the accuracy on the original images. Indeed, we can observe that this original accuracy remains almost constant and near its value of the case without adversarial training (i.e. robust epoch 0). Overall, this shows that the robust training loop is able to increase the robustness according to the criterion we consider here.

However, the results for the adversarial sampling approach concerning the adversarial part seem to be slightly higher. This can be explained by the fact that the test set used to evaluate this metric has been generated like the one used in the training

of this approach. Therefore, the data learned in the generation approach may have minor differences with the test set, leading to this small gap in the curves.
Recall that one of our goals with the adversarial sampling approach was to have an approximation of the generation version because of the time needed to create the images. If we only consider these two curves, we can say that our objective is reached as the results are very similar.

We can have a deeper look at the adversarial accuracy for each approach by considering the score for each class of the dataset. Figure 6.8 provides details about this test.
We can say that none of the classes appears to be preferred and virtually drags the accuracy upwards. An interesting element to notice is that certain classes have a good adversarial score, even without specific training.



(a) No adversarial training (b) Adv. sampling approach (c) Generation approach

Figure 6.8: Adversarial accuracy according to each label of the MNIST dataset, with $\epsilon = 0.5$. The model used is the same as the one previously described.

The major difference between the two approaches is the time. The values reported are the time required to generate 500 adversarial images at one robust epoch (or what would correspond to that epoch in the case of the adversarial sampling version). As the sampling approach does not use any generation, we show here the times recorded prior to the robust loop.
It is clear that the generation approach takes a significantly larger amount of time than the other case. This is due to many factors. First, the adversarial data are generated on five models (the ones used in the majority voting process) instead of a single one. More importantly, in the generation case, these images are created with models that should be more and more robust due to their increasing adversarial training. This can be seen in the increase of time, which grows almost linearly with the number of robust epochs.
A small peak followed by a decrease can be observed. This exposes that some images may be harder or easier to transform into adversarial than others.

This increase in the time of the generation is another proof that the robustness of the model is indeed increased at each epoch. This observation can also motivate the use of a threshold on the adversarial accuracy to stop the robust training loop early.

One can therefore try to find a trade-off between the two approaches. It appears clear that, according to the graphs, the adversarial sampling approach should always be preferred. It provides slightly better results for adversarial accuracy than the generation approach. It also performs that in much less time.
However, this sampling approach is not necessarily the best one for robustness when we look at it from another point of view.

Once the models have been fully trained, i.e. they have reached the last adversarial epochs, we can further investigate their robustness.
We propose here to look at various statistics we can obtain when performing an attack on these resulting models. As we have mentioned in the literature part, the two major ways to assess the robustness are to evaluate the robustness accuracy and to see how well the model can resist to an attack.

We consider here the same attack as before, the EEV one, on $1,000$ images. The table 6.3 contains the results with $\epsilon = 0.5$, the same as used in the training. The generation time reported here corresponds to the mean time required to find one adversarial image. The "no TLE" version simply removes the TLE cases from the computation of the mean (recall that the time out is fixed at 90 seconds).
In order to avoid a preference towards adversarial training, the images used with the attack come from the test set. Therefore, none of the three cases has ever seen an adversarial image corresponding to the original image given as input.

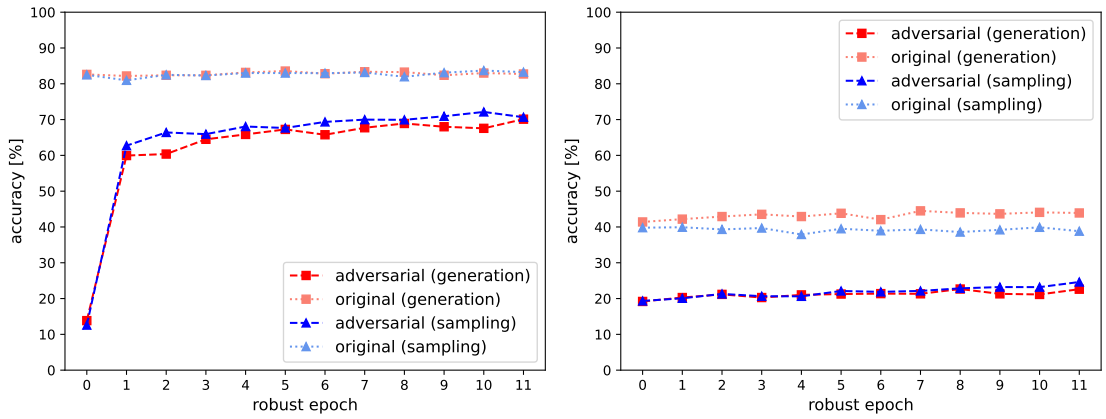
Model case	SAT	TLE	generation time [s]	generation time, no TLE [s]
No adversarial training	92%	8%	23.282	16.847
Adv. sampling approach	86.3%	13.7%	36.545	25.898
Generation approach	72.4%	27.6%	46.177	25.746

Table 6.3: Statistics of the adversarial attacks against trained models for MNIST with $\epsilon = 0.5$, both for training and the attack

This experiment provides evidence that the generation approach can lead to better resistance against the attack than the adversarial sampling approach. Even if it takes more time, the generation inside the loop seems to provide stronger adversaries that, once learned, can improve the inherent robustness of the model.

It is difficult to decide which of the two scores is the most relevant as they both evaluate parts of robustness that are difficult to compare. On one hand, the adversarial accuracy could be preferred when we simply want to have a model that is deployed in real situations and that could face adversarial (or noisy) images. On the other hand, the robustness against the attack may be favored when we know the model is going to be thoroughly tested, in research with other attacks for example.

To validate the results obtained with these two approaches, we can apply the same tests on Fashion-MNIST and CIFAR-10. Figure 6.9 presents the same graph as figure 6.7 but on these two other datasets. The setup used is also the same, with the only exception of the ϵ value used with CIFAR-10.



(a) Fashion-MNIST, $\epsilon = 0.5$

(b) CIFAR-10, $\epsilon = 0.1$

Figure 6.9: Comparison of the adversarial sampling and generation approaches on other datasets. For both sets, the model is a majority voting with 5 BNNs of type 1, trained with 12 robust epochs and 5 robust epochs on $10,000$ original samples. We add 500 adversarial samples at each robust epoch.

Concerning the adversarial accuracies, we observe the same tendencies as with MNIST, at least for the other gray-scaled dataset. However, the case with CIFAR-10 presents little to no improvement regarding the robustness. It echoes with the literature as we discussed that models with lower basic accuracy have more difficulty becoming robust.

To complete our validation, we also report in tables 6.4 and 6.5 the score for the robustness of the models against the attack.
Here, the results are different from what was expected, especially for the adversarial sampling approach. Indeed, instead of improving the resistance against the attack,

Model case	SAT	TLE	generation time [s]	generation time, no TLE [s]
No adversarial training	93.8%	6.2%	18.837	13.358
Adv. sampling approach	95.5%	4.5%	21.187	21.106
Generation approach	74.7%	25.3%	44.257	36.693

Table 6.4: Statistics of the adversarial attacks against trained models for Fashion-MNIST with $\epsilon = 0.5$, both for training and attack parts

Model case	SAT	TLE	generation time [s]	generation time, no TLE [s]
No adversarial training	61.3%	38.7%	65.97	37.148
Adv. sampling approach	72.8%	27.2%	58.62	38.99
Generation approach	55%	45%	80.401	38.911

Table 6.5: Statistics of the adversarial attacks against trained models for CIFAR-10 with $\epsilon = 0.1$, both for training and attack parts

it appears that the robust training loop has made it weaker than before. However, the generation time is still higher after any adversarial training, proving that the attack has more difficulties to obtain results (even if it manages to obtain them!).

In both cases, the behavior of MNIST with the generation approach is confirmed. The robustness against the attack is increased, even for CIFAR-10.
In conclusion of this comparison, adversarial sampling and generation approaches both present advantages and inconveniences. The time and adversarial accuracy would favor the sampling version of the robust training loop. More extensive research about attacks against the network should instead use the output of the generation approach. Therefore, if the goal is simply to deploy the model in real situations, the adversarial sampling approach is the one to prefer.

6.3.2 Model comparison

In the previous subsection, we used a majority voting model composed of five BNNs. However, in order to save time, is there another model that could achieve similar or better results while taking less time to train?
To find an answer, we will only consider the adversarial sampling approach as we have seen that both can provide similar results for the adversarial accuracy.
We will compare BNNs of type 1 but in different settings: our classical majority voting made of five networks ("majority"), a single model ("1 model") and the mean of five models ("5 models").

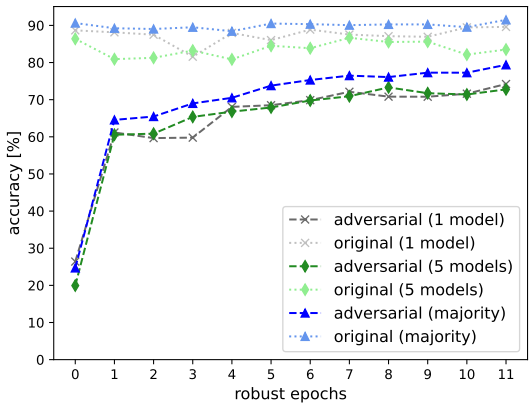


Figure 6.10: Robust training loop with different models using the adversarial sampling approach. All the models are trained with 12 robust epochs and 5 training epochs on $10,000$ original samples and 500 adversarial samples are added each robust epoch.

According to the accuracy shown in figure 6.10, it is clear that the majority voting model provides the best results. It shows at the same time a higher adversarial score and a more stable original accuracy than the two other cases. This can be explained simply by the fact that the mistake from one BNN can be compensated by the other networks.

The two other models, single and mean, show similar behavior for the adversarial accuracy. It is expected as the mean model should be the same as the single one minus the impact of the randomness during the training. The original accuracy is however quite different. It may simply show that the single model had more chance than usual for its prediction on this test set.

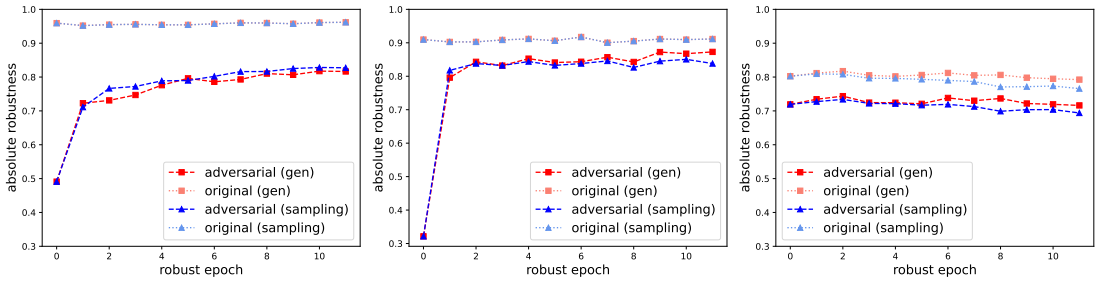
In the end, we will continue to mainly use the majority voting model. Even if it takes more time to train, the results are higher and more reliable than with a single model. Since we seek to improve the robustness, we believe that the choice of such model can bring an inherent consistency and lower the effect of randomness that could appear in the training.

6.3.3 Choice of robustness metric

Back in the proposed approach, we suggested other ways to evaluate the robust training loop than with the common adversarial accuracy. The motivation was to define a metric that uses directly the probabilities given by the model instead of only the classification label.

With such metrics, is there any new information we can use from them? Is the adversarial accuracy used until now sufficient? We answer these two questions in this subsection.

The first new metric we introduced was the absolute robust score, presented in section 4.1.1. If we apply the robust training loop with this metric, we observe the results displayed in figure 6.11.



(a) MNIST, $\epsilon = 0.5$ (b) Fashion-MNIST, $\epsilon = 0.5$ (c) CIFAR-10, $\epsilon = 0.1$

Figure 6.11: Robust training loop with the absolute robust score as robust evaluation criterion. The model is the same majority voting system with the same training as before.

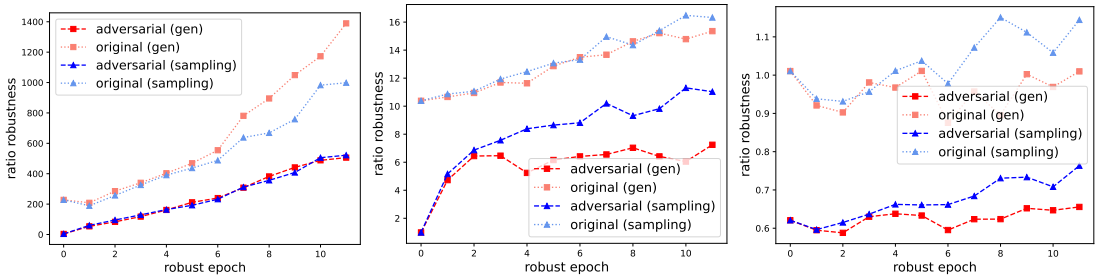
This metric has some interesting advantages. In addition to having similar curves to the adversarial accuracy, it allows us to compare our three datasets within the same range of robust evaluation as we can see that the obtained values are closer to one another than with the accuracy. Indeed, the direct use of the probabilities that are used to classify the samples will always be in the range from 0 to 1, independently of the difficulty of the data used. Instead, it will vary with the number of classes we consider, as these probabilities will be distributed over all the possibilities. In our case, we always have 10 different classes, allowing validation with the different datasets.

Note that in this case, we see an increase for MNIST and Fashion-MNIST but not for CIFAR-10. The first two can be explained by the fact that the model becomes more and more sure of its prediction for the adversarial images as it becomes more and more robust. Indeed, the metric uses the mean difference between the actual and the probability used to predict the label. The global increase of this metric is then directly linked with the accuracy, hence the same observed behavior.

However, for CIFAR-10, the results are high from the first robust epoch. It means that the gap between the probability used for the prediction and the one corresponding to the true label is not too large. It may explain why we have more difficulty to make it more robust with our approach, as we have seen in figure

6.9. This difference being smaller than the one for the other datasets may prevent the adversarial images to bring useful information to the network, as it already classifies them almost correctly.

The next two other metrics are the robust ratio evaluation and its inverse version, introduced in section 4.1.2. As we expected, they give almost the same results, which is why we only talk about the results of the original version, shown in figure 6.12.



(a) MNIST, $\epsilon = 0.5$ (b) Fashion-MNIST, $\epsilon = 0.5$ (c) CIFAR-10, $\epsilon = 0.1$

Figure 6.12: Robust training loop with the robust ratio as a robust evaluation criterion. The model is the same majority voting system with the same training as before.

This criterion is more difficult to analyze and exploit. Indeed, each dataset has its own range of results due to its inherent difficulty. We can only assess the fact that the logits used for the predictions are growing closer with the robust epochs.

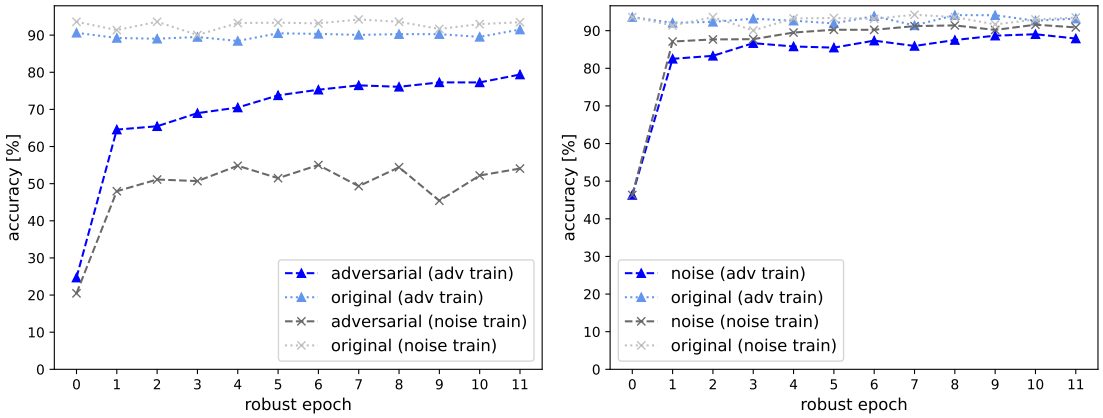
To answer the first question, the robust score evaluation can indeed provide some additional insights to the evaluation of the robustness inside the robust training loop. As we have discussed, the use of probabilities allows us to compare the datasets on the same range. However, to answer the second question, we believe the adversarial accuracy is sufficient in our case. It can indeed be directly and easily interpreted. It also has the advantage to be similar to the classical accuracy, widely used in the context of machine learning.

6.3.4 Comparison with random noise

Until now, we have used adversarial images created with the EEV attack. Due to their appearance, we wonder if the use of simple noisy images would bring the same information to the model and also improve the adversarial accuracy. In order to answer this question, we perform two versions of the robust training loop for the data augmentation part: one with the use of adversarial images and the other with noisy

images. We used the adversarial sampling approach to compare them as the generation approach with noisy images is internally not different from the sampling one.

We used the same setup for the models as before: 12 robust epochs, 5 training epochs and 10,000 original samples and adding 500 images each robust epoch. Here, we add either adversarial images or noisy images and we test the majority voting models on adversarial or noisy test sets. The noisy images used are the ones presented before but not the high noise.



(a) Accuracy on adversarial images

(b) Accuracy on noisy images

Figure 6.13: Comparison of the accuracies when performing the robust train loop with adversarial or noisy images

Figure 6.13 shows our previous result of the adversarial sampling robust loop with the new values for the networks trained with noisy images. Different points can be observed. It is clear that the training with the noise does not improve the adversarial accuracy as much as the adversarial training. The increase at the start is most probably due to the information that noise exists (as the network never saw it in the original dataset) but then this accuracy stabilizes around 50%. The adversarial training, as said before, continues to increase the robustness at each robust epoch.
On the other hand, when we test our models on noisy images, the adversarial training performs almost as well as the noisy one.
This tends to confirm what was discussed in the theoretical part: adversarial images are more than just images with random noise. We can even add that the adversarial training is more powerful than expected as it is able to generalize the improvement of the performance to noisy images.

6.3.5 Application to classical neural networks

We said before that the robust training loop algorithm could easily be used with other deep learning models, possibly non-binary. Moreover, nothing prevents us to use the pre-generated adversarial images with other models, as for the adversarial sampling approach.
Therefore, we want to know if an equivalent classical neural network shows the same behavior and performance as our BNN version when trained with the robust training loop.

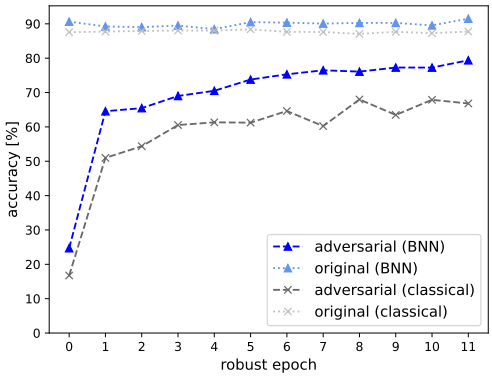


Figure 6.14: Comparison of the robust training loop between a classical neural network and a BNN. Both models are based on the majority voting of 5 versions of their corresponding architecture (either classical or BNN type 1) with 12 robust epochs and 5 training epochs. We use $10,000$ original samples and 500 adversarial samples are added at each robust epoch.

For this experiment, we used the same majority voting as before. The classical neural network follows the same architecture with the exception of the activation functions. Instead of the sign function mandatory for the BNNs, we used the classical ReLU function. As for the BNN version, we also used a majority voting of five classical neural networks with the described architecture.
Figure 6.14 details the results obtained for this comparison.

In both cases, the adversarial accuracy is increased. However, we see a large difference between the classical and binarized versions for this criterion. Additionally, the original accuracy is higher with the BNN than with the other which is unexpected as the binarization of the weights and activation values generally leads to a decrease of the performance.

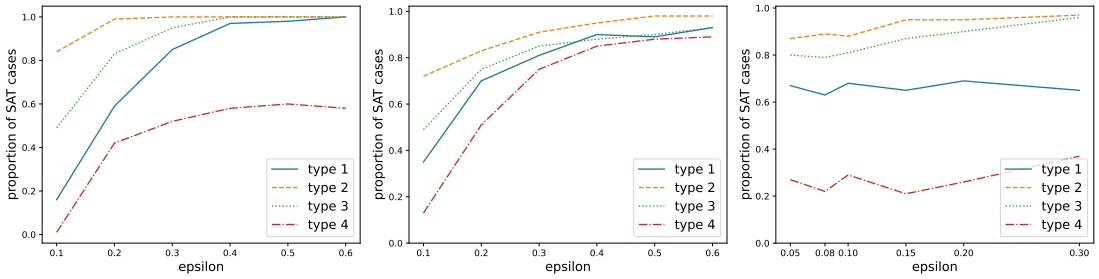
We believe different factors can explain these results. The architecture of the classical neural network is the same as the one used in the BNN (except for the

activation function). However, it may not be well suited for the floating-point values in such network. Moreover, recall that the images have been created with BNNs and, even if the transferability is good as we can see at robust epoch 0, there is a possibility that such images are more useful to this type of neural network.
Finally, as there is still an increase of the adversarial accuracy, it is possible that more robust epochs would improve the classical neural networks in the end. As convergence for the BNNs is faster than their original version, it seems logical to assume that classical models will catch up later, following the same curve.

6.3.6 Different perturbation values

Now that we have presented our various tests for the robust training loop, we consider one final experiment to evaluate the robustness of the final models. Similarly to the tables 6.3, 6.4 and 6.5, we perform attacks against these models but, instead of using a single ϵ value, we use many values to see how the robustness evolves with growing perturbation.

To run the attacks, we used majority voting models and a defined range of ϵ values. As a baseline, we consider first the curves obtained for each type of BNN and no adversarial training. The values for this are displayed in figure 6.15.



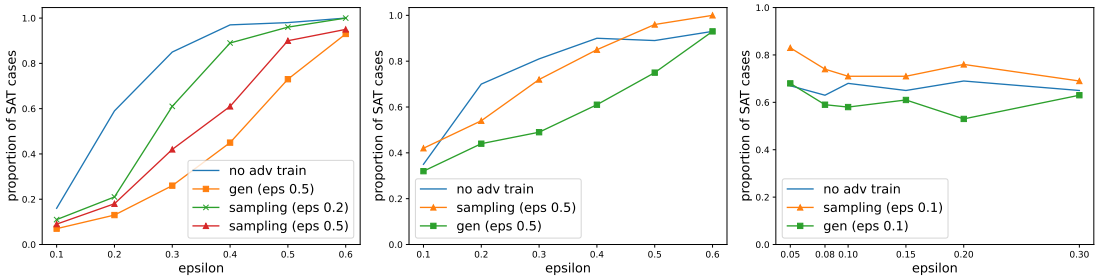
(a) MNIST (b) Fashion-MNIST (c) CIFAR-10

Figure 6.15: Number of SAT cases (i.e. number of adversarial images found) according to increasing perturbation value ϵ for each dataset and each BNN type. The test is done with 100 different images coming from the test set.

A first interesting observation that can be made here is that the resistance of a model is linked with its architecture. It echoes what we have said in the literature since this point was already studied for classical neural networks. We can then conclude that the binarized version also follows the rule: a larger network (in terms of parameters) will have more basic, natural robustness.
Concerning the number of adversarial images we can find, MNIST and Fashion-MNIST follow the same, and expected, tendency: the higher the perturbation is,

the more SAT cases we have. This is however not the case for CIFAR-10 where the ratio stays roughly the same, even with a high perturbation value for this dataset. We believe this counter-intuitive result is most probably due to the data themselves but it would require more extensive work on these particular images to understand how we can improve our performance on them.

As a follow-up to this test, We use now the same principle but with our adversarially trained models to assess if we have indeed improved the resistance to a full range of attacks.



(a) MNIST (b) Fashion-MNIST (c) CIFAR-10

Figure 6.16: Comparison, with type 1 BNNs, of the robustness against a range of perturbation value ϵ. Again, we used 100 images from the test set, the same as before.

For the MNIST datasets, the results correspond to what we were expecting and confirm the observation from table 6.3. Both sampling and generation approaches improve the robustness of the model and, in such case, the generation one lead to fewer SAT cases. We can also observe that the higher the perturbation is, the more we can find adversarial images with the attack.

An adversarial training with a lower ϵ value is still better than no adversarial training but grows faster than the cases we consider during all of our previous tests (i.e. $\epsilon = 0.5$). It may be caused by the fact that less perturbation does not provide enough generalization to the model in order to become more robust. Also, recall that the adversarial sampling approach uses pre-generated data and that images with a perturbation of 0.2 have less transferability than the case with $\epsilon = 0.5$, as seen in figure 6.6. Therefore, the images used may not all be adversarial, leading to these worse results.

However, for the two other datasets, the results are not so convincing. The generation approach is still the best one for this experience in both cases but we see the same phenomenon as before for CIFAR-10. Concerning the Fashion-MNIST case, the adversarial sampling approach under-performs for the extreme value of

the perturbation (as we have seen in table 6.4) but follows the expected behavior most of the time. The higher number of SAT cases reflects that this dataset is more complicated than its digit version, as we have mentioned many times before.

6.3.7 Conclusion on the results

We have now extensively investigated the adversarial images and their properties, assessed the methods to evaluate the robustness and also experimented with our proposed adversarial training.

Using our generated adversarial images, we have seen that the robust training loop can indeed improve the robustness. According to the considered criterion, the adversarial accuracy or the resistance against the attack, one could prefer the adversarial sampling or the generation approach. Recall however that the sampling approach has been created as an approximation of the other one in order to save time.

Overall, the results observed are coherent between all the tests we have performed. The transferability detailed in figure 6.5 is still observed in all our graphs of the robust training loop for the robust epoch 0.
In these graphs, we also showed the original accuracy, just like in the table 6.1. The difference is simply explained by the fact that we used much less data in our tests for the robustness in order to save time.

Concerning all our tests for the robustness itself, robust training loops and attacks on trained models, the results are consistent with each other. Indeed, we observe both an increase in the accuracy and an improvement in the resistance for the two approaches.
The results for these criteria with MNIST are convincing but the validation on Fashion-MNIST and, especially, CIFAR-10 is often complicated. However, it tends to make sense as these images are more difficult to classify. Therefore, the increase of the adversarial accuracy is lower and the resulting resistance against attacks is not as good as for MNIST.

In conclusion, the robust training loop with the generated adversarial images does seem to improve the robustness as we have defined it in this work. It is however difficult to compare the results and conclusion with the literature as very few work on adversarial training of BNNs exists. Moreover, it is difficult to use classical attack (and therefore adversarial training from another publication) due to the binary weights and specific gradients of such quantized networks.

Chapter 7

Discussion

With all our experiences, we have reached what we believe to be good results. Both the adversarial accuracy and the resistance against the attack are improved. Moreover, the robust training loop allows us to stop the adversarial training when we see fit and according to a threshold value we can define before the learning phase.

It is however difficult to conclude whether or not our results are better than the ones we have talked about in the literature part. Indeed, most classical attacks could not work as they should against BNNs due to the sign activation function. As the different adversarial trainings are often linked to their attack, adapting them for the BNNs would end up as a whole full work, which was not our objective here. We therefore chose to compare to the simple baseline of a model without adversarial training.

Another important note we would like to address is the actual robustness of our models. We showed through various tests that we have increased the adversarial accuracy and the resistance of the network. However, the results are for the Efficient and Exact Verification, not for all possible attacks. It is then difficult to conclude that the models we provide at the end are indeed truly robust as another type of attack may still make it fail easily. Such comment is actually applicable to many of the publications we used in the literature part, letting us think that more work is still needed to reach global robust deep learning models.

During our work, we mainly focused on one perturbation value which is arguably high. The main reasons behind it were to speed up the adversarial image generation but also to have sufficient transferability. More research and experiments could be done with lower values but we would then recommend using another attack than EEV.

Chapter 8

Conclusion

The main goal of this master's thesis was to design and test a new adversarial training for binarized neural networks. After investigating the generation of adversarial images and many experiences related to this training, we finally conclude this work.
Our motivation was to propose an algorithm that can improve the robustness of a BNN according to the adversarial accuracy and its resistance against an attack. Moreover, we wanted a method that could be easily adapted to other attacks and other deep learning models.
The reason behind all this work is that robustness is a key property of neural networks. The interest in this domain is growing as people are more and more concerned with the safety of deep learning. To the best of our knowledge, no extensive work has already been realized for binarized neural networks, which are a promising architecture to use deep learning on embedded devices.

We first explained the necessary theoretical background related to the models we used. We focused then on the attacks against the deep networks and how to improve their robustness against them. We started with an explanation of adversarial images, the basic component of both of these parts. There, we described properties that one can expect from them, such as the transferability of their adversarial characteristic from one model to another. Next, we explained how one generally evaluates the robustness of a model. If many researchers look simply at the adversarial images, we, however, focused more on the model. With this in mind, we defined the very important adversarial accuracy that we used in the rest of our work.

Of course, most of the existing publications have been done on real-valued neural networks. For such classical architectures, there are many possible attacks to generate adversarial samples. We have presented the Fast Gradient Sign Method, which is the most largely used in practice, but also its variant Projected Gradient

Descent and others. From there, we discussed the possible adversarial training with such attacks. The common approach, as we detailed in this part, is to modify the loss function and to generate adversaries in the training and use them directly in this learning phase.

After that, we investigated the state-of-the-art for binarized neural networks. Classical attacks based on the gradient may not provide the expected results due to the sign activation function. It is therefore necessary to use other approaches such as the Satisfiability Modulo Theory or a SAT solver with a matching encoding of the model. However, no specific algorithm for adversarial training seemed to exist.

To answer the need for adversarial training with BNNs, we proposed a simple modification of the training algorithm we called the robust training loop. As we explained, its principle is to dissociate the training of the model from the attack. Therefore, we can use various deep learning models and various attacks.
In our experiences, we compared the generation approach and its approximation, the adversarial sampling approach. We saw that both of them can improve the robustness of the models according to the adversarial accuracy. Moreover, the generation version is also able to reinforce the network to make the attack against it less efficient. We continued our test to conclude that the best case to consider here is to use a majority voting model as it will provide better performance and the aggregation of networks will provide a stronger natural robustness. We then confirmed that noisy images do not lead to the same results and therefore are indeed different from adversarial images. After a small comparison with a classical neural network, we ended our experiments with different perturbation values to conclude that the robustness, under the criterion of the number of adversarial images found, is indeed improved compared to a model with no specific training.

Further work is always possible, especially since there are almost no publications to compare our work with. Other attacks against binarized neural networks could be developed and used with the robust training loop. We could also try to adapt the classical attacks to this quantized architecture to be able to use their corresponding adversarial training. With an extension of the encoding part, more complex binarized models could also be used, allowing us to reach even higher robustness.

Bibliography

[1] Kai Jia and Martin Rinard. "Efficient exact verification of binarized neural networks". In: *Advances in neural information processing systems* 33 (2020), pp. 1782–1795.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[3] Michel Verleysen and John Lee. *Course LELEC2870 : Machine Learning : regression and dimensionality reduction*. Ecole Polytechnique de Louvain. 2022.

[4] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[5] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[6] Itay Hubara et al. "Binarized neural networks". In: *Advances in neural information processing systems* 29 (2016).

[7] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[8] Chunyu Yuan and Sos S Agaian. "A comprehensive review of binary neural network". In: *Artificial Intelligence Review* (2023), pp. 1–65.

[9] Mohammad Rastegari et al. "Xnor-net: Imagenet classification using binary convolutional neural networks". In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV*. Springer. 2016, pp. 525–542.

[10] Shuang Liang et al. "FP-BNN: Binarized neural network on FPGA". In: *Neurocomputing* 275 (2018), pp. 1072–1086.

[11] Joseph Bethge et al. “Back to simplicity: How to train accurate bnns from scratch?” In: *arXiv preprint arXiv:1906.08637* (2019).

[12] Tom Bannink et al. “Larq compute engine: Design, benchmark and deploy state-of-the-art binarized neural networks”. In: *Proceedings of Machine Learning and Systems* 3 (2021), pp. 680–695.

[13] Joao Marques-Silva, Inês Lynce, and Sharad Malik. “Conflict-driven clause learning SAT solvers”. In: *Handbook of satisfiability*. IOS press, 2021, pp. 133–182.

[14] Kevin Eykholt et al. “Robust physical-world attacks on deep learning visual classification”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 1625–1634.

[15] Christian Szegedy et al. “Intriguing properties of neural networks”. In: *arXiv preprint arXiv:1312.6199* (2013).

[16] Andras Rozsa, Manuel Günther, and Terrance E Boult. “Are accuracy and robustness correlated”. In: *2016 15th IEEE international conference on machine learning and applications (ICMLA)*. IEEE. 2016, pp. 227–232.

[17] Aleksander Madry et al. “Towards deep learning models resistant to adversarial attacks”. In: *arXiv preprint arXiv:1706.06083* (2017).

[18] Nicholas Carlini and David Wagner. “Towards evaluating the robustness of neural networks”. In: *2017 ieee symposium on security and privacy (sp)*. Ieee. 2017, pp. 39–57.

[19] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *arXiv preprint arXiv:1412.6572* (2014).

[20] 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*. 2016, pp. 2574–2582.

[21] Arjun Nitin Bhagoji et al. “Practical black-box attacks on deep neural networks using efficient query mechanisms”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 154–169.

[22] Yandong Li et al. “Nattack: Learning the distributions of adversarial examples for an improved black-box attack on deep neural networks”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3866–3876.

[23] Florian Tramèr et al. “Ensemble adversarial training: Attacks and defenses”. In: *arXiv preprint arXiv:1705.07204* (2017).

[24] Eric Wong, Leslie Rice, and J Zico Kolter. “Fast is better than free: Revisiting adversarial training”. In: *arXiv preprint arXiv:2001.03994* (2020).

[25] Ali Shafahi et al. “Adversarial training for free!” In: *Advances in Neural Information Processing Systems* 32 (2019).

[26] Xiaofeng Mao et al. “GAP++: Learning to generate target-conditioned adversarial examples”. In: *arXiv preprint arXiv:2006.05097* (2020).

[27] Ian Goodfellow et al. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[28] Ali Shafahi et al. “Label smoothing and logit squeezing: A replacement for adversarial training?” In: *arXiv preprint arXiv:1910.11585* (2019).

[29] Angus Galloway, Graham W Taylor, and Medhat Moussa. “Attacking binarized neural networks”. In: *arXiv preprint arXiv:1711.00449* (2017).

[30] Van-Ngoc Dinh et al. “A Study on Adversarial Attacks and Defense Method on Binarized Neural Network”. In: *2022 International Conference on Advanced Technologies for Communications (ATC)*. IEEE. 2022, pp. 304–309.

[31] Elias B Khalil, Amrita Gupta, and Bistra Dilkina. “Combinatorial attacks on binarized neural networks”. In: *arXiv preprint arXiv:1810.03538* (2018).

[32] Guy Katz et al. “Reluplex: An efficient SMT solver for verifying deep neural networks”. In: *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer. 2017, pp. 97–117.

[33] Nina Narodytska et al. “Verifying properties of binarized deep neural networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[34] Yann LeCun. “The MNIST database of handwritten digits”. In: *http://yann. lecun. com/exdb/mnist/* (1998).

[35] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms”. In: *arXiv preprint arXiv:1708.07747* (2017).

[36] Tien Ho-Phuoc. “CIFAR10 to compare visual recognition performance between deep neural networks and humans”. In: *arXiv preprint arXiv:1811.07270* (2018).

Computational resources have been provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region.

Appendices

Appendix A

Dataset samples

A.1 MNIST

Figure A.1: Images from MNIST dataset

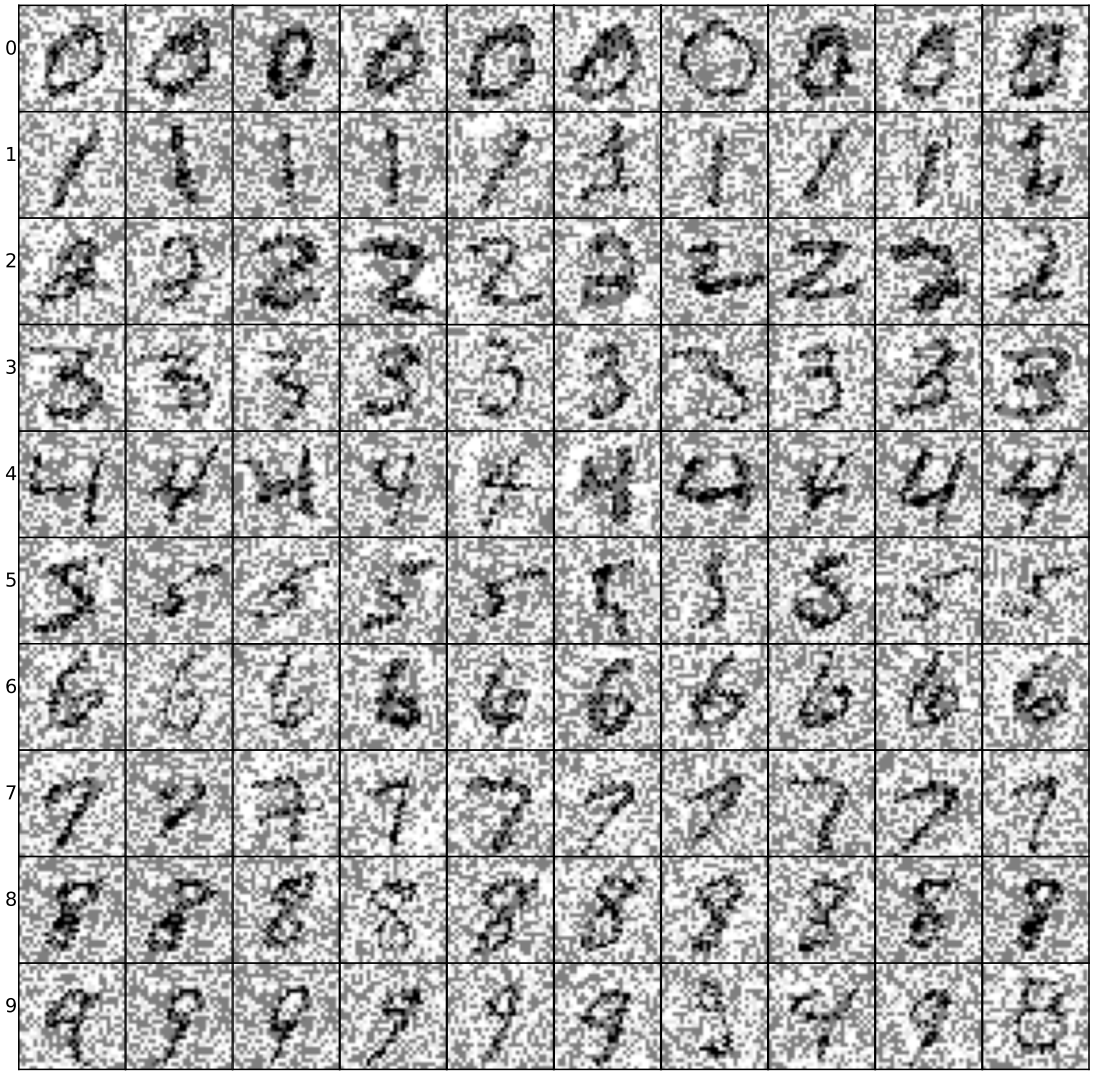


Figure A.2: Adversarial images for MNIST with $\epsilon = 0.5$

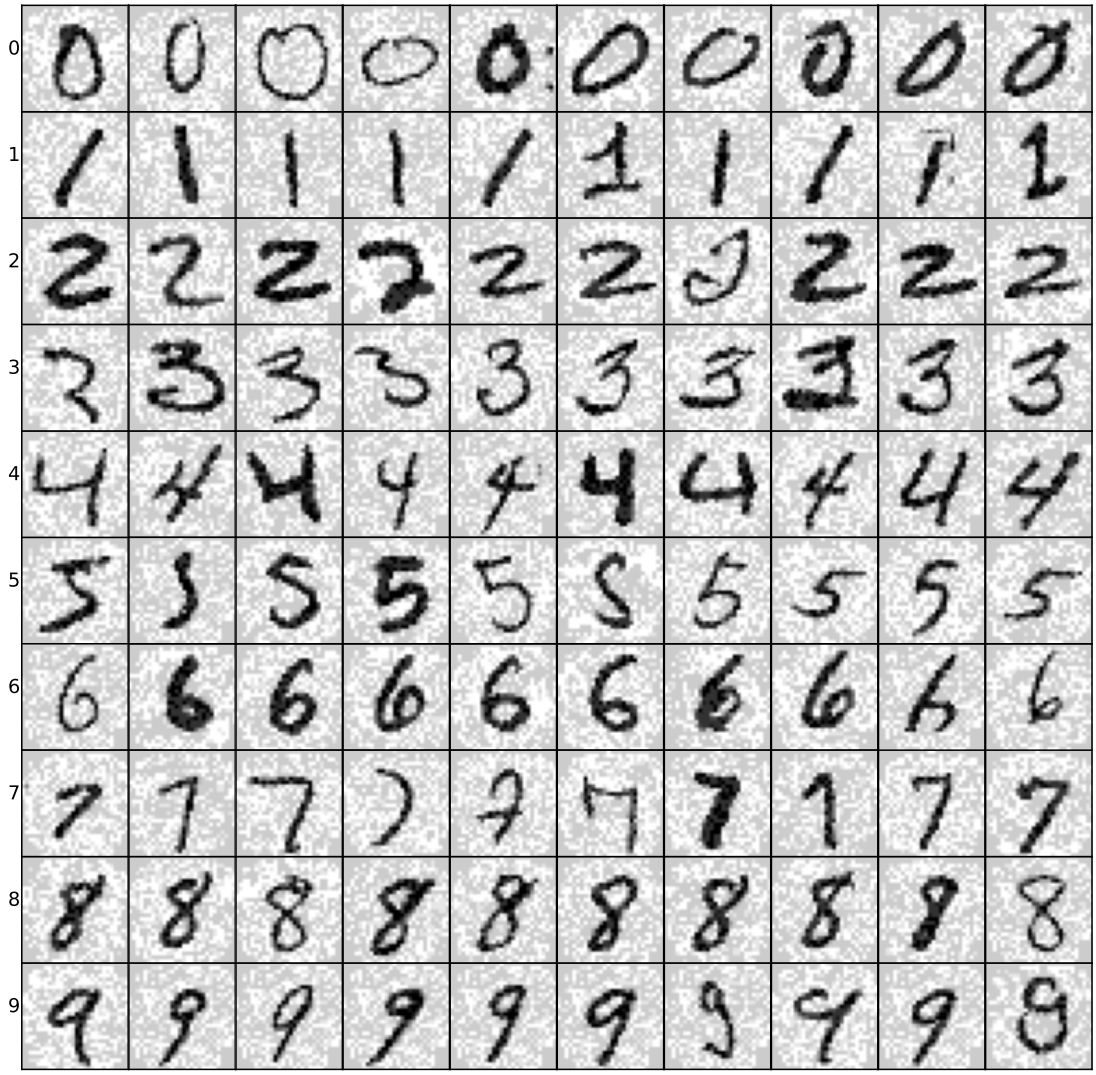


Figure A.3: Adversarial images for MNIST with $\epsilon = 0.2$

A.2 Fashion-MNIST

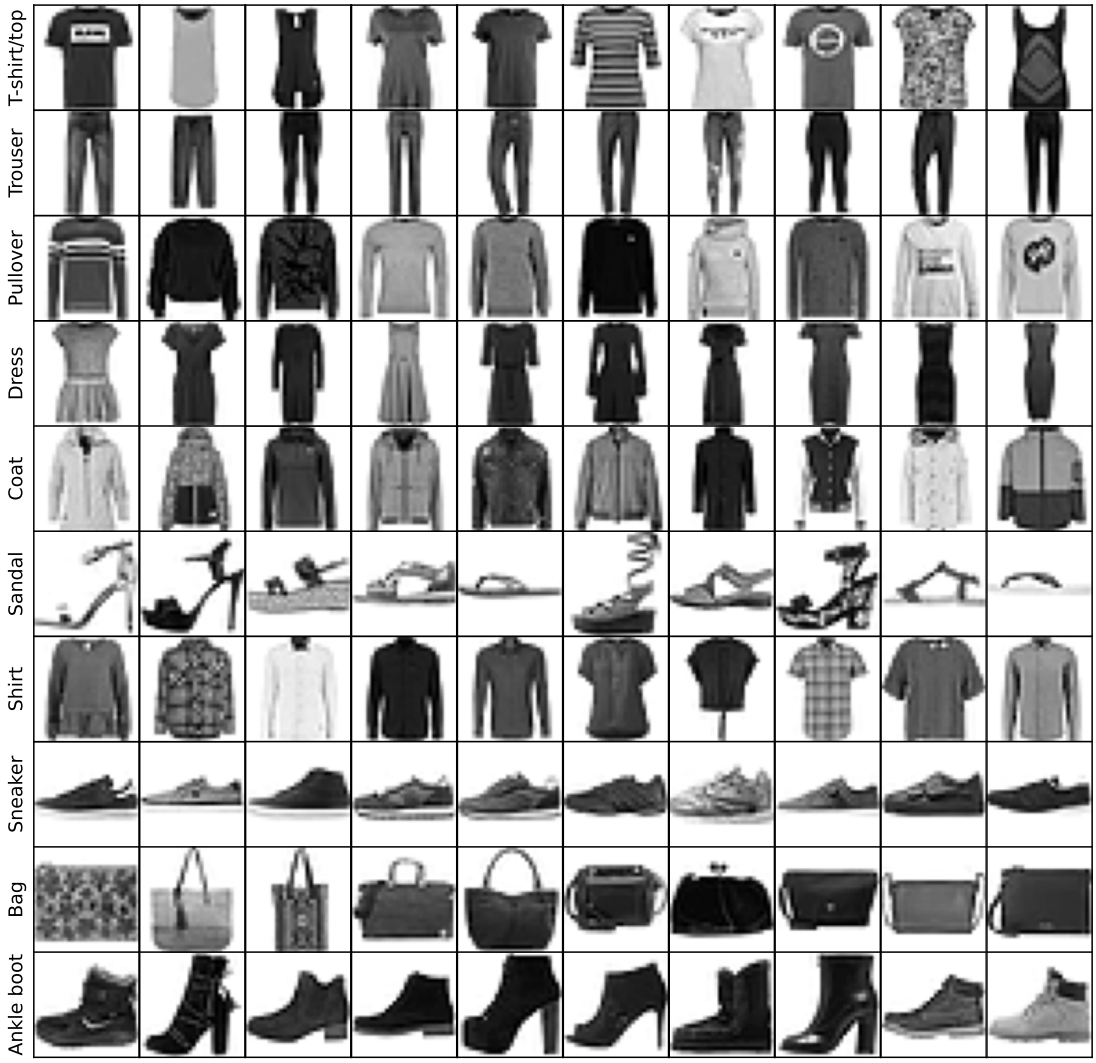


Figure A.4: Images from Fashion-MNIST dataset

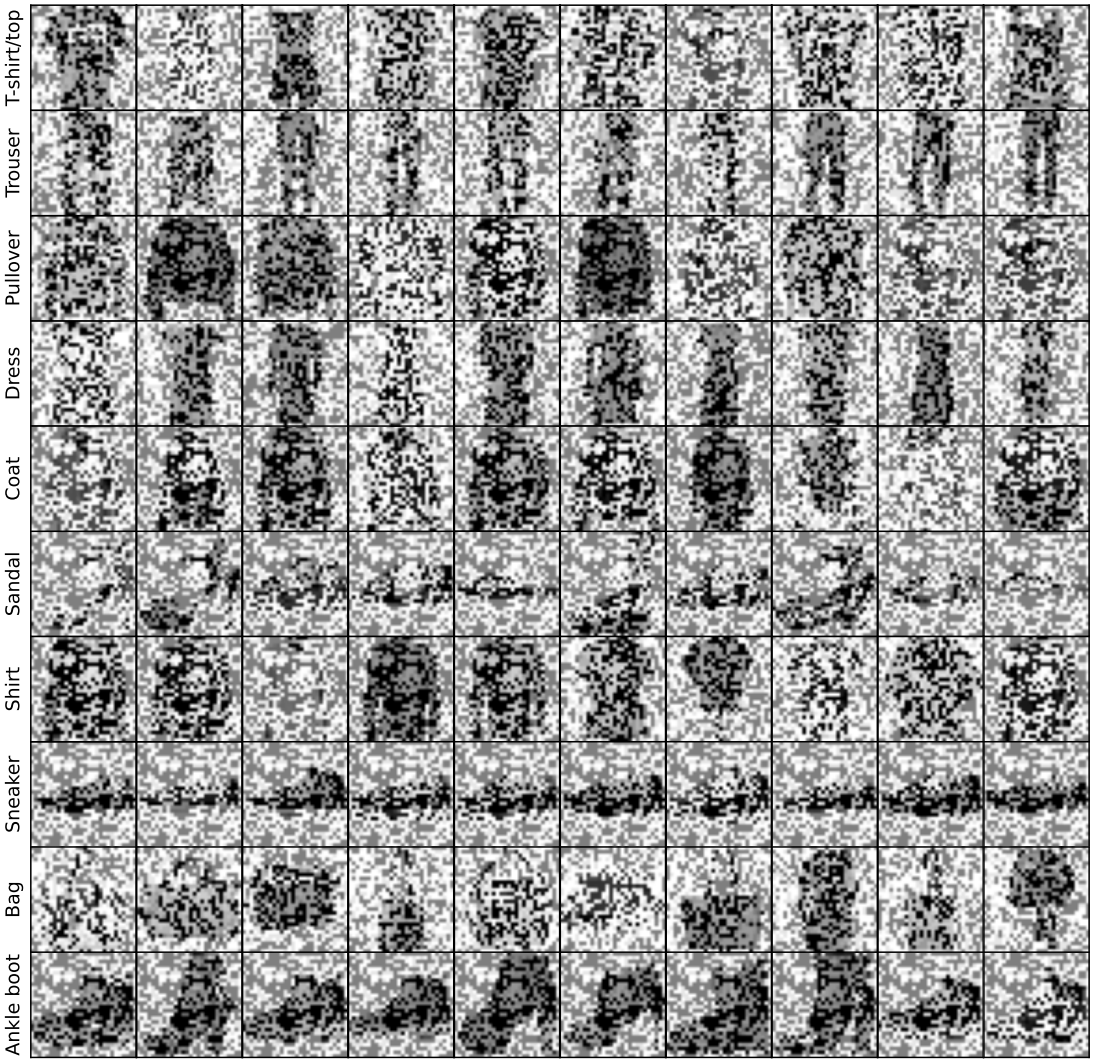


Figure A.5: Adversarial images for Fashion-MNIST with $\epsilon = 0.5$

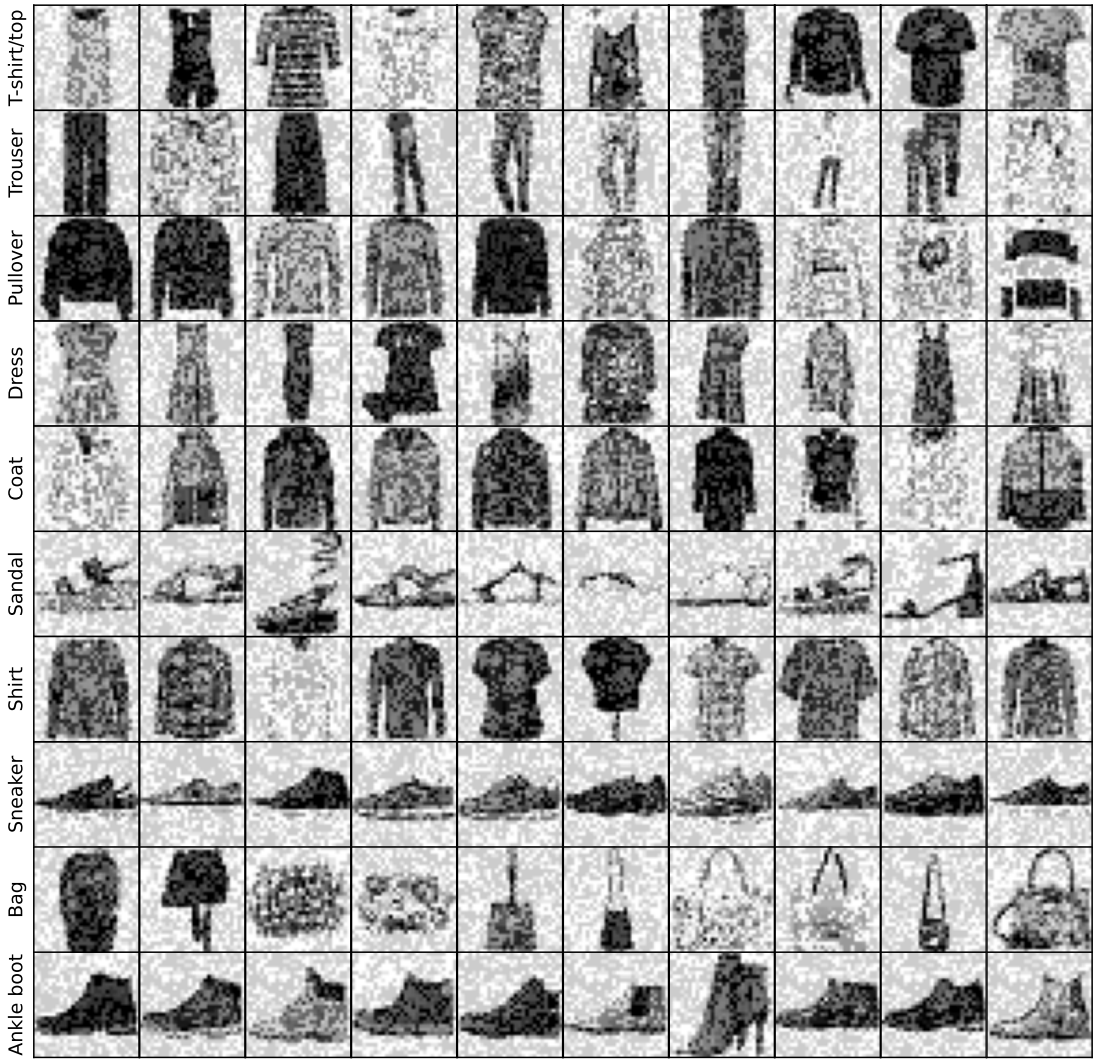


Figure A.6: Adversarial images for Fashion-MNIST with $\epsilon = 0.2$

A.3 CIFAR-10

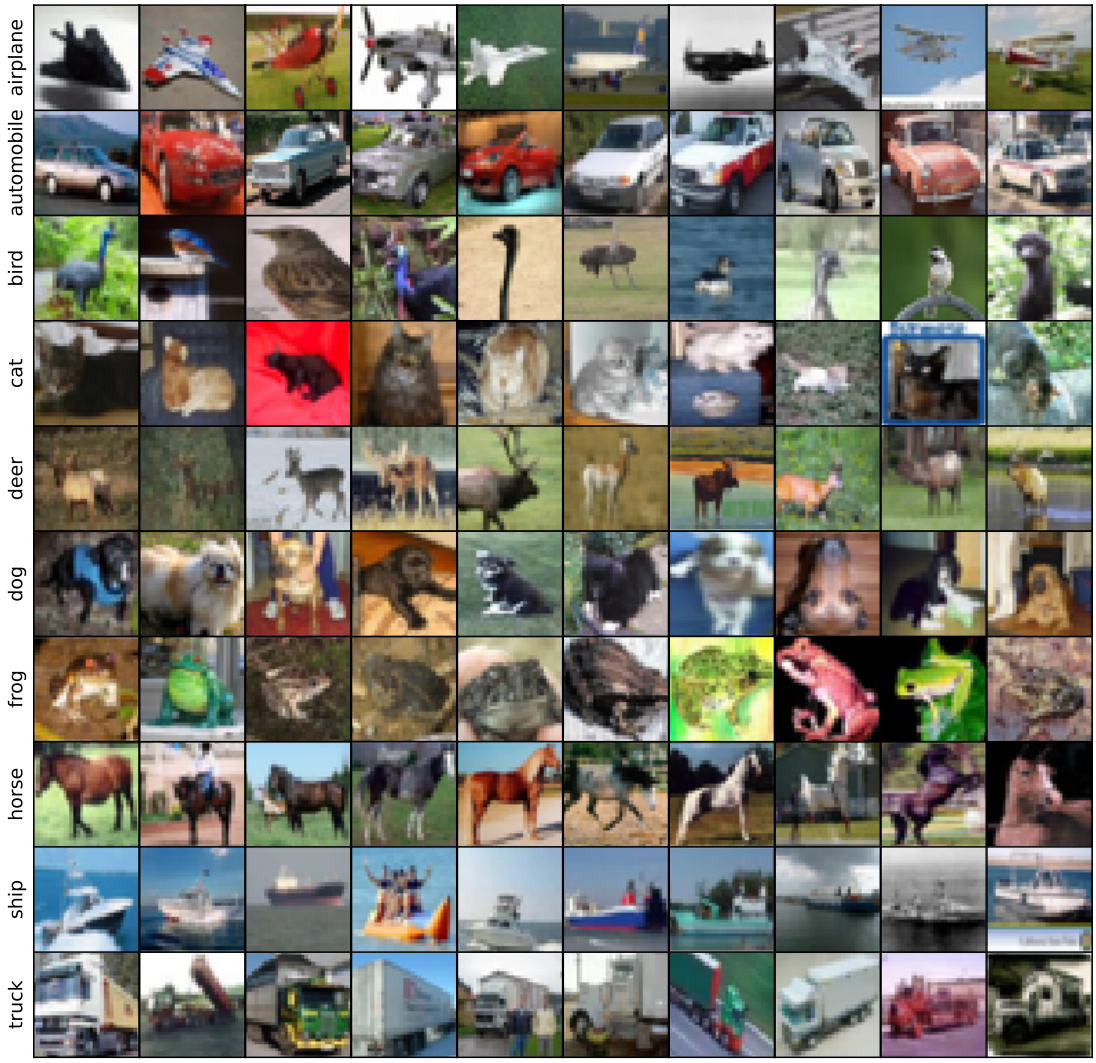


Figure A.7: Images from CIFAR-10 dataset

Figure A.8: Adversarial images for CIFAR-10 with $\epsilon = 0.1$

Appendix B

Additional classification details

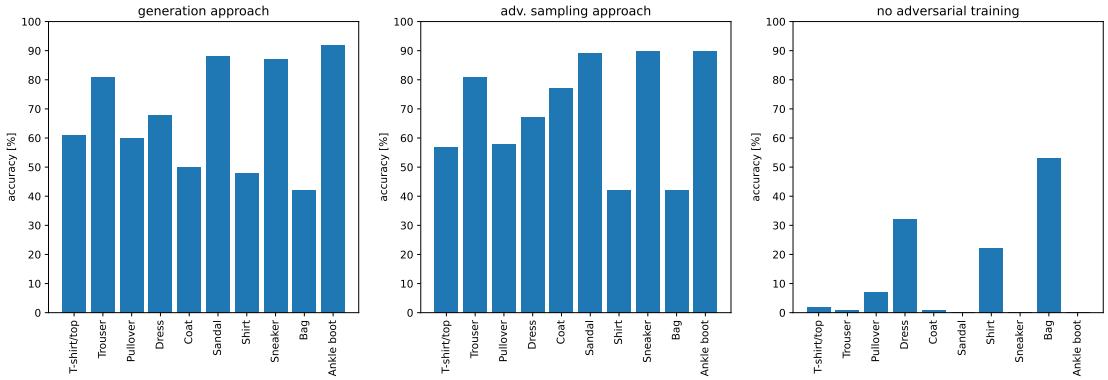


Figure B.1: Details of the adversarial accuracy for Fashion-MNIST with $\epsilon = 0.5$

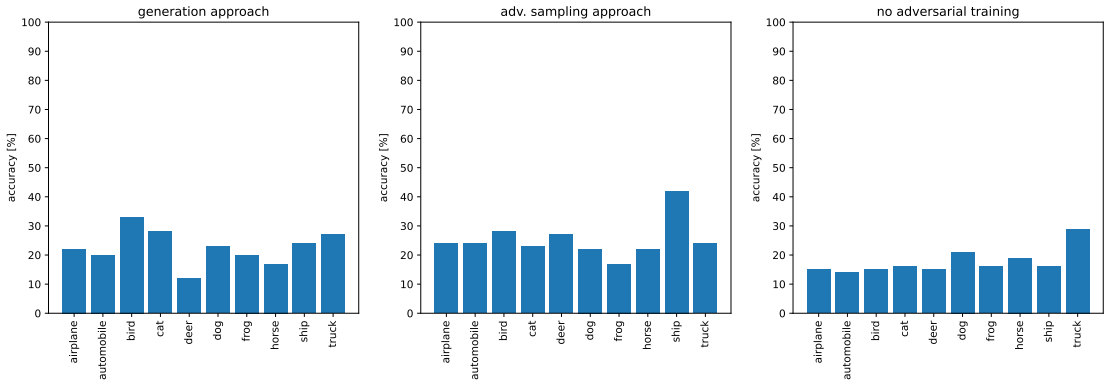


Figure B.2: Details of the adversarial accuracy for CIFAR-10 with $\epsilon = 0.1$

Appendix C

Additional experiences with the robust training loop

We show here less important experiences with the robust training loop.
The setup used is the same as in the other tests, except when mentioned. This means a majority voting model with 5 types 1 BNN, 12 robust epochs, 5 training epochs, $10,000$ original samples and we add 500 adversarial samples each robust epoch.

C.1 Reverse case

This case is just to verify the "reverse" case of the robust training loop. It means that we inverse the use of the adversarial and original images. Therefore, we start with $10,000$ adversarial samples and add 500 original images each "robust" epoch.

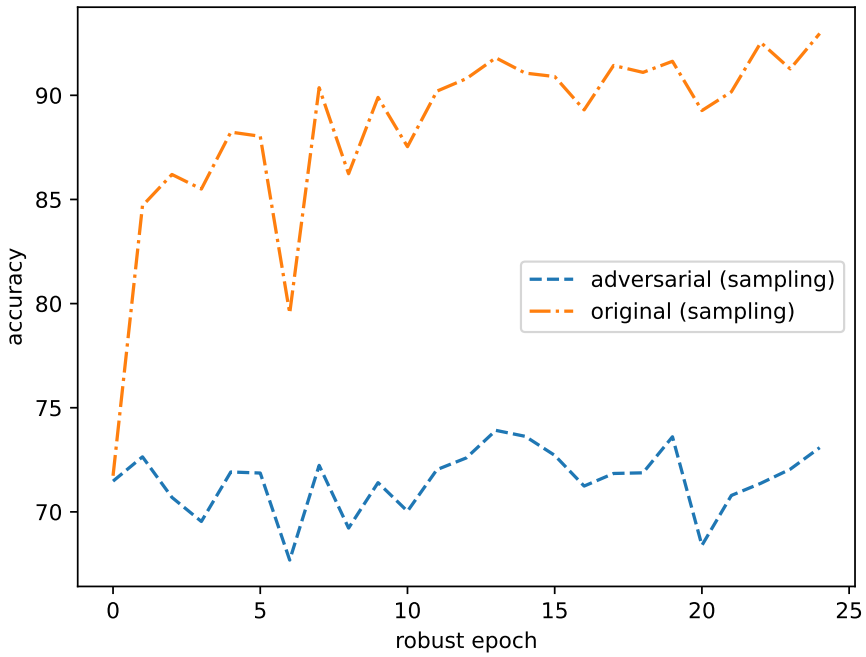


Figure C.1: Reverse case of the robust training loop

The goal here is to ensure that it is indeed the training with the adversarial images that is useful and not simply the fact of adding new, never seen before, images.

As we can see, the basic training with adversarial images can directly provide good original accuracy. When we start adding new samples, the original accuracy is increased. It is expected since more samples will always tend to give better performance. What is interesting here is that the adversarial accuracy does not show a clear increase as in the previous tests. We can therefore conclude that it is indeed the addition of the adversarial images that can help to improve robustness.

C.2 Full adversarial training

In this case, we used all our generated adversarial samples to perform adversarial training. In addition, we also used all the original samples and not just $10,000$. We also add $1,000$ adversarial samples instead of 500 at each robust epoch.

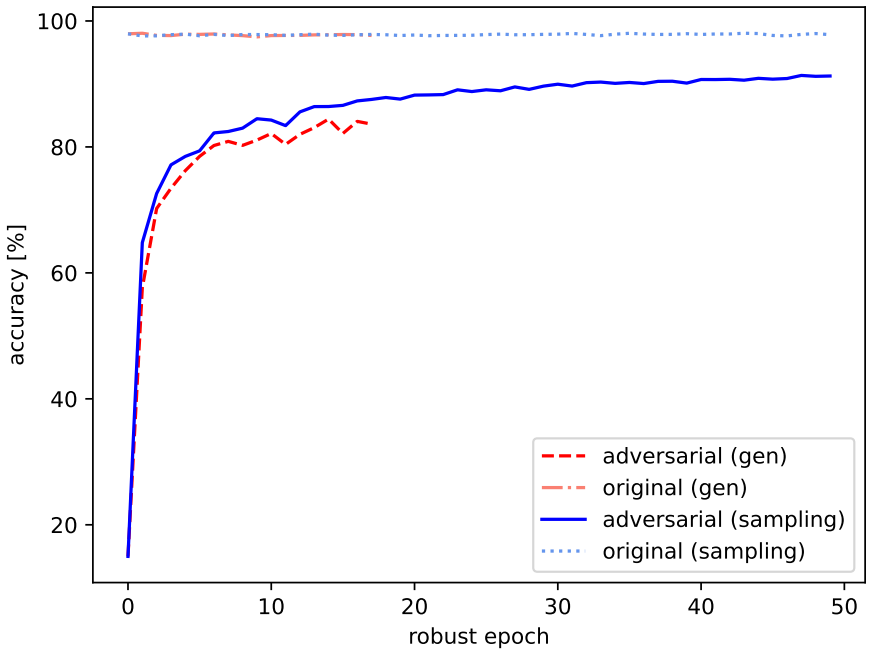


Figure C.2: Full adversarial training for MNIST with $\epsilon = 0.5$

We can see that the adversarial accuracy keeps increasing for each epoch. As before, the generation approach gives lower results as we have discussed in the thesis.

An important point to address here is that the generation approach is more limited than the adversarial sampling one. Indeed, as the model becomes more and more robust at each robust epoch, it becomes more and more difficult to generate new adversaries on the 5 BNNs that constitutes it. Therefore, we will reach the end of the training dataset very quickly.

C.3 Lower perturbation value cases

Here, we show the results for the robust training loop with lower perturbation values.

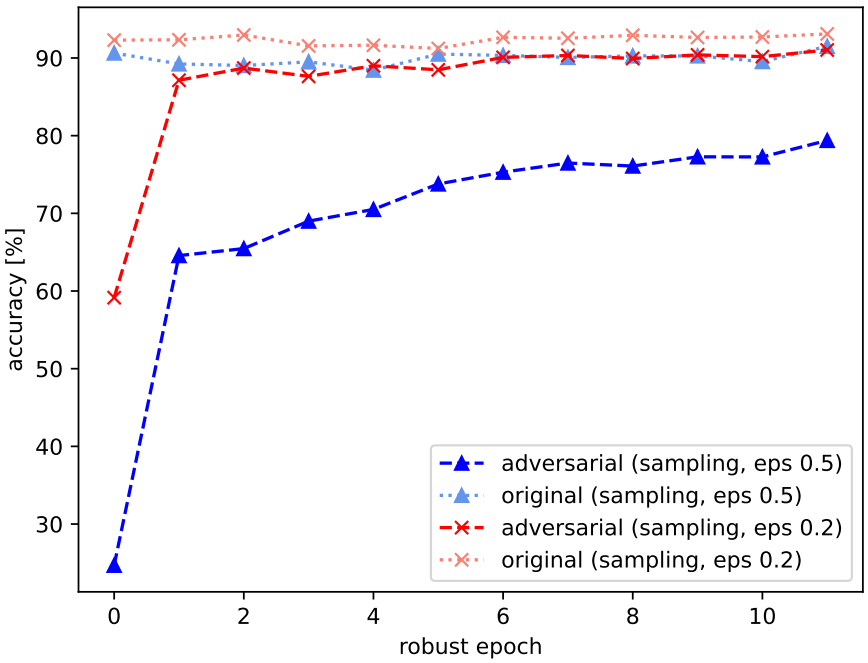


Figure C.3: Robust training loop for MNIST with $\epsilon = 0.2$

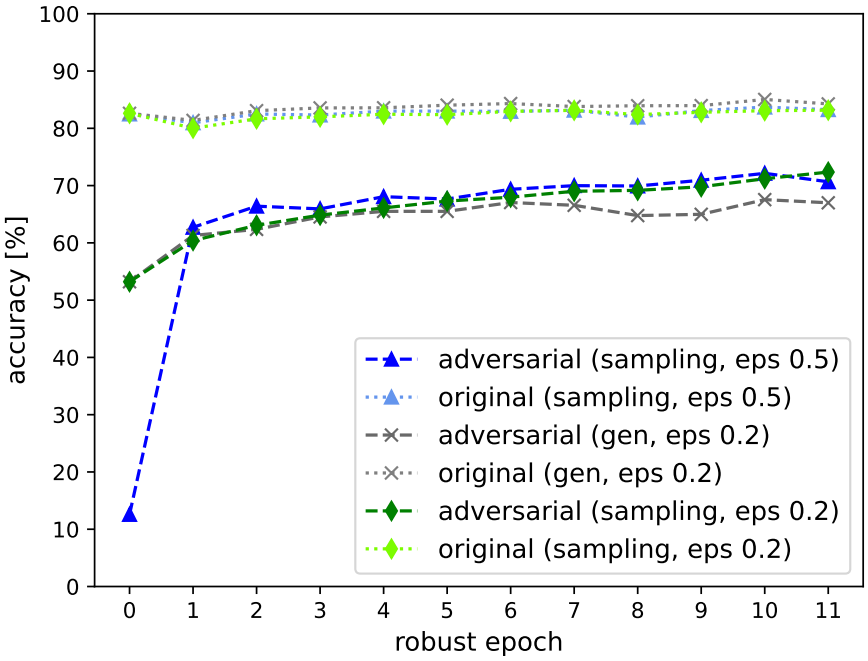


Figure C.4: Robust training loop for Fashion-MNIST with $\epsilon = 0.2$

In the MNIST case, the adversarial accuracy is far greater with $\epsilon = 0.2$. It tends to make sense as the images are less disturbed and therefore easier to classify by the models.
However, for the Fashion-MNIST dataset, the difference is only for the first robust epoch. Then, the observed increase is slower than the case with $\epsilon 0.5$. It explains why we used mainly this value in the thesis even if the images are less "readable".

C.4 Robust loop with type 4

We show here the results of a robust training loop using a majority voting model made of five type 4 BNNs.

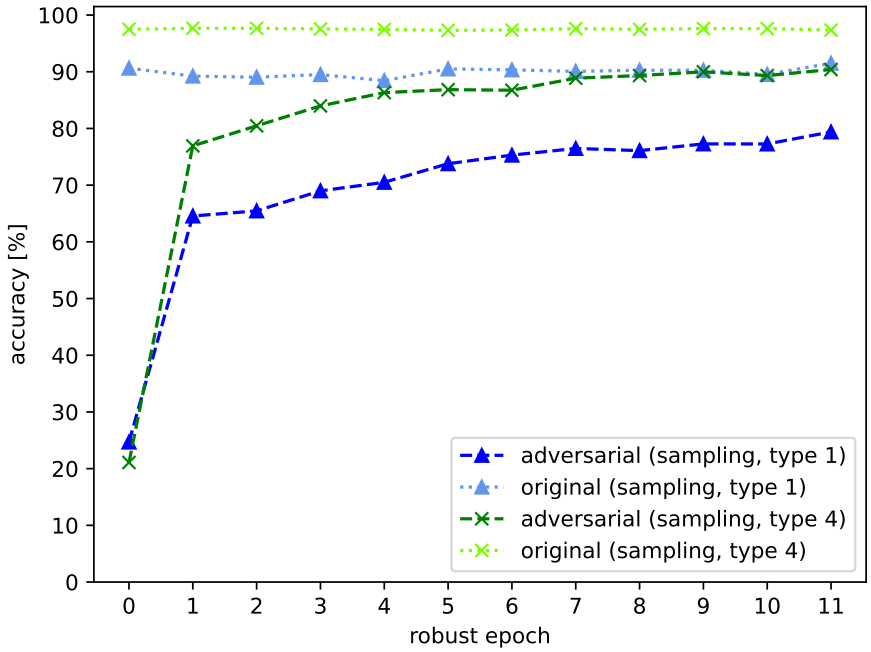


Figure C.5: Robust training loop with type 4 BNNs

We can see that the adversarial accuracy with type 4 BNNs can go even higher than with the type 1, even reaching their original accuracy. This makes sense as the model is more complex. However, the cost for this result is the higher time needed to train the model.

Appendix D

Attacks against type 4 BNN

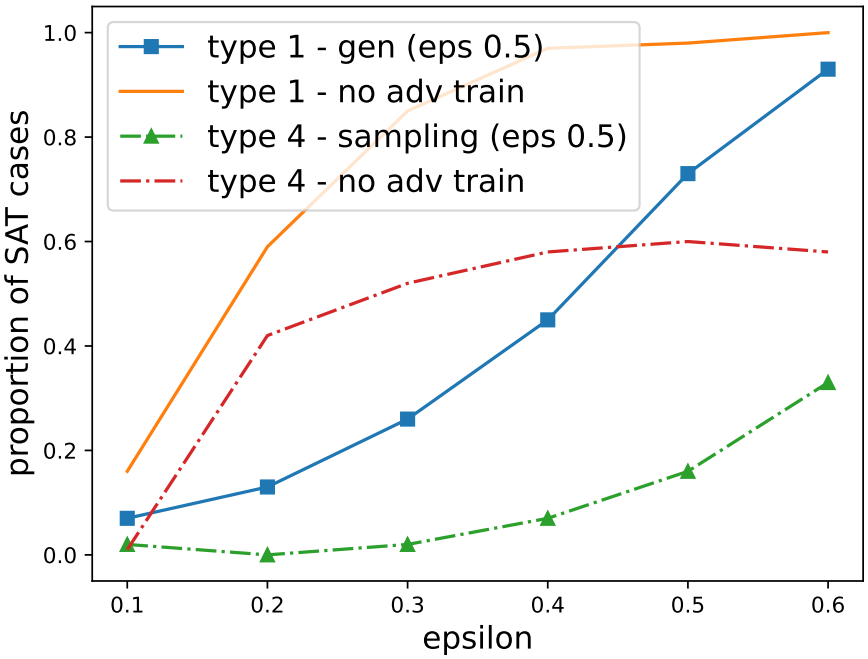


Figure D.1: Comparison of the behavior of the different types of BNN against attacks with a given perturbation range on MNIST

We can observe that the type 4 is once again more robust and that it is even more the case after even an adversarial training with the adversarial sampling approach.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**