Heriot-Watt University

Masters Thesis

Generative versus logical training against adversarial attacks

Author:
Marco Casadio

Supervisor:
Dr. Ekaterina Komendantskaya

A thesis submitted in fulfilment of the requirements for the degree of MSc.

in the

School of Mathematical and Computer Sciences

August 2020

Declaration of Authorship

I, Marco Casadio, declare that this thesis titled, 'Generative versus logical training against adversarial attacks' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Marco Casadio

Date: 26/04/2020

Abstract

The last decade has seen a growing trend in using Machine Learning algorithms and in particular Neural Networks, for the most varied applications. Nowadays, it is not unusual to come in contact with this technology in both commercial and personal environments.

This broader use brought to attention a particular property of this technology: robustness. Machine Learning algorithms notoriously present low robustness and can be highly sensitive to small inputs modification.

The science that deals with adversarial examples and aims to improve robustness is called verification. Most of the verification techniques are based on placing constraints on inputs after the algorithms are already trained.

However, those techniques are too slow and the approach does not scale to big networks. That is why in the latest years researchers focused more on introducing verification constraints into training.

This project aims to better understand whether and how verification can be integrated with learning procedures and evaluate state-of-the-art methods for verification that allows translating logical constraints into a loss function.

Acknowledgements

I would like to thank my project supervisor Dr. Ekaterina Komendantskaya for dedicating her time to guide and advise me on this project, and both her and Prof. Nick Taylor for the feedback provided on my report. I also would like to thank the members of LAIV for sharing their knowledge and competence and introducing me to the subject of verification of machine learning.

Contents

List of Figures

List of Tables

Abbreviations

DL	**D**eep **L**earning
DL2	**D**eep **L**earning **w**ith **D**ifferentiable **L**ogic
DNN	**D**eep **N**eural **N**etwork
GAN	**G**enerative **A**dversarial **N**etwork
ML	**M**achine **L**earning
NN	**N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork

Chapter 1

Introduction

1.1 Motivation

The last decade has seen a growing trend in using Machine Learning (ML) algorithms for the most varied applications. Nowadays, it is not unusual to come in contact with this technology in both commercial and personal environments. This broader use brought to attention a particular property of this technology: robustness.

According to Serban et al. [2019], a robust solution is one that has the ability to perform well under a certain level of uncertainty.

Neural networks (NNs) are a set of algorithms modeled after the human brain, that are designed to recognize patterns. Mathematically, the simplest form of a neural network, also called a Perceptron, is given by a linear function of n inputs:

$$f(X_1, \ \dots \ , X_n) = w_0 + w_1 \times X_1 + \dots \ w_n \times X_n$$

Deep Neural Networks (DNNs) compose such functions, achieving more complex computational properties. Machine Learning algorithms (of which DNNs belong) famously present low robustness and can be highly sensitive to small, possibly intentional, inputs modification. Such inputs are called adversarial examples (2).

Although this characteristic is common in general for all ML algorithms, people worry the most about DNNs. This is mainly for two reasons: there is so much hope around

this technology and their complexity makes it very hard to analyze and understand what is wrong.

Because adversarial examples may be intentional, a security issue needs to be taken into consideration. Overall, adversarial examples are mainly important for two reasons:

- An attacker might exploit them.
- They show DNNs are not robust, meaning that their outputs can be wrong when it is least expected.

For these reasons, adversarial examples can potentially lead to catastrophic consequences.

Over the years many different approaches have been developed to prevent these attacks but none of them has proven to be effective enough. Indeed Carlini and Wagner [2017] evaluated different adversarial detection methods and proved that none of them efficiently detect adversarial examples and that most detectors perform very poorly when faced with powerful iterative attackers.

Verification techniques are often based on checking constraints satisfaction of inputs a posteriori. But recently, several researchers advocated the idea of verification-driven learning, that incorporates verification constraints as loss functions into the training algorithms.

We can argue that it is a better approach because we want the network to be independent. If we are checking that inputs satisfy constraints at test time and the check results negative, either we need human intervention or we have to discard those inputs. That does not feel like the network is robust. On the other hand, if the verification constraints are embedded in the loss, the training should make the network more robust and be able to autonomously handle all sorts of inputs.

Fischer et al. [2019] developed a new system called DL2 (Deep Learning with Differentiable Logic) that introduces a new method to train networks to meet logical specifications in a declarative way. An important feature is its ability to train with constraints that place restrictions on inputs outside the training set, while most prior work focuses on local constraints.

Past work with verification constraints as loss functions have also other limitations: the gradient can be zero, non-linear constraints cannot be expressed, or is intractable for large constraints in many variables. DL2 claims to overcome these limitations and is the aim of this thesis to verify that it gives state-of-the-art results against adversarial attacks.

1.2 Aims and Objectives

The aim of this research is to better understand whether and how verification can be integrated with learning procedures. In particular, to understand the capacities and limitations of DL2, one of the state-of-the-art systems for embedding verification constraints into ML algorithms.

Our research hypothesis is that DL2 has gaps and it is not definitive, thus it would admit improvements in the future.

To achieve our aim, we define different objectives:

- to systemically evaluate DL2, using the benchmark data sets: MNIST [Yann, 2013], Fashion MNIST [Xiao et al., 2017] and German Traffic Sign [Stallkamp et al., 2011].
- to verify that DL2 [Fischer et al., 2019] is performing significantly better on constraints training while not exhibiting a remarkable loss in prediction accuracy.
- to evaluate its performance against an AC-GAN network.
- to evaluate DL2 against a network trained on adversarial examples.
- to make general conclusions about the robustness of DL2 networks against adversarial attacks.
- to eventually implement our own methodology/software to perform training and verification simultaneously.

1.3 Methodology Overview

Due to the subtlety of the planned work and evaluation, we had to choose our benchmarks and baseline comparisons carefully. We have several tiers of baseline architectures, to allow for fair, informative and systematic evaluation.

Convolutional architectures with standard loss functions, trained on the standard data sets are an obvious first level of baseline comparison. Further, several methods exist in the literature that train standard architectures on adversarial data sets. This is our second level of baseline comparison. Finally, there are methods like DL2 (but also PLS [Kimmig et al., 2012], PaRoT [Ayers et al., 2020]) that modify the loss functions when training against attacks. These are the methods that we will compare against the two baselines.

We will compare and test the above-mentioned methods on 3 different benchmark data sets: MNIST, FASHION and GTSRB.

We will have 12 networks in total, 4 for each data set: 1 baseline network trained on the standard data set with a standard loss function, 2 networks trained on adversarial data sets with a standard loss function and 1 network trained with DL2 loss on the standard data set.

The first of the two adversarial data sets will be generated through an AC-GAN (a state-of-the-art-method), while the second will be created with a script developed by myself.

This methodology will be explained in more detail in Chapter 3.

1.4 Results Overview

The experiments show that DL2 does not improve robustness against the majority of the tested attacks. This is the main result of this thesis and it is a significant discovery because, in the last few years, methods of constraint-driven training such as DL2 (but also Hu et al. [2016], Xu et al. [2017], Ayers et al. [2020]) have been regarded by the machine learning community as the most promising novel solution to the problem of

adversarial vulnerability of machine learning models. This thesis questions the feasibility of and grounds for this research direction.

Although we do recommend that further studies are conducted to confirm our conclusions, we made all efforts to show that our conclusions are replicable with several models, attacks methods, data sets and training environments. In particular we show that DL2 does not produce robust networks for the following.

AC-GAN Attacks DL2 performs similar to the baseline network against AC-GAN attacks, while being significantly worse than the AC-GAN trained network consistently in all data sets and models (Table 4.2).

FGSM Attacks We can see (Figures 4.1, 4.2) that DL2 performs worse than both the baseline networks and the networks trained on adversarial examples against FGSM attacks. However, DL2 seems to slightly improve robustness against the FGSM attacks on the GTSRB dataset from a certain point onward (Figure 4.3).

Summarizing, DL2 does not significantly drop in prediction accuracy and it performs outstandingly good in constraint accuracy (intended as the percentage of inputs that satisfy the chosen DL2 constraint), but nonetheless it does not produces robust networks.

It is beyond the scope of this thesis to propose a novel U-turn direction for the machine learning community. However, we may conjecture that the best interpretation of our results leads to the realisation that, although methods that augment loss functions with constraints can be used to fine-tune neural networks, they cannot replace proper verification methods when it comes to guaranteeing safety and security of machine learning systems.

Chapter 2

Literature Review and Background

In this chapter, we will present and discuss the related work, starting with defining what is AI and what it means for us and finishing with a detailed explanation of the tools that will be used in this dissertation. We will also explain and discuss other fundamental concepts as Adversarial examples and Verification.

2.1 AI and Machine Learning

Artificial Intelligence (AI) is an umbrella term for several research domains in computer science and engineering that work on methods and applications which simulate human intelligence. Machine Learning (ML) is one of its sub-fields. In particular, ML are algorithms that improve and learn automatically from past data or experiences. In this research we will only consider Machine Learning, and AI will be treated as a synonym of ML.

In the field of Machine Learning there are four major learning paradigms:

> **Supervised learning** is the ML task of learning patterns, functions that map inputs to outputs. That is accomplished by feeding into the algorithm a set of paired inputs and desired outputs. The algorithm then learns a function that can

be used for mapping new examples. The most suited tasks for supervised learning are pattern recognition (classification) and regression (function approximation).

Unsupervised learning is the type of machine learning that, differently from the previous one, learn patterns in a data set with no labels. This type of learning aims to group training data into clusters, by looking at inputs' features similarity. The most common Tasks for unsupervised learning are clustering, estimation of statistical distributions, compression, and filtering.

Semi-supervised learning is an approach to machine learning that falls between the two previously mentioned types. Indeed a small amount of labeled data (as in supervised learning) and a large amount of unlabeled data (as in unsupervised learning) are combined during training. Semi-supervised learning is used in situations where the cost of labeling enough data is unfeasible while the acquisition of unlabeled data is relatively inexpensive. That is because unlabeled data, when used together with a small amount of labeled data, can produce a considerable improvement in learning accuracy.

Reinforcement learning is the area of machine learning concerned with how software agents will take actions in an environment in order to maximize a reward. Reinforcement learning (RL) doesn't make use of labeled data, like unsupervised learning. RL algorithms focus on finding a balance between exploration and exploitation. Reinforcement learning is mainly used in tasks such as vehicle routing, video games, natural resource management, and medicine.

While we started from the very broad concept of AI and narrowed down to the learning paradigms of ML, in this research we will only consider supervised learning because ours is a task of image classification.

2.2 Neural Networks

This section will talk about Neural Networks to introduce a particular type called Convolutional Neural Networks that will be used in this project.

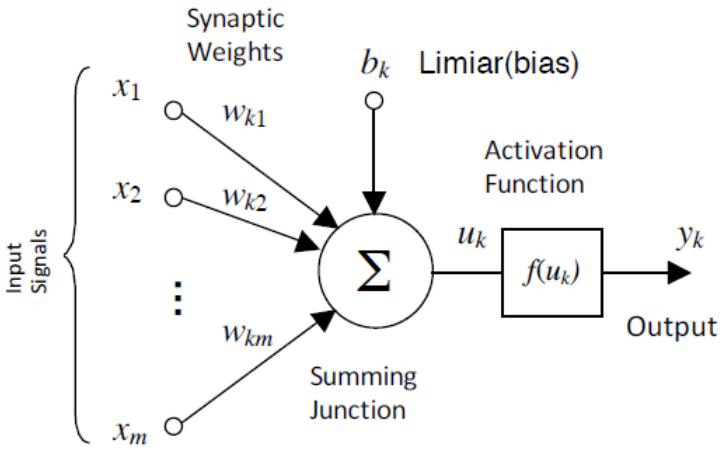

FIGURE 2.1: A McCulloch-Pitts artificial neuron. Vargas [2018]

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

FIGURE 2.2: Most used activation functions. Vargas [2019]

In the introduction, we already mentioned the view on NNs as linear functions. We now illustrate how the first and most famous NN model, the Perceptron [Rosenblatt, 1958], models a linear function.

Figure 2.1 shows the structure of an artificial neuron k:

- $x_1 ... x_n$ are the inputs. In the case of an image, they correspond to the pixels.
- $w_{k1} ... w_{km}$ are the weights. They will be modified as the neuron learns.
- b_k is the bias of the neuron. This is introduced to shift the decision boundary away from the origin, because it can cause problems with calculations, and does not depend on any input value. The bias can also be aggregated to the weights, giving it an input always equal to 1.
- u_k is the sum of all the inputs multiplied by their respective weight, plus the bias: $u_k = \sum_{j=1}^{m} w_{kj} x_j + b_k$.
- $f(u_k)$ is the activation function. This is a linear function that determines the output of the neuron y_k. In figure 2.2 are represented the most used activation functions.

To calculate the output of a neuron k we use the formula:

$$y_k = f(u_k) = f(\sum_{j=1}^{m} w_{kj} x_j + b_k). \tag{2.1}$$

Artificial neurons can be combined together in layers. A single-layer perceptron is the most basic Neural Network and consists of an input layer and an output layer [Figure 2.3].

A multi-layer perceptron (MLP) is a Neural Network with at least one hidden layer, meaning with at least one layer between the input and the output layers. MLPs are also called Deep Neural Networks (DNNs).

The training process for a DNN can be divided into two main steps:

Feed-forward: the first step is forward propagation. When multiple perceptrons are combined in a Neural Network, each output neuron operates independently of all the others. Thus, calculating each output can be considered in isolation using formula 2.1 and combining them in a vector of outputs. This procedure is repeated for each layer, starting from the first and arriving at the output of the NN.

Back-propagation: invented by Werbos [1975], this is the algorithm that allows a Neural Network to modify and adjust its parameters, namely weights and biases, in order to reduce the error, or in other words the number of wrong predictions.

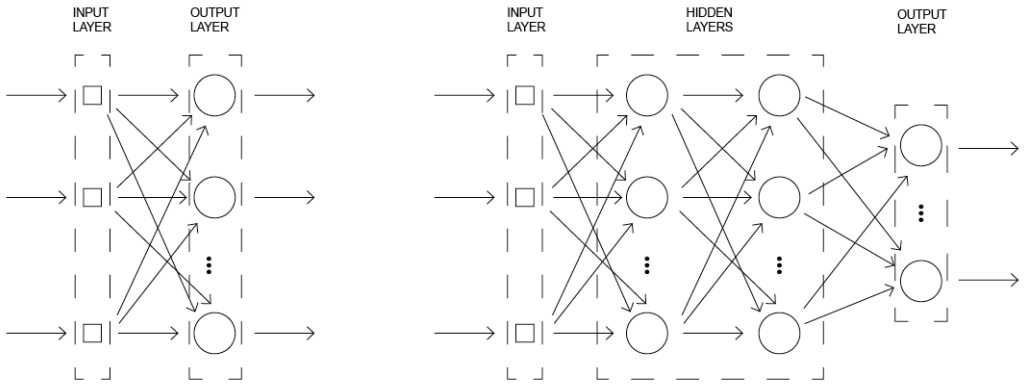

FIGURE 2.3: On the left is a single-layer perceptron. On the right is a multi-layer perceptron.

To quantify this error we introduce the loss function. This function calculates the difference between the network output and its expected output after a training example has passed forward through the network.

Back-propagation calculates, for each neuron in a layer, the gradient of the loss function. Then weights and biases are updated via an optimization algorithm that tries to minimize the gradient.

This process is done starting from the output layer and propagates back to the first layer, thus the name back-propagation.

Training a Neural Network means to apply these two steps multiple times consecutively, where each time is called an epoch. To measure if a network is learning, the accuracy metric is utilized. The learning process can be considered complete when the accuracy calculated on the test set is not improving anymore.

Deep Neural Networks have proven over the years to be the best Machine Learning algorithm for the task of Image Classification. Furthermore, the software we are going to evaluate (DL2) is built upon their use. For those reasons, we will utilize them, and in particular an evolution of DNNs called Convolutional Neural Network (CNN) that is introduced in the next section.

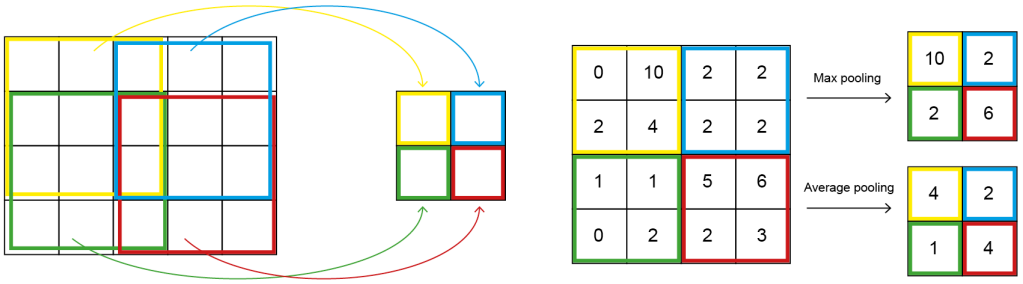

FIGURE 2.4: On the left is a convolution with a 3x3 filter and a stride of 2. On the right are represented the two main implementation of pooling extraction with a 2x2 filter and a stride of 2.

2.3 Convolutional Neural Networks

As previously anticipated, Convolutional Neural Networks are a particular form of Deep Neural Networks. They are most commonly applied to analyzing visual imagery. Indeed CNNs are currently the state of the art for the task of detecting what an image is (also known as image classification or recognition), or what is contained in the image (also known as object detection).

Since ours is a task of image classification, they are relevant and will be used in this project.

Since CNNs are still DNNs, it means they are made of multiple layers connected in sequence. But their singularity can be found in the name: they make use of special layers called convolutional layers.

A Convolutional Neural Network is composed of different types of layers, we can divide them into three main types.

Convolutional layers: The convolutional layer is the core building block of a CNN. Its parameters consist of a set of learnable filters.

Convolution is a specialized kind of linear operation that from two functions produces a third one. In a convolutional layer, this mathematical operation is achieved by sliding a filter over the inputs. This produces a 2-dimensional activation map of that filter (2.4 left).

As a result, the network learns filters that activate when they detect a specific feature in the input. This is called feature extraction.

The output of the convolutional layer is formed by the activation maps for the filters.

Pooling layers: Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. A pooling layer is placed after one or more convolutional layers and it serves to progressively reduce the spatial size of the representation.

Pooling is applied on the input feature map and, similarly to the convolutional layer, a filter is used to divide it into smaller regions.

There are several non-linear functions to implement pooling. The most commons are max pooling where only the highest value of a region is kept and average pooling where the output represents the average in the region (2.4 right).

Fully connected layers: They are also called dense layers. As the name suggests, neurons in a fully connected layer have connections to each and all the neurons in the previous layer.

DNNs or MLPs are usually a sequence of fully connected layers. Differently, CNNs consist of an input layer, an alternating series of convolutional and pooling layers, and one or more dense layers, the last of which is the output layer.

The activation function for the convolutional and fully connected layers is commonly a Rectified Linear Unit (ReLU). Often, ReLU is preferred over other functions because it speeds up the training process without loosing much on accuracy, but we can sometimes see other activation functions like Hyperbolic Tangent (Tanh).

However, for the output layer, the activation function used is the Softmax function. This function normalizes the input vector into a probability distribution. Before softmax, some vector components can be smaller than zero, or bigger than one and might not sum to 1. However, after applying softmax, each component of the vector will be in the interval (0,1), and they will add up to 1 so that they can be interpreted as probabilities. These activation functions can be seen in figure 2.2.

In this project we are going to utilize the two architectures used in the DL2 paper so we can compare the results: For the MNIST and FASHION-MNIST datasets we will use a CNN with 6 convolutional layers followed by 2 linear layers, while for the GTSRB and CIFAR10 datasets we will use the ResNet18 architecture [He et al., 2015].

The first architecture is really small but achieves state-of-the-art results with the two cited datasets because they are really simple. The ResNet18 is a residual neural network (ResNet). That means a CNN with the possibility to skip connections by jumping over some layers. This has been proven to speed the learning and ResNets are the state-of-the-art CNNs for the more complex datasets.

If DL2 proves to be effective, future experiments should be made with bigger datasets and bigger networks.

2.4 Adversarial Examples

This section formally introduces and explains different views on the subject of Adversarial Examples and gives its definition in regards to this research.

The term adversarial example was first used by Szegedy et al. [2013]:
"Consider a state-of-the-art deep neural network that generalizes well on an object recognition task. We expect such network to be robust to small perturbations of its input, because small perturbation cannot change the object category of an image. However, we find that applying an imperceptible non-random perturbation to a test image, it is possible to arbitrarily change the network's prediction. These perturbations are found by optimizing the input to maximize the prediction error. We term the so perturbed examples adversarial examples."

Serban et al. [2019] provide a formal definition of adversarial examples. Given a classification function f and a sample X which gets correctly classified by f with the label l. An adversarial examples X' is constructed by applying the minimal perturbation η to the input X such that it gets classified by the model with a different label l'.

$$\min_{X'} \|X' - X\|_p,$$

$$s.t. \;\; f(X') = l',$$

$$f(X) = l,$$

$$l \neq l',$$

$$x' \in [0,1]^m,$$

FIGURE 2.5: Adversarial examples with very low perturbation. For both images, the pictures in the first column are correctly classified inputs. The pictures in the last column are adversarial examples, crafted to cause a misclassification. In between, the perturbation used to cause the misclassification. Szegedy et al. [2013]

where $\|.\|_p$ is the distance between the samples (p-norm) and $X' - X = \eta$ is called a perturbation.

So according to these definitions, an adversarial example must have some characteristics:

- It is constructed. Although this is not always important, for example the intentionality is meaningless for local robustness (refer to section 2.6).
- The perturbation must be very small, such that a human cannot distinguish it.
- The DNN must classify it with the wrong label.

However, there are other arguments to be made. Unlike humans, machines cannot distinguish between big or small perturbations, so on occasions where there isn't a human observer this constraint is not relevant for an attacker. In these cases, adversarial examples are just crafted inputs that get classified with a wanted label.

Nevertheless, minimal perturbations become important again when the property of robustness is discussed. There are real-life scenarios, for example self-driving cars, where small changes in distributions should not drastically impact the behavior of a model. If the car misclassifies a street sign for another it can lead to disaster. And it does not matter if the reason is that someone put a sticker on the sign or it's a bit corroded from the rain or a tree covers it a little.

As seen, depending on the situation, adversarial examples can have different definitions and properties. There are two major fields that treat adversarial examples:

Security: adversarial examples can be used to attack neural networks in regard to an objective that must be defended. There are mainly three targets for the attacker:

- Confidence reduction. Where an attacker can reduce the output confidence score of a classifier, thus introducing class ambiguity.
- Random misclassification. Where an attacker modifies an input in order to output any class different than the correct one.
- Targeted misclassification. Where an attacker modifies an input in order to output a specific, target class.

In this context, I believe that the best definition of adversarial examples is the one provided by Serban et al. [2019], although sometimes the minimalism of the perturbation can be omitted.

Verification: this field is more concerned with the property of robustness, defined as the "insensitivity of an algorithm to small deviations from the underlying assumptions" [Serban et al., 2019]. Therefore minimal perturbations are fundamental while intentionality is not anymore.

Since our aim is verification, we will say that in our scope an adversarial example is an input that gets classified with a wrong label and it comes either from the real distribution or it is crafted with a very small perturbation. In other words, if a human would correctly classify it but the network doesn't.

2.5 Explainability of AI

This paragraph introduces an AI technique that is often paired with Verification and can be useful to understand network predictions.

Explainable AI (Artificial Intelligence) refers to methods and techniques used to make predictions and solutions of AI models understandable by human experts. While Machine Learning is at the core of many recent advances in science and technology, the

majority of ML models are black boxes, meaning that it is not possible to explain why the model arrived at a specific solution. This challenge of explaining AI decisions is also known as the interpretability problem. If users cannot understand why a model came to a specific decision, then they may not be able to trust it.

There are cases when trust in the models is important, for example, a doctor will not operate a patient just because the model said so. But even in not at hazard situations, like when choosing a TV series, some trust is required to follow AI recommendations. Understanding the rationale behind model predictions would certainly help users decide when to trust them or not. When an engineer uploads an ML model to production, he is implicitly trusting that it will make sensible predictions. Usually, such assessment is done by looking and considering the accuracy of the model. However, a model can achieve high accuracy and correctly predict the majority of the inputs while doing it for completely wrong reasons. AI explainability comes into play to help understand those reasons and can be a fundamental tool when deciding if a model is trustworthy or not.

The first approaches to this topic [Louppe, 2014] were focused on what was at that time the most efficient algorithms: Random Forests. But as more powerful algorithms like neural networks have emerged, a new generation of explainability techniques suitable to any machine learning model appeared.

A state-of-the-art approach is LIME (Local Interpretable Model-Agnostic Explanations) [Ribeiro et al., 2016]. Since LIME is model-agnostic, it cannot peak into the model. Thus, in order to achieve its goal to interpret models, it perturbs inputs around their neighborhood and sees how predictions change. It then weights these perturbed data points by their proximity to the original input and learns an interpretable model on those and the associated predictions. LIME is a powerful tool and in the example 2.6 are shown some of its capabilities.

LIME is often seen as the initial comparison for researchers in this field. It is the starting point to implement new tools for AI explainability. After LIME many other systems have been created that are the state-of-the-art. To name a few, we find DeepLIFT [Shrikumar et al., 2017], Layer-Wise Relevance Propagation [Binder et al., 2016], and SHAP [Lundberg and Lee, 2017].

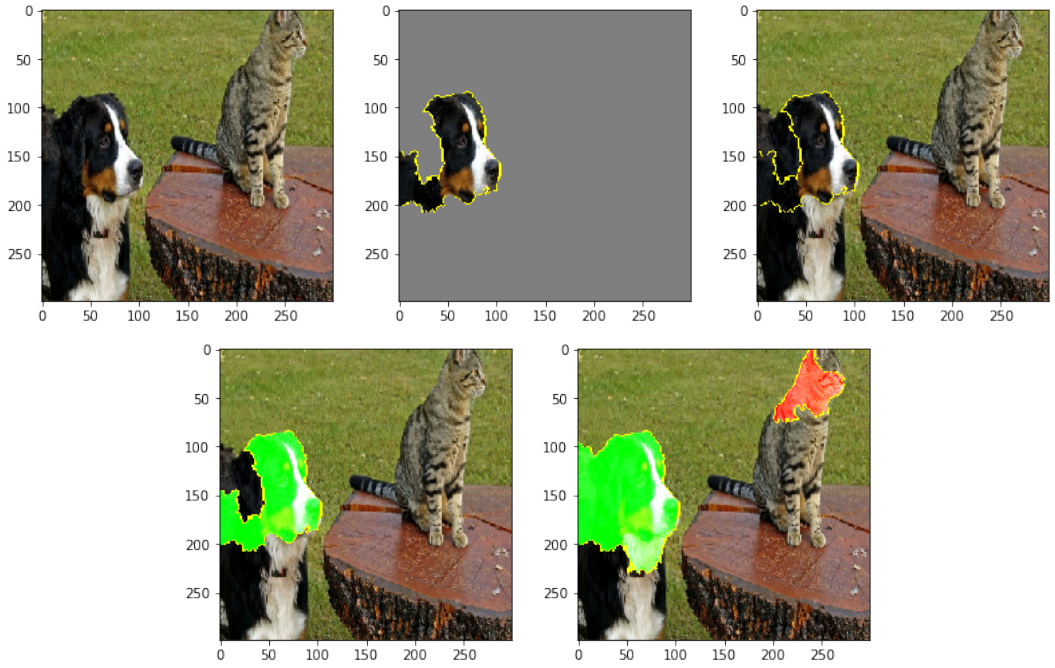

FIGURE 2.6: In this example we have an image of a cat and dog. The model predicts the label 'dog'. We use LIME to interpret the prediction. Top-left: the original image. Top-center: the part of the image responsible for the prediction according to LIME. Top-right: the same as the previous but with the full image. Bottom-left and bottom-right: in green is the part of the image that positively contributed the most for the prediction of 'dog'. In red is the part of the image that negatively contributed the most for the same prediction. Ribeiro et al. [2016]

AI explainability is fundamental to understand ML models but it is not a tool for verification in itself. However, being able to interpret the behavior of a Neural Network can ease the task of verification and can help to see why and where the model is not robust. Indeed, with the knowledge of which areas of the picture are important and which are latent, we can inform our adversarial attacks or defenses.

Explanation and verification are often heard together and, even if explainability is out of scope in this project, it is worth talking about as it can be integrated into future works.

2.6 DL2

This section goes deep into explaining the DL2 system, which is the main subject of this research. It introduces previous work and why DL2 is an improvement. It follows then with its different features and with a discussion of the ones which are of interest in this project.

2.6.1 Previous work

Over the years, several verification techniques have been developed in order to make Machine Learning algorithms more robust. While some of them can assure some robustness for some input perturbations [Huang et al., 2016], the problem is yet to be solved and the challenge is still open. Verification techniques are often based on checking constraints satisfaction of inputs a posteriori or at test time. For example is very common to use SMT (satisfiability modulo theories) solvers for verification like Z3 [de Moura and Bjørner, 2008].

Although SMT solvers can be set up to do full verification, they are way too slow and the approach does not scale to big neural networks [Katz et al., 2017]. That is why in the last few years researchers started to think that is might be more efficient to introduce verification constraint into training, rather than verifying the networks after training. Indeed, this research aims to better understand how to ingrate verification within learning procedures.

DL2 (Deep Learning with Differentiable Logic) is hereby of great interest because it introduces verification constraints via loss functions at training time.

However, Fischer et al. [2019] is not the first to apply verification methods in training. Kimmig et al. [2012] built the framework PLS (Probabilistic Soft Logic) that translates logical constraints into loss functions. The problem is that with this method the gradient may be zero, making the use of this loss to find satisfying assignments useless.

Another approach is from Hu et al. [2016], that expanded on PSL and present a framework that introduces rules into the training phase. They formulate rule satisfaction as a convex problem with a closed-form solution. But their framework also has problems. Indeed, their formulation is restricted to rules over inputs and output classes, and cannot express rules over output's numerical values. Furthermore, this method cannot express non-linear constraints since the convexity and existence of a closed-form solution derive from the linearity of the rules.

Xu et al. [2017] as well developed a methodology for using symbolic knowledge in deep learning through a loss function. But their work, like for Hu et al. [2016], is restricted to constraints over output classes and does not scale well for large constraints with many variables.

DL2 supposedly overcomes these limitations and is our task is to confirm or deny its efficacy.

Framework	Type	ZeroGrad	Speed	Scale	Expressibility
SMT	Black Box	No	Slow	Small	-
PLS	White Box	Yes	Medium	Medium	Medium
Hu et al.	White Box	No	Medium	Large	Low
Xu et al.	White Box	Yes	Fast	Small	Medium
PaRoT	White Box	No	Fast	Large	Medium
DL2	White Box	No	Medium	Large	High

TABLE 2.1: DL2 and previous works comparison.

2.6.2 Features

Fischer et al. [2019] describes its work as

> " ... a system for training and querying neural networks with logical constraints. Using DL2, one can declaratively specify domain knowledge constraints to be enforced during training, as well as pose queries on the model to find inputs that satisfy a set of constraints. DL2 works by translating logical constraints into a loss function with desirable mathematical properties. The loss is then minimized with standard gradient-based methods. We evaluate DL2 by training networks with interesting constraints in unsupervised, semi-supervised and supervised settings. Our experimental evaluation demonstrates that DL2 is more expressive than prior approaches combining logic and neural networks, and its loss functions are better suited for optimization. ... "

As we can see, DL2 introduces two main features:

- A way to train networks to meet logical specifications.
- A way to query networks for inputs meeting logical constraints.

While the query method is powerful to inquire about a network decision making, it is not useful for verification thus it is not of interest in this project. This thesis will focus solely on DL2 capability of embedding logical constraints into training to produce more robust networks.

DL2 can express powerful and various combinations of constraints over inputs, neurons, and outputs of NNs and allows to translate them into a non-negative loss with two key properties:

- The loss is zero exactly if the constraints are satisfied.
- The loss is differentiable almost everywhere.

These two properties combined allow training with constraints by minimizing a loss with off-the-shelf optimizers. Indeed, the optimizer used is PGD (projected gradient descent), which has already been shown successfully in training with robustness constraints [Madry et al., 2017]. The expressiveness of DL2 along with tractable optimization with PGD enables training with powerful and interesting constraints [Fischer et al., 2019].

Of important relevance is DL2's ability to express constraints that place restrictions on inputs outside the training set. This can be used for both querying and training a network. Most of the related previous work (e.g, Xu et al. [2017]) focuses on placing constraints locally within the training set.

Nonetheless, Fischer et al. [2019] defines both a local ($Robustness^T$) and a global ($Robustness^G$) robustness constraint which will be analyzed a little further on.

2.6.3 From Logical Constraints to Loss

The language for logical constraints created by Fischer et al. [2019] involves boolean combinations of comparisons between terms. Terms can be inputs, neurons, and outputs of neural networks.

Indeed, the building blocks for a constraint are the following comparisons: $t = t'$, $t \neq t'$, $t \leq t'$, $t < t'$. Where t and t' are two terms.

A constraint φ can be, then, one of the followings:

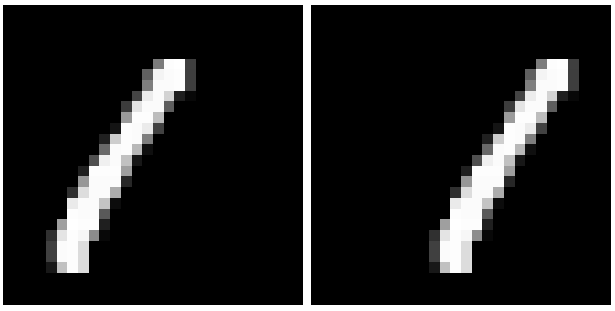

FIGURE 2.7: Two images representing the same '1' from the MNIST dataset moved out of center in different positions.

- A comparison of two terms (one of the four above).
- A conjunction $\varphi' \wedge \varphi''$.
- A disjunction $\varphi' \vee \varphi''$.
- A negation $\neg\varphi$.

Where φ' and φ'' are constraints.

Each constraint φ is associated to a non-negative loss function $L(\varphi)$ such that $L(\varphi)$ is differentiable almost everywhere and $L(\varphi) = 0$ iff φ is satisfied.

2.6.4 Training

They evaluate DL2 on different tasks: supervised, semi-supervised, and unsupervised learning. This project will only evaluate supervised learning and is not interested in the other tasks but future work can be expanded on them.

As anticipated, they consider two types of constraints:

> **Local constraints** where all variables refer to samples from the training set.
>
> **Global constraints** which have (possibly more than one) universally quantified variable z.

Among the constraints they define, we are interested in the ones that can be used for improving robustness: a local robustness constraint that they call $Robustness^T$ and a global robustness constraint which they call $Robustness^G$.

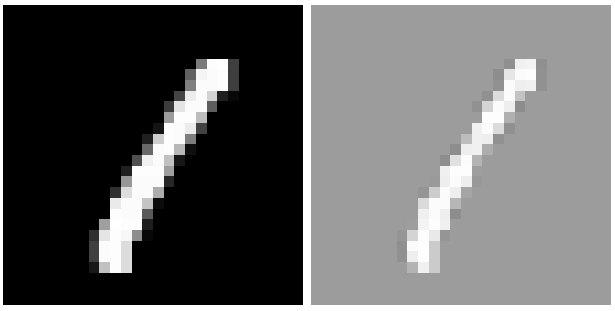

FIGURE 2.8: Two images representing the same '1' from the MNIST dataset but the second has the background changed.

For local robustness they inspire from Szegedy et al. [2013] and define the constraint:

$$\| X - X' \|_2 < \epsilon_1 \implies KL(p_\theta(X) \parallel p_\theta(X')) < \epsilon_2 \tag{2.2}$$

That translated into words becomes: if two random inputs from the dataset X and X' are close (their distance is less than a given ϵ_1, with respect to the L_2 norm), then the KL-divergence of their respective output activations is smaller than ϵ_2 [Fischer et al., 2019].

On paper this constraint seems highly effective and powerful but there are some fallacies. I found their implementation very poorly done: they take random batches of images and check the similarity using the euclidean distance and this raises some problems. First of all, not every image is compared with every other image in the dataset but only with some random sample. That alone makes the constraint accuracy higher by not checking and finding all the inputs that do not satisfy the constraint.

Furthermore, I believe that checking similarity with euclidean distance is wrong. For example, in figure 2.7 we can see that the two images represent the same "1" from the MNIST dataset but they are both moved a bit on the opposite sides. Calculating their distance would produce a large enough number to make the first part of the constraint 2.2 false thus satisfying said constraint independently from the network prediction. Another example is figure 2.8 where is shown again the same "1" this time in the same position in both images. However, in the second image the background is modified and that again renders the distance too high.

This constraint with this implementation is useless because it guarantees to always get a very high constraint accuracy not considering a lot of inputs that would not satisfy it.

Moreover, MNIST and FASHION-MNIST are really simple datasets and their images are all centered and with the same background. But with more complex datasets there is almost not any pair of images similar enough and the constraint accuracy will be near 100% every time.

However, I agree with the concept that if two images are similar they should be predicted into the same class. But the concept of similarity must be expressed differently. We do not want similarity on the whole images but just in the subject. It should not matter the position or the background or the scale. If we could achieve to check the subject similarity, I believe that this constraint could be useful for improving robustness.

On the other hand, for global robustness they define the constraint:

$$\forall z \in B_\epsilon(X) \cap [0,1]^d \,.\, log\ p_\theta(z)_y > \delta \tag{2.3}$$

That is: for any input X with classification y, inputs in its ϵ neighborhood which are valid images (pixels between 0 and 1), have a high probability for y. For numerical stability, instead of directly checking the probability, here is checked that the corresponding log-probability is larger than a given threshold δ [Fischer et al., 2019].

This constraint seems more robust than the previous one: it checks if the inputs and their neighborhoods are predicted into the same class. But we can equiparate this approach to training with adversarial examples. Hence, in order to test it, we will compare its performance and accuracy with a network trained with labeled adversarial examples generated via a GAN (Generative Adversarial Network).

2.6.5 Notes

In general, previous works do not apply to global constraints. Also, for the previously mentioned limitations, DL2 can handle constraints far more complex than prior works.

This research will test the framework against other state-of-the-art methods for verification like GANs to prove the efficacy of the aforementioned constraints. But also aims to understand if DL2 can be used to formulate new and better constraints for improving network robustness.

2.7 Generative Adversarial Networks

This section is about another state-of-the-art method for verification which we will use as a comparison for DL2: Generative Adversarial Networks (GANs). First we introduce GANs and then we look at the variant (AC-GAN) that we will use in this project.

Generative Adversarial Networks were first invented by Goodfellow et al. [2014a] and are algorithmic architectures that use two neural networks competing one against the other. They are used widely in image generation, video generation, and voice generation but also they can be used in verification.

GANs are composed of two networks:

- A generative model G called **the generator**, which takes as input a random noise vector z and gives as output an image $G(z)$. It is typically an inverse convolutional neural network.
- A discriminative model D called **the discriminator** which takes images as input and returns the probability for them to be real or fake (generated by G). It is typically a convolutional neural network.

Substantially, the generator captures the data distribution and generates new data instances, while the discriminator estimates the probability that a sample came from the training data rather than G.

Citing Goodfellow et al. [2014a]: "The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles.".

Training this model initially involves training the discriminator with the real dataset until it achieves an acceptable accuracy. The generator then trains on maximizing the probability of D making a mistake. Thereafter, candidates generated by G are evaluated by the discriminator. Backpropagation is applied in both networks so that the generator produces better images, while the discriminator becomes more skilled at flagging synthetic images.

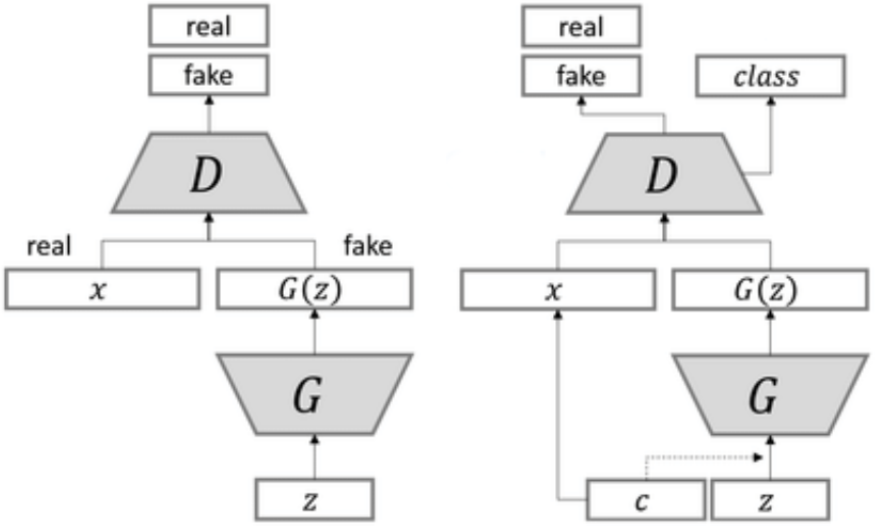

FIGURE 2.9: On the left is represented the structure of a GAN, while on the right the structure of an AC-GAN. [Mino and Spanakis, 2018]

GANs can be used for verification in two ways:

- By placing the discriminator before your network. This will prevent the network from being fooled by adversarial examples by blocking them before they are fed to it. However, this method is more useful as a defense method against adversarial attacks than for proper verification.
- By generating adversarial examples with the generator. These images will be used together with the real dataset to train your network. That will result in an improvement in robustness.

In this project we are going to follow the second method. However, as it is, the generated images are not labeled. Indeed GANs were initially used in unsupervised learning [Goodfellow et al., 2014a].

Since we are focusing on supervised learning though, we need a way to label them. For that reason, instead of normal GANs, we will utilize a variant that generates labeled images: Auxiliary Classifier GANs (AC-GANs).

AC-GANs [Odena et al., 2016] differ very little from the standard GAN but that difference is key, as shown in figure 2.9:

- The generator, besides the noise vector z, takes as input the corresponding class label c.

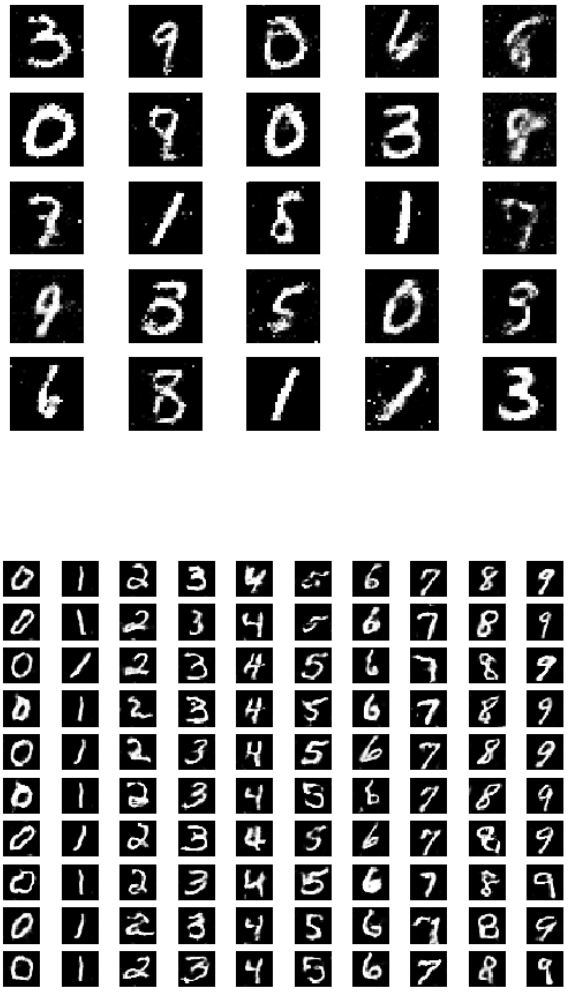

FIGURE 2.10: Images generated from a preliminary model of a GAN (above) and an AC-GAN (below).

- The discriminator outputs not only the probability for an image to be real or fake but also a probability distribution over the class labels.

Odena et al. [2016] demonstrate that these changes also improve the quality of the generated samples.

With this model, we are able to generate accurate labeled images that seem real to a human eye. Figure 2.10 shows some images generated with both a GAN and an AC-GAN that we made. Although the two networks are very simple and for the experiments we will need bigger models, they already achieve good results. We will use them along

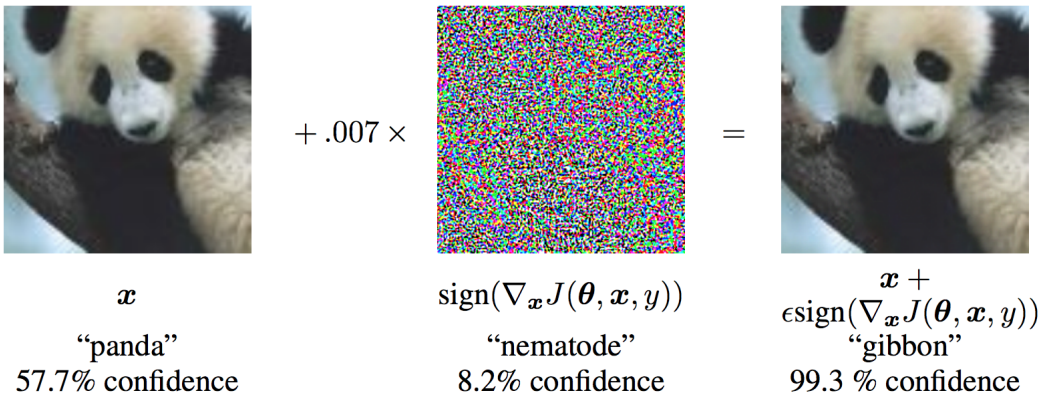

FIGURE 2.11: An example of FGSM attack. The image is firstly classified correctly as a panda and, after applying the noise generated by the attack, it gets classified as a gibbon. [Goodfellow et al., 2014b]

with the real dataset to train the previously cited CNNs. That will be the comparison for DL2 $Robustness^G$ constraint.

2.8 FGSM

Here I will introduce a method to attack a network: Fast Gradient Sign Method (FGSM). This attack will be used as a test for our DNNs.

This is one of the first and most popular adversarial attacks, introduced by Goodfellow et al. [2014b]. Figure 2.11 shows a famous example of the attack, while Figures 4.6, 4.7 and 4.8 show some examples generated in this thesis.

The attack is based on leveraging the network's gradients: while the network tries to minimize the loss by back-propagating the gradients, the attack modifies the inputs to maximize said loss based on the same gradients.

We will use this attack as a further test against our networks to evaluate robustness as explained in the next section.

2.9 Evaluation criteria

This section will explain the parameters on which our networks will be tested.

As stated in Chapter 3, for each dataset we will conduct 4 different experiments, or in other words we will train 4 different networks. We will then evaluate and compare them based on different metrics:

- Prediction Accuracy: that is the number of test inputs correctly classified.
- Constraint Accuracy: that is the number of test inputs that satisfy the DL2 $Robustness^G$ constraint.
- AC-GAN Accuracy: that is the accuracy of the inputs generated by the AC-GAN.
- FGSM Accuracy: that is the accuracy from the FGSM attacks. For this test I will conduct different attacks with different ϵ values.

It has already been noticed in the literature that prediction accuracy and robustness are at odds (Serban et al. [2019], Fischer et al. [2019]). However, it is fundamental that the prediction accuracy of a network be as high as possible. Therefore we are measuring it and monitoring that it is not dropping too much.

Constraint accuracy is distinctive of DL2 framework and it measures how much a network satisfies the chosen DL2 constraint. Fischer et al. [2019] propose that a high constraint accuracy entails a high robustness, but from the results it seems to not be the case (Chapter 4). However it is the metric used in their experiments so we are reproducing it here as well.

AC-GAN and FGSM are again accuracy measures, however on different, *adversarial data sets*. Such sets consist of images newly generated by an AC-GAN in the first case, and of images that are specially perturbed copies of the images from the testing data sets in the second case. We are using both metrics because, indeed, we want to test if DL2 produces networks that are robust against different attacks.

All these metrics will be used to test and prove the robustness of DL2 with regards to the other networks and methods.

2.10 Conclusions

In this chapter we treated and explained the necessary theory and tools to understand and proceed with the experiments.

From the broader term that is AI, we narrowed to the concepts that are important for this research: Supervised Learning, Neural Networks, CNNs, Adversarial Examples, and Verification.

We then went in detail about the software and systems for verification that we are going to test: DL2, GANs, and AC-GANs.

Lastly we explained the evaluation criteria and metrics for these experiments.

In the next chapter we are going to talk about the requirements and methodology that will allow us to achieve our objectives.

Chapter 3

Methodology

This chapter will initially state the research hypoteses for this thesis, following by the set up and implementation, and it will finish with the project's requirements.

3.1 Hypotheses

We start with clear formulation of research hypotheses that are present, implicitly or explicitly, in the research papers that argue for using constraint-driven training in neural networks (Hu et al. [2016], Xu et al. [2017], Fischer et al. [2019], Ayers et al. [2020]).

RH1 DL2 does not perform significantly worse on prediction accuracy than standard networks, meaning networks trained on the standard data sets with a standard loss function.

RH2 DL2 networks are more robust against all the attacks than standard networks, defined as above.

RH3 DL2 produces networks more robust against all the attacks than GANs.

RH4 Training with DL2 achieves higher robustness against all the attacks than networks trained on the adversarial examples generated by my script.

RH5 Both Global and Local robustness constraints are useful for training neural networks.

Our goal is to either prove or disprove each of them, using some systematic methodology.

3.2 Global versus Local robustness

We have been able to carry out some preliminary experiments on the DL2 system. The original plan was to create a new data set with images that did not satisfy the constraint in a not trained network. Therefore we fed the MNIST data set to a fresh network to see which inputs didn't satisfy the $Robustness^T$ (Local robustness) constraint.

However, we found out that constraint accuracy is always 1 before training. Further experiments showed that the fault is in the second part of Formula 2.2: $KL(p_\theta(x) \parallel p_\theta(x')) < \epsilon_2$ is always True.

We then wrote a script to test that constraint on the MNIST data set without the use of a network: the script compared each image in the data set to all the others and if they were similar (their euclidean distance was less than ϵ_1), we checked if they had the same label. However, the search took about 4 hours to finish and in the end, we only got 6 results. And those results were two 7 that were similar to some 1, even a human eye could mistake them. Therefore we concluded what we said in chapter 2 and that $Robustness^T$ is a poorly implemented and useless constraint.

For those reasons, we disprove Hypothesis **RH5** and we will carry on the experiments only on the $Robustness^G$ (Global robustness) constraint.

3.3 Experimental set up

To achieve our objectives, we are going to compare DL2 to other three models: a baseline network trained on the standard data sets and two networks trained on adversarial data sets, all three trained using a standard loss function.

Since we found that the $Robustness^T$ constraint is not solid, we will test only DL2 $Robustness^G$. As we explained in Chapter 2, this constraint is defined as: for any input X with classification y, inputs in its ϵ neighborhood which are valid images (pixels between 0 and 1), have a high probability for y.

This can be seen as comparable to generating new images and using them as additional inputs. Therefore we are going to generate two additional (adversarial) data sets for each previously mentioned data set (we will refer to the original data sets as $DatasetsA$):

- The first data set will be generated using an AC-GAN. We will call them $DatasetsB$.
- The second data set will be created through a script that will generate a set of new images in the ϵ neighborhood of the images in the $DatasetsA$. We will refer to these as $DatasetsC$.

Since both $DatasetsB$ and $DatasetsC$ are labeled data sets we can use them in supervised learning.

Then, 4 networks will be trained and compared for all the four original datasets:

- A CNN will be trained using DL2 loss with the $Robustness^G$ constraint on the $DatasetsA$. This is the DL2 network that will be evaluated against the other methods.
- A CNN will be trained on the $DatasetsA$ with a standard loss. This will be a network with no improved robustness and will be the base to see if the other methods will be an improvement (Hypotheses **RH1** and **RH2**).
- A CNN will be trained on the $DatasetsB$. This is to test if DL2 is better of other state-of-the-art methods for verification, namely GANs (Hypothesis **RH3**).
- A CNN will be trained on the $DatasetsC$. This is to test if the DL2 loss is useful or just redundant (Hypothesis **RH4**).

After all the networks are trained we will compare the results, evaluating both prediction accuracy and robustness. Depending on the results, we can then follow different paths:

- If DL2 proves to perform significantly better on constraint training while retaining a high prediction accuracy, we will research and develop new and better constraints.
- If DL2 does not achieve a high performance, we will implement our own methodology/software to perform verification during training.

Depending on the time available, we can implement our methodology even if DL2 proves to be effective.

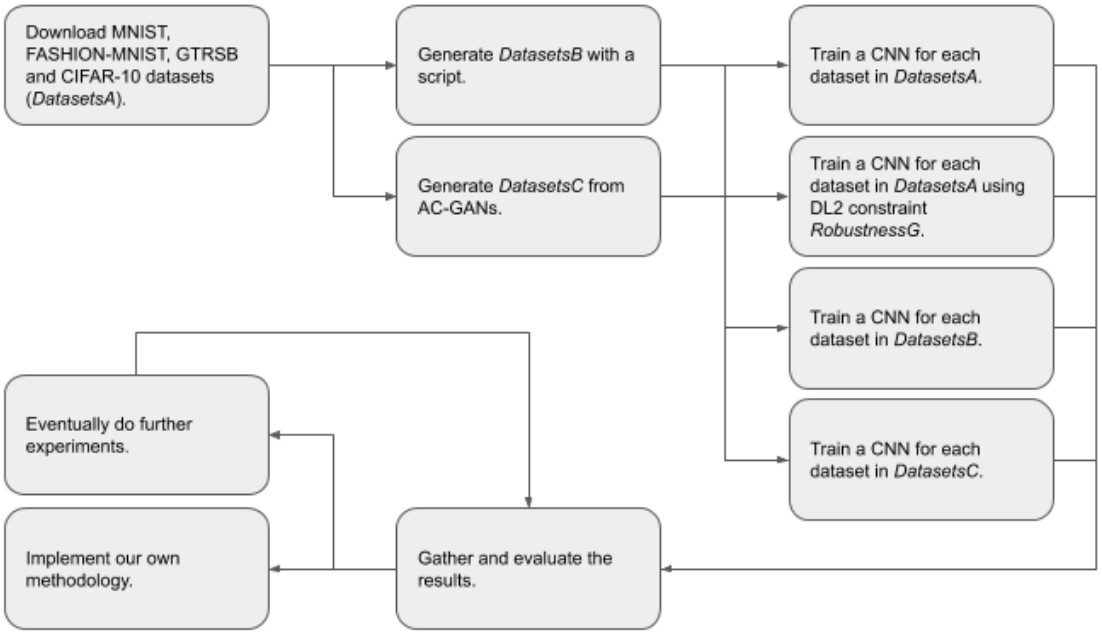

FIGURE 3.1: Project overview

3.4 Implementation

Figure 3.1 shows the workflow of this project:

- First we downloaded the datasets mentioned in Section 3.4.
- Secondly we generated the *DatasetsB* and *DatasetsC* and prepared them for PyTorch, a library for machine learning (section 3.5).
- Then we could train the 4 networks explained in section 3.2 on the different datasets. However before that, we had to spend a considerable amount of time understanding, fixing and adapting the DL2 code.
- Lastly we gathered and evaluated the results. We needed to carry further experiments than initially predicted so we did not have time to implement our own methodology.

3.5 Datasets

In this research, three benchmark datasets will be used.

FIGURE 3.2: 10 images for each class from the MNIST dataset. Yann [2013]

MNIST [Image 3.2]:

This dataset contains 60,000 training images and 10,000 testing images of handwritten digits. Each image is a 28x28 gray-scale image, associated with a label from 10 classes: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

FASHION-MNIST [Image 3.3]:

This dataset consists of 60,000 training images and 10,000 testing images of clothing items. Each image is a 28x28 gray-scale image, associated with a label from 10 classes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot.

GTSRB:

This dataset contains more than 50,000 images of German traffic signs. There are more than 40 classes and image sizes vary between 15x15 to 250x250 pixels.

However we will use a modified version that contains 12,660 training images and 4,170 test images. These images are all center-cropped, gray-scaled, 48x48 pixel and reduced to belong only to 10 classes.

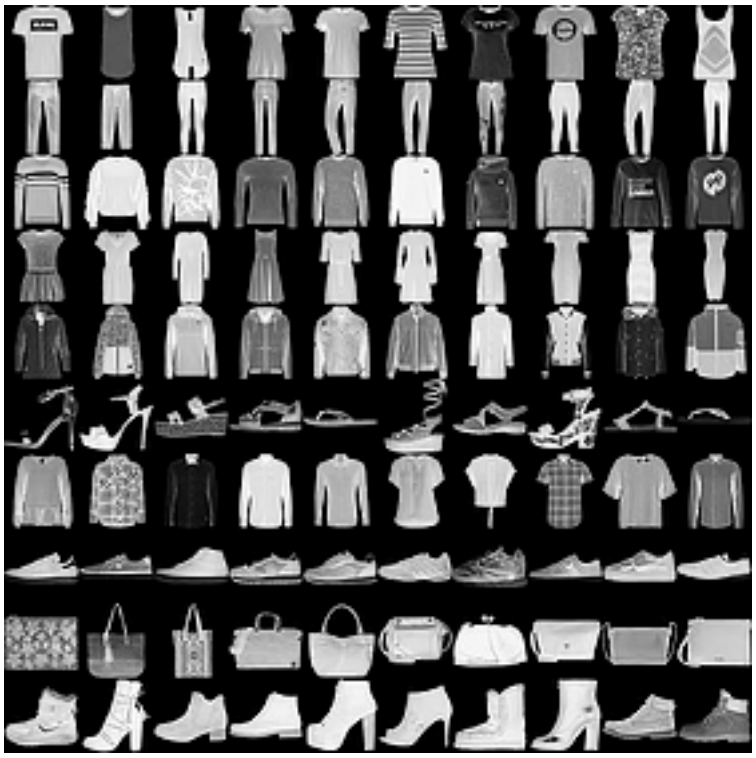

FIGURE 3.3: 10 images for each class from the FASHION-MNIST dataset. Xiao et al. [2017]

3.6 Libraries

This section explains the various Python libraries used in this project.

PyTorch: is an open-source machine learning library developed by Facebook's AI Research lab (FAIR). It is one of the most used libraries for deep neural networks. DL2 uses this library for its CNNs, therefore the models will be built using the PyTorch library.

TensorFlow: is another open-source machine learning library, and it is the most used worldwide. It was developed by the Google Brain team for internal Google use and then released under the Apache License 2.0 in November 2015. This library will be used to create the AC-GANs that will generate the new datasets. I will use TensorFlow instead of PyTorch because I am more familiar with it and because there are more resources for it. This will save time because AC-GANs are not built-in in neither of the two libraries.

DL2: is the software that we are going to test. The code is divided into three main folders: dl2lib, querying, and training. Following is a break down of the files inside those folders.

dl2lib: this folder contains utility files for the other two.

api.py - This file translates text into queries for a network.

parser.py - When you run DL2 you have to specify different parameters, this file handles the parsing.

query.py - This file is a bridge between your code and the *api.py* file. First, you generate a text query, then you call this file that will call the *api.py* file to translate the query.

args.py - This file can be called to add default arguments to the parser.

diffsat.py - In this file are defined the basic constraint components and their satisfaction method. This is a core file for DL2 and it is called when it is time to check whether or not our constraint is satisfied.

querying: this folder contains the code that allows querying networks.

context.py - This file defines the database and the network to be queried. A context object gets passed to the query.py file along with the text query.

evaluation_queries.py - This file contains some pre-defined queries that can be ran.

run.py - This file runs the queries from *evaluation_queries.py*.

training: this folder contains the code to train networks and it is divided into three more sub-folders: semisupervised, supervised, and unsupervised. The semisupervised and unsupervised folders only contain a main file that trains the networks. However, we are not interested in those types of learning and we will only focus on supervised learning.

constraints.py - This builds upon the *diffsat.py* file to create the constraints specified in the paper and that are used in their experiments. Here are implemented the $Robustness^T$ and $Robustness^G$ constraints.

main.py - This file runs all the experiments that are shown in the DL2 paper.

models.py - In this file is created the small CNN used with the MNIST and FASHION-MNIST datasets.

oracles.py - This file is a bridge between your code and the *constraints.py* file. It returns the constraint accuracy and, when called during training, it returns the DL2 loss that will be summed to the normal loss.

resnet.py - As the name suggests, in this file is created the ResNet, that is the CNN used with the GTSRB and CIFAR-10 datasets.

In this project, we are not interested in querying but only in training, therefore we will only be re-using some of the DL2 files that are present in the *supervised* and *dl2lib* folders.

The procedure to use DL2 to train is as follows: load the dataset, create a new model, create a constraint object, create an oracle object, for each epoch call the *train* method followed by the *test* method from *main.py*, save the model, save the accuracy measures.

3.7 Requirements Analysis

As this project splits its research objectives between data science, algorithm analysis, and software engineering, we split the project requirements as follows:

A: Non-functional requirements 1: data collection and analytics.

B: Non-functional requirements 2: infrastructure for experiment design.

C: Functional requirements: the software that the project produces.

The requirements are listed in Table 3.2.

Requirement description	Type	Importance	Achieved
Download the MNIST, FASHION-MNIST and GTSRB datasets.	A	H	YES
Make the datasets usable with PyTorch.	A	H	YES
Generate the *DatasetsB* through an ACGAN.	A	H	YES
Generate the *DatasetsC* through a script.	A	H	YES
For each model trained and for each epoch, collect prediction accuracy and constraint accuracy measures.	A	H	YES
Attack every network with an ACGAN.	A	H	YES

Attack every network with an FGSM attack.	A	H	YES
Collect the accuracy values for the ACGAN attack.	A	H	YES
Collect the accuracy values for the FGSM attack.	A	H	YES
Formulate the hypotheses.	B	H	YES
Understand the results.	B	H	YES
Reformulate the hypotheses based on the results.	B	H	YES
Understand what to do next based on the results.	B	H	YES
Automatic generation of confusion matrices.	B	M	NO
Automatic generation of prediction accuracy and constraint accuracy tables.	B	M	NO
Automatic generation of ACGAN and FGSM attacks accuracy tables.	B	M	NO
Graphical representation of the results.	B	L	YES
Python script to download, save and make usable the aforementioned 3 datasets for PyTorch and Tensorflow libraries.	C	H	YES
Python script for training and saving an AC-GAN model for each dataset.	C	H	YES
Python script for using the saved AC-GAN models to generate images for each dataset.	C	H	YES
Python script for generating images in the ϵ neighborhood of the datasets.	C	H	YES
Python script for generating and saving new datasets from the images created in the two previous requirements.	C	H	YES
Python script that runs DL2 to train 4 CNNs for each dataset (12 networks in total as explained in Chapter 3).	C	H	YES
The last python script will also save the model, prediction accuracy, constraint accuracy and time variables in a json dictionary for each epoch.	C	H	YES

Python script for attacking the networks with an ACGAN and saving the results.	C	H	YES
Python script for attacking the networks with an FGSM attack and saving the results.	C	H	YES
Python script for selecting the best model for each of the 12 networks.	C	M	NO

TABLE 3.2: Requirements

The majority of the requirements were achieved, and those that were not did not have a serious impact on the project. Indeed we manually selected the best models and manually generated the tables because it was faster than creating the scripts.

3.8 Software Release

All the code used for the experiments of this thesis is freely available on GitHub.

The first folder *GenerateDatasets* contains the code for generating the *DatasetsA*, *DatasetsB*, *DatasetsC* and the images for the AC-GAN attack. While the second folder *dl2* contains the code for creating, training, evaluating and attacking all the networks.

This library can be used, in the first place, to reproduce the results of this thesis. However, more interestingly, it can be used to carry out further experiments.

It is very easy to add new data sets: if the data set is available in the TensorFlow library, modify the file *create_datasetsA.py* in the first folder by copying the code for MNIST or FASHION, and if the data set comes from an external source, modify the same file by copying the code for the GTSRB data set. Then just change the name of the variables to mach that of your data set.

Since the generation of the *DatasetsB*, *DatasetsC* and AC-GAN attack test sets is achieved starting from *DatasetsA*, it is straightforward to then modify the code, as it is the same independently from the data set.

Furthermore, it is also easy to change the networks parameters and configurations as they are defined in the *models.py* file and it needs only to change one line of code in the *main.py* file to use a different architecture.

Finally, it is possible to create new constraints to test using the DL2 framework and to change the parameters of the $Robustness^G$ constraint there is not even the need to change any code: just express the chosen parameters via command line when launching the *main.py* file.

All the attacks and metrics are independent from the above characteristics and , a part from stating the name of the chosen model, they do not need any code modification to work on new networks or with new data sets.

In conclusion, this library can be used to reproduce the results described in this document or to conduct further experiments with different data sets, other architectures or new constraints, all this with minimum modification required.

Chapter 4

Results

In this chapter, we will present and discuss the results of this project. We will start by introducing and motivating the chosen experiments and we will follow with a discussion and considerations about the achieved results.

4.1 Experiments

In order to prove, or disprove, our hypotheses (Chapter 3) we design 4 experiments. This section will motivate and explain in detail each of them. But first we will specify the models' architectures.

Since we want to be consistent and methodical, for the same data set, all the 4 experiments will share the same architecture. Furthermore, for the MNIST and FASHION data sets we use the same architecture as Fischer et al. [2019] use in the DL2 paper: a CNN with 6 convolutional layers and 2 linear layers.

For the GTSRB data set, they did not use it in their experiments, therefore we arbitrarily chose the architecture. We maintained the same layout: a CNN with 6 convolutional layers and 2 linear layers, however we changed the dimensions of the layers to match them with the input size of 48x48 of this data set instead that of the size of 28x28 of the other 2 data sets.

These architectures are small CNNs but, bearing in mind that our primary goal is to achieve consistency amongst our experiments to systematically evaluate them, they work perfectly for this project.

Now that we have specified the architectures, we can proceed to explain the experiments. For each data set we define 4 experiments:

Experiment DL2: DL2 Model This model is trained using the DL2 loss with the $Robustness^G$ constraint, as explained in Chapter 2.

This model is trained on the *Datasets A*. That means 60,000 images for MNIST and FASHION and 12,660 images for GTSRB.

All the experiments' models are tested on the same original test sets for consistency on the prediction accuracy results. That means 10,000 images for MNIST and FASHION and 4,170 images for GTSRB.

This is the DL2 model that will be compared and evaluated against the other experiments.

Experiment A: Baseline Model This model is trained using a standard loss.

This model is trained on the *Datasets A*. That means 60,000 images for MNIST and FASHION and 12,660 images for GTSRB.

We define this model as the standard network and we will compare it to the DL2 model to test Hypotheses **RH1** and **RH2**.

Experiment B: AC-GAN Model This model is trained using a standard loss.

This model is trained on the *Datasets B*. They are created by adding to the *Datasets A* new images generated by an AC-GAN. This is possible since AC-GANs generate labeled images (Chapter 2).

I created 10,000 new images per label and, since all the 3 data sets used have 10 labels, this resulted in 100,000 new images per data set. Therefore *Datasets B* contain 160,000 images for MNIST and FASHION and 112,660 images for GTSRB.

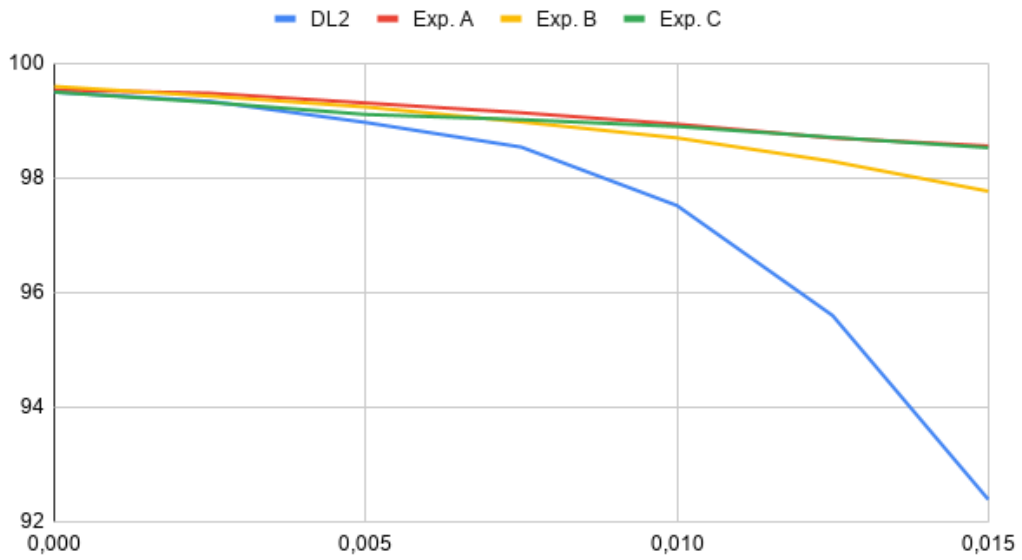

FIGURE 4.1: This graph displays the results for the MNIST dataset in table A.6. On the y axis is represented the accuracy in percentage, while on the x axis is represented the ϵ value of the FGSM attack. We can see that for this dataset the DL2 network is the least robust amongst all the networks.

This model will be used for testing Hypothesis **RH3**.

Experiment C: My Model This model is trained using a standard loss.

This model is trained on the *Datasets C*. They are created by adding to the *Datasets A* new images generated by a script of my own invention.

The $Robustness^G$ constraint, that we are using to train DL2 networks, checks that the images in the *epsilon ball* of each input are classified correctly as well. The idea behind my script is to mimic this behaviour by creating new images by randomly modify the originals, while not exceeding the *epsilon ball*.

I created 2 new images per input, therefore *Datasets C* contain 180,000 images for MNIST and FASHION and 37,980 images for GTSRB.

This model will be used for testing Hypothesis **RH4**.

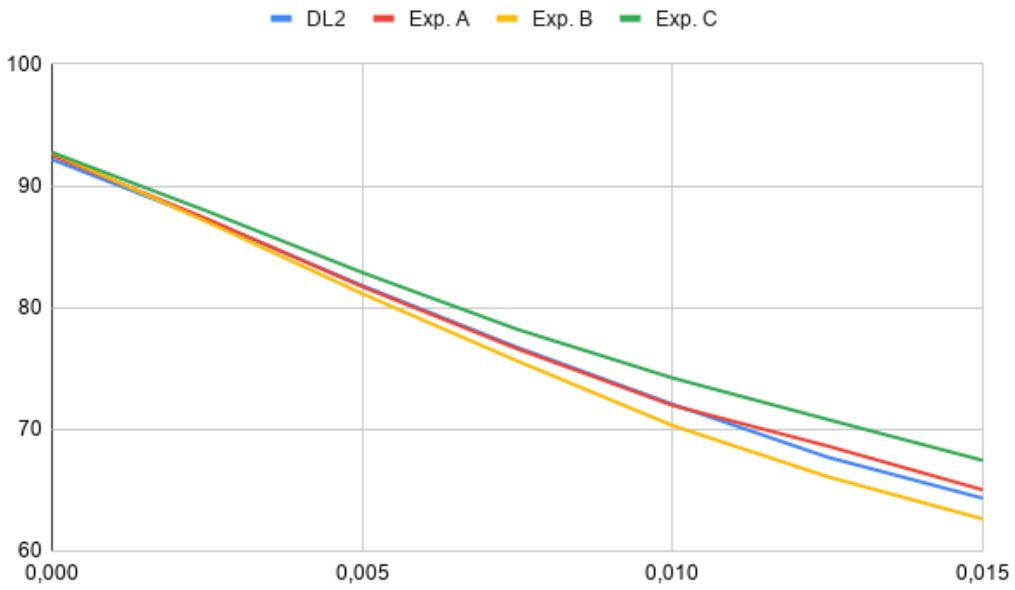

FIGURE 4.2: This graph displays the results for the FASHION dataset in table A.6. On the y axis is represented the accuracy in percentage, while on the x axis is represented the ϵ value of the FGSM attack. We can see that the DL2 network has a similar performance as the baseline model (Exp. A) but is less robust than Exp. C.

4.2 Results

In this section will be reported the results, along with the knowledge necessary to interpret them.

As stated in Section 2.9 we have 4 metrics for evaluation: Prediction Accuracy, Constraint Accuracy, AC-GAN Accuracy and FGSM Accuracy. The first 3 are reported in Table 4.2 while the FGSM Accuracy is reported in Figures 4.1, 4.2, 4.3, 4.4 and 4.5.

Dataset	Experiment	P acc.	C acc. $eps = 0.3$	C acc. $eps = 0.01$	ACGAN
MNIST	DL2	99.50	**100.0**		99.87
	Exp. A	99.54	00.00	99.91	99.84
	Exp. B	**99.60**	00.00	99.57	**99.97**
	Exp. C	99.50	00.00	99.85	99.86
FASHION	DL2	92.18	**99.92**		97.42
	Exp. A	92.51	00.00	92.45	98.25
	Exp. B	92.73	00.00	90.62	**99.92**
	Exp. C	**92.76**	00.00	88.63	98.05

GTSRB	DL2	99.09	**99.95**		75.30
	Exp. A	**99.33**	00.00	90.81	76.78
	Exp. B	99.14	00.00	78.36	**99.35**
	Exp. C	99.14	00.00	89.80	75.28

TABLE 4.2: This table reports the Prediction Accuracy, Constraint Accuracy and AC-GAN Accuracy resulted from our experiments.

Notice how Table 4.2 has 2 columns reporting the Constraint Accuracy: the first with $eps = 0.3$ and the second with $eps = 0.01$. The first column is the one used to evaluate the experiments, because is the same *epsilon* that we use to train the DL2 models.

However, the accuracy value for the experiments that use a standard loss is always 0. Therefore we ran a test with a smaller *epsilon* in order to see if it was an implementation error or it was correct. Since the results in the second column are always not equal 0, we conclude that the results are not an error.

With that in mind, a more precise definition of what is *epsilon* is needed here. In this thesis we have two different *epsilons*: the first being the *epsilon* used by DL2 and the second being the *epsilon* used by the FGSM attacks.

In DL2, the *epsilon* refers to the range of the *eps ball* around the inputs. In other words, it the the maximum distance from the original image.

In the FGSM attacks, the *epsilon* refers to the intensity of the perturbation that will be applied to the original image.

To better understand the difference, we can look at Figures 4.6, 4.7 and 4.8. In each triple we find the original image (left), the perturbation (center), and the modified image (right). We can see that the modified images are not distinguishable from the originals to the human eye. That is because only a fraction of the perturbation is applied. That fraction corresponds to the *epsilon* of the FGSM attack. On the other hand, the DL2 *epsilon* is the Euclidean Distance between the original images on the left and the perturbed images on the right.

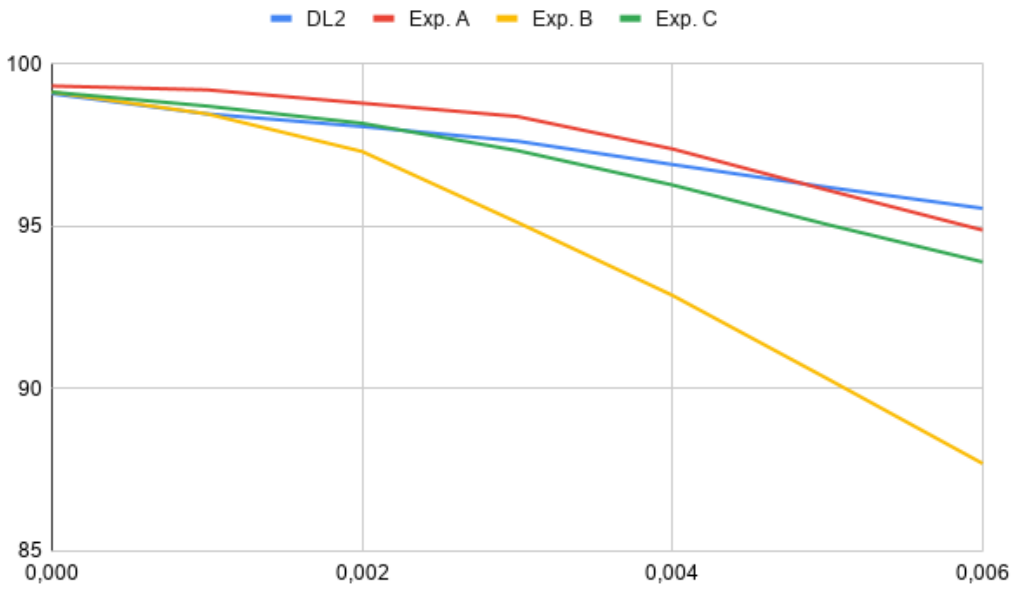

FIGURE 4.3: This graph displays the results for the GTSRB dataset in table A.6. On the y axis is represented the accuracy in percentage, while on the x axis is represented the ϵ value of the FGSM attack. We can see that for this dataset the DL2 network is initially similar in accuracy as Exp. C and worse than the baseline model (Exp. A). However, for $\epsilon = 0.005$ and greater, it becomes the most robust. This is the only case where the DL2 network actually improves upon all the other networks.

In the FGSM attacks we chose *epsilon* values that produce adversarial images that are distant less than 0.3 at maximum. That because we trained the DL2 models with such a value.

Now we have all the knowledge to understand the results and in the next sections we will interpret them and see if the Hypotheses are proven or not.

4.2.1 DL2 does not perform significantly worse on Prediction Accuracy than standard networks.

As we can see from Table 4.2, DL2 has a slightly lower Prediction Accuracy than *Experiments A*. However, that was expected as it is often the case with this type of training (Hu et al. [2016], Xu et al. [2017], Fischer et al. [2019], Ayers et al. [2020]). Indeed, we obtained similar results as Fischer et al. [2019] in the DL2 paper and we can say that **RH1** is proven to be correct: DL2 does not perform significantly worse on Prediction Accuracy than standard networks.

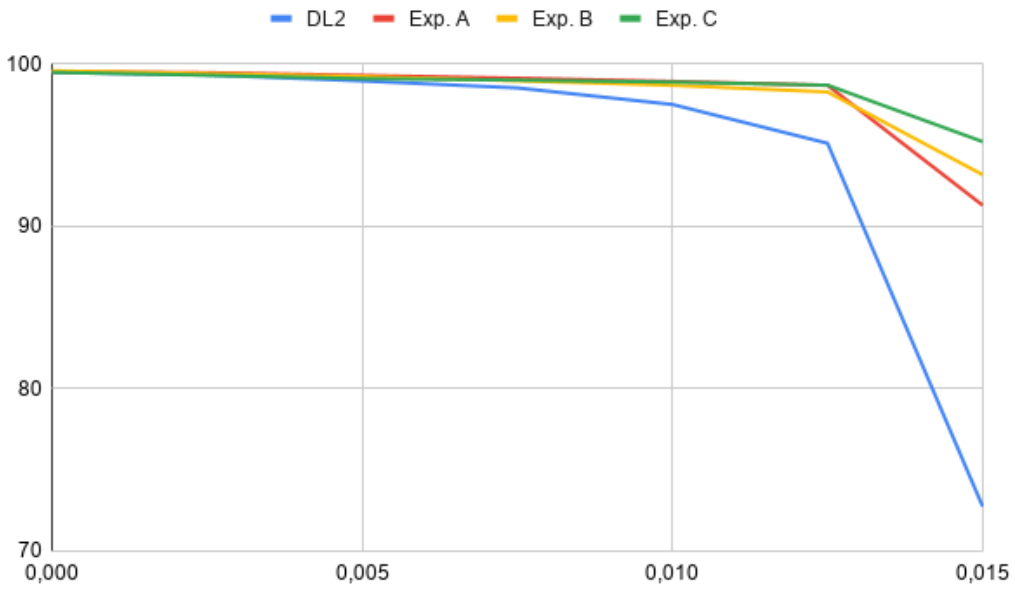

FIGURE 4.4: This graph displays the results for the MNIST dataset in table A.7. On the y axis is represented the accuracy in percentage, while on the x axis is represented the ϵ value of the FGSM attack. We can see that the results are not changed from Figure 4.1: the DL2 network performs the worst amongst all of them.

4.2.2 DL2 is less robust than standard networks.

As we can see from Table 4.2, DL2 networks achieve an extremely high Constraint Accuracy while the *Exp. A* networks achieve 0.

However, against the AC-GAN attacks, DL2 has a very minor improvement on the MNIST data set but it is worse than the standard networks on both FASHION and GTSRB data sets.

Furthermore, Graphs 4.1, 4.2 and 4.3 show that against FGSM attacks:

MNIST: DL2 robustness is heavily below that of a standard network.

FASHION: DL2 robustness is similar to that of a standard network.

GTSRB: DL2 robustness is below that of a standard network for the most part but, after $\epsilon = 0.005$, it improves and it becomes better than that.

In conclusion, we can say that **RH2** is proven to be wrong: DL2 is generally less or equally robust than standard networks and, when it is more robust, the improvement is negligible.

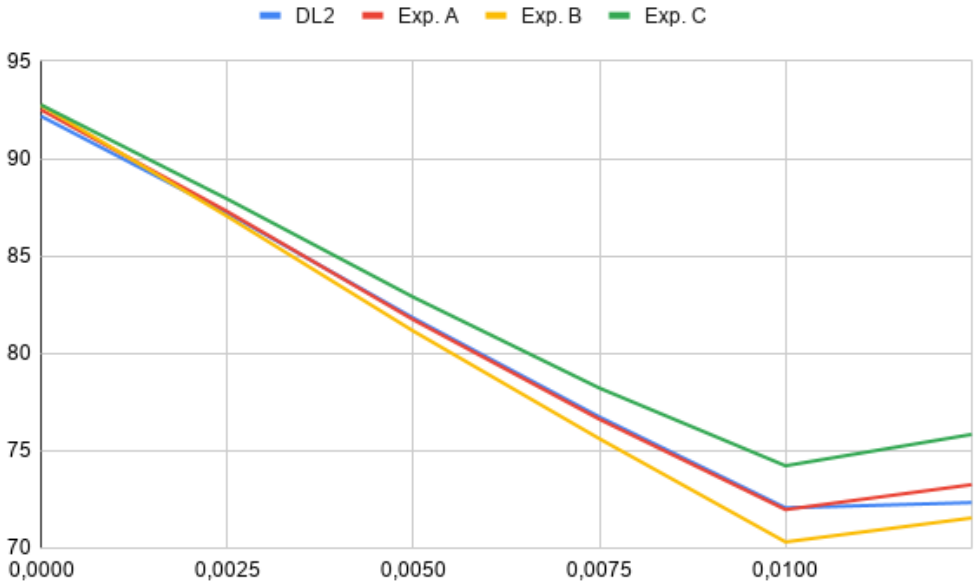

FIGURE 4.5: This graph displays the results for the FASHION dataset in table A.7. On the y axis is represented the accuracy in percentage, while on the x axis is represented the ϵ value of the FGSM attack. We can see that the results are not changed from Figure 4.2: the DL2 network performs similar to the baseline model (Exp. A) and worse than Exp. C.

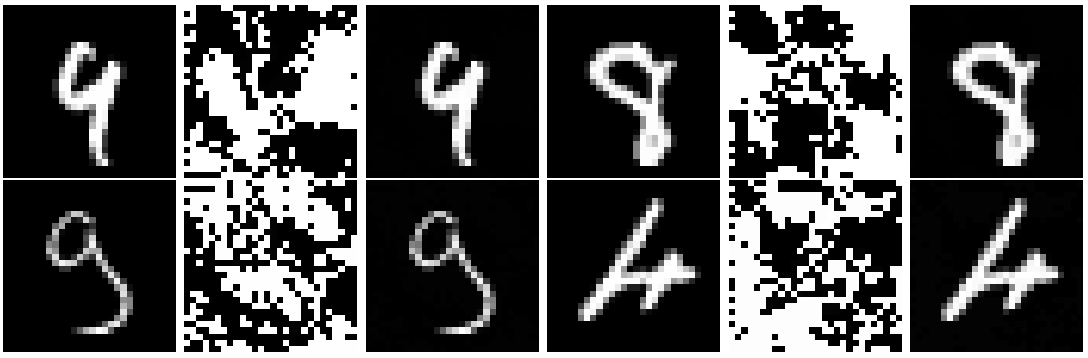

FIGURE 4.6: Examples of FGSM attacks on the MNIST data set.

4.2.3 DL2 is more robust than GANs against FGSM attacks but less robust against GANs attacks.

As we can see from Table 4.2, DL2 networks achieve an extremely high Constraint Accuracy while the *Exp. B* networks achieve 0.

However, against the AC-GAN attacks, DL2 performs significantly worse the AC-GAN networks.

For what concern FGSM attacks, Graphs 4.1, 4.2 and 4.3 show that:

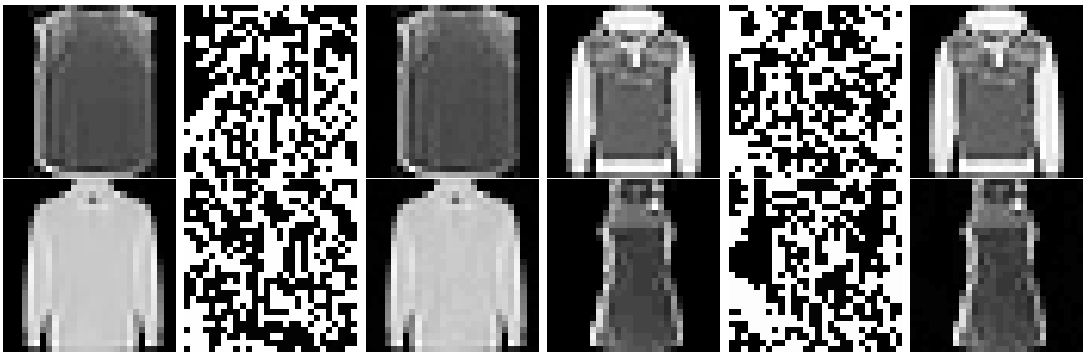

FIGURE 4.7: Examples of FGSM attacks on the FASHION data set.

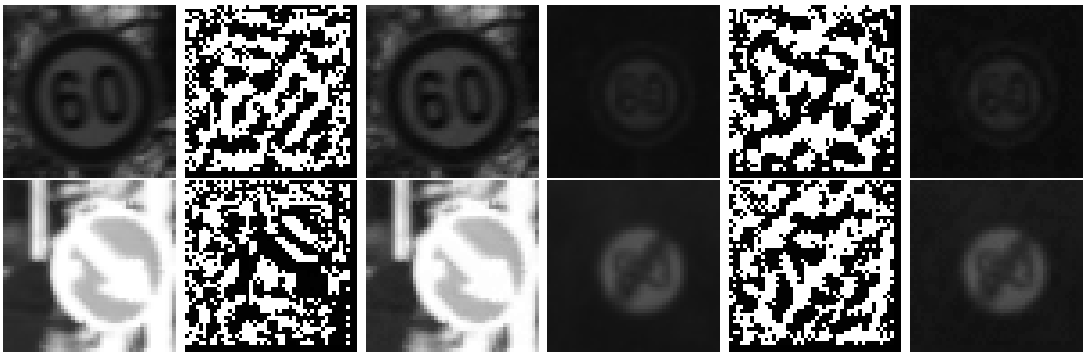

FIGURE 4.8: Examples of FGSM attacks on the GTSRB data set.

MNIST: DL2 robustness is heavily below that of an AC-GAN network.

FASHION: DL2 is slightly more robust then an AC-GAN network.

GTSRB: DL2 is significantly more robust then an AC-GAN network.

In conclusion, we can say that **RH3** is proven to be wrong: DL2 is more robust than GANs against some attacks but not against all of them.

4.2.4 DL2 is less robust than networks trained on adversarial examples generated by my script.

As we can see from Table 4.2, DL2 networks achieve an extremely high Constraint Accuracy while the *Exp. C* networks achieve 0.

However, against the AC-GAN attacks, DL2 performs similarly on the MNIST data set but it is worse than the *Exp. C* models on both FASHION and GTSRB data sets.

Furthermore, Graphs 4.1, 4.2 and 4.3 show that against FGSM attacks:

MNIST: DL2 robustness is heavily below that of the *Exp. C* network.

FASHION: DL2 is less robust then the *Exp. C* model.

GTSRB: DL2 is more robust then an the *Exp. C* network.

In conclusion, we can say that **RH4** is proven to be wrong: DL2 is generally less robust than networks trained on adversarial examples generated by my script.

Chapter 5

Conclusion and Future Work

In this chapter we will give the final conclusion of this thesis and we will talk about possible future work.

5.1 Conclusion

In this project we systematically evaluated the DL2 framework and compared it to other methods for improving robustness.

From our experiments, the main result is that *DL2 does not improve robustness against the majority of the tested attacks and, when it does, the improvement is negligible.*

It is true that DL2 does not significantly drop in prediction accuracy, however this does not come with an improvement in robustness.

Indeed, against AC-GAN attacks DL2 performs similar to both the standard networks and the models trained on adversarial examples generated by my script. However it consistently performs significantly worse than the AC-GAN networks.

Furthermore, against FGSM attacks DL2 performs generally worse than both the standard networks and the models trained on adversarial examples generated by my script.

Constraint accuracy is the only metric where DL2 excels while all the other methods give specially poor results. Nevertheless, against the tested attacks this does not help to raise robustness.

Moreover, DL2 is almost always worse than the networks trained on the examples generated by my script. That script is designed to mimic the functionality of DL2 $Robustness^G$ constraint but it only generates 2 images per input, while the constraint checks for 50 of them.

From these results we can infer that constraint-driven training is worse than generative training because the first does not produce more robust networks against any attack, while the latter produces networks with an higher robustness at least against generative attacks (GANs). It is true that against gradient-based attacks (FGSM) is weaker than standard models, but at least it can be useful in targeted scenarios.

Finally, from our experiments we can conclude that constraint training does not seem to entail robustness, hence it should not be seen as a replacement for proper verification methods.

5.2 Future Work

Although we made all efforts to show that our conclusions are replicable with several models, attacks methods, data sets and training environments, we recommend that further studies are conducted to confirm our conclusions.

In particular, one could increase the number of neighbours that are checked by the constraint to see if after a certain number it starts to actually improve robustness.

Moreover, one could create new and different constraints, maybe it is not the constraint training that is ineffective but this particular kind of constraint.

Furthermore, one could train a model by initially using a standard loss and, after the network converged, training the model with the constraint loss.

If any of these work turns out to improve robustness, we could say that there is a way of using constraint training for verification. However, with our results, our conclusion remains that *DL2 and constraint-driven training does not improve robustness and is futile in this scope.*

Appendix A

Appendix

A.1 Risk Analysis

While I was trying to break down and understand DL2, I encountered some problems.

First and foremost, the paper is not very clear in explaining the mathematical properties of its definitions. For example to really understand the definition of $Robustness^T$ and $Robustness^G$ it was necessary to dig into the code.

Furthermore in the paper is not explicitly explained how they translated their constraints into code and the code does not have comments, therefore it took a lot more time to break it down than necessary.

Since I already broke down and understood DL2 definitions and code, the aforementioned problems can be considered solved and will not affect the plan.

The code also presented some bugs: it could not run all the datasets correctly, it missed some important parts and some of it did not work with windows. I already fixed them and retrieved the missing code but other bugs could turn up, thus delaying the plan.

Lastly, early experiments showed unexpected results (as elaborated in the previous section) and required a lot more experimentation to fully understand the reasons behind those results. Depending on the results we could have to take additional experiments and that could be a great loss of time.

Table A.1 shows the risks.

Risk	Likelihood	Impact	Mitigation
Datasets take more time than expected to be generated	Low	Low	Since this task is at the beginning, in case of necessity we can allocate more time to it.
Bugs in DL2 code	Medium	Low	As explained above there could be new bugs but since we already did early experiments, if found, they should not cause major disruptions to the plan. In any case DL2 code will be used early on in the project and we can allocate more time.
Networks take too much time to train	High	Low	As seen with the early experiments, training with $Robusntess^G$ takes about 50 minutes per epoch but the timing can change depending on the computer conditions and usage at each moment. Since we will train 16 networks in total (4 networks for each of the 4 datasets), if the training takes too much time we can skip one of the datasets.
Not enough time to implement our own methodology	Medium	Medium	Since this task is at the end and it is impossible to predict how much time it will need, there is the risk that we have to let it unfinished. However this task is secondary, the primary goal is to test DL2 with the other methods.
Experiments can show unexpected results	Low	High	Depending on the results, it can be required to run additional experiments. That could be a great loss of time and could lead to abandon the implementation of our own methodology.

TABLE A.1: Risk assessment

A.2 Planning

The plan is shown in the Gantt chart [Figure A.1]. Training the networks will be automated and will require a lot of time. Therefore while the software is running I will start to write the thesis.

A.3 Professional and Legal Issues

This project will follow and respect the British Computer Society (BSC) Code of Conduct.

The libraries used in this project are all free licenses, open-source, and available for everyone to use. The programming language used is Python version 3.71 which is developed under an OSI-approved open source license. Therefore, it is free to use and distribute.

The datasets run in these experiments are under the terms of the CC-BY license that allows users to share and adapt them so long as they give credit.

Since this project regards verification, it will follow the ORBIT principles for responsible research and innovation. The experiments will be reproducible and thoroughly documented. All results will be fairly reported and explained, including all the unexpected results. The conclusions will be justified and all the confound variables will be disclosed.

The software produced will be open source and made available on GitHub for any person to use, modify, and distribute without limitation.

A.4 Ethical and Social Issues

This project does not involve experiments with human participants and the datasets do not contain any personal, sensitive, or confidential data.

However, AI and Machine Learning themselves can raise ethical and social issues. They are powerful tools but can be used against ethics. This project concerns verification, an instrument with the goal to improve the robustness of Machine Learning algorithms. For that reason, who inappropriately uses them can also potentially benefits from this project by having more resilient networks.

A.5 Results Tables

Tables A.2, A.3, A.4 and A.5 are introductory and useful to better understand the accuracy results for the FGSM attacks in tables A.6 and A.7. Indeed, they show the correlation between the *eps* of the FGSM attacks (the columns labels) and the *eps* of DL2 (the values), that is the euclidean distance.

In table A.2 there are the values corresponding to the mean distance for each attack. In table A.3 are represented the same values but only taking into account the adversarial examples, or in other words the inputs that get wrongly classified. Tables A.4 and A.5 are similar to the previous two but they represent the max distance instead of the mean.

These tables are useful because we want to test our networks on examples in the *eps ball* of radius 0.3, since DL2 networks are trained with such value.

Table A.4 contains the accuracy results for the FGSM attacks, while table A.5 contains the accuracy values without counting the examples generated with a distance greater than 0.3.

Finally tables A.8 and A.9 refer to the mean distance and accuracy of older experiments with bigger *eps* values.

Dataset	Experiment	0	0.0025	0.005	0.0075	0.01	0.0125	0.015
MNIST	DL2	0.000	0.057	0.113	0.170	0.225	0.280	0.336
	Exp. A	0.000	0.055	0.109	0.163	0.216	0.268	0.321
	Exp. B	0.000	0.055	0.111	0.165	0.219	0.273	0.326
	Exp. C	0.000	0.054	0.107	0.161	0.213	0.265	0.317
FASHION	DL2	0.000	0.056	0.112	0.168	0.224	0.279	0.335
	Exp. A	0.000	0.056	0.112	0.168	0.224	0.280	0.336
	Exp. B	0.000	0.056	0.113	0.169	0.225	0.281	0.336
	Exp. C	0.000	0.056	0.112	0.168	0.224	0.280	0.336
		0	0.001	0.002	0.003	0.004	0.005	0.006
GTSRB	DL2	0.000	0.045	0.090	0.136	0.181	0.226	0.271
	Exp. A	0.000	0.045	0.091	0.136	0.181	0.226	0.272
	Exp. B	0.000	0.045	0.090	0.136	0.181	0.226	0.271
	Exp. C	0.000	0.045	0.090	0.136	0.181	0.226	0.271

Table A.2: Mean distance values for the FGSM attack.

Dataset	Experiment	0	0.0025	0.005	0.0075	0.01	0.0125	0.015
MNIST	DL2	0.000	0.014	0.057	0.108	0.173	0.239	0.302
	Exp. A	0.000	0.006	0.037	0.076	0.124	0.175	0.221
	Exp. B	0.000	0.017	0.052	0.100	0.152	0.210	0.270
	Exp. C	0.000	0.014	0.047	0.079	0.116	0.163	0.210
FASHION	DL2	0.000	0.024	0.071	0.124	0.179	0.235	0.290
	Exp. A	0.000	0.026	0.073	0.127	0.182	0.235	0.291
	Exp. B	0.000	0.027	0.077	0.131	0.187	0.243	0.299
	Exp. C	0.000	0.025	0.072	0.124	0.178	0.233	0.289
		0	0.001	0.002	0.003	0.004	0.005	0.006
GTSRB	DL2	0.000	0.019	0.048	0.085	0.130	0.175	0.219
	Exp. A	0.000	0.007	0.040	0.080	0.136	0.190	0.239
	Exp. B	0.000	0.020	0.062	0.113	0.161	0.209	0.256
	Exp. C	0.000	0.015	0.048	0.093	0.141	0.189	0.236

Table A.3: Mean adversarial distance values for the FGSM attack.

Dataset	Experiment	0	0.0025	0.005	0.0075	0.01	0.0125	0.015
	DL2	0.000	0.066	0.132	0.198	0.264	0.329	0.394
MNIST	Exp. A	0.000	0.061	0.123	0.184	0.243	0.303	0.362
	Exp. B	0.000	0.064	0.129	0.191	0.254	0.317	0.380
	Exp. C	0.000	0.061	0.122	0.183	0.243	0.302	0.359
	DL2	0.000	0.069	0.139	0.208	0.277	0.346	0.415
FASHION	Exp. A	0.000	0.069	0.138	0.207	0.276	0.345	0.414
	Exp. B	0.000	0.069	0.138	0.207	0.276	0.345	0.414
	Exp. C	0.000	0.069	0.138	0.206	0.275	0.344	0.413
		0	0.001	0.002	0.003	0.004	0.005	0.006
	DL2	0.000	0.046	0.092	0.138	0.184	0.230	0.276
GTSRB	Exp. A	0.000	0.046	0.092	0.138	0.184	0.230	0.276
	Exp. B	0.000	0.046	0.092	0.138	0.184	0.230	0.276
	Exp. C	0.000	0.046	0.092	0.138	0.184	0.230	0.276

Table A.4: Max distance values for the FGSM attack.

Dataset	Experiment	0	0.0025	0.005	0.0075	0.01	0.0125	0.015
	DL2	0.000	0.060	0.126	0.189	0.250	0.316	0.384
MNIST	Exp. A	0.000	0.058	0.116	0.176	0.224	0.292	0.354
	Exp. B	0.000	0.060	0.125	0.188	0.249	0.311	0.373
	Exp. C	0.000	0.057	0.115	0.174	0.232	0.288	0.343
	DL2	0.000	0.068	0.139	0.208	0.277	0.346	0.415
FASHION	Exp. A	0.000	0.069	0.138	0.206	0.275	0.345	0.414
	Exp. B	0.000	0.068	0.136	0.204	0.276	0.345	0.414
	Exp. C	0.000	0.069	0.137	0.206	0.274	0.343	0.411
		0	0.001	0.002	0.003	0.004	0.005	0.006
	DL2	0.000	0.046	0.092	0.138	0.184	0.230	0.276
GTSRB	Exp. A	0.000	0.046	0.092	0.138	0.184	0.230	0.276
	Exp. B	0.000	0.046	0.092	0.138	0.184	0.230	0.276
	Exp. C	0.000	0.046	0.092	0.138	0.184	0.230	0.276

Table A.5: Max adversarial distance values for the FGSM attack.

Dataset	Experiment	0	0.0025	0.005	0.0075	0.01	0.0125	0.015
	DL2	99.50	99.34	98.97	98.54	97.52	95.60	92.39
MNIST	Exp. A	99.54	**99.48**	**99.31**	**99.14**	**98.94**	98.70	**98.56**
	Exp. B	**99.60**	99.43	99.24	98.98	98.70	98.29	97.77
	Exp. C	99.50	99.32	99.11	99.02	98.90	**98.71**	98.53
	DL2	92.18	87.21	81.82	76.74	72.07	67.72	64.35
FASHION	Exp. A	92.51	87.29	81.73	76.62	71.98	68.64	65.04
	Exp. B	92.73	87.04	81.15	75.62	70.32	66.12	62.65
	Exp. C	**92.76**	**87.93**	**82.89**	**78.22**	**74.23**	**70.82**	**67.46**
		0	0.001	0.002	0.003	0.004	0.005	0.006
	DL2	99.09	98.47	98.08	97.63	96.91	**96.21**	**95.56**
GTSRB	Exp. A	**99.33**	**99.21**	**98.80**	**98.39**	**97.39**	96.12	94.89
	Exp. B	99.14	98.47	97.31	95.13	92.88	90.31	87.70
	Exp. C	99.14	98.71	98.18	97.34	96.28	95.06	93.91

Table A.6: Accuracy values for the FGSM attack.

Dataset	Experiment	0	0.0025	0.005	0.0075	0.01	0.0125	0.015
	DL2	99.50	99.34	98.97	98.54	97.52	95.13	72.76
MNIST	Exp. A	99.54	**99.48**	**99.31**	**99.14**	**98.94**	98.70	91.31
	Exp. B	**99.60**	99.43	99.24	98.98	98.70	98.29	93.19
	Exp. C	99.50	99.32	99.11	99.02	98.90	**98.71**	**95.23**
	DL2	92.18	87.21	81.82	76.74	72.07	72.34	00.38
FASHION	Exp. A	92.51	87.29	81.73	76.62	71.98	73.26	01.06
	Exp. B	92.73	87.04	81.15	75.62	70.32	71.55	00.27
	Exp. C	**92.76**	**87.93**	**82.89**	**78.22**	**74.23**	**75.84**	**02.16**
		0	0.001	0.002	0.003	0.004	0.005	0.006
	DL2	99.09	98.47	98.08	97.63	96.91	**96.21**	**95.56**
GTSRB	Exp. A	**99.33**	**99.21**	**98.80**	**98.39**	**97.39**	96.12	94.89
	Exp. B	99.14	98.47	97.31	95.13	92.88	90.31	87.70
	Exp. C	99.14	98.71	98.18	97.34	96.28	95.06	93.91

Table A.7: Accuracy values for the FGSM attack excluding inputs with distance greater than 0.3.

Dataset	Experiment	0.02	0.03	0.04	0.05
MNIST	DL2	0.446	0.667	0.887	1.108
	Exp. A	0.426	0.636	0.846	1.056
	Exp. B	0.433	0.648	0.862	1.075
	Exp. C	0.421	0.628	0.836	1.043
FASHION	DL2	0.446	0.668	0.889	1.109
	Exp. A	0.447	0.669	0.890	1.111
	Exp. B	0.448	0.671	0.893	1.115
	Exp. C	0.447	0.670	0.891	1.113
		0.04	0.05	0.06	
GTSRB	DL2	1.798	2.245	2.687	
	Exp. A	1.804	2.252	2.695	
	Exp. B	1.800	2.247	2.690	
	Exp. C	1.800	2.247	2.690	

Table A.8: Mean distance values for the FGSM attack with bigger *eps*.

Dataset	Experiment	0.02	0.03	0.04	0.05
MNIST	DL2	88.12	82.42	77.68	73.08
	Exp. A	**98.17**	97.27	96.17	94.60
	Exp. B	96.52	93.80	91.23	88.27
	Exp. C	98.13	**97.44**	**96.50**	**95.27**
FASHION	DL2	58.88	51.23	46.14	41.78
	Exp. A	59.17	51.11	44.03	38.35
	Exp. B	56.70	48.91	43.10	38.49
	Exp. C	**62.23**	**53.58**	**47.39**	**42.64**
		0.04	0.05	0.06	
GTSRB	DL2	**68.78**	**63.65**	**59.88**	
	Exp. A	61.65	56.93	52.61	
	Exp. B	29.50	20.96	14.39	
	Exp. C	58.58	51.41	45.83	

Table A.9: Accuracy values for the FGSM attack with bigger *eps*.

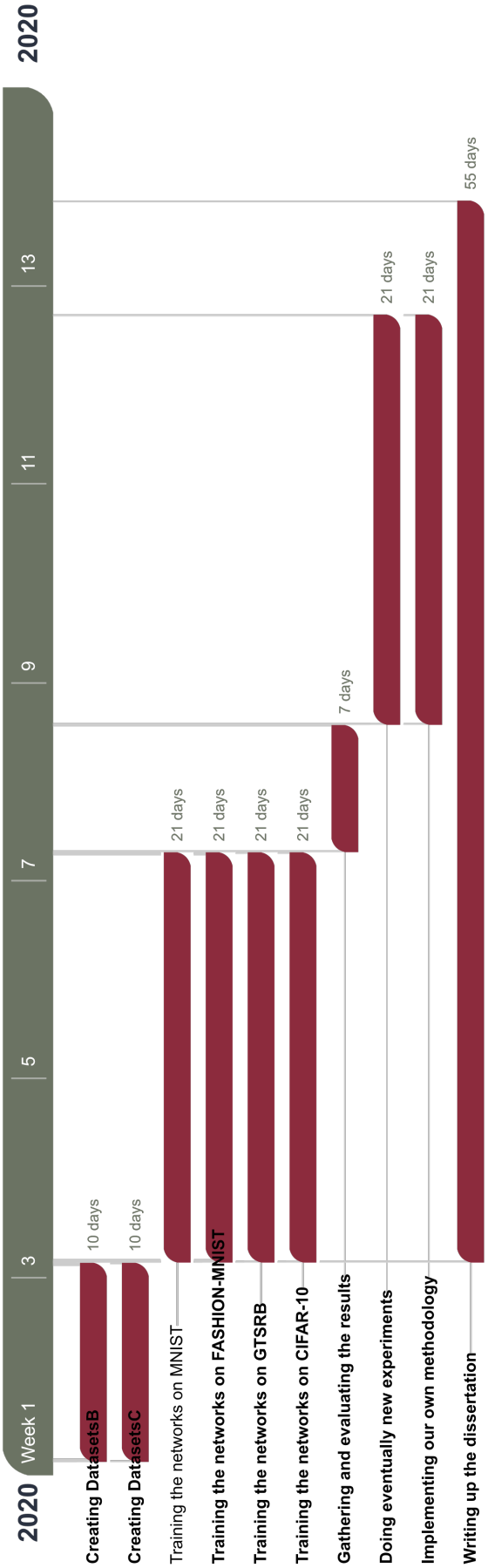

FIGURE A.1: Gantt Chart

Bibliography

Ayers, E., Eiras, F., Hawasly, M., and Whiteside, I. (2020). Parot: A practical framework for robust deep neural network training.

Binder, A., Montavon, G., Bach, S., Müller, K.-R., and Samek, W. (2016). Layer-wise relevance propagation for neural networks with local renormalization layers.

Carlini, N. and Wagner, D. (2017). Adversarial examples are not easily detected: Bypassing ten detection methods.

de Moura, L. and Bjørner, N. (2008). Z3: an efficient smt solver. volume 4963, pages 337–340.

Fischer, M., Balunovic, M., Drachsler-Cohen, D., Gehr, T., Zhang, C., and Vechev, M. (2019). Dl2: Training and querying neural networks with logic. In *International Conference on Machine Learning*.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014a). Generative adversarial networks.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). Explaining and harnessing adversarial examples.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

Hu, Z., Ma, X., Liu, Z., Hovy, E., and Xing, E. (2016). Harnessing deep neural networks with logic rules.

Huang, X., Kwiatkowska, M., Wang, S., and Wu, M. (2016). Safety verification of deep neural networks.

Katz, G., Barrett, C., Dill, D., Julian, K., and Kochenderfer, M. (2017). Reluplex: An efficient smt solver for verifying deep neural networks.

Kimmig, A., Bach, S., Broecheler, M., Huang, B., and Getoor, L. (2012). A short introduction to probabilistic soft logic. In *NIPS Workshop on PPFA*.

Louppe, G. (2014). Understanding random forests: From theory to practice.

Lundberg, S. and Lee, S.-I. (2017). A unified approach to interpreting model predictions.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks.

Mino, A. and Spanakis, G. (2018). Logan: Generating logos with a generative adversarial neural network conditioned on color.

Odena, A., Olah, C., and Shlens, J. (2016). Conditional image synthesis with auxiliary classifier gans.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144.

Rosenblatt, F. F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408.

Serban, A. C., Poll, E., and Visser, J. (2019). Adversarial examples - a complete characterisation of the phenomenon.

Shrikumar, A., Greenside, P., and Kundaje, A. (2017). Learning important features through propagating activation differences.

Stallkamp, J., Schlipsing, M., Salmen, J., and Igel, C. (2011). The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks.

Vargas, P. A. (2018). Biologically inspired computing: Neural computation. lecture 2.

Vargas, P. A. (2019). Biologically inspired computing: Neural computation. lecture 4.

Werbos, P. (1975). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

Xu, J., Zhang, Z., Friedman, T., Liang, Y., and den Broeck, G. V. (2017). A semantic loss function for deep learning with symbolic knowledge.

Yann, L. (2013). The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*.