National School of Computer Science and Systems Analysis - RABAT

End of Year Project Report : ADVERSARIAL MACHINE LEARNING

Made by :

Nada ARBIB
Zineb DAHMANI

Supervised by :

Pr. Abdellatif EL AFIA

Academic year 2021/2022

Acknowledgement

First of all, we would like to express our gratitude to our professor Abdellatif El Afia, for his confidence, his availability and especially this opportunity to master the Adversarial Machine Learning and to initiate our career with such a beautiful subject.

We would also like to thank all those who contributed to the success of this project.

Abstract

The present report is a synthesis of the work carried out within the framework of the project of end of year project , carried out within the ENSIAS, a big school of engineers specialized in Technologies of the Information and the Communication. Its missions are the training of state engineers and research for the technological and economic development of Morocco.

The mission we have been given is part of understanding attacks and defense adversarial machine .

Table des matières

Introduction

Adversarial ML is a way to misguide the machine learning model with a malicious input so that the model makes incorrect predictions. Some areas in which Adversarial ML are applicable include fraud detection, spam detection, intrusion detection, and malware detection.

We model secure learning systems as a game between an attacker and a defender—the attacker manipulates data to mistrain or evade a learning algorithm chosen by the defender to thwart the attacker's objective. This game can be formalized in terms of a learning algorithm H ; and the attacker's data corruption strategies $A^{(train)}$ and $A^{(eval)}$ The resulting game can be described as follows :

— **Defender :** Choose learning algorithm H for selecting hypotheses based on observed data
— **Attacker :** Choose attack procedures $A^{(train)}$ and $A^{(eval)}$ (potentially with knowledge of H)
— **Learning :** Learn hypothesis : f $\leftarrow$ H($D^{(train)}$)
— **Evaluation :** Compare predictions f (x) to y for each data point $(x, y) \in D^{(eval)}$

In 2004, Nilesh Dalvi and others noted that linear classifiers used in spam filters could be defeated by simple "evasion attacks" as spammers inserted "good words" into their spam emails. (Around 2007, some spammers added random noise to fuzz words within "image spam" in order to defeat OCR-based filters.) In 2006, Marco Barreno and others published "Can Machine Learning Be Secure ?", outlining a broad taxonomy of attacks. As late as 2013 many researchers continued to hope that non-linear classifiers (such as support vector machines and neural networks) might be robust to adversaries, until Battista Biggio and others demonstrated the first gradient-based attacks on such machine-learning models (2012-2013). In 2012, deep neural networks began to dominate computer vision problems ; starting in 2014, Christian Szegedy and others demonstrated that deep neural networks could be fooled by adversaries, again using a gradient-based attack to craft adversarial perturbations.

Recently, it was observed that adversarial attacks are harder to produce in the practical world due to the different environmental constraints that cancel out the effect of noises. For example, any small rotation or slight illumination on an adversarial image can destroy the adversariality. In addition, researchers such as Google Brain's Nicholas Frosst point out that it is much easier to make self-driving cars miss stop signs by physically removing the sign itself, rather than creating adversarial examples.[14] Frosst also believe that the adversarial machine learning community incorrectly assumes models trained on a certain data distribution will also perform well on a completely different data distribution. He suggests that a new approach to machine learning should be explored, and is currently working on a unique neural network that has characteristics more similar to human perception than state of the art approaches.

While adversarial machine learning continues to be heavily rooted in academia, large tech companies such as Google, Microsoft, and IBM have begun curating documentation and open source code bases to allow others to concretely assess the robustness of machine learning models and minimize the risk of adversarial attacks.

This work is a hands-on introduction to this topic of adversarial robustness in deep learning. The goal is combine both a mathematical presentation and illustrative code examples that highlight some of the key methods and challenges in this setting.

Chapitre 1

Presentation of the mission of the project

1.1 Context of the project

Deep neural network (DNN) architecture based models have high expressive power and learning capacity. However, they are essentially a black box method since it is not easy to mathematically formulate the functions that are learned within its many layers of representation. Realizing this, many researchers have started to design methods to exploit the drawbacks of deep learning based algorithms questioning their robustness and exposing their singularities through adversarial ML.

Adversarial Machine Learning in Image Classification is currently a very active research path which is responsible for most of the work in the area, with novel papers produced almost daily.However, there is neither a known efficient solution for securing Deep Learninng models nor any fully accepted explanations for the existence of adversarial images yet. Therefore, keeping in mind the importance of Adversarial Machine Learning in Image Classification to the development of more robust defenses and architectures against adversarial attacks.

1.1.1 Objective

Implementation and validation of adversarial attacks and defenses .

1.1.2 Approach

— Attack method : FGSM (Fast Gradient Sign Method).
— Defense method : GAN (Generative adversarial Networks).

Chapitre 2

Adversarial Attacks

2.1 Adversarial Attack

Adversarial examples are malicious inputs designed to fool machine learning models. They often transfer from one model to another, allowing attackers to mount black box attacks without knowledge of the target model's parameters.

- Creating an adversarial example :

So how do we manipulate the data to make the model believe it is something else ? To answer this, note that by common approach to training a model is to optimize the parameters , so as to minimize the average loss over some training set $x_i \in X, y_i, i = 1, ..., m$ which we write as the optimization problem

$$minimize_\theta \frac{1}{m} \sum_{i=1}^{m} l(h_\theta(x_i), y_i)$$

Which is typically solved by (stochastic) gradient descent. I.e., for some minibatch $B \subseteq 1, ..., m$, we compute the gradient of our loss with respect to the parameters θ and make a small adjustment to θ in the negative direction.

This is exactly what we're going to do to form an adversarial example. But instead of adjusting the data to minimize the loss, we're going to adjust the image to maximize the loss. That is, we want to solve the optimization problem :

$$maximize_{\hat{x}} \frac{1}{m} \sum_{i=1}^{m} l(h_\theta(\hat{x}), y)$$

where $\hat{x}$ denotes our adversarial example that is attempting to maximize the loss. Of course, we cannot just optimize arbitrarily over $\hat{x}$. So we instead need to ensure that $\hat{x}$ is close to our original input x.By convention, we typically do this by optimizing over the perturbation to x, which we will denote δ, and then by optimizing over δ ;

$$minimize_{\delta \in \Delta} \frac{1}{m} \sum_{i=1}^{m} l(h_\theta(x + \delta), y)$$

Where Δ represents an allowable set of perturbations. Characterizing the "correct" set of allowable perturbations is actually quite difficult : in theory.

2.2 Categories of attacks on ML

2.2.1 Attack Timing

- Attacks on training data (attacks on algorithms) :

The adversary can make arbitrary modifications to a small subset of training data points. The goal would be to design algorithms which are robust to such arbitrary training data as long as the corrupted data is small.

One common class of attacks are label-flipping attacks where the adversary is allowed to change labels of at most C data points in training data.

In the most cases one assumes that the algorithm and feature space are known to the adversary.

- Attacks on Decision Time (attacks on models) :

Of all the attack classes we will consider, evasion attacks—a major subclass of attacks which take place at decision time—are perhaps the most salient historically.

Are used in the decision (test) phase, assuming that the model has already been learned , the attacker changes its behavior or make changes to the observed environment.

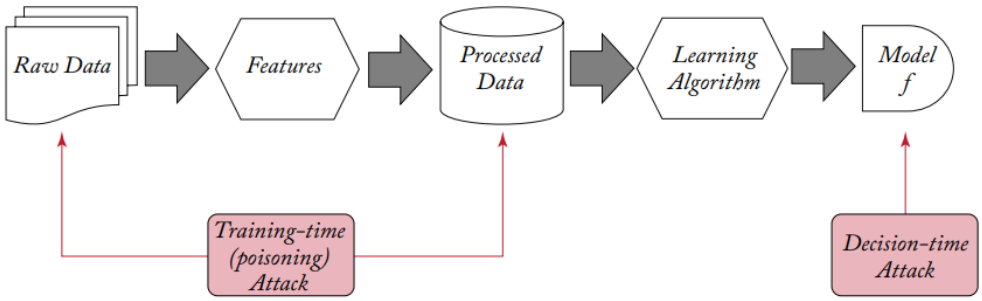


FIGURE 2.1 – A schematic representation of the distinction between decision-time attacks and poisoning attacks

2.2.2 Information available to the attacker

- White-box Attacks :

The adversary Knows all the model parameters including features and in the case of poisoning, the hyper-parameters of the algorithm.

If a learner can be robust to white box attacks, they are surely robust also to attacks which are informationally limited.

- Black-box Attacks :

In the context of decision time : At one extreme, no information is available to the adversary at all.A more informed adversary may have some training data which is different from the data on which the actual model had been trained, but no information about the particular model class being learned, or features used. A more informed attacker yet may know the model class and features, and perhaps the learning algorithm, but have no training data, and an even more informed adversary may also have training data sampled from the same distribution as the data used for learning.

Black-box data poisoning attacks would allow for a range of knowledge about the algorithm used by the defender.

2.3 Attack Methods

In this section, we review pioneer methods for generating adversarial examples attacks. Almost each one of these methods forms the basis of the real-world attacks and has the power of significantly affecting machine learning target models in practice.

2.3.1 Fast Gradient Sign Method - FGSM

FGSM is a sequential algorithm proposed by Goodfellow et al to sustain his linear hypothesis for explaining the existence of adversarial examples. The main characteristic of FGSM is its low computational cost, resulted from perturbing, in just one step (limited by a given upper bound), a legitimate image at the direction of the gradient that maximizes the model error. Despite its efficiency, the perturbations generated by FGSM are usually greater and less effective to fool models than the perturbations generated by iterative algorithms.

2.3.1.1 Untargeted attacks

It's a single step attack, ie.. the perturbation is added in a single step instead of adding it over a loop (Iterative attack). The perturbation in FGSM is given by the following equation :

$$x_{adv} = x + \epsilon.sign(\nabla_x J(x, y_{true})) \tag{2.1}$$

where :

$$\begin{cases} \mathbf{x} : CleanInputImage \\ x_{adv} : Adversarialexample \\ \mathbf{J} : Lossfunction \\ y_{true} : Modeloutputforx \\ \epsilon : Tunableparameter \end{cases}$$

The above equation performs an untargeted attack on the image ie.. the adversarial image generated is not targeted to maximize predictions for a particular class. Gradient can be calculated using back-propagation.

2.3.1.2 Targeted Attacks

For a targeted FGSM attack, the following equation is used :

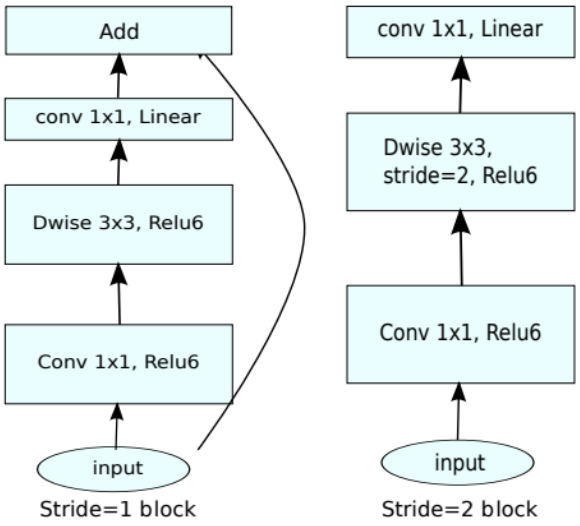
$$x_{adv} = x - \epsilon.sign(\nabla_x J(x, y_{target})) \tag{2.2}$$

2.3.1.3 Application

Let's try and fool a pretrained model using FGSM :

2.3.1.3.1 Model : We have used MobileNetV2 model, pretrained on ImageNet :

MobileNetV2 : MobileNetV2 is a convolutional neural network architecture that seeks to perform well on mobile devices. It is based on an inverted residual structure where the residual connections are between the bottleneck layers. The intermediate expansion layer uses lightweight depthwise convolutions to filter features as a source of non-linearity. As a whole, the architecture of MobileNetV2 contains the initial fully convolution layer with 32 filters, followed by 19 residual bottleneck layers.



(d) Mobilenet V2

FIGURE 2.2 – MobileNetV2 architecture

ImageNet : is an image dataset organized according to the WordNet hierarchy. Each meaningful concept in WordNet, possibly described by multiple words or word phrases, is called a "synonym set" or "synset". There are more than 100,000 synsets in WordNet ; the majority of them are nouns (80,000+). In ImageNet, we aim to provide on average 1000 images to illustrate each synset. Images of each concept are quality-controlled and human-annotated. In its completion, we hope ImageNet will offer tens of millions of cleanly labeled and sorted images for most of the concepts in the WordNet hierarchy. The project has been instrumental in advancing computer vision and deep learning research. The data is available for free to researchers for non-commercial use .

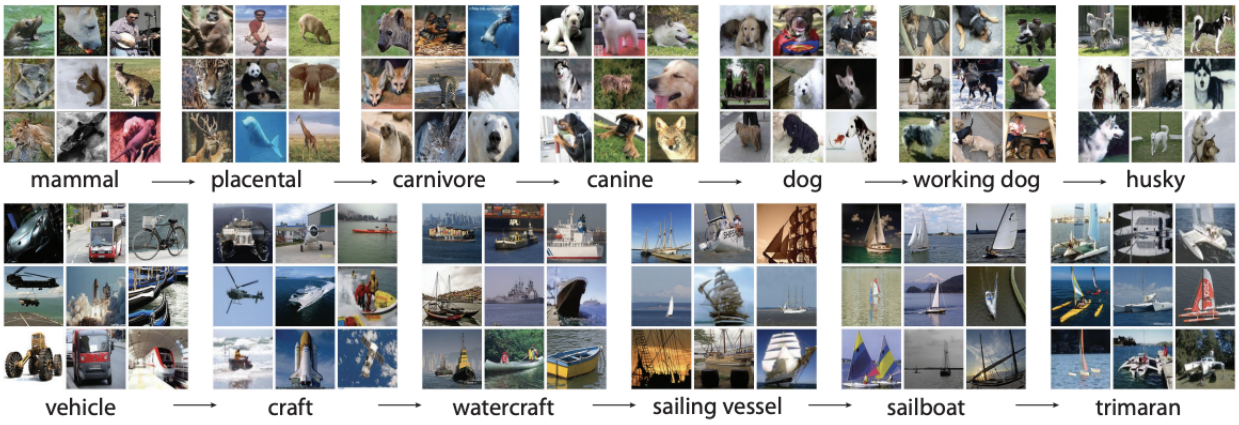


FIGURE 2.3 – Some examples from the Imagenet

Example choosed to work on : Let's use sample images and create adversarial examples from it. The first step is to preprocess it so that it can be fed as an input to the model :

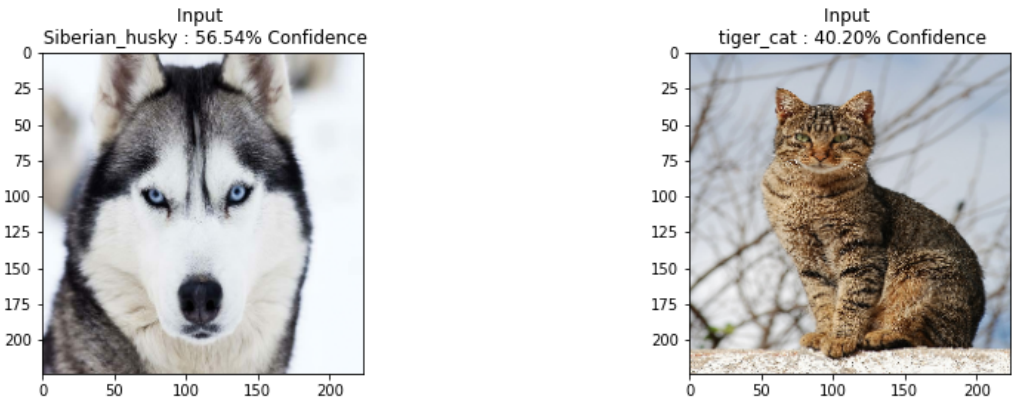


FIGURE 2.4 – Clear input images

Then we implement fast gradient sign method FGSM to create perturbations which will be used to distort the original image and to fool the model resulting in an adversarial image.

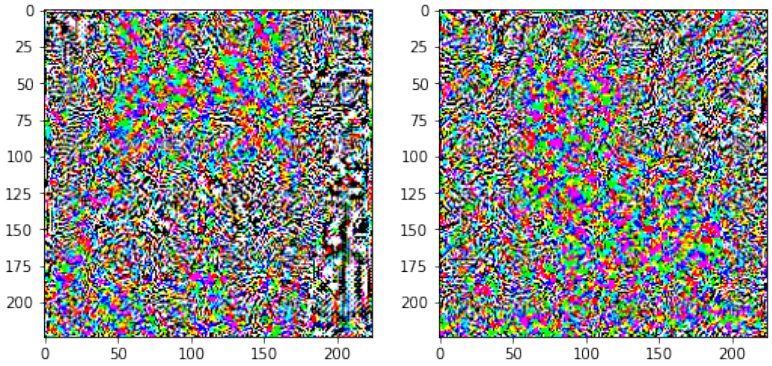


FIGURE 2.5 – Perturbations created respectively to the two images

So what does this Sebrian_husky and tiger_cat look like after applying FGSM ? Extremely similar to our original inputs, magnificently !So essentially, by adding a tiny multiple of this random-looking noise, we're able to create an image that looks identical to our original image, yet is classified very incorrectly.

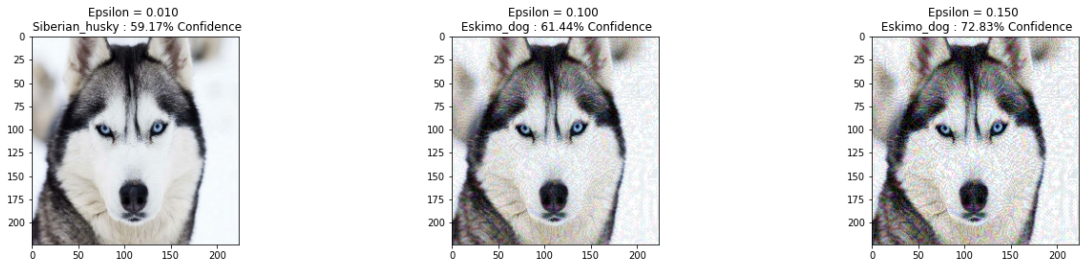


FIGURE 2.6 – Results for different epsilons for the first image

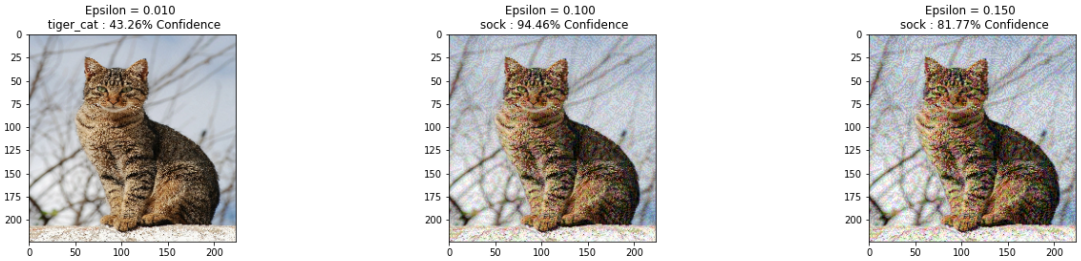


FIGURE 2.7 – Results for different epsilons for the second image

The conclusion, of course, is that with adversarial attacks and deep learning, you can make cats socks .

At first the model has recognized sebrian_husky with 56.54 % confidence and the cat with 40.20 % confidence. Instead, it turns out after applying FGSM that this classifier is quite sure with confidence 72.83 %[*epsilon = 0.15*] that the first image is for an eskimo_dog and the second one 94.46 % [*epsilon = 0.1*] is for sock.

FIGURE 2.8 – Eskimo dog and sock : Predicted classes after attack

2.3.2 Comparaison of attack algorithms

Algorithm and Reference	Perturbation Scope	Perturbation Visibility	Perturbation Measurement	Attacker's Knowledge	Attack Specificity	Attack Approach
FGSM [66]	individual	optimal, visible	L_∞	white-box	untargeted	gradient
JSMA [147]	individual	optimal	L_0	white-box	targeted	gradient
L-BFGS [175]	individual	optimal	L_∞	white-box	targeted	gradient
POBA-GA [29]	individual	optimal	custom	black-box	targeted, untargeted	decision
AutoZoom [182]	individual	optimal	L_2	black-box	targeted, untargeted	decision
DeepFool [132]	individual, universal	optimal	L_1, L_2, L_∞	white-box	untargeted	gradient
LaVAN [91]	individual, universal	visible	L_2	white-box	targeted	gradient
Universal Adversarial Networks (UAN) [73]	universal	optimal	L_2, L_∞	white-box	targeted	gradient
Expectation Over Transformation (EOT) [6]	individual	optimal	L_2	white-box	targeted	gradient
Local Search Attack (LSA) [136]	individual	optimal	L_0	black-box	targeted, untargeted	gradient
Natural Evolutionary Strategies (NES) [86]	individual	optimal	L_∞	black-box	targeted	approximation
Boundary Attack (BA) [15]	individual	optimal	L_2	black-box	targeted, untargeted	decision
CW Attack [23]	individual	optimal	L_0, L_2, L_∞	white-box	targeted, untargeted	gradient
GenAttack [3]	individual	optimal	L_2, L_∞	black-box	targeted	decision
BIM and ILCM [98]	individual	optimal	L_∞	white-box	untargeted	gradient
Momentum Iterative Method (M-BIM) [44]	individual	optimal	L_∞	white-box, black-box	untargeted	gradient
Zeroth-Order Optimization (ZOO) [31]	individual	optimal	L_2	black-box	targeted, untargeted	transfer, score
Hot-Cold Attack [152]	individual	optimal	L_2	white-box	targeted	gradient
Projected Gradient Descent (PGD) [123]	individual	optimal	L_1, L_∞	white-box	targeted	gradient
UPSET [156]	universal	optimal	L_2	black-box	targeted	gradient
ANGRI [156]	individual	optimal	L_2	black-box	targeted	gradient
Elastic-Net Attack (EAD) [30]	individual	optimal	L_1	white-box	targeted, untargeted	gradient
Hop-Skip-Jump Attack (HSJ) [27]	individual	optimal	L_2, L_∞	black-box	targeted, untargeted	decision
Robust Physical Perturbations (RP2) [51]	individual	physical	L_1, L_2	white-box	targeted	gradient
Ground-Truth Attack [20]	individual	optimal	L_1, L_∞	white-box	targeted	gradient
OptMargin [76]	individual	optimal	$L_0 1, L_2, L_\infty$	white-box	targeted	gradient
One-Pixel Attack [171]	individual	visible	L_0	black-box	targeted, untargeted	decision
BPDA [5]	individual	optimal	L_2, L_∞	black-box	untargeted, targeted	approximation
SPSA [183]	individual	optimal	L_∞	black-box	untargeted	approximation
Spatially Transformed Network (stAdv) [189]	individual	optimal	custom	white-box	targeted	gradient
AdvGAN [188]	individual	optimal	L_2	grey-box, black-box	targeted	gradient
Houdini [34]	individual	optimal	L_2, L_∞	black-box	targeted	gradient
Adversarial Transformation Networks (ATNs) [9]	individual	optimal	L_∞	white-box	targeted	gradient

FIGURE 2.9 – Main adversarial attack algorithms in Computer Vision

Chapitre 3

Adversarial Defense : Defence Mechanism

3.1 Adversarial Training

A popular approach to defend against adversarial noise is to augment the training dataset with adversarial examples. Adversarial examples are generated using one or more chosen attack models and added to the training set. This often results in increased robustness when the attack model used to generate the augmented training set is the same as that used by the attacker.
However, adversarial training does not perform as well when a different attack strategy is used by the attacker. Additionally, it tends to make the model more robust to white-box attacks than to black-box attacks due to gradient masking.

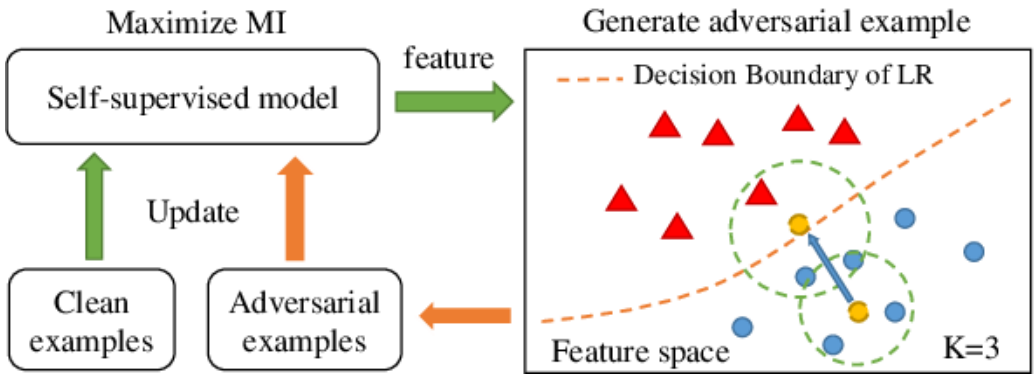


FIGURE 3.1 – DCGAN generator

3.2 Defensive Distillation

Distillation is a heuristic technique for training deep neural networks initially proposed for transferring knowledge from a more to a less complex model (essentially, for compression). Using distillation to make deep neural networks more robust to adversarial noise.
The distillation approach works as follows :
1. Start with the original training dataset $D = \{x_i, y_i\}$, where labels yi are encoded as onehot vectors (i.e., all zeros, except for a 1 in the position corresponding to the true class of x_i).
2. Train a deep neural network after replacing the softmax function in the softmax layer with

$$F_i(x) = \frac{\exp \frac{Z_i(x)}{T}}{\sum_j \exp \frac{Z_i(x)}{T}}$$

for an exogenously chosen shared temperature parameter T .
3. Create a new training dataset $D' = \{x_i, y_i'\}$, where $y_i' = F(x_i)$with $F(.)$ the soft (probabilistic) class labels returned by the previously trained neural network.
4. Train a new deep neural network with the same temperature parameter T as the first one,but on the new dataset D'.
5. Use the retrained neural network after eliminating the temperature T from the final (softmax) layer (i.e., setting $T = 1$ at test time, scaling the softmax terms down by a factor of T , and thereby increasing the sharpness

of predicted class probabilities).

Defensive distillation trains the classifier in two rounds using a variant of the distillation method. This has the desirable effect of learning a smoother network and reducing the amplitude of gradients around input points, making it difficult for attackers to generate adversarial examples . It was, however, shown that, while defensive distillation is effective against white-box attacks, it fails to adequately protect against black-box attacks transferred from other networks .

3.3 Generative Adversarial Networks

3.3.1 What are GANs

GANs, originally introduced by Goodfellow et al. (2014), consist of two neural networks,G and D. $G : R \to R$ n maps a low-dimensional latent space to the high dimensional sample space of x. D is a binary neural network classifier.In the training phase, G and D are typically learned in an adversarial fashion using actual input data samples x and random vectors z. An isotropic Gaussian prior is usually assumed on z. While G learns to generate outputs $G(z)$ that have a distribution similar to that of x, D learns to discriminate between "real" samples x and "fake" samples $G(z)$.

$$min_G max_D V(D,G) = E_{x\sim p_{data}}[logD(x)] + E_{z\sim p_z(z)}[log(1 - D(G(z)))]$$

It was shown that the optimal GAN is obtained when the resulting generator distribution $p_g = p_{data}$.

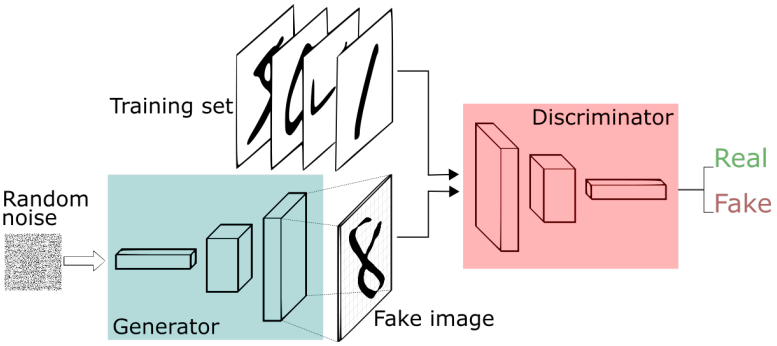


FIGURE 3.2 – GANs Representation

GANs havebeen known to be unstable to train, often resulting in generators that produce nonsensicaloutputs.There has been very limited published research in trying to understand and visualize what GANs learn, and the intermediate representations of multi-layer GANs.

3.3.2 How do GANs work

GANs make use of generative algorithms — contrasted from discriminative algorithms which classify input data, or features, by predicting their labels, categories, or fields — to determine the distribution or probability of a data point being classified as a particular feature based on its label, category, or field. These two aforementioned algorithms are executed by the discriminator and generator, respectively, as illustrated in the image below :

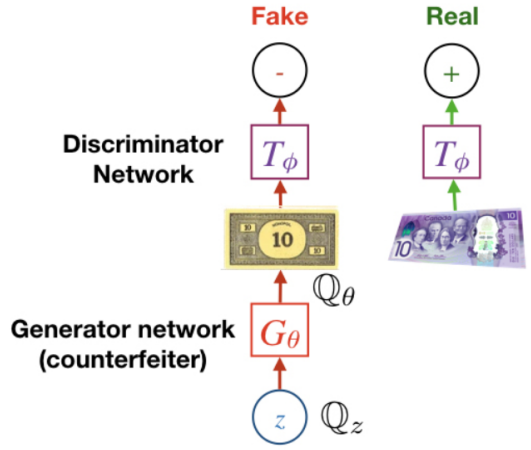


FIGURE 3.3 – Generator versus Discriminator

3.3.3 Theoretical Results

The generator G implicitly defines a probability distribution p_g as the distribution of the samples $G(z)$ obtained when $z \sim p_z$. Therefore, we would like the Algorithm to converge to a good estimator of p_{data}, if given enough capacity and training time. The results of this section are done in a nonparametric setting(statistic that does not make any assumptions about the characteristics of the sample), e.g. we represent a model with infinite capacity by studying convergence in the space of probability density functions.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

FIGURE 3.4 – GANs Algorithm

— **Convergence of Algorithm :**

If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion

$$E_{x \sim p_{data}}[logD^*_G(x)] + E_{x \sim p_g}[log(1 - D^*_G(x))]$$

then p_g converges to p_{data}.

3.4 Deep convolutional GAN : DCGAN

3.4.1 Learning From Unlabeled Data

Unsupervised representation learning is a fairly well studied problem in general computer vision research, as well as in the context of images. A classic approach to unsupervised representation learning is to do clustering on the data (for example using K-means), and leverage the clusters for improved classification scores. In the context of images, one can do hierarchical clustering of image patches to learn powerful image representations. Another popular method is to train auto-encoders (convolutionally, stacked , separating the what and where components of the code , ladder structures) that encode an image into a compact code, and decode the code to reconstruct the image as accurately as possible. These methods have also been shown to learn good feature representations from image pixels. Deep belief networks have also been shown to work well in learning hierarchical representations.

3.4.2 Generating Natural Images

Parametric models for generating images has been explored extensively . However, generating natural images of the real world have had not much success until recently. A variational sampling approach to generating images has had some success, but the samples often suffer from being blurry.Generative Adversarial Networks generated images suffering from being noisy and incomprehensible. A laplacian pyramid extension to this approach showed higher quality images, but they still suffered from the objects looking wobbly because of noise introduced in chaining multiple models. A recurrent network approach and a deconvolution network approach have also recently had some success with generating natural images. However, they have not leveraged the generators for supervised tasks.

3.4.3 Model Architecture

Core to our approach is adopting and modifying three recently demonstrated changes to CNN architectures.
- The first is the convolutional net which replaces deterministic pooling functions (such as maxpooling) with strided convolutions, allowing the network to learn its own spatial down-sampling. We use this approach in our generator,allowing it to learn its own spatial up-sampling, and discriminator.

- Second is the trend towards eliminating fully connected layers on top of convolutional features.The strongest example of this is global average pooling which has been utilized in state of the art image classification models. We found global average pooling increased model stability but hurt convergence speed. A middle ground of directly connecting the highest convolutional features to the input and output respectively of the generator and discriminator worked well. The first layer of the GAN, which takes a uniform noise distribution Z as input, could be called fully connected as it is just a matrix multiplication, but the result is reshaped into a 4-dimensional tensor and used as the start of the convolution stack. For the discriminator, the last convolution layer is flattened and then fed into a single sigmoid output.

- Third is Batch Normalization which stabilizes learning by normalizing the input to each unit to have zero mean and unit variance. This helps deal with training problems that arise due to poor initialization and helps gradient flow in deeper models. This proved critical to get deep generators to begin learning, preventing the generator from collapsing all samples to a single point which is a common failure mode observed in GANs. Directly applying batchnorm to all layers however, resulted in sample oscillation and model instability. This was avoided by not applying batchnorm to the generator output layer and the discriminator input layer.

- The ReLU activation is used in the generator with the exception of the output layer which uses the Tanh function. We observed that using a bounded activation allowed the model to learn more quickly to saturate and cover the color space of the training distribution. Within the discriminator we found the leaky rectified activation to work well, especially for higher resolution modeling. This is in contrast to the original GAN paper, whichused the maxout activation.

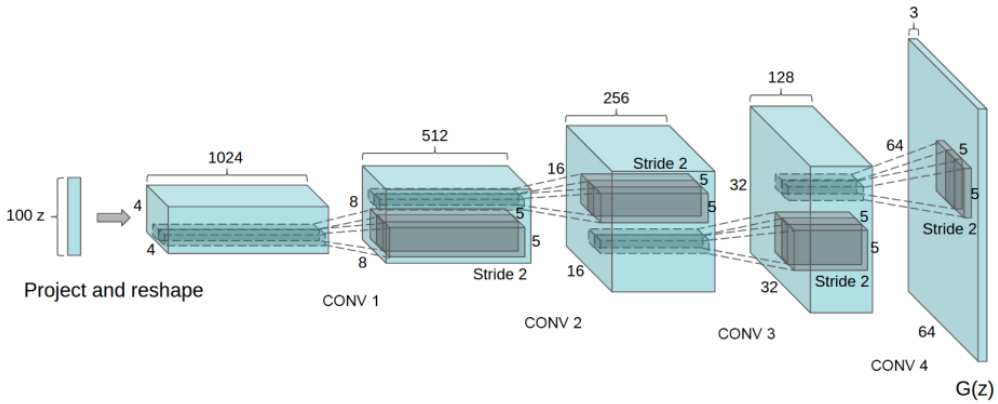


FIGURE 3.5 – DCGAN generator

3.5 Defense GAN

3.5.1 Wasserstein GAN

GANs turned out to be difficult to train in practice , and alternative formulations have been proposed. Introduced Wasserstein GANs (WGANs) which are a variant of GANs that use the Wasserstein distance, resulting in a loss function with more desirable properties :

$$min_G max_D V(D,G) = E_{x\sim p_{data}}[D(x)] + E_{z\sim p_z(z)}[D(G(z))]$$

we use WGANs as our generative model due to the stability of their training methods.

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.
while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

FIGURE 3.6 – WGAN Algorithm

There are significant practical benefits to using it over the formulation used in standard GANs.
We claim two main benefits :
- a meaningful loss metric that correlates with the generator's convergence and sample quality.
- improved stability of the optimization process

One of the benefits of WGAN is that it allows us to train the critic till optimality. When the critic is trained to completion, it simply provides a loss to the generator that we can train as any other neural network. This tells us that we no longer need to balance generator and discriminator's capacity properly. The better the critic, the higher quality the gradients we use to train the generator.

3.5.2 Defense GAN algorithm

Defense-GAN is a defense strategy to combat both white-box and black-box adversarial attacksagainst classification networks. At inference time, given a trained GAN generator G and an imagex to be classified, z^*

is first found so as to minimiz

$$min_Z ||G(z) - x||_2^2$$

$G(z^*)$ is then given as the input to the classifier. The GAN is trained on the available classifier training dataset in an unsupervised manner. The classifier can be trained on the original training images, their reconstructions using the generator G, or a combination of the two.

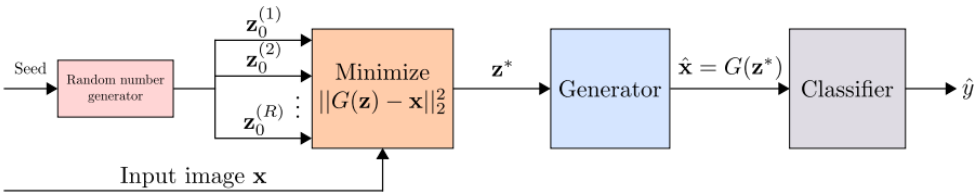


FIGURE 3.7 – Overview of the Defense-GAN algorithm.

- Optimization Algorithm :

L steps of Gradient Descent are used to estimate the projection of the image onto the range of the generator.
1- Pick a random number 'z' , generate G(z)
2- Check the loss function between G(z) and the input image 'x'
3- Iterate towards best G(z) through Gradient Descent -L steps
4- Pick another random 'z' , repeat steps 1-3 for R random restarts
5- Find the arg min from all the generated G(z)

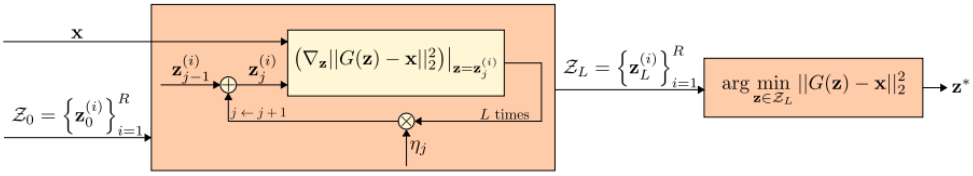


FIGURE 3.8 – L steps of Gradient Descent Optimization Algorithm

3.6 Applications

3.6.1 MNIST Data-Set

For the different implementation in this section we used the MNIST dataset.MNIST is a widely used dataset of handwritten digits that contains 60,000 handwritten digits for training a machine learning model and 10,000 handwritten digits for testing the model. It was introduced in 1998 and has become a standard benchmark for classification tasks. It is also called the "Hello, World" dataset as it's very easy to use. MNIST was derived from an even larger dataset, the NIST Special Database 19 which not only contains digits but also uppercase and lowercase handwritten letters.

In the MNIST dataset each digit is stored in a grayscale image with a size of 28x28 pixels. In the following you can see the first 10 digits from the training set :

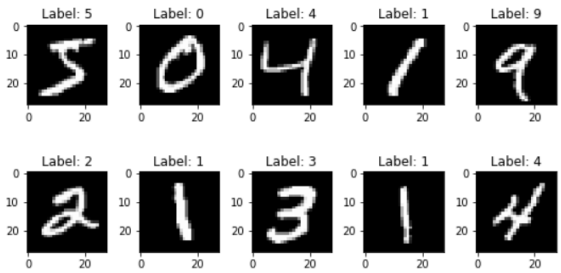


FIGURE 3.9 – Some examples of the MNIST dataset

3.6.2 Adversarial Example :Attack and defense

In this example we did a PyTorch implementation of the non-target adversarial example attack (white box) and one defense method as countermeasure to those attacks.

-Attack method : Fast Gradient Sign Method(FGSM) .

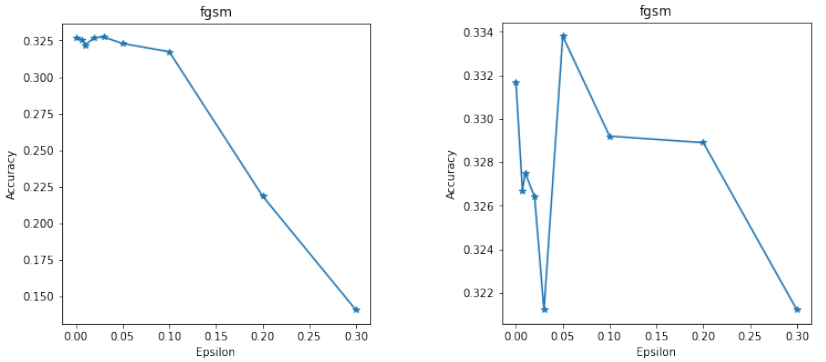
-Defence method : Defensive Distillation .

-Results :

- Applied the attack methods and defense using MNIST dataset on the model based on pytorch example model for mnist.

Here, the attacks are white box as all the knowledge of network hyperparameter setting with the network's achitecture.

- Results tell that FGSM attack reduces the test accuracy from 97.% to 24.84% with epsilon from 0 to 0.3.



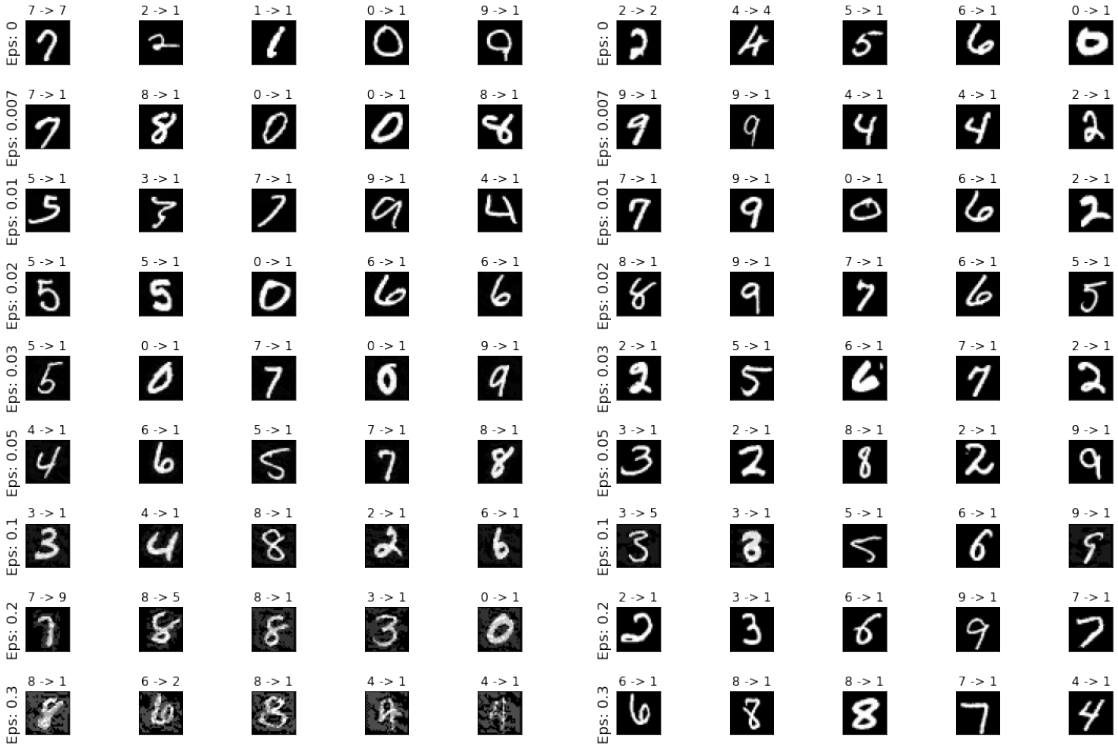
(a) Test Accuracy during attacks

(b) Test Accuracy during attack using defensive distillation

FIGURE 3.10

During the defensive distillation used same network as Net F and for Net F' reduced number of filters to half in each layer to reduce the number of parameters. Temperature of 100 was taken in our case. Results tell that FGSM attack reduces test accuracy from 90.33% to 88.01% with same epsilon range. We can say that defensive distillation for the proposed network with temp of 100 was successful and it only reduced 2% of test accuracy in our case for max epsilon of 0.3.

Sample Adversarial Examples



(a) Sample Adversarial Examples

(b) Sample Adversarial Examples using defensive distillation

FIGURE 3.11

3.6.3 GAN's Implementation

We trained GAN on MNIST dataset. The generator nets used Rectified Linear Unit activations (LeakyReLU), and we did same for the discriminator net . While our theoretical framework permits the use of dropout and other noise at intermediate layers of the generator, we used noise as the input to only the bottommost layer of the generator network.

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 784)               0

 dense_4 (Dense)             (None, 512)               401920

 leaky_re_lu_8 (LeakyReLU)   (None, 512)               0

 dense_5 (Dense)             (None, 256)               131328

 leaky_re_lu_9 (LeakyReLU)   (None, 256)               0

 dense_6 (Dense)             (None, 1)                 257

=================================================================
Total params: 533,505
Trainable params: 533,505
Non-trainable params: 0
```

FIGURE 3.12 – The generator's model

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 256)               25856

 leaky_re_lu_10 (LeakyReLU)  (None, 256)               0

 batch_normalization_10 (Bat  (None, 256)              1024
 chNormalization)

 dense_8 (Dense)             (None, 512)               131584

 leaky_re_lu_11 (LeakyReLU)  (None, 512)               0

 batch_normalization_11 (Bat  (None, 512)              2048
 chNormalization)

 dense_9 (Dense)             (None, 1024)              525312

 leaky_re_lu_12 (LeakyReLU)  (None, 1024)              0

 batch_normalization_12 (Bat  (None, 1024)             4096
 chNormalization)

 dense_10 (Dense)            (None, 784)               803600

 reshape_2 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 1,493,520
Trainable params: 1,489,936
Non-trainable params: 3,584
_________________________________________________________________
```

FIGURE 3.13 – The discriminator's model summary

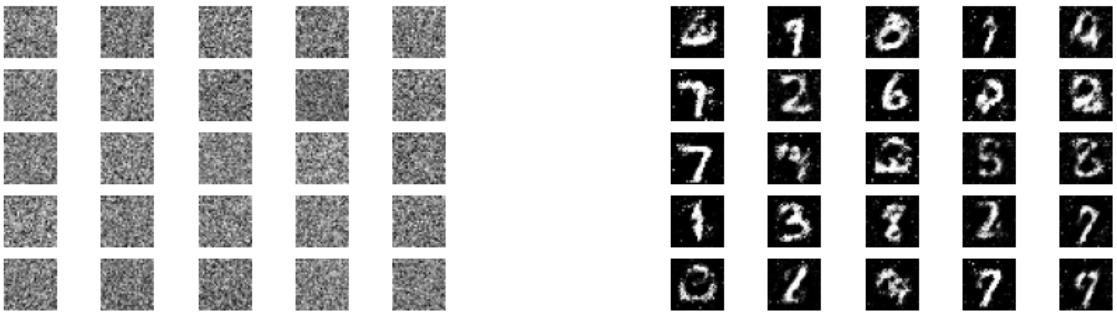
Leaky Rectified Linear Unit, or Leaky ReLU, is a type of activation function based on a ReLU, but it has a small slope for negative values instead of a flat slope.

Training :
Training will involve alternating between training the discriminator and the generator. We'll use our functions real loss and fake loss to help us calculate the discriminator losses in all of the following cases.

Discriminator training :
- Compute the discriminator loss on real, training images .
- Generate fake images .
- Compute the discriminator loss on fake, generated images .
- Add up the real and fake loss .
- Perform backpropagation + an optimization step to update the discriminator's weights .
- Generator training :

Generate fake images
- Compute the discriminator loss on fake images, using flipped labels.
- Perform backpropagation + an optimization step to update the generator's weights .

-Results : Generator samples from training : After training for about 30000 epochs. Below I'm showing the generated images as the network was training, every 200 epochs.



(a) (b)

FIGURE 3.14 – Results

It starts out as all noise. Then it learns to make only the center white and the rest black.

3.6.4 DCGAN's Implementation

The difference between the simple GAN and the DCGAN, is the generator of the DCGAN uses the transposed convolution (Fractionally-strided convolution or Deconvolution) technique to perform up-sampling of 2D image size. DCGAN are mainly composes of :
- Convolution layers without max pooling or fully connected layers .
- It uses convolutional stride and transposed convolution for the downsampling and the upsampling .

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 6272)	633472
reshape (Reshape)	(None, 7, 7, 128)	0
up_sampling2d (UpSampling2D)	(None, 14, 14, 128)	0
conv2d_4 (Conv2D)	(None, 14, 14, 128)	147584
batch_normalization_3 (BatchNormalization)	(None, 14, 14, 128)	512
activation (Activation)	(None, 14, 14, 128)	0
up_sampling2d_1 (UpSampling2D)	(None, 28, 28, 128)	0
conv2d_5 (Conv2D)	(None, 28, 28, 64)	73792
batch_normalization_4 (BatchNormalization)	(None, 28, 28, 64)	256
activation_1 (Activation)	(None, 28, 28, 64)	0
conv2d_6 (Conv2D)	(None, 28, 28, 1)	577
activation_2 (Activation)	(None, 28, 28, 1)	0

Total params: 856,193
Trainable params: 855,809
Non-trainable params: 384

FIGURE 3.15 – The discriminator's model summary

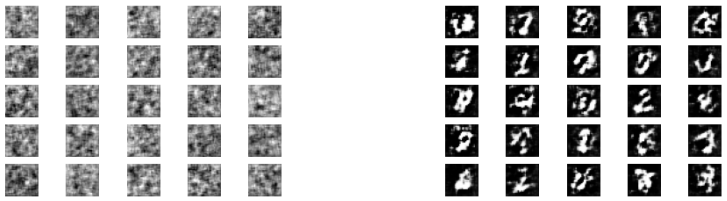
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 32)	320
leaky_re_lu (LeakyReLU)	(None, 14, 14, 32)	0
dropout (Dropout)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 7, 7, 64)	18496
zero_padding2d (ZeroPadding2D)	(None, 8, 8, 64)	0
batch_normalization (BatchNormalization)	(None, 8, 8, 64)	256
leaky_re_lu_1 (LeakyReLU)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 128)	73856
batch_normalization_1 (BatchNormalization)	(None, 4, 4, 128)	512
leaky_re_lu_2 (LeakyReLU)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
conv2d_3 (Conv2D)	(None, 4, 4, 256)	295168
batch_normalization_2 (BatchNormalization)	(None, 4, 4, 256)	1024
leaky_re_lu_3 (LeakyReLU)	(None, 4, 4, 256)	0
dropout_3 (Dropout)	(None, 4, 4, 256)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 1)	4097

Total params: 393,729
Trainable params: 392,833
Non-trainable params: 896

FIGURE 3.16 – The generator's model summary

-Results :

After training for about 4000 epochs. Below I'm showing the generated images as the network was training, every 50 epochs. We noticed that for the 300 first epochs the DCGAN generator was doing well (fig 3.12) but after that the image generated was noisy and fuzzy (fig 3.13), and that is a common problem in DCGAN that the discriminator is too good, then generator training can fail due to vanishing gradients.
The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training.



(a) The initial noise genarated (b) image generating after 250 epochs

FIGURE 3.17

(a) The image generating after 2500 epochs

FIGURE 3.18

3.6.5 WGAN's Implementation

In WGAN, the discriminator does not classify the input as real or fake instead, it outputs a number. Discriminator training tries to make the output bigger for real images than for fake images. It does this by removing the last Sigmoid() layer and have a linear layer at the end of the discriminator's neural network. As the discriminator is no longer classifying between real and fake images and outputs a score or a rating, the discriminator in WGAN is referred to as a Critic.

-Training : Set the clip_value parameter to ensure that the critic's parameters do not exceed a value between -0.01 to 0.01.

Also, training the critic more than the generator using the n_critic parameter ensures that the critic does not saturate to prevent mode collapse.

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_21 (Dense)            (None, 6272)              633472

 reshape_7 (Reshape)         (None, 7, 7, 128)         0

 up_sampling2d_12 (UpSamplin  (None, 14, 14, 128)      0
 g2D)

 conv2d_50 (Conv2D)          (None, 14, 14, 128)       262272

 batch_normalization_39 (Bat  (None, 14, 14, 128)      512
 chNormalization)

 activation_18 (Activation)  (None, 14, 14, 128)       0

 up_sampling2d_13 (UpSamplin  (None, 28, 28, 128)      0
 g2D)

 conv2d_51 (Conv2D)          (None, 28, 28, 64)        131136

 batch_normalization_40 (Bat  (None, 28, 28, 64)       256
 chNormalization)

 activation_19 (Activation)  (None, 28, 28, 64)        0

 conv2d_52 (Conv2D)          (None, 28, 28, 1)         1025

 activation_20 (Activation)  (None, 28, 28, 1)         0

=================================================================
Total params: 1,028,673
Trainable params: 1,028,289
Non-trainable params: 384
_________________________________________________________________
```

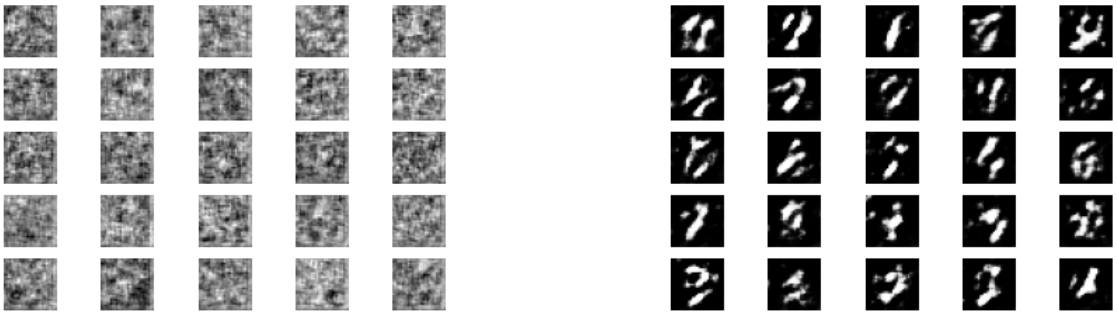
FIGURE 3.19 – The discriminator's model summary

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_46 (Conv2D)          (None, 14, 14, 16)        160

 leaky_re_lu_33 (LeakyReLU)  (None, 14, 14, 16)        0

 dropout_28 (Dropout)        (None, 14, 14, 16)        0

 conv2d_47 (Conv2D)          (None, 7, 7, 32)          4640

 zero_padding2d_7 (ZeroPaddi  (None, 8, 8, 32)         0
 ng2D)

 batch_normalization_36 (Bat  (None, 8, 8, 32)         128
 chNormalization)

 leaky_re_lu_34 (LeakyReLU)  (None, 8, 8, 32)          0

 dropout_29 (Dropout)        (None, 8, 8, 32)          0

 conv2d_48 (Conv2D)          (None, 4, 4, 64)          18496

 batch_normalization_37 (Bat  (None, 4, 4, 64)         256
 chNormalization)

 leaky_re_lu_35 (LeakyReLU)  (None, 4, 4, 64)          0

 dropout_30 (Dropout)        (None, 4, 4, 64)          0

 conv2d_49 (Conv2D)          (None, 4, 4, 128)         73856

 batch_normalization_38 (Bat  (None, 4, 4, 128)        512
 chNormalization)

 leaky_re_lu_36 (LeakyReLU)  (None, 4, 4, 128)         0

 dropout_31 (Dropout)        (None, 4, 4, 128)         0

 flatten_8 (Flatten)         (None, 2048)              0

 dense_20 (Dense)            (None, 1)                 2049

=================================================================
Total params: 100,097
Trainable params: 99,649
Non-trainable params: 448
```

FIGURE 3.20 – The generator's model summary

-Results : After training for about 4000 epochs. Below I'm showing the generated images as the network was training, every 50 epochs.

WGAN solves the issues seen in DCGAN like the mode collapse and vanishing gradients challenges by using Wasserstein loss. W-Loss need to satisfy a special condition called as 1-Lipschitz Continuous, which can be implemented by clipping the parameters of the critic or using the gradient penalty where we penalize the gradient norm of the critic's output with respect to the interpolated image, which is a randomly weighted average between real and fake image.



(a) The initial noise genarated

(b) The image generating after 600 epochs

FIGURE 3.21

Conclusion

This project was very enriching for us because it allowed us to discover and make a step towards the topic of adversarial Machine learning, its actors, its constraints but also to participate concretely in its stakes through the missions which were given to us : Implementing Attacks and defense techniques .

Through this project we were able to meet the requested tasks that we discussed at the beginning of chapter 1.This was accomplished by using FGSM as attack method and GAN , DCGAN and WGAN as defense methods.

We have faced several problems to deal for the first time with new librairies such as : tensotflow and pytorch .

Bibliographie

[1] Generative Adversarial Nets,<https://arxiv.org/pdf/1406.2661.pdf>.

[2] Adversarial Attacks and Defences : A Survey,<https://arxiv.org/pdf/1810.00069.pdf>.

[3] Defending Against Adversarial Attacks by Leveraging an Entire GAN ,<https://arxiv.org/pdf/1805.10652.pdf>.

[4] Unsupervised Representation Learning With Deep Convolutionnal Generative Adversarial Networks,<https://arxiv.org/pdf/1511.06434.pdf>.

[5] Wasserstein GAN , <https://arxiv.org/pdf/1701.07875.pdf>.

[6] Adversarial Machine Learning in Image Classification : A Survey Towards the Defender's Perspective ,<https://arxiv.org/pdf/2009.03728.pdf>.