**UNIVERSITY KASDI-MERBAH OUARGLA**
**Faculty of New Technologies of Information and Communication**
**Department of Computer Science and Information Technologies**

**Master Thesis**

**Specialty:** Computer Science Fundamentals

**To obtain diploma**
**Master of Computer Science**

**Thesis subject:**

# Generating Arabic Letters using Generative Adversarial Networks (GANs)

**Submitted by :**
Abderrahmane CHEBOUAT

Mr.Abdelhakim CHERIET **as Supervisor**
Mr.Amine KHALDI (P)**as Examiner**
Mr.Fouad BEKKARI **as Examiner**

University Year: 2018/2019

## Abstract

The Generative modeling became recently one of the most important field in deep learning, and the Generative Adversarial Networks is the new research line in this field, the GANs have proven their ability to generate a high resolution of images and achieved remarkable success for computer vision in general. by the same way, we assume that this tool will be able to generate accurate Arabic letters.

The expected benefit of this work is creating a perception by using GANs and explore the potential of this deep learning powerful tool to serve the Arabic calligraphy. As an impact for further studies in the future, this work may provide an insight into the possibility of generating new kinds of Arabic fonts (calligraphy).

Keywords: Com-puter vision , Arabic calligraphy, Deep learning, Generative modelling, Generative Adversarial Networks, .

## Résumé:

La modélisation générative est récemment devenue le domaine le plus important dans apprentissage approfondi, et les réseaux d'adversaire génératif sont la nouvelle ligne de recherche dans ce domaine. Les GANs ont démontré leur capacité à générer une haute résolution des images et ont obtenu un succès remarquable pour la vision par ordinateur en général. de la même manière, nous supposons que cet outil sera capable de générer des lettres arabes précises.

Le profit attendu de ce travail est de créer une perception de l'utilisation des GANs et d'explorer le potentiel de cet outil puissant de DL pour servir la calligraphie arabe. Ce travail, qui pourrait avoir un impact sur les futures études, pourrait fournir des informations sur la possibilité de générer de nouveaux types de polices arabes (calligraphie). Mots-clés: vision par ordinateur, Arabe calligraphie,Apprentissage approfondi, Modélisation générative, Réseaux d'adversaire génératif.

# Contents

# List of Figures

# General introduction

Text generation has had a large application in recent years, but there are still great prospects to be discovered in this field, using new deep learning techniques such as Generative Adversarial Networks we can push the development in this area and make it a very smooth.
With the appearance of GANs there was no tendency to use them for text generation, and there are only very few studies related to this subject, And as it the calligraphy processing intersect in large portion with image processing we would like to explore the potential of GANs algorithms regarding letters generation in order to achieve a study that allows us to judge the prospects in this topic.
This work aims to study the possibility of generating letters with Generative Adversarial Networks, the GANs belong to generative models where the architectures are combined from different serval architectures of Neural Networks, that's might make it powerful enough to generate a very kinds of data, such an images, audio, video , and text, below a set of research questions, are enumerated, to be answered through this study:

- What're the differences between a simple neural network and a GANs?
- Why are some GANs architectures performing better than other ones?
- Is the quality of the dataset affecting the performance of GANs?
- How will be used an Arabic letters dataset to generate a new type of Arabic calligraphy?

- Is it possible to get enough level of quality in the generated Arabic letters, so that makes the used model to produce new Arabic calligraphy?
- Finally, What're the ideal and best tools  libraries that should be used to achieve the goal of this study?

The study project will be structured as follows: the first chapter will give an overview of the deep learning field to understand better the current situation. In the second chapter, will be fully dedicated to understand the GANs by explaining all related details, then we will explain in detail the DCGAN which is a direct extension of the GANs with algorithms and architectures that have performed better than GAN. In the last chapter, we will explain the implementation of our work, with used environments then review the obtained results, and Conclusion.

# Chapter 1

# Deep Learning

## 1.1 Introduction

Deep learning is a subset of Machine Learning, which is a subset of artificial intelligence (AI), the idea of deep learning based on simulating or imitates the human brain way in processing data and extracting patterns to learn and make intelligent decisions.
The deep learning has become recently (last few years) a very interesting and attractive search domain for all companies working to produce AI systems.

In this chapter, we will try to define the deep learning then we will explore the methods and generative models used in deep learning.

## 1.2 Definitions

#### Definition1

Specialized Education: a complete way of learning something that means you fully understand it and will not forget it.
Specialized Computing: a type of artificial intelligence that uses algorithms ( = sets of mathematical instructions or rules) based on the way the human brain operates. [1]

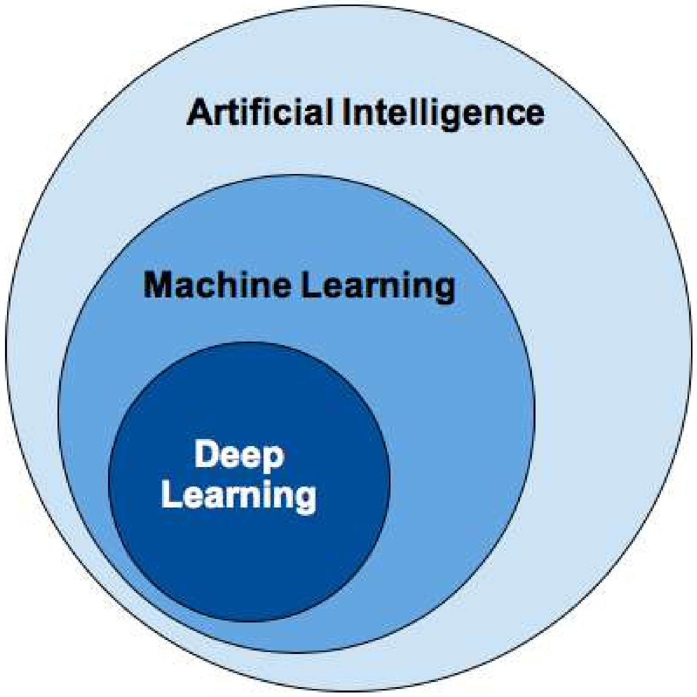


Figure 1.1: Artificial Learning, Machine Learning and Deep Learning

**Definition2**

A class of machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification.[2].

**Definition3**

Deep Learning is a new area of Machine Learning research, which has been introduced with the objective of moving Machine Learning closer to one of its original goals: Artificial Intelligence. Deep Learning is about learning multiple levels of representation and abstraction that help to make sense of data such as images, sound, and text. [3]

**Definition4**

”Deep learning is a class of machine learning algorithm that uses multiple stacked layers of processing units to learn high level representations from unstructured data.” [4]

**Definition5**

Deep learning is a subset of machine learning that use a learning algorithms on multiple levels of distributed representations to allow us extract the useful patterns from data automatically with less and less as possible of human intervention.

## 1.3 Why deep learning?

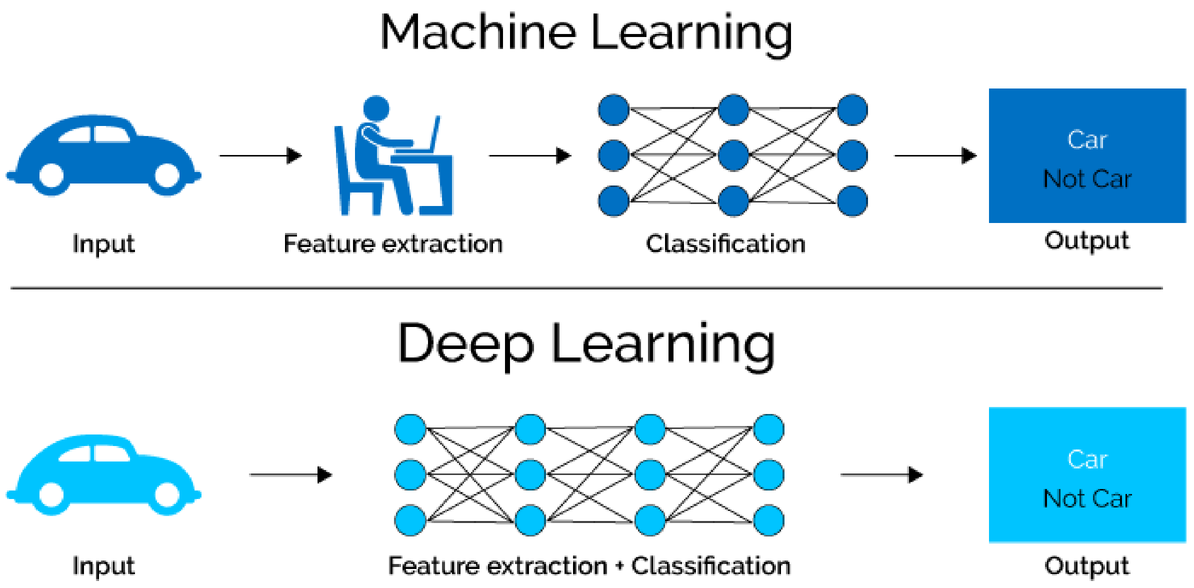


Figure 1.2: Difference between Machine learning and Deep learning

As is shown in figure2 deep learning automates much of the extraction of features from data and gets us closer without the new human involvement, as was done before with machine learning algorithms, so we should learn deep learning because it is a powerful technique to increase the automation of IA.[5] [6] [7]

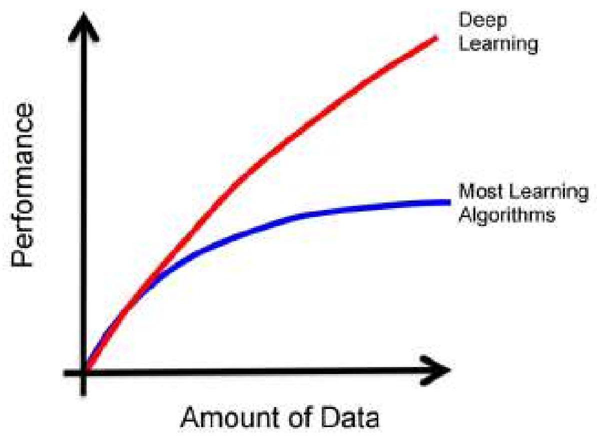


Figure 1.3: Amount of Data vs Performance DL ML

for machine learning algorithms you must identify all needed features in advance by an domain expert to make patterns more visible to learning algorithms to work, but the deep learning algorithms start to learn high-level features and extract patterns from data incrementally, this minimize the need of domain expertise for feature extraction every time. This is a big advantage for deep learning that make able to use a massive amounts of data " era of big data" and this will provide huge amounts of opportunities for new innovations.

## 1.4 When to use deep learning?

We should use deep learning when we trying to find complex patterns, and it comes to complex problems like image classification, audio recognition, and natural language processing, also, when simpler models (like logistic regression) don't achieve the needed accuracy level, in these cases the deep learning shines here.

When we have a high level of data dimension, or when there is a need to sequences, (in this case the time dimension will be present in our vectors).

When you do not have sufficient support from domain experts to understand features must be extracted, with deep learning we will not worry too much about the lack of support specialists.

When the data is large (big data), we just have to worry about the results is needed in a reasonable time, so we should to have infrastructure with a high level of performance.

## 1.5 History and Chronology of ideas and concepts used in deep learning

(Approximate dates for appearance and evolution).

- 1943: Neural networks by Warren McCulloch  Walter Pitts.
- 1957: Perceptron by Frank Rosenblatt.
- 1974 to 1986: appearance of backpropagation, RBM, and RNN.
- 1989 to 1998: appearance of CNN by LeCun, MNIST dataset, LSTM,Bidirectional,RNN.
- 2006: appearance of “Deep Learning”, DBN.
- 2009: appearance of ImageNet.
- 2012: appearance of AlexNet, Dropout.
- 2014: appearance of GANs,  DeepFace.
- 2016: appearance of AlphaGo.
- 2017: appearance of AlphaZero,  Capsule Networks.
- 2018: appearance of BER.

## 1.6 Neural Networks

### 1.6.1 Definition

A neural network is a type of machine learning algorithm inspired from the way of how a human brain learn, by creating a neural network from multilayer perceptron able to learn from inserted learning data.

### 1.6.2 Activation Function

Activation function is an important feature of neural networks. it's decide if a neuron should be activated or not, and if the information that the neuron is receiving related to the given information or should it be ignored.

$$Y = Activation(\sum((weight * input) + bias))) \tag{1.1}$$

The activation function is the non linear transformation applied on input. This transformed output is then sent to the next layer of neurons as input.

[8], [9], [10], [11]

### 1.6.3 Types of activation functions

#### Binary Step Function

Activation function would be a threshold based classifier i.e. whether or not the neuron should be activated. If the value Y is above a given threshold value then activate the neuron else leave it deactivated.

f(x) = 1 if x$\geq 0, f(x) = oifx < 0$.

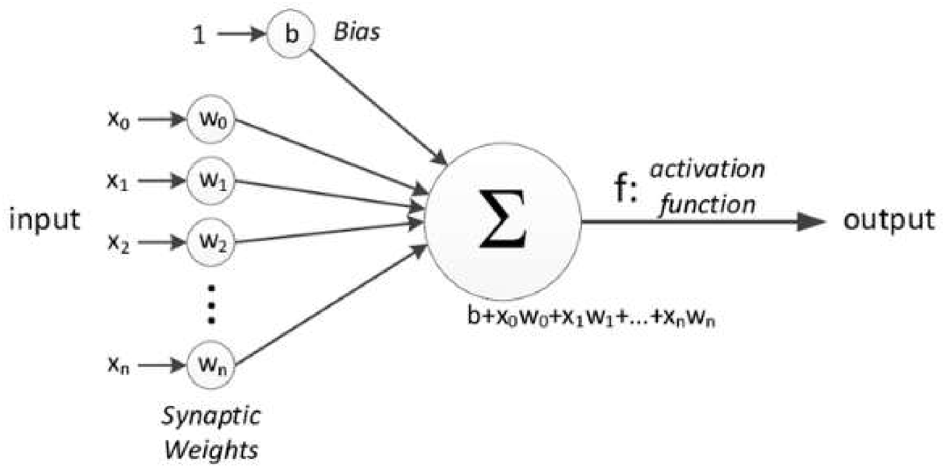


Figure 1.4: perceptron

**Linear Function**

In the step function, the gradient being zero, it was impossible to update gradient during the backpropagation. Instead of a simple step function, we can try using a linear function.

$$f(x) = ax \tag{1.2}$$

**Sigmoid Function**

Sigmoid is a widely used activation function. It is of the form

$$f(x) = \frac{1}{(1+e^{-x})} \tag{1.3}$$

**Tanh Function**

The tanh function is very similar to the sigmoid function. It is actually just a scaled version of the sigmoid function, tanh(x)=2 sigmoid(2x)-1, it can be directly written as

$$\tanh(x) = \frac{2}{(1+e^{-2x})-1} \tag{1.4}$$

**ReLU Function**

The ReLU function is the Rectified linear unit. It is the most widely used activation function. It is defined as: f(x)=max(0,x).it can be graphically represented as shown in figure 6. .

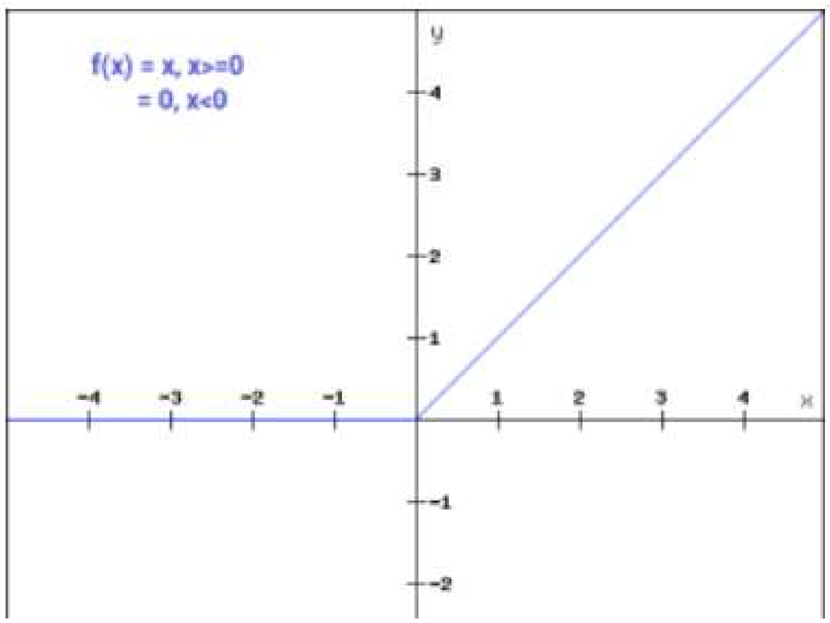


Figure 1.5: graphical representation of ReLU

### 1.6.4 Choosing the right Activation Function

However depending upon the properties of the problem, we might be able to make a better choice for easy and quicker convergence of the network.

- Sigmoid functions and their combinations generally work better in the case of classifiers.
- Sigmoids and tanh functions are sometimes avoided due to the vanishing gradient problem.

- ReLU function is a general activation function and is used in most cases these days.
- The ReLU function should only be used in the hidden layers.
- As a rule, is better to start with using ReLU function and then move over to other activation functions in case ReLU doesn't provide with optimum results.[8], [9], [11]

### 1.6.5 Back-Propagation:

Back-propagation is simply a method to compute the partial derivatives (or gradient) of a function, which has the form as a function composition (as in Neural Nets). When we solve an optimization problem using a gradient-based method (gradient descent is just one of them), we want to compute the function gradient at each iteration. [?]

## 1.7 Methods in deep learning

### 1.7.1 Classes of Deep Learning Networks:

The classification of methods used in deep learning leads us to a large aspects, techniques, and architectures of machine learning, depending how these architectures was designed to represent linear  non-linear information on multiple layers, and techniques and techniques are intended like classification, recognition,..etc. three major categories or classes can be obtained for methods used in deep learning nets:

#### A.Deep networks for unsupervised or generative learning:

which are intended to capture high-order correlation of the observed or visible data for pattern analysis or synthesis purposes when no information about target class labels is available. Unsupervised feature or representation learning in the literature refers to this category of the deep networks. When used in the generative mode, may also be intended to characterize joint statistical distributions of the visible data and their associated classes when available and being treated as part of the visible data. In the latter case, the use of Bayes rule can turn this type of generative networks into a discriminative one for learning, examples RBMs, DBNs, DBMs,RNNs, DBM. [2]

**B.Deep networks for supervised learning:**

which are intended to directly provide discriminative power for pattern classification purposes, often by characterizing the posterior distributions of classes conditioned on the visible data. Target label data are always available in direct or indirect forms for such supervised learning. They are also called discriminative deep networks, examples CNN, DNN. [2]

**C.Hybrid deep networks:**

where the goal is discrimination, which is assisted, often in a significant way, with the outcomes of generative or unsupervised deep networks. This can be accomplished by better optimization or/and regularization of the deep networks in category (B). The goal can also be accomplished when discriminative criteria for supervised learning are used to estimate the parameters in any of the deep generative or unsupervised deep networks in category (A) above. [2]

### 1.7.2 Convolutional Neural Networks CNNs

**Definition1:**

The convolutional neural networks or CNNs is a special kind of assembled and connected multilayer perceptrons used for patterns classification, also, we can say it's a special class of neural networks used in order to extraction (learn) higher level of features through the linear operation " convolution ".

**Definition2:**

Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. [12]

**Objective of CNNs**

The main objective of a CNNs is to highe level of extracted features from data via convolutions, and it's very well used for object recognition, identify faces, individuals, street signs, and many other aspects of visual data.

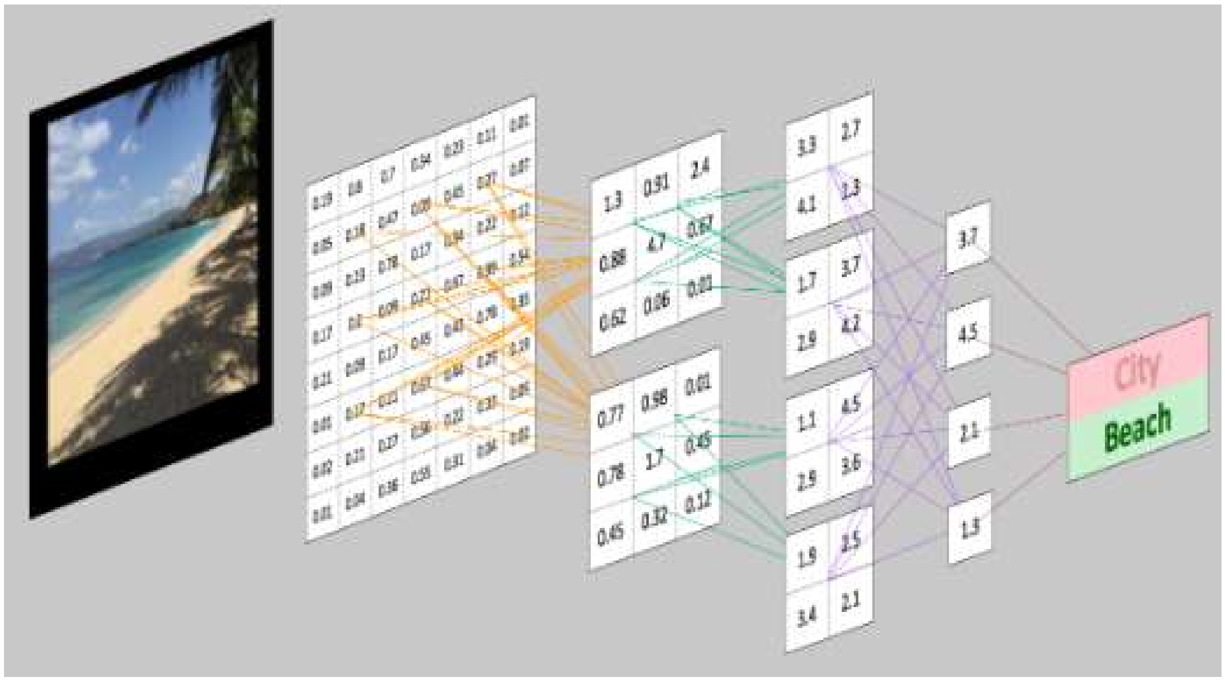


Figure 1.6: CNNs and computer vision

The efficacy of CNNs in image recognition is one of the main reasons why the world recognizes the power of deep learning. As Figure 1.6 illustrates, CNNs are good at building position and (somewhat) rotation invariant features from raw image data. CNNs are powering major advances in machine vision, which has obvious applications for self-driving cars, robotics, drones, and treatments for the visually impaired. [6]

Add to two-dimensional image, the CNNs are applied also to three-dimensional datasets. here are some examples :.

- MRI data: magnetic resonance imaging (MRI) datasets,
- 3D Shapes dataset,
- Graph data,
- NLP applications: Natural Language Processing.

.

**CNN Architecture:**

As figure 1.7 illustrates, the CNN architecture usually has three layers: a convolutional layer, pooling layer, and fully connected layer. **Convolution Layer:**

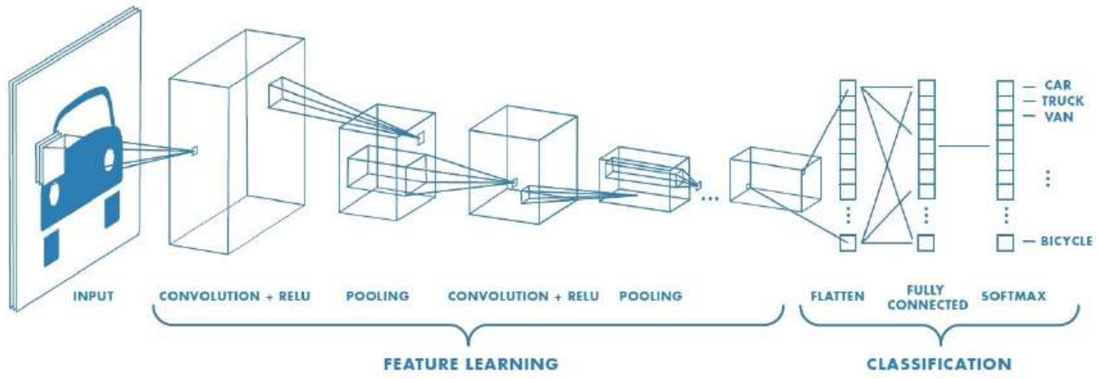


Figure 1.7: Architecture of a CNN

The convolution layer is the core building block of the CNN. It carries the main portion of the network's computational load.
This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image, but is more

in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels. During the forward pass, the kernel slides across the height and width of the image producing the image representation of that receptive region. This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride. If we have an input of size W x W x D and D out number of kernels with a spatial size of F with stride S and amount of padding P, then the size of output volume can be determined by the following formula:

$$Wout = (W - F + 2P)/S + 1 \tag{1.5}$$

This will yield an output volume of size W out * W out * D out.[7] .

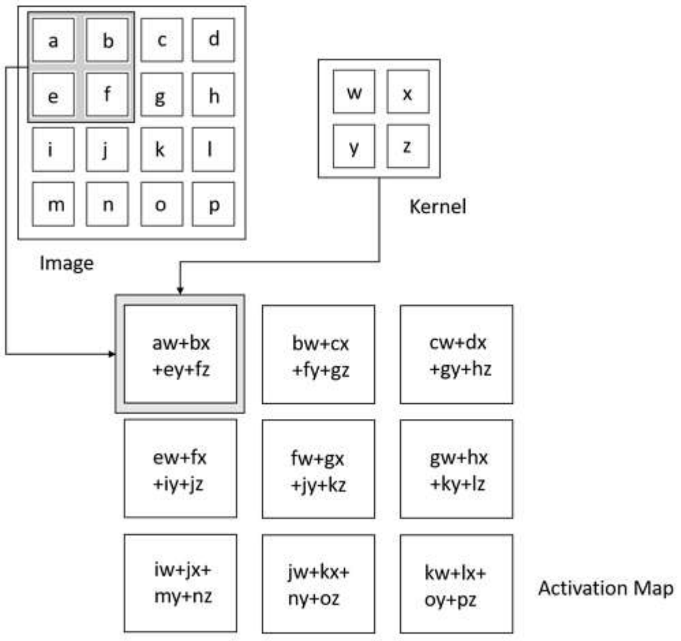


Figure 1.8: Convolution Operation (DL.Goodfellow, Y Bengio, and A Courville)

**Pooling Layer:**

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually.
There are several pooling functions such as the average of the rectangular neighborhood, L2 norm of the rectangular neighborhood, and a weighted average based on the distance from the central pixel. However, the most popular process is max pooling, which reports the maximum output from the neighborhood

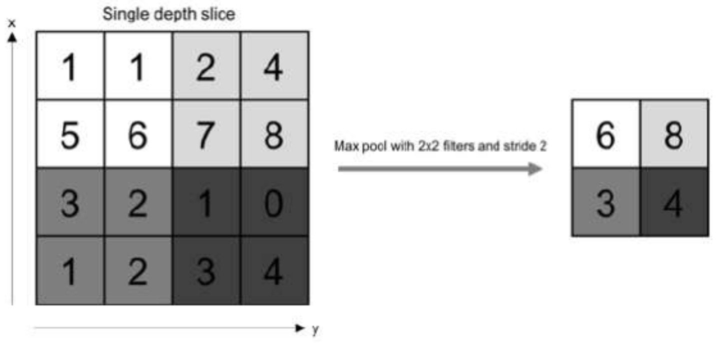


Figure 1.9: Pooling Operation (Source: O'Reilly Media)

If we have an activation map of size W x W x D, a pooling kernel of spatial size F, and stride S, then the size of output volume can be determined by the following formula:

$$Wout = W - F/S + 1 \tag{1.6}$$

This will yield an output volume of size Wout x Wout x D.

In all cases, pooling provides some translation invariance which means that an object would be recognizable regardless of where it appears on the frame.[7].

**Fully Connected Layer:**

Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect.

The FC layer helps map the representation between the input and the output.[7].

**Popular architectures of CNNs:**

- LeNet: one of the earliest successful architectures of CNNs developed by Yann Lecun, used to extract digits, read zip codes from images,...etc.
- AlexNet: ILSVRC 2012 winner The first work that popularized Convolutional networks in computer, developed by Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton.
- ZFNet: ILSVRC 2013 winner, Introduced the visualization concept of the Deconvolutional Network, Developed by Matthew Zeiler and Rob Fergus.
- GoogLeNet:ILSVRC 2014 winner, developed by Christian Szegedy and his team at Google, development of an Inception Module that dramatically reduced the number of parameters in the network.
- VGGNet: Runner-Up in ILSVRC 2014, developed by Karen Simonyan and Andrew Zisserman, Its main contribution was in showing that the depth of the network is a critical component for good performance.
- ResNet: residual network was the winner of ILSVRC 2015. It features special skip connections and a heavy use of batch normalization. [7] . [13]

.

## 1.8 Generative models

In this section, we will describe powerful and advanced models in deep learning that have become widely used in many subfields of machine learning, "Generative models" or "Deep generative models " are an active research area advancing rapidly to find more connections between the diverse classes of deep learning models.

In recent years, and with the remarkable progress in methodology and technology of deep learning, the scientists and specialists of this field has started to ask the question if we are now able to build machines that is in itself creative?.
The generative modelling aims to progress in this field to get answers for the posed question.

Based on unsupervised learning the generative models are able to learn structure and features from data then generate data that is similar to data the model has been trained with, using neural network combined with progress in stochastic optimization methods, this have enabled scalable modeling of complex, high-dimensional data including images, text, and speech.

### 1.8.1 What is Generative Models?

"Generative modeling is the art and science of engineering a family of probability distributions that is simultaneously rich, parsimonious, and tractable". [14]

"A model is generative if it places a joint distribution over all observed dimensions of the data". [14]

"A generative model describes how a dataset is generated, in terms of a probabilistic model. By sampling from this model, we are able to generate new data." [4]

Richard Feynman: “What I cannot create, I do not understand”.

Generative modeling: “What I understand, I can create”. [15]

### 1.8.2 Why Generative Models?

- Excellent test of our ability to use high-dimensional, complicated probability distributions.
- Realistic samples for artwork, super-resolution, colorization, etc.
- Generative models of time-series data can be used for simulation and planning (reinforcement learning applications).
- Extensive toolchains around optimization and automatic differentiation.
- Useful for engineering highly parameterized distributions.
- A way to build nonparametric and semiparametric statistical models.
- A model that allows us to learn a simulator of data.
- Models that allow for (conditional) density estimation.[14], [16]

## 1.9 Sample of Generative Models

In generative models there are two of the most commonly used and efficient approaches are Variational Autoencoders (VAE) and Generative Adversarial Networks (GAN). VAE aims at maximizing the lower bound of the data log-likelihood and GAN aims at achieving an equilibrium between Generator and Discriminator. In the next section we will try to explain the pricipe of VAE, after we will mention some other generative models,and the next chapter will be fully dedicated to explaining and detailing The GANs.

### 1.9.1 Variational Autoencoders:(VAE)

" The variational auto-encoder (VAE) has recently been introduced as a powerful method for unsupervised learning, from the neural net perspective, a VAE is a generative model consists of an encoder, a decoder, and a loss function.

Firstly we will describe the autoencoder, then we will explain the introduced differences with Variational Autoencoders.

#### Autoencoders

An autoencoder is a neural network made up of two parts:

**An encoder:** network that compresses high dimensional input data into a lower dimensional representation vector.

**A decoder:** network that decompresses a given representation vector back to the original domain.This process is shown in Figure 1.10.

The network is trained to find weights for the encoder and decoder that minimize the loss between the original input and the reconstruction of the input after it has passed through the encoder and decoder.
The representation vector is a compression of the original image into a lower dimensional, latent space. The idea is that by choosing any point in the latent space, we should be able to generate novel images by passing this point through the decoder, since the decoder has learned how to convert points in the latent space into viable images. [4]

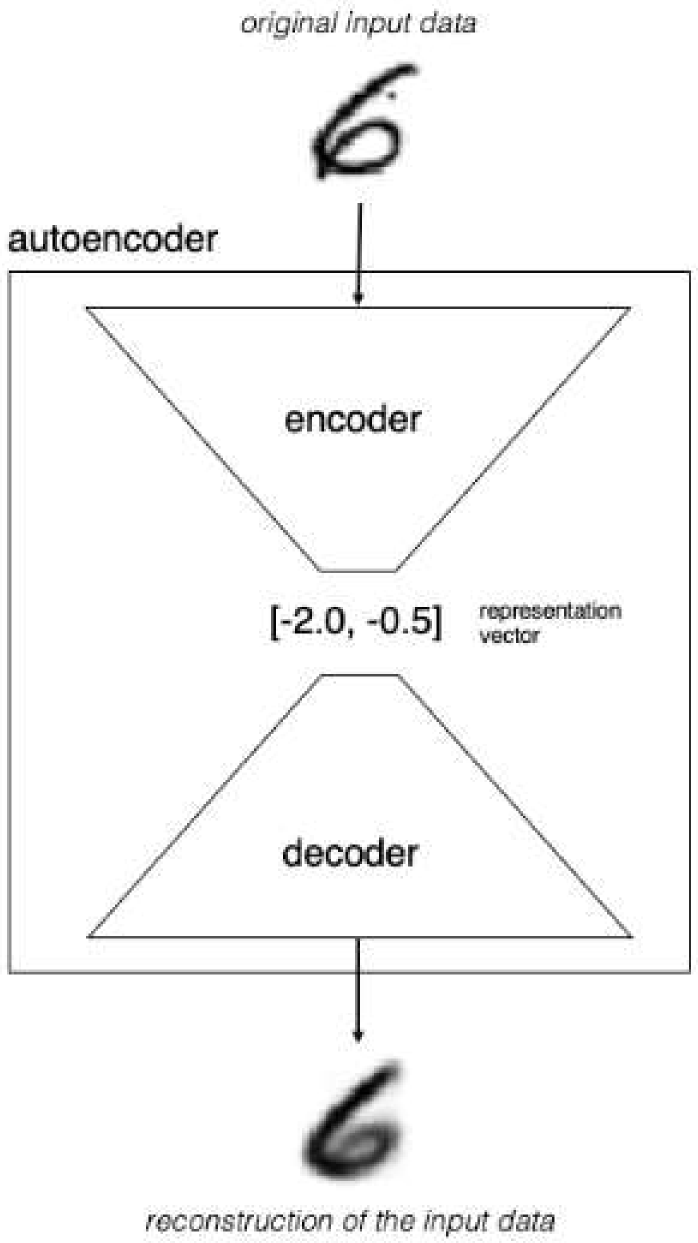


Figure 1.10: Diagram of an autoencoder.

### 1.9.2 Variational Autoencoders:(VAE)

Comparing the Variational Autoencoders with Autoencoders, there are two parts that should be changed - the encoder and the loss function.

**The Encoder:** In an autoencoder, each image is mapped directly to one point in the latent space. In a variational autoencoder, each image is instead mapped to a multivariate normal distribution around a point in the latent space, as shown in Figure 1.11

.

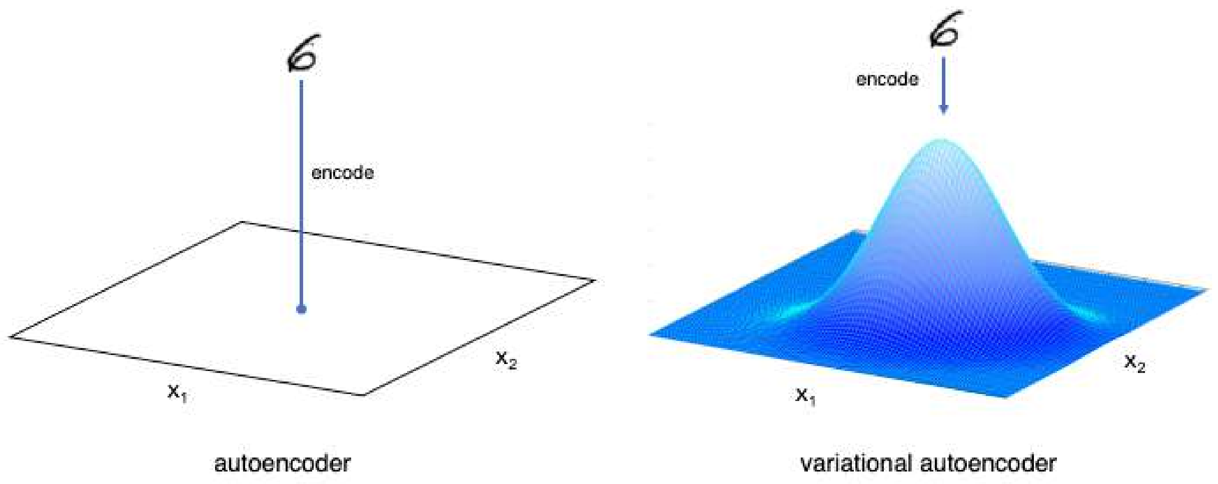


Figure 1.11: Encoders - the difference between an autoencoder and a variational autoencoder

To summarize, the encoder will take each input image and encode it to two vectors, that together define a multivariate normal distribution on the latent space:

- mu - the mean point of the distribution.
- log-var - the logarithm of the variance of each dimension.

to encode an image into a specific point z in the latent space, we can sample from this distribution, using the following equation:

$$\sigma = \exp(log - var/2) \tag{1.7}$$

$$z = \mu + \sigma * \epsilon \tag{1.8}$$

Where epsilon is a point sampled from the standard normal distribution. [4]

**Why this small change to the encoder help?**

Previously, we saw how there was no requirement for the latent space to be continuous. Now, since we are sampling at random from an area around mu, the decoder must ensure that all points in the same neighborhood produce very similar images when decoded, so that the reconstruction loss remains small. This is a very nice property that ensures that even when we choose a point in the latent space that has never been seen by the decoder, it is likely to decode to an image that is well-formed. [4], [17]

**The Loss Function**

Previously, our loss function only consisted of the RMSE loss between images and their reconstruction after being passed through the encoder and decoder. This reconstruction loss also appears in a variational autoencoder, but we also require one extra component - the KL divergence.KL divergence (Kullback–Leibler divergence) is a way of measuring how much one probability distribution is different from another. In a VAE, we want to measure how different our normal distribution with parameters mu and log-var is from the standard normal distribution.In this special case, the KL divergence has the closed form given below:

$$kl - loss = -0.5 * \sum (1 + \log - var - \mu^2 - \exp(log - var)) \tag{1.9}$$

The sum is taken over all the dimensions in the latent space. kl-loss is minimised to 0 when mu = 0 and log-var = 0 for all dimensions. As these two terms start to differ from 0, kl-loss increases.In summary, the KL divergence term penalises the network for encoding observations to mu and log-var variables that differ significantly from the parameters of a standard normal distribution, namely: mu = 0 and log-var = 0.

**Why does this addition to the loss function help?:**

Firstly, we now have a well-defined distribution that we can use for choosing points in the latent space - the standard normal distribution. If we sample from this distribution, we know that we're very likely to get a point that lies within the limits of what the VAE is used to seeing. Secondly, since this term tries to force all encoded distributions towards the standard normal, there is less chance that large gaps will form between point clusters. Instead, the encoder will try to use the space around the origin symmetrically and efficiently." Figure 1.12 shows the classification of generative models based on the type of density. [4], [17]

### 1.9.3 Other generative models:

- Generative Adversarial Network.
- Boltzmann Machines.
- PixelRNN and PixelCNN
- Generative stochastic networks or GSNs.

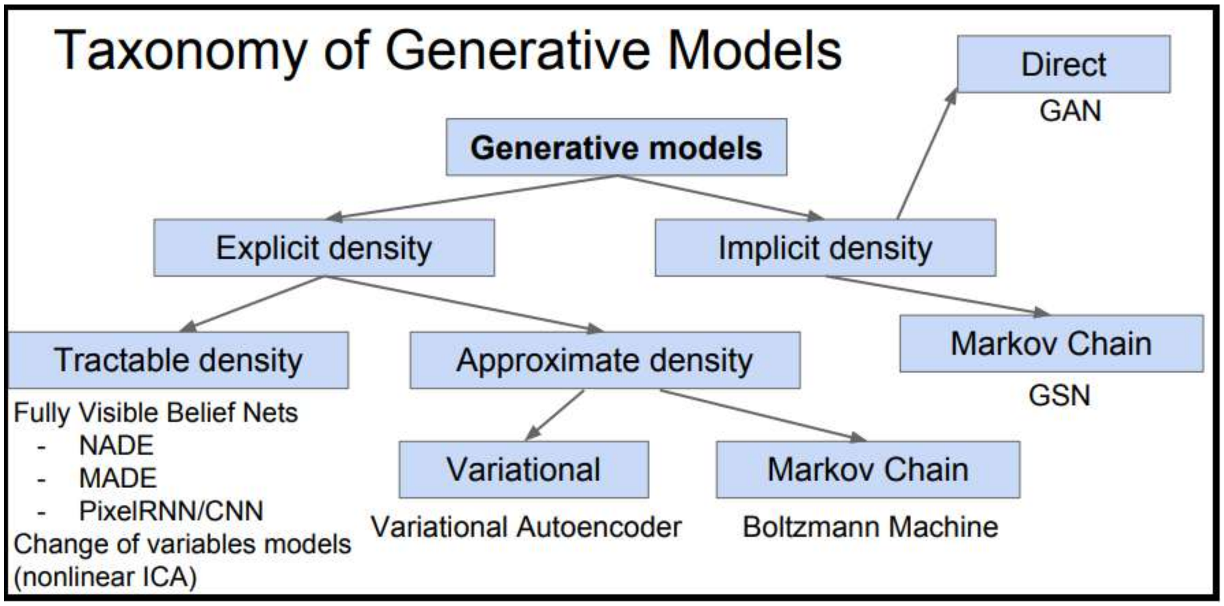


Figure 1.12: Taxonomy of Generative Models.

## 1.10 Conclusion

In this chapter, we introduced the concepts of deep learning, why and when to use deep learning, as we had explained the difference between deep learning and machine learning, where the machine learning is a method of statistical learning where each instance in a dataset is described by a set of features or attributes. but deep learning is a method of statistical learning that extracts automatically the features or attributes from raw data. also, we explained the different classes of deep learning which led us to the most important concept currently "generative models" where we presented one of the most important types of this model "Variational Autoencoders", in the next chapter, we will introduce and explain the most commonly used and efficient model "generative adversarial Networks" , this will give us the necessary foundations to perform the implementation effectively at the last port of this study.

# Chapter 2

# Generative Adversarial Network (GANs)

## 2.1 Introduction:

John Romero says, “You might not think that programmers are artists, but programming is an extremely creative profession. It’s logic-based creativity”. Recently the deep learning has given a new boost to this creativity through generative models, especially through Generative Adversarial Networks, or (GANs), this deep learning powerful tool allowed Christie’s (auctions house) to sells its first AI portrait for \$ 432,500, beating estimates of \$ 10,000 ( a portrait was generated by a GANs shows the figure of what looks like an 18th century gentleman).

Also, taking into account that, The 2018 Turing award has been given to a trio of researchers who laid the foundations for the current boom in artificial intelligence: Yoshua Bengio, Geoffrey Hinton, and Yann LeCun, ( this prize known as the “Nobel Prize of computer sciences,”), This is a significant reward for the deep learning and all fields of artificial intelligence because the artificial intelligence had almost no consideration like this from the IT community before.

Yann LeCun describe GANs as: “the most interesting idea in the last 10 years in Machine Learning”.

All of the above-mentioned means that the impact and potential of deep generative models has become very huge for innovations in many domains.

Generative adversarial networks (introduced by Ian Goodfellow in 2014) define a research line in generative models field that which showed very impressive results in computer vision.

In this chapter, we will present the theoretical understanding of GANs, in order to explore the possibility of generate Arabic letters using DCGANs (Deep Convolutional GANs) which is the main objective of this project.

**What Are Generative Adversarial Networks?**

A Generative Adversarial Nets (GANs) are kind of neural network, where the architecture based on generative modeling . (described in the 2014 paper by Ian Goodfellow).

More generally, Generative modeling involves using a model to generate new examples that plausibly come from an existing distribution of samples, such as generating a new similar images but specifically different from a dataset of existing images.

The GAN model architecture involves two sub-models: a generator model for generating new examples and a discriminator model for classifying whether generated examples are real, from the domain, or fake, generated by the generator model.

- Generator. Model that is used to generate new plausible examples from the problem domain.

- Discriminator. Model that is used to classify examples as real (from the domain) or fake (generated).

Generative adversarial networks are based on a game theoretic scenario in which the generator network must compete against an adversary. The generator network directly produces samples. Its adversary, the discriminator network, attempts to distinguish between samples drawn from the training data and samples drawn from the generator.[12] [17] [4]

## 2.2 Architecture of GANs

The basic architecture of GAN described by Ian Goodfellow 2014. A GAN consists of two neural networks playing a game with each other. The discriminator tries to determine whether information is real or fake. The other neural network, called a generator, tries to create data that the discriminator thinks is real.

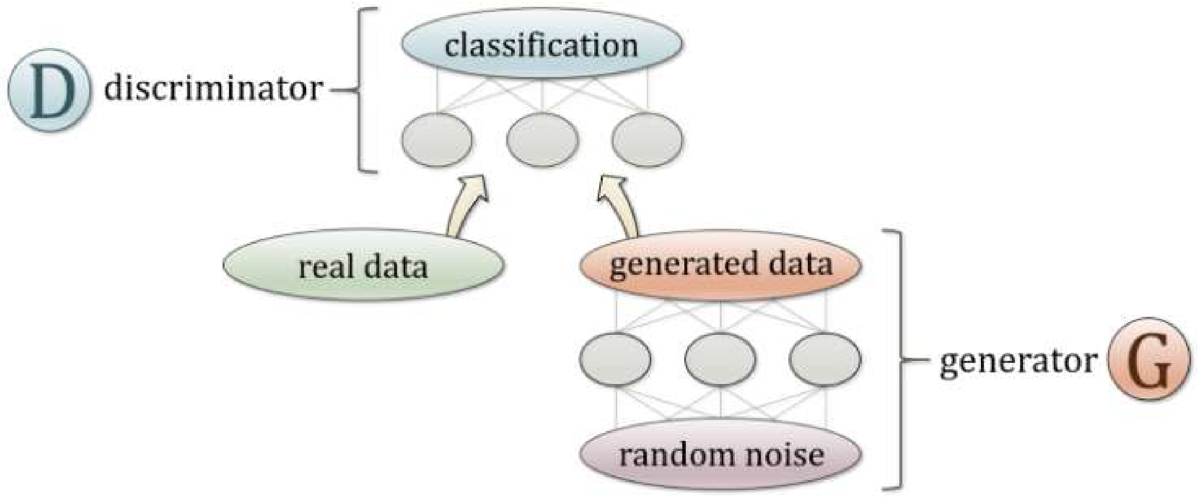


Figure 2.1: A basic GAN architecture

This concept is depicted in Figure 2.1 The neural network at the top is the discriminator, and its task is to distinguish the training set's real information from the generator's creations. In the simplest GAN structure, the generator starts with random data and learns to transform this noise into information that matches the distribution of the real data.

The generator never sees the genuine data; it must learn to create realistic information by receiving feedback from the discriminator. This is called adversarial loss, and when implemented correctly it works surprisingly well. In fact, regularization techniques such as dropout layers are often used in GANs because the generator can overfit the training set through this entirely indirect learning process.

The longer these two neural networks play this game, the more they sharpen each other's skills. The discriminator becomes very good at detecting fake data while the generator learns to produce information that is indistinguishable from what is observed in the real world.

When we end up with two GAN neural networks that are very good at what they do, how could we use them? A trained discriminator can be used for detecting abnormalities, outliers, and anything out of the ordinary. This could be very valuable in fields such as cybersecurity, radiology, astronomy, and manufacturing.

A skilled generator is used for making creations. Once the generator learns the distribution of the training data, we can sample the generator an unlimited number of times for realistic outputs such as images, language, pharmaceuticals, numerical simulations, and just about anything else one can imagine. [18], [4], [17]

### 2.2.1 Generative vs. Discriminative Algorithms

To understand GANs, we should know how generative algorithms work, and for that, contrasting them with discriminative algorithms is instructive. Discriminative algorithms try to classify input data; that is, given the features of an instance of data, they predict a label or category to which that data belongs.

For example, given all the words in an email (the data instance), a discriminative algorithm could predict whether the message is spam or not-spam. spam is one of the labels, and the bag of words gathered from the email are the features that constitute the input data. When this problem is expressed mathematically, the label is called y and the features are called x. The formulation p(y—x) is used to mean "the probability of y given x", which in this case would translate to "the probability that an email is spam given the words it contains."

So discriminative algorithms map features to labels. They are concerned solely with that correlation. One way to think about generative algorithms is that they do the opposite. Instead of predicting a label given certain features, they attempt to predict features given a certain label.

The question a generative algorithm tries to answer is: Assuming this email is spam, how likely are these features? While discriminative models care about the relation between y and x, generative models care about "how you get x." They allow you to capture p(x—y), the probability of x given y, or the probability of features given a label or category. (That said, generative algorithms can also be used as classifiers. It just so happens that they can do more than categorize input data.) [19]

Another way to think about it is to distinguish discriminative from generative like this:

- Discriminative models learn the boundary between classes.

- Generative models model the distribution of individual classes.

### 2.2.2 The Maths Behind Generative Adversarial Networks

. Kullback–Leibler and Jensen–Shannon Divergence, are two metrics for quantifying the similarity between two probability distributions.

**(1) KL (Kullback–Leibler) divergence** measures how one probability distribution p diverges from a second expected probability distribution q.

$$D_{KL}(p\|q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx \tag{2.1}$$

.

DKL achieves the minimum zero when p(x) == q(x) everywhere.

It is noticeable according to the formula that KL divergence is asymmetric. In cases where p(x) is close to zero, but q(x) is significantly non-zero, the q's effect is disregarded. It could cause buggy results when we just want to measure the similarity between two equally important distributions.

**(2) Jensen–Shannon Divergence** is another measure of similarity between two probability distributions, bounded by [0,1]. JS divergence is symmetric and more smooth.

$$D_{JS}(p\|q) = \frac{1}{2} D_{KL}(p\|\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q\|\frac{p+q}{2}) \tag{2.2}$$

Some believe that one reason behind GANs' big success is switching the loss function from asymmetric KL divergence in traditional maximum-likelihood approach to symmetric JS divergence.

**GAN consists of two models:**

- A discriminator D estimates the probability of a given sample coming from the real dataset. It works as a critic and is optimized to tell the fake samples from the real ones.
- A generator G outputs synthetic samples given a noise variable input z (z brings in potential output diversity). It is trained to capture the real data distribution so that its generative samples can be as real as possible, or in other words, can trick the discriminator to offer a high probability.

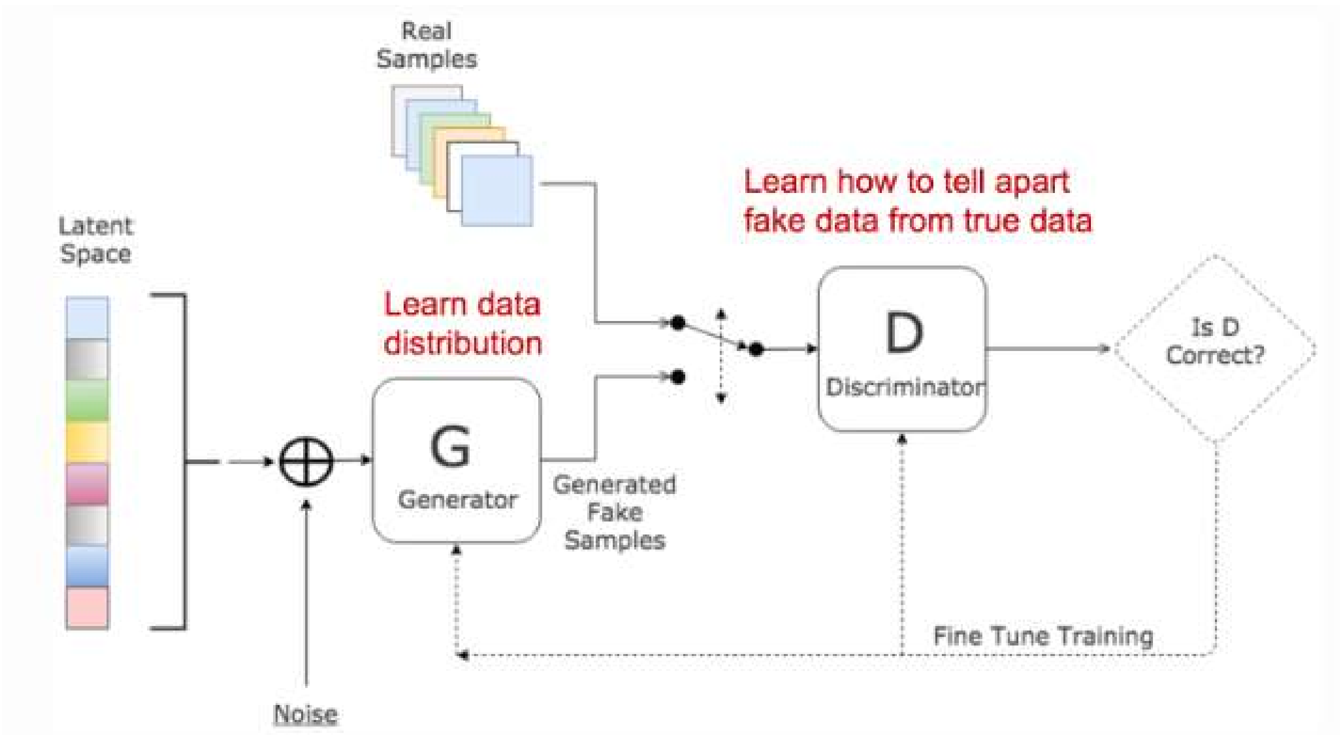


Figure 2.2: Architecture of a generative adversarial network.

.

These two models compete against each other during the training process: the generator G is trying hard to trick the discriminator, while the critic model D is trying hard not to be cheated. This interesting zero-sum game between two models motivates both to improve their functionalities.

Given,

- Pz: Data distribution over noise input z.
- Pg: The generator's distribution over data x.
- Pr: Data distribution over real sample x.

On one hand, we want to make sure the discriminator D's decisions over real data are accurate by maximizing $\mathrm{E}_{x \sim p_r(x)}[\log D(x)]$ Meanwhile, given a fake sample. G(z), z $\sim p_z(z)$, the discriminator is expected to output a probability, D(G(z)), close to zero by maximizing $\mathrm{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$.

On the other hand, the generator is trained to increase the chances of D producing a high probability for a fake example, thus to minimize
$\mathrm{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$

When combining both aspects together, D and G are playing a minimax game in which we should optimize the following loss function:

$$\begin{aligned}\min_G \max_D L(D, G) &= E_{x \sim p_r(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \\ &= E_{x \sim p_r(x)}[\log D(x)] + E_{x \sim p_g(x)}[\log(1 - D(x)]\end{aligned} \tag{2.3}$$

($\mathrm{E}_{x \sim p_r(x)}[\log D(x)]$ has no impact on G during gradient descent updates.)

**What is the optimal value for D?**

Now we have a well-defined loss function. Let's first examine what is the best value for D.

$$L(G, D) = \int_x \bigg( p_r(x) \log(D(x)) + p_g(x) \log(1 - D(x)) \bigg) dx \tag{2.4}$$

Since we are interested in what is the best value of D(x) to maximize L(G,D), let us label: $\widetilde{x} = D(x), A = p_r(x), B = p_g(x)$.

And then what is inside the integral (we can safely ignore the integral because x is sampled over all the possible values) is:

$$\begin{aligned} f(\widetilde{x}) &= Alog\widetilde{x} + Blog(1 - \widetilde{x}) \\ \frac{df(\widetilde{x})}{d\widetilde{x}} &= A\frac{1}{ln10}\frac{1}{\widetilde{x}} - B\frac{1}{ln10}\frac{1}{1 - \widetilde{x}} \\ &= \frac{1}{ln10}(\frac{A}{\widetilde{x}} - \frac{B}{1 - \widetilde{x}}) \\ &= \frac{1}{ln10}\frac{A - (A + B)\widetilde{x}}{\widetilde{x}(1 - \widetilde{x})} \end{aligned} \tag{2.5}$$

Thus, set, df($\widetilde{x}$)$/d\widetilde{x} = 0$,e get the best value of the discriminator:

D$^*(x) = \widetilde{x}* = \frac{A}{A+B} = \frac{p_r(x)}{p_r(x)+p_g(x)} \in [0, 1]$.

Once the generator is trained to its optimal, p$_g$ gets very close to p$_r$

When p$_g = p_r, D^*(x)$ become 1/2.

**What is the global optimal?**

When both G and D are at their optimal values, we have p$_g = p_r$ and D$^*(x) = 1/2$ and the loss function becomes:

$$\begin{aligned} L(G, D^*) &= \int_x \bigg( p_r(x) \log(D^*(x)) + p_g(x) \log(1 - D^*(x)) \bigg) dx \\ &= \log \frac{1}{2} \int_x p_r(x) dx + \log \frac{1}{2} \int_x p_g(x) dx \\ &= -2 \log 2 \end{aligned} \tag{2.6}$$

**What does the loss function represent?**

According to the formula listed in Sec. 2, JS divergence between p$_r$ and p$_g$ can be computed as:

$$\begin{aligned} D_{JS}(p_r \| p_g) =& \frac{1}{2} D_{KL}(p_r || \frac{p_r + p_g}{2}) + \frac{1}{2} D_{KL}(p_g || \frac{p_r + p_g}{2}) \\ =& \frac{1}{2} \bigg( \log 2 + \int_x p_r(x) \log \frac{p_r(x)}{p_r + p_g(x)} dx \bigg) + \\ & \frac{1}{2} \bigg( \log 2 + \int_x p_g(x) \log \frac{p_g(x)}{p_r + p_g(x)} dx \bigg) \\ =& \frac{1}{2} \bigg( \log 4 + L(G, D^*) \bigg) \end{aligned} \tag{2.7}$$

Thus,

$$L(G, D^*) = 2 D_{JS}(p_r \| p_g) - 2 \log 2 \tag{2.8}$$

Essentially the loss function of GAN quantifies the similarity between the generative data distribution p$_g$ and the real sample distribution p$_r$ by JS divergence when the discriminator is optimal. The best G that replicates the real data distribution leads to the minimum L(G$^*$, $D^*$) $= -2\log 2$ which is aligned with equations above. [20]

### 2.2.3 Extensions of GANs

GANs are now a very active research area actually, and there are many different types of GAN implementation, below we mention some important ones and widely used in researches:

- Vanilla GAN: the simplest type of GAN. where the Generator and the Discriminator are simple multi-layer perceptrons.(SGD used for optimization).
- Conditional GAN (CGAN): CGAN some conditional parameters added to help Generator for generating data, and help Discriminator distinguish the real data from the fake ones.
- Deep Convolutional GAN: DCGAN is the most popular and successful implementation of GAN( we will explain it in detail in the next section).
- Context-Conditional GAN,
- Wasserstein GAN,
- and there are many other types.

## 2.3 Deep Convolutional GANs

Current deep-learning research still focused on extracting of features from huge datasets that can be reused again, for that reason in computer vision, using unlabeled images to extract good features will be very helpful in many kinds of supervised learning like image classification, for this objective, there is a solution based on the training of GANs, the generator, and discriminator networks will be used as feature extractors for supervised

tasks, but the training of GANs is unstable, and the generator sometime give a nonsensical results.
So, in order to make the GANs more stable at training, a new extension of the GAN has been introduced called Deep Convolutional Generative Adversarial Networks (DCGANs) by Alec Radford  his team.

DCGANs it's mainly composed of convolution layers without max pooling or fully connected layers. It uses convolutional stride and transposed convolution for the downsampling and the upsampling, for that the DCGANs became the most popular and successful in GANs family, that way today most GANs are at least loosely based on the DCGAN architecture

### 2.3.1 Architecture of DCGANs

What has changed by coming of DCGANs ?

Below the architecture guidelines that make the Deep Convolutional GANs stable:

- Use transposed convolution for upsampling.
- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

The figure 2.3 below show the network design for Generator. [21]

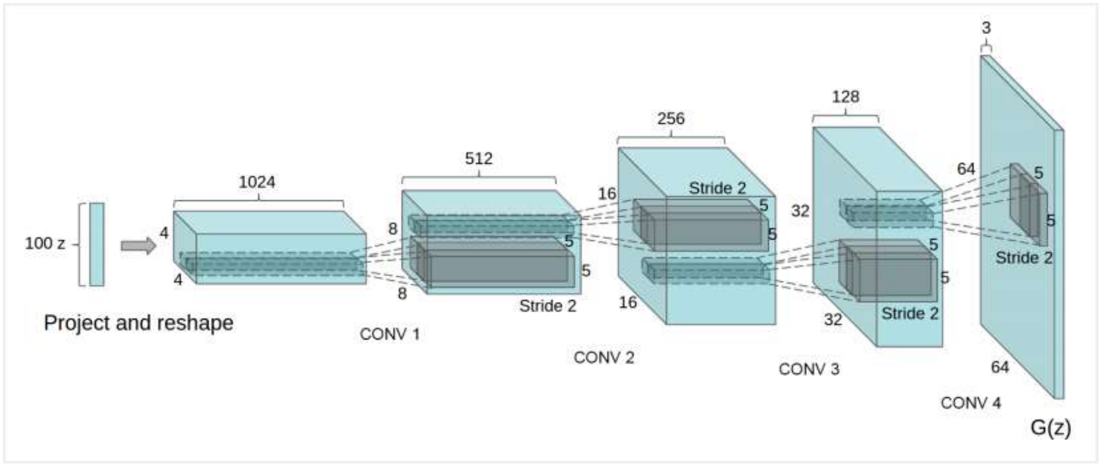


Figure 2.3: Architecture of DCGANs Generator.

**Recommended parameters for DCGANs:**

- Training mini-batch size is 128.
- All weights should be initialized from a zero-centered Normal distribution with standard deviation 0.02.
- For LeakyReLU, the slope of the leak was set to 0.2 in all models.
- Use the Adam optimizer with tuned hyperparameters. with learning rate is 0.0002, and the hyperparam beta1 at 0.5.

The figure 2.4 below show the network design for Discriminator. [21]

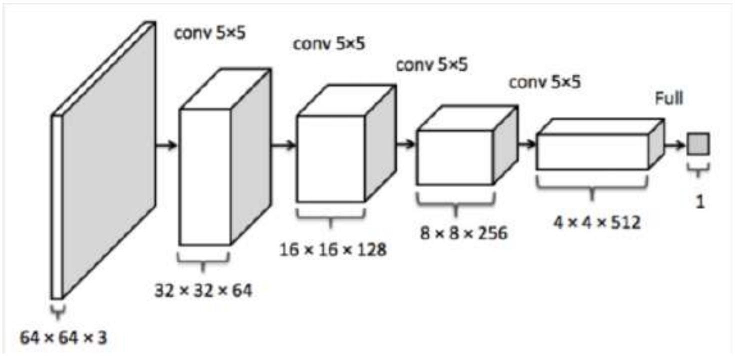


Figure 2.4: Architecture of DCGANs Discriminator.

## 2.4 Conclusion

In this chapter, we have explained GANs  DCGANs, with their architectures and parameters, where we have concluded that the increasing of complexity of the generator does not necessarily improve the image quality, Also the simplicity of DCGAN makes it stable and able to be good start point for other project, in the next chapter “implementation” we will try to build DcGANs that will generate Arabic letterers, this allows us to test what we have already studied in this chapter.

# Chapter 3

# Implementation

## 3.1 Introduction

Arabic calligraphy (Islamic calligraphy) is the artistic practice of handwriting and calligraphy in various languages and lands that use Arabic letters and share a common Islamic cultural heritage, It includes Arabic Calligraphy, Ottoman, and Persian calligraphy, and,it is characterized by written connected, that makes it possible to acquire different geometric shapes through the tide, reverb, rotation, elevation, and overlap.
Also It is known in Arabic as khatt Islami ( ), meaning Islamic line, design, or construction, It is also used in the desalination of manuscripts and books, especially copies of the Holy Quran.
Popular Arabic calligraphy: 1- Naskh 2 - Diwani 3- Thuluth 4- Reqa 5- Koufi.

In this chapter, we aim to implement DCGANs to generate Arabic letters, based on the training of discriminator on Arabic handwritten dataset, to explore the horizons may be opened by this wonderful technology that has achieved impressive successes in images generation, so that allows maybe in future studies to generate new Arabic calligraphy through GANs.

## 3.2 Environment of Implementation

To implement any project, it is very important to provide the necessary environment and tools, equipment, software, and licenses. Students and scientists in the field of deep learning are very lucky, because there are many free open source tools, like programming languages, libraries, and free platforms that provide you all that you need. Also, there are many free web clouds services, that make at our disposal high-performance computers and servers(such a google Colaboratory).

The development environment that we have worked on consists of the following tools:

- Python Language
- Anaconda Distribution
- PyTorch library
- Google Colaboratory

### 3.2.1 Python Language

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.[22]

- Used Version: 3.7.3 64-bit,
- open source: you are allowed to view and modify the source

### 3.2.2 Anaconda Distribution

Anaconda is the most trusted distribution for data science, also it's a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system conda.

- Anaconda Version: 3.
- Spyder 3: Is a powerful scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It features a unique combination of the advanced editing, analysis, debugging and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection and beautiful visualization capabilities of a scientific package.n Furthermore, Spyder offers built-in integration with many popular scientific packages, including NumPy, SciPy, Pandas, IPython, QtConsole, Matplotlib, SymPy,tonsorflow, and pytorch.
- Jupyter Notebook 5: is a web application, a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.[23]

### 3.2.3 Google Colaboratory

Deep learning developers often clash with the limitations of their devices, especially students, where their programs need high-performance computers with GPU or TPU to be implemented, and this is not available in most cases, cause of the high prices of such types of equipment, but all these it does not matter very much since the launch of google-colab service.

Goole Colaboratory is a research project created to help disseminate machine learning education and research. it's a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. with Colab you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from browser.

**Advantages provided by Google Colab**

- We can connect colab to our local machine through the local Jupyter Notebook and work directly on google colab GPUs TPUs.
- Comes with pre-installed frameworks and libraries for deep learning, with a possibility to customize the environment by adding a new libraries.
- From colab we can map google drive to save , import , and share our projects and data.
- Provide us NVIDIA Tesla K 80 as GPU runtime environment with 12GB of RAM and 360 GB disk space.
- Provide us TPUs.
- Pre-requirement to benefit these advantages is only must have a Gmail account.

### 3.2.4 Pytorch

PyTorch is a python based library built to provide flexibility as a deep learning development platform. The workflow of PyTorch is as close as you can get to python's scientific computing library – numpy. it is free and open-source, PyTorch provides many high-level features:

- Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU).
- Deep neural networks built on a tape-based autodiff system.
- Easy to use API, simple as python can be.
- Python support – As mentioned above, PyTorch smoothly integrates with the python data science stack. It is so similar to numpy that you might not even notice the difference.
- Dynamic computation graphs – Instead of predefined graphs with specific functionalities, PyTorch provides a framework for us to build computational graphs as we go, and even change them during runtime. This is valuable for situations where we don't know how much memory is going to be required for creating a neural network. [24]

## 3.3 Program Explanation

The practical part of our project is a program written in Python, where we will train our DCGAN to generate new Arabic letters after training it on many handwritten letters pictures (dataset), all this by using Pytorch library was previously explained. the explanation of the implementation will start with the description of the used dataset, then we will define the needed input and parameters for the running of the program, in the last part, we will see what the program produces as results.

### 3.3.1 Dataset

The dataset is composed of pictures for Arabic alphabet (from 'alef' to 'yeh' 480 images per character) handwritten with images size 32x32 (for dcgan the standard image size is supposed to be 64x64), because it's the only dataset available for handwritten Arabic letters we found ourselves forced to introduce some changes of standard architecture of generator and discriminator of the DCGAN to adapt it for size 32x32.

The arabic-chars-mnist.zip dataset is available on "https://www.kaggle.com/rashwan/arabic-chars-mnist." after downloaded, we extracted the images into Google drive folder called train, then, we mapped this folder to google colaboratory. Now the dataset is ready for use from Google Colab.

### 3.3.2 Input

to run the program we should define some inputs:

- GPUs-number: on colab the value will be 1 for GPU or 0 to use CPU, to run the program on local computer without GPU like Nvidia graphic card we should use 0 as value.
- num-epochs: Number of training epochs(e will try many values).
- img-size: The size of training images, for our program we use size 32.
- btch-size : Batch size during training, we will use 128 or 64.
- data-path = It's the root directory for dataset: (../content/drive/My Drive/dataset/train2) the dataset mapped folder from google drive.
- nc: if it's a color training images we will use 3, else , 1 for black and wight images.
- lr: Learning rate for optimizers
- Beta1: hyperparam for Adam optimizers we will use 0.5 (standard of DCGANs)

### 3.3.3 Output

**First Output:**

we will use the subdirectories images in dataset folder, to create the data loader in order to run it on, then visualize some training images.

**Second output:**

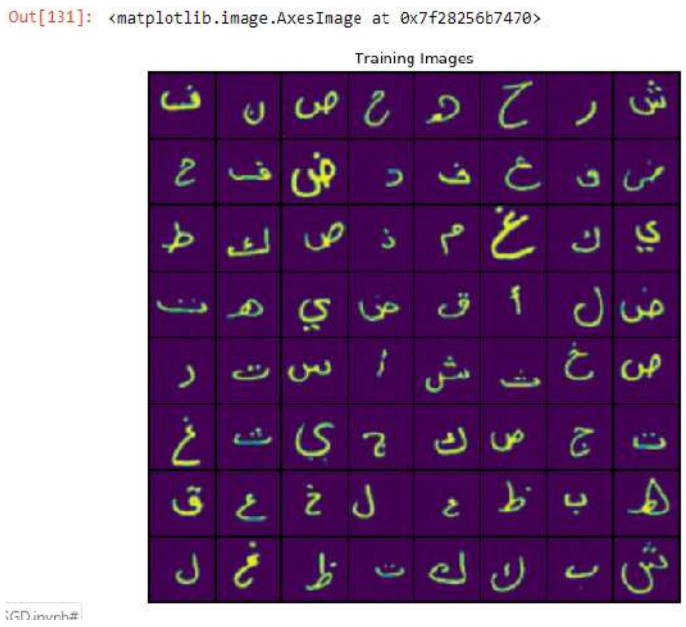
we will get the printed model of generator and discriminator to see how are structured, then we will get the training Loop : depending on the number of the chosen number of epochs this step might take a while.

**Third output:**

- show the changed loss during training, by plot of D  G's losses versus training iterations.
- we will visualize output of noise batch from G for every epoch.
- we will show a batch of real data next to a batch of fake data from G.

## 3.4 Results:

Here we visualize some training images



```
Out[131]: <matplotlib.image.AxesImage at 0x7f28256b7470>
```

Figure 3.1: visualize Training Images

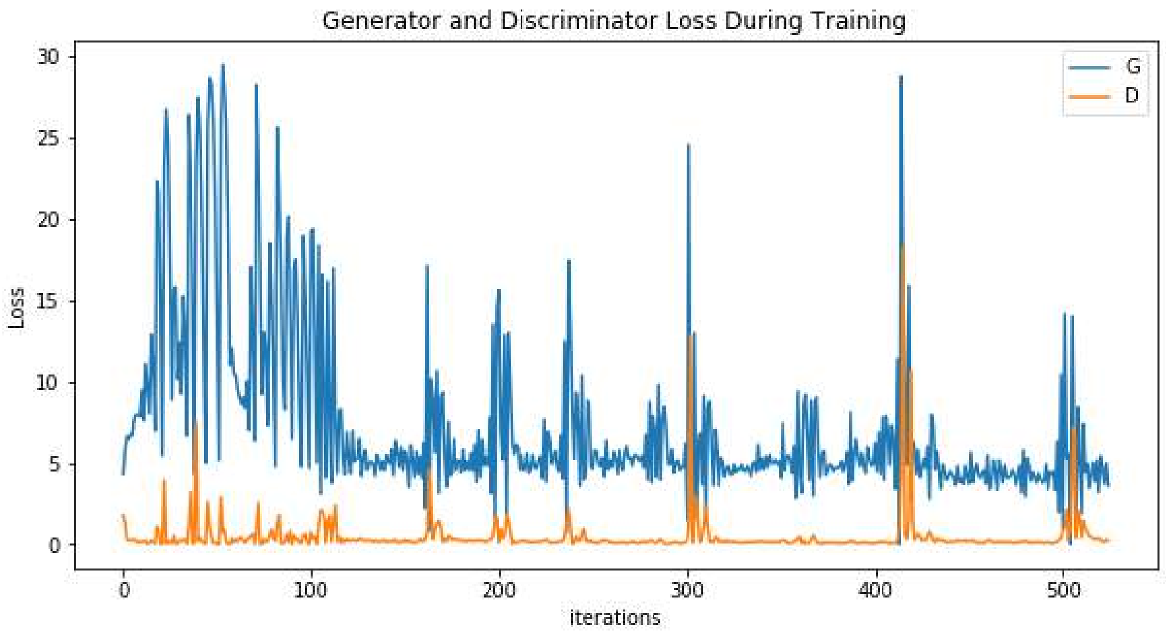


Figure 3.2: plot of D Vs G losses Optim: Adam, size: 64, epochs:5

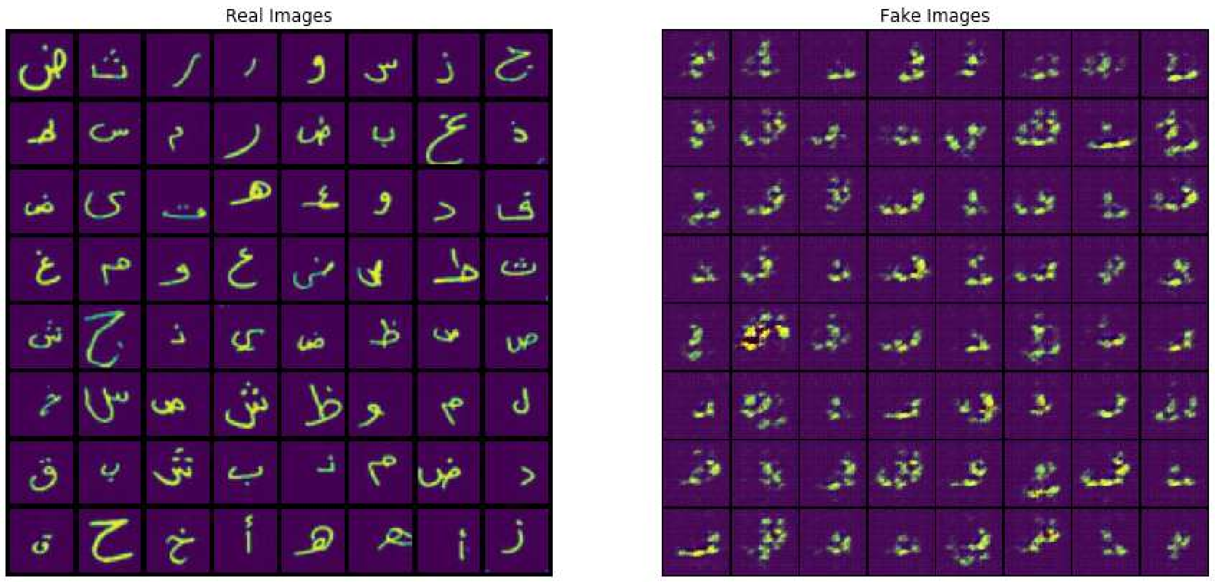


Figure 3.3: Real vs Fake losses Optim: Adam, size: 64, epochs:5

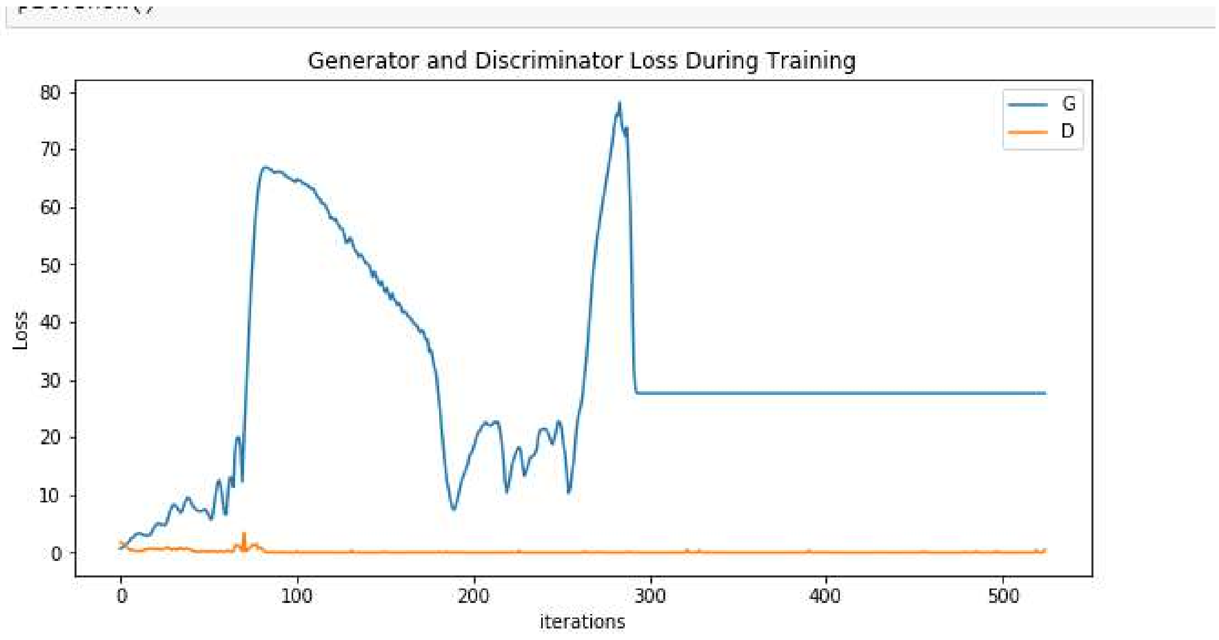


Figure 3.4: plot of D Vs G losses Optim: SGD, size: 64, epochs:5

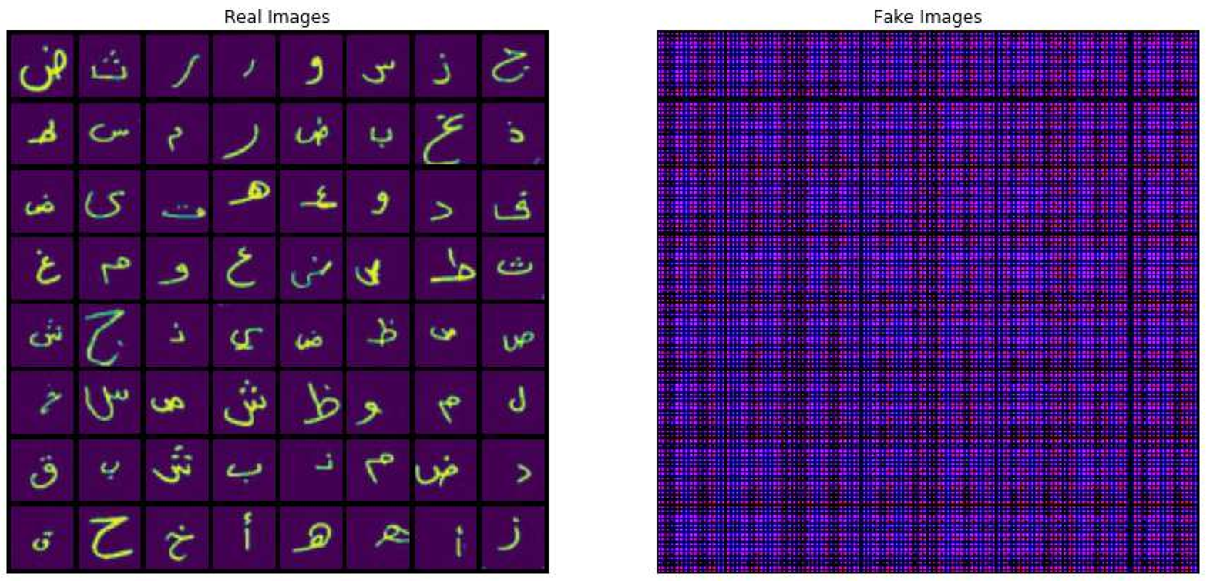


Figure 3.5: Real vs Fake losses Optim: SGD, size: 64, epochs:5

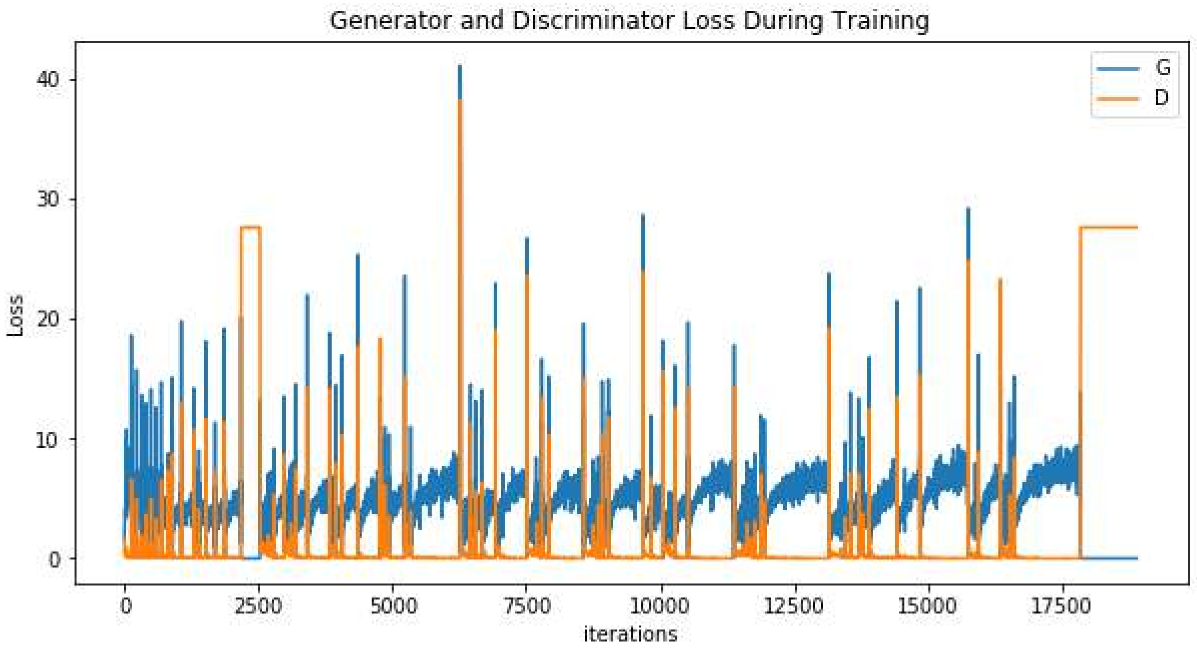


Figure 3.6: plot of D Vs G losses Optim: Adam, size: 32, epochs:180

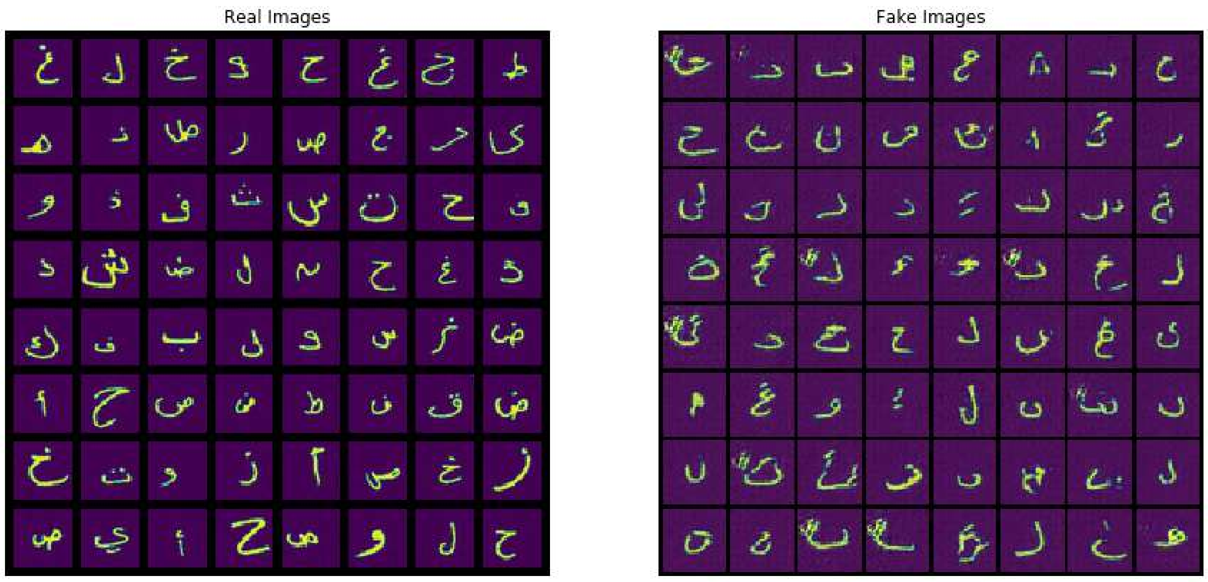


Figure 3.7: Real vs Fake losses Optim: Adam, size: 32, epochs:180

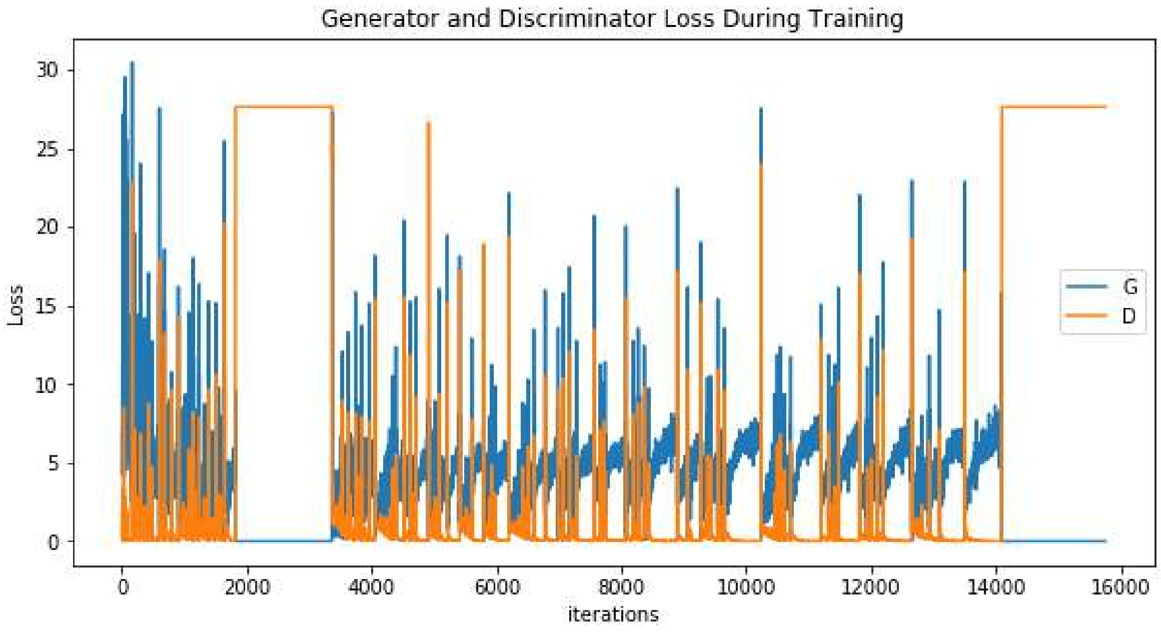


Figure 3.8: plot of D Vs G losses Optim: Adam, size: 64, epochs:150

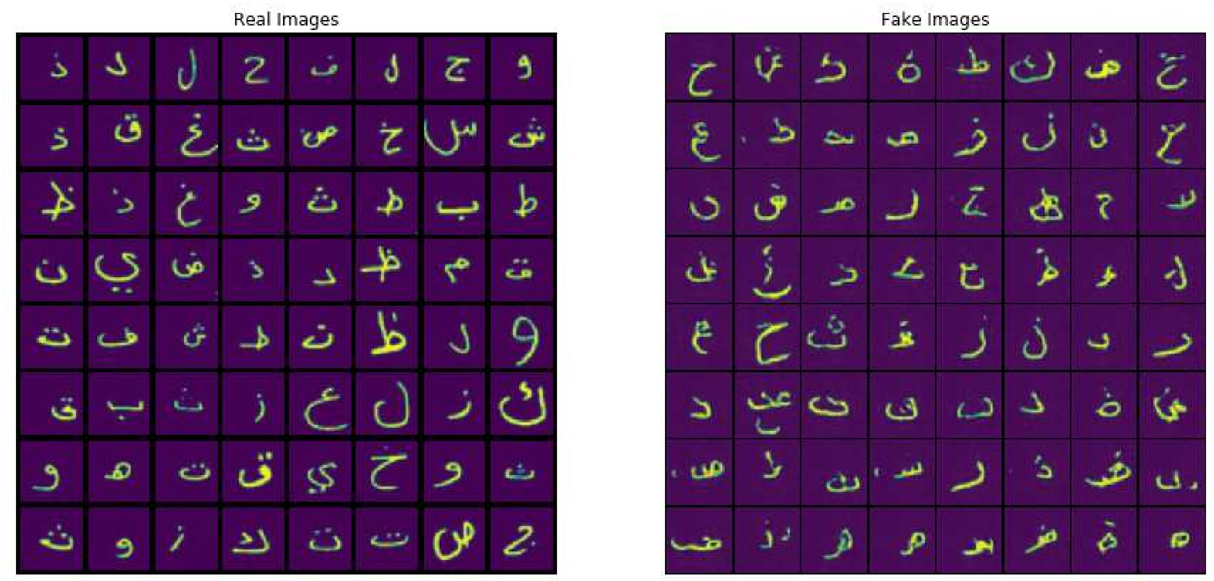


Figure 3.9: Real vs Fake losses Optim: Adam, size: 64, epochs:150

## 3.5 Discussion and Conclusion

The implementation section has allowed us to expand and discover many techniques and platforms that make the implementation of deep learning projects more smooth, easy, and allow us to get results in a resonable time.
Google Colabotory make GPUs  TPUs allowed for anyone interested to implement deep learning projects, also the choice of right frameworks and libraries (like PyTorch) make the programming of GANs in particular and neural networks in general very easy after the real understanding of the topic. The achieved results in terms of success to generate Arabic letters can be customized as follows:

- If we compare figures (3.2  3.3) with (3.4  3.5) as it can be seen the Adam optimizer achieves substantially better results than the SGD one. Therefore, the use of SGD Optimizer has been abandoned in the standard of this model (DCGAN) unlike the Adam optimizer.

- Now if we compare the figures (3.2  3.3) with (3.6  3.7) and (3.8  3.9) where we used the same optimizer (Adam) but with different training epochs: it shows The evolution of achieved results when we apply more training epochs, the difference seems very exciting.

- The last results comparison will be between (3.6  3.7) and (3.8  3.9) where it seems there are no significant differences between the two results, this is what makes us assume there may be limits to learning from small representations size (this is just an assumption that may be it will be verifiable in further studies)

# General Conclusion

In this work, we tried to build a perception by the projection of GANs success in computer vision to generating Arabic letters using a specific GAN extension called DcGAN, (which has a structural difference). The results were impressive although the problem of limited size of dataset, Where we are forced to modify the standard model of DCGAN to adapt it with the size of training images.
From the architectures tested during this work, conclusions have been compiled and described after the implementation some are not conclusive conclusions, but it based on the result of the experimentation.

After we completed this work we can say that: the Generative Adversarial Networks adjusted with gradient based optimization methods is powerful tools technique has proved its ability to absorb many representations (in our case: arabic handwritten letters), this makes the generating of a new Arabic-calligraphy using DcGAN seems like a very possible horizon in the near future.

# Bibliography

[1] https://dictionary.cambridge.org/dictionary/english/deep-learning.

[2] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[3] https://github.com/lisa-lab/deeplearningtutorials.

[4] David Foster. https://learning.oreilly.com/library/view/generative-deep-learning/9781492041931/.

[5] https://medium.com/the-deep-learning-methods.

[6] Josh Patterson and Adam Gibson. *Deep learning: A practitioner's approach.* " O'Reilly Media, Inc.", 2017.

[7] https://www.datascience.com/blog/convolutional-neural-network.

[8] https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning.

[9] Christopher M Bishop. *Pattern recognition and machine learning.* springer, 2006.

[10] Simon S Haykin, Simon S Haykin, Simon S Haykin, Kanada Elektroingenieur, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson education Upper Saddle River, 2009.

[11] Gilbert strang https://http://math.mit.edu/ gs/learningfromdata/.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[13] http://cs231n.github.io/convolutional-networks/.

[14] Ryan P. Adams: Princeton University. Deep probabilistic generative models, https://lips.cs.princeton.edu.

[15] Stefano Ermon and Aditya Grover. https://deepgenerativemodels.github.io.

[16] Fei-Fei Li Justin Johnson Serena Yeung. Generative models, http://cs231n.stanford.edu.

[17] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[18] https://towardsdatascience.com/adversarial-training-creating-realistic-fakes-with-machine-learning-c570881d0e81.

[19] https://skymind.ai/wiki/generative-adversarial-network-gan.

[20] Lilian Weng. From GAN to WGAN, http://arxiv.org/abs/1904.08994. *CoRR*, abs/1904.08994, 2019.

[21] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[22] https://www.python.org.

[23] https://docs.anaconda.com.

[24] https://pytorch.org/docs/stable/index.html.