UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

AI SCHOOL

An Overview of Generative Adversarial Networks (GAN)

Student
Cristian Cosci

Second Edition 2022

Contents

Chapter 1

Introduction

Generative Adversarial Netorks were originally introduced by Goodfellow et al. [3] in 2014. He introduced GAN as a new framework for estimating generative models via an adversarial process, in which a generative model G captures the data distribution, while a discriminative model D estimates if the sample came from the training data rather than G.
In other word, GAN are a particular neural network that take random noise as input and generate picture as outputs. This picture appear to be a sample from the distribution of the training set (e.g. MNIST or other image based dataset).
A GAN structure consist in two different (and with different scope) models which are trained simultaneously and in an adversarial way:

- A **Generative** model (also called **Generator**) that captures the distribution of training set.
- A **Discriminative** model (also called **Discriminator**) that estimates the probability that a sample is a real or a fake picture (e.g. is from training data or is generated by the other model).

The simplest explanation about GANs training is to think at Generator as a banknotes' counterfeiter and at Discriminator as a policeman specialized in banknote recognition (see Figure 1.1).
The policeman are taught how to determine if a dollar bill is either genuine or fake. Samples of real dollar bills from the bank and fake money from the counterfeiter are used to train the policeman. However, from time to time, the counterfeiter will

attempt to pretend that he printed real dollar bills. Initially, the police will not be fooled and will tell the counterfeiter why the money is fake. Taking into consideration this feedback, the counterfeiter hones his skills again and attempts to produce new fake dollar bills. As expected the policeman will be able to both spot the money as fake and justify why the dollar bills are fake [5].

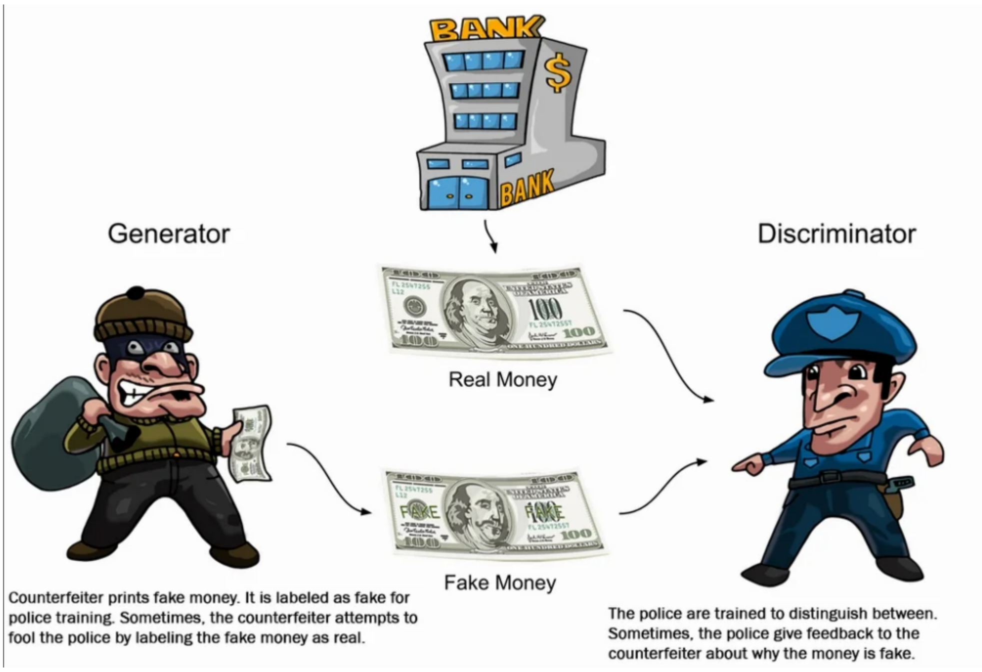

Figure 1.1: Generator vs Discriminator examples [5].

In other words the Generator is trying to learn the distribution of real data (by taking random noise as input, and producing realistic-looking images) and is the network which we're usually interested in. On the other hand, the Discriminator tries to classify whether the sample has come from the real dataset, or is fake (generated by the generator).

During the game the goal of the generator is to trick the discriminator into "thinking" that the data it generates is real. The goal of the discriminator, on the other hand, is to correctly discriminate between the generated (fake) images and real images coming from some dataset (e.g. MNIST).

Therefore, in the training phase the Discriminator model give a feedback to the Generator model in order to tell him if his work is good or bad. In this manner the

Generator correct and improve his generation phase. However, at the same time, also the Discriminator improve his classification while taking as input also real sample from the dataset and receiving a supervisioned feedback on his work.

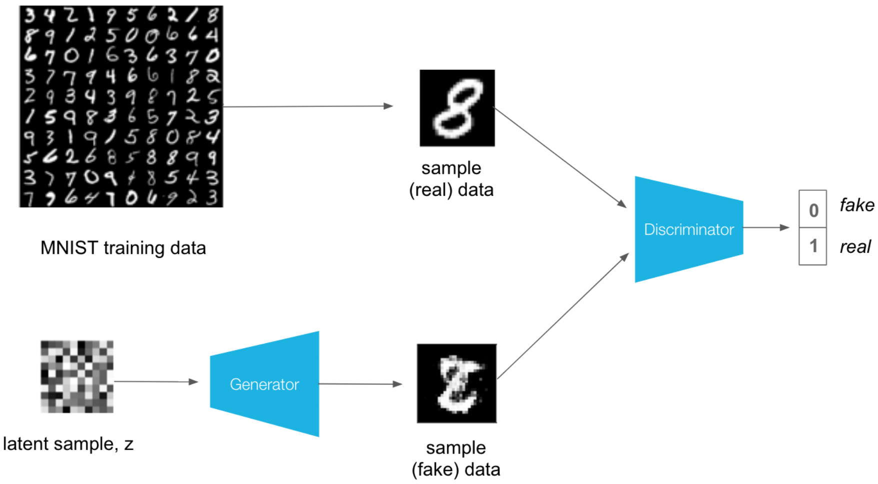

Figure 1.2: Simple GAN structure [1].

1.1 GAN: Generator and Discriminator

In this section i provide a little bit more deep explanation about the principal components of GAN. I will describe the structure of Generator and Discriminator, the training phase and the objective function that guides it.

1.1.1 Generator

Generator is a Neural Network, which given a dataset X_real tries to capture his distribution, by producing images X_fake from noise Z as input. Noise input is a random set of values, usually sampled from a multivariate-gaussian distribution. This is often called **latent vector** and that vector space is called **latent space**. The GAN's Generator acts quite like the decoder component of VAE: it project latent space to an image. The difference is that the Generator's latent space is not forced to learn exactly a Gaussian distribution but it can learn and model more complex distribution. A problem that can bring GAN to poor performance is **Mode collapse**, which happens when the Generator can only produce a single type of output or a small set of outputs. This may happen due to problems in training, such as the generator finds a type of data that is easily able to fool the Discriminator and thus keeps generating that one type.

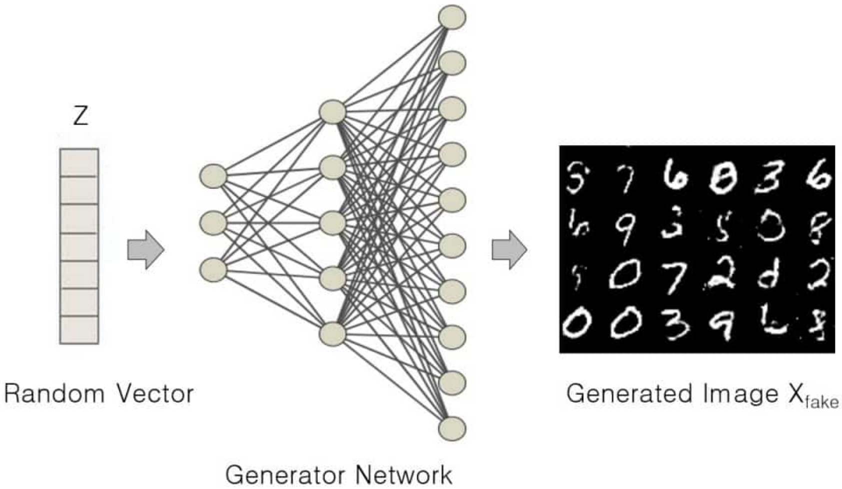

Figure 1.3: GAN's Generator example [7].

1.1.2 Discriminator

The Discriminator is more simple to understand than the Generator. The Discriminator is like a model trained to classify if an input image is real or is generated by another model. In other words the Discriminator is a binary classifier trained with a supervisioned classification mode.

The Discriminator task is to predict a label (e.g. *True* or *Fake* or 0-1 label) from an input. While doing so, the supervisioned train gives to it a feedback and this permit the model to learn a set of parameters (weights and bias) in order to map the correct label.

The principal scope of the Discriminator is to give a feedback to the Generator model, and to improve its generation phase. Therefore the Discriminator is used to train the Generator and after the training process, the Discriminator model is discarded as we are interested in the generator.

Sometimes, the generator can be repurposed as it has learned to effectively extract features from examples in the problem domain. Some or all of the feature extraction layers can be used in transfer learning applications using the same or similar input data.

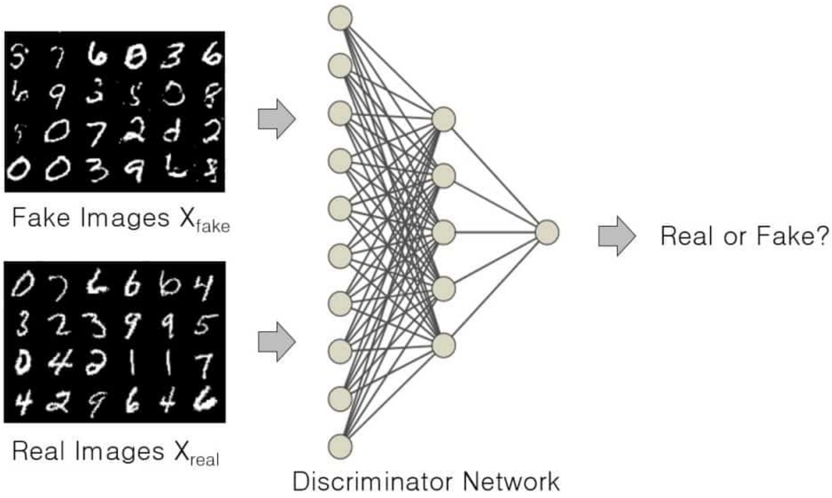

Figure 1.4: GAN's Discriminator example [7].

1.2 Training Phase

In this section i will describe the principal aspects of training phase on Generative Adversarial Networks.
But first a brief recap: the two models, the Generator and Discriminator, are trained together. The generator generates some samples, and these, along with real examples from the training dataset, are provided to the discriminator which classify these as real or fake. The Discriminator receive a feedback (is a supervisioned task) and is then updated to get better performance at discriminating real and fake samples in the next round. Also the Generator is updated based on how well, or not, the generated samples fooled the Discriminator.
Before start the explanation, let's denote some of the key elements:

- $X \rightarrow$ Training samples
- $Z \rightarrow$ Noise or Latent Vector
- $D \rightarrow$ Discriminator

- $G \rightarrow$ Generator
- $X_{real} \rightarrow$ a sample from training dataset
- X_{fake} or $G(z) \rightarrow$ Generator's output
- $D(x) \in (0, 1) \rightarrow$ Discriminator's output

The training of the two models (D and G) is done in an alternating way:

1. **Step 1**:
   - The Generator take Z as input and produce $G(x)$ (X_{fake}).
   - X_{fake} and a sample X_{real} are passed to the Discriminator.
   - The Discriminator gives $D(x)$ as output. This tells the probability score of each input passed to D to be real or fake.
   - As a normal supervised training, the predictions are compared with the ground truth, and a **Loss** is calculate (e.g. Binary Cross-Entropy).
   - The Gradient is the backpropagated only through the Discriminator and is parameters are updated.
2. **Step 2**:
   - The Generator produce an image $G(x)$, that is again passed through the Discriminator.
   - The Discriminator gives a prediction $D(x)$ as before, but only for image passed by the Generator.
   - The Loss is computed as before.
   - In this step, because we want to enforce the Generator to produce images, as similar to the real images as possible (i.e., close to the true distribution), the Loss is backpropagated only through the Generator network, and its parameters are optimized suitably.

It is clear that the Discriminator is very important for the Generator's training. The Discriminator permits to compute a Loss and that is backpropagate through the generator to improve and update its weights and bias.

However there is a need for both networks to be strong enough because if the Discriminator is a weak classifier, also low quality images produced by Generator will be accepted as real images and then there will be no improvement for the Generator's images production. In other hand, if the Generator is weak, it will not be able to fool the Discriminator.
The best case scenario is when the Discriminator is no more able to distinguish if an image produced by the Generator is real or fake, so its predictions are like a toss of a coin i.e. it choose randomly (50% is fake and 50% is real). At the same time, obviusly, the Generator has to be strong enough to produce very high quality images that his consistent with data distribution.

As described, both the Generator and the Discriminator's training is based on the classification score given as output by the last layer of D model. The output score is based on a binary classification problem, so the Binary Cross-Entropy function is used to compute the Loss:

$$L(\hat{y}, y) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log \hat{y_i} + (1 - y_i) \log(1 - \hat{y_i})]$$

We can go a little bit more deep on that Loss function:

- The negative sign at the beginning of the equation is there because the output of a neural network is normalized between 0 and 1, compute the *log* of a number in that range result in a value less than zero.
- The $\frac{1}{N}$ term identify the Loss' means computed across the batch (N refer to batch size).
- $\hat{y_i}$ is the Discriminator's prediction.
- y_i is the true label.
- The equation's first term is valid when the true label is 1 (real), and the second is valid when the true label is 0 (fake).

As said before, we can describe GAN's training phase as a **minmax** game played by Generator and Discriminator. Is possible to explain it with the following value function $V(G, D)$ [3]:

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

However, the previously value function may not provide sufficient gradient for the generator to learn the data distribution so well [3]. Training this way will achieve only half the objective. Though the discriminator definitely becomes more powerful for it can now easily discriminate the real from the fake, the generator lags behind. It has still not learned to produce realistic-looking images. The consequence is that quickly in the learning, when the Generator's performance is poor, the Discriminator can distinguish real and fake samples with high confidence. In this case, $-\log(1 - D(G(z)))$ saturates. Hence, rather than training G to minimize $-\log(1 - D(G(z)))$, better performances occurs training G to maximize $-log D(G(z))$ [3].

To get more detail, we can start from the Discriminator's **objective function**. The Discriminator is a binary classifier, that given an input x, returns a probability between 0 and 1 called $D(x)$. For istance the label for X_{real} is 1 and the label for X_{fake} is 0. A value for $D(x)$ closer to 1 means that the discriminator predicts that the input is a real image. On the other hand, a value for $D(x)$ closer to 0 means that the input is fake.
Thus, the objective function of the Discriminator becomes:

- **Maximizing** $D(X_{real})$ and **Minimizing** $D(X_{fake})$, where $X_{fake} = G(Z)$.

The objective function is modelled by Binary Cross-Entropy loss function, and we get the final formula substituting terms in the loss equation [7]:

- y = 1 and $\hat{y} = D(X_{real})$

$$D_{loss_{real}} = -\log D(X_{real}) \tag{1.1}$$

- true label y is 0, and the predicted output $\hat{y}$ is $D(X_{fake})$ where X_{fake} is equal to $G(z)$. Putting these values in the BCE loss function, we get:

$$D_{loss_{fake}} = -\log(1 - D(G(z)) \quad (1.2)$$

- Therefore, the cumulative Discriminator loss becomes:

$$D_{loss} = D_{loss_{real}} + D_{loss_{fake}} \quad (1.3)$$

$$D_{loss} = -\log D(X_{real}) - \log(1 - D(G(z)) \quad (1.4)$$

The Generator's **objective function** has the scope to Maximizing $D(G(Z))$ i.e. bringing it closer to 1 and fool the Discriminator. For this objective, i.e., to maximize the probability $D(G(z))$ by the Discriminator, the true label y is 1, and the predicted output $\hat{y}$ is $D(G(z))$. Putting these values in the BCE loss function, we get [7]:

$$G_{loss} = -\log D(G(z)) \quad (1.5)$$

These are the most important thing to know about GANs, their components and their training phase. In the next Chapters i will describe the various type of GANs which were implemented after the Goodfellow et al.'s paper and their implementations.

Chapter 2

Some GAN's variants

After Goodfellow's article on GAN [3], many researchers have become interested in the Generative Adversarial Networks. Many variants of the original GAN architecture have been proposed and studied.
In the next section i will propose some of the most important variations like DCGAN, cGAN and cDCGAN.

2.1 Deep Convolutional GAN (DCGAN)

Deep Convolutional Generative Adversarial Network, also know as DCGAN is a particular and new GAN's architecture that improve his quality using **convolutional layers**. It was introduced around 2016 by Alec Radford and other researcher in a paper published at ICLR conference [6].
The main components introduced in DCGAN is:

- **fractionally-strided convolutional layers** in the Generator, with the scope of upsample the images (see Figure 2.1).
- **strided convolutional layers** in the Discriminator, with the scope of downsample the images.

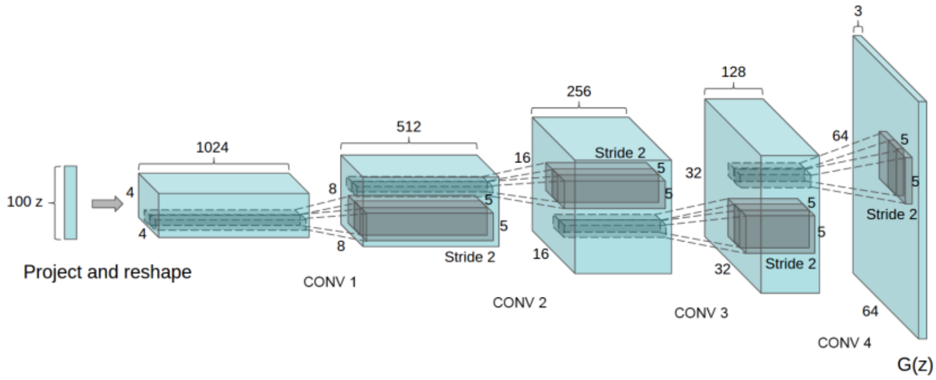

Figure 2.1: DCGAN generator used in the Radford's introduction paper for scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64 × 64 pixel image. Notably, no fully connected or pooling layers are used [6].

In the Figure 2.2 we can see an example of a strided convolutional layer. In DCGAN the authors used a Stride of 2, meaning the filter slides thorough the image moving 2 pixels per step. The use of strided convolutional layer replaced the use of pooling operation like **maxPooling or avgPooling** because these operation have not learnable parameter. The reason why they used a strided convolutional layer is because in this manner the network (the Discriminator in that case) is allowed to learn its own spatial downsampling [6].

The **Fractionally-strided convolution** operation is in essence the opposite of a convolution operations. In a traditional convolution operation there is a downsampling of the input dimension in the output (see Figure 2.2). Instead, in a fractionally-strided convolution operation, is produced a larger output from the input.

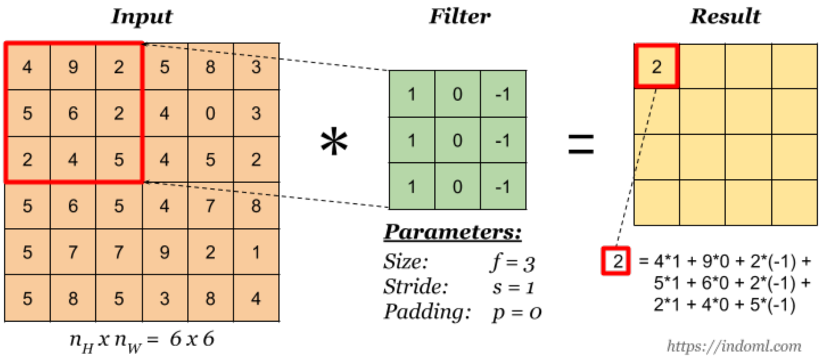

Figure 2.2: Example of a traditional convolution operation [2].

The power of this upsampling operation is that fractionally-strided convolution has **learnable parameters**. This parameters can be trained in order to follow the objective function and maximize the performance.

2.2 Conditional GAN (cGAN)

In the previous section i explained DCGAN, that is a particular GAN which differs from the original version by using particular convolutional layer (both in the Generator and Discriminator). The main functionality once trained is exactly the same as GAN: take as input an noise vector from latent space and give as output an image that match the training dataset distribution.

Traditional GAN is trained in a completely unsupervised and unconditional fashion, meaning no labels involved in the training process. Though the GAN model is capable of generating new realistic samples for a particular dataset, we have zero control over the type of images that are generated [8].

This particular variant, **Conditional GAN**, were introduced by Mehdi Mirza and Simon Osindero [4]. To understand the cGAN mechanism, think about a traditional GAN trained to generate images that match a dataset with handwritten number (e.g. MNIST dataset). With a GAN you pass a noise vector as input and you get an images as output. You don't have power on the number that GAN will output to you (e.g. it may be a 9 or it may be a 1). Of course you can have explore the latent space and find the area of the distribution which you can sample a latent vector that will give you a specific number as output, but to obtain this you have to do a type of brute-force until you find the area which corresponds to a certain number. This problem occurs with any dataset you used to train a GAN. The cGAN is a solution to this by giving some type of control on generation phase. In the Figure 2.3 we can see the differences on the structure of GAN and cGAN.

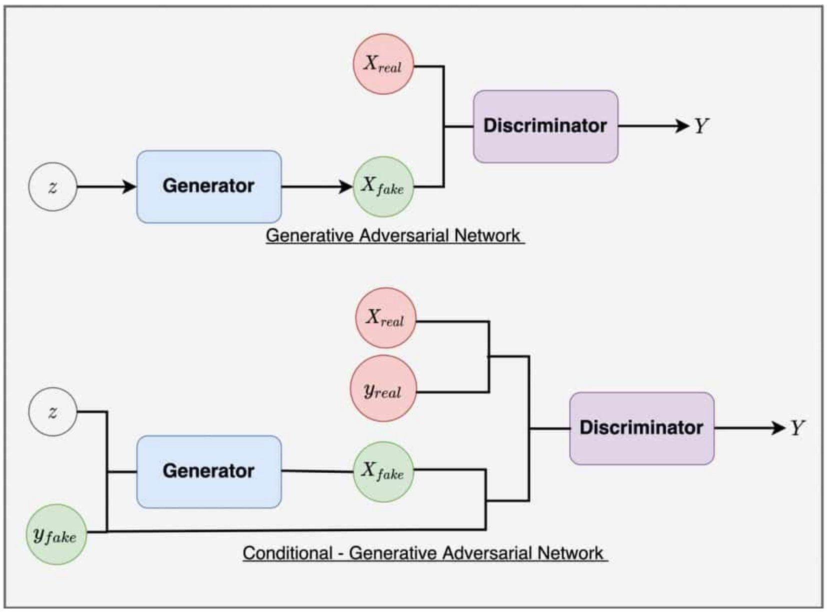

Figure 2.3: Flow Diagram representing GAN and Conditional GAN [8].

As you can see in the Figure 2.3, the Conditional GAN structure introduce an external information (all the other components is exactly the same as original GAN). This information could be a class label and is used to condition the model during the training so that is possible to condition both the generation phase.

2.2.1 cGAN's Generator

In GAN, the Generator use only a latent vector of noise to generate a sample image. In a conditional GAN the Generator also has an auxiliary information that tells which particular class sample to produce. So, there is a **conditioning label y** that is passed to G with noise vector Z:

$$GAN \rightarrow G(Z) \tag{2.1}$$

$$cGAN \rightarrow G(Z, y) = Z|y \tag{2.2}$$

Therefore, is not enough for the Generator to produce a realistic-looking images, but it is equally important that the generated sample match the label y. If the Generator has been properly trained, is possible to condition the image generation using a label.

2.2.2 cGAN's Discriminator

At the same time, the Discriminator receives both true and false examples with labels. In this way, the discriminator learns both to recognize real data from false ones and whether these samples match the labels. Then, the discriminator will reject realistic looking samples (generated by Generator) but not matching the label in the same way as fake samples.

2.3 Conditional Deep Convolutional GAN (cDCGAN)

Conditional Deep Convolutional GAN is a conditional GAN that use the same convolution layers as DCGAN that is described previously. cDCGAN generate more realistic images than cGAN thanks to convolutional layers and in the next Chapter some examples will be presented.

Chapter 3

Implementation and results

Chapter 4

Conclusion

Bibliography

[1] Afnan Amin Ali. Gan (generative adversarial network). `https://medium.com/swlh/gan-generative-adversarial-network-3706ebfef77e`, 2017. Accessed: 2022-09-22.

[2] Akshay Cboulwar. The art of convolutional neural network. `https://medium.com/@achoulwar901/the-art-of-convolutional-neural-network-abda56dba55c`, 2019. Accessed: 2022-09-22.

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. URL `https://arxiv.org/abs/1406.2661`.

[4] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[5] Packt. Principles of gans. `https://subscription.packtpub.com/book/big-data-and-business-intelligence/9781788629416/4/ch04lvl1sec24/principles-of-gans`, None. Accessed: 2022-09-22.

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

[7] Aditya Sharma. Introduction to generative adversarial networks (gans). `https://learnopencv.com/introduction-to-generative-adversarial-networks/`, 2021. Accessed: 2022-09-22.

[8] Aditya Sharma. Conditional gan (cgan) in pytorch and tensorflow. https://learnopencv.com/conditional-gan-cgan-in-pytorch-and-tensorflow/#CGAN_TensorFlow, 2021. Accessed: 2022-09-22.