Convolutional Generative Adversarial Networks

Sergi Sánchez Hernández

Resum– L'objectiu d'aquest treball és implementar tres models de xarxes GAN diferents, com són la DCGAN, WGAN i WGAN-GP, per tal de generar imatges similars a les que conformen els datasets de MNIST i de CelebA. Per a cadascun dels tres models es realitzen quatre entrenaments, utilitzant diferents nombres d'imatges dels dos datasets per tal d'obtenir una sèrie de mètriques quantitatives com són la FID, IS i Precision/Recall per tal d'analitzar la qualitat de les imatges generades i comparar els models entre si. Els entrenaments duts a terme ens han indicat que el model DCGAN és el que assoleix millors resultats, a pesar de les millores que inclouen els altres dos models.

Paraules clau– Intel·ligència Artificial, CelebA, Xarxes Generatives Adversàries, Distribució d'Imatges, Generació d'Imatges, InceptionV3, MNIST, Distància de Wasserstein, Joc de Suma Zero

Abstract– The aim of this project is to implement three different GAN network models, such as the DCGAN, WGAN and WGAN-GP in order to generate images similar to those that make up the MNIST and CelebA datasets. For each of the three models, four training sessions are carried out, using different numbers of images from the two datasets with the intention of obtaining a series of quantitative metrics such as the FID, IS and Precision/Recall in order to analyze the quality of the images generated and compare the models with each other. The experiments carried out have shown us that the DCGAN model is the one that obtains better results, despite the improvements included in the other two models.

Keywords– Artificial Intelligence, CelebA, Generative Adversarial Networks, Image Distribution, Image Generation, InceptionV3, MNIST, Wasserstein Distance, Zero-Sum Game

1 Context

The popularity of Artificial Intelligence has been growing by leaps and bounds in recent years due to the release of multiple potent models as CHATGPT or DALL-E, which are able to perform tasks such as dialogue, process natural language, generate fictitious images, etc. In this work, we will delve into one of the most important innovations in the field of Machine Learning, like the GAN's. GAN's (Generative Adversarial Networks) [1] are an approach to generative models that use Deep Learning algorithms, such as convolutional neural networks, to perform unsupervised machine Learning tasks. This approach is based on treating the problem as a zero-sum game between two neural networks, one network in charge of learning to generate data similar to the real input data (Generator), and the other network learning to differentiate or classify the real data (Discriminator). In this work we will implement three GAN models to generate fictitious images from two datasets, one dataset of black and white images and another dataset of color images. The networks that we will develop are:

- DCGAN [2] (Deep Convolutional GAN): It is the first GAN network that incorporates convolutional layers in both Generator and Discriminator, being a great advance over the non- convolutional GAN network.
- WGAN [3] (Wasserstein GAN): This network seeks to correct the instability when training DCGAN's, seeking to obtain better results when trying to bring the probability distributions of the Generator and Discriminator closer.
- WGAN-GP [4] (Wasserstein GAN with Gradient Penalty): It is considered an enhancement to the WGAN network as it adds a gradient penalty that reduces the time it takes for the Generator and Discriminator to converge/train.

We can appreciate that each model tries to improve

• Contact e-mail: sergisanher@gmail.com
• Specialization: Computing
• Work tutored by: Jordi Casas Roma (Computing)
• Course 2022/23

the previous one, so we hope to be able to notice these improvements when comparing the results of this work.

2 OBJECTIVES

The main objective of this project is to be able to successfully implement the proposed GAN models (DCGAN, WGAN and WGAN-GP), and to compare their performances and results in different datasets. Additionally, we must carry out a series of preliminary objectives that will allow us to fulfill the main objective. These preliminary objectives are the following:

1. Study the operation of GAN networks and the models that we will implement, both their architecture and the key points that differentiate the models.
2. Carry out the implementation of the proposed networks in Python language from scratch, by using the PyTorch library.
3. Implement a set of functionalities that will allow us to visualize the results obtained by the networks, and be able to compare them.
4. Adjust the hyperparameters of the models, train them with two datasets and study the obtained results using the metrics/functionalities.

3 METHODOLOGY AND PLANNING

To carry out this project, we'll use the work management methodology Scrumban [5], which is a union of the Scrum and Kanban agile methodologies. Scrumban was born originally as a transition methodology for work groups that wanted to transition from Scrum to Kanban or vice versa, in order to make the change in methodology gradually and more easily. Despite its initial purpose, the combination of these two strategies was found to be more beneficial in some cases.

This strategy uses a Kanban board where the phases through which the tasks can be organized in different columns are represented, and contain all the tasks to be completed in the project. The tasks are represented by cards that are initially in the first phase, and they will advance through the board according to their progress until they are finished. Scrumban also inherits the iterative component of Scrum, dividing the course of the project into 2-week Sprints, which will have different tasks to perform in order to accomplish the Sprint goal.

In order to carry out this methodology, we will use the KanbanTool [6] web application, which is specifically designed for this strategy, and we can configure it according to our project.

To develop this project, a series of tasks have been determined that roughly represent the most important steps to be carried out and are grouped into three main phases:

- **Initial:** In this first phase, the analysis and study of the subject are carried out in order to define the objectives, planning, and methodology to follow. This first phase has an estimated duration of one month.
- **Development:** In the development phase, we will implement the proposed models, the qualitative metrics, and all the extra features in order to perform the model trainings and obtain the final results. The estimated duration of the development phase is two months.
- **Final:** With the results obtained, we will have them graphically, and we will finish writing this work. This last phase has an estimated duration of nine days, ending on June 13.

The Gantt chart of the planning is shown in A.1. A 14-day margin has been left in order to contain all the possible risks and inconveniences that may arise during the execution of the project. We will carry out the tasks in 2-week Sprints, in which each Sprint will have set objectives with different tasks to be done. In addition, we will also carry out a version control of the project with Git [7], which will allow access to the code for anyone who would like to reproduce the results of this project.

4 STATE OF THE ART

In this section, we will present in broad strokes the most important steps that have been taken on Generative Adversarial Networks. In 2014, computer engineer Ian J. Goodfellow presented the first generative model that used an antagonic process to train two neural networks in order to generate synthetic images [1].

This first GAN model presented a revolution in the area of deep learning, both for the innovation of the method to obtain the results and for the results themselves. It was only one year later that Alec Radford used convolutional and filtering layers in the networks of the GAN model, in order to improve feature extraction from the images and thus improve the quality of the generated images, proposing the DCGAN model [2]. Despite the results obtained, the training of this type of model had many problems, and many corrections and improvements were proposed to solve them. One of the models that best manages to correct these problems was the WGAN model proposed by Martin Arjovsky in 2017 [3], which also tries to reduce the distance between the distances of the real and fake image distributions using a different cost function. There were other models that also tried to improve DCGAN, such as the WGAN-GP [4], or BEGAN [8], by changing the cost function. In addition, other new models innovated in the architecture of the Generator network, being the most important since it has to learn to generate the best possible images.

Among these models is the ProGAN, which was published in 2017 by Tero Karras [9], which progressively trains both networks in terms of resolution, starting with 4x4 images up to 1024x1024 images. In this way, a higher level of detail is achieved, resulting in images with higher resolution. GANs have been a wide field of study for computer engineers since they became known, so there are a lot of different models, and naming them all is beyond the scope of this paper. Among these, we would like to mention Tero Karras StyleGAN [10], which uses the latent space of image features to generate images with a set of desired features. Xinato Wang's Enhanced Super Resolution GAN (ESRGAN) published in 2018 [11], increases the resolution of images with great quality.

5 DATASETS

We have selected two datasets to test the different models, which are well-known datasets and have already been used to compare the results of our models.

- **MNIST:** The MNIST Dataset [12] is made up of 70,000 gray-scale images of handwritten characters with 28x28 pixels. This set of images is used in the entire area of Machine Learning, and it will serve as a starting point to implement and test our models. The advantage of this dataset is that they are black-white images, so the training will be much faster.

Fig. 1: Images from MNIST Dataset

- **CelebA** The CelebA dataset [13] is the quintessential GAN dataset, being the most widely used to show the results that a GAN model can generate and having special relevance for the StyleGAN model. CelebA or CelebFaces Attribute Dataset, is a set made up of more than 200,000 color images of famous people's faces, with a resolution of 178x218 pixels.

Fig. 2: Images from CelebA Dataset

6 GENERATIVE ADVERSARIAL NETWORKS

Generative Adversarial Networks are a group of neural network models that are characterized by using two deep neural networks that perform a zero-sum game between them, one in charge of generating images similar to the real ones (Generator), and another in charge of classifying between real and fake images (Discriminator).

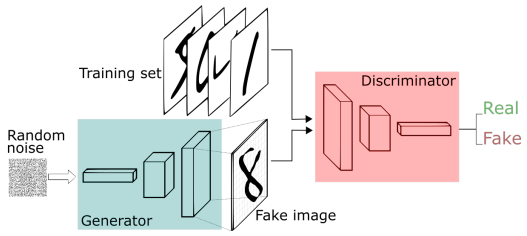

Fig. 3: Generative Adversarial Network Behaviour [14]

This process carried out by the two networks can be seen in figure 3. The Generator, starting from a vector of random data, will try to generate images similar to the real ones. This set of false images together with a set of real images, will be the input of the Discriminator, which will have to classify them according to whether they are real or fake. In the early stages of training, the Discriminator will classify these sets quite well, but as training continues, it will become increasingly difficult until it reaches the point where the Discriminator is not able to differentiate between the two sets. This point is called Nash Equilibrium, and it will occur when the classifier is forced to guess if the images are real or not (with a 50% probability), and we obtain loss values of 0.5 in both networks. The Nash Equilibrium is practically impossible to achieve in practice[15].

6.1 Loss Function

Loss functions in deep learning algorithms measure the distance between the obtained and expected results so that the model can update its parameters and obtain better results. In the case of GANs, their loss function is the zero-sum game presented below.

$$\mathcal{L}_{GAN}(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1-D(G(\mathbf{z})))] \quad (1)$$

Fig. 4: GAN Min-Max Loss Function

The first term of equation (1) corresponds to the loss function of the Discriminator, and the second term of the expression corresponds to the Generator loss function. The first term calculates the weighted mean of the logarithm of the Discriminator results on the real images, and the second term does the same calculation on the images generated by the Generator, but is subtracted from 1 to calculate the logarithm. In this way, classifying the real images as 1's and 0's as the false ones, the Discriminator wants to maximize both terms by obtaining logarithms of 1, which results in 0, while the Generator wants to minimize the second term by obtaining logarithms of 0, which results in minus infinity. As we have mentioned before, the Nash Equilibrium will be reached when the result of the equation is 0.5, since the Discriminator in both terms will classify the images randomly.

6.2 GAN Training Problems

Despite how idyllic the GAN training process appears, several problems arise when dealing with real data.

- **Hyperparameters:** GAN networks are very sensitive to hyperparameters, being able to obtain very different results depending on their values, and it being difficult to find the optimal values.
- **Non-Convergence and Instability:** The convergence is achieved when the Generator creates images indistinguishable from the real ones, and therefore the percentage of hits of the Discriminator is 50% (Nash Equilibrium). This convergence may not be achieved due to the following situations:
  - The Discriminator network perfectly distinguishes between real and fake images, and therefore the Generator network cannot learn.
  - The Generator network manages to always fool the Discriminator, so the same Generator will never modify its parameters, and it will not improve.

  - Another possible scenario would be when the losses from both networks oscillate in each iteration due to an incorrect Learning Rate value, which controls the updating of the parameters of both networks.

- **Mode Collapse:** This problem refers to when the Generator produces very similar synthetic images with very similar characteristics. In the case of the MNIST dataset, the Generator would only be able to create images from one or two certain numbers.

Despite the fact that these problems do not have a complete solution that prevents them, improvements have been made to the original GAN and DCGAN models in order to alleviate them.

6.3 DCGAN

The DCGAN (Deep Convolutional GAN) is a model based on Ian J. Goodfellow's GAN, but it adopts the CNN (Convolutional Neural Network) architecture for the Generator and the Discriminator. Instead of using linear layers, the DCGAN used convolutional layers that allow us to learn more information from the images as well increase their dimension or vice versa. Both networks are composed of a series of layers that are repeated in their architectures. In our code, we have defined a block that contains these repeated layers in order to have a shorter and more understandable code. Next, we will describe a block of the Generator network.

Generator Block:

- **ConvTranspose2d:** Applies a two-dimensional transposed convolution to each of the image channels. This transposed convolution allows us to extract the characteristics of the image using a matrix/kernel, which at the same time increases their dimensionality. In the function, you have to define the input and output sizes of the images, the size of the kernel that will extract the features and transform their dimensionality, the stride that defines the dimensionality change, and the padding that allows to omit N pixels of the edges of the images (commonly not interesting pixels).
- **BatchNorm2d:** Normalizes the resulting images for each channel after applying the convolution. This normalization is applied for each batch of images and allows for speeding up the training of the network since it reduces the change in the distribution of the activations of the network during the tuning of the parameters.
- **ReLU:** Activation function that transforms the negative parameter values of the network to a value of 0 and does not change the positive values. The formula for this function is to calculate the maximum value between 0 and the value of the parameter.

The Generator block is composed of a Transposed Convolutional layer that increases the dimensionality of the input data, a standardization of the images so that they have a mean close to 0 and a standard deviation close to 1, and finally the ReLU activation layer that transforms negative values to 0.

Block	Input Shape	Output Shape
Block 1	100 x 1 x 1	1024 x 4 x 4
Block 2	1024 x 4 x 4	1024 x 8 x 8
Block 3	512 x 8 x 8	1024 x 16 x 16
Block 4	256 x 16 x 16	1024 x 32 x 32
ConvTranspose2d Tanh	128 x 32 x 32	64 x 64

TABLE 1: GENERATOR NETWORK ARCHITECTURE.

The Generator is made up of 4 consecutive blocks where the dimensionality is increased from the initial noise vector of 100x1x1, up to a 64x64 image. The last layer does not contain a standardization layer, and the activation function is the Tanh. This architecture is given in Table 1.

This structure will be the same in the Discriminator, changing the Transposed Convolution to a Convolution, which decreases the dimensionality instead of increasing it, and the ReLu to the LeakyReLU activation function (except in the last layer, where the Sigmoid function will be used).

6.4 WGAN and WGAN-GP

As we have mentioned in section 6.2, the first GAN models suffer from different problems, such as training instability and hyperparameter sensitivity, among others. The WGAN and WGAN-GP models arose to try to combat some of these problems by adding improvements to the DCGAN model. It is worth mentioning that the mathematical concepts behind these two models are quite complex, therefore, due to the space limitations of this project, reading Alexirpan's summary [16] is recommended, to delve into the mathematics behind these two models.

6.4.1 WGAN

This new model presents a paradigm shift, treating the images generated by the Generator and the real ones as data distributions, which we want to overlap or be as close as possible to obtain high quality images. In order to calculate the distance between two distributions, WGAN chooses to use a loss function that implements the Wasserstein Distance instead of the Jensen-Shannon (JS) divergence that is used in DCGAN. Jensen-Shannon divergence is derived in many cases in unstable training due to issues with the gradients [17].

Also, the Jensen-Shannon Divergence does not take into account the distance between these distributions, so in two different scenarios where we have separate distributions (bad Generator images) being in one scenario much more noticeable than in the other, the gradients that we would obtain would be the same, and would be insignificant. For this reason, the DCGAN is very unlikely to converge, and it is reflected in the loss values, since when one network improves, the other worsens, like this indefinitely.

As we see in equation (2), this new loss function is quite similar to the DCGAN, where the averages of the values returned by the Discriminator are calculated for both real and fake images. This new function dispenses with logarithms, performs subtraction of the terms instead of addi-

$$\mathcal{L}_{\text{WGAN}}(D,G) = \mathop{\mathbb{E}}_{\mathbf{x}\sim P_{\text{data}}}[D(\mathbf{x})] - \mathop{\mathbb{E}}_{\mathbf{z}\sim P_{\mathbf{z}}}[D(G(\mathbf{z}))] \quad (2)$$

Fig. 5: WGAN Loss Function

tion, and dispenses with subtracting the result of the second term from 1. In WGAN, the final activation function Sigmoid is removed from the Discriminator, in such a way that the results will not be values in range [0, 1]. For this reason, the Discriminator is renamed as Critic. In short, the objective of the Critic is to separate the results of the terms as much as possible (maximize), and the Generator wants the result of the second term to be very similar to that of the first term. This gives us a termination criterion for the training, since when the generated images are very similar to the real ones, the result of the loss function will tend to zero. The termination criterion is correlated with generating high-quality images, according to the empirical results of the WGAN authors. Therefore, when the loss values of the two networks are close to 0, the quality of the fake images will be practically unbeatable.

6.4.2 WGAN - Weight Clipping

We have mentioned that in the DCGAN, when the Discriminator improves its loss, the Generator worsens, and vice versa indefinitely. In order to solve this problem, the WGAN Critic seeks to limit the gradients, and thus be able to reach a minimum. For this, the condition is added to the Critic that it must be 1-Lipschitz, a mathematical property that limits the result of the Critic to be between [0, 1], in order to limit the slope of the function and therefore the gradients. In order to comply with this restriction, the WGAN chooses to adjust the Critic weights to values between [-0.01, 0.01], and in this way obtain values close to [0, 1], although they will not necessarily reach 1-Lipschitz. As the authors of the WGAN indicate in their paper [3], weight clipping is a terrible way to force the Critic to be 1-Lipschitz, since if the weights of the network are greatly reduced, it will take a long time to converge, and if they are reduced little, it will result in vanishing gradients.

6.4.3 WGAN-GP

The WGAN-GP model differs from the WGAN in the way that it forces the Lipschitz constraint on the Critic. WGAN-GP stands for WGAN with Gradient Penalty, since it replaces Weight Clipping with a restriction on the Critic gradients norm. In the WGAN, the network architecture is modified since we are clipping the parameters of the network, but with the gradient penalty, only the gradients that will update these weights are modified.

$$\text{GP} = \lambda \mathbb{E}_{\hat{\mathbf{x}}\sim p_{\hat{\mathbf{x}}}(\hat{\mathbf{x}})}\left[\left(\left\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\right\|_p - 1\right)^2\right] \quad (3)$$

This penalty or regularization is added to the loss function of the WGAN, and its objective is that the norm of the gradients generated by the loss function be at most 1, in order to comply with the Lipschitz restriction. The norm of the gradient vector is used to measure the "steepness" of the loss function at current parameter values. Therefore, limiting the norm of the gradients to a value of 1 ensures that the loss function is 1-Lipschitz continuous. Next, we will explain the steps carried out to calculate the penalty that is represented in equation (3):

1. **Image Interpolation:** given a set of real and generated images, a linear interpolation of these is performed, represented as $\hat{\mathbf{x}}$.
2. **Gradients Computing:** the Critic gradients are computed over the interpolated images, represented as $\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})$.
3. **Gradients Norm:** the norm of the gradients is calculated, where $|\cdot|_p$ represents the norm operator.
4. **Penalty Compute:** The penalty calculation is performed where λ is a hyper parameter that controls the weight of the penalty. We have used a λ value of 10.

This gradient penalty penalizes the Critic gradients with norms greater than 1, forcing the norm to be close to 1, thus satisfying the Lipschitz constraint.

6.4.4 Gradient Penalty vs Weight Clipping

Both gradient penalty and weight clipping satisfy the Lipschitz constraint, limiting the GAN training in order to stabilize it and having a useful loss value. Despite this, the gradient penalty has many advantages over weight clipping:

1. **Improved Training Dynamics:** Weight clipping involves constraining the weights of the Critic to a certain range, and this can lead to training instability. It may struggle to find the balance between the Critic and the Generator, resulting in poor convergence. In contrast, gradient penalty helps to stabilize the training by penalizing the gradients without imposing a constraint on the weight values.
2. **Avoidance of Mode Collapse:** Better training stability prevents Mode Collapse which causes a poor diversity of images. Since gradient penalty stabilizes better, it helps to avoid it, and promotes the generation of diverse and high quality samples.
3. **Flexibility in the Critic:** Weight clipping can be sensitive to the clipping value, requiring careful adjustments to achieve the desired effect. On the other hand, the gradient penalty doesn't need to modify those parameters and doesn't need manual tuning.
4. **Consistency of Wasserstein Distance:** The gradient penalty approach is directly motivated by the Wasserstein distance, which measures the discrepancy between real and generated images. Unlike weight clipping, it provides a theoretical foundation for the WGAN-GP and ensures that the Critic approximates the Wasserstein distance more accurately.

Overall, the gradient penalty in the WGAN-GP offers better training stability and generates diverse and high quality images. It is also important to mention that the WGAN-GP takes longer to train than the WGAN.

7 Evaluation Metrics

In order to verify that the models we are training are actually learning, and can reach the objective of generating good false images, we need some metrics that help us see the evolution of the models during training and also at the end.

- **Fidelity:** refers to how similar the false images generated are compared to the real ones.
- **Diversity:** refers to the variety of the generated images, which is the diversity of real images and their distributions.

With these two properties, we can have a broad vision of how well our Generator learns. There are many metrics used for image comparison that are used in the GAN field: FID (Fréchet Inception Distance), IS (Inception Score), Precision/Recall, KID (Kernel Inception Distance), PPL (Perceptual Path Length), among others. We have decided to implement FID, IS and Precision/Recall, since the first two are most commonly used metrics to compare the results of GAN models, and Precision/Recall because they are widely used metrics in the area of Machine Learning.

7.1 Inception Score

The IS and the FID are the most commonly used metrics in GAN network evaluation, and they are the ones that are mainly used to compare the results between the different models that exist. This metric measures both the fidelity and diversity of the generated images, being, according to its authors, a metric that has a good connection with the human evaluation of the images. The Inception-V3 [18] classifier is used, which has already been trained with the ImageNet [19] data set that contains 1.000 different classes, in order to obtain the probability distribution of the generated images for each of the classifier classes. These probabilities show us the probability that each of the classes appears in the images, being indicative of diversity if we have an extended distribution or fidelity, if we have classes of high probability. In the case of figure 6, we see that the probability distribution of the image has a certain resemblance to a normal distribution but is spiky. Ideally, we want each of the images to have few classes with a high probability (high fidelity), but overall, the dataset should be more evenly distributed (high diversity). The Inception Score seeks to compare the distribution of each image with the total image distribution, obtaining a large difference between the distributions if the ideal case occurs and therefore a large IS value. To calculate the IS value, the KullBack-Leibler Divergence statistical formula is used, which measures how different two distributions are, obtaining zero as the worst result or infinity, although this does not happen in practice.

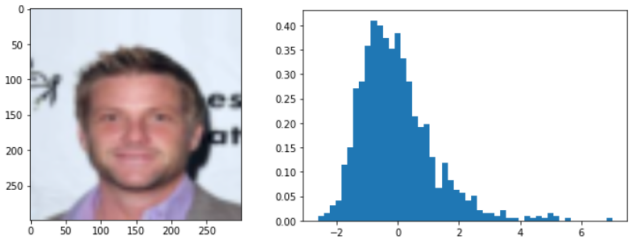

Fig. 6: Predictions Distribution of a CelebA Image

Despite being one of the most used metrics to compare models, it is not very reliable since the calculation of this metric does not take into account the real images, and the InceptionV3 classifier has not been trained with images from our datasets, so the best results may not be linked to better generated images.

Despite this, the FID metric that has replaced the IS can alleviate and solve some of these limiting factors.

7.2 Fréchet Inception Distance

As we have mentioned in the previous section, despite the fact that both the FID and the IS are used to compare the different GAN models, the FID is a much more reliable metric. This metric is based on the Fréchet Distance, which is a distance metric for curves and can be extended to distributions.

It works especially well with normal multivariate distributions, which generalize the idea of a normal distribution to high dimensions, and allow more complex distributions to be modeled in a single parameterization, that is, a single mean and a single standard deviation. In order to obtain these distributions, we will use the Inception-V3 model pretrained with the ImageNet dataset as with the IS, but we will remove the last layer of the network (SoftMax) and be left with the activations of the last lineal layer of the model. We will obtain these feature vectors for a subset of real and fake images, which will be our multivariate normal distributions, and then calculate the Fréchet Distance between these distributions.

The final objective of this metric is to ensure that the characteristics of the real images and the false ones are as similar as possible, so we will obtain smaller values the shorter the distance between the activations and larger values the greater the distance.

Despite being a metric that compares real and fake images and gives us a good estimate of how similar they are, it also has its limiting factors:

- Like the IS, it uses the pre-trained Inception-V3 model, so it may not capture all the characteristics of the images of our datasets.
- In order to obtain a good FID, it is advisable to use a large number of images to make the comparison and obtain a better result. Since this is a computationally expensive operation, it can greatly delay the trainings.
- The resulting FID value tends to be better (a smaller value) the more samples are used to perform the calculation. This results in a lot of improvement in the FID that is not reflected in the Generator model.
- Limited statistics were used. FID only uses the mean and the covariance, which are the two main properties of distributions, but they do not cover all aspects of them.

7.3 Precision and Recall

Precision and Recall are two of the most used metrics in neural networks, especially in classifier models, where it is easy to keep track of well and badly classified samples.

Our Generator model creates fake images with the goal of

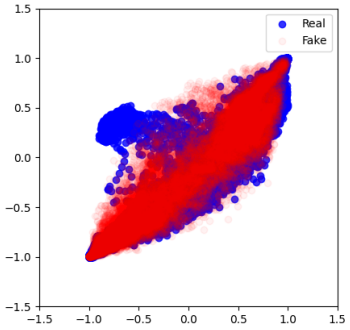

Fig. 7: Fake and Real Image 2D Distribution

being very similar to the real ones therefore, we want their distributions to be equal or the same. In figure 7 we can observe the distribution of 5 real images and 5 false ones of the CelebA dataset at epoch 8 in DCGAN, where the distribution of the false images is quite accurate to the real one.

7.3.1 Precision

Precision evaluates the fidelity of the images. Taking figure 7 as an example where we have two distributions, the precision counts all those red values that overlap with blue values, and divides this number by the total of red points. In short, Precision measures the percentage of false images that appear real over all the false images, although we could suffer Mode Collapse if it only generates images from a subset of the real distribution.

7.3.2 Recall

The Recall evaluates the diversity of the images. As in precision, it counts all those overlapping points between both distributions but divides this number by the number of blue points. Recall measures the percentage of false images that appear real over the real images.

7.3.3 Implementation - KNN

In order to implement Precision and Recall we have used the Inception-V3 network to obtain the feature vectors of a subset of real and false images, just like the FID. We have carried out a KNN algorithm, where for each of these feature vectors we have calculated the distance to the other vectors of the same set of vectors (real or false images), in order to keep the third shortest distance and use it as a radius of the vector. Once we have these radii, we have to calculate the distances between each of the false image vectors and each of the real image vectors, and if the distance is less than the vector radius, we increase the counter of precision by 1. This process is also done in reverse to increase the recall counter, and finally we divide these counters by the total number of real vectors (or false if they are the same size).

In this way, using the characteristics of the images that are vectors of 2.048 values, we can calculate the Precision and Recall more easily than using the values of the pixels of the images.

8 Models Implementation

This section explains in broad strokes the steps we have followed to carry out the training, from downloading the raw datasets to obtaining the results. It is worth mentioning that we will not explain step by step the training loop, nor the differences between the models, which are a few. However, you can consult our source code for a better understanding.

8.1 Data Acquisition

Both datasets (MNIST and CelebA) have been downloaded from their respective official pages in order to preprocess them. MNIST contains a total of 70,000 images, and the CelebA dataset contains a total of 202.599. The fact of processing the original files has allowed us to create different files with different numbers of images for each dataset, allowing us to carry out training with more or less images, and check if the more images for the training have an impact on better results. We have decided to create two sets of images for each dataset, one of 30.000, and another of 70,000, which we will compress to be able to upload them to Google Drive, and to be able to use them in Google Colab.

8.2 Data Processing

To load the data from the compressed files, we have implemented a CustomDataset class, which will allow us to load the images dynamically during the training, and apply transformations to them. The images need to have the same dimensions as the images generated by the Generator (64x64), so they must be resized. We also have to normalize the images for each channel to a mean and standard deviation of 0.5, so that they have values between [-1, 1], and finally, we convert the image arrays to tensors.

8.3 Before Training

In addition to loading the images, we have to declare other objects/variables that are necessary to perform the training of both neural networks, such as the gradient descent algorithm, the loss function, and other hyper-parameters such as the learning rate. In order to calculate the quantitative metrics, we have created a class called GAN-Evaluator, which has a set of methods that calculate the metrics (IS, FID and Precision/Recall), and others that attend to them. Before starting the training, it must be initialized, since it will load the InceptionV3 model, and it will also calculate and save the activations of the real images for later use in FID and Precision/Recall. We also have created the GAN utils class, which has a great variety of methods that will be used mainly to generate and save plots, images generated by the Generator, training checkpoints, and the results in text format. This class must be initialized before loading the dataset, since it also sets a seed to determine the randomization of the training, which will allow our results to be reproducible.

8.4 Training Strategy

We have to carry out 12 different training sessions, since we have 4 datasets and 3 models. Each training consists of a double loop, where we pass all the dataset images to the model in batches, so that it generates false images and the Discriminator/Critic compares them with that batch of real images, and then modify the network parameters. This process of passing all the images to the network occurs 50 times, also called epochs. Apart from the actual training of the networks, for each epoch or number of epochs, we will perform some calculations in order to generate and save the results:

- **Loss Values:** for each epoch we will save the loss values from the Discriminator/Critic and the Generator.
- **FID and IS:** for each epoch we will generate 1.000 images to calculate the FID and IS, and then we will store these values in an array.
- **Precision and Recall:** every 5 epochs we will calculate and save in an array, the Precision and Recall using the 1.000 images generated for the FID and IS calculations.

Finally, when the 50 training epochs have been completed, the following is performed:

- **Losses Chart:** the chart of the losses from the Discriminator/Critic and the Generator per epoch is displayed, and save it.
- **Training Time:** the total training time is calculated.
- **FID and IS Charts:** the chart of the FID and IS values per epoch is displayed and save it.
- **Final Model:** the Generator parameters of the model are saved, obtaining the already trained model, which can later be used to generate fake images.
- **Text Results:** a text file containing all the numerical results of the training is saved.

9 Results

In this section, we will show and comment on the results obtained by each of the models, analyzing and comparing them with each other. Before we begin, we want to highlight that the IS metric has not given conclusive results since, as we already explained, this metric is based on the results obtained from the InceptionV3 classifier, which has not been trained with our datasets.

9.1 DCGAN

To carry out the DCGAN trainings, we have used the T4 GPU, which is the least powerful offered by Google Colab. As we see in table 2, the duration of the trainings has been relatively short, since for the datasets of 70,000 images, they have lasted approximately 2 hours and 40 minutes. The results of the FID are in accordance with expectations, obtaining better results the more images have been used for the training, being a behavior that is repeated in the other training sessions, and that is logical. The same effect could also be expected in Precision and Recall although, as we will see in this model and in the following ones, it is not always true. In the MNIST datasets, the Precision and Recall values are quite low, so we can intuit that the Generator is not capable of copying all the font styles that are in the dataset, and can generate images that differ from the real ones. On the other hand, in the CelebA dataset, we do obtain a good value in Precision, indicating that the images generated are of high quality, but there is little diversity due to low recall.

Dataset	Imgs.	Time	FID	IS	Pr	Rc
MNIST	30k	1h 21m	51.15	2.17	0.05	0.13
MNIST	70k	2h 42m	49.54	2.13	0.08	0.10
CelebA	30k	1h 16m	59.25	2.30	0.39	0.06
CelebA	70k	2h 38m	51.40	2.38	0.50	0.11

TABLE 2: Trainings Table: DCGAN

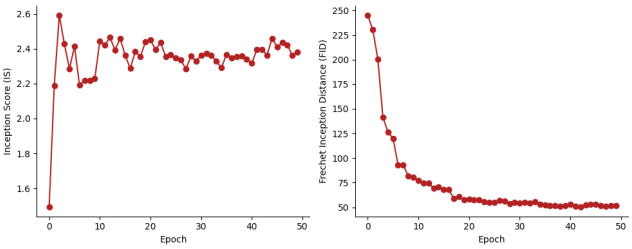

Fig. 8: FID and IS values/epoch on CELEBA 70k

The FID metric has served us both in the DCGAN and in the other models as a good metric to check for improvement during training. Figure 8 shows the FID values obtained for the CelebA70k dataset, obtaining high values at the beginning, and converging to a value of 51 in the last epochs. For datasets with 70,000 images, the FID has taken more epochs to converge, and it could even have continued to improve. On the other hand, the training sessions with fewer images have either converged faster, as in the case with MNIST, or they needed more training to converge in CelebA. On the other hand, we can see that the IS values do not have a substantial improvement as training progresses, but that their value is very similar in the first and last epochs of training.

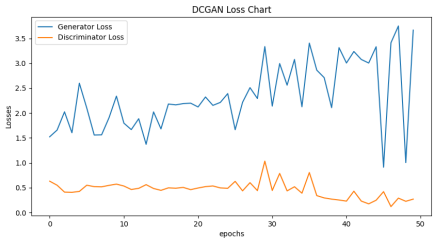

Fig. 9: Losses Chart of CELEBA 70k

As we have already mentioned, the values of the loss function in the DCGAN do not provide any information to check the state of the training. In the chart of figure 9 we can see that the loss values of both the Generator and Discriminator move away from each other, without reaching the Nash Equilibrium where both values are close to 0.5. Despite this, the loss values are not triggered on one network and set to 0 on the other, so as long as there are these loss values, both networks are learning.

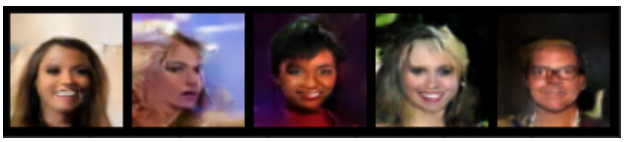

Fig. 10: DCGAN Generated Images of CelebA 70k dataset

Figure 10 shows 5 images generated with the CelebA 70k dataset, where we can see that the model has been able to imitate facial characteristics quite well, and generate images with diverse features.

9.2 WGAN

To carry out the WGAN trainings, we have used the V100 GPU offered by Google Colab, which is more powerful than the T4 used in the DCGAN and has allowed us to perform the trainings 2.5 times faster than with the T4 GPU.

Dataset	Imgs.	Time	FID	IS	Pr	Rc
MNIST	30k	1h 37m	70.58	2.12	0.02	0.06
MNIST	70k	3h 23m	54.72	2.16	0.07	0.17
CelebA	30k	2h 3m	70.05	1.97	0.30	0.03
CelebA	70k	4h 19m	58.66	2.17	0.50	0.07

TABLE 3: TRAININGS TABLE: WGAN

Despite the fact that we are using a more powerful GPU, the training times are a bit higher than in the DCGAN, needing 4 hours and 20 minutes for the CelebA70k dataset, as shown in table 3. In this model, the difference between training with grayscale and color images is notorious, since the time is longer for color images. The Precision and Recall values are generally lower than in the DCGAN.

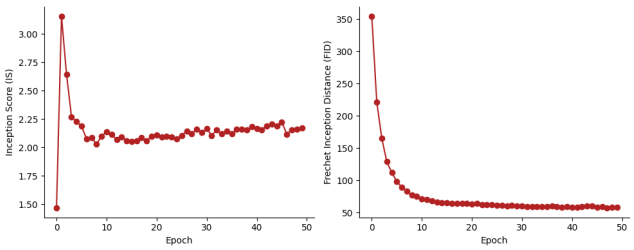

Fig. 11: FID and IS values/epoch on CELEBA 70k

The FID values obtained by this model are worse compared to those of the DCGAN, but instead, as shown in figure 11, the FID values converge much earlier and in a smoother way. This is because the Lipschitz restriction limits the learning of the networks.

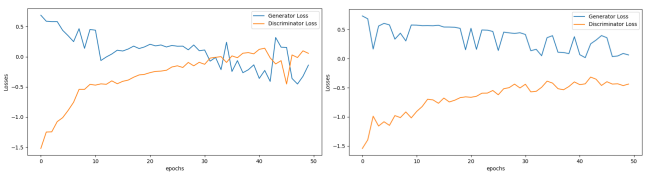

Fig. 12: Losses Charts of CELEBA 70k and MNIST 70k

In the case of the WGAN, this convergence is also noticeable in the values of the loss function. Figure12 shows us the chart of the loss functions of the CelebA and MNIST datasets with 70,000 images, and we can see that their loss values tend to converge. This convergence is correlated with high-quality image generation, so we can assume that in the case of CelebA, where there is a convergence to 0, the quality of the images will no longer be improved. This point of convergence is at epoch 30, and if we look at the previous FID chart, we can see that it is true that at epoch 30 the values practically do not get better. In the MNIST dataset it has not converged, so more training would be necessary to obtain better results and images.

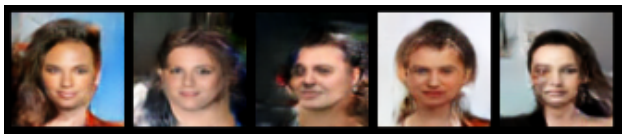

Fig. 13: WGAN Generated Images of CelebA 70k dataset

With the images shown in figure 13, we can confirm that the WGAN has obtained worse results than the DCGAN, since a slight worsening can be seen in the images.

9.3 WGAN-GP

To carry out the training of this model, we have used the V100 GPU, as for the WGAN.

Dataset	Imgs.	Time	FID	IS	Pr	Rc
MNIST	30k	3h 1m	88.14	2.33	0.01	0.05
MNIST	70k	6h 40m	63.52	2.29	0.05	0.05
CelebA	30k	3h 28m	99.95	2.57	0.15	0.08
CelebA	70k	7h 52m	77.44	2.28	0.37	0.06

TABLE 4: TRAININGS TABLE: WGAN-GP

As we can see in table 4, the training times have practically multiplied by two respect to those of the WGAN since the calculation of the gradient penalty for this model is computationally expensive. The longest training has been with the CelebA dataset with 70,000 images, which has lasted almost 8 hours. The results obtained from the FID, Precision, and Recall are worse compared to those of the DCGAN and WGAN. This is because this model takes longer to converge than the others since it limits the gradients and therefore the speed at which the networks converge. These

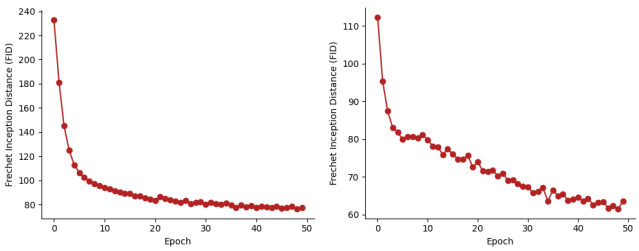

Fig. 14: FID Charts of CELEBA 70k and MNIST 70k

FID results are not the best possible results for this model since, as we can see in figure 14 where the FID values for the CelebA and MNIST datasets with 70,000 images are shown, the FID has not yet stagnated. This fact is much more noticeable for the MNIST than for CelebA, although

if we look at the values, we can see that they are decreasing. We can also appreciate that the FID obtained is much better for datasets with a larger number of images, so with more images, the results would be even better. Therefore, with more training and a greater number of images, we could obtain better results, even better than those obtained with WGAN.

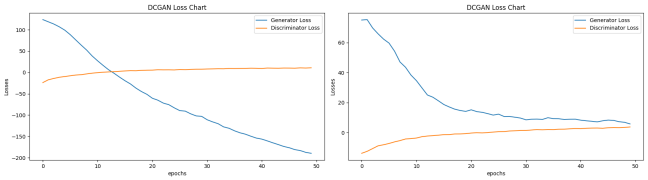

Fig. 15: Losses Charts of CELEBA 70k and MNIST 70k

On the other hand, we have the loss function values, which should be similar to those of the WGAN, trying to converge to 0. As we see in figure 15, where we have the loss values of CelebA and MNIST for 70,000 images, this is more or less fulfilled. In the CelebA training, the loss of the Generator continues to decrease once it has reached the value of 0, due to the fact that the Critic distinguishes false and real images very well.

Despite this, the FID values continue to decrease little by little, meaning that the quality of the images increases. It would be necessary to carry out a longer training to check the behavior of the Generator, and to see if its large loss can greatly increase the quality of the images to finally converge to 0. On the other hand, in the case of MNIST, the two

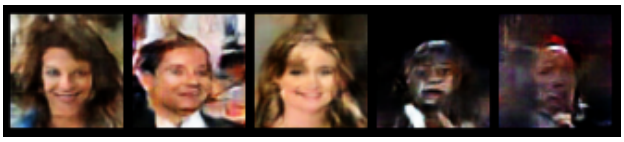

Fig. 16: WGAN Generated Images of CelebA 70k dataset

loss values do converge to the value of 0. The low results of the WGAN-GP compared to the other models are also notorious in its images, since in figure 16 we can appreciate that the quality of the first three images is not very good, and the last two have not even formed a face.

10 Conclusions

We can affirm that we have achieved all the objectives proposed for this project since we have successfully implemented the three GAN models, together with a set of metrics to compare their results. The images generated by the three models are distinguishable from the real ones, but we have achieved some resemblance between them, especially in the MNIST datasets. Regarding the quantitative metrics, we have achieved acceptable results, despite the fact that we expected the WGAN-GP model to obtain better results and it has not. It is also necessary to take into account the limiting factors of these, since the use of GPUs has been limited due to their costs and the metrics are based on a pre-trained classifier with a different dataset.

11 Acknowledgments

I would like to first thank my project tutor, Jordi Casas Roma, who has been a fundamental support and has guided me in carrying out this work. I also want to thank both my family and my romantic partner, for their confidence, and the moral support they have given me, which has been essential to keeping me motivated and focused.

References

[1] Ian J. Goodfellow and Pouget-Abadie. Generative Adversarial Networks. *arXiv*, 27, 2014.

[2] Alec Radford and Metz. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv*, abs/1511.06434, 2015.

[3] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *ArXiv*, abs/1701.07875, 2017.

[4] Ishaan Gulrajani and Ahmed. Improved Training of Wasserstein GANs. *arXiv*, abs/1704.00028, 2017.

[5] S. Laoyan. Scrumban: lo mejor de dos metodologías ágiles [2022] • Asana. https://asana.com/es/resources/scrumban. [Online; accessed on 03/12/2023].

[6] Kanban Tool – Tablero Kanban para Empresas. https://kanbantool.com/es, 2023. [Online; accessed 2. Jul. 2023].

[7] S.H. Sergi. Convolutional generative adversarial networks. https://github.com/sergissh/TFG.git, 2023.

[8] David Berthelot and Tom Schumm. BEGAN: boundary equilibrium generative adversarial networks. *CoRR*, abs/1703.10717, 2017.

[9] Tero Karras and Timo Aila. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017.

[10] Tero Karras and Samuli Laine. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018.

[11] Xintao Wang and Ke Yu. ESRGAN: enhanced super-resolution generative adversarial networks. *CoRR*, abs/1809.00219, 2018.

[12] MNIST Dataset. https://yann.lecun.com/exdb/mnist, 2013. [Online; accessed on 03/06/2023].

[13] Ziwei Liu and Luo. Deep learning face attributes in the wild. 2015.

[14] Thalles Santos Silva. A Short Introduction to Generative Adversarial Networks - Thalles' blog. https://sthalles.github.io/intro-to-gans, March 2023.

[15] Farzan Farnia and Asuman E. Ozdaglar. Gans may have no nash equilibria. *ArXiv*, abs/2002.09124, 2020.

[16] Read-through: Wasserstein GAN. https://www.alexirpan.com/2017/02/22/wasserstein-gan.html, May 2023.

[17] Lilian Weng. From GAN to WGAN. https://lilianweng.github.io/posts/2017-08-20-gan, August 2017. [Online; accessed on 06/07/2023].

[18] Christian Szegedy and Vincent Vanhoucke. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[19] Olga Russakovsky and Jia Deng. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

APPENDIX

A.1 Project Planning

Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
Final Degree Project	**102 días?**	**mar 07/02/2**	**mié 28/06/23**	
Initial Phase	**24 días**	**mar 07/02/2**	**vie 10/03/23**	
Theme Study and Analysis	14 días	mar 07/02/23	vie 24/02/23	
Objectives Definition	4 días	lun 27/02/23	jue 02/03/23	3
Select Methodology	3 días	lun 27/02/23	mié 01/03/23	3
Planning of the realization of the Project	6 días	vie 03/03/23	vie 10/03/23	4
Development Phase	**58 días?**	**lun 13/03/23**	**mié 31/05/23**	**2**
Design and Construction	**21 días?**	**lun 13/03/23**	**lun 10/04/23**	
Implement Models	16 días?	lun 13/03/23	lun 03/04/23	
Implement Support Features	5 días	mar 04/04/23	lun 10/04/23	9
Test & Run Test	**37 días?**	**mar 11/04/2**	**mié 31/05/23**	**8**
Adjust Hyperparame	5 días?	mar 11/04/23	lun 17/04/23	
Test Models	5 días?	mar 18/04/2	lun 24/04/23	12
Start Trainings	10 días?	mar 25/04/23	lun 08/05/23	13
Models Comparative	17 días?	mar 09/05/23	mié 31/05/23	14
Final Phase	**9 días?**	**jue 01/06/23**	**mar 13/06/23**	**7**
Write Final Proposal	7 días?	jue 01/06/23	vie 09/06/23	
Final Delivery of the Project	2 días?	lun 12/06/23	mar 13/06/23	17

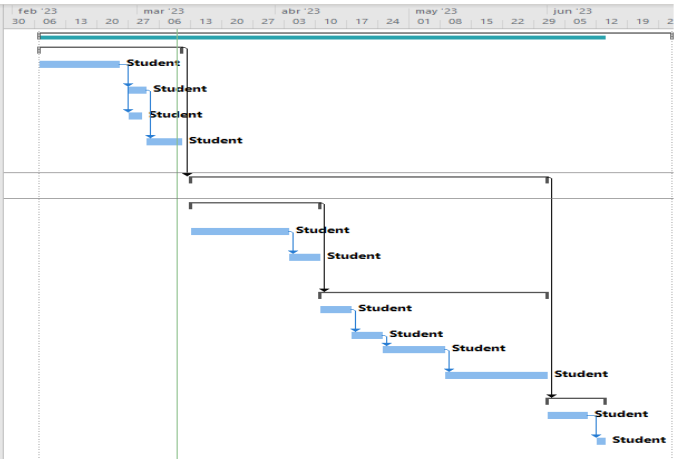

Fig. 17: Tasks and Gantt Chart

A.2 Generated Images

A.2.1 DCGAN

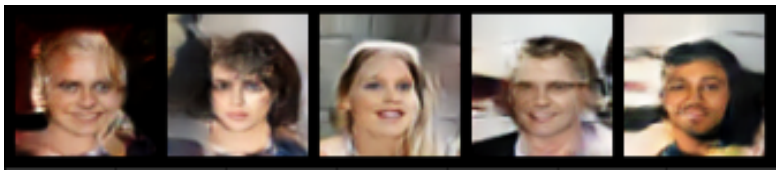

Fig. 18: Images Generated for CelebA 30k dataset

Fig. 19: Images Generated for MNIST 30k dataset

Fig. 20: Images Generated for MNIST 70k dataset

A.2.2 WGAN

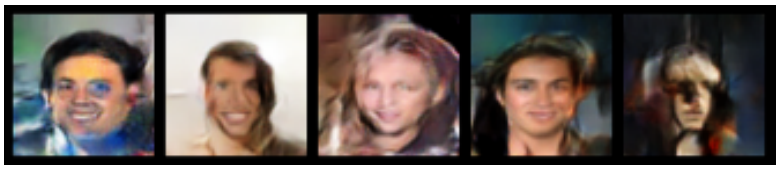

Fig. 21: Images Generated for CelebA 30k dataset

Fig. 22: Images Generated for MNIST 30k dataset

Fig. 23: Images Generated for MNIST 70k dataset

A.2.3 WGAN-GP

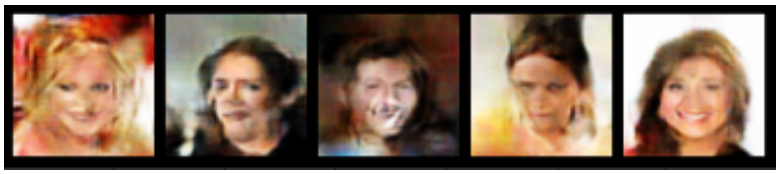

Fig. 24: Images Generated for CelebA 30k dataset

Fig. 25: Images Generated for MNIST 30k dataset

Fig. 26: Images Generated for MNIST 70k dataset