GANs for Monet Paintings
A scientific comparison of different GAN architectures

E. Musacchio[a], A. Silletti[a]

[a] *Università degli Studi di Bari Aldo Moro*

June, 2022

Abstract

The objective of this paper is to compare different Generative Adversarial Networks (GANs) architectures for the generation of paintings in Monet's style. In particular, we will show different possible approaches to solve this task and we will describe what are the advantages and disadvantages of each technique.

1. Introduction

The generation of images, paintings in our case, using Generative Adversarial Networks is a challenging task. Not only it is computationally heavy, but several architectures exist that can be exploited to fulfill this challenge, meaning that it is important to understand why we should choose to solve our problem using one model rather than another. Furthermore, the quality of the generated paintings is particularly important, as attested by the recent sale of an Artificial Intelligence generated portrait for the stunning price of $432,000$ dollars[6].

1.1. Problem Description

The problem is inspired by the Kaggle competition "I'm Something of a Painter Myself"[1]. To summarize, given a dataset of 300 Monet's paintings (of size 256×256) and 7028 landscape photos (of size 256×256), we want to train a model able to produce 10000 realistic Monet paintings (of size 256×256), either starting from the landscape photos or generating them from scratch. The generated paintings are then evaluated by Kaggle using the MiFID metric.

1.2. The Shortage of Data

We decided to extend the challenge domain and consider more GANs, since an inherent problem of many architectures is that they require several data to train properly.

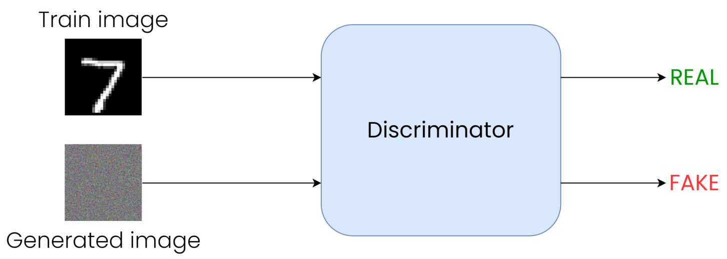
> *It typically takes* 50000 *to* 100000 *training images to train a high-quality GAN. But in many cases, researchers simply don't have tens or hundreds of thousands of sample images at their disposal. With just a couple thousand images for training, many GANs would falter at producing realistic results.[15]*

Since the competition only provided 300 images as training set, only a few specific architectures (e.g. *CycleGAN*) would be able to properly train and return satisfactory results, even when applying some data augmentation techniques. Therefore, we extended the dataset and we will later explain the strategy that we applied.

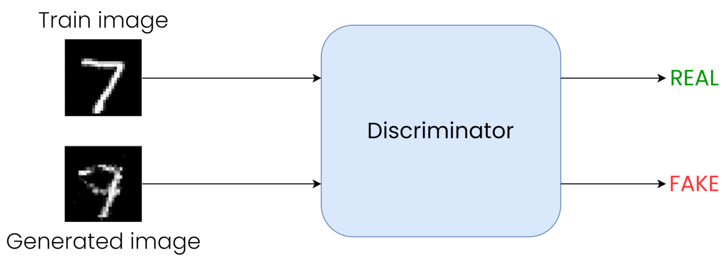
2. Theory

2.1. Vanilla GAN[7]

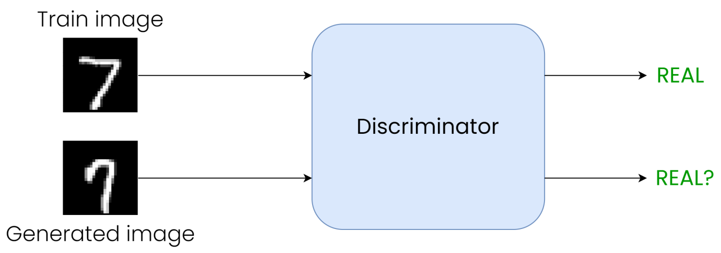
According to the original paper, a GAN consists of two neural networks in competition with each other. These two Neural Networks are called *Generator* and *Discriminator*. The Generator's job is to produce images that are as similar as possible to the training images. The Discriminator's job, instead, is to distinguish between the training images (the real images) and the images created by the Generator (the fake images). Thanks to this deception game the two networks play, they train each other becoming progressively better at their respective jobs.
As can be seen in fig. 1(a), in the very first steps, the images generated by the Generator are of very poor quality and the Discriminator is capable of



(a) Initial stage GAN



(b) Intermediate stage GAN



(c) Final stage GAN

Figure 1: Different stages of GAN training

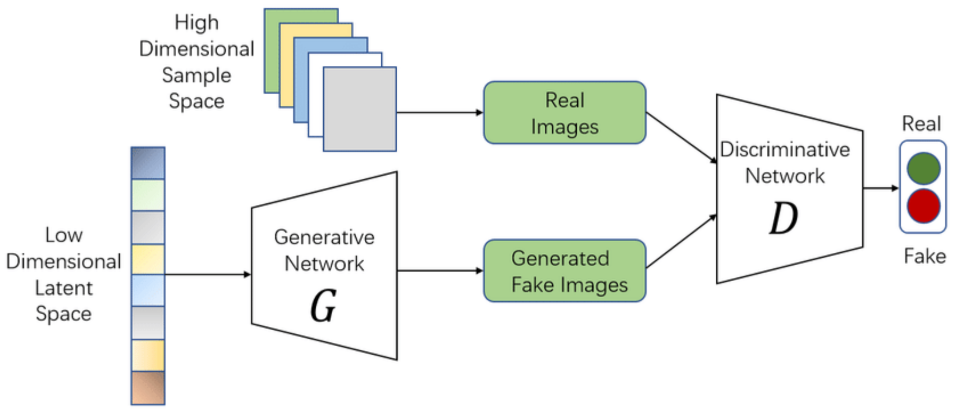


Figure 2: The architecture of vanilla GANs [2]

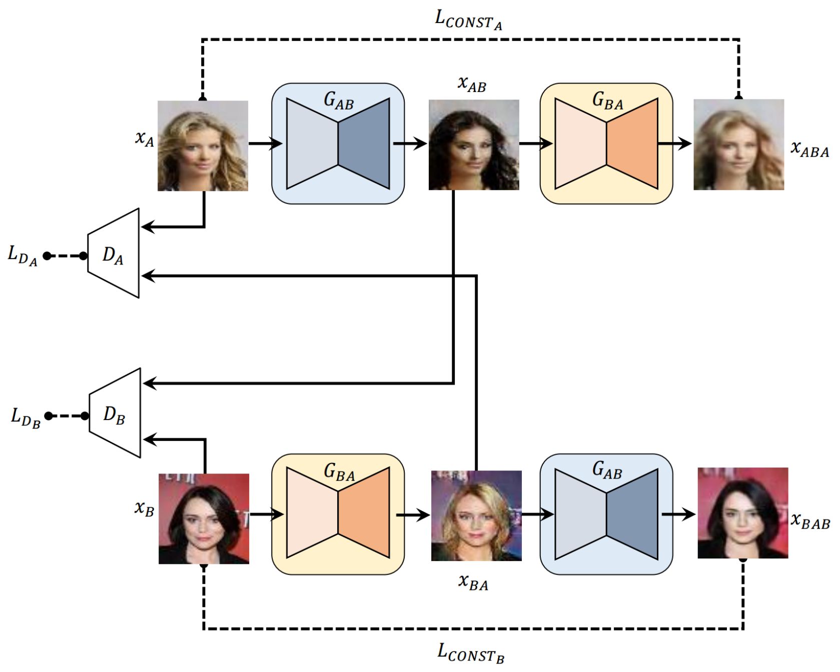


Figure 3: The architecture of unpaired Image-to-Image Translation GANs [11]

easily discerning between real and fake images. As the training progresses (fig. 1(b)), the quality of the generated images increases and the Discriminator's job starts getting more difficult. Finally, if the generator is properly trained, the generated images will be very similar to the real ones, making the distinction between them very difficult for the Discriminator (fig. 1(c)).
We provide an example of the complete architecture in fig. 2.

From a mathematical point of view, in the paper that introduced GANs, the generator tries to minimize the following function while the discriminator tries to maximize it:

$$\mathbb{E}_x[log(D(x))] + \mathbb{E}_z[log(1 - D(G(z)))] \qquad (1)$$

Where:

- $D(x)$: discriminator's estimate of the probability that real data instance x is real.
- $\mathbb{E}_x$: expected value over all real data instances.
- $G(z)$: generator's output when given noise z.
- $D(G(z))$: discriminator's estimate of the probability that a fake instance is real.
- $\mathbb{E}_z$: expected value over all random inputs to the generator (in effect, the expected value over all generated fake instances $G(z)$)

The generator can't directly affect the $log(D(x))$ term in the function, so, for the generator, minimizing the loss is equivalent to minimizing $log(1 - D(G(z)))$. This and more can be found here.

2.2. *Unpaired Image-to-Image Translation GAN [17][11]*

A different type of GAN architecture, is the one where we have two Generators and two Discriminators as well as two different domains. Considering a domain of images A and a domain of images B (where images from both domains share a common characteristic), the goal is to learn something from a domain in order to apply that to images of the other domain. So we have a Generator which generates images of domain B starting from images of domain A and a Generator which generates images of domain A starting from images of domain B. Therefore, we have a Discriminator for the images of domain A and a Discriminator for the images of domain B. (fig. 3)

2.3. *Evaluation Metrics*

The metrics we considered during our experiments are the **Fréchet Inception Distance (FID)** and **Memorization-Informed FID (MiFID)**, which are both state-of-the-art metrics when evaluating GAN generated images. The following explanation of the metrics is taken from the "Evaluation" section of the Kaggle challenge.[1]
In the FID metric, we use a pre-trained Inception network to extract features from an intermediate layer (usually the **Inception v3** network trained on **ImageNet** is used). Then we model the data distribution for these features using a multivariate

Gaussian distribution with mean μ and covariance Σ. The FID between the real images and generated images is computed as:

$$FID = \left\|\mu_r - \mu_g\right\|^2 + Tr(\sum_r + \sum_g -2(\sum_r \sum_g)^{1/2}) \tag{2}$$

where Tr sums up all the diagonal elements.
The MiFid metric, instead, is represented by the minimum cosine distance of all training samples in the feature space, averaged across all user generated image samples. This distance is thresholded, and it's assigned to 1.0 if the distance exceeds a pre-defined epsilon (in our case, we considered the one suggested by Kaggle, which is 10^{-15}). The calculations that are done are the following:

$$d_{ij} = 1 - \cos(f_{gi}, f_{rj}) = 1 - \frac{f_{gi} \cdot f_{rj}}{|f_{gi}| \cdot |f_{rj}|} \tag{3}$$

$$d = \frac{1}{N} \sum_i min_j d_{ij} \tag{4}$$

$$d_{thr} = \begin{cases} d, & \text{if } d < \varepsilon \\ 1, & \text{otherwise} \end{cases} \tag{5}$$

$$MiFID = FID \cdot \frac{1}{d_{thr}} \tag{6}$$

In eq. (3) f_g and f_r represent the generated/real images in feature space (defined in pre-trained networks); and f_{gi} and f_{rj} represent the i^{th} and j^{th} vectors of f_g and f_r, respectively.
Equation (4) is the minimum distance of a generated image when compared to all the real images.
Equation (5) performs the threshold operation.
Finally, eq. (6) applies the memorization term to the FID value.

3. Techniques

For the comparison, we considered six different GAN architectures (and one additional architecture which exploits **Transfer Learning**). The architectures that we considered and the reason why we considered them are the following:

- **DCGAN** and **WGAN** (with Gradient Penalty): because they both represent a good baseline since they are very popular and simple architectures working only with convolutional layers for both Generator and Discriminator;
- **BEGAN** and **iBEGAN**: because they introduce the concept of equilibrium which balances the power of the Discriminator against the Generator and are a more complex version of Vanilla GANs;
- **CycleGAN** and **DiscoGAN**: because they both represent very good architectures for the task of **Unpaired Image-to-Image Translation**;
- **StyleGAN3**: because in this case we don't train the architecture from scratch, but finetune it using a pre-trained model.

3.1. DCGAN [14]

The DCGAN paper is a seminal work for the implementation and theory related to GAN architectures. It is the most basic approach which, as previously said, only considers convolutional layers for both Generator and Discriminator. The loss used by DCGAN is the *Binary Cross Entropy Loss*.

3.2. WGAN with Gradient Penalty [8]

The WGAN with Gradient Penalty replaces the minimax loss function of Vanilla GANs with the Wasserstain Loss and also considers Gradient Penalty in this calculation. Wasserstain loss is preferred over minimax loss because it solves the two main problems of DCGAN that are *mode collapse* (the inability to generate different classes) and *vanishing gradients* (if the gradient becomes too small, the weights won't change and therefore the model won't improve). It is important to note, however, that the WGAN requires more time for training when compared to DCGAN.
The formula of the Wasserstein Loss after considering the Kantorovich-Rubinstein duality principle is the following:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \min_G \max_{D \in \mathcal{D}} \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] - \mathbb{E}_{\tilde{x} \sim \mathbb{P}_g}[D(\tilde{x})]$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions. To enforce the Lipschitz constraint (that is, a function must have a maximum gradient), the original WGAN applied clipping of the weights in order to have them in an interval $[-c, c]$. However, weight clipping introduces different problems, in particular the choice of this interval, if it is not chosen properly leads to bad results. To solve this limitation, the WGAN with Gradient Penalty paper considers that a function f is 1-Lipschitz if has gradient norm 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$ and, instead of applying clipping, it sums to the Wasserstein loss the gradient penalty multiplied to a lambda factor set to 10. This penalizes the model if the gradient norm moves away from the value 1.
The gradient penalty formula is the following:

$$\lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}}[(\left\|\nabla_{\hat{x}} D(\hat{x})\right\|_2 - 1)^2]$$

Where $\hat{x} \sim \mathbb{P}_{\hat{x}}$ refers to random samples.

3.3. *BEGAN [5]*

BEGAN architecture was born following the rise of EBGAN which aims to model the discriminator as an *energy function*: this allows the network to converge more stably and faster, making it more robust to hyper-parameter variations. The key idea of BEGAN is to use an auto-encoder as a discriminator and maintain a balance between the generator and discriminator losses (*L1 loss*). The two losses are considered to be at equilibrium when:

$$\mathbb{E}[\mathcal{L}(x)] = \mathbb{E}[\mathcal{L}(G(z))]$$

We can relax the equilibrium with the introduction of a new hyper-parameter $\gamma \in [0, 1]$ defined as:

$$\gamma = \frac{\mathbb{E}[\mathcal{L}(G(z))]}{\mathbb{E}[\mathcal{L}(x)]}$$

In BEGAN network, the discriminator has two competing goals: auto-encode real images and discriminate real from generated images. The γ term balances these two goals: lower values of γ will lead to lower image diversity because the discriminator focuses more heavily on auto-encoding real images. γ can be referred to as the **diversity ratio**. The BEGAN objective is:

$$\begin{cases} \mathcal{L}_{\mathcal{D}} = \mathcal{L}(x) - k_t \cdot \mathcal{L}(G(z_D)) \\ \mathcal{L}_{\mathcal{G}} = \mathcal{L}(G(z_G)) \\ k_{t+1} = k_t + \lambda_k(\gamma\mathcal{L}(x) - \mathcal{L}(G(z_G))) \end{cases}$$

To maintain the equilibrium $\mathbb{E}[\mathcal{L}(G(z))] = \gamma\mathbb{E}[\mathcal{L}(x)]$ a new variable $k_t \in [0, 1]$ is defined which controls how much emphasis is put on $\mathcal{L}(G(z_D))$ during gradient descent. It is initialized at 0 ($k_0 = 0$) and λ_k is the learning rate for k.
Finally, thanks to the equilibrium concept we can frame the convergence process as finding the closest reconstruction $\mathcal{L}(x)$ with the lowest absolute value of the instantaneous process error for the proportion control algorithm $\big|\gamma\mathcal{L}(x) - \mathcal{L}(G(z_G))\big|$. This measure is formulated as the sum of these two terms:

$$M_{global} = \mathcal{L}(x) + \big|\gamma\mathcal{L}(x) - \mathcal{L}(G(z_G))\big|$$

This measure is used in the BEGAN architecture to determine when the network has reached its final state or if the model has collapsed.

3.4. *iBEGAN [12]*

The iBEGAN architecture, as the name states (*Improved BEGAN*), tries to improve the original formulation of BEGAN architecture.
In particular, it has been proved that by training a BEGAN model at low values of γ the generated images looked uniform with many noise-like regions, while at high values ($\gamma = 0.7$), the diversity of the generated images increased but the quality declined. Another shortage of BEGANs is that the generator cannot learn the low-probability features, even with the highest value of *diversity ratio*. The iBEGAN tackles the above two issues with two changes to the BEGAN architecture:

- The **Batch Normalization** technique is added after each convolutional layer both in the discriminator and generator in order to improve diversity of the generated images;
- A **denoising loss** is added to the Discriminator in order reduce noise-like regions in the generated images.

3.5. *CycleGAN [17]*

The CycleGAN represents one of the main techniques used for Unpaired Image-to-Image Translation. The training idea is the following:

> *Our goal is to learn a mapping $G : X \rightarrow Y$ such that the distribution of images from $G(X)$ is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained, we couple it with an inverse mapping $F : Y \rightarrow X$ and introduce a cycle consistency loss to enforce $F(G(X)) \approx X$ (and viceversa)*

The CycleGAN mainly makes use of two losses:

- **Adversarial Loss**: which is the same concept as the minimax loss, but applied to both pairs of generators and discriminators (so generator that translates the images to domain X and discriminator for domain X);
- **Cycle Consistency Loss**: which encourages the image translation cycle to revert images to the original state;

In particular, the formula for the Cycle Consistency Loss (which is the main loss in this kind of architecture) is the following:

$$\begin{aligned} L_{cyc}(G, F) = E_{x\sim p_{data}(x)}[\big\|F(G(x)) - x\big\|_1] \\ + E_{y\sim p_{data}(y)}[\big\|G(F(y)) - y\big\|_1] \end{aligned}$$

Where $z \sim p_{data}(z)$ represents the data distribution for x and y and G and F refer to the generators as we have cited previously.

3.6. *DiscoGAN [11]*

The DiscoGAN follows the same core principles as CycleGAN. The main difference between DiscoGAN and CycleGAN (aside from the architectures of Generator and Discriminator), is that DiscoGAN makes use of a Reconstruction Loss rather than a Cycle Consistency Loss. The Reconstruction Loss measures how well the original input is reconstructed after a sequence of two generations. Different possible measures are proposed in the paper as possible Reconstruction Losses: MSE, cosine distance and hinge-loss.

3.7. *StyleGAN3 [10]*

The Style Generative Adversarial Network, or StyleGAN for short, is an addition to the GAN architecture that introduces significant modifications to the Generator model. StyleGAN produces the simulated image sequentially, originating from a simple resolution and enlarging to a huge resolution (up to 1024×1024). We won't go into detail in the StyleGAN3 architecture, the key idea is that by transforming the input of each level individually, it examines the visual features that are manifested in that level, from standard features (e.g. pose, face shape) to minute details (e.g. hair color), without altering other levels. We used a StyleGAN3 checkpoint pre-trained on the LHQ dataset for 25 Mimg, capable of generating 256×256 photorealistic landscapes [9]. We then fine-tuned said model by freezing the last 10 layers of the StyleGAN architecture for 100 kimg using as training set the 300 Monet paintings provided by the competition.

4. Experiments

4.1. *Dataset Preprocessing*

As we mentioned in section 1, one of the problems that we encountered almost immediately in our project was the lack of data. Because of how limited the training set was (it consisted of 300 Monet paintings), most models would underfit by using only the provided training images.
To overcome this problem, we considered another dataset "Best Artworks of all Time" [3] and we extracted from this dataset all Impressionist and Post-Impressionist paintings, together with five random paintings made by artists of other artistic movements *previous* to Monet (that is, artists which date of birth was previous to the date of Monet's death). The assumption behind this strategy is that Monet was influenced by previous artists, and so will our trained model.
To avoid the model fitting with unrelated paintings and to enhance the importance of Monet and Impressionist paintings, we doubled their number by performing data augmentation via the *TrivialAugmentWide* [13] operation. The final amount of training data that we obtained was 6755, which was still small for most GAN architectures but also better when compared to what we started with.

4.2. *GAN architectures implementation*

We implemented from scratch the GAN architectures in Pytorch (mostly to have more control over the training phase) and we followed exactly the guidelines provided by the paper associated to each GAN for hyperparameters, loss functions and Generators and Discriminators architectures. We only modified parameters such as batch size or number of training epochs, which are not inherent to the architectures but related to their training (we modified them to fit our experiments' needs, as we will see in the next section).

4.3. *Experimental Protocols*

As first experiment, we trained our models from scratch with the paper suggested hyperparameters (except for the batch size of the CycleGan for which we used 64 instead of 1 to reduce computational effort) considering an image size of 64×64 and 200 epochs. The dataset used for training is our extended dataset, and, for domain B of unpaired image-to-image translation architectures, the landscape photos provided by the competition are used as dataset. The number of epochs is lowered by a lot when compared to the suggested one for some architectures (e.g. the WGAN with Gradient Penalty suggests $200k$ epochs to obtain satisfactory results), but the computational complexity and the time required by the task were too heavy to perform a complete optimal training for all architectures. Therefore, we decided to compare their performance on this reduced number of epochs. In the second experiment, we considered an image size of 256×256 and trained CycleGAN and DiscoGAN from scratch but fine-tuned StyleGAN3. We used the Kaggle competition dataset without any data augmentation for StyleGAN3 and CycleGAN architectures, and our extended dataset for the DiscoGAN. For the latter, we considered this approach because experiments with DiscoGAN using only the competition dataset and a batch size of 1 (since the training data was heavily reduced) returned poor results. In this case, for unpaired image-to-image translation architectures we considered the already mentioned dataset as dataset of domain B. We trained CycleGAN and DiscoGAN for 200 epochs and fine-tuned StyleGAN3 for 100 epochs. Refer to fig. 3 to see some of the images generated by our architectures in the test phase. Furthermore, results that we obtained both during training (e.g. Generator or Discriminator losses) and during testing (e.g. FID and MiFID values) can be found at the following link.

4.4. Experiments tests and results

We performed the testing phase using FID and MiFID metric (already presented in section 2). Both FID and MiFID compare extracted features from real images and extracted features from the generated ones: because of this, we need to define the set of real images.
The Kaggle competition uses a set which is not publicly provided. Because of this, the values that we obtained for the metrics, and that we will present later in tabular form, are not comparable to the ones that can be found in the Kaggle leaderboard. We considered two different set of Monet paintings as set of real images:

- Monet paintings provided as train set by the competition (300 paintings);
- Monet paintings extracted from the wikiart dataset (1369 paintings) [4].

Furthermore, since the competition required the generation of 10000 paintings but it only provided 7038 real landscape photos, we used a subset of the landscape dataset that was used to train from scratch StyleGAN3 [16] as images of the domain B for the test phase of unpaired image-to-image translation architectures (CycleGAN and DiscoGAN). When considering the 64 × 64 experiment, the best quantitative results were given by WGAN with Gradient Penalty and DCGAN, but from a qualitative point of view, the results generated by CycleGAN were slightly better. We believe this is due to the fact that we modified the batch size from the original one suggested by the paper.
The really interesting results were given by the 256 × 256 experiments. The StyleGAN performed much better than the rest, both from a qualitative and quantitative perspective.

GAN Architecture	*FID*	*MiFID*
Style 256	**73.397**	**73.397**
Cycle 256	115.119	115.119
Disco 256	177.463	177.463
Wgan GP 64	183.116	183.116
Cycle 64	195.057	195.057
DCgan 64	207.782	207.782
Disco 64	219.252	219.252
Began 64	285.631	285.631
iBegan 64	334.62	334.62

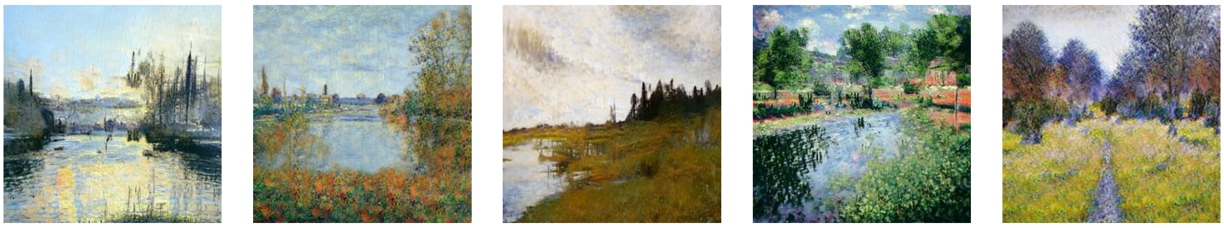
Table 1: Results obtained considering as set of real images the train set of the kaggle competition

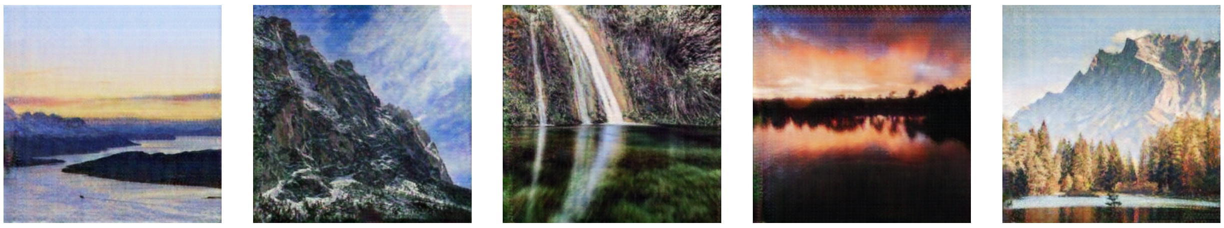
GAN Architecture	*FID*	*MiFID*
Style 256	**47.415**	**47.415**
Cycle 256	89.281	89.281
Disco 256	117.351	117.351
Wgan GP 64	150.207	150.207
DCgan 64	162.639	162.639
Cycle 64	164.714	164.714
Disco 64	181.912	181.912
Began 64	269.822	269.822
iBegan 64	316.749	316.749

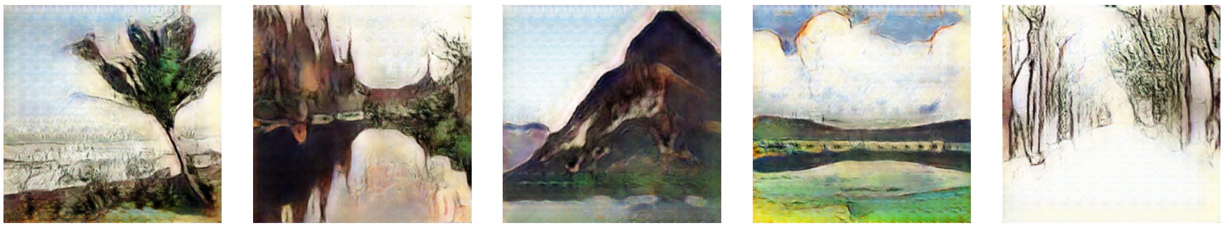
Table 2: Results obtained considering as set of real images Monet paintings extracted from WikiART

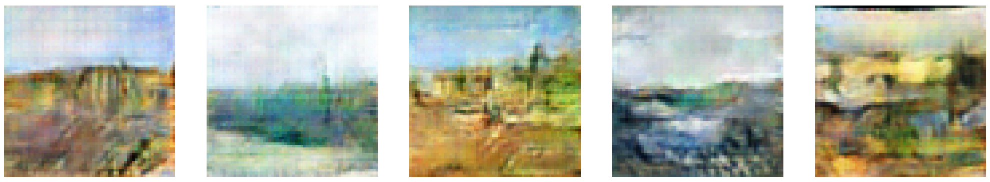
5. Conclusion

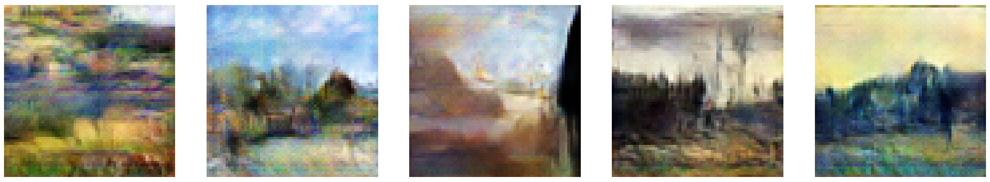
We approached the problem of generating Monet paintings. We first introduced the theory behind two different kinds of GANs, that are vanilla GANs and unpaired image-to-image translation GANs. We briefly introduced the main concepts behind each GAN that we used to tackle the issue (DCGAN, WGAN with Gradient Penalty, BEGAN, iBEGAN, CycleGAN, DiscoGAN and StyleGAN3). We explained and described how we pre-processed the datasets that we used and the experiments which we carried out using the proposed architectures. We compared the obtained results, both from a quantitative perspective (that is with the results obtained from the FID and the MiFID metric) and from a qualitative one (providing some sample images that we generated using our trained architectures).
We can conclude that the best approaches were StyleGAN and CycleGAN, which are expected results since StyleGAN3 exploits transfer learning by using a pre-trained model, and CycleGAN because it didn't require as much data and as much epochs as the other architectures.
Future works could further improve this project by training the architectures that performed better for more epochs.

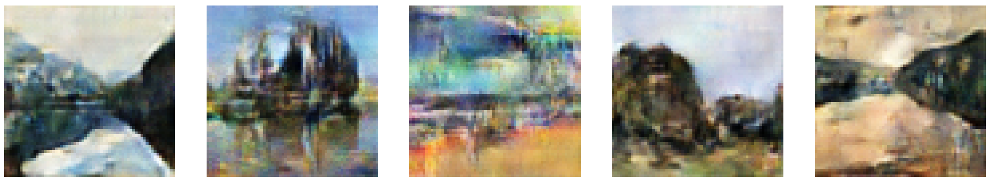

(a) Stylegan3 256x256 generated images

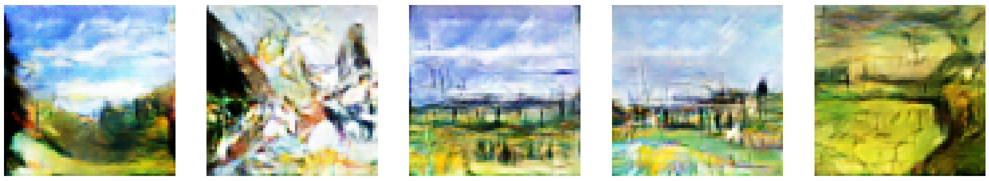

(b) CycleGAN 256x256 generated images

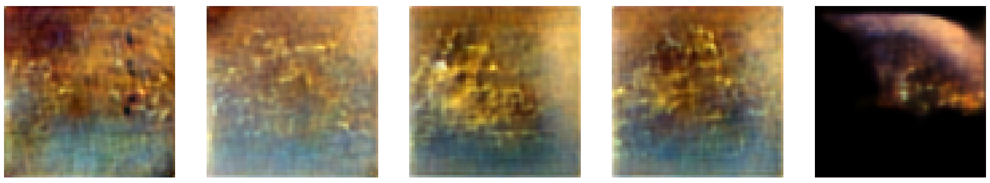

(c) DiscoGAN 256x256 generated images


(d) WGAN 64x64 generated images


(e) DCGAN 64x64 generated images


(f) CycleGAN 64x64 generated images


(g) DiscoGAN 64x64 generated images


(h) BEGAN 64x64 generated images

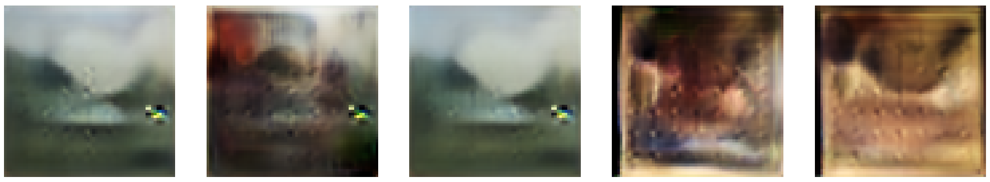

(i) IBEGAN 64x64 generated images

Figure 3: Five samples generated for each architecture

References

[1]https://www.kaggle.com/competitions/gan-getting-started/overview.

[2]https://www.researchgate.net/figure/The-architecture-of-vanilla-GANs_fig1_340458845.

[3]https://www.kaggle.com/datasets/ikarus777/best-artworks-of-all-time.

[4]https://www.kaggle.com/datasets/varnez/claude-monet-pictorial-works-dataset-wikiart.

[5]D. Berthelot, T. Schumm, and L. Metz, *Began: boundary equilibrium generative adversarial networks*, 2017, 10.48550/ARXIV.1703.10717.

[6]G. Cohn, (2018) https://www.nytimes.com/2018/10/22/arts/design/christies-art-artificial-intelligence-obvious.html.

[7]I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial networks*, 2014, 10.48550/ARXIV.1406.2661.

[8]I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, *Improved training of wasserstein gans*, 2017, 10.48550/ARXIV.1704.00028.

[9]L. Justin Pinkney, https://twitter.com/Buntworthy/status/1460980442409185287?s=20.

[10]T. Karras, M. Aittala, S. Laine, E. Härkönen, J. Hellsten, J. Lehtinen, and T. Aila, «Alias-free generative adversarial networks», in Proc. neurips (2021).

[11]T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, *Learning to discover cross-domain relations with generative adversarial networks*, 2017, 10.48550/ARXIV.1703.05192.

[12]Y. Li, N. Xiao, and W. Ouyang, «Improved boundary equilibrium generative adversarial networks», IEEE Access **6**, 11342–11348 (2018) 10.1109/ACCESS.2018.2804278.

[13]S. G. Müller and F. Hutter, *Trivialaugment: tuning-free yet state-of-the-art data augmentation*, 2021, 10.48550/ARXIV.2103.10158.

[14]A. Radford, L. Metz, and S. Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*, 2015, 10.48550/ARXIV.1511.06434.

[15]I. Salian, (2020) https://blogs.nvidia.com/blog/2020/12/07/neurips-research-limited-data-gan/.

[16]I. Skorokhodov, G. Sotnikov, and M. Elhoseiny, «Aligning latent and image spaces to connect the unconnectable», arXiv preprint arXiv:2104.06954 (2021).

[17]J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, «Unpaired image-to-image translation using cycle-consistent adversarial networks», in Computer vision (iccv), 2017 ieee international conference on (2017).