Generating Lissajous Figures using Generative Adversarial Networks: An Empirical Study

Duccio Meconcelli

2023

Abstract

Generative Adversarial Networks (GANs) are a type of deep learning neural network architecture used for generative tasks, such as creating new images, music, or text. Lissajous figures are parametric curves that are commonly used in physics and engineering to represent simple harmonic motion. The goal of this study was to use a GAN to generate Lissajous figures. However, this task was challenging due to the oscillatory nature of GANs. Despite various attempts to modify the parameters and layers of the network, the results obtained were not satisfactory. The best-performing network still struggled to accurately imitate Lissajous figures. This highlights the difficulties in using GANs for generating complex patterns.

Contents

1 Introduction 3

2 Dataset 4
- 2.1 Create Lissajous Figures . . . 4
- 2.2 Size . . . 4
- 2.3 The DPI . . . 4
- 2.4 Transformation and Splitting . . . 4

3 Methods and Metrics 6
- 3.1 Equipment and starting point . . . 6
- 3.2 Metrics . . . 6
  - 3.2.1 Generator Metrics . . . 6
  - 3.2.2 Discriminator Metrics . . . 6
  - 3.2.3 Exception . . . 7
  - 3.2.4 Human Evaluation . . . 7
- 3.3 Methods . . . 7

4 Results 8

5 Conclusion 13

1 Introduction

Lissajous figures are graphical representations of the superposition of two simple harmonic waves. They are defined by equations that describe the x and y coordinates as a function of time. The appearance of the figures can be controlled by changing the frequency, phase, and amplitude of the two waves [3].
Generative Adversarial Networks (GANs) are deep neural networks designed for generative tasks. They consist of two main components: a generator and a discriminator [1]. The generator is trained to produce synthetic data that is similar to the training data, while the discriminator is trained to distinguish between the generated data and the real training data. The generator and discriminator are trained in an adversarial manner, with the generator attempting to produce data that can fool the discriminator, and the discriminator trying to correctly identify the source of the data.
The goal of this project is to create a GAN that is capable of generating Lissajous figures. This involves training the generator to produce synthetic Lissajous figures that are similar to real Lissajous figures, and training the discriminator to distinguish between real and synthetic Lissajous figures.
Generating Lissajous figures is not a straightforward task due to their great variability. In this study, a significant challenge was quantifying the quality of the generated Lissajous figures. Unlike a classification problem, where the network can be evaluated based on its accuracy in categorizing the data, it is much more difficult to obtain precise metrics for the evaluation of generated figures. To overcome this challenge, we used a combination of qualitative and quantitative methods to assess the quality of the generated figures, including visual comparison and mathematical analysis of the generated figures.
In this study, we also experimented with different parameters and architectures of the GAN in order to improve the quality of the generated figures. This allowed us to gain insights into the optimal parameters and architectures for generating Lissajous figures using GANs.

Additionally, we used a specialized dataset of Lissajous figures to train the network, allowing us to control for the variability of the input data and to better understand the impact of the network's parameters on the quality of the generated figures.

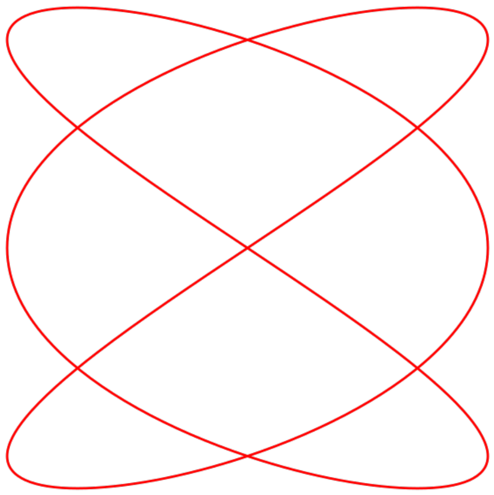


Figure 1: The most common Lissajous figures, from Wikipedia [3]

2 Dataset

2.1 Create Lissajous Figures

To create the Lissajous figures dataset for this study, we started from a Python script obtained from Github [5]. The creation of the images was made using the overlapping of multiple pendulums from 2 to 4 following two functions:

$$x = \sum_{i=2}^{4} d \cdot \left(\alpha_i \cdot \sin\left(t \cdot \mathrm{f}_i^x + \mathrm{p}_i^x\right)\right)$$

$$y = \sum_{i=2}^{4} d \cdot \left(\alpha_i \cdot \sin\left(t \cdot \mathrm{f}_i^y + \mathrm{p}_i^y\right)\right)$$

Where i varies from 2 to 4 and represents the pendulums while d, α, f_i^x and p_i^x are randomly generated factors like phase and amplitude of the pendulums.

2.2 Size

Initially, the dataset consisted of 15'000 total images, but after preliminary experiments, it became evident that the dataset was heavily biased towards circular figures that covered the majority of the image (2).

To address this issue, we recreated the dataset to a total of 20'000 images, both 128 x 128 pixels and 256 x 256 pixels. This was accomplished by removing unsuitable images and selecting the best images to ensure a balanced distribution of Lissajous figures. Moreover, by using a larger dataset of 20'000 images, we were able to better train the network and to generate more accurate and detailed figures. This was particularly important because the complexity of Lissajous figures can vary widely, and a larger dataset allowed us to capture this variability.

(a) 15'000 dataset (b) 20'000 dataset

Figure 2: Differences between 15'000 dataset (a) and 20'000 dataset (b)

2.3 The DPI

Another factor to consider when generating the images for the dataset was the dpi (dots per inch). A high dpi can darken the image too much, while a low dpi can make it too light and nearly invisible (3). After several tests during the reconstruction of the dataset after the preliminary trials, a dpi of 50 was chosen for the 128 x 128 images and a dpi of 40 was chosen for the 256 x 256 images. This helped to strike a balance between image clarity and visibility, which was crucial for training the network accurately and generating high-quality Lissajous figures.

2.4 Transformation and Splitting

80% of the dataset was allocated for training, 10% for testing, and the remaining 10% for validation. The images were then normalized between -1 and 1 and with a 50% probability, they were horizontally mirrored. This was done to increase the variability of the input data and to make the network more robust to changes in the orientation of the figures.

The order of the images in the dataset was randomized to ensure that the network was not biased towards any specific type of Lissajous figure. This allowed us to train the network on a diverse set of Lissajous figures and to generate new, diverse figures using the network.

In conclusion, the creation of a specialized dataset of Lissajous figures, combined with normalization and data augmentation, allowed us to control for the variability of the input data and to better understand the impact of the network's parameters on the quality of the generated figures.

(a) 10 DPI

(b) 60 DPI

Figure 3: Differences between 10 DPI (a) and 60 DPI (b)

Additionally, it is important to note that the final dataset was diverse in terms of both the type of Lissajous figures and their orientation, including circular, elliptical, and more complex figures. This ensured that the network was exposed to a variety of different figures and allowed us to test its ability to generate new, diverse figures that were not included in the training set. The use of a specialized dataset of Lissajous figures allowed us to better control for the variability of the input data and to better understand the impact of the network's parameters on the quality of the generated figures.

In conclusion, the creation of a specialized, diverse, and large dataset of Lissajous figures was an essential step in training the GAN and achieving accurate and diverse results.

3 Methods and Metrics

3.1 Equipment and starting point

The creation of the Neural Network was implemented using Python and the Pytorch library. As a starting code we used the one present in the yandexre Github repo ([7]) , obviously the code was changed heavily and served mostly as a starting point to set up the project and the environment. The dataset was taken using the code as already discussed in Chapter 2. A home machine with an RTX 3060 graphics card was used to run the code and later Google Colab was also used to run experiments in parallel. Most of the experiments will focus on 128x128 images, this is mainly due to the onerousness of carrying out repeated experiments with 256x256 images.

It is also true some experiments with images using the 256 x 256 pixel dataset will be carried out. The basic model of the Network was chosen inspired by a DCGAN [11]: by placing side by side convolutional/convolutional layers transposed to batch norm layers, and LeakyReLU activations.

3.2 Metrics

In order to evaluate the performance of the GAN, several metrics were used. For the generator, the Structural Similarity Index (SSIM), Peak Signal-to-Noise Ratio (PSNR), and Sliced Wasserstein Distance (SWD) were used to quantify the similarity between the generated and real Lissajous figures.

3.2.1 Generator Metrics

1. The structural similarity index (SSIM) is a widely used quality index for evaluating the similarity between two images.
$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{1}$$
where x and y are the two images being compared, μ_x and μ_y are the means of x and y, σ_x^2 and σ_y^2 are the variances of x and y, and σ_{xy} is the covariance between x and y. The constants c_1 and c_2 are used to stabilize the division with weak denominator. The implementation was taken by a Python Library [6] Optimized to speed up calculations.

2. The peak signal-to-noise ratio (PSNR) is another common quality index for comparing two images. It measures the average difference between the pixel values of the two images. It is defined as:
$$PSNR = 10\log_{10}\left(\frac{1}{MSE}\right) \tag{2}$$
where MSE is the mean squared error between the two images.

3. The sliced wasserstein distances (SWD) is a variation of the wasserstein distances, And it is another widely used metric to compare the similarity between 2 images. The implementation was taken by a Github Repository ([2])

3.2.2 Discriminator Metrics

For the discriminator, accuracy, recall, precision, and f1-score were used to measure the performance of the discriminator in identifying real and generated Lissajous figures [1].

Accuracy measures the fraction of correctly classified instances, recall measures the fraction of positive instances that were correctly identified, precision measures the fraction of positive predictions that were correct, and F1-score is the harmonic mean of precision and recall. These metrics are defined as:

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Samples} \tag{3}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{4}$$

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{5}$$

$$\text{F1-score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6}$$

[1]**The metrics used for the discriminator were largely ignored as they provided inconclusive results.**

3.2.3 Exception

Commonly used metrics for GAN evaluation, such as the Fréchet Inception Distance (FID) and Inception Score (IS), were not applicable in this study. This is due to the fact that they are based on an implementation of a neural network, Inception, that was created with 3 channels while the dataset used in this study consists of single channel images. Despite efforts to adapt these metrics to the single channel case, the results were not conclusive.

3.2.4 Human Evaluation

While these metrics provided quantitative evaluations of the generated figures, the primary factor in determining the success of the GAN was subjective human evaluation. The generated figures that were deemed to have the most visually appealing and accurate Lissajous-like patterns were selected.
The focus of the evaluation was on the ability of the generator to produce visually convincing Lissajous figures.

3.3 Methods

The first precautions were carried out before starting the actual experimentation. In fact, in the preliminary phase, changes were made to the basic model of the DCGAN such as:

- modify the noise from a sampling of elements of a linear distribution to a superimposition of multiple Gaussian distributions [4]
- use the dropout at each layer of the model Generator to increase the noise and make the model more general as well as lighten the computation [4]
- use SGD for discriminator and ADAM for generator [4]

The experiments were carried out in order to meet the goal of improving upon a baseline DCGAN network consisting for the generator 1 Linear layer and 4 layers of convolutional transpose, and for the discriminator 5 convolutional layer and 2 Linear layer, no gradient penalty, a learning rate of 0.0002 for both the generator and the discriminator over 100 epochs with a seed set to 42 for reproducibility. The approach taken to achieve this goal involved experimenting with different network architecture modifications suggested by various papers. The changes include:

- adding layers for both the generator and the discriminator (hypothesis: adding more complexity/Deeper Layers produces more complex results)
- changing the type of the Generator Layers (Upsample with Convolutional 2D Layers or Convolutional 2D Transpose Layers)
- changing the loss function between BCE (standard for GAN) or Wasserstein Loss ([10])
- altering the learning rate for both the generator and the discriminator
- implementing or removing the gradient penalty ([9])
- modifying the batch size
- adjusting the relative training time for the discriminator compared to the generator (Increase or decrease the number of steps so that the network updates the generator parameters [4])

Each experiment was performed in a manner such that if after 100 epochs the modified network was unable to produce images that were better than those generated by the baseline network, the experiment was discontinued. However, if a network achieved improved results, it was selected as the new comparison network in place of the baseline.

It should be noted that the oscillatory nature of GANs means that this approach may not always be reliable, as it is possible for a modified network to improve significantly after 100 epochs but then perform worse than the baseline network after further training. Despite this limitation, the described experimental process was followed due to time constraints.

4 Results

In this section, we present the results obtained from a series of experiments carried out on our Generative Adversarial Network (GAN) designed to generate Lissajous figures. In order to optimize the performance of the network we conducted 20 experiments by modifying various parameters such as network architecture, loss function, learning rate, gradient penalty, batch size. These modifications were carried out with the goal of improving the generated images compared to the original network architecture.
The following paragraphs will provide a detailed discussion of the findings from these experiments. The experiments were performed using 128x128 images and were trained for 100 epochs. Table 1 shows the complete list of all the architectures of the experiments. The order of the experiments may seem confusing but a parallel approach was used to carry out the experiments so in order to optimize the times several experiments were carried out simultaneously thus mixing the various changes made to each new result. The variation of the learning rate did not lead to noteworthy results.

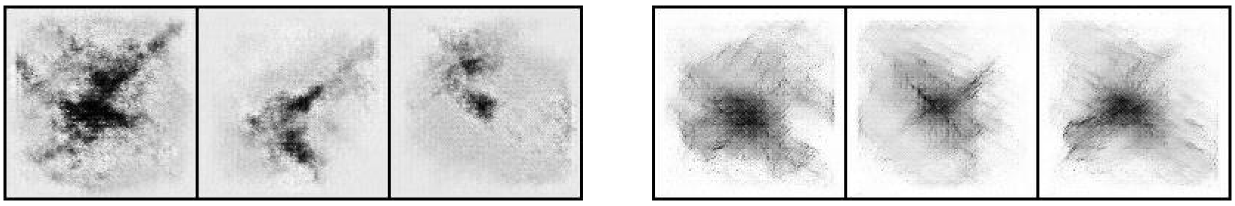

(a) CLEAN (b) experiment 0

Figure 4: Differences between the base DCGAN 9(a) and the first experiment(b)

The results of our experiments show several key insights regarding the development of our Generative Adversarial Network (GAN) for generating Lissajous figures.

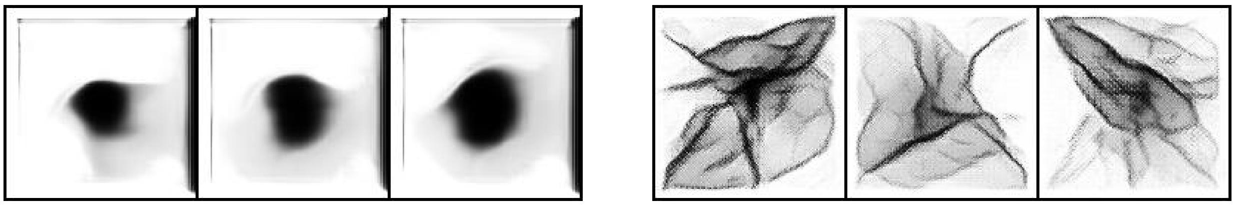
1. The use of upsample and conv2d instead of conv2d transpose significantly reduced the quality of the generated images (Figure 5).
2. A reduction in batch size improved the training efficiency, however, it also increased the overall execution time (banale spiegazione che la rete viene addestrata maggiormente) (Figure 6).
3. The introduction of a gradient penalty, although it increased the execution time, had a positive effect on the final image quality (Figure 7).
4. The results indicate that training the discriminator more heavily than the generator leads to better quality images. (Figure 8).
5. It is not always the case that increasing the depth of the generator produces better results (Figure 10).
6. Our results showed that there was no difference between using the Wasserstein Loss and the Binary Cross Entropy (BCE) Loss (Figure 11).

   - Mixed Result: adding additional convolutional layers to the discriminator led to more complex images, with more intricate details but a deeper investigation is needed (Figure 9)

The results obtained from the experiments conducted on the GAN network aimed at creating Lissajous figures showed a mixture of outcomes. Despite using well-established metrics such as SSIM, SWD, and PSNR to evaluate the quality of the generated images, the results obtained often contrasted with human perception, for example using the SSIM metric we should have chosen 2 as the best image Which is very far from resembling a Lissajous figure. Instead, following the SWD metric and taking the model with the lowest value, we obtain model 17 which is in any case a good compromise between all the possible experiments carried out (Figure 13).

This highlights the difficulty in comparing and analyzing GANs and underscores the need for further research in this area as already discussed in some papers [8] [12]. Additionally, while adding layers to the discriminator resulted in increased complexity, the resulting images were often similar to those with fewer layers, potentially due to the wave-like nature of GANs. This further emphasizes the complexity of the GANs and the need for more in-depth analysis to better understand the relationships between the different components of GANs and the resulting images.

Additionally, 3/4 experiments were also tried with 256x256 images, but as the size of the network was scaled to increase the depth of the layers, the calculation time required for the test phase increased exponentially, making it prohibitive to perform these tests at this moment. Having said that, it should however be noted that the difference between 256-0 and 256-1 of the architecture has led to an increase in complexity in the final image all the same (Figure 12). This is further confirmation of the fact that increasing the depth of the discriminator affects the creation of more complex geometries even if this does not translate directly into an image more similar to those of the dataset.

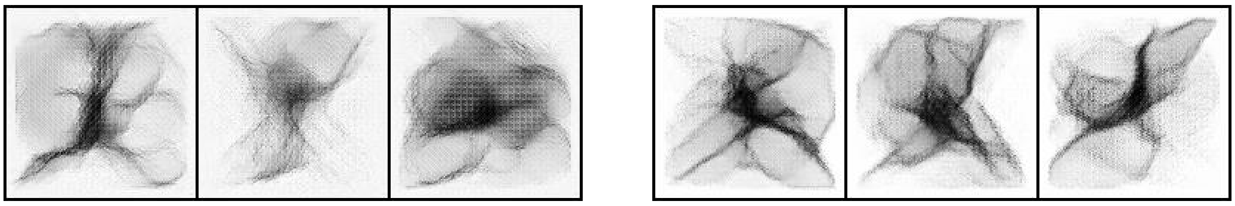

(a) experiment 14 with Upsample + conv2d

(b) experiment 17 Conv2d Transpose

Figure 5: Upsample and Convolution Layer (a) VS Convolution Trasnpose Layer (b)

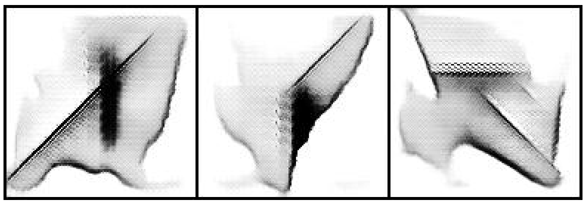
Note: It should be noted that each experiment lasted between 2/2:30 hours using the GPU and therefore on about 22 experiments it was necessary to use several computers in parallel, this was possible thanks to the use of several sessions on google colab and in parallel also the use of the home computer.

Our experiments have shown that, despite the challenges in comparing and analyzing GANs, the best model according to the metrics described in Chapter 3 was the one from Experiment 22 (Figure 14). While the quality of the images produced is not the best, we can still see some shapes, but far from the harmony and geometry of the Lissajous figures. The PSNR, SWD, and SSIM values obtained during the testing phase also concur in saying that Model 22 is acceptable from a purely numerical point of view.
However, it should be noted that all the tests were carried out over 100 epochs, which is a relatively small number compared to the standard of GANs, and this could be one of the reasons why the images are not very good.


(a) experiment 7 with 80 batch size

(b) experiment 6 with 50 batch size

Figure 6: Batch Size differences


(a) experiment 10 with 0 gradient penalty

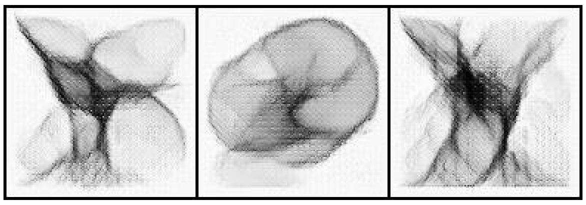
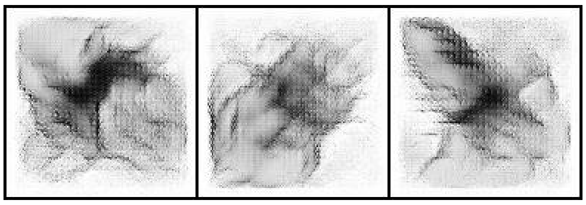

(b) experiment 9 with 2 gradient penalty

Figure 7: Gradient Penalty Differences


(a) experiment 15 with Disc. Step 2

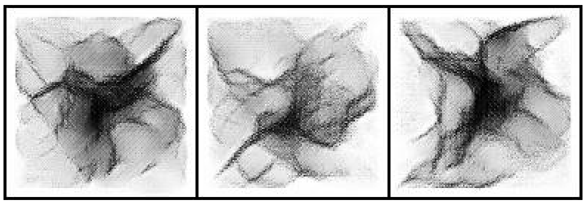
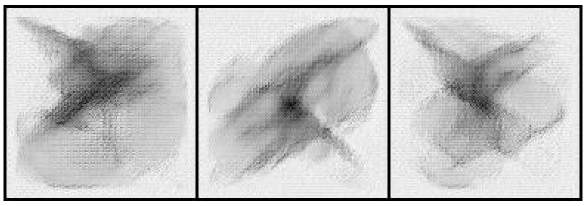

(b) experiment 16 with Disc. Step 4

Figure 8


(a) experiment 1 with 6 Conv2d

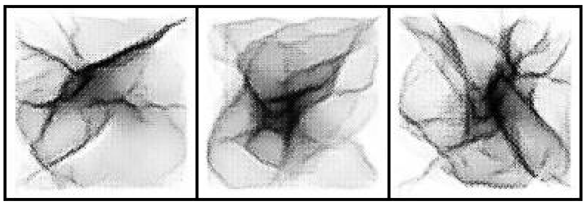
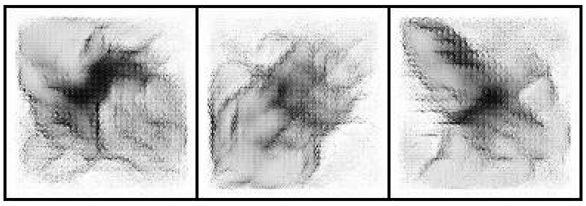

(b) experiment 21 with 10 Conv2d

Figure 9: Discriminator (a) with less layer and (b) with more layer


(a) experiment 15 with 5 Conv2d Transpose

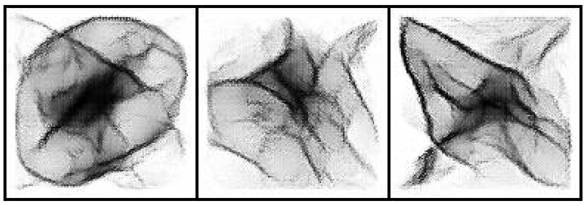
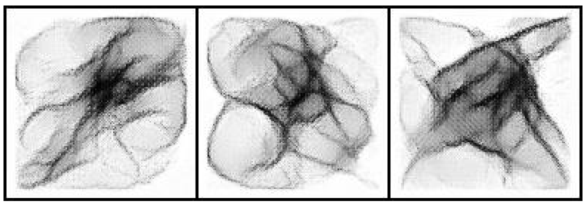

(b) experiment 18 with 4 Conv2d Transpose

Figure 10: Generator (a) with more layer and (b) with less layer


(a) experiment 20 with Wasserstein Loss

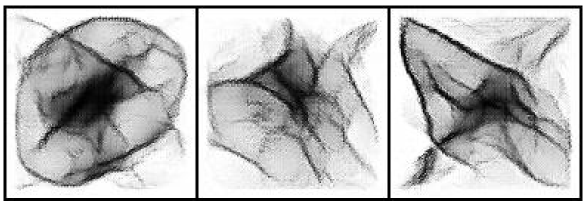
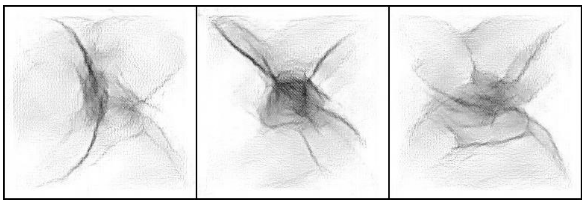

(b) experiment 18 with BCE loss

Figure 11: No notable difference between the two Loss


(a) experiment 256-0 with 5Conv2d+2Lin

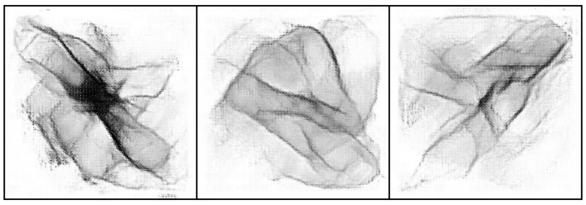
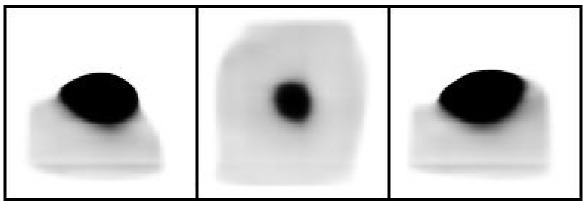
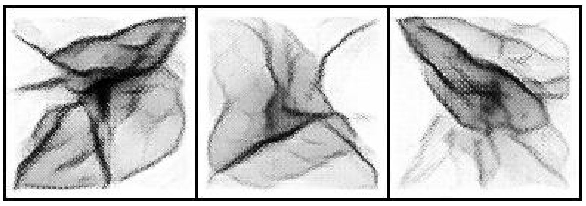

(b) experiment 256-1 with 7Conv2d+2Lin

Figure 12: Differences between 2 256x256 pixel images with different architecture


(a) experiment 2


(b) experiment 17

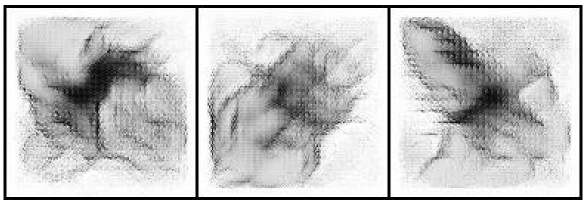

(c) experiment 15

Figure 13: (a) best for SSIM, (b) best for SWD, (c) best for PSNR

Number	batch size	Layer Disc.	Layer Gen	Disc. Step	Penalty	Loss
CLEAN	100	5Conv2d+2Lin	1Lin+4ConvTrans2d	1	0	BCE
0	80	6Conv2d+2Lin	1Lin+4ConvTrans2d	3	1.5	BCE
1	80	6Conv2d+2Lin	1Lin+4ConvTrans2d	2	1.5	BCE
2	100	6Conv2d+2Lin	1Lin+4Conv2d+4Upsample	2	1.5	BCE
3	100	6Conv2d+2Lin	1Lin+4Conv2d+4Upsample	2	1.5	BCE
4	100	6Conv2d+2Lin	1Lin+4Conv2d+4Upsample	2	1.5	BCE
5	100	5Conv2d+2Lin	1Lin+4ConvTrans2d	2	0	BCE
6	50	5Conv2d+2Lin	1Lin+4ConvTrans2d	2	1.5	BCE
7	80	5Conv2d+2Lin	1Lin+4ConvTrans2d	3	1.5	BCE
8	80	6Conv2d+2Lin	1Lin+4ConvTrans2d	3	1.6	BCE
9	64	5Conv2d+2Lin	1Lin+4ConvTrans2d	2	2.0	BCE
10	64	5Conv2d+2Lin	1Lin+4ConvTrans2d	2	0	BCE
11	80	7Conv2d+2Lin	1Lin+5ConvTrans2d	2	1.6	BCE
12	80	7Conv2d+2Lin	1Lin+4ConvTrans2d	2	1.5	BCE
13	80	8Conv2d+2Lin	1Lin+5ConvTrans2d	3	1.75	BCE
14	80	8Conv2d+2Lin	1Lin+4Upsample+4Conv2d	4	1.75	BCE
15	80	8Conv2d+2Lin	1Lin+5ConvTrans2d	2	1.7	BCE
16	80	8Conv2d+2Lin	1Lin+5ConvTrans2d	4	1.75	BCE
17	80	8Conv2d+2Lin	1Lin+4ConvTrans2d	4	1.6	BCE
18	80	8Conv2d+2Lin	1Lin+4ConvTrans2d	2	1	BCE
19	80	8Conv2d+1Lin	1Lin+4ConvTrans2d	1	1.7	WASS
20	80	8Conv2d+1Lin	1Lin+4ConvTrans2d	2	1.5	WASS
21	80	10Conv2d+1Lin	1Lin+4ConvTrans2d	2	1.5	BCE
22	80	10Conv2d+1Lin	1Lin+4ConvTrans2d	4	1.5	BCE
256-0	80	5Conv2d+2Lin	1Lin+3ConvTrans2d	3	1.7	BCE
256-1	86	7Conv2d+2Lin	1Lin+3ConvTrans2d	3	1.7	BCE

Table 1: a view of all the architectures used in the experiments, note that "WASS" stand for Wasserstein Loss function, "Disc. Step" for Number of iteration (step) of Discriminator before train the Generator, "Penalty" is for Gradient Penalty

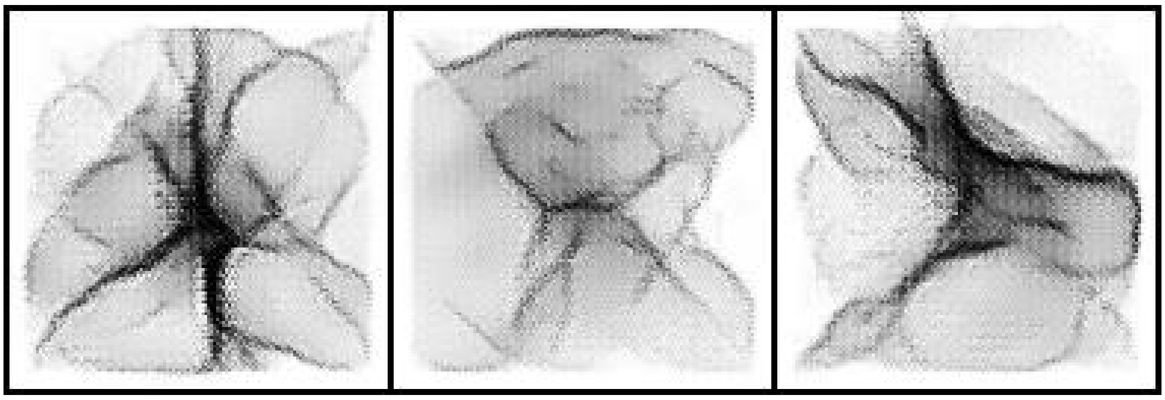


Figure 14: 3 images from the "Best Model"

Number	Loss D	Loss G	SSIM	PSNR	SWD	Accuracy	Precision	Recall	F1-score
CLEAN	-	-	-	-	-	-	-	-	-
0	-	-	-	-	-	-	-	-	-
1	0.4465	1.6653	0.129	8.39	-	0.49	0.25	0.49	0.33
2	0.13	6.56	0.35	6.36	-	0.5	0.25	0.5	0.33
3	0.12	7.7	0.02	1.43	-	0.5	0.25	0.5	0.33
4	nan	0	0.26	7.42	1096.63	0.5	0.25	0.5	0.33
5	1.43	0.68	0.15	7.44	767.75	0.5	0.25	0.5	0.33
6	0.21	2.87	0.2	8.42	770.87	0.5	0.25	0.5	0.33
7	0.48	2.13	0.16	8.62	960.23	0.5	0.25	0.5	0.33
8	0.23	2.21	0.2	8.59	870.64	0.5	0.25	0.5	0.33
9	0.41	2.3	0.14	8.31	806.32	0.5	0.25	0.5	0.33
10	very low	very low	very low	very low	very low	0	0	0	0
11	0.05	4.21	0.13	7.98	922.01	0.5	0.25	0.5	0.33
12	0.11	3.25	0.15	8.32	840.07	0.5	0.25	0.5	0.33
13	0.31	1.87	0.18	8.64	879.62	0.5	0.25	0.5	0.33
14	0.02	6.44	0.34	7.92	1207.93	0.5	0.25	0.5	0.33
15	0.38	1.33	0.14	8.65	889.09	0.5	0.25	0.5	0.33
16	0.29	1.56	0.19	8.33	702.76	0.5	0.25	0.5	0.33
17	0.19	1.94	0.2	8.05	637.77	0.5	0.25	0.5	0.33
18	0.23	2.87	0.18	8.33	748.62	0.5	0.25	0.5	0.33
19	0.01	-1	0.01	5.32	2740.26	0.5	0.25	0.5	0.33
20	-0.77	-0.22	0.19	8.25	689.49	0.5	0.25	0.5	0.33
21	0.19	1.95	0.2	8.23	668.57	0.5	0.25	0.5	0.33
22	0.18	2.44	0.2	8.28	664.37	0.5	0.25	0.5	0.33

Table 2: All the data related to the experiments in general SSIM and PSNR The higher the better and for SWD lower is better, some indexes are missing due to my mistake...

5 Conclusion

In this final chapter, we summarize the main findings of our research on the behavior of Generative Adversarial Networks. Through a series of experiments, we aimed to understand the impact of different hyperparameters on the performance of GANs. The results of these experiments provide insight into the workings of GANs and highlight the trade-offs between performance and computational complexity.

This report has presented a study on the behavior of Generative Adversarial Networks (GANs) under different parameters and configurations. Through the course of 22 experiments, we have investigated the impact of batch size, number of layers in both generator and discriminator, upsampling method, loss functions, and gradient penalties on the quality of generated images. Our results have shown that modifying the gradient penalty and the number of iteration in the discriminator have had a significant impact on the quality of the images. However, it is also worth noting that the best network using metrics such as SSIM, SWD, and PSNR does not necessarily correspond to the network with the best images according to human evaluation.

Furthermore, our findings have revealed that adding layers to the generator does not necessarily lead to an improvement in image quality, and that there is no clear difference between using the Wasserstein Loss and the BCE loss. It is also worth mentioning that the results obtained by adding layers to the discriminator were inconsistent, as although an increase in complexity was evident, the resulting images were often similar to those generated with fewer layers. This might be due to the wave-like nature of GANs.

In conclusion, while the results of this study may not be the best, this project has allowed us to delve into the concept of neural networks and, more specifically, Generative Adversarial Networks (GANs), gaining a deeper understanding of their functioning and their pros and cons. Through experimentation with various parameters such as batch size, number of layers in the generator and discriminator, and different loss functions, we have seen both the strengths and limitations of GANs. While the results may not be perfect, this project has provided valuable insight into the nature of GANs and the potential they hold. However, it is important to acknowledge that the field of GANs is still in its infancy, and there is much to be discovered about these networks. We hope that this work will encourage further exploration of this exciting and rapidly growing area.

References

[1] Generative adversarial nets. https://elearning.unisi.it/pluginfile.php/546867/mod_resource/content/4/gan_paper_nips_5423-generative-adversarial-nets.pdf. (Accessed on 02/02/2023).

[2] koshian2/swd-pytorch: Sliced wasserstein distance (swd) in pytorch. https://github.com/koshian2/swd-pytorch. (Accessed on 02/02/2023).

[3] Lissajous curve - wikipedia. https://en.wikipedia.org/wiki/Lissajous_curve. (Accessed on 02/02/2023).

[4] soumith/ganhacks: starter from "how to train a gan?" at nips2016. https://github.com/soumith/ganhacks. (Accessed on 02/02/2023).

[5] tuxar-uk/harmonumpyplot: Harmonographs generated with numpy and matplotlib. plain (random) and gui versions. https://github.com/tuxar-uk/Harmonumpyplot. (Accessed on 02/01/2023).

[6] Vainf/pytorch-msssim: Fast and differentiable ms-ssim and ssim for pytorch. https://github.com/VainF/pytorch-msssim. (Accessed on 02/02/2023).

[7] yandex-research/gan-transfer: Supplementary code for "when, why, and which pretrained gans are useful?" (iclr'22). https://github.com/yandex-research/gan-transfer. (Accessed on 02/02/2023).

[8] Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

[9] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

[10] Anna Hu, Siqiong Chen, Liang Wu, Zhong Xie, Qinjun Qiu, and Yongyang Xu. Wsgan: An improved generative adversarial network for remote sensing image road network extraction by weakly supervised processing. *Remote Sensing*, 13(13), 2021.

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

[12] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. How good is my gan? In *Proceedings of the European conference on computer vision (ECCV)*, pages 213–229, 2018.