Advanced Topics in Deep Learning- Final Course Project

Ori Malca (ID. 315150599), Anastasia Lurye (ID. 332644848),
Tom Zur (ID. 208287516), Noam Gil Or Yizraeli (ID. 206459307)

Submitted as final project report for the Advanced Topics in Deep Learning course, Colman, 2023

1 Introduction

Our project aims to tackle the generation problem presented in the Kaggle competition "I'm Something of a Painter Myself". Specifically, the challenge is to generate Monet-style images using GANs. To accomplish this task, we plan to implement two different GAN architectures: CycleGAN and DCGAN.

1.1 Data

We received a labeled dataset consisting of 300 Monet paintings in JPEG format, 300 Monet paintings in TFRecord format, 7028 photos in JPEG format and 7028 photos in TFRecord format. All of the photos was sized 256x256x3
As a project requirement we resized the photos to a size of 320x320x3.

1.2 Problem

Our project involves a generation problem presented in the Kaggle competition "I'm Something of a Painter Myself". We plan to solve this problem using two approaches: CycleGAN and DCGAN.

2 CycleGAN

2.1 First Experiment

We chose to use the UNET architecture as our generator for the first experiment, with modifications made to accommodate an input size of 320x320x3. Meanwhile, we opted for a 16x16 receptive field patchGAN as our discriminator.

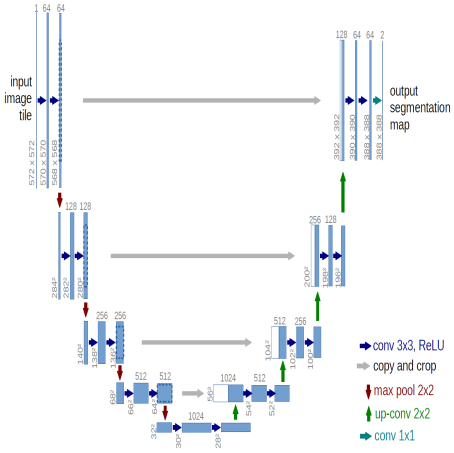


Figure 1: UNET Architecture

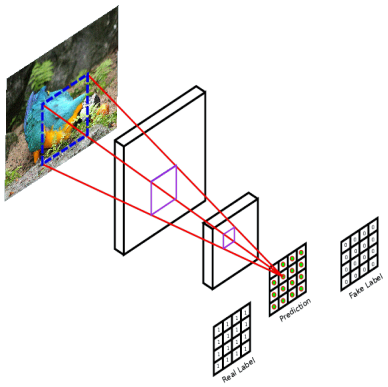


Figure 2: PatchGAN Architecture

We chose to use the UNET architecture as our generator for several reasons. Firstly, it is a powerful auto-encoder architecture that has been proven to be effective in generating solutions to various problems. Additionally, UNET utilizes skip connections, similar to those used in ResNet architecture. These connections allow for an uninterrupted gradient flow from the first layer to the last layer, effectively addressing the issue of vanishing gradients. The use of concatenative skip connections provides an alternative method for ensuring feature reusability with the same dimensionality from earlier layers, which is widely utilized. Long skip connections are also used to recover spatial information lost during downsampling by passing features from the encoder path to the decoder path. Lastly, short skip connections help stabilize gradient updates in deep architectures.

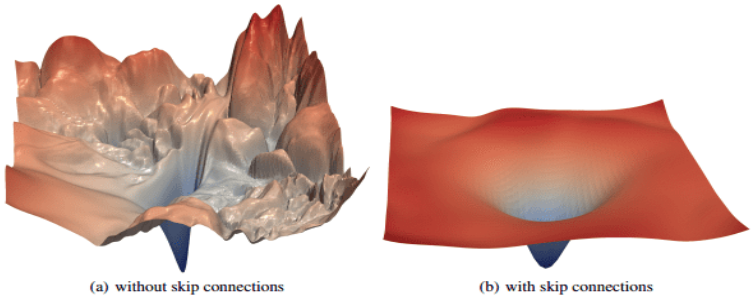


Image is taken from this paper.

We chose to utilize a 16x16 receptive field PatchGAN as our discriminator for two reasons. Firstly, it was a project requirement. Secondly, according to the paper, it is the second-best performing architecture after the 70x70 receptive field PatchGAN (as shown in the images below from the paper).

6.1.2 Discriminator architectures

The 70 × 70 **discriminator** architecture is:
`C64-C128-C256-C512`

After the last layer, a convolution is applied to map to a 1-dimensional output, followed by a Sigmoid function. As an exception to the above notation, BatchNorm is not applied to the first `C64` layer. All ReLUs are leaky, with slope 0.2.

All other discriminators follow the same basic architecture, with depth varied to modify the receptive field size:

1 × 1 **discriminator:**
`C64-C128` (note, in this special case, all convolutions are 1 × 1 spatial filters)

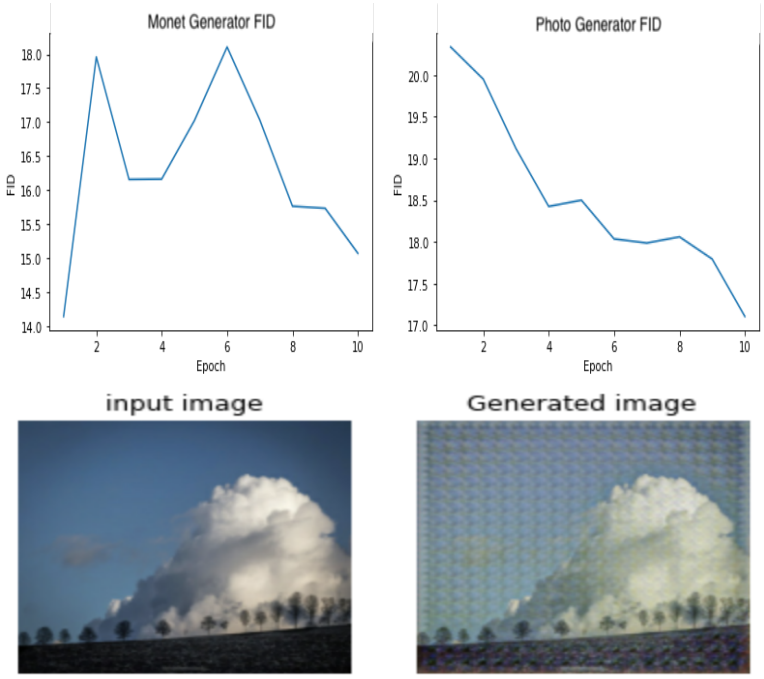
16 × 16 **discriminator:**
`C64-C128`

Discriminator receptive field	Per-pixel acc.	Per-class acc.	Class IOU
1×1	0.39	0.15	0.10
16×16	0.65	0.21	**0.17**
70×70	**0.66**	**0.23**	**0.17**
286×286	0.42	0.16	0.11

We opted to use Instance Normalization instead of Batch Normalization, as it has been found to be effective for GANs. This is because Instance Normalization can reduce internal covariate shift in the generator network, leading to more stable and faster training. This is crucial for GANs, which require a balance between the generator and discriminator networks, and any instability during training can result in mode collapse or other issues. Mode collapse is a common problem in GANs, where the generator produces limited or repetitive output instead of generating diverse and distinct samples.
Additionally, we augmented the data to increase the variety of images, improve model robustness, and reduce overfitting.
We standardized the data between minus one and one (zero-centered, mean of 0 and variance of 1). This was done to prevent an elongated cost function and to achieve a more symmetric cost function, which speeds up the learning process. As Andrew Ng. explains here, a more symmetric cost function is easier for the loss function to converge to. Additionally, standardizing the data makes the network more robust to changes in the data distribution (makes the network more stable to covariate shifts).
We selected the Adam optimizer for our generator and discriminator models, which is a combination of SGD with momentum and RMSProp. Adam combines the exponential moving average update technique of the gradients used in SGD with momentum and RMSProp (as explained in this source).
We chose to use a batch size of 1 from the intuition that when using a batch size of 1 the generator and discriminator are updated after every single training example, which means that the generator can adapt to the specific characteristics of each image in the source and target domains and this can result in more precise mappings between the two domains and better image quality.

We chose to use cross-entropy as our loss function because it is a convex function, which facilitates convergence to a global minimum and helps to prevent the model from getting stuck in local minimum during training.
We monitored the FID score for each epoch during training and saved a model checkpoint corresponding to the epoch with the best FID score.
The Frechet Inception Distance (FID) score is a metric used to evaluate the similarity between two sets of images. It compares the statistics of the real and generated images by calculating the distance between the means and covariances of their feature representations, which are obtained from a pre-trained Inception neural network.
We trained the model for 10 epochs.

The metrics for the first experiment:



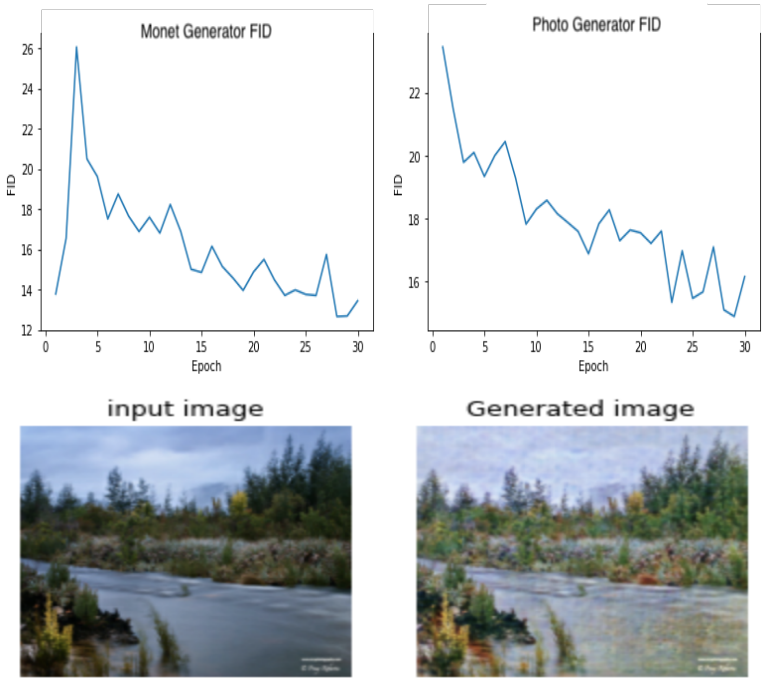
FID score: 15.076563527753413

2.2 Second Experiment

For our second experiment, we utilized the same architecture as in experiment 1 but modified the architecture of the 16x16 PatchGAN to a 70x70 PatchGAN, as suggested by the paper and illustrated in the images above, since it is reported to perform better.
We decided to maintain a batch size of 1 throughout training.
The model was trained for 30 epochs, with a checkpoint saved for the best FID

score achieved during training.

The metrics for the second experiment:



FID score: 13.43757864563977

2.3 Third Experiment

In the third experiment, we utilized the same architecture as the second experiment, with the only difference being the use of a linearly scheduled learning rate (a scheduled learning rate in a linear fashion), we observed that this modification led to an improvement in our model's performance.
The reason we thought to use a linearly scheduled learning rate is because from observing the photos generated in the previous experiment we believed that the loss function was very close to its minimum value in the parameter space.

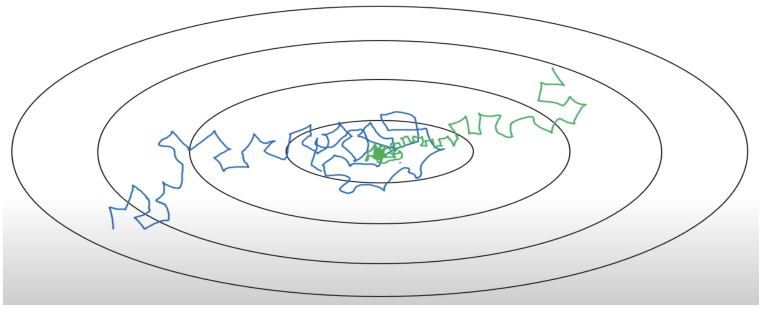
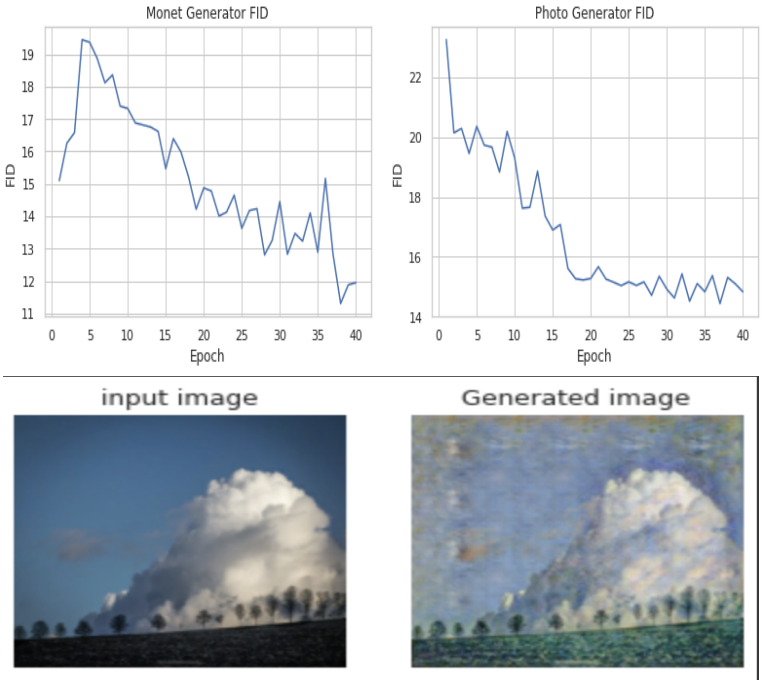


Photo is taken from Andrew Ng. lecture on DeepLearning.ai course

In the photo, the BLUE line depicts convergence without a learning rate schedule, while the GREEN line shows convergence with a learning rate schedule. We trained the model for 40 epochs with a batch size of 1 and FID score checkpoint callback.

The metrics for the third experiment:



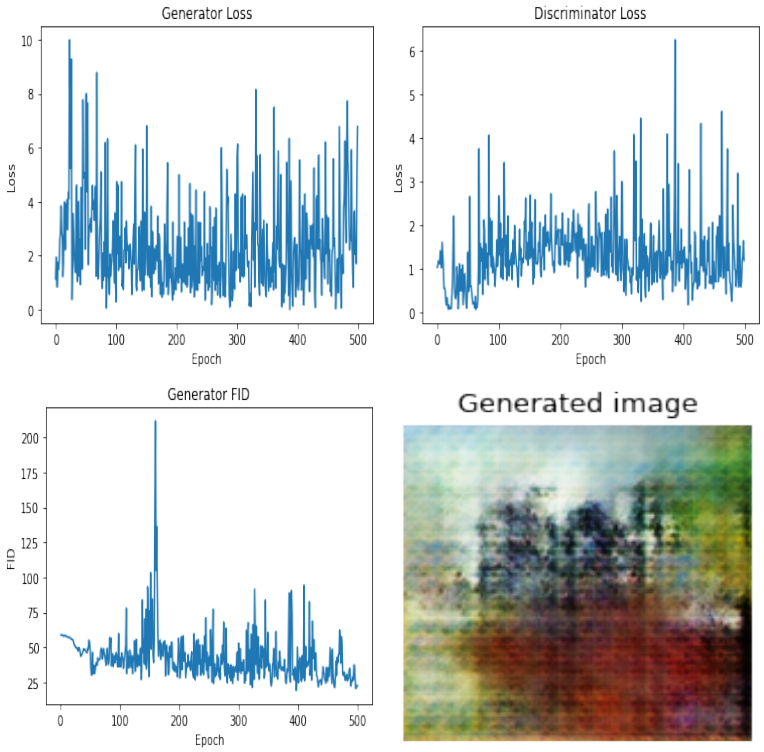
FID score: 11.955208682180288

3 DCGAN

3.1 First Experiment

In our first experiment, we designed a custom neural network from scratch, drawing inspiration from the architectures used in the CycleGAN section. Specifically, we took the UNET architecture from the 10x10x1024 layer after the latent space layer. To do this, we added a dense layer that takes the noise input and returns the sum of 10 X 10 X 1024 = 102400. We then used a reshape layer to convert this output to a 10x10x1024 tensor.
We chose to use the same discriminator as the one used in the CycleGAN section, as it was found to work well with that architecture.
We trained the model for 500 epochs with a batch size of 32.

The metrics and results for the first experiment:



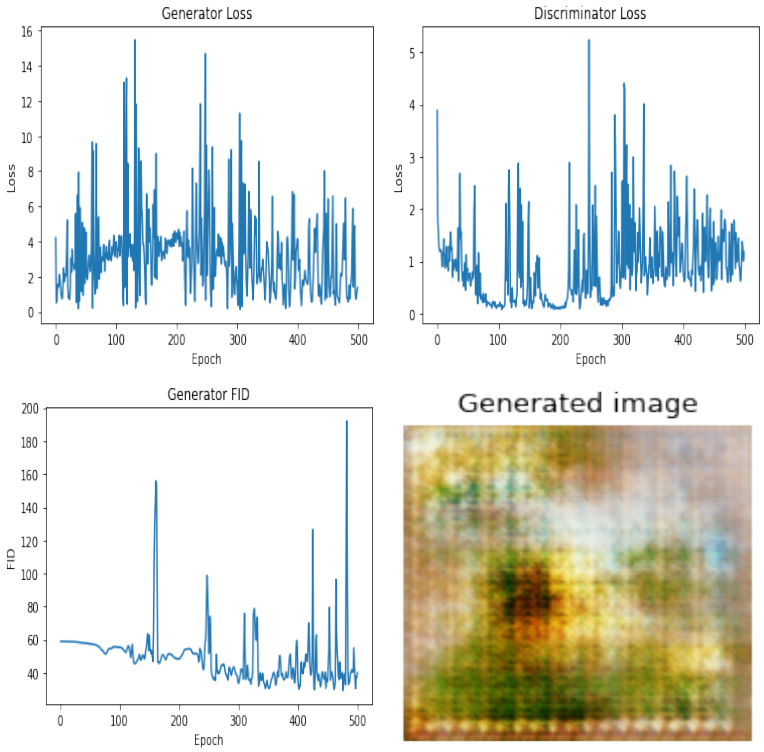
FID score: 22.568988418196703

3.2 Second Experiment

In the second experiment, we utilized the identical architecture employed in the first experiment but made two alterations. Firstly, we increased the batch size

to 128, and secondly, we integrated batch normalization and ReLU activation at the start of the generator architecture, right after the dense layer at the beginning. The rationale behind switching to a batch size of 128 was to encourage the generator to produce more diverse monet photos by generating a larger number of images per batch. A larger batch size would penalize the generator more for repeating the same mistake multiple times, thus compelling it to explore new connections and possibilities in the domain of the generated photos. In contrast, a small batch size, such as 1, might constrain the generator to generate only one possible monet photo for each noise input during the learning process, leading to a lack of diversity in the generated images. This could occur because the generator would repeatedly modify the same photo after the discriminator "caught" it and would only aim to perfect this single photo, resulting in the generator being capable of generating only one monet photo in the domain of monets photos.
Furthermore, we chose to use a batch size that is a power of 2 since most GPU's memory is organized in this manner, resulting in smoother processing flow through the GPU.
We trained the model for 500 epochs.

The metrics and results for the second experiment:

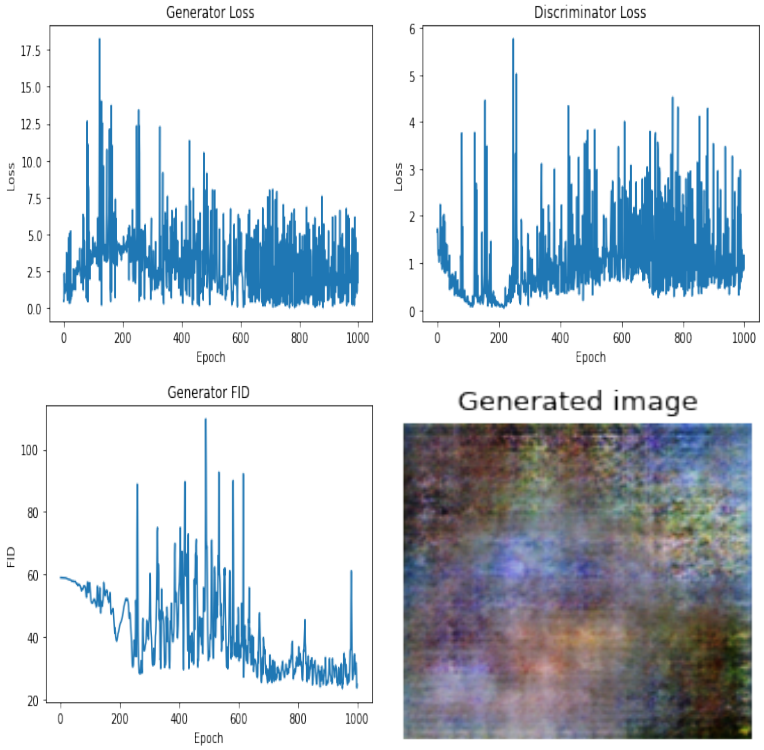


FID score: 39.96566985928735

3.3 Third Experiment

For our third experiment, we employed the identical architecture utilized in experiment 2 and trained it for 1000 epochs. We believed that it was necessary to train for more epochs based on the outcomes.

The metrics and results for the third experiment:



FID score: 24.661785945438943

4 Platform and Tools

We carried out our experiments on the Google Colaboratory platform.
NumPy library was used for metric calculations, Matplotlib library was used for visualization of different metrics, Seaborn library was used for visualizing the scheduled learning rate. TensorFlow/Keras was utilized for constructing the models in the CycleGAN and DCGAN architectures.

5 Discussion

This project was an enjoyable and satisfying experience. We were able to tackle a challenging generation problem using two different approaches: CycleGAN and DCGAN. It was exciting to see how each approach performed and to compare the results. Furthermore, it was particularly exciting to observe how potent and beneficial CycleGAN is for problems involving unpaired data.

6 Code

To access the **train** notebook click here
To access the **test** notebook click here