Deep Learning- Final Course Project

Simon Raviv (ID. 312847478), Adam Gavriely (ID. 309677284)

Submitted as final project report for the DL course, BIU, 2021

1 Introduction

"Every artist dips his brush in his own soul, and paints his own nature into his pictures."
-Henry Ward Beecher

We recognize the works of artists through their unique style, such as color choices or brush strokes. The "je ne sais quoi" of artists like Claude Monet can now be imitated with algorithms thanks to generative adversarial networks (GANs). Our paper goal is to recreate the style of Claude Monet from scratch using photographs and limited number of Monet paintings provided to the model, by using GAN architecture.

Computer vision has advanced tremendously in recent years using deep learning techniques. One of the most developed techniques in recent years are GANs which now capable of mimicking objects in a very convincing way. But creating museum-worthy masterpieces is thought of to be, well, more art than science.

The challenge we aim to solve: Create a Deep Learning model that can create realistic Monet paintings from photos, using only **30 style paintings**.

Our approach is to use **Cycle GAN**, built on top of the regular GAN. The goal of Cycle GAN architecture is to create a mapping between two domains without paired examples between the domain. We will use it to create transformation from real world photos to Monet paintings.

Our challenge is focused on using small number of style paintings from the Monet paintings domain. In order to alleviate this limitation, we will be experimenting with different types of an augmentation techniques: standard picture augmentation, e.g. rotation, crop, etc and advanced augmentation technique called Differentiable Augmentation.

1.1 Related Works

Generative Adversarial Networks (GANs), by Goodfellow *et al*, are powerful generative models, which have achieved impressive results on different computer vision tasks, e.g., image generation, editing and in painting. However, GANs are difficult to train, since it is hard to keep the balance between the generator and the discriminator, which makes the optimization oscillate and thus leading to a collapse of the generator.

The model consists of two neural networks: a generator model and a discriminator model. The generator is a neural network that creates the images. This generator is trained using a discriminator, which is also a neural network. The two models will work against each other, with the generator trying to trick the discriminator, and the discriminator trying to accurately classify the real vs. generated images.

Unpaired Image-to-Image Translation. To alleviate the issue of pairing training data, Zhu *et al.* introduce CycleGAN, which learns the mappings between two unpaired image domains without supervision. The 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, there is an additional inverse mapping $F : Y \rightarrow X$ and a *cycle consistency loss* to enforce $F(G(X)) \approx X$ (and vice versa).

U-Net: Convolutional Networks for Biomedical Image Segmentation, by Ronneberger *et al* The key idea: skip connections. In this architecture, up-sampled feature maps concatenated with skipped connections from the encoder, add convolutions and non-linearities between each up-sampling step. The skip connections have shown to help recover the full spatial resolution at the network output, by preserving and sharing knowledge between the encoding and decoding steps.

Differentiable Augmentation for Data-Efficient GAN Training, introduced by Zhao *et al*. It is a set of differentiable image transformations used to augment data during GAN training. The transformations are applied to the real and generated images. It enables the gradients to be propagated through the augmentation back to the generator, regularizes the discriminator without manipulating the target distribution, and maintains the balance of training dynamics. Three choices of transformation are preferred by the authors in their experiments: Translation, CutOut, and Color.

2 Solution

2.1 General Approach

Our approach is to use GAN architecture to solve this challenge, in particular CycleGAN. The motivation of this approach based on the superior performance CycleGAN showed on the problem of unpaired image to image translation.

The plan was to start with the implementation of the original paper model by Zhu *et al*, as a baseline model. We plan to modify the model to be suited better for the problem at hand by experimenting with train duration(number of epochs), smaller network, regularization techniques, hyper parameters tuning etc.

Data augmentation. We had limited number of style pictures, therefore we plan to implement and experimented with 2 types of data augmentation techniques. The first technique was using regular augmentation that doesn't affect the style of the pictures, e.g. rotations, cropping, scaling, etc.
The second technique was using advanced augmentation introduced in *Differentiable Augmentation for Data-Efficient GAN Training*, introduced by Zhao *et al*.

Based on our observations from the experiments we will conduct, we plan on trying different approaches from papers that came after the original paper, in order to improve the performance of our model.

2.2 Notataion

We use the following notation in the next sections.

Let `cXsY-Z` denote a $X \times$X Convolution-InstanceNorm-ReLU layer with Z filters and stride Y.
Let `dk` denote down-sample block with a 3×3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2. Reflection padding was used to reduce artifacts.
Let `dUk` denote a UNET down-sample a 4×4 Convolution-InstanceNorm-LeakyReLU layer with k filters and stride 2.
Let `Rk` denote a residual block that contains two 3×3 convolutional layers with the same number of filters on both layer.
Let `uk` denote a 3×3 fractional-strided-Convolution-InstanceNorm-ReLU layer with k filters and stride $\frac{1}{2}$.
Let `uUk` denote a UNET up-sample a 4×4 Convolution-InstanceNorm-LeakyReLU layer with k filters and stride 2.
Let `Ck` denote a 4×4 fractional-strided-Convolution-InstanceNorm-ReLU layer with k filters and stride $\frac{1}{2}$.
Let G be the mapping generator from domain X to Y and F vice versa generator networks.

2.3 Design

2.3.1 Architecture

The final architecture we chose is CycleGAN from *Generative Adversarial Networks (GANs)*, by Goodfellow *et al*, following all the experiments and observations we describe at the experiments section.

Generator architecture
`c7s1-64,d128,d256,R256,R256,R256,R256,R256,R256,u128,u64,c7s1-3`

Discriminator architecture
We use 70×70 PatchGAN. After the last layer, we apply a convolution to produce a 1-dimensional

output. We do not use InstanceNorm for the first `C64` layer. We use leaky ReLUs with a slope of 0.2. The discriminator architecture is: `C64-C128-C256-C512`

2.3.2 Loss Functions

Generator Loss - $MSE(1, G(X))$
Discriminator Loss - $(MSE(1, X) + MSE(0, G(X))) * \alpha_{disc}$
Cycle Loss - $MAE(X, F(G(X)) * \alpha_{cycle}$
Identity Loss - $MAE(Y, G(Y)) * \alpha_{cycle} * \alpha_{identity}$

2.3.3 Optimizer

$Adam(learning\_rate = 0.0002, beta_1 = 0.5)$

2.3.4 Kernel Initialization

We use the following random normal distribution: $\mathcal{N}(0,\, 0.02)$.

2.3.5 Platform

We use Colab as a development platform with the following HW.

- **CPU** - 4 Cores Intel(R) Xeon(R) CPU @ 2.20GHz
- **GPU** - NVIDIA Corporation GP100GL Tesla P100 PCIe 16GB

2.3.6 Code

TensorFlow with Keras API as our SW framework.

2.3.7 Technical Challenges

We had several technical issues we describe below, the major one is a bug in our kernel initialization.
Kernel initialization bug
The bug was that we have defined the kernel initialization objects as global objects, this means we have initialized all the kernel weights in all the layers with exact the same weights, which is of course very bad for neural network training.
We did a lot of work, for longs days, multiple experiments with different architectures and didn't see the improvements and reasonable results. We found and fixed the bug about 8 days before the submission. After that we did all the experiments and architectural changes again and saw very different results, which we describe in the experiments section.

GPU determinism
We seed all our random operations, but we saw after several tests that our experiments are not deterministic. We did research and found that there is an issue with determinism when using GPU with TensorFlow. We read the paper *Non-Determinism in TensorFlow ResNets* by Morin *et al* and understood why. We decided to use the GPU in anyway, due to its soupier performance. We have experimented with TPU, but saw that the results are not deterministic there as well.

3 Experimental Results

We have conducted multiple experiments, starting with the baseline model. At each experiment listed below, we try to optimize and improve certain part of the model. We will present the empiric results and the motivation for the conducted experiments. The experiments below, are subset of the all the experiments we did.

3.1 Evaluation Metric

We use Fréchet Inception Distance (FID) introduced by Heusel *et al* in *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*, as our evaluation metric of the model performance. The FID captures the similarity of generated images to real ones better than the Inception Score.
We use the FID calculation at each epoch of the training in order to understand if our model generates the distribution of Monet paintings in an accurate manner. We split our real photos dataset to 40/60, 40% of the real photos used for FID evaluation.

3.2 Base Experiment

The experiment based on the model from the original CycleGAN architecture.

3.2.1 Architecture

The same main architecture as described in section *2.3 Design*.

3.2.2 Results

Parameters: epochs = 500; steps per epoch = 30; batch size = 1; no augmentation;

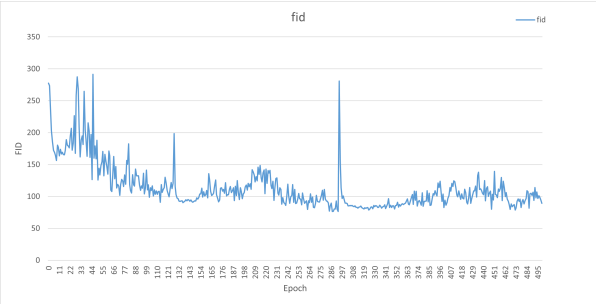

Figure 1: FID over epochs

Figure 2: moving avg. FID

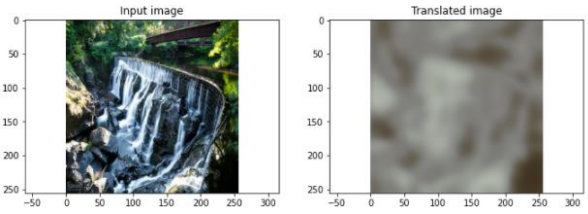

Figure 3: after 1 epoch

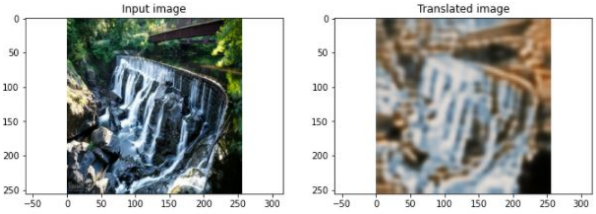

Figure 4: after 10 epochs

3.2.3 Observations

In figure 1, as expected for SGD, we can see that using batch size of 1 leads to high fluctuation for the FID score, but also yielded better results.

3.3 Augmentation Experiment

We wanted to enrich the dataset so we used the same model and tried two different augmentations individually and combined.

1. Regular augmentation that applies rotations, flips, transformations and cropping each with random probability.

2. Differentiable augmentation introduced in *Differentiable Augmentation for Data-Efficient GAN Training*, introduced by Zhao *et al*

3.3.1 Architecture

Same architecture as above.

3.3.2 Results

Parameters: epochs = 470; steps per epoch = 30; batch size = 1/5
With regular data augmentation (flips, rotations, resizes and crops)

Figure 5: moving avg. FID with batch size 1
With Diff-Augmentation

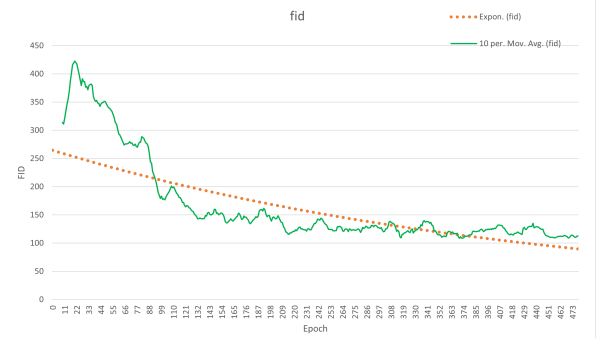

Figure 6: moving avg. FID with batch size 5

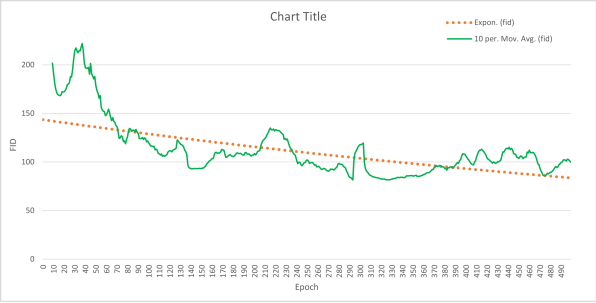

Figure 7: moving avg. FID with batch size 1

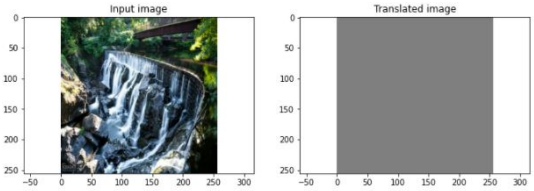

Figure 8: after 1 epoch

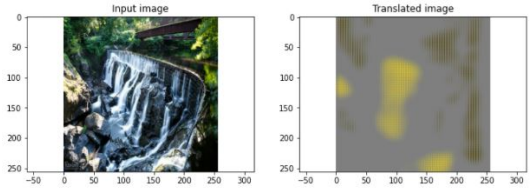

Figure 9: after 10 epochs

3.3.3 Observations

From figures 2 and 5, no augmentation and regular augmentation, we see that having this kind of augmentation leads to less fluctuation, but does not help improving the results as we thought in the beginning. We assume the reason is this due to the cropping and scaling of the images at the augmentation phase.

From figures 5 and 6, we see that increasing the batch size leads to higher FID scores. We assume the cause is the high variance in Monet's images, batching the images together meaning averaging the gradient's directions which is a problem because they differ too much.

From figures 5 and 7, we see that diff augmentation showed minor improvement over the regular augmentation.

From figures 4 and 9, we see that augmenting on every step make it harder for the model to learn the real images. We tried giving each augmentation in the augmenting process a probability to occur to enlarge the variance of augmentations which had minor effect. We tried controlling the rate at which the augmentation

happens at all, thus exposing the model to real Monet images more than augmented images. This showed improved results comparing to augmenting on every step but had lower results than leaving augmentation out.

3.4 UNET Generator Experiment

We saw in previous experiments that our model struggles with recognizing the real photos content. It does recognize the content, but not fast enough in our opinion. In addition, we saw the Diff-Augmentation, made it harder for the model to recognize the photos. Therefore, we have decided to switch our generator network to UNET, which transfers the knowledge between the encoding to the decoding stages using skip connection between the two phases. By doing this, we assume, the content will be learned easier and faster thus improving the performance of the generators networks.

3.4.1 Generator Architecture

`dU64,dU128,dU256,dU512,dU512,dU512,dU512,dU512,uU512,uU512,uU512,uU256,uU128,uU64, uU3`, with the following modifications: The first down-sample layer didn't contain instance transform normalization; first 3 up-sample layers did also dropout with the probability of 0.5; last layer used TanH activation function; A skip connection between each down to up sample blocks.

3.4.2 Discriminator Architecture

The same discriminator network as above.

3.4.3 Results

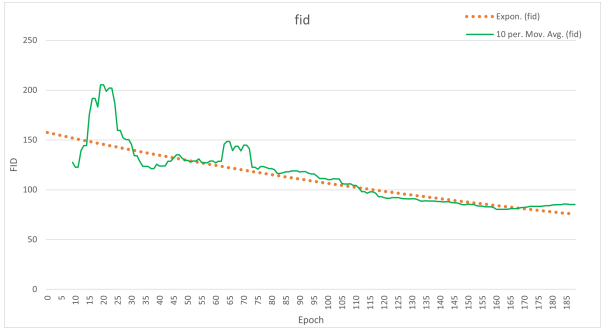

Figure 10: moving avg. FID with batch size 1

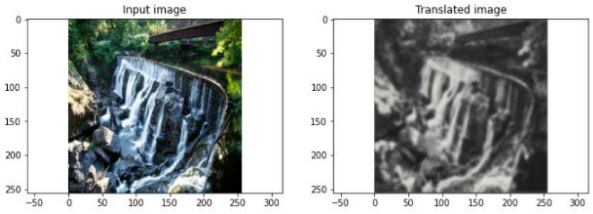

Figure 11: after 1 epoch

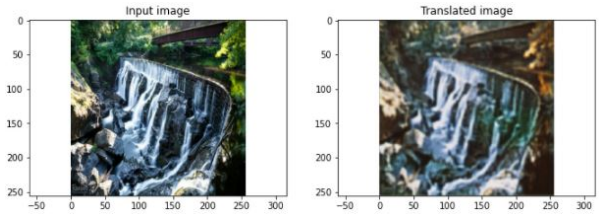

Figure 12: after 10 epochs

3.4.4 Observations

As expected, from figures 3, 4, 8, 9, 11 and 12 we see visually that indeed UNET network learned the content of the photo much faster. However, the FID scores were still lower than expected. We assume this is because the UNET generator focused too much on the content of the image instead of the style we wanted to learn, leading to generating images that are too similar to the domain they came from, instead generating images from the distribution of the target domain, Monet paintings. As we can see in figure 10, the learning seems to be good and stable with the UNET architecture, but it reaches the limit of around FID 80, and doesn't converge anymore. We assume that this is due to the same reason above.

3.5 Input Experiments

We noticed that the input images chosen effected greatly on the network output so we inspected the Monet paintings and noticed he had more than one style on his paintings. The majority of Monet's images were nature so we decided on focusing on the nature he draws since due to the several styles he had and the small dataset of non-nature paintings the model had worse results for images containing people, animals, geometric figures and so we manually chose 30 nature paintings that had the same style.

For the real photos, we noticed lower results for darker images and non-nature images. We created out own dataset using the given dataset, expanded with more nature images from the internet. We removed the darker images and removed non-nature images we found until we reached the minimum of 7000 for the competition.

This procedure had a great effect on our results between 10-20 FID.

3.6 Additional Experiments

In addition to all the above experiments, for every model we implemented, we also tried to change the following

- **Amount of layers** - removing small amount of layers from each generator and discriminator
- **Epochs and Steps per epoch** - optimizing the number of epochs and steps per epoch needed to reach optimal solution.
  We noticed that performing many steps per epoch but small epoch is the same as performing small amount of epochs with many steps each.
- **Batches** - different amount of batches
- **Stages of augmentations** - applying augmentation statically before running, applying dynamically while running before each step. Applying to the entire dataset and applying just to Monet's images
- **Dual headed discriminator** - When we suspected we reached model collapse we found article that suggests implementing 2 heads for the Monet discriminator.

  One discriminator rewards high scores for real Monet images whilst the other, conversely, favoring data from the generator, and the generator produces data to fool both two discriminators.
- **Filter sizes** - We tried changing the filter sizes between 3-7 to try and achieve better perception fields
- **Additional regularization** - We have experimented with adding dropout to all/some layers, in addition to the 3 dropout layers at the beginning of the up-sample layers in the generators. This is in order to generalize more and not overfit on the 30 Monet paintings we had. It didn't help much the training.
- **Edge start to Monet Generator** - We tried to give the Monet generator an edge by training the model, than taking partially trained generator and start with it instead of randomly initiated weights for this generator. This didn't help to train the generator better.
- **Higher weight for Monet loss** - We observed that Monet generator loss, was lower than photo generator loss. We have modified the loss functions to give higher weight to the Monet generator, to make them even. It didn't help the training.

4 Discussion

Technique Since we are new to this area and did not know what will work or what will we chose to implement a mechanism that saves a snapshot of the model whenever he reaches a new high score throughout the training.

We tried 3 architectures and chose the ResNet Generator with the Discriminator discussed above. We trained the model using SGD and did not use augmentation. the save-best mechanism caught a snapshot of the model at the best results which was better than all other models.

Insights

- Image selection is a critical. It may take quite some time but it is very important to inspect the dataset, look for outliers and clean the database. The same model could fail or be outstanding based on the images selected.
- There is not silver bullet. every model we tried succeeded in different parts and it is important to take the time and understand the problem before trying to use solutions that worked for others.

Summary of the process We decided to start from the CycleGAN paper architecture, implement it and then research how to improve the model to the problem of limited number of Monet paintings. We have described the path we took, researching the papers came after the original one. We have implemented different architectures and conducted the experiments we described.
After all the path we went through, we saw that the first model we have implemented gave the best performance.

4.1 Future Work

Due to the submission time constraints we could not try everything we wanted, we offer some ideas that can be considered in order to improve the results.

- **Image Selection** - Use advanced algorithmic ideas such as Genetic algorithm to choose the best subset of images to train from.
- **Environment** - Train on a better GPU environment in order to avoid Out-Of-Memory problems in models with big batches and larger networks.
- **Loss Factors** - In the model there are a lot of factors for each loss computed, trying different factors might give better results.
- **Learning Decay** - We observed in some experiments a certain point, the FID score starts to fluctuate around the same scores for quite a lot of epochs. We assumed that it is due to the optimization algorithm reached a point where the learning rate step was too big to converge to a better minimal point.
- **Discriminator soft noisy smoothing** - Let the discriminator output random number between 0-0.1 for generated images and between 0.9-1 for real images.
- **Loss functions** - Alter between different loss functions.

5 Code

All the required submissions can be seen the the following GitHub repository:
Please refer to the README.md file in the repository, everything explained there.

- GitHub Repository:
`https://github.com/SimonRaviv/deep_learning_biu_2021_final_project`
- GitHub README:
`https://github.com/SimonRaviv/deep_learning_biu_2021_final_project/blob/main/README.md`