Deep Learning- Final Course Project

Ohad Bruker ID. (027163153), Itay Lotan (ID. 308453935)

Submitted as final project report for the DL course, BIU, 2023

1 Introduction

In this project, we worked on creating a generative model that receives an image as input, and produces an image with a similar layout, but looks like a monet painting. There are two main goals that we strived to achieve - (1) create a model that will generate Monet painting that will look convincing to human eyes (which is pretty subjective), and (2) get the highest possible score in the project's Kaggle competition. Achieving both goals was challenging for us, because we didn't have prior experience in the field of generative models, and we were able to use only a small portion of the competition's train Monet images dataset of the (30 out of the 300 images of in the dataset). These challenges motivated us to experiment and try different approaches, and to utilize the techniques that were taught in class.

1.1 Related Works

In this project, we drew inspiration from the following sources: the original Cycle-Gan paper, the A Neural Algorithm of Artistic Style paper, and the shared work of other competitors in the same Kaggle competition. We were very impressed with the relative simplicity of the Cycle Gan architecture, and its powerful ability to produce high quality results after a relatively short training process.

2 Solution

2.1 General approach

In this project, we were instructed to implement 2 approaches (architectures) for solving the problem. One of these architectures was conditioned to be a Cycle Gan, and we decided to implement it with a patch Discriminator specifically. This architecture was the most promising one from the start, because the use case of the competition is very similar to the original use cases from the Cycle Gan paper. In addition, the duration that is required for its training process is relatively short, which was critical for our ability to experiment and improve its

performance for the provided task.

For the second approach, we have tested the Variational Autoencoder, but the results were poor. We believe that the VAE is a more suitable model for problems with a large dataset that incudes the source and the target images (for example, BW source images with colored target images as target).

At the end, we decided to use the Neural Style Transfer technic which blends the content of an image with the style of another image, monet style in this case.

2.2 Design

In order to solve the problem described above, we needed to run many trials in an efficient manner. We chose to run most of the experiments in Kaggle's environment for 3 reasons - (1) it allows the usage of 30 weekly hours of strong GPU machines (P100) which reduced the runtime of our experiments drastically, (2) it allows for long processes to run in the background, which isn't possible in Google Colab, and (3) making submissions for the competition and relying on their score to make decisions was simple through this environment. When working in this environment, the training process of most of our development models took a few minutes, which allowed flexibility in our exploration.

We used git for source control, and committed our progress frequently. This allowed us to compare the code to older versions of the notebook when something went wrong.

One of the main challenges that we had during the development was the insatiable nature of these generative models. We found out that even small changes in the experiment settings had a large impact on the training process of the model. In order to handle that, we needed to be very orginized in the planning of our experiments and their order. We documented the settings and results of each experiment in a dedicated excel sheet. Each experiment was built on top of the previous one, and its 'baseline' settings were set to the ones that yielded the best results in the previous experiment.

3 Experimental results

3.1 Architecture 1 - Cycle Gan with Patch Discriminator

This was the main architecture of this project. We invested a lot of our efforts in implementing it, planning its experiments and executing them. We also believe that it yielded the best results, judging by the Kaggle competition scores. First, we implemented a basic version of this mode, which produced results that we felt confident with (as a starting point). This basic version was trained on the

entire training set, and was built mainly by the concepts that were described in the original Cycle Gan paper. Then, we divided the improvement iterations of this model into 5 sequential experiments that covered these 5 aspects:

- The training stability and convergence (Which includes the number of training epochs and the optimizer settings)
- The selection of the 30 Monet training images (out of the 300)
- Changes in the Generator networks layout
- Changes in the Discriminator networks layout
- Usage of image augmentations to produce a richer version of the train dataset

The results of these experiments are described below, and their execution is included in the submitted notebook as well. Notice that when discussing the model's loss, the values of the losses values of the 4 models that participate in this architecture are mentioned - the Monet / Real photo generators, which generates an image that is similar to the input image in a Monet / Real world style, and the Monet / Real photo discriminators, which classify the input image as a real or a fake Monet / Real world images. Showing only the loss of the Monet generator model isn't indicative enough, because it needs to be evaluated in relation to its adversary, and the overall quality of the CycleGan components. In addition, please notice that we didn't have the room to include the output images of each experiment in this report. These outputs (5 prediction images for each results row in the table below) are orginized and included in the submitted notebook.

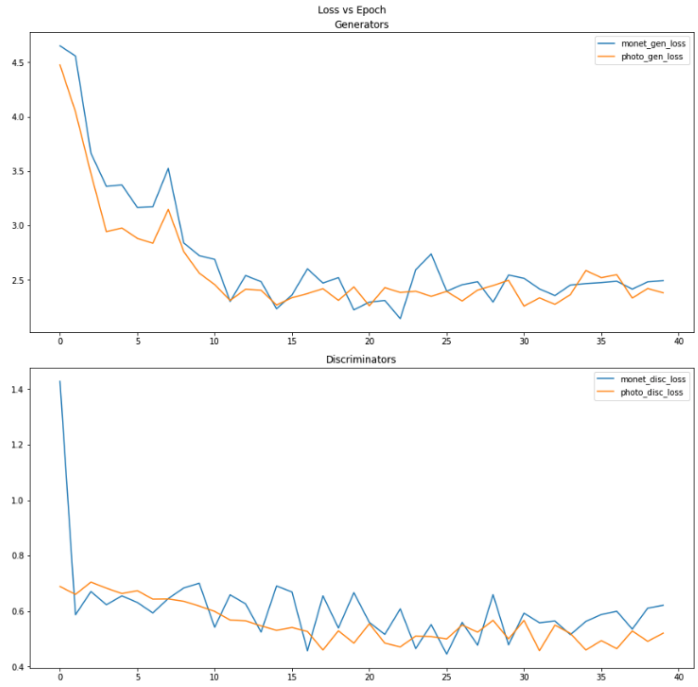
Experiment 1 - training stability and convergence
First, we wanted to find an optimal configuration of the optimizer and decide how many epochs needed for the training. To find the optimal optimizer configuration we checked at which learning rate the model was not able to converge. Then, we picked a slightly higher learning rate. Then, we have tested how many epochs are required until there is no effect on the model losses. In this case, we examined all the losses of the Cycle gun. Note that we have observed that a small decay of the optimizer learning rate (though not required for Adam optimizer) gives slightly more stable results. The optimizer configuration has been slightly changed during the experimentation. Our Adam optimizer learning rate vary between 0.001 to 0.0002 with decay of 0.001. 40 epochs were enough for the training to complete with this configuration.

Experiment 2 - selection of the 30 Monet training images
As described above, we were required to select 30 images out of the 300 training Monet dataset. Selecting a set of 30 images that will be effective for the train process was very important, so we decided to try several approaches. The general idea that drove this experiment is that in order to allow the model to

learn and find generic patterns in the Monet dataset, the 30 images that we'll choose need to be as diverse and different from each other as possible. For that end, we used a simple search algorithm that finds a subset of 'farthest' objects of size N out of a set of objects of size S, when provided a distance function between two objects (similar to the K-Means algorithm). We used this (link) blog post to learn about useful methods for defining distance between two images, and chose to include these 3 methods - pixel distance, structural distance, and earth mover's distance. We trained and evaluated our model using each of these methods, and here are the results that we got:

30 Images Selection Method	Monet Gen Loss	Photo Gen Loss	Monet Disc Loss	Photo Disc Loss	Kaggle Score
Farthest By Pixel Distance (selected)	2.49	2.38	0.62	0.52	80.87
Closest By Pixel Distance	12.23	239.79	0.67	0	Didn't Try
Farthest By Structural Distance	12.45	20.98	0.69	0.1	Didn't Try
Closest By Structural Distance	14.23	23.48	0.7	0	Didn't Try
Farthest By Earth Mover's Distance	2.19	2.18	0.47	0.68	81.06
Closest By Earth Mover's Distance	12.05	28.32	0.64	0.17	Didn't Try

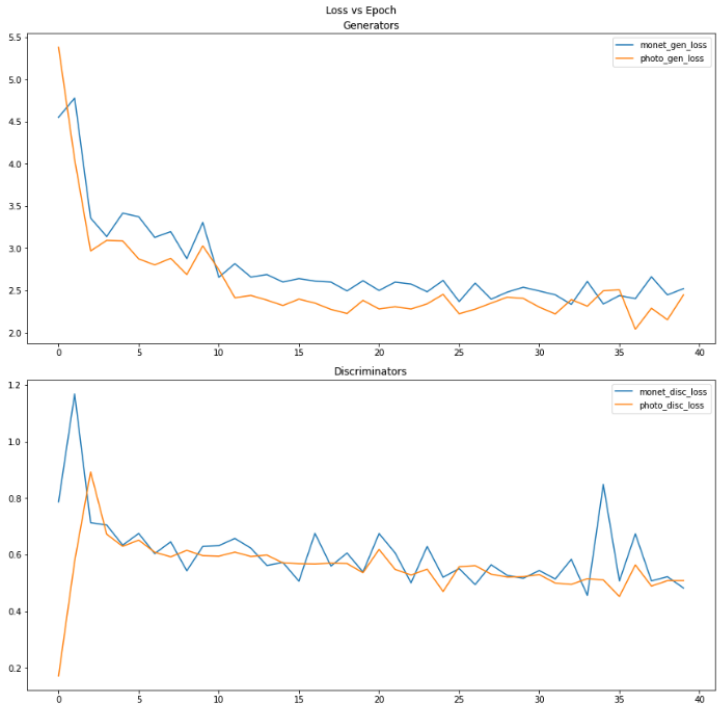
Notice that we tried the 'closest' version of each method in addition to the 'farthest' as a kind of sanity, to validate our assumption that a set of 30 very similar images will lead to poor performance of the train process, and we can see that the results are supporting this assumption. In addition, it was interesting to see that the earth mover's distance method worked poorly, and didn't produce a viable model. We ended up choosing the 'farthest by pixel distance' approach, because it yielded the best results, and the group of images that were selected by it looked very diverse for us as well. Here is a graph of the training process using this approach:



Experiment 3 - generator networks layout
In this experiment, we wanted to better understand the impact that the structure of the generator neural network has on the overall performance of the model. Notice that in order to keep this experiment focused, we decided to always use the same structure for both the network of the Monet and the Real photo generators. We tested 3 different setups for the generators' structure, and compared them to the baseline setup (which is the same as the selected setup from the previous experiment). The exact structure of each setup is defined clearly in the notebook in this (link) section. Here are the results that we got:

Generators network Structure setup	Monet Gen loss	Photo Gen loss	Monet Disc loss	Photo Disc loss	Kaggle score
Baseline	2.49	2.38	0.62	0.52	80.87
Thin (selected)	2.52	2.45	0.48	0.51	75.91
Wide	2.83	2.44	0.55	0.57	78.89
Deep	2.67	2.49	0.52	0.46	78.02

It was interesting to see that all the setups yielded better results then the baseline setup, which led us to believe that if we had more time we should've invented more time in this direction, and try other setups as well. We ended up choosing the 'Thin' setup, because it yielded the best kaggle score, and judging by the loss values it looks like it improved the discriminators' quality without hurting the generators' quality too much. Here is a graph of the training process of the 'Thin' setup:



Experiment 4 - discriminator networks layout

In a similar fashion to the previous experiment, In this experiment we wanted to better understand the impact that the structure of the discriminator neural network has on the overall performance of the model. For the same reason we had in the previous experiment, we decided to always use the same structure for both the network of the Monet and the Real photo discriminators. We tested 4 different setups for the discriminators' structure, and compared them to the baseline setup (which is the same as the selected setup from the previous experiment). The exact structure of each setup is defined clearly in the notebook in this (link) section. Please notice the distinction between the 'ThinNetwork' and 'WideNetwork' setups, which refer to the sizes (number of neurons) of the network layers, to the 'ThinReceptiveField' and 'WideReceptiveField' setups, which has the same size of layers as the baseline setup, but a different parameters for the convolutional operations (kernel size + strides), which effect the the receptive field of the Patch discriminator. In addition, notice that as required, all of these setups (including the baseline setup) have a receptive field different then 70x70. Here are the results that we got:

Discriminator network Structure setup	Monet Gen loss	Photo Gen loss	Monet Disc loss	Photo Disc loss	Kaggle score
Baseline (selected)	2.52	2.45	0.48	0.51	75.91
ThinNetwork	2.59	2.43	0.57	0.555	82.84
WideNetwork	2.4	1.98	0.53	0.64	77.27
ThinReceptiveField	2.27	2.18	0.56	0.68	77.51
WideReceptiveField	3.42	5.45	0.63	0.25	89.74

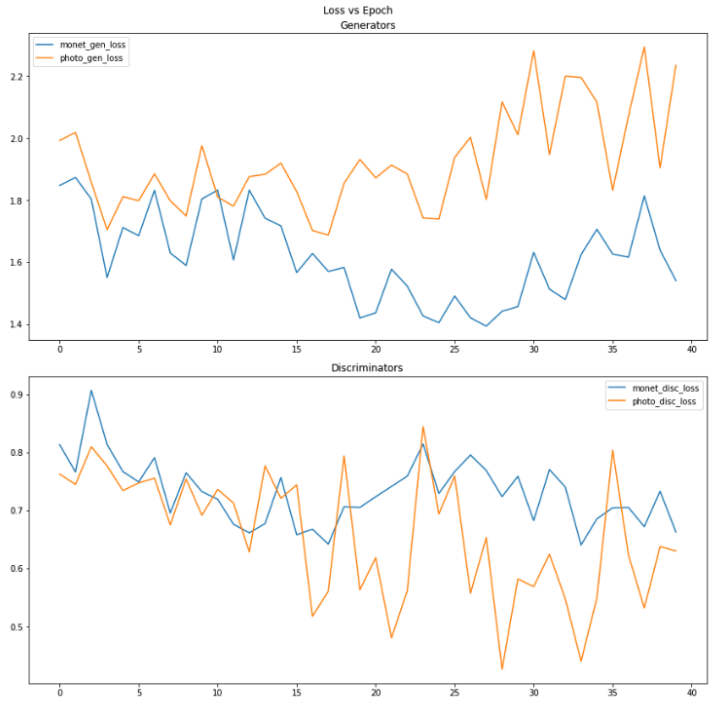
This time, none of the setups of this experiment yielded any improvement in comparison to the baseline setup when taking into account the kaggle score. It's hard to tell if there is an improvement of any setup when comparing its loss values to the baseline, but from that perspective there isn't a clear winner either. For these reasons, we decided to keep the baseline setup and not to change the structure of the discriminators.

Experiment 5 - usage of image augmentations to enrich the train dataset

The fact that we were allowed to use a small number of train Monet images motivated us to find ways to get the most out of each train image. In this experiment, we decided to use each train image several times, with a combination of random augmentations applied on it each time it's used. The random augmentations that we used were - random flip, random rotation of 30 degrees, and a random 'zoom out' (up to 15%). We weren't certain how these augmentations would impact the train process of this model (or on generative models in general), but we decided to try and were pleased with the results. In addition, we weren't sure if we should apply these random augmentations on the train Monet images, the train real photos dataset or both, so we tried all the combinations:

Monet dataset Augmentations	Photo dataset augmentations	Monet Gen loss	Photo Gen loss	Monet Disc loss	Photo Disc loss	Kaggle score
No	No	2.52	2.45	0.48	0.51	75.91
Yes (10 image repetitions)	No	1.97	1.98	0.72	0.6	66.22
No	Yes (No repetitions)	2.59	2.31	0.53	0.46	75.97
Yes (10 image repetitions)	Yes (No repetitions)	1.61	2.03	0.82	0.49	64.22
Yes (20 image repetitions) (selected)	Yes (No repetitions)	1.76	2.55	0.64	0.45	61.98

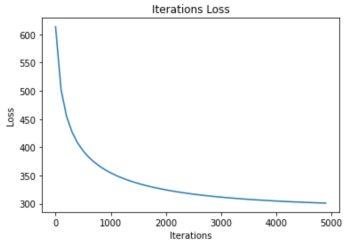
Notice that the augmentations usage on both datasets seemed to have a very positive impact on the training process. Furthermore, the major improvement was due to the usage of the augmentations on the Monet train dataset, which validated our initial assumption that led to this experiment. The best model from this experiment is the final model that we consider to produce the best result from this architecture. Here is a graph of his training process:



3.2 Architecture 2 - Neural Style Transfer

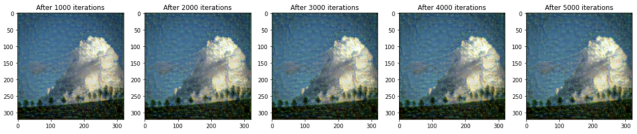
Neural style transfer is a technique that combines the content of one image with the style of another image. It requires using a pre-trained CNN to extract both the content and style features from the input images. We have used vgg19 as the pre-trained CNN

Experiment 1 - optimize iterations
In the first experiment, we wanted to understand how many iterations are required to converge the loss properly and provide a visually good-stylish image.



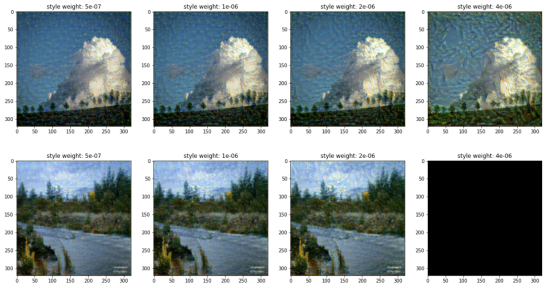
The loss plot shows that the model converges correctly.

We also printed the output image for every 1,000 iterations. We couldn't notice any significant differences between the images, so we decided to continue the rest of the experiments with 1,000 iterations (as it seems to be sufficient).

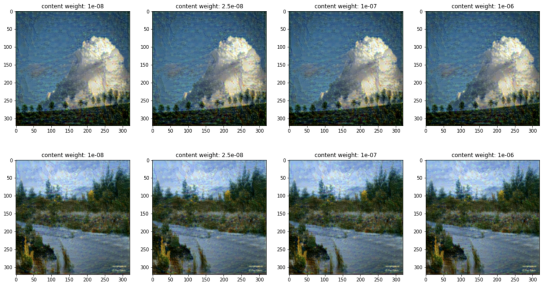


Experiment 2 - different loss weights
In this experiment, we wanted to test the visual effect of changing the weights of the style, content, and total-vairation losses. We tested each of the weights separately to understand its impact on the result image.

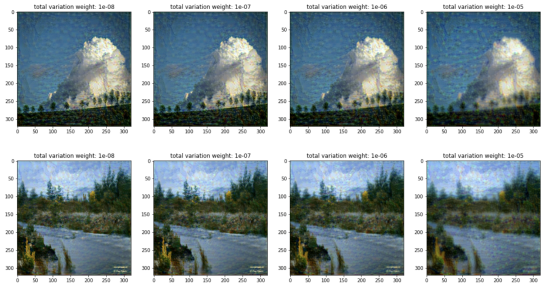
Style loss weight
We have examined different weights for the style loss. As expected, the higher the weight is, the more monet-like image we get. We also noticed that extreme values (compared to what we started with) often result in a black image. Selected value: 2e-6



Content loss weight
We expected to get a more realistic photo with greater weight values, but the observed differences were minor. Selected value: 2.5e-8



Total-variation loss weight
This loss is a regularizer component that favors the same color/value for adjusted pixels. As expected, the picture is more blurred with greater total-variation loss weight. Selected value: 1e-6



4 Discussion

First, we were able to build generative models that take real photos and produce monet-like images - which is very fulfilling and self assuring. One of the main challenges that we have encountered is the data. Not only the limited amount for monet-style images, but also that we didn't have trained data for pre-monet style and after-monet style. This, in our opinion, is the main reason why we were not able to train a Variational Autoencoder model with decent results. To overcome this, we used two unique modelling approaches. One is the cycle GAN and the other is the Neural Style Transfer. The cycle GAN fits this task as it doesn't require trained data with pre/post monet styling, and it also manages to fit on very small datasets. Augmentation is a key factor to improve the model results. The Neural Style Transfer is a unique approach that blends the content of one image with the style of another. The visual output of this model is remarkable. The downside of this model is that it requires significant resources to produce a styled image, therefore it takes a very long time to produce thousands of styled images (and this is the reason why not used this model in Kaggle competition). Another challenging aspect in this project was the lack of simply why to evaluate the improvements in our models. In order to handle it, we decided to treat the loss values as proxy for the expected performance of the final generative model, and frequently validated our assumptions by submitting our predictions in Kaggle and checking the updated score.

5 Code

- Here is a link to our train notebook in Google Colab
- Here is a link to our test notebook in Google Colab
- Here is a link to the Kaggle submission of our best model. Please notice that although there are a few submissions with a better score on our team, they are old test submissions and are irrelevant (for example, trains that used all the 300 Monent images). Unfortunately, there is no way to delete these submissions from Kaggle.