Advanced Topics in Deep Learning- Final Course Project

Albert Abiev Stav Elizur
Tommy Afek Daniel Oleynyk

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

1 Introduction

For the final course of the Advanced Topics in Deep Learning, we were asked to enter a Kaggle competition and present a solution for the style-image generating problem. We had to use the knowledge we accquired in this course, many experiments and research in order to solve our problem in the best way we can.

2 Problem

The style-image generating problem is a problem which we take 2 images, one for the base image and the second is for the style reference image and we need to generate an image which has the content of the base image and the style of the style reference image.

3 Data

Our data is a directory of base images and a directory of style reference images of the artist Claude Monet.

4 Solution

We presented 2 solutions to this problem: using the CycleGan model and the Neural Style Transfer technique.

4.1 Cyclegan

4.1.1 General approach

In order to solve our image-style generation problem we took different approaches, one of them being the cycle gan which we learned during the course.

The Cycle Generative Adversarial Network, or CycleGAN, is an approach to training a deep convolutional neural network for image-to-image translation tasks. The Network learns mapping between input and output images using unpaired dataset.

4.1.2 Design

To build our neural network, we used Python and Numpy together with the Keras library. The architecture of the CycleGAN model for the photo-to-Monet paintings task typically involves three key components: The generator network for mapping photos to paintings, which typically consists of several downsampling and upsampling blocks with residual connections to facilitate the flow of information. The discriminator network for distinguishing between real and fake paintings, which typically consists of several convolutional layers followed by a binary classification layer. For the discriminator, we used a PatchGAN architecture, which classifies image patches as real or fake, with a selected receptive field. The cycle-consistency loss, which penalizes the difference between an original image and its reconstruction after being translated and then translated back to the original domain.

Generator architecture:

We use reflection padding to pad the boundaries of an image or feature map by reflecting the data at the edges. It helps to preserve the spatial information of the input data and reduce artifacts in the output feature map.

In our generators we have one Reflection padding and then one convolutional layer with 64 filters and kernel size of 7x7 with relu activation function, and instance normalization layer that normalizes the activations for each sample independently. We use instance normalization and not batch normalization,because it normalizes the activation's of each pixel independently, thus preserving the spatial information. This is especially important in tasks such as style transfer, where the style of an image is determined by its texture and color, and the spatial information must be preserved. And now we have two down sample layers that first we double the number of filters and create the down sample layers, each down sample layer consist of one convolutions layer with kernel size of 3x3 with padding same and strides of 2x2 with Instance normalization and relu activation. After the down samples block we have nine residual block, each block has two block of reflection layer and convolutional layer with 3x3 layer and strides 1x1 with instance normalization the first block has relu activation, In addition we add layer between the input layer and the output layer.instance normalization normalizes the activations for each sample independently Now we need to implement two up sample layers like the down sample layers at the beginning of the model, but instead of add conventional layer, we add transpose conventional layer. At the end of the model we add one Reflection padding and then one convolutional layer with 64 filters and kernel size of 7x7 and padding valid with instance normalization layer and tanh activation function.

5 Experimental results

5.1 Cyclegan

5.1.1 Experiment 1

In this experiment, we want to check if we find receptive field different from 70x70 that claimed to be the best according to this article.

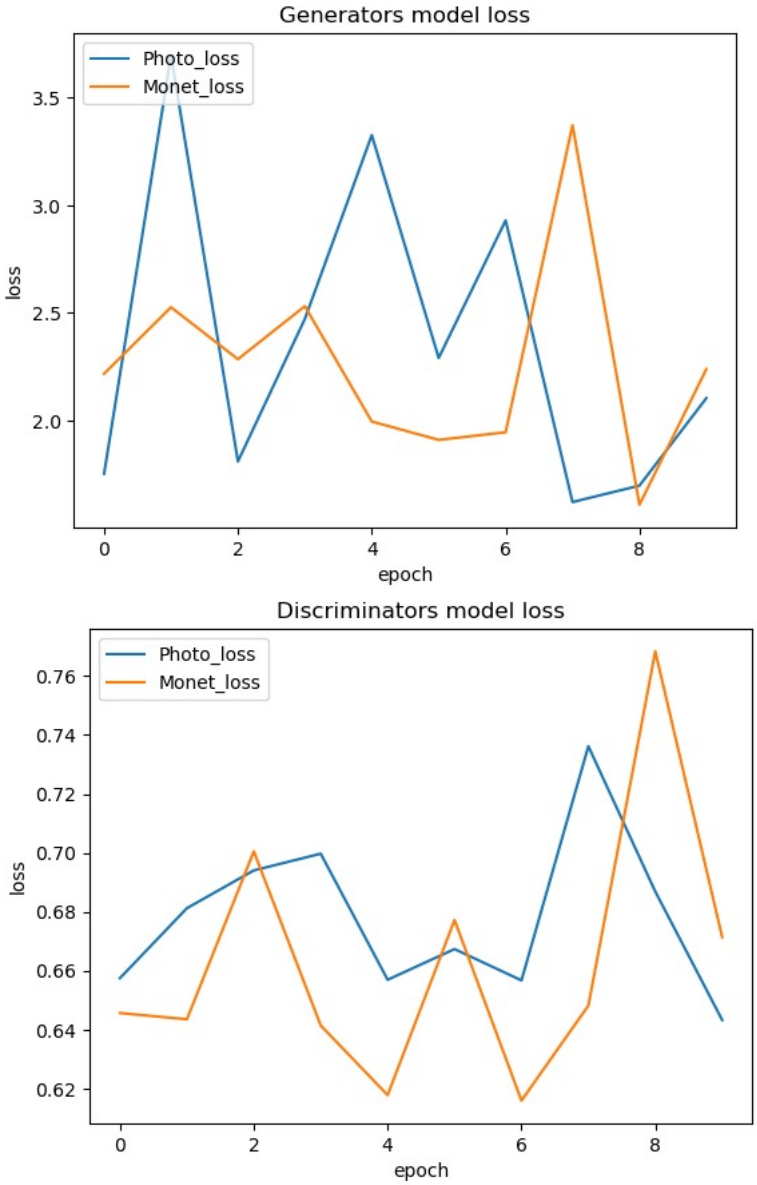
Hyperparameters:

Batch size: 1 Learning rate: 0.0001 Epochs: 10

Discriminator architecture:

3 layers of down sampling convolutional blocks that initialize with 8 filters and the next layers is multiplied by 2 with kernel size 2x2 and stride 2 and relu activation. Next we have output layer with filter value 1 and kernel size 4x4 and stride 1, and padding is valid. Note: The down sample is the same as the generator.

Graphs:



5.1.2 Experiment 2

In this experiment, we want to check receptive field 70x70 that claimed to be the best according to this article.

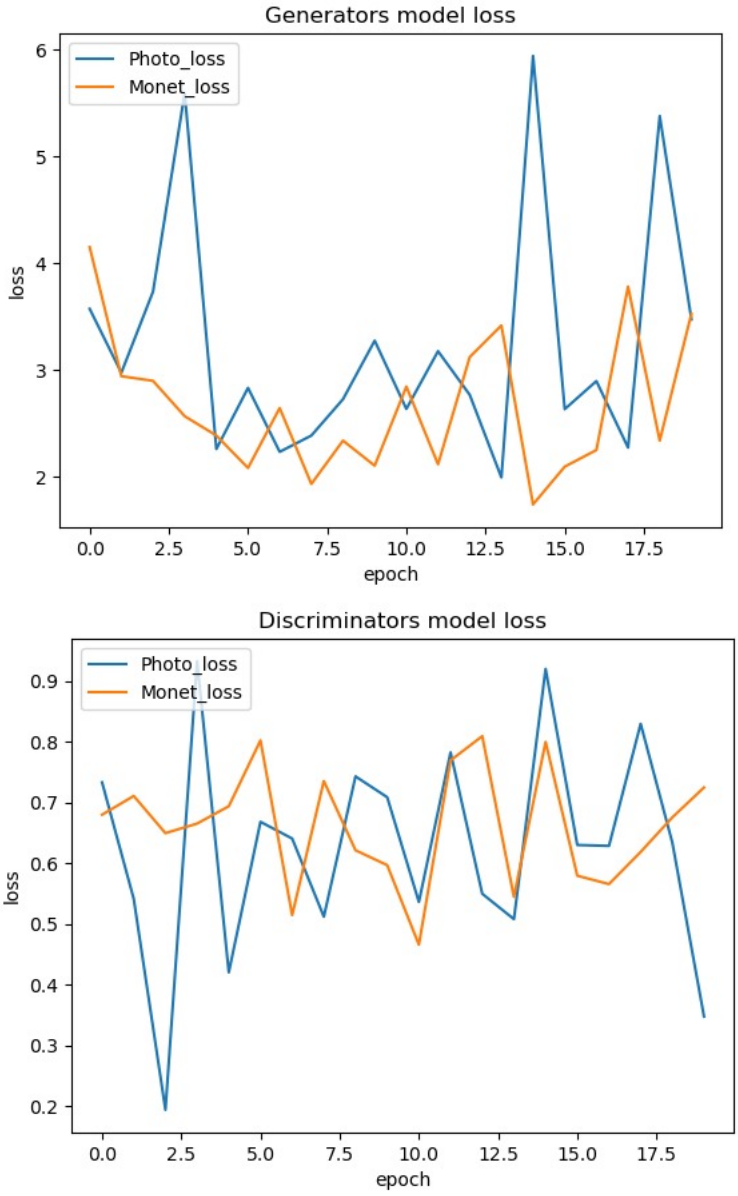
Hyperparameters:

Batch size: 1 Learning rate: 0.0001 Epochs: 20

Discriminator architecture:

4 layers of down sampling convolutional blocks that initialize with 64 filters and the next layers is multiplied by 2 with kernel size 2x2 and stride 2 and relu activation. Next we have output layer with filter value 1 and kernel size 4x4 and stride 1, and padding is valid. Note: The down sample is the same as the generator.

5.1.3 Graphs:



5.1.4 Best model

In this experiment, we want to check receptive field 32x32 that claimed to be the best according to this article.

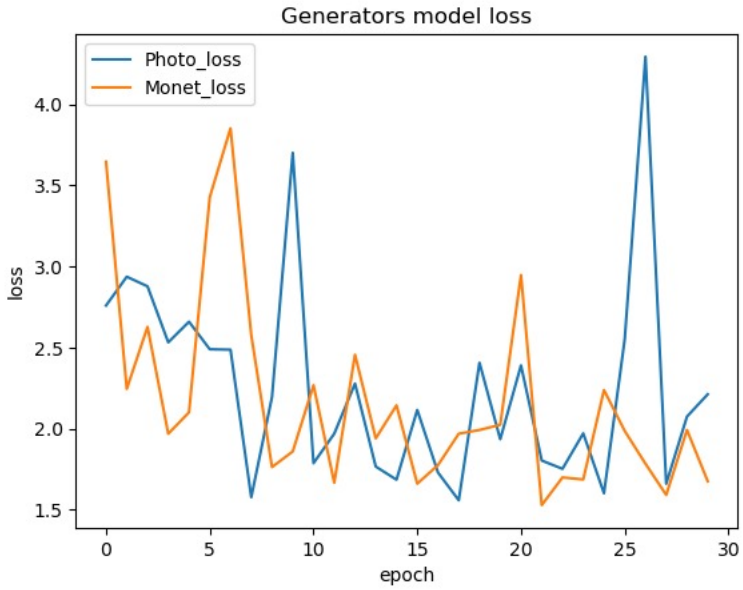
Hyperparameters:

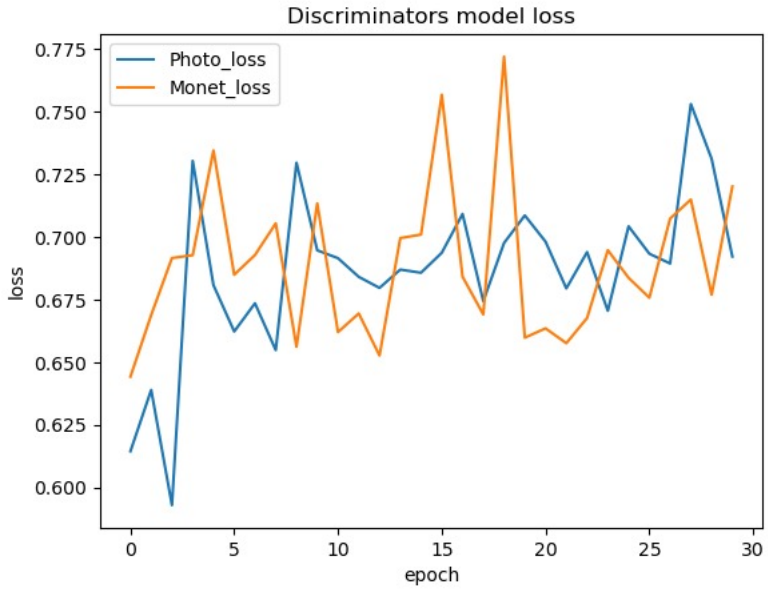
Batch size: 1 Learning rate: 0.0001 Epochs: 30

Discriminator architecture:

3 layers of down sampling convolutional blocks that initialize with 8 filters and the next layers is multiplied by 2 with kernel size 2x2 and stride 2 and relu activation. Next we have output layer with filter value 1 and kernel size 4x4 and stride 1, and padding is valid. Note: The down sample is the same as the generator.

Graphs:


Generators model loss
Photo_loss
Monet_loss
loss
epoch
4.0
3.5
3.0
2.5
2.0
1.5
0
5
10
15
20
25
30


Discriminators model loss
Photo_loss
Monet_loss
loss
epoch
0.775
0.750
0.725
0.700
0.675
0.650
0.625
0.600
0
5
10
15
20
25
30

6 Neural Style Transfer

6.1 General approach

In order to solve our image-style generation problem we took different approaches, one of them being the Neural Style Transfer which we learned during the course. Neural Style Transfer is basically an optimization technique that takes 2 images, one of which is the content image and the second, is the style reference image. This technique takes both images, and blends them together so the output image will look like the content image, but painted with the reference style we chose, which is the solution to our problem.

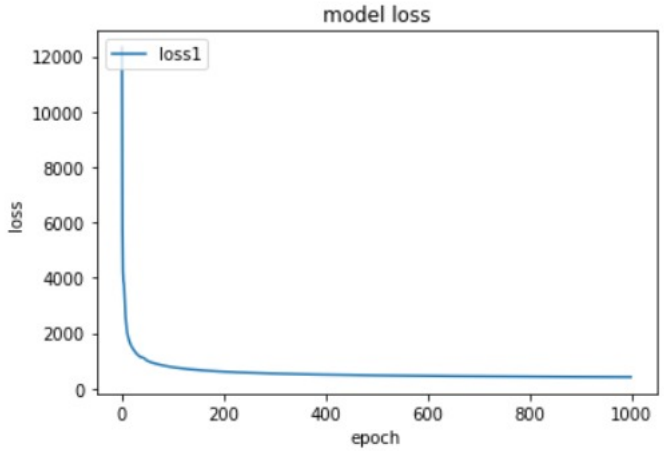
6.2 Design

We divided our code into the following sections: Fetch Data, Model Generation, Pre-processing, Loss Calculations, Training, Image Generation, Predicting, Test Environment, and 2 experiments. In the first section we first the data, from the "kaggle/gan-getting-started" we signed as a group using our user's API key. After we fetched the data we generate our VGG19 transfer learning model which takes the first 5 convolutional layers as styling layers because the first layers represent the "styling" of the image and also we take by default the second convolutional layer in the fifth block of the model. Then, we ran the content and style reference images with the pre-processing method which loads the image, resize it, and formats it into appropriate tensors we can work with. After we pre-processed the data, we train it in order to calculate the average loss for our Kaggle images training set. In the fit process, we compute the loss and gradients and mutate the original image until we finish the iterations. For the loss calculations we use the GRAM matrix, We calculate the style loss by multiplication of the Gram matrices of the style features and the combination features of each style layer we have, then we normalize it by using the style weights. For the content loss we sum all the squared values of the substraction between the base image features and the combination features to evaluate how much did the content of the base image change during the iteration. we add both of the losses together and we add to it the total variation loss which is responsible of keeping the generated image locally coherent. Then we have the compute loss and grads function which generates the gradients by the total loss we calculated and the combination image and we use the SGD optimizer to apply those gradients into the combination image to modify its actual pixels. Then we show the loss per iteration as a graph and we continue to the predicting section. In the predicting section, we use the Image Generation function which takes a base image, a style image, max iterations, and callback parameters(patience, delta) and generates an image, deprocesses it, saves it, and returns its file name. This function runs the preprocess function on the base and style image and runs over the max iteration parameter or until we early stop which happens after we stop reducing the loss by the number of patience we defined with the delta we have. After that, the predict calls a display function that displays the original

content and style images with the output of the generated image. The prediction function continues to run with random-style images until we finish running all the images we have in the Kaggle competition or until the limit we define. Then, we have the Test Environment which generates images with the images the user uploads over random style reference images. Then, we have the 2 experiments, one is mainly focused on the hyperparameters, and the second one is mainly focused on the model architecture.

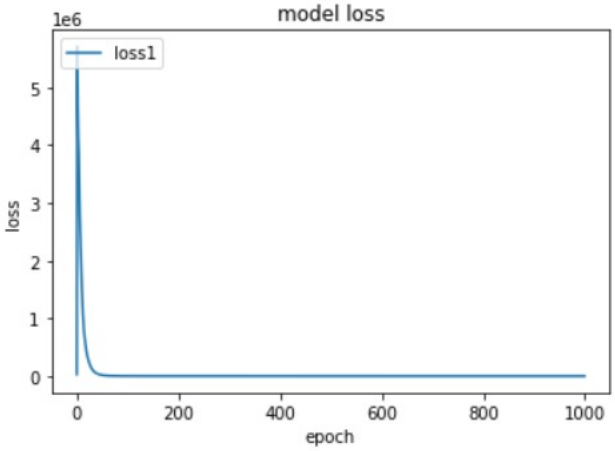
6.3 Architecture

Our model architecture involves the VGG19 transfer learning model. VGG19 is a pre-trained convolutional neural network that is trained on more than a million images from the ImageNet database. This model classifies images into over 1000 different classes. We use this model because it was trained for many images and its first layers represent the best styling of our input image. We take the content layer of the last style layer which has the "content" we need to mutate in the style neural transfer algorithm.



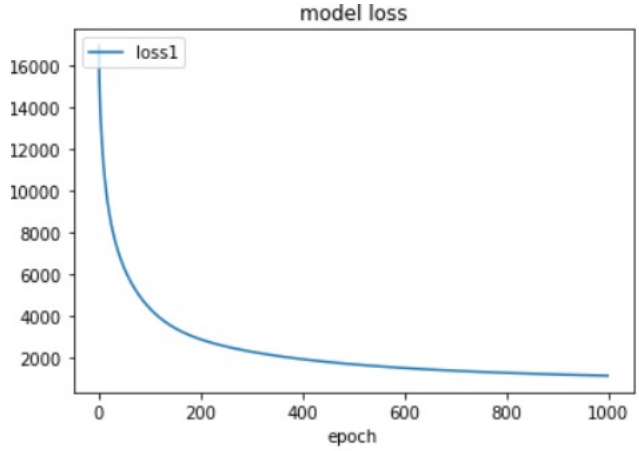
6.4 Experiment 1

In this experiment, we focus more on the model hyperparameters and more on the logic of the Neural Style Algorithm. So, we create our model with a different optimizer. In this experiment, we used the Adam optimizer. Adam optimizer is the extended version of stochastic gradient descent which is implemented by many neural networks. Also, we change the number of iterations in order to minimize the loss in the training/fit function. The value we had for the number of iterations was 600.



6.5 Experiment 2

In this experiment, we focus more on the model architecture. We give different style layer names and a different content representation layer of our image. The main idea of this experiment was to understand better the changing of the layers we choose from our VGG19 model in the algorithm and how it affects the model's loss and training time. We found it best to have the default layers we chose.



7 Discussion

After implementing the Neural Style Transfer solution for our image-style generation problem we learned that we can get good results using deep learning methods instead of using GAN models. We thought that CycleGan model was

the best solution for our problem because its main goal is to generate images from an input and after a long time of training we would have amazing results for an image. In the beginning of the implementation cycle gan model we faced with the main idea of the architecture, especially the layers that create a shortcut in the model. We know neural networks are universal function approximators and that the accuracy increases with increasing the number of layers. But there is a limit to the number of layers added that results in an accuracy improvement. So we decided to implement our generators with residual block, that are skip-connection blocks that learn residual functions with reference to the layer inputs, instead of learning unreferenced functions. The Neural Style Transfer technique was a little bit hard to understand because of its complicated loss function but after a research we were able to understand it and then we ran many experiments such as different content and style reference layers from the VGG model and different optimizer and weights for the loss function. After we chose to best architecture, we results similiar to our CycleGan model. Also, the generating time was much shorter than the CycleGan because we also implemented our own early stopping method which stopped the iterations of the generating process after the loss had stopped improving. Because of that we were able to run many different experiments which we explained above to have better results for many images.

8 Code

Colab project Colab Test project