Deep Learning- Final Course Project

Kanna Nahir 211319876, Shachar Hananya 324165133

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

1 Introduction

In this project, we build generators models that generates Monet-style images using 30 images.

We are about to chose the 30 images for the training, and to build two deep-learning models that use those images for generate new Monet-style images.

We will explain the models' architectures, report and analyze their performances, and finally discuss the results we found.

2 Related Works

2.1 Generative Adversarial Networks

Generative adversarial networks are a kind of artificial intelligence algorithm designed to solve the generative modeling problem. The goal of a generative model is to study a collection of training examples and learn the probability distribution that generated them. Generative adversarial networks are based on a game, in the sense of game theory, between two machine learning models - a generator and a discriminator.

The generator model is trained to generate new examples, and the discriminator model tries to classify examples as either real (from the domain) or fake (generated). The two models are trained together in a zero-sum game, adversarial, until the discriminator model is fooled about half the time, meaning the generator model is generating plausible examples.

2.2 Cycle-Consistent Adversarial Networks

Image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs. However, for many tasks, paired training data will not be available. CycleGAN is an approach for learning to translate an image from a source domain X to a target domain Y in the absence of paired examples.

The model includes two mappings G : X → Y and F : Y → X. In addition and two adversarial discriminators D_X and D_Y , where D_X aims to distinguish between images x and translated images F(y); in the same way, D_Y aims to discriminate between y and G(x). The objective contains kinds of two terms: adversarial losses for matching the distribution of generated images to the data distribution in the target domain and a cycle consistency loss to prevent the learned mappings G and F from contradicting each other.

2.3 Neural Style Transfer

Neural style transfer is a deep learning technique that has gained significant attention in recent years due to its ability to generate visually appealing images by transferring the style of one image onto the content of another. This process involves training a neural network to extract features from both the content and style images separately, and then combining them to generate a new image. The process involves optimization of a loss function that balances the content and style reconstruction of the generated image, thereby ensuring that the final image preserves the content of the original image while adopting the style of the style image.

3 Data

The dataset contains two types of inputs: 300 Monet painting images and 7028 photo images. Both photos and paintings are of 256X256X3 size, and are represented in an RGB format.

Figure 1: Data instances, the first row is Monet paintings while the second is photo images

3.1 Choosing Monet Paintings

In order to choose 30 Monet paintings out of the given 300 ones, we aimed to select such paintings that would yield the most varied set for the training process. This goal was motivated by the claim that assuming all paintings have a similar base "style" (since they were all painted by Monet), the more the set is varied, the lower the chance that the model will turn into a mode collapse.

To achieve this goal, we followed the following framework:

1. Represent the paintings in a lower dimension.
2. Cluster the lower-dimension image embeddings into groups of similar paintings.
3. Define a measure of how much a given painting belongs to its cluster and choose the ones that least belong.

3.2 Selection Framework Implementation

Each part of this general framework described above has its own practical implementation method:

3.2.1 Lower-Dimension Embeddings

The first stage, of reducing the 256X256X3 images into lower, more suitable for comparison calculations, was done using the Principal Component Analysis (PCA) algorithm.

PCA is a statistical technique that identifies patterns in data and expresses them as linear combinations of the original variables.

Then, we had to find a proper size for the embedding space, which is, on the one hand, as lower as possible for efficient and well-utilized embeddings, but on the other hand, still captures most of the information and minimizes the loss of data.

To solve this, we iterate over all possible embedding sizes from 1 to 300, and measured the variance captured by the embeddings of each size. We finally chose the minimal embedding size that is able to store at least 90% of the variance - which is **134**.

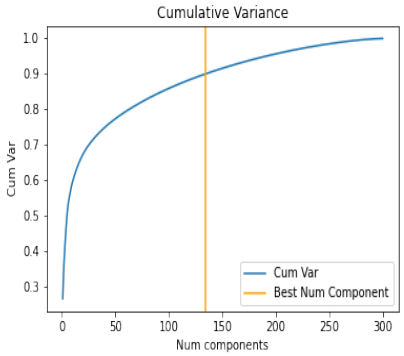


Figure 2: Varaince stored by each embedding size using PCA algorithm.

3.2.2 Clustering

Given the 134-D embeddings for the paintings found in the previous stage, in the current one, we cluster those embeddings, aiming to find groups of similar paintings from the given pool.

The clustering method used for this task is K-Means. K-means is a well-known clustering algorithm, that aims to group similar data points together based on their distance from each other. The algorithm works by first randomly selecting K initial cluster centers and then iteratively assigning each data point to the nearest center and updating the center's position to the mean of the assigned points.

One limitation of this algorithm is that the number of clusters has to be pre-defined. Therefore, to overcome this disadvantage, we iterated over all the possible values from 2 to 30, performed the K-Means algorithm using each of those values, evaluate the algorithm performance, and finally find the most appropriate number of clusters.

The performance evaluation was done by a measure called "Silhouette Score" (will be described in the next stage), and the final choice was done by a method named the "Elbow Method" - which is a visual way that is looking for the "elbow" point in the plot of the evaluation measure vs the number of clusters. In our experiment, we found it to be **5**.

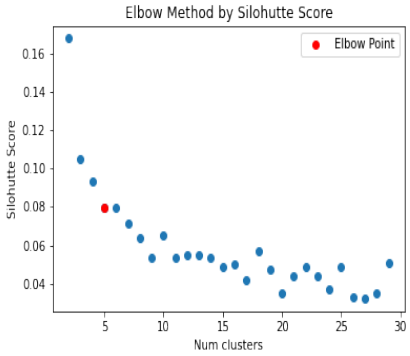


Figure 3: K-Means performance for each of a various number of clusters

3.2.3 Paintings Selection

The final stage of the selection process is to rank the clustered paintings and find those that yield the most diverse set. To do so, we used the "Silhouette Score" measure, commonly used for clustering evaluation. This score, calculated for each data point in a given clustered set (a score whole set is obtained by averaging the scores of all its data points), measures how well it belongs to its cluster, rather than the other clusters. For the final choice of paintings for the training data, we chose the paintings with the highest silhouette scores in the training set - that is because this approach selects the paintings that are very close to their clusters compared to the other clusters, which leads to a highly diverse training set, and achieves the goal we aimed to.

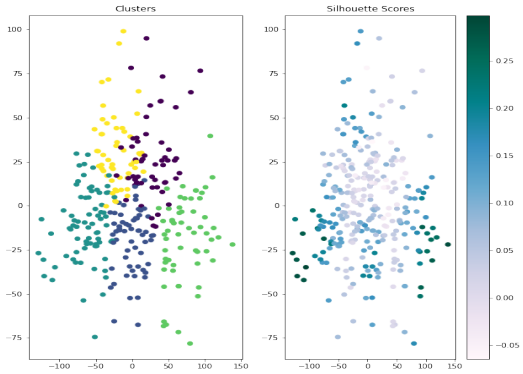


Figure 4: The paintings' embeddings space colored by: the clusters - on the left, and the silhouette scores - on the right

4 Solution

4.1 General approach

Our mission is to add Monet-style to photos images - which we can't do (at least, naturally), using classic GANs or VAEs. In addition, we do not have ground truth to those photos - a pair of a photo and its Monet-style painting, therefore method like pix2pix is irrelevant. Instead, we can use a CycleGAN architecture, which only requires images from both domains - without explicit cross-domain ground truth. Moreover, once the restriction of using a pretrained model was removed, we can also use a Neural Style Transfer architecture, that can make use of the large pretraind VGG19 model's feature maps for solving the problem.

4.2 Design

During the work on this project, most of the time was spent on finding the most appropriate architectures, hyper-parameter tuning, and generally trying to exploit the best results in relation to the time and resources we had for this project.

4.2.1 CycleGAN Architecture

The first model we implemented was a CycleGAN architecture. Our generator has similar characteristics to both U-Net and DCGAN - it also compresses the original image using 3 convolutional layers with kernel sizes [4, 2, 2] and strides [4, 2, 2] respectively, and symmetrically returns it to the original size using 3 deconvolutional layers with kernel sizes of [2, 2, 4] and strides of [2, 2, 4]. Batch normalization layers were also added after each conv / deconv layer, each followed by a LeakyReLU activation function (except for the last deconv layer, which was followed by a TanH activation). The discriminator of this model is a type of PatchGAN. It consists of 5 convolutional layers of kernel sizes [4, 2, 2, 4, 4], strides [4, 2, 2, 2, 1], and batch normalization layers after each of the three middle layers. In addition, each layer was followed by a LeakyReLU activation, except for the last, where a Sigmoid was used (for the prediction).

Regarding the hyper-parameters, we used Adam optimizers for training both the generators and the discriminators, with learning rates of 1e-3 and 1e-4, respectively. The batch size was set to 10. In addition, regarding the specific model hyper-parameters, we used λ_{Cycle} of 10, Binary Cross-Entropy for the GAN losses, and Mean Squared Error for the cycle loss. We add also another component to it generator loss, besides the GAN and the cycle losses, which encourages the images in each domain not to change while passing through the generator of the other domain. This loss component, also called "identity loss", was calculated as an L1 loss, with a coefficient λ_{ID} of 0.5.

Moreover, we applied some adjustments that we found helpful for the model convergence. Those included normalizing the input to the range -1 to 1 and

using soft labels while training the discriminator. We also determined a policy that states that at the 15 epochs, only the discriminator is trained, and then it is only trained once in 10 epochs. The generator, on the other hand, is trained only when the discriminator doesn't.

4.2.2 Neural Style Transfer Architecture

The second model we chose to implement is Neural Style Transfer. This model utilizes the pretrained VGG19 model feature maps, using its capability to store "style" characteristics at the beginning of the network and "content" once towards its end. It updates the generated image iteratively while trying to capture the content of the first input image and the style of the second one. More precisely, the output image is initialized as the content image, and in each epoch, the model extracts the feature maps of both the first 3 and last 3 convolutional layers of the pretrained VGG19, and treats them as style and content features respectively. It then aims to minimize the MSE loss between the content features of the content and the generated image, while also minimizing the MSE loss between the style features (after a transformation called the Gram Matrix) of the style and the generated image. The optimization is done using Adam optimizer with a learning rate of 0.01. In addition, the proportion between the content loss to the style loss was set to $1e-6$.

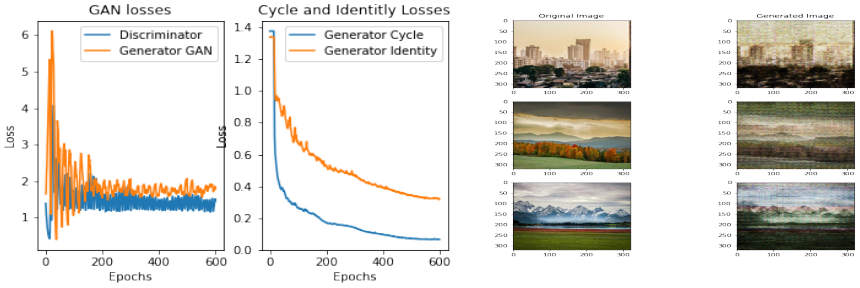
Although this model is considered robust and can obtain high-quality results, it is has a significant limitation compared with models like CycleGAN, which is its non-scalability - this model is trained for a specific pair of content and style images, and not on a pool of images. Therefore, the testing phase of this model requires a new training phase, done with the specific pair of images to be tested.

5 Experimental Results & Discussion

First of all, before we explain the experiments we have done during the process of this project, here are some examples of our final results:

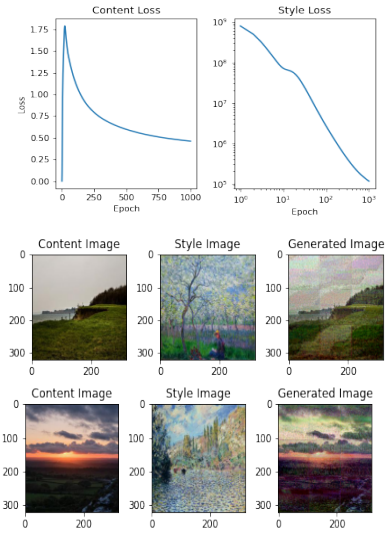
5.1 CycleGAN Results

Here are the convergence graphs, as well as some examples for the CycleGAN model performance:



5.2 NST Results

Here are the convergence graphs, as well as some examples for the Neural Style Transfer model performance:



5.3 First Experiment: Augmentation

The first experiment we have done is related to the use of augmentation in the first model training process. We were motivated to check this because of the restricted number of Monet paintings, thinking whether an augmentation of rotating the image by 90 degrees, which still keeps the Monet style in it, can enrich the training data.

To check it, we trained the first model twice, with an without the augmentation, and compared the results. Here are them:

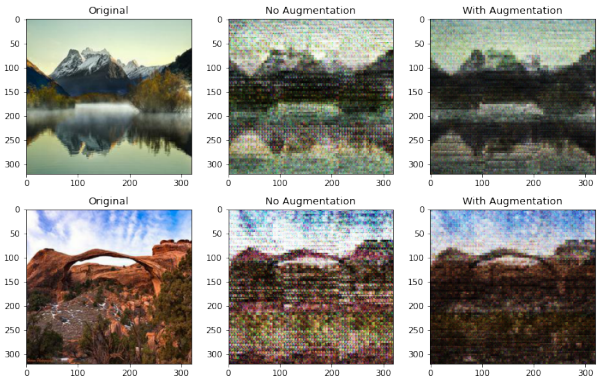


Figure 5: First Experiment Results

From the visual evaluation, it seems like the augmentation does have a slight effect on results, making the generated images "smoother" and less distorted. We also calculated the FID score for both of the models: The model with the augmentation gets a 128.81 FID score, compared with 143.14 without the augmentations.

5.4 Second Experiment: Lambda Cycle

The second experiment also relates to the CycleGAN model, and it's goal is to check the effect of the λ_{Cycle} hyperparamter on the convergence of the CycleGAN framework. The λ_{Cycle} hyperparameter, as mentioned earlier, is the coefficient of the cycle loss in the generator loss calculation.

Here are the results:

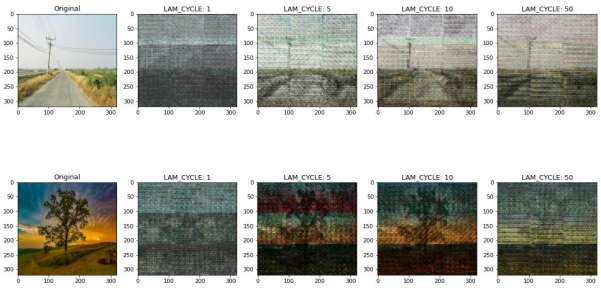


Figure 6: Second Experiment Results

We can notice that the λ_{Cycle} has a sagnifiacnt impact on the model results, and it has an optimized point which is not to low, so the model can still return

to a similar image, but not too big as well. In this experiment, the best value is $\lambda_{Cycle} = 10$. It also reflects in the FID score, where the score were: [255.3, 152.2, 127.02, 128.2] for λ_{Cycle} of [1, 5, 10, 50], respectively.

5.5 Third Experiment: Edge vs Mid Convolutional Layers

The last experiment, is now regarding the NST model. As described before, the Neural Style Transfer framework uses the earlier and latter feature maps extracted from the VGG19 model, as an indication for "style" and "content", respectively. This experiment is about checking whether the definitions of "earlier" and "latter" layers should be treated as "the very first" and "the very last" layers or a softer meaning of "start-mid" or "mid-end" layers is more appropriate. We therefore trained two versions of the model: one that takes the [1, 2, 3] conv layers for style and [14, 15, 16] for content, and another one that uses the [4, 6, 8] for style and [10, 12, 14] for content. We compared the results:

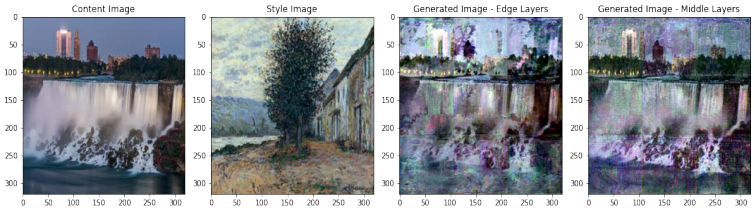


Figure 7: Third Experiment Results

From the visual aspect, it seems that the model indeed captures different "levels of style" and "content". While the very first and last conv layers yields "messy" and "aggressive", the use of the middle layers outputs smoother and cleaner results. Therefore, we chose the middle layers as the more appropriate for the NST model.

6 Code

Attaching the links to the notebooks:

- Train: Train Notebook
- Test: Test Notebook