Monet-Style Image-to-Image Translation with CycleGAN

Rohit Arunachalam
rarunachalam@gatech.edu

Sharan Sathish
ssathish6@gatech.edu

Eshani Chauk
eshani@gatech.edu

Harsha Karanth
hkaranth3@gatech.edu

Georgia Tech
801 Atlantic Dr NW

Abstract

Many people research the most efficient, lightweight methods for style transfer as new models enter the DL field. The traditional approach takes paired images and learns how to transfer from one image to another, but this becomes infeasible as image collections grow in size and diversity. Thus, we present a deep dive into Generative Adversarial Networks (GANs) for image-to-image translation. First, we looked at DCGAN for identifying patterns from an input dataset and training a GAN to generate images that fit in the training dataset's style, which is Monet-style images for our project. Then, we researched CycleGAN to take an input image and transfer its painting style of Monet onto the input image. From our experiments, we found that CycleGAN performed exceptionally well in terms of the Memorization-informed Fréchet Inception Distance (MiFID) score. The images generated in the Monet style also looked plausible to the human eye, which is an important factor since we as humans are the consumers of this content and we should be able to believe the results.

1. Introduction/Background/Motivation

1.1. Introduction

Claude Monet is one of the world's most distinguished and successful painters. He has a unique style to his paintings that combines Impressionism with Modern Art. Monet's paintings have a special type of vibrancy, and he manages to capture the emotions and liveliness of a scene. Wouldn't it be amazing if we could paint just like Monet, even if we are not the best artists? Thus, we attempted to generate Monet-style paintings by applying Monet style to normal images. We were interested in this problem because we were curious about another method of performing style transfer, which is a technique in which we can change a picture to match the artist's style of another picture. In Project 3, we learned that we can modify an image to match the style of another image by finding the differences between the content and style of each image and then changing the pixels of a new image to combine the content and loss. In this project, we wanted to find another method of performing style transfer by providing images of Monet images and expecting Monet-style pictures when we give a non-Monet-style painting. This project was inspired by the "I'm Something of a Painter Myself" Kaggle competition.

1.2. Related Works

The most straightforward method of image-to-image translation is to train a model to learn the mapping between pairs of input and output images. Currently, Pix2Pix [2] utilizes a conditional GAN architecture to learn how to translate an input image to a paired output image. Normal GANs are capable of classifying whether an image is real or fake based on the input dataset while simultaneously training a generative model to create images that can pass off as real. Conditional GANs work similarly except that they are conditioned on an input image so that they generate a corresponding output image, thus allowing us to do paired image-to-image translation by learning the mapping from input to output image. The problem with this approach is that obtaining paired data is difficult and expensive, and sometimes even impossible when trying to learn with historic images of people or objects that do not exist anymore.

As a result, unpaired image-to-image translation became a popular problem in the field. This would allow us to simply provide the model with a collection of input images and another collection of output images, and there is no explicit mapping from input to output required. One proposed so-

lution is DiscoGAN [5]. This model couples 2 GANs together that learn the mapping from one domain to another and also the reverse mapping. DiscoGAN relies on reconstructing the original image to understand the model's accuracy. It uses reconstruction loss to measure how similar the input image is to the image generated from passing the input through both GANs and does the same for the reverse mapping. The issue with DiscoGAN is that while it succeeds at converting the input to output and back to the input, it seems to fail on a simple input-to-output problem.

1.3. Motivation

We want to tackle image generation and translation because it has multiple positive impacts on society. Tools using image generation can help level the playing field in visual creativity, allowing anyone – regardless of economic background or artistic skill – to produce high-quality images, granting everyone the capability to artistically express themselves. Tools can help improve communication as well; complex ideas can now be conveyed through images, which is useful when there are lingual barriers.

Additionally, we attempt to create models that are not too complicated. Lightweight models further improve on the existing benefits of image generation with faster turnaround rates. Although state-of-the-art models are extremely impressive with their recognition and depth in a large number of categories, they require an abundance of computational resources and time. Lightweight models provide many of the same benefits but require far less computational power. Iterations can be made faster, and with many lightweight projects being open-source, models are accessible for experimentation to more people.

1.4. Dataset

We participated in the "I'm Something of a Painter Myself" competition on Kaggle, and we got our dataset through Kaggle. The competition gave one directory with normal images in jpg format and another directory with Monet images in jpg format. They also gave another two directories with the same normal and Monet images in TFRecord format. This format is used to store the sequence of binary records and optimizes the access time of data stored in the format. Additionally, this format is integrated into the Tensorflow data API and we can do operations like batching on the files.

More specifically, our data is focusing on landscapes and man-made structures like buildings. This is because Monet focused his artwork on these two topics. There was no label or target in the dataset. Instead, we will compare our output image to the input image of our model and compute the difference. Specifically, for the Cycle-GAN, we will provide a normal image to the first GAN that generates an image in the Monet domain. We will send this output image to the second GAN that takes in a Monet image and then generates a normal picture. We will compare this output normal image to the original input of the first GAN, which creates the cycle of learning the relation between the two domains.

The most important aspect of our dataset is how vivid the Monet pictures are. This causes our model to create bright and vibrant landscapes or building images similar to the style of Monet. One limitation of this dataset and Monet's paintings, in general, is that he did not focus on painting people. As a result, our dataset does not have Monet paintings of people, so if our model receives a normal picture of a person, it may not accurately output a Monet painting of a person.

2. Approach

As mentioned earlier, paired image-to-image translation proves to be a difficult task due to the lack of large amounts of paired image data. We decided to implement a model similar to CycleGAN and experiment with this model to allow us to perform unpaired image-to-image translation. All of our models are implemented with TensorFlow.

2.1. GANs in General

First, we must understand the general concept of GANs. Generative Adversarial Networks (GANs) [1] are a zero-sum, adversarial game between the generator and discriminator where the generator learns how to make better fake images in the output domain while the discriminator learns how to become better at catching fake images. The generator randomly selects a noise vector from the latent space and upsamples it to generate an image, and it is trying to make this generated image match the style and content of those in the input domain. The discriminator takes an image from the input domain along with the generated image and tries to distinguish the real image from the fake image. The objective of GANs is only to generate images in a specified style.

Deep Convolutional GANs (DC-GANs) are a type of GAN that uses convolutional and convolutional-transpose layers in the generator to create an image. We first used this model to test if a GAN can create Monet images, but it only solves the problem of image generation. We define our generator as a series of Dense, Conv2DTranpose, and LeakyReLU layers. The discriminator is created with Conv2D, BatchNormalization, LeakyReLU, Dropout, and Dense layers. Our DCGAN model takes a random noise vector and uses the generator to generate a fake image. Then, it goes to the discriminator to evaluate how close the real and fake images are based on how well it can identify the fake image. After this step, the model computes generator and discriminator losses, computes gradients for both, and updates optimizers for the generator and discriminator. The loss for the generator is a binary cross-entropy loss on

the fake image, and the discriminator's loss comes from a sum of binary cross-entropy loss on the real and fake images. We have opted to use Adam optimizers with a learning rate of 0.0004 for the DCGAN model because they have fast computation time and do not need many parameters for tuning.

2.2. CycleGAN Architecture and Characteristics

For our image-to-image translation task, we implemented a model that performs similarly to CycleGAN [6]. CycleGAN adopts a similar architecture to normal GANs except that it has GANs within the larger model. It also requires a set of two image collections, as it learns the relation between these two domains. The first GAN of CycleGAN takes an image in the first collection and tries to generate an image that could be categorized into the second domain. The second GAN takes an image from the second collection and attempts to create an image that can be classified as the first domain. To ensure that this model is truly learning the relationship between both collections, CycleGAN implements cycle consistency. This process consists of using the image outputted from the first GAN as input into the second GAN and checking that the output from the second GAN matches the input of the first GAN. By creating cycle consistency, we can see if the generated images are able to cycle back to the original image we were trying to modify. Our model is an implementation of CycleGAN from scratch based on the CycleGAN Paper [6] and Kaggle competition tutorial [3] with some experiments of our own, which are highlighted in the Experiments and Results section, to improve our results. While these resources were helpful to get started, they did not explain many of the decisions made so it was important to fully research and understand the choices before implementing them in our own model.

The structure of our problem was to allow users to transfer the style of Monet onto their own images. Our data reflects this structure – the first collection contains any images that users can input, and the second collection of images is a set of Monet paintings since this is the style we want to transfer. These JPEG images are converted into tensors to be used by the model, and we make sure to keep them in their respective collections. We define the CycleGAN to have two generators and two discriminators, one of each for the normal photos and another of each for the Monet paintings. The generators are composed of multiple downsamples followed by a series of upsamples where the downsample layers are a combination of Sequential, Conv2D, and LeakyReLU, and the upsample layers are Sequential, Conv2DTranspose, InstanceNormalization, and ReLU. We create our discriminators with RandomNormal initializers, Conv2D, InstanceNormalization, and LeakyReLU. We manually define our initializers for better results. Since both of these elements contain Convolution layers, they will have learned parameters that are modified by the loss and gradient updates.

The first generator will take a real photo from the first collection and attempt to generate a fake Monet painting. Then, for the cycle aspect, we run the fake Monet photo into the second generator to get an image in the normal photo collection. Similarly, we follow a similar process but give a Monet image to generate a normal photo, then use that normal photo to generate a cycled Monet image. After the image generator steps, our CycleGAN implementation will use discriminators to check if the fake images can be classified into their target collections. Next, the model computes generator loss, cycle consistency loss, and discriminator loss. The generator loss is a BinaryCrossentropy Loss, and the discriminator loss is the sum of two BCEs for the real and generated images. The cycle consistency loss comes from the mean of the magnitude of difference between the original image and the cycled image which should be a reconstructed version of the original one.

The generator loss function measures how well the generated image can pass off as the specified domain, and the discriminator loss function measures the similarity between the generated image and the ground truth image. The cycle consistency loss shows us how well the model does at reproducing the input image after passing it through two generators. Based on these loss values, we update the gradients and optimizers for both generators and discriminators. We implemented Adam optimizers and experimented with its learning rate hyperparameter, which is explained more in-depth later in the paper.

2.3. CycleGAN Implementation Issues

Some of the problems we anticipated before the project were relatively high training times with variational encoders and later on possible training problems with generative adversarial networks. When we began with training our variational autoencoder for dog image generation, we found with the limited computational power we had training times were incredibly long by epoch.

This did not leave much space for experimentation because it became infeasible to train even a single model in a reasonable amount of time. Due to this issue, we switched to using generative adversarial networks, which we actually did not find to many difficulties with during training.

An unexpected issue we did encounter was problems with the layers of the GAN. After creating a baseline CycleGAN, we tried using ReLU layers. The issue we did not anticipate came from changing the final layer of the generator to be a ReLU layer, as opposed to a Tanh layer. This severely hindered the performance of this model. The reasoning for this seems to come from the output range of the

ReLU function compared to how images are normalized.

$$\text{ReLU} \in [0, \infty) \qquad \text{where as} \qquad \text{image} \in [-1, 1]$$

Since the ReLU function does not conform to the normalization used for images, the values become nonsensical, reducing the model's performance.

3. Experiments and Results

There were a few approaches we used to determine relative success. By the nature of image generation problems, we can self-determine to some extent whether a generated image is a failure. When the image looks nearly nothing like the expected output, we know the model failed. To be more deterministic, however, there are some measures we used.

3.1. MiFID as Measure of Success

We submitted our kernel to Kaggle for this competition and received a Memorization informed Frechet Inception Distance (MIFID) score as part of Kaggle's evaluation [4], which incorporates a memorization term with an FID score. The Frechet Inception Distance (FID) score evaluates the quality of the generated images compared to the real images. The calculation involves taking the difference between two multivariate Gaussians

$$\text{FID} = \|\mu_r - \mu_g\|^2 + \text{Tr}\left(\Sigma_r + \Sigma_g - 2\sqrt{\Sigma_r \Sigma_g}\right)$$

where μ and σ represent the mean and covariance, respectively, of the real r and generated g images in the latent space. The issue with some generative models and the FID score is that some generative models may memorize a specific pattern/sample from the correct answers and then apply it to all answers. To stop this, we will use MiFID that uses a memorization term, which is the minimum cosine distance between training data

$$d_{ij} = 1 - \cos\left(f_{gi}, f_{rj}\right) = 1 - \frac{f_{gi} \cdot f_{rj}}{|f_{gi}||f_{rj}|}$$

This term is bounded by a predefined constant (ϵ). If the term is less than epsilon, then the term has the same value. If it is greater than epsilon, however, then the term ($d_t hr$) is equal to one.

$$\text{MiFID} = \frac{\text{FID}}{d_{thr}}$$

The MiFID is calculated above and avoids the issue of generative models memorizing patterns. A lower MiFID means the quality of the generated images is similar to the quality of the real images while a higher score means there are major differences.

3.2. Experimental Hyperparameters

We ran multiple experiments with our DCGAN and CycleGAN to understand how GANs perform, what deep learning architectures are effective, and what hyperparameters produce better results.

3.2.1 DCGAN

We used the DCGAN to understand how GANs work and provide a baseline. One experiment we ran with DCGAN was understanding the importance of the learning rate on our problem. We first used a learning rate of 0.002 and got a high MiFID score of 74.3. We then used a learning rate of 0.004 and got a lower MiFID score of 69.3. Increasing the learning rate makes the model reach the closest local minimum of the loss function faster; however, increasing it too much will make the model miss the minimum. As a result, we did not increase the rate too much.

We were able to produce the below images using our DCGAN implementation. Monet images were randomly created by the GAN and compared to a ground truth input set. Qualitatively, we can see that these images could easily resemble those of Monet's, which indicates to us that our model is a good starting point. In terms of loss, it is fairly low, stabilizing around 2 in the later epochs.

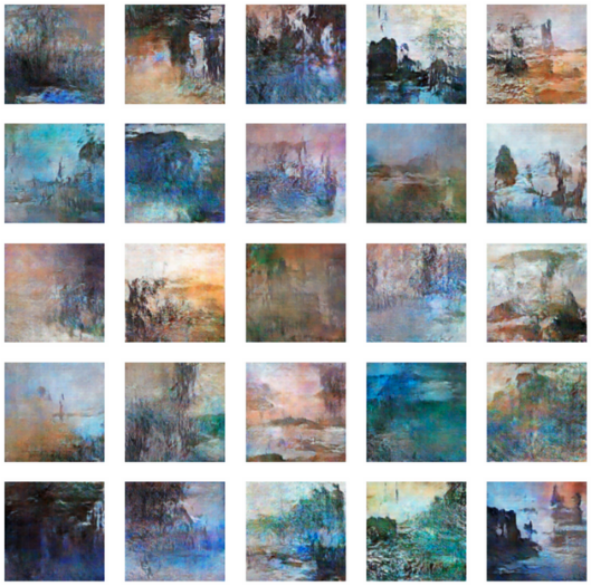

Figure 1. DCGAN Generated Monet Images

3.2.2 CycleGAN

In our CycleGAN experiments, we tested a variety of hyperparameters that could allow us to increase our model's accuracy. We experimented with learning rates, number of epochs, and image augmentation.

Model	Score
DC-GAN (lr: .002)	74.3
DC-GAN (lr: .004)	69.4
Cycle-GAN Baseline (20 Epochs)	59.64
Cycle-GAN (All Relu Activation)	100.34
Cycle-GAN (Augmented Images, 20 Epochs)	61.45
Cycle-GAN (80 Epochs)	48.35
Cycle-GAN (200 Epochs)	45.23
Cycle-GAN (200 Epochs + Aug. Images)	45.03

Table 1. MiFID Scores by Model

Similar to the DCGAN learning rate tuning, we wanted to see if we can find the most optimal learning rate for the CycleGAN. We found that once again, the best learning rate was 0.004. This rate will allow the model to take a large enough step that the learned parameters can be modified to find a local minimum of the loss function, and it ensures that the step is not so large that it would cause the model to miss the minimum. As for the number of epochs, we started at 20 epochs and ran up to 200 epochs without much overfitting. As the epoch count increased, the models improved. We started with a baseline score of 59.64, and as we added more epochs, the MiFID score decreased to 45.23. We decided to experiment with this hyperparameter because increasing the number of epochs gives the model more time to learn the most optimal set of weights for our problem. We had to remain careful not to increase it too much as that would lead to overfitting.

Image augmentation involves the process of changing images in the training dataset in order to increase the amount of data the model can learn from. By altering the images, we can provide the model with more data to improve its performance without causing it to overfit because the model is still trained on the same images. Image data was normally processed unless it was specified to use augmentation; in that case, our data pre-processing required rotating and transposing the images to augment them. The baseline CycleGAN model with 20 epochs combined with image augmentation resulted in a MiFID score of 61.45. This was a worse model than our baseline CycleGAN model, likely because this was not enough time for the model to properly fit, and adding image augmentation, which is used to help better generalize and avoid overfitting, made it a larger struggle to fit properly. We found that image augmentation was only significant after a high number of epochs closer to 200, reaching a MiFID score of 45.03. Our model showed improvement at this point and thus was able to generalize better.

In general, overfitting was not a huge issue because we were mainly bottlenecked by the amount of time that the model could run. 60 epochs or under didn't really suffer from overfitting, and we added Dropout layers to our model's architecture that also helped prevent overfitting. Overfitting can be judged through the MiFID score. If the score is 0, then we know our model has overfitted and created an image that is the exact same as the monet images. However, our models scored around 30-70, which shows that our model did not overfit.

Quantitative scores generally went along with the quality of the outputs of different models. Overall, our final model is successful, outputting great results. Images truly look like Monet-style paintings of different ground truth images. In the Kaggle competition we entered, we score quite well, placing within the top 30.

Figure 2. CycleGAN Monet-Style Transferred Images

4. Future Work

The next step for this project would be changing the hyperparameters more. For example, we can further increase the number of epochs to 300 and 400 and see how much lower the MiFID score can decrease. Training time for this many epochs is long, so we stopped at 200 epochs. Some external resources recommended adding a dropout layer hyperparameter to the discriminator and a batch normalization hyperparameter for both the discriminator and generator, which we did not do. Future work would incorporate testing these hyperparameters and seeing if it improves the performance.

Additionally, we would attempt text-to-image generation with Monet pictures. We would use a transformer to analyze the text and create token embeddings. After this, we can use a Image Creator/Stable Diffusion model like UNet to convert the token embeddings to the image. A main challenge we would have to overcome is if the user asks for a Monet picture with people. This is because Monet did not paint many paintings with people so the model may not be accurate if the user says for a painting of a person.

5. Work Division

See Table 2 for work division in this project.

6. Repository

Here is a link to our code. The url is:

`https://github.gatech.edu/rarunachalam8/DeepLearningMonetGeneration`

References

[1] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *OpenAI*, 2017.

[2] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *Berkeley AI Research (BAIR) Laboratory, UC Berkeley*, 2018.

[3] Amy Jang. Monet cyclegan tutorial. *"I'm Something of a Painter Myself" Kaggle Competition*, 2020.

[4] Kaggle. "i'm something of a painter myself" evaluation.

[5] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. 2017.

[6] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *Berkeley AI Research (BAIR) laboratory, UC Berkeley*, 2020.

Student Name	Contributed Aspects	Details
Rohit Arunachalam	Data Preprocessing and CycleGAN	Scraped the dataset for this project and set up data preprocessing (decoding images, reading tf record, loading data) to convert images to tensors. Created the CycleGAN model and set up training loop. Implemented image augmentation to improve results. Contributed to the Dataset, GANs in General, and Experimental Hyperparameters sections of the report.
Harsha Karanth	DCGAN and Loss Functions	Created the generator and discriminator elements for DCGAN. Implemented generator and discriminator loss functions. Experimented with more epochs. Contributed to the Introduction, MiFID, Experimental Hyperparameters, and Future Work sections of the report.
Eshani Chauk	DCGAN and CycleGAN	Created the downsample function for CycleGAN. Set up the DCGAN model and corresponding training loop. Experimented with learning rate. Contributed to the Abstract, Related Works, CycleGAN, and Experimental Hyperparameters sections of the report.
Sharan Sathish	CycleGAN and Loss Functions	Wrote the upsample function for CycleGAN. Created the generator and discriminator for CycleGAN and also implemented cycle consistency loss. Experimented with more epochs combined with image augmentation. Contributed to the Motivation, Implementation Issues, and Experimental Hyperparameters sections of the report.

Table 2. Contributions of team members.