GANs to create Monet-style Art

Daniel Swartz, Qi Song, Zhongyao Qian

March 25, 2024

Abstract

One of the popular applications of AI techniques is to map the input image from one style to another. For example, by uploading a selfie, people can obtain a cartoon version of themselves. Unlike image-to-image translation, for many tasks, paired training data will not be available. In this project, we are faced with the challenge of transforming a real-life photo into a Monet-style painting. We trained a CycleGan model based on 300 Monet paintings achieved photos similar to Monet's style after tuning the parameters.

1 Introduction

Great painters are often defined by their works of art or unique styles, such as van Gogh's post-impressionism, Picasso's cubism, or Monet's impressionism. Their unique styles allow viewers to identify who the artist is behind the work, while also leaving a lasting effect of inspiration and interest in their works. However, with advancements in artificial intelligence, particularly in generative adversarial networks (GANs), it has become possible for computers to mimic these art styles. We found this very interesting, and want to determine how well a GAN can replicate the style of an artist. Using a dataset of Monet's impressionism works and a wide variety of photos, we implement a cycle GAN that converts the photos to Monet style and Monet images to photo-realism. The challenge in this comes from attempting to create and train a GAN using unpaired data that can fool not only the discriminator built into the model but also our human eyes into believing that a generated image truly follows a certain art style.

2 Data

The dataset used is supplied by Kaggle [1] in the *I'm Something of a Painter Myself* competition. The overall data set is split into two subsets: Monet paintings and photos. Examples of the artworks can be seen in Figure 1, while examples of the types of photos present are displayed in Figure 2. What is depicted in the photos varies, with many photos of landscapes in nature, but also images of buildings, animals, and plants. The distribution of the data is shown in Figure 3, where there are 300 Monet paintings and 7038 photos.

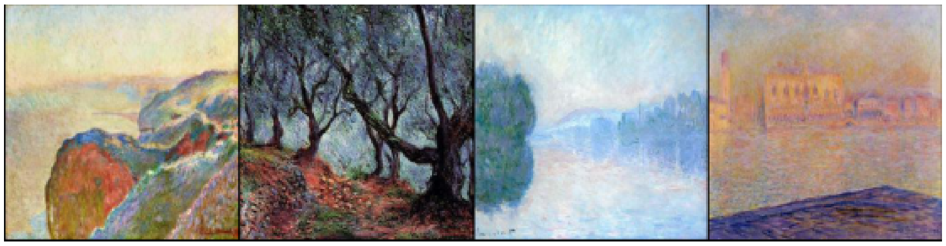

Figure 1: Validation Set of Monet paintings.

Figure 2: Validation set of photo data.

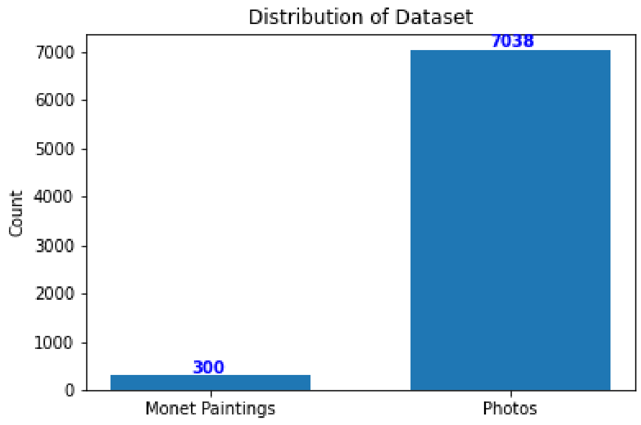

Figure 3: The distribution of the dataset, the number of Monet paintings and photos.

3 Model Selection

To address the challenge of creating a GAN that can effectively replicate Monet's styles onto photos, we chose to implement a CycleGAN [2] using PyTorch. The CycleGan aims to provide image-to-image translation abilities through learning the mapping between an input and output image. Typically this uses a training set with paired examples, but this procedure isn't feasible with our dataset, as there is no exact pair between the photos and Monet's art. However, even without the exact pairs, the CycleGan can still learn mappings through the use of cycle consistent adversarial networks [3]. This ability to conduct image translation on unpaired data made the CycleGan ideal for our purposes.

4 Model

Our model consists of two generators and two discriminators. The generators are more complex than the discriminators, as they feature an encoder-decoder architecture with added ResNet[4] blocks. The encoders use convolutions to downsample the data. The blocks each consist of two convolutional layers, each with reflection padding, and the first layer features a ReLU activation function. Nine of these blocks are then placed after the encoder layers, to apply weights to the extracted features. The decoder layers are then added to upsample to generate the output image. The discriminators are defined using the architecture for a PatchGAN[5] discriminator, consisting of layers of convolutions with a Leaky ReLU activation function. The discriminators are trained directly on both real data and the new images from the generator, while the generators are trained using feedback from the discriminator.

Training the model consists of two different parts, training the generators and training the discriminators. The generators are trained by calculating their total loss, which consists of a combination of GAN loss, cycle loss, and identity loss. GAN loss is measured by generating a fake image (either a fake Monet or fake photo depending on which generator is being trained) and comparing the result of the discriminator on that image to the true label. We take the mean square error (MSE) to determine this loss. To determine cycle loss, we first generate a fake image, then run generation on that image to convert it to its original style. We then measure the L1 loss between the original image and the final generated image to get this loss. Lastly, identity loss is determined by using the wrong generator on data to create a fake image in the same style as that data. For example, if the real image is a photo, we would run the generator to that converts a Monet to a photo on this photo. The difference between the original image and the generator's output is measured using L1 loss. Training the generator tries to minimize the total loss through each epoch, as this means the generator is producing more ideal fake images.

To train the discriminator, we determine two losses. The first loss measures if a discriminator correctly identifies that a real image is real. The second loss measures if the discriminator correctly identifies a fake image or not. These two losses are combined to determine the MSE, which serves as the total loss.

Both the generators and discriminators attempt to minimize their losses during training. Training of the generators attempts to trick the discriminator to falsely identify fake images as real while training the discriminators attempts to catch all false data. The total loss for the generators and discriminators can be seen in Figure 4 (top). Because of this interaction between the generators and discriminators, the cycle loss of the generators fluctuates between larger and smaller values between epochs, as the generators and discriminators improve. The other two losses for the generators consistently decrease, as they do not actively compete with the discriminator. The three different losses for generators can be seen in Figure 4 (bottom).

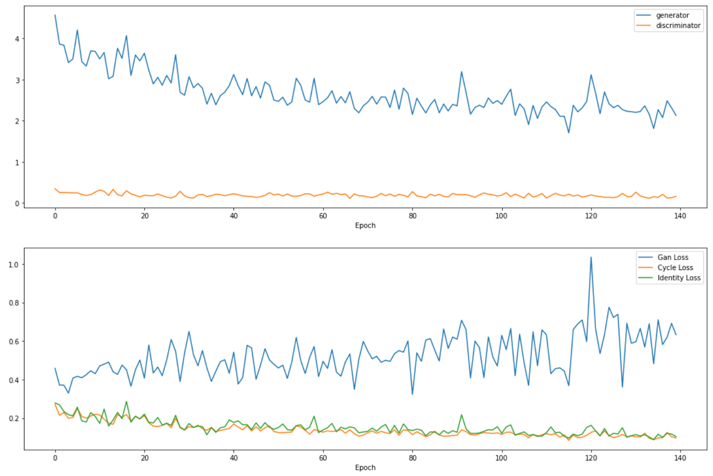

Figure 4: Three generator losses.

The validation process is unique compared to other machine learning models as it is not mathematically based. Instead, a batch of data is used as the validation set. After each epoch, the validation batch is printed, displaying the original images and then the generated images below them. As seen in Figure 5, both the photo-to-Monet and Monet-to-photo validation after an epoch are displayed.

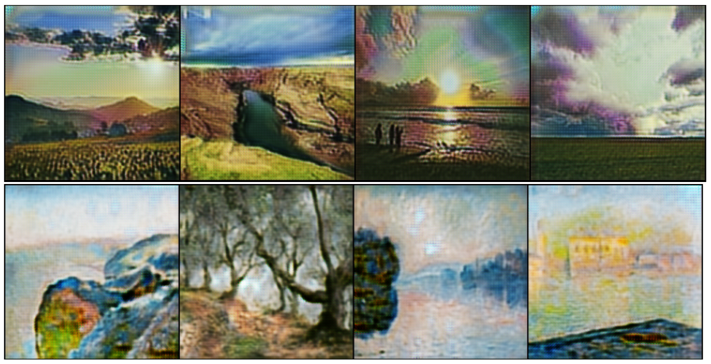

Figure 5: Validation batch after an epoch.

5 Result

We generated 7038 fake Monet paintings from the photos. From what we observed, most of the resulting images are very decent. There is no standard method to evaluate the model quantitatively. However, Kaggle provides a MiFID score for our reference, which evaluates the consistency across the output images. With this criterion, our best run performs at 54.64974. However, a lower MiFID score does not necessarily mean a better transformation of style. We sample a couple of the generated Monet images and take a close look.

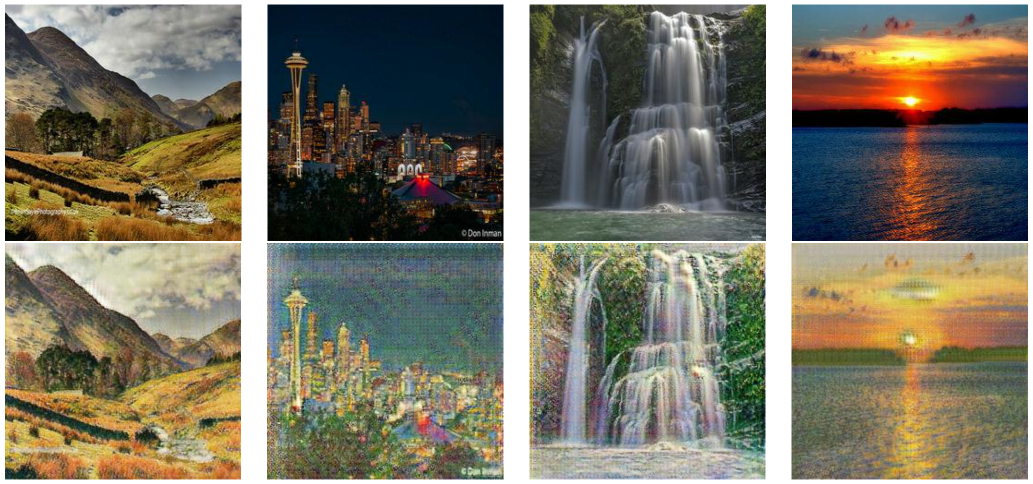

Figure 6: Original photos (top) compared with the generated results.

As shown in Figure 4, the generated images do show a trend towards a more Monet-like style. There is a stark difference between the original image and the generated version, such as less sharp edges, less contrast between colors, and an almost pigmentation-like filter over the entire image. Based upon these results, we can conclude that our GAN model is able to generate images that replicate the style of Monet to a certain extent. The images follow Monet's art style, however there are some differences that allow one to identify if it is a real Monet or a generated image.

There are a few hyper-parameters we tried to improve the performance of the model. In the first place, we tried to use batch normalization with no bias and the result does not end up well, the result pictures were surprisingly consistent with the input pictures. The style doesn't change from the input pictures. We believe that was caused by a lack of variance in our model and this causes the model to be trapped in a local minimum. Thus, we tried Instance normalization and enabled the bias, and we could see a significant improvement in the result.

6 Contributions

Team members:

Daniel Swartz - Researched what model to use. Visualizing data. Collaborated on the report.

Qi Song - Researched what model to use. Built up the baseline model, collaborated in the report.

Zhongyao Qian - Researched what model to use. Tuning the model, collaborated in the report.
5mm

References

[1] Kaggle.com. *I'm Something of a Painter Myself Dataset*. URL: https://www.kaggle.com/c/gan-getting-started/data. (accessed: 2/25/22).

[2] Jun-Yan Zhu et al. “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”. In: *CoRR* (2017). URL: http://arxiv.org/abs/1703.10593.

[3] Aamir Jarda. *A Gentle Introduction to Cycle Consistent Adversarial Networks*. URL: https://towardsdatascience.com/a-gentle-introduction-to-cycle-consistent-adversarial-networks-6731c8424a87#:~:text=A%5C%20cycle%5C%20consistency%5C%20loss%5C%20function,is%5C%20the%5C%20Generative%5C%20adversarial%5C%20network. (accessed: 2/28/22).

[4] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* (2015). URL: https://arxiv.org/abs/1512.03385.

[5] Phillip Isola et al. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *CoRR* (2016). URL: http://arxiv.org/abs/1611.07004.