CycleGan: Transfer of style from image to Monet

Meiyi Luan Xavier Morin

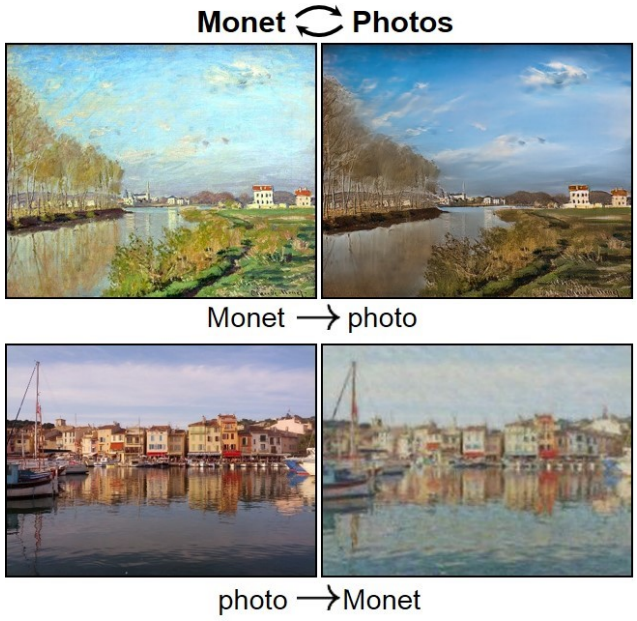

Figure 1: This figure represents our goal [3] .

Abstract

We chose our project from Kaggle: "I'm Something of a Painter Myself"[2]. Our goal was to generate paintings in the style of Monet. We had in mind to use a GAN, by training a Generator to map a latent space to an image. This goal has been simplified to transferring the style of Monet to existing images. We used Image-to-Image translation with Cycle-Consistent Generative Adversarial Network[3]. We did not achieve the resolution or quality that we aspired to. However, we learned a lot and are satisfied with the results.

1. Introduction

We were amazed by the various accomplishments of GANs before taking this course, for its applications in Science to improve astronomical images or to generate new molecules, or its use in Art and Fashion. The whole idea behind GANs is also thrilling, having two networks competing one against each other. This said we thought that we could train a GAN to generate paintings of Monet from ground up using Progressive Growing GAN (See our tentative to train a Progressive Growing GAN at the end of this document). It was rather naïve, and we rapidly changed our direction to use a Cycle GAN in order to do image translation like in Figure 1. We give all the credits to **Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks**[3].

2. Theory

Cycle GAN

A GAN is composed of two Neural Networks: a Discriminator and a Generator. Which are trained simultaneously to compete against each other. A Cycle GAN is composed of two GANs : two Discriminators and two Generators. One Gan is learning the mapping $G : X- > Y$ and the other the mapping $F : Y- > X$. The idea behind using two GANs is to be able to introduce a second parameter to the loss function. Here are the default loss functions for a GAN:

$$L(D) = \frac{1}{2m}\sum_{i=1}^{m}[(D(x_i)-1)^2] + \frac{1}{2m}\sum_{i=1}^{m}[(D(G(z_i))^2]$$

$$L(G) = \frac{1}{m}\sum_{i=1}^{m}[(D(G(z_i)-1)^2]$$

Figure 2: Translating Monet to Picture without Cycle Loss

Those loss functions basically cover the characteristics of the style, the Discriminator recognizes Monet's style or not and the Generator does well if it can convey Monet's style. However, with only those loss functions, the Genera-

tor generally will not preserve the scenery of the input image. Since we are only asking it to convey the style not to preserve the original image structure, we need to introduce the cycle loss.

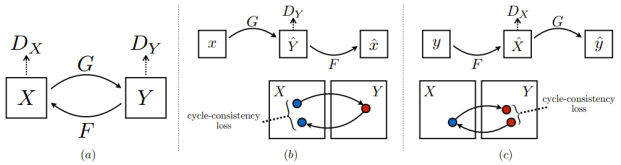

Figure 3: The Cycle Loss [3]

The cycle loss will assure that the Generator changes the style of the image and also preserves the original image scenery. Finally, we also applied an Identity loss, whose function will tell the generators to let an image unchanged if it is already in the desired domain. It also helps to preserve the colors of the input when applying the full cycle. Here are those two extra losses:

$$Cycle_L(G) = \frac{1}{m}\sum_{i=1}^{m}(y_i - G_{x->y}(G_{y->x}(y_i)))^2$$

$$Identity_L(G) = \frac{1}{m}\sum_{i=1}^{m}(y_i - G_{x->y}(y_i))^2$$

Residual Block

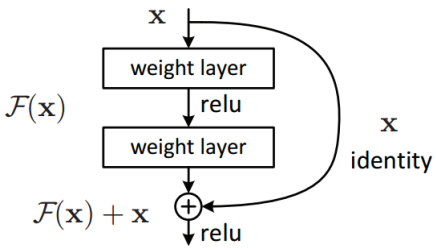

Figure 4: A Residual Block

There is a limit to how many layers can be stacked in a neural network. When the error is propagated from the last to the first layer, it slowly vanishes. If the neural network is too big, the first layer would never get updated and would never learn anything. This is known as the vanishing gradient problem. The Residual Block proposes a solution to this problem, by letting the input flow through the network unchanged. This elegant solution, allows to train bigger networks and also lets the network directly learn the Identity function.

ResNet

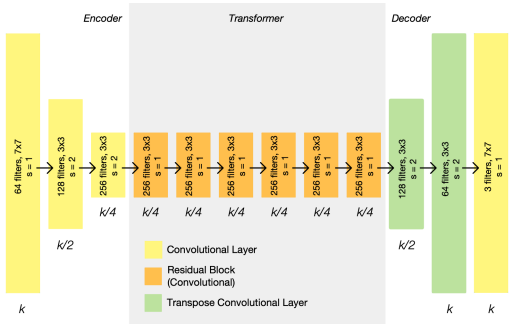

Figure 5: ResNet Generator Architecture .

The ResNet Generator applied in [3] makes use of the Residual Block. It is composed of an Encoder, a Transformer and a Decoder. The Encoder is composed of three Convolutional layers that each reduces the size of the Image by a factor of 4. The transformer of the image is composed of a series of Residual Blocks, 6 for 128x128 and 9 for 256x256 images. The Decoder brings back the image to its original form performing deconvolution three times.

Patch GAN

A Patch GAN is a type of Discriminator that only penalizes structures at the patch level. Each patch is a NxN subImage taken from the Image. The Discriminator evaluates if each patch is part of the desired Domain. To calculate the loss, we simply take the mean of all those patches. This type of Discriminator has been proven to be highly effective to discriminate different types of texture.

U-Net

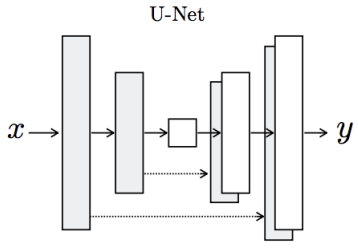

Figure 6: Architecture of the U-Net Generator Model .

The U-Net Generator is composed of Encoders and Decoders that meet at a bottleneck. The Strength of the U-Net

is that each Encoder is directly connected to its correspondent Decoder, the first Encoder with the Last Decoder and so on. These Generators rely on Convolution, Deconvolution, LeakyReLU and ReLU activation layers and BatchNormalization.

3. Methodology & Experimental Results

We wanted to start simple with Cycle GAN and found an assignment from the University of Toronto on the subject, and did it. We learned the basics with this remarkably simple architecture.

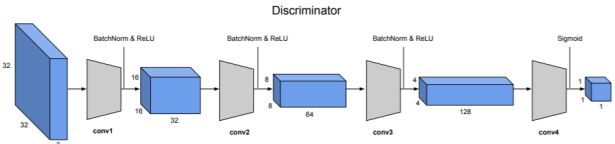

Figure 7: The Discriminator [1] .

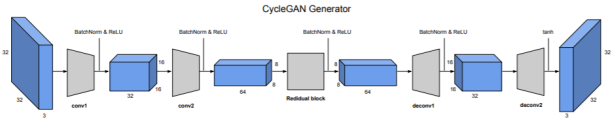

Figure 8: The Generator [1] .

Note that this implementation takes input images of size 32x32 and that the discriminator is not a patch GAN, it simply returns 0 or 1. Our first trial was with Emojis (Figure 9). We tried to convert Apple's Emojis to Windows' Emojis.

Figure 9: Translating emojis from Apple to Window's style

After learning the basics, we started working with the dataset provided by Kaggle. During all of our experimentations we kept some factors constant, following the recommendations found in [3] . We used the Adam solver, we set lambda=10 and all networks have been trained from scratch with a learning rate of 0.0002. For most of the experimentations we used Batch Normalization with Batch size 16. Have a look at the timeline.

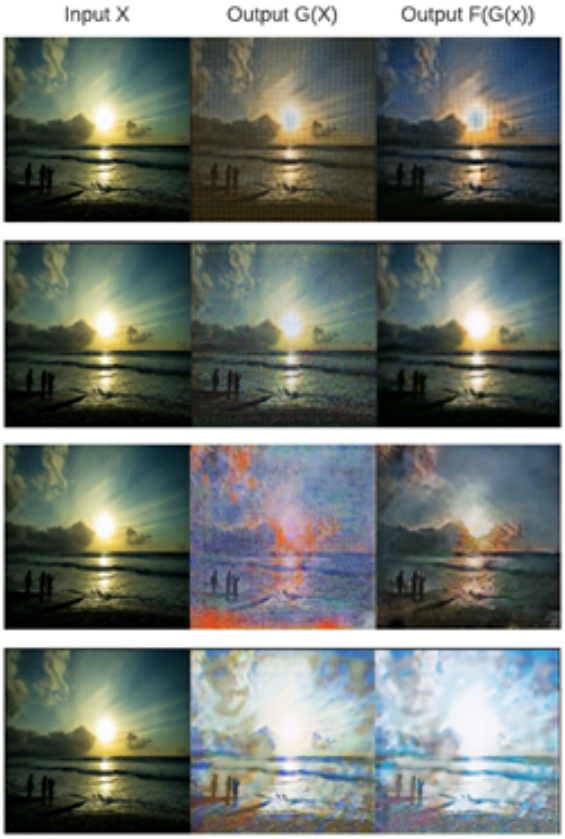

Figure 10: Left: Input X, Center: Ouput G(X), Right: Output F(G(X)). From top to bottom: Scaled Archictecture, Scaled Architecture and Patch GAN, U-Net, ResNet

In our first attempt, we adjusted the architecture used with Emojis and used 256x256 images as inputs. We kept exactly the same logic but with seven convolutional layers in the Discriminator to reduce the input image to an 1x1 output. We also added two extra layers to the Generator, one convolutional and one deconvolutional. The architecture of the Discriminator made it almost impossible for it to differentiate real images from the fake. The translations of the Generator were not satisfying, mainly because the Discriminator was not giving correct directions. However the Generator was able to perform the cycle and the identity transformations.

That is when we discovered the Patch GAN and decided to introduce it. We implemented a 70x70 Patch GAN with a total of five layers of convolutions, resulting in a 30x30 out-

put. But introducing the new Discriminator wasn't enough and it even seemed to decrease the quality of the result. We concluded that the Discriminator was too powerful compared to the Generator.

When searching for a new architecture we encountered U-Net Generator model, which finally gave us our first convincing translation. We did a lot of experiments with the U-net Generator. We met lots of funny results along the way. I invite you to look at the end of the document for our translation Monet -¿ Monet on Fire. Wait, wait, finish to read our report before going at the end . . .

Finally, we wanted to try the architecture of the Generator used in the original paper. Our best results came from the ResNet. We changed to use Instance Normalization with Batch size 1. With this architecture we also introduced the concept of a Pool for fake Images. This way the Discriminator was trained on both past and new Generated images, making the learning more stable. Note that the last image is not good, because the Generator from Monet to picture broke when we saved it.

We kept this architecture until the end. To improve our result we did some data augmentation. We wanted to threefold the number of Paintings of Monet, so we flipped the Paintings horizontally and vertically.

Figure 11: Data Augmentation

Our last endeavour was with scheduling. We trained the Models over 70 000 images with a constant learning rate and again over 70 000 images with a linearly vanishing learning rate to zero.

4. Conclusion

We are very glad to have chosen this project, we didn't really think that it was possible. We learned enormously about PyTorch and how to conduct a Neural Network project. We are overall satisfied with our submission, but we plan on training the network again in the Winter vacation and submit our project to Kaggle.

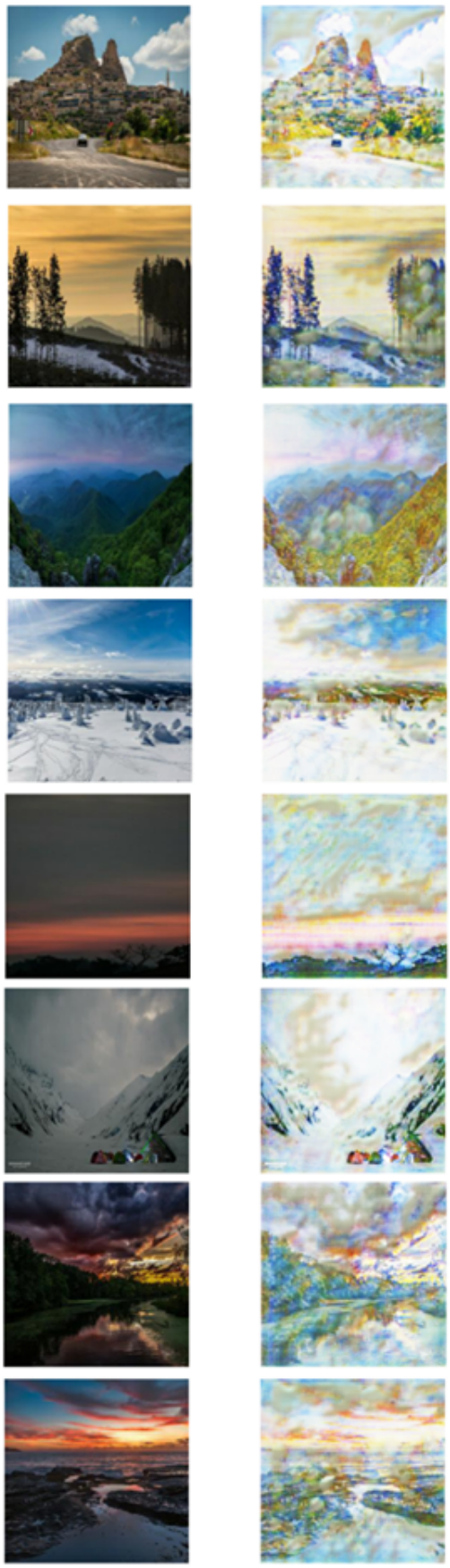

Figure 12: Final Results

References

[1] Cyclegan a4 from university of toronto. 3

[2] I'm something of a painter myself. 1

[3] Jun-Yan Zhu Taesung Park Phillip Isola Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. 2017. 1, 2, 3

Appendix: Extra Results (Optional)

Figure 13: Monet to Monet on Fire

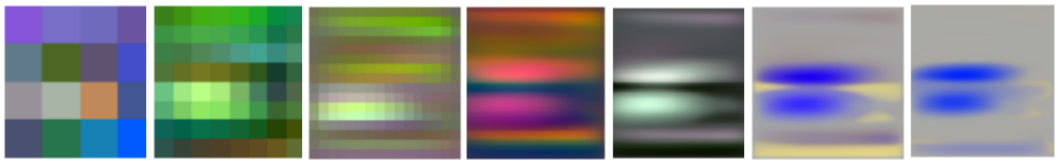

Figure 14: Progessive Growing GAN, from 4x4 to 256x256