Machine Learning II Individual Final Report

Zichu Chen

1. Introduction.

I'm something of a painter of myself is a competition on Kaggle which requires to generates 7,000 to 10,000 Monet-style images from real photos using GAN. In this project, we trained our CycleGAN model with 300 Monet-style paintings and generated 7038 Monet-style images using given photos.

All the group members gave opinions on which topic to choose and decided at last to choose the GAN challenge on Kaggle. We all learned about the architecture of GAN and CycleGAN and implemented some notebook on Kaggle.

Even though we used different framework, we shared our understanding of data augmentation and the network architecture in our model. We also discussed about the influence of changing the learning rate and data size.

For the presentation part, we finished the slides and final report together on Google Sheet.

2. Individual work

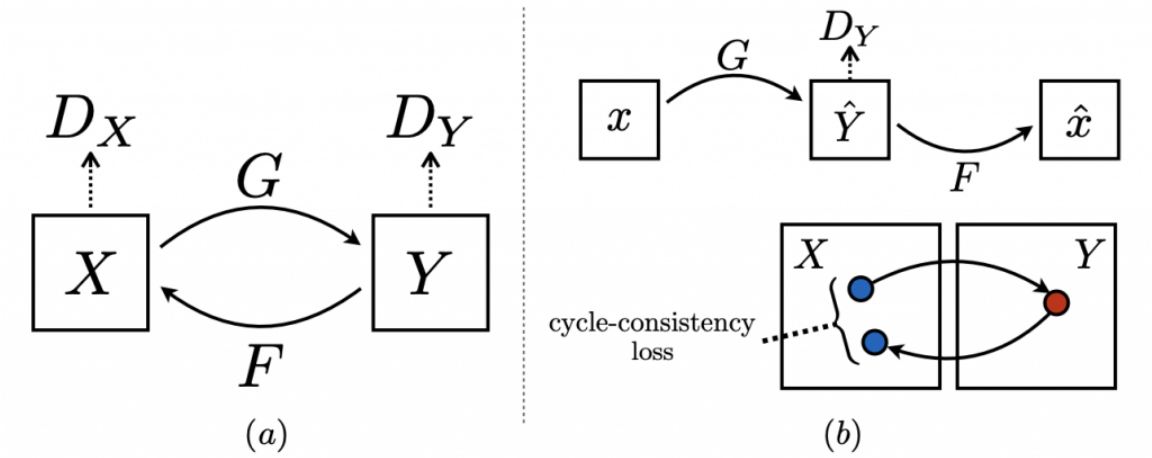
My work in the group includes part of data preprocessing, building the generator and discriminator of CycleGAN, part of hyper parameters tuning and training the model. For the report, I took charge of the network description and experimental setup of model on TensorFlow. Below is the Cycle GAN description, including its composition and how the loss works:

Generative adversarial networks (GANs) are algorithmic architectures that use two neural networks, pitting one against the other (thus the "adversarial") in order to generate new, synthetic instances of data that can pass for real data. They are used widely in image generation, video generation and voice generation.

A GAN is comprised of two models: a generator model and a discriminator model. The generator takes a point from a latent space as input and generates new plausible images from the domain and the discriminator takes an image as input and predicts whether it is real or fake. Both models are trained and expected to generate more realistic images.

The CycleGAN is an extension of the GAN architecture that contains two generators and two discriminators. In this case, we can summarize the generator and discriminator models as follows:

Generator G: Takes real photos as input and generate fake Monet paintings as output.
Discriminator D_X: Takes real Monet paints and fake Monet paintings generated by G as input and output the judgment.
Generator F: Takes real Monet photos as input and generate fake photos as output.
Discriminator D_Y: Takes real photos and fake photos generated by F as input and output the judgement



The loss of CycleGAN contains adversarial loss and cycle consistency loss. Adversarial loss is a standard GAN loss. In this case we have 2 adversarial loss:

$$L(G_X, D_Y, X, Y) = E_{y\sim p(y)}[logD_Y(y)] + E_{x\sim p(x)}[1 - logD_Y(G_X(x))]$$

$$L(G_Y, D_X, Y, X) = E_{x\sim p(x)}[logD_X(x)] + E_{y\sim p(y)}[1 - logD_X(G_Y(y))]$$

Cycle consistency loss compares an input photo to the Cycle GAN to the generated photo and calculates the difference between the two:

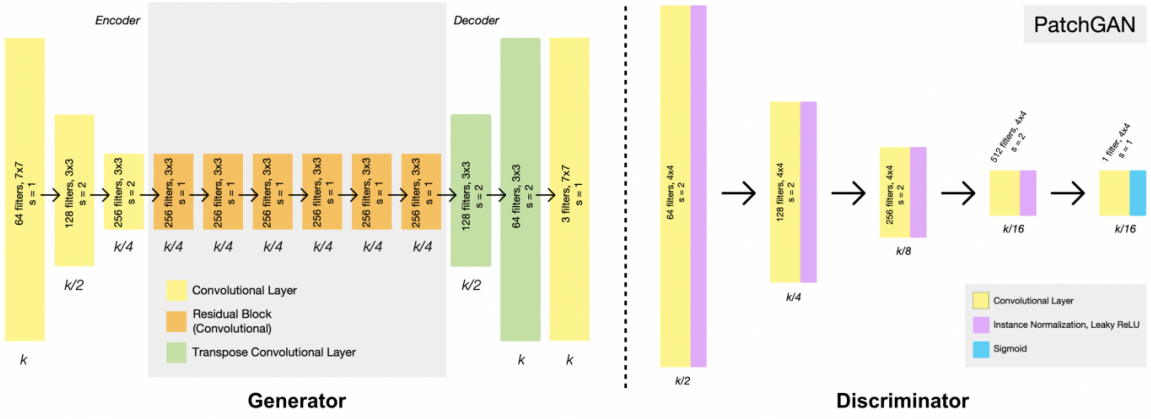
$$L(G_X, G_Y) = E_{x\sim p(x)}[||G_Y(G_X) - x||_1] + E_{y\sim p(y)}[||G_X(G_Y) - y||_1]$$

The whole loss is:

$$L(G_X, G_Y, D_X, D_Y) = L(G_X, D_Y, X, Y) + L(G_Y, D_X, Y, X) + \lambda L(G_X, G_Y)$$

3. Portion of work

In the whole project, Lau and Vishal took the Pytorch implement and basic model while Kelvin and I took the TensorFlow model and the improved model. For all the code, I took the charge of building and debugging the model, including the function of encoder, Resnet block, decoder and using them to build the generator and decoder. Here is the architecture of the whole system:

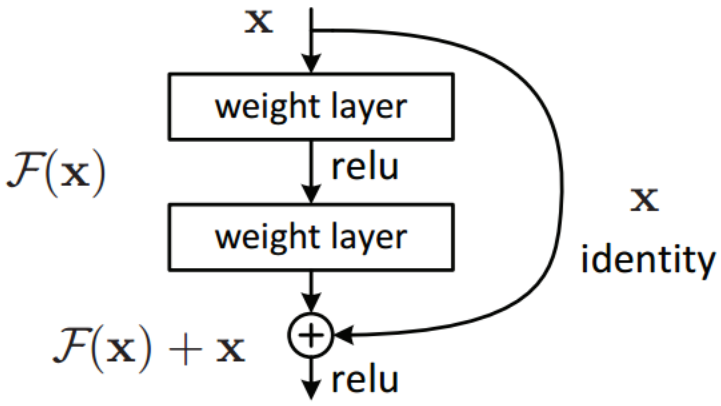


A generator in CycleGAN normally consists of Encoder and Decoder. In this improved model, a transformer with 6 Resnet blocks were added to avoid the possible gradient vanishing resulted from adding the depth of the network.

Having batch size as 1, GAN uses instance normalization rather than batch normalization like other models. Leaky ReLU is normally used in GAN architecture, while in this case, it is only used in decoder block in generator. The encoder of GAN down sample input images and extract features from all the images. In our model, it has architecture below:

Layer	Filters	Kernel Size	Stride	Normalization	Activation
Conv2D	64	7*7	1	Instance Normalization	ReLU
Conv2D	128	3*3	2		
Conv2D	256	3*3	2		

Complex model sometimes comes with degradation problem, which is, it fails to learn simple functions. Resnet is used to solve the problem. In this case, six same Resnet block is connected to Encoder and each Resnet Block has architecture and parameters below:



The decoder in GAN uses several transposed convolutional layers to up sample the feature matrix. Leaky ReLU is also used in this block. To solve gradient vanishing problem, skip connection is also set between encoder and decoder layers with same size. The detailed parameters are shown below:

Layer	Filters	Kernel Size	Stride	Normalization	Activation
Conv2D	64	7*7	1	Instance Normalization	ReLU
Conv2D	128	3*3	2		
Conv2D	256	3*3	2		

Discriminator in GAN works as a classifier and identify the whether the input is real images or images generated by generator. Convolutional layers with stride are used instead of convolutional layers with pooling layers. The architecture is shown below:

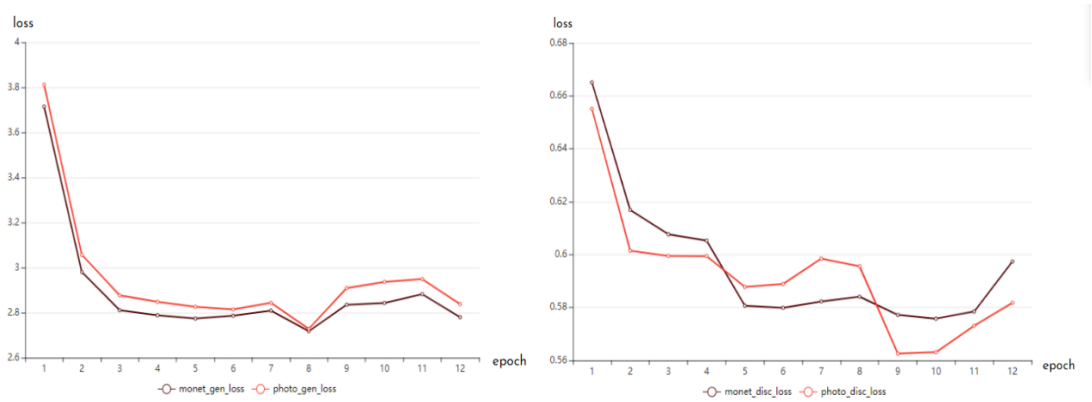
Layer	Filters	Kernel Size	Stride	Normalization	Activation
Conv2D	64	4*4	2	/	ReLU
Conv2D	128	4*4	2	Instance Normalization	
Conv2D	256	4*4	2		
Conv2D	512	4*4	1		

4. Results

The loss of 4 components of CycleGAN is shown as follow:

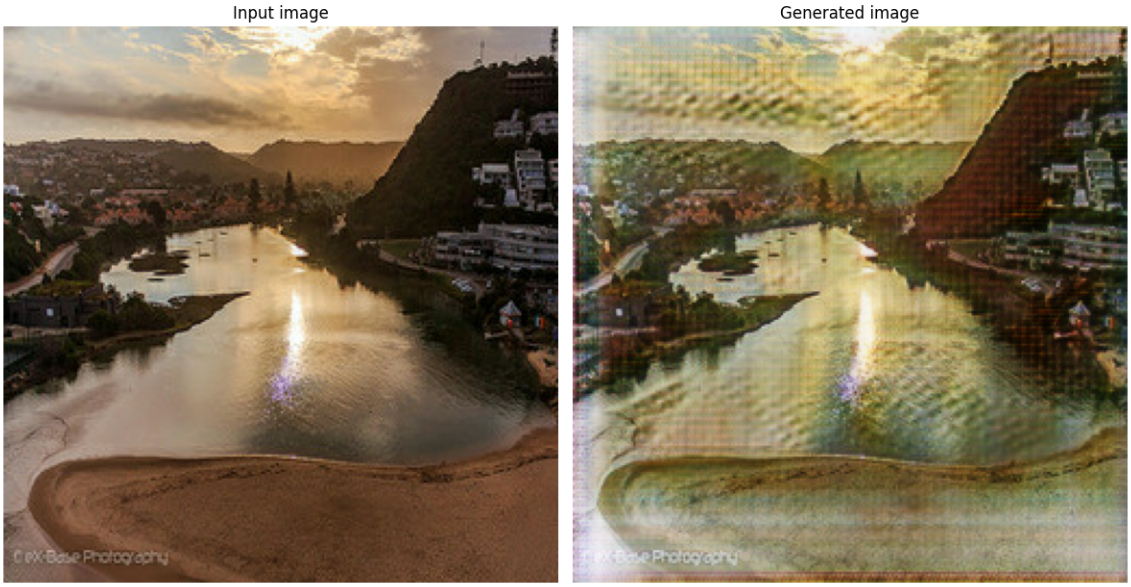
Loss	**moent_gen_loss**	**photo_gen_loss**	**monet_disc_loss**	**photo_disc_loss**
TensorFlow	3.9370	4.0263	0.4280	0.4367

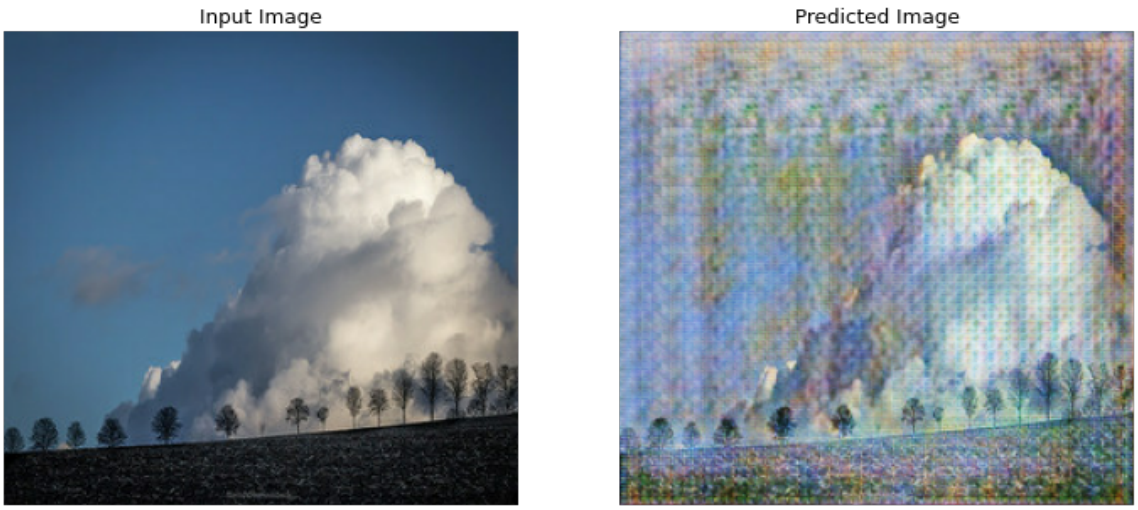
The loss curve of the model from 1 to 12 epoch is shown below:



The discriminator has a fluctuation at the beginning and reaches the lowest in the 9th-10th epoch. When the generator learned how to generate more realistic models, the discriminator loss began to increase at 11th epoch. Because of the time limit, we chose 10 epochs model as our final output summary and conclusions.

To compare the real photos and generated paintings, we noticed that monet generator has a better performance when dealing with photos with more colors. However, when dealing with those larger blocks of color, the generator usually has a bad performance, and the generated images show more noise.





In this project, I better understood the model of GAN and learned about how CycleGAN was improved from GAN:

- Convolutional layer with strides in Discriminator and Transposed convolutional layer instead of pooling layer
- Use instance normalization when batch size is 1
- Use LeakyReLU instead of ReLU in decoder
- Use Resnet block and skip connection to avoid gradient vanishing

Also, TPU could be used to better accelerate training process. Here is the performance with different accelerators:

Accelerator	GPU on AWS	GPU on Kaggle	TPU on Kaggle
Time	600s/epoch	233s/epoch	113s/epoch

Also, using the learning rate scheduler could help improve the model.

Our submission on Kaggle got a score of 64.0 and ranked at 134th.

5. Copied code percentage

The percentage of copied code from Internet is 156-32/362-32*100%=37.5%

References.

[1] https://machinelearningmastery.com/what-is-cyclegan/

[2] https://www.lyrn.ai/2019/01/07/instagan-instance-aware-image-to-image-translation/

[3] https://www.kaggle.com/amyjang/monet-cyclegan-tutorial

[4] https://www.kaggle.com/dimitreoliveira/introduction-to-cyclegan-monet-paintings

[5] https://www.kaggle.com/dimitreoliveira/improving-cyclegan-monet-paintings

[6] https://www.kaggle.com/doanquanvietnamca/the-beauty-of-cyclegan

[7] https://www.tensorflow.org/tutorials/load_data/tfrecord