I AM SOMETHING OF A PAINTER MYSELF

--Individual Report

Group 3

Reported by

Kaiqi Yu

December 7, 2020

Contents

1. Introduction

1.1 Overview of project

The goal of this project is to use different frameworks (Keras and PyTorch) to build a GAN that generates 7,000 to 10,000 Monet-style images. We used Keras and PyTorch separately to create models. To improve the performance of our model, we tried different parameters and data augment methods.

The dataset was provided by Kaggle and it contained four directories: “monet_tfrec”, “photo_tfrec”, “monet_jpg”, and “photo_jpg”. The “monet_tfrec” and “monet_jpg” directories contain the same painting images, and the “photo_tfrec” and “photo_jpg” directories contain the same photos.

1.2 Shared work

We discussed the choice of the topic together. Then we divided the four people into two groups to use different frameworks (Keras / PyTorch) to implement the cycleGAN model.

In Keras group, we shared the model. Besides, we will summarize and communicate about our work every night to discuss what kind of improvement is feasible.

We also met the PyTorch group once or twice a week to learn about the latest development and performance of their model.

For the final report and slides, we finished it together by sharing Google Docs and Google Slides.

2. Individual work

2.1 Overview

My job was data preprocessing, data augment, parameter adjustment and visualization of results in Keras group. As for documentation work, I wrote the proposal and several parts in final report (data preprocessing, data augment, results and parts of summary).

2.2 Data Preprocessing

2.2.1 read

Because the data format is tfrec files, I use TFRecord to read the data, and use parse_single_example() to parse the input tf.train.Example proto.

2.2.2 decode

Next, I used the method of parsing jpeg files that comes with tf.image to read the image data and convert it to the size of [256,256,3].

2.2.3 data augment

Data augmentation for GANs should be done very carefully, especially for tasks like style transfer. If our conversion causes the brightness, contrast, and other patterns of the data to change, it may cause the generator to fail to learn the basic styles effectively.

Therefore, I only used flip and rotation to generate new pictures. The specific situation will be introduced below in section 2.3.

2.2.4 normalize

In order to make the pixel value of the picture between [-1, 1], I used the following formula to normalize it.

$$img = (img \div 127.5) - 1$$

This normalization method can get better results than the usual method of changing the pixel value to [0,1].

2.3 Data Augment

2.3.1 Repeat

When adding data, I used the dataset.repeat() method to generate data repeatedly. In the beginning, I repeated the data of Monet paintings three times, but due to the long time required to train the data, I changed it to repeat twice.

Later I tried to set the step_per_epoch parameter. And if there is this parameter when training the model, the data.repeat() method does not need parameters. It will automatically copy until it reaches the set amount.

2.3.2 Rotation

When designing the image rotation, I first initialized a random number, and then decided to rotate the image by 90, 80 or 270 degrees by judging the range of the random number.

2.3.3 Flip

In the same way, by judging the range of the generated random number to perform random left-right flipping and up-down flipping, and again judging the random number range to decide whether to transpose the data matrix.

2.3.4 Crop

At the same time, I also tried to resize the image to a larger size and then crop it. But I found that the result was not ideal because the cropping may have caused the lack of border information.

2.4 Parameter adjustment

2.4.1 Num of epochs

In the beginning, I set the number of epochs to 10, and the result showed that the loss gradually decreased. So, I tried to set more epochs. First, I set it to 25 epochs, and the results showed that the loss value began to fluctuate after the 10th epoch. Finally, I set the epoch number to 12, so that the change in loss value can be seen intuitively, and the specific analysis is in section 2.5.

2.4.2 Steps per epoch

In the beginning I did not set this parameter but let the model train all the Monet image data in each epoch. Later I set it to the integer part of the quotient of the maximum number of Monet paintings and the number of photos divided by 16. But there is not much difference between the two results.

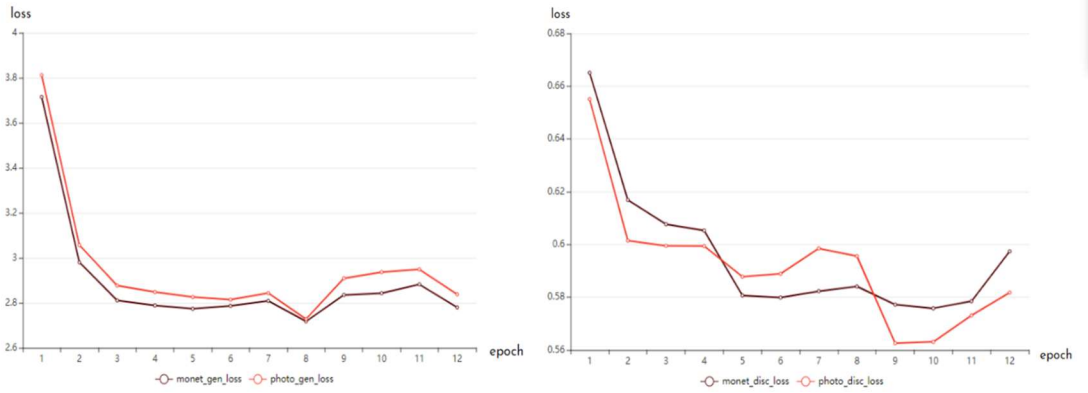
$$step\ per\ epoch = \max(number\ of\ monet\ paintings, number\ of\ photos) \div 16$$

Use only the integer part of the above formula.

2.5 Visualization of results

2.5.1 Loss

The loss curve of the Keras model over time is shown below:

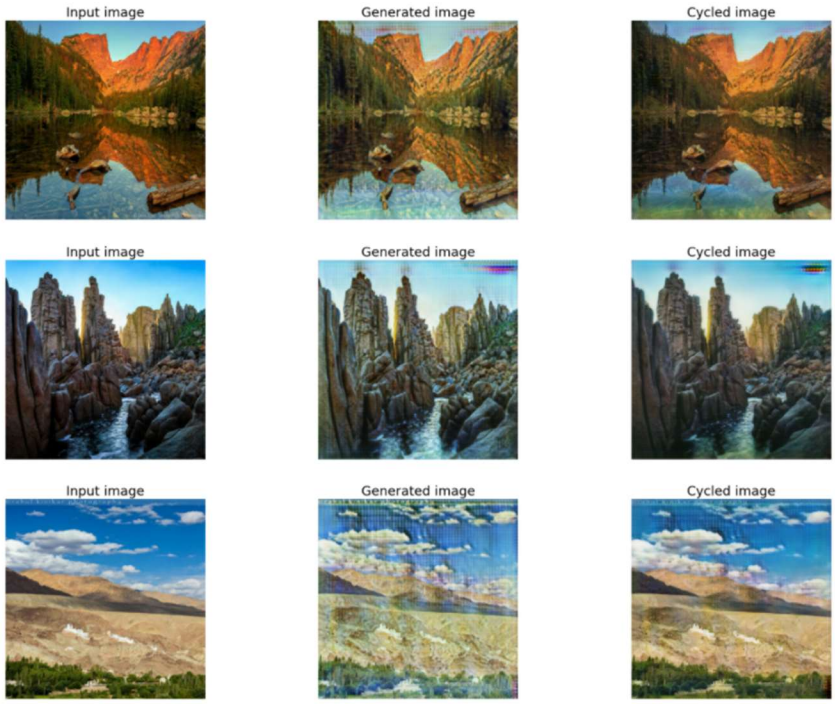


As shown in the figure above, the generator's loss value reaches the lowest at the 8th epoch, and then it starts to fluctuate.

The discriminator has fluctuations 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.

2.5.2 Cycle Evaluation

The picture generated by the generator cycle is shown below:



The figure above shows Monet's paintings generated by monet generator with input photo. The image on the right are photos generated by photo generator taking fake Monet's paintings as input

We can see that generators have better performance on some pictures like that in the first and second set, and the last set of pictures show much noise because of large color blocks.

The figure above shows photos generated by photo generator with Monet paintings data as input. Then take photos as input and generate Monet paintings through a monet generator.

It can be seen from the figure that the photo -> Monet's paintings -> photo cycle works well.

2.5.3 Monet Generation

The main purpose of this project is to generate Monet paintings from photos. Some manually picked images generated by monet generator are shown as follows:

As shown in the figure, some output images of the network are not very realistic. We can see some pictures with Monet's style, but others having noise. There is still a long way to go to.

2.5.4 Running time comparison

We have tried to use several platforms and different hardware to train models. The time comparison results are as follows:

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

As shown in the table, using the TPU on Kaggle to train the model has a large advantage than that on GPU. However, there is a time limit for using GPU or TPU on Kaggle, so if you need long time and multiple training, GPU instance on AWS should be chosen.

2.5.5 Gif

In the process of model training, I used a custom GANMonitor class inherited from Callback to save the pictures after each round of epoch. Then I made all the saved pictures into a gif picture. The gif picture can visually show the changes of the picture during the training process.

3. Summary

3.1 Conclusion

In the comparison of the two models, our improved TensorFlow & Keras model with more preprocessing has a better performance.

Besides, Under the trial of different platforms and hardware, the TPU on Kaggle got the best results and the shortest time.

When training the TensorFlow & Keras model, it takes long to train on more data and more epochs, which means we could not get satisfactory results with more suitable epoch or learning rate. But the method is feasible, and more adjustment is available.

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

3.2 Future work

In future work, I will improve the Keras models by using learning rate scheduler to control the model and choose the best learning rates. And try to use many kinds of data augmentation methods and train more epochs to get a best model.

In addition, I will continue to upload the code to Kaggle and get better scores by constantly improving the model.

4. Calculation

Calculate the percentage of the code that found or copied from the internet: 30%.

5. References

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

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

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

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

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