Style Transfer: An odyssey to anime world

Xinrui Jiang Harvard Summer School & Fudan University

Abstract

In this project, I try to solve such a kind of problem: given a set of real-world images, together with another set of anime images, I want to make photos containing real-world content have the animation style of the animation pictures, meanwhile keeping the original image content. This problem is also referred to as Style Transfer. To solve this problem, I constructed my own dataset, containing real-world photos and cartoon images that have never been used before. After that, I referred to former achievements such as CartoonGAN and built a generative adversarial model(GAN) structure, which improved on the basis of traditional GAN structure. By applying techniques such as GAN and Transfer Learning, together with supplementing content loss and style loss as loss function, this deep learning model presented strong style transfer ability. Even having limited training epochs due to expensive computation and coding difficulty, our model produced reasonable output on given test images.

Motivation

At present, there are more and more high-quality animation creations, and the popularity and attention of animation are increasing day by day. At the same time, People's demand for high-quality animation creation increased so much that some people even hope to see what the environment around them looks like when applying anime style. However, hand-painted works are time-consuming and difficult to get started with, making it hard to realize such an idea.

Deep learning techniques offer a possible way out. Generative Adversarial Networks (GAN) empowers us to generate specific images on a large scale. In the field of style transfer, VGG model based on Convolutional Neural Network(CNN) has proven its strong ability in extracting key features of an image, which can be used to judge style and content of an image. Combining these two techniques makes it possible to quickly animate real-life pictures, without requirements for painting skills. Previous related research has made some progress, but it still needs to be optimized. Therefore, building style transfer model by hand to learn about the latest developments and train on datasets with more aesthetic value becomes meaningful.

Task definition

We are basically dealing with a “prediction” problem. When the model is finally implemented, we want to input a picture in real life, and generate the result of animation of this picture as the output.

After firstly understanding the current achievements in this field, we will try to apply CNN, GAN, and relevant possible techniques to build a model. In order to achieve better results, I apply transfer learning and call pre-training layers from previous VGG19 model. After defining our model architecture and selecting suitable loss functions, I train the model with datasets and try to find suitable hyperparameters. At the same time, I also try to optimize the architecture of the original model.

Dataset

Our dataset will be made up of two kinds of picture, one is the scene pictures in real life as input, and the other is the ones in the anime as output.

Training dataset, which is made up of real-scene images, are extracted from youtube videos which are about tourist areas around the world. Besides, some images previously used for training CycleGAN are also added to my dataset. In total, we have 5229 images as training set, 1252 images as validation set, and 501 images as prediction set.

Anime images are important to this task. In order to make the generated images have more aesthetic value, I chose the Japanese animation movie "violet eternal garden" as data source. Through the methods of key frame extraction and image cutting, I obtained 2823 pictures as the training set, 706 images as validation set.

Due to the particularity of style transfer, I also smoothed the animation pictures to obtain better effects. Therefore, I additionally have 2823 smoothed anime images as training set, 706 images as validation set.

Each image mentioned above is RGB image, having the size of 256×256, excepting prediction images. Each prediction RGB image has the size of 1920×1080.

Hardware environment

·Dell Inspiron 16 7610, 16GB RAM, 512GB storage

·64-bit operating system, x64-based processor

·CPU 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz

·GPU NVIDIA GeForce RTX 3060 Mobile 6GB GDDR6

Software environment

·Win 11 operating system

·V11.2.152 Cuda

·V8.2.0 cuDNN

·2.8.0 tensorflow-gpu

·2.8.0 keras

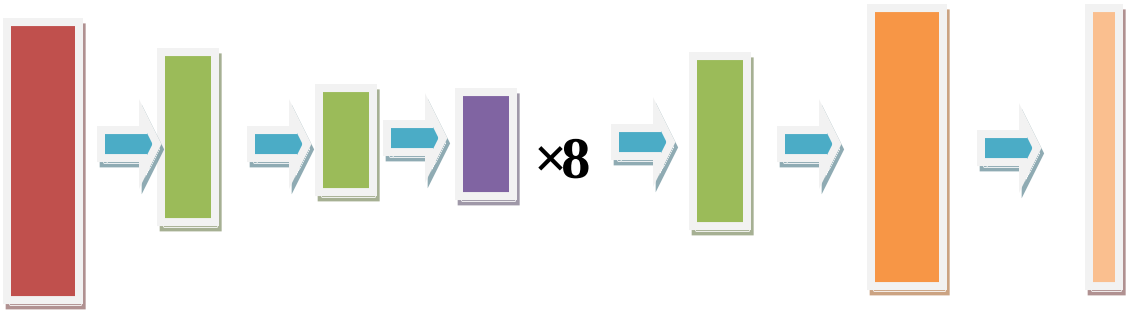
Installation and configuration

Cuda, cuDNN installation packages are downloaded from the official website and installed in the local environment. Tensorflow GPU version and keras are installed in the command line window under Anaconda. Specifically, in https://developer.nvidia.com/cuda-toolkit-archive, you can get desired Cuda version. In https://developer.nvidia.com/rdp/cudnn-archive, you can get desired CuDNN version. For tensorflow, use ‘pip install tensorflow -gpu==2.8.0’ to download it. For keras, currently(2022.8) you can simply use ‘pip install keras’. Specify keras version according to your need.

Techniques

Traditional GAN model uses an encoder structure as generator and stacks convolutional layers to build the discriminator. Generator acts as a role to produce desired outputs, such as a specific kind of images. Discriminator tries to distinguish generated fake images and targeted images as much as possible. At first generator receives random noises and produces respective image output. After that, these output images, together with targeted images, will be sent to discriminator to judge their labels. This result will then be used to optimize both generator and discriminator. According to relationships between generator and discriminator, after enough iteration, the model will reach convergence and the generator will produce desired images.

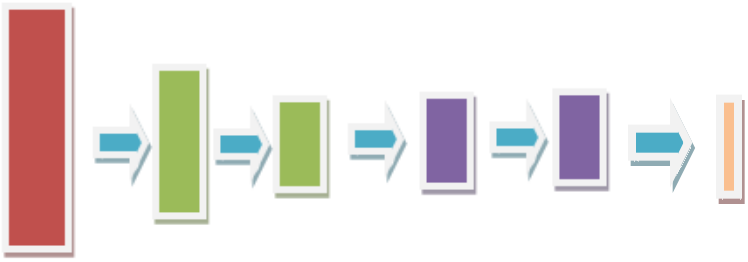
In style transfer domain, such GAN structure is modified. Since we are solving an image-to-image task——given a batch of images and produce respective anime images, generator should apply standard autoencoder structure. At the same time, since the style of an image is heavily decided by different local areas of it, such as clear edges and smooth texture, the discriminator can not simply produce a scalar as output. Instead, it will produce a small feature map, each pixel of which represents local areas of an image. The model structure we adopted is as follows.



Generator

From left to right, the meaning of each unit block is:

1.Input image.
2.Convolutional layer(Conv) with kernel size(k) 7x7, feature maps(f) 64, strides(s) 1, padding=same. After that follows an Instance Normalization layer proposed by some scholars, and a ReLU layer.
3.Conv with k3, n128, s2, padding=same; followed by conv with k3, n128, s1, padding=same; normalization layer; ReLU layer.
4.Residual layer: conv with k3, n256, s1, padding=same; normalization layer; ReLU layer; conv with k3, n256, s1, padding=same; padding=same; normalization layer. The output of this layer will also add the input of this layer by executing element-wise addition.
5.ConvTranspose with k3, n128, s2, padding=same; ConvTranspose with k3, n128, s1, padding=same. Normalization layer; ReLU layer.
6.ConvTranspose with k3, n64, s2, padding=same; ConvTranspose with k3, n64, s1, padding=same. Normalization layer; ReLU layer.
7.Conv with k7, n3, s1, padding=same.



Discriminator

From left to right, the meaning of each unit block is:

1.Input image.
2.Conv with k3, n32, s1, padding=same; ReLU layer.
3.Conv with k3, n64, s2, padding=same; ReLU layer. Conv with k3, n128, s1, padding=same; normalization layer; ReLU layer.
4.Conv with k3, n128, s2, padding=same; ReLU layer. Conv with k3, n256, s1, padding=same; normalization layer; ReLU layer.
5.Conv with k3, n256, s1, padding=same; normalization layer; ReLU layer.
6.Conv with k3, n1, s1, padding=same

Loss function is also modified. Content loss and style loss are also added to generator loss. After passing the generated image and the original image to VGG19 with modified top structure, we will get two different feature maps. By computing their Gram Matrix and mean squared error with these two images, we will get respective content loss. By switching the original image to anime image, we will similarly compute style loss.

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}. \qquad (3)$$

The formula above is the definition of Gram Matrix. A value in the Gram matrix corresponding to the feature graph F, i and j are channel indexes, and k is the position index in the space composed of the width and height of the feature graph

Since we also use smoothed anime images as input to discriminator, our discriminator loss also has to take it into consideration. Here is the final loss function.

$$\mathcal{L}(G, D) = \mathcal{L}_{adv}(G, D) + \omega \mathcal{L}_{con}(G, D)$$

$$\mathcal{L}_{adv}(G,D) = \mathbb{E}_{c_i \sim S_{data}(c)}[\log D(c_i)] + \mathbb{E}_{e_j \sim S_{data}(e)}[\log(1 - D(e_j))] + \mathbb{E}_{p_k \sim S_{data}(p)}[\log(1 - D(G(p_k)))].$$

$$\mathcal{L}_{con}(G,D) = \mathbb{E}_{p_i \sim S_{data}(p)}[||VGG_l(G(p_i)) - VGG_l(p_i)||_1]$$

Training tutorial

This part of content is the same as the working demo.

1.Download all my codes and the supporting training dataset. The functions of each Python file in the folder are as follows:

dataloader: The function in it is imported and used in 'model' file to build an iterator, which generate a batch of file address of the pictures each time. Training phase requires such kind of data loader.

gan_loss: The functions in it define loss functions for optimizing GAN model, including generator loss, discriminator loss for general GAN model, and content loss, style loss for our specific style transfer task.

unit_layer: The functions in it define unit structure for our JanpanimationGAN model

video2image: This file can be used to convert a video to many images. This program is not written by me. I use this tool provided by others to convert videos into many pictures to build my data set. This method is realized by extracting video key frames. The author and source of the program have been marked in the python file.

image_cutting: This program is used to crop pictures extracted from video. Since the size of the original pictures are too large, the program will randomly cut out 6 pictures in the six areas divided in each picture. After running this program, we will get a set of RGB images, all of which have the size of 256×256.

Image_smoothing: Smooth edges of anime images, in order to let the model learn features of this kind of images better.

model: The main part of this project. The program includes the code of Gan model construction and training.

test_on_new_img: Use the trained model to realize animation style conversion on your own images.

2.(Optional) Build your own dataset: By using 'video2image' and 'image_cutting' programs mentioned above, you can converted your videos into suitable datasets. To do this, open each file and change video file address and new data set storage address mentioned in these two files. Note that you may have to manually delete some training images, if you build your dataset in this way.

This is because some images in the dataset may be basically the same, and some images might be worthless for training the model. After that, run ‘image_smoothing’ program to get smoothed anime images as dataset.

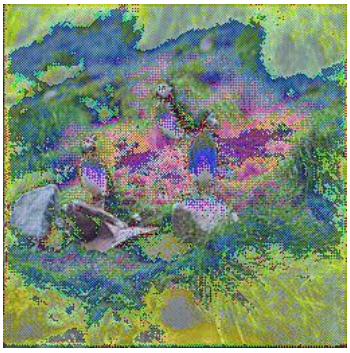
3.If you want to achieve better effects, train the GAN model for several epochs. Adjust the file directory specified when reading the data set in the "model" program, specify the storage directory of model parameters and prediction results, and run the "train" program. According to your needs, you can freely set the training epoch. It is suggested to run pretraining process for 2 epochs, and run training process for 150 epochs and above.

4.Put the image you want to achieve style conversion into the / Pretrain / origin directory. After specifying the storage directory of the new file, run the program named "test_on_new_img" to achieve style transfer.

For more detailed personalized parameter settings, please refer to the annotations in the program files. You can simply use default parameters.

Result

Here is the result after training 28 epochs on GAN and 2 epochs pretraining on generator. Although training epochs are not so sufficient comparing to complexity of this model, our GAN already started to produce some reasonable outputs for inputs with specific contents.



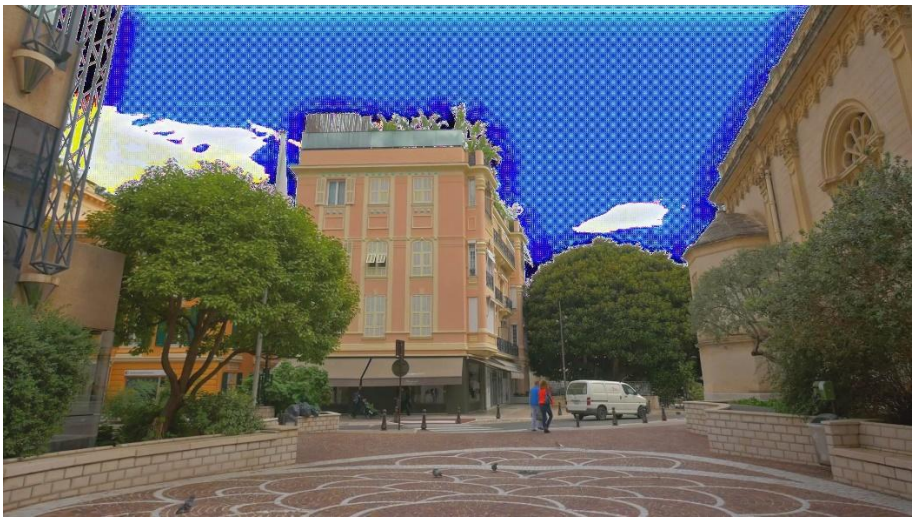
Pretrain generator after 2 epochs, output of generator

Output of generator after13 training epochs

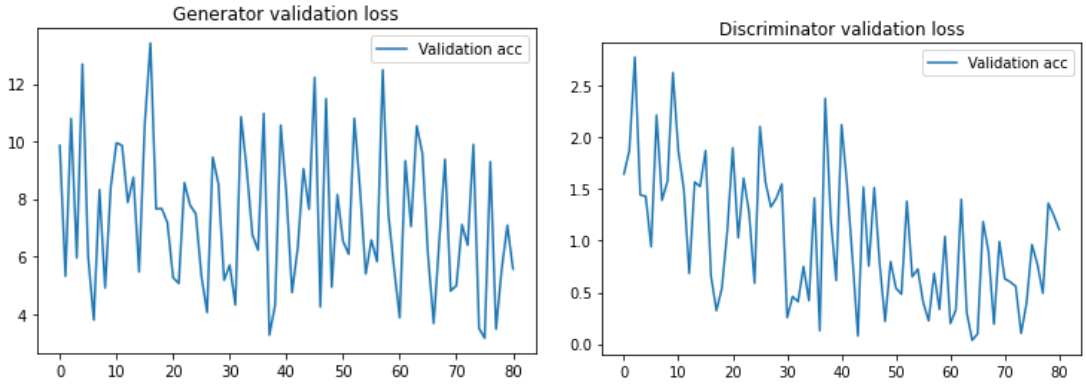
Prediction made by generator after 28 training epochs

It can be seen that for specific contents such as ocean and sky, the generator will learn how to transfer anime style to real-life images more quickly. This is might because in my anime dataset, many anime images also have ocean and sky as contents. Therefore, for other kinds of training images, the generator outputs become less desirable.

Output of generator after17 training epochs



Prediction made by generator after 28 training epochs



From the results above, currently the model have these problems:

1. Tend to maintain original style of input images. Maybe the weight for content loss is set too high.

2. Tend to recognize and modify specific areas of images, such as areas with ocean and sky as contents. This is due to the lack of more training epochs.

3. For some contents, it failed to transfer anime style. The results of partial style migration show that the model simply adds a layer of filters after splitting out its recognizable areas. Such kind of filters still do not make sense at early training period. This can be seen from the prediction made by generator after 28 training epochs.

Introspect and conclusion

In general, the design of model structure and loss function are proved to work, even with limited training epochs. However, their complexity makes it hard to train the model well enough without enough GPU resources. Even though I use RTX 3060 laptop, each pretraining epoch takes nearly 20 minutes, and each training epoch takes around 40 minutes. In order to achieve best effects, maybe I should train the model for 200 epochs. On that case, I have to train it at least 5 days without closing my computer. It will also becomes extremely hard to tune hyperparameters of the model.

Besides, the way of presenting and recording loss value seemed not to work so well. Although I tried to use tensorflow APIs such as tf.summary.metric() to record loss values during training, I had to manually open the file containing these results, record them again to plot loss function in the end, due to my insufficient understanding of tensorflow APIs.

As for hyperparamer selection, it seems that my chosen learning rate for generator is a little bit high. The oscillation of generator loss value is obvious. In addition, it may be better to appropriately reduce the weight of the content loss in the generator loss. I set this weight to 10, but this may make the generator tend to produce the same content as the original image as the output in the early stage.

In conclusion, we I learned from this project is that, firstly, tasks related to computer vision field may rely heavily on computation resources. In future research towards deep learning and

computer vision, maybe I should join in one research group in my university and utilize provided GPU resources. Secondly, it is worth inspecting all codes, test each small parts of the program before formally running the whole of it. Running codes and meeting an error halfway seems to waste more time than spending time examining codes in advance. Moreover, sometimes it is harder to debug during training. Instead, one should carefully write each line of code at the first time.

Dataset URL

https://github.com/SheeranJ/JapananimeGAN