Project Plan of Deep Learning 2021 fall
CycleGAN to achieve the image style translation from the real world photos to the artistic style of Van Gogh

Zihua Fang
NetID: zf2054

Image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs. However, usually we are not able to get the paired samples easily. And with CycleGAN, we can achieve the translation without input-output pairs. In this model, we only need to input two datasets of different types of objects to achieve the image-to-image translation.

Figure 1: Transferring from real world photos into GATV style

And there are many successful and useful applications of CycleGAN, including collection style transfer, object transfiguration, season transfer and photo enhancement. And I was inspired by these applications and came up with my idea, which is to transfer some daily pictures we are pretty familiar with into some really fantasy styles, just like the anime styles, the video games styles or some artist's styles. At first, I attended to finish a translation from the real world photos to the video game styles. But the problem is the only suitable dataset I found is GTAV dataset and the video style of GTAV is pretty realistic which lead to the results trained on this dataset is not obviously different with the original pictures. So in order to derive a more obvious visual difference, I determine to finish a translation from the real world pictures to the artistic style of Van Gogh.

literature survey about Image-to-image translation

The idea of image-to-image conversion can be traced back at least to Hertzmann et al.'s Image Analogies [3], who used a non-parametric texture model on a single input-output training image pair. And many recent methods use a data set of input-output examples to learn a parametric translation function with CNNs.

Several other methods also deal with unpaired settings, with the goal of associating two data domains: X and Y. A Bayesian framework [7] is proposed by Rosales et al., which includes a prior based on a patch-based Markov random field calculated from a source image and a likelihood term obtained from multiple style images. Recently, CoGAN [5] and cross-modal scene networks [1] use weight sharing strategies to learn universal representations across domains.

And another method we can not ignore when we talk about the CycleGAN is pix2pix [4]. The CycleGAN approach is built on the pix2pix framework, which uses a conditional generative adversarial network to learn a mapping from input to output images. Pix2pix model is also based on GAN. And it needs the paired data sets to train the mapping. Maybe due to this reason in a certain degree, it did not attract the attention of as many people as cycleGAN, which just appeared later and did not need the paired data set.

Model Details: CycleGAN

After some reading about the CycleGAN [9], I got some basic ideas of the training method of this image-to-image translation without input-output pairs. There are two generators in the CycleGAN model, $Generator_{A-B}$ and $Generator_{B-A}$. The first generator transforms an input image from domain D_A to domain D_B. The new generated image is then fed to the second generator $Generator_{B-A}$ which converts it back into an image from the original domain D_A. So, this output image must be close to the original input image to define a meaningful mapping that is absent in and unpaired dataset.

As Figure 1, two inputs are fed into each discriminator(one is original image corresponding to that domain and other is the generated image via a generator) and the job of discriminator is to distinguish between them, so that discriminator is able to defy the adversary (in this case gener-

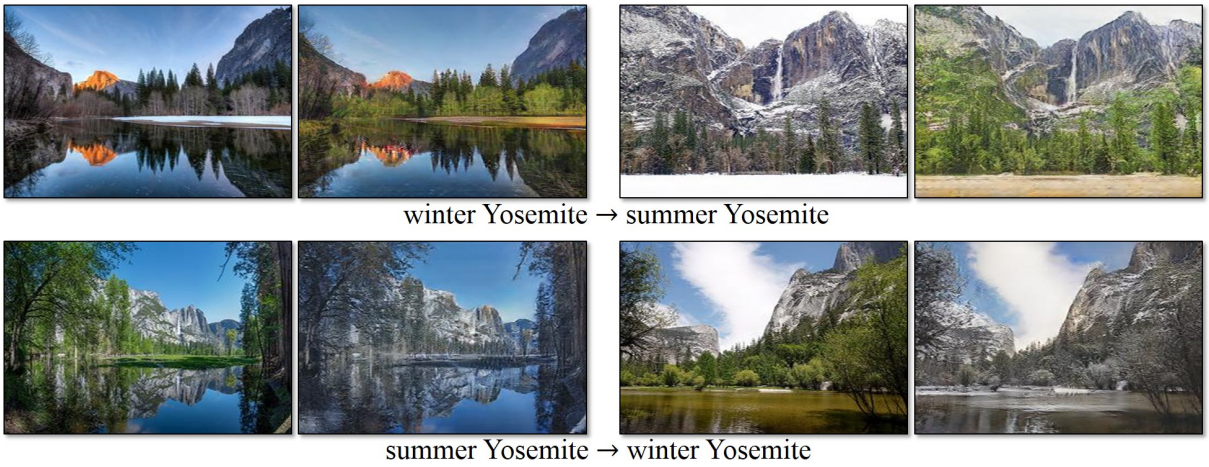

Figure 2: Transferring input images between seasons

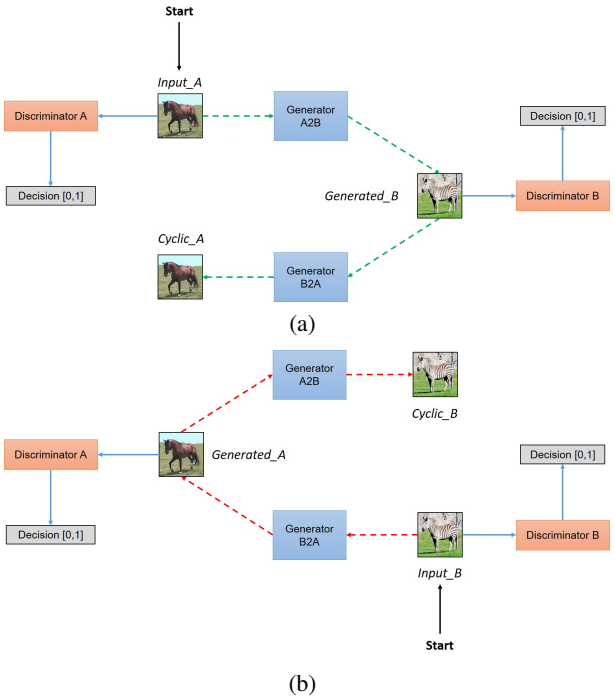

Figure 3: Simplified view of CycleGAN architecture

ator) and reject images generated by it. While the generator would like to make sure that these images get accepted by the discriminator, so it will try to generate images which are very close to original images in Class D_B . (In fact, the generator and discriminator are actually playing a game whose Nash equilibrium is achieved when the generator's distribution becomes same as the desired distribution)

Loss Function

And another siginificant thing is about the loss function. In the CycleGAN model, there exist two types of loss function: adversarial losses [2] for matching the distribution of generated images to the data distribution in the target domain; and cycle consistency losses to prevent the learned mappings G and F from contradicting each other.

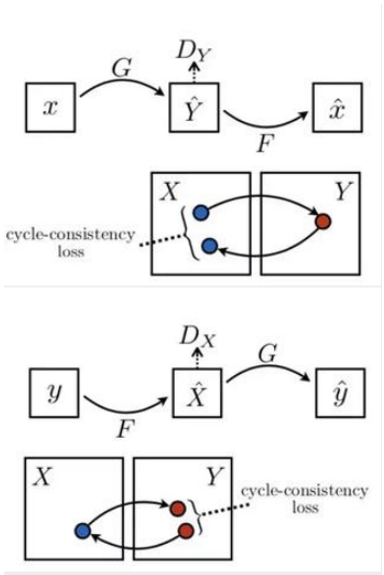

Figure 4: forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

The model has two data domins: X and Y. And correspondingly there are two mapping functions: $G : X \rightarrow Y$ and $F : Y \rightarrow X$ and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y , and vice versa for D_X and F. The model applies adversarial losses to both map-

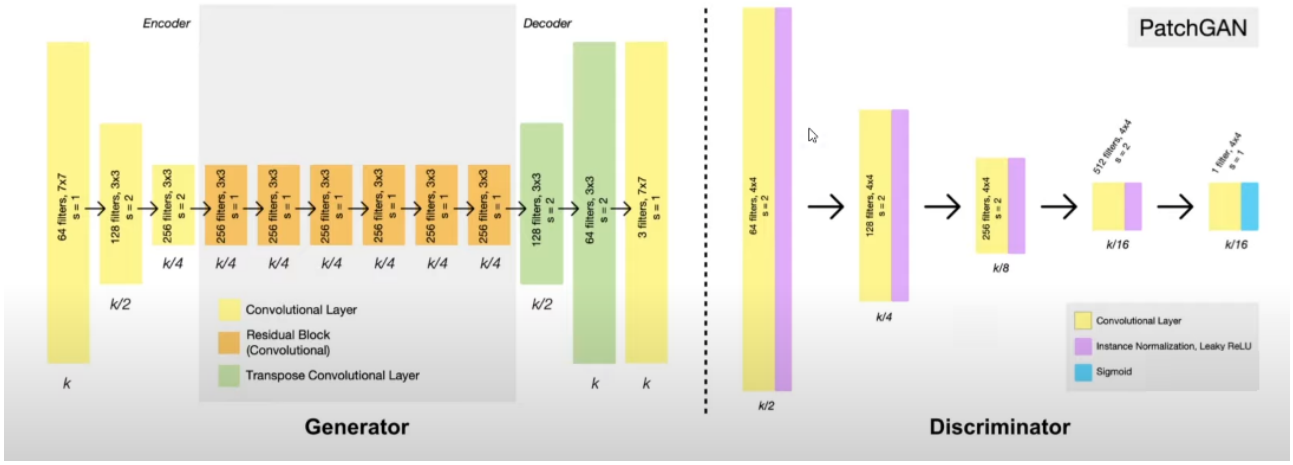

Figure 5: Model summary: generator and discriminator constructions

ping functions. For mapping function G, the object is:

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}[log(1 - D_Y(y))]$$

where G tries to generate images $G(x)$ that look similar to images from domain Y , while D_Y aims to distinguish between translated samples $G(x)$ and real samples y. The adversarial loss for mapping function $F : Y \rightarrow X$ is similar: $\mathcal{L}_{GAN}(F, D_X, Y, X)$.

To further regularize the mappings, the model introduces two cycle consistency losses that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started, as shown in Figure 3.

The model incentivize the behavior using the cycle consistency loss:

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[||F(G(x)) - x||_1] + \mathbb{E}_{y \sim p_{data}(y)}[||G(F(y)) - y||_1]$$

So, the total Loss function will be:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F)$$

Dataset

I have found some datasets for my program. The first one is Makoto. It is a dataset of pictures in Makoto Shinkai's anime works. There are about 260 pictures. The second one is the GTA Dataset [8], which is a much larger dataset than the first one. It is about the game GTA. And the third one is the Cityscapes Dataset, a new large-scale dataset that contains a diverse set of stereo video sequences recorded in street scenes from 50 different cities. And I also find some other datasets of some artists' painting images, like the Monet's painting and Van Gogh's painting. I also mentioned that there is a dataset about the Disney, and someone has already finished the translation from real person photos to Disney style characters and it seems really nice.

And finally, after I determined my project content, achieving a translation from the real world pictures to the art style of Van Gogh, I still need some real world photographs dataset.

So now, the final two datasets used in my program are the one of Van Gogh's paintings(401 images) and the one of the real world photographs(6853 images). Specially, the dataset of Van Gogh's paintings is downloaded from Wikiart.org. The dataset is processed by Junyan Zhu et al. [9]. In the dataset, some artworks that were sketches or too obscene were pruned by hand and the dataset included only Van Gogh's later works that represent his most recognizable artistic style.

Implementation

To implement the CycleGAN model, I build the network as Junyan Zhu et al. [9] do. The input images are scaled to 256×256 pixels. And this network contains three convolutions, 9 residual blocks, two fractionally-strided convolutions with stride $\frac{1}{2}$, and one convolution that maps features to RGB. For the discriminator networks, the model uses 70×70 PatchGANs [6], which aim to classify whether 70×70 overlapping image patches are real or fake. There are also some other techniques to improve the performance of the model in Junyan Zhu et al.'s work [9]. They replace the negative log likelihood objective by a least-squares loss in the loss function $\mathcal{L}_{GAN}$. This loss is more stable during training and generates higher quality results.

The following parts are the details when I build the whole program. I have read many different versions of the CycleGAN code, like the original code from the paper [9], the cycleGAN program code based on Paddlepaddle et al. Finally, I decided to build my code on the base frame of Aladdin

Persson's.

Model constructions

Thinking about the whole model, we need two parts, the generator and the discriminator. The summary of the model construction is shown in figure 5.

For the generator, we need to do basically two down samplings with a stride of two on the convolution layers. And then, a bunch of residual blocks are followed. The next part of the generator is the transpose convolution layers which do the up samplings. And the end is the output layer to map the output results to RGB channels.

Compared to generator, the discriminator will be simpler. There are four convolution layers with the stride of 2. And the following is the PatchGANs [6]. The output is not a single scalar value between 0 and 1, it will be a grid of values and each of the value of the grid will be between 0 and 1. Each of the value only pay attention to one certain patch of the original image. That is why it is called PatchGAN.

Discriminator Before building the discriminator, I defined a common block class of the convolution layers, so that it can be called later more efficiently. we need the in channel number, out channel number and stride as our input parameters. As for some other parameters, I followed the instructions in the paper. Like, for the convolution layers, the kernel size is set to 4 which is always the case. The padding is 1, bias is true and the padding model is reflect as the paper mentioned. With a padding mode as reflect, it helps to reduce the artifacts. What following the convolution layer are the instance norm layers and leaky Relu layers.

At the begin of the discriminator is the initial layer, which do barely the same thing as the common block class as before, except the instance norm layer. This layer changes the input images from RGB 3 channels to 64 channels. And the following are 2 normal convolution layers with stride of 2 and one with stride of 1. The last is the output layers. Unlike generator which are supposed to output the fake images, here the discriminator just need to output the values indicating whether the image is fake or ture. The sigmoid layer will ensure the value between 0 and 1.

Generator It will be more complcated fro building the generator. As we have mentioned before, the generator has three parts down sampling layers, residual layers and up sampling layers. For up sampling layers, we need to change the concolution layers to the convolution transpose layers. And some parameters are the same with the discriminator, like the padding mode. One of the different things is about the activation. In the residual layers, the paper does a really interesting thing, in the even-numbered layers the activation is set to false while other are ture. Actually I do not know why this makes sense and I just followed it.

So, when building the generator, the first one is the initial part with a convolution layer and Relu layer. And then the down sampling part. There are 2 convolution layers each with an instance norm layer and a Relu layer. The third part is the residual part. Here we need to set the number of residual layers as a parameter. The default value is 9. And as I have mentioned, the even-numbered layers shouldn't use Relu layers for activation. In the end, there is up sampling part, which is respective to the down sampling part. 2 convolution transpose layers with padding 1 and out output padding 1. And in order to output the fake images, we still need one more layer to convert the results into RGB channel. After all these layers the output size is set to the same as the input size.

After all these generator and discriminator model building, now we have successfully constructed the CycleGAN network model.

Dataset loading

For the program, there are four parts of the whole dataset, 2 training sets and 2 test sets. Both images of photos and images from Van Gogh have one training set and one testing set to finish the style transferring work.

The first problem is about the length of the datasets. Since we do not need the paired images to train our model, the length of the two training datasets or the testing datasets can be different. At first, I thought I can select two images from both of the datasets randomly. However, in order to simplify the processes, I just save the lengths of both datasets and then do a modulus operation to in case of the index out of bounds. Maybe there will be some images in the smaller data set which will be shown more often than others. I'm not sure whether this will affect the results.

And other things are quite same like we usually do, such as loading the the images, converting images to array.

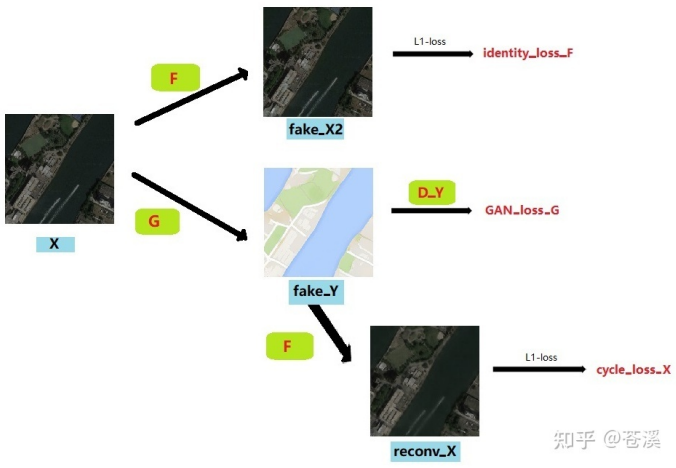

Figure 6: Three kinds of loss in CycleGAN.

Training code

In order to finish that translation between the photo and Van Gogh's artistic style in both direction, in the training part we need to build two generators and two discriminators. So, it is easy to do with the model function we have defined in the model building part.

And before we really go into the training process, since I intend to do my training in the Colab, it is important to deal with the checkpoints. Actually, at my first some times

of training, the connection duration time of the colab did be a problem, when I didn't save the checkpoints. After finishing the saving and loading checkpoint code and other preprocess like load the dataset, initialize the dataloader, it is time to figure out how to train the whole model.

The first thing is about the loss functions. As mentioned before, there are actually two kinds of loss functions in the CycleGAN model, one is the cycle consistency loss and the other is the identity loss. In this program, for the cycle consistency loss we will use L1 loss. And for the identity loss, we choose the mean squared error.

In each epoch of training, the first thing is the loop. The program will go through all the data images to train the model. In each iteration, the program will first train the discriminators. With the current generators, the program will produce two fake pictures, one is the fake photo and the other one is the fake pictures in Van Gogh's style. With the corresponding discriminator, we can calculate out two results from the original input images and the fake images we produced. For the results from the real input, we can calculate the real identity loss value with the function $torch.ones\_like()$. And fake loss value with the fake results and $torch.zeros\_like()$ function. And then add them up. We can finish this calculation in both direction. And there will be two loss values in both direction. The final discriminator loss value will be the sum of these two. And then there are backward, step and update operations.

And the next thing in the iteration is to train the generators. Things are really similar with the process in training the discriminator. things are just reverse compared to computing in the discriminator part. With corresponding discriminator, we can get two results from the two images. And then calculate the MSE loss like the we do in the discriminator part. Add the two loss up and then do it in the other direction of the translation. The final generator loss value will be the sum of the two values in two direction, photo to Van Gogh's style and reverse.

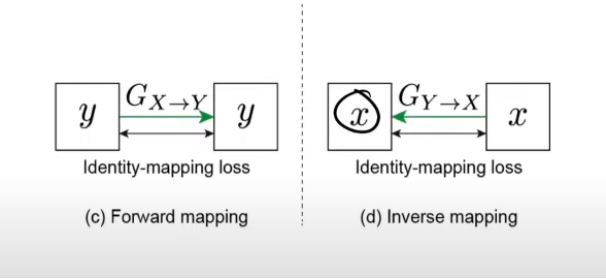

Figure 7: Identity Loss: Suppose that the current transferring direction is from A to B. There should be a generator G_{A-B}. If we calculate the generator with a input images P, which is just a images in domain B. Then the generator G_{A-B} should do nothing, since the input is already B. The output P' should be the same with the input. The identity loss is just the difference between the input images P and the output images P'.

Here, the difference is that the program should calculate two kind of loss. Now, we have the two fake images and two real images. For the cycle consistency loss, the program will convert the two fake images back into their original style with the corresponding generator. We can call these two images as the cycle images. And with the L1 loss function, we can easily calculate out two cycle consistency loss between the cycle images and the original input images. And for the identity loss, what we do is to calculate the result of the generator which is supposed to generate a photo with its input is already a real photo. And then calculate the L1 loss between the input real photo and produced fake photo. It is kind of confusing to describe in words, but it is quite clear in code. And there is also a short interpretation in figure 6. And finally, there will be three kinds of loss values, add them up with their corresponding multiplier we defined, and we will get the loss value of the generator.

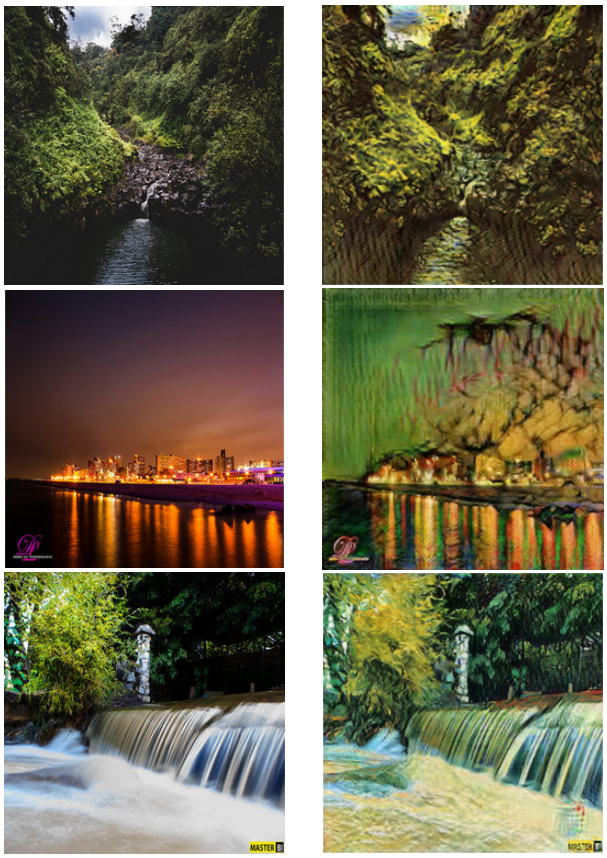

Figure 8: Some results from the pretrained model from the official page of the paper [9]

One thing I noticed is that in the original code of the paper, they set the third part of the loss value of the generator training part to zero. The paper said this loss function is to make sure the color of the output images will be consistent to the input images. I am not sure whether I should reserve it. Actually, I think this loss function can do much more than just making the color consistent. So at first, I trained my model with this part as 0 for around 15 epochs. And then I adjusted the parameter as 0.1 in the next epochs.

And at the final part of the training function, I save the two fake images every 200 iterations.

Training process

I trained my model in Colab. One important thing about training in Colab is to do the backup. Actually, at my first some times of training, the connection duration time of the colab did be a problem, when I didn't save the checkpoints and lost my trained model and results since the disconnection. One thing to solve this problem is to mount the Google Drive in the colab. Such that, you can save the images and checkpoints to the google drive in time and restart from the previous work even if you lost the connection to colab or the reach the max connection time.

Pretrained model results

Before I built my own code, I once tried the official colab code from the paper. I downloaded the pretrained model from the official page and then attempt to do some training. Each epoch cost around 45 mins with the official colab code. So due to time cost, I decided not to continue training this model and to build my code to train from sketch. However, I saved some results images from the pretrained model.Here are some examples in figure 8.

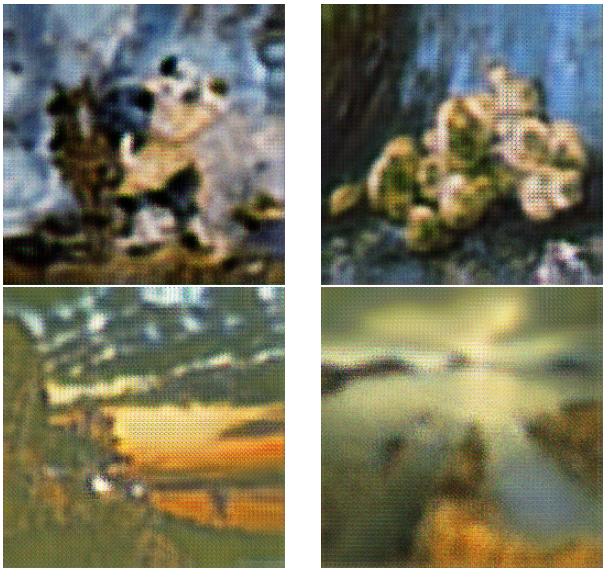

Figure 9: Several results from the first two epochs. The first line of the images fake photo images generated from Van Gogh's pictures. And the second line is the fake pictures of Van Gogh's artistic style generated from real photos.

Here we can see that in the first image result in figure 8, the bound between the forest and the river is not obvious and the river become green. And in the second result, there are many unnecessary colors in the result. And it seems the green color always appears where it shouldn't be, the river and the dark sky.

Figure 10: The original input images and different output images. The images in line 1 are the original images, real images. The images in the second line are the fake images, transferred into the other style. And the third line is the cycle images, which are generated from the fake images and transferred back to their original style. The last line is the identity images, which are generated from the real images with the generator directing to their own style. Pictures in the last line should be the same with the first line.

Figure 11: Samples from different epochs. Photo to Van Gogh's style.

Figure 12: Samples from different epochs. Van Gogh's style to photo.

Results

My code was run in colab pro. It took around one and a half hour to finish one epoch. On the basic Colab, it will take around 4 hours to finish one epoch. To reach the same quality results like the paper, I was suggested to train the model for 200 epochs and for the first 100 epochs set the learning rate as 0.0002 and for the last 100 epochs linearly decay the rate to zero. That will be a huge time cost. And till I finish this program report, my code is still running in the colab. But fortunately, I got some great results from the finished around 50 epochs.

Since my model is trained from sketch, the results of the first several epochs are some fuzzy graphics that only have basic shapes. Shown in figure 9.

And in figure 10, there are original input images and different kinds of output images from the model. We can find that the in the two fake images, the result images transferred from photos to Van Gogh's style seems better then the converse direction result. And this is like the common situation in the whole training process. I think this may because of the style consistence of the two style dataset. In Van Gogh's painting dataset, the paintings are selected. In the dataset, some artworks that were sketches or too obscene were pruned by hand and the dataset included only Van Gogh's later works that represent his most recognizable artistic style. However, the other dataset contains numerous different kinds of photos. So the over all version style of Van Gogh's dataset is more consistent, which may lead the model to learn the style easier.

And in line 3, compared to the original images, the cycle images lost many details but keep in the same style and seems pretty similar with the original images. And identity images are quiet the same with the original images. There are only insignificant differences.

For figure 11 and 12, those are the same images produced in different epochs. The images in the same column are from the same epochs. And the first column is the original images.

Till now, I have finished about 50 epochs. From the results above, it seems that my model still need more epochs to reach the paper's transferring effect. Here are some samples from the paper in figure 13.

Problems

From the above results we can find that in some results, there sometimes will be some obtrusive black lines. Maybe this problem is about the training epochs. but I also found there are many people have similar question, there are some discontinuous colors or lines in their results. Even the results of the paper also has this problem. And I noticed that someone tried to use the attention method to solve this problem. And the results seem great. His work is to finish a translation of peoples' pictures cross gender. I have no idea about whether it will work on the style transferring.

And another thing is about the different effect in the two directions. We can find that the in the two fake images, the result images transferred from photos to Van Gogh's style seems better then the converse direction result. And this is like the common situation in the whole training process. I

Figure 13: Several results from the paper.

think this may because of the style consistence of the two style dataset. In Van Gogh's painting dataset, the paintings are selected. In the dataset, some artworks that were sketches or too obscene were pruned by hand and the dataset included only Van Gogh's later works that represent his most recognizable artistic style. However, the other dataset contains numerous different kinds of photos. So the over all version style of Van Gogh's dataset is more consistent, which may lead the model to learn the style easier.

Figure 14: Bad result sample.

References

[1] Yusuf Aytar, Lluis Castrejon, Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Cross-modal scene networks. *IEEE transactions on pattern analysis and machine intelligence*, 40(10):2303–2314, 2017.

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[3] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analo-

gies. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 327–340, New York, NY, USA, 2001. Association for Computing Machinery.

[4] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

[5] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. In *Advances in neural information processing systems*, pages 700–708, 2017.

[6] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016.

[7] Resales, Achan, and Frey. Unsupervised image translation. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 472–478 vol.1, 2003.

[8] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision (ECCV)*, volume 9906 of *LNCS*, pages 102–118. Springer International Publishing, 2016.

[9] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.