Full-body Anime Character Generation using Generative Adversarial Networks

Qiucan Huang
HKUST
Clear Water Bay, Hong Kong
qhuangag@connect.ust.hk

Sida Chen
HKUST
Clear Water Bay, Hong Kong
schenci@connect.ust.hk

Yifei Lin
HKUST
Clear Water Bay, Hong Kong
ylinbz@connect.ust.hk

Abstract

Automatic generation of facial images has been well studied after the Generative Adversarial Network (GAN) by Goodfellow et al. [1] came out. There exist some attempts applying the GAN model to the problem of generating facial images of anime characters, but few of the existing work try to generate full body images. In this work, we explore the training of GAN models specialized on full-body anime character generation. We address the issue from both the data and the model aspect, by using a clean and well-suited full body dataset and leverage proper, empirical application of some variation of GAN. We also use the idea of progressive generation [3] by generating in an order of line sketch, colorization, and ornaments. Moreover, to assist people with anime character design, we use the data collected from mobile games such as Girls' Frontline, which makes the output style more similar with modern anime characters.

1. Introduction

Many people like anime characters and want to create their own ones. However, it takes many efforts to learn the skill of drawing a custom character, after which designing their own characters becomes available. To bridge this gap, automatic generation of anime characters offers an opportunity to bring a custom character into existence without the need of professional skill. Besides the benefits for normal fans, a professional anime creator may take advantages of the automatic generation for inspiration on animation and game character design; a mobile character reduces designing costs in game production.

In this report, we propose a model that produces full-body anime characters. Our work can be described as three: dataset collection and processing, a suitable GAN model, based on Style GAN [5], and our approach to train a GAN from images.

2. Related Work

Generative Adversarial Network (GAN) [1], proposed by Goodfellow et al., shows impressive results in image generation, image transfer, super-resolution, and many other generation tasks. The essence of GAN can be described as training a generator model and a discriminator model simultaneously, where the discriminator model tries to distinguish the real example, sampled from ground-truth images, from the samples generated by the generator. On the other hand, the generator tries to produce realistic samples that the discriminator is unable to distinguish from the ground-truth samples. Above idea can be described as an adversarial loss that applied to both generator and discriminator in the actual training process, which effectively encourages outputs of the generator to be similar to the original data distribution.

Some researchers had done some work in this area. Our method is based on one of research, which is style GAN [5]. Style GAN was proposed by a team from Nvidia. Previous works usually inject the latent vector to the generator directly. while the style GAN tries to find laten vector to a large space and inject it to all the layers. Besides, we are also using a technique in style GAN called progressive generation [3]. Instead of training the whole network, the progressive generation tries to train the lower layers, which generate low resolution images and then fix that area. Fix the lower layer before training the higher layer, this technique helps training become more efficient while the quality of image isn't affected much.

3. Data

We gathered the data from hot anime games on the mobile market, by extracting images directly from ".apk" files. Most of the extracted pictures are flawed, as they need further processing to get the original picture. we can find an original image whose Type is Texture2D and the other is a transparency background image whose Type is Texture2D or Sprite. Then we apply channel-separated images combination to get the original image. The result is shown in Figure 1.

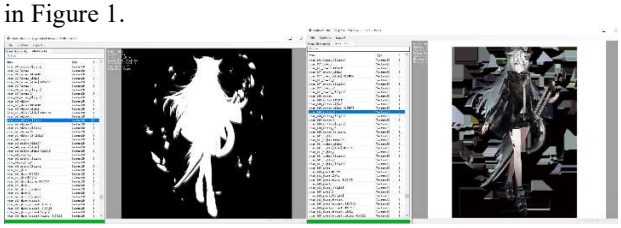
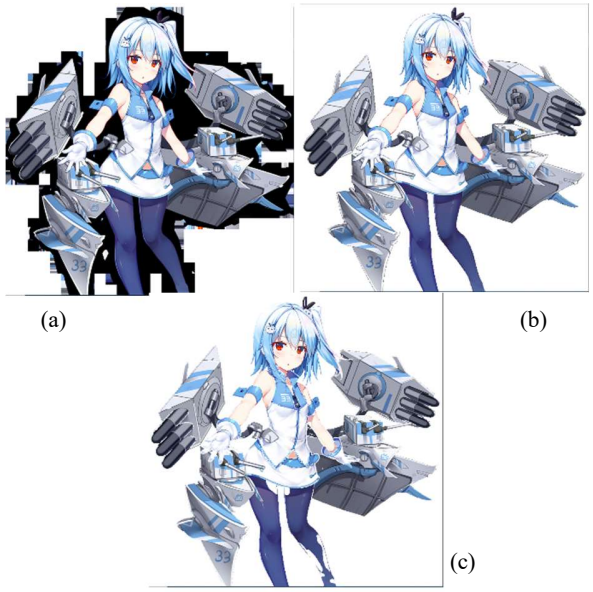


Figure 1: Extraction steps of images from ".apk" files.

Images we got from games are in two types, one is PNG format with 3 channels, RGB, another is PNG format with 4 channels, RGBA. We use OPENCV[1] library for some basic image operations, such as reading images, getting the channel information, and file type conversion. We keep the images with 3 channels. For the images with 4 channels, we use a script to delete Alpha channels. In the current game industry, the Alpha channel is used to create shade and gradual changing. Figure 2 (a) shows the image with all information, Figure 2 (b) shows the image with low Alpha channel information, and Figure 2 (c) shows the image without Alpha channel information.



(a) (b) (c)

Figure 2: (a) show the image with all information, (b) shows the image with low Alpha channel information, and (c) shows the image without Alpha channel information.

To get better results, we write a script[2] to convert all images into 3 channels and save them.

Another dataset we used is anime face images crop from previous full anime body images. Here we find a pre-trained anime face detection network, "lbpcascade_animeface"[3] to detect anime faces.

There are two problems with this model. Firstly, the output files don't contain all heads. We slightly change the model and output a larger range of the faces detected by model to get half body. Comparing Figure 3 (a) and Figure 3 (b), Figure 3 (a) shows the original output of the files, and Figure 3 (b) shows the final dataset we used.

(a) (b)

Figure 3: (a) shows the original output of the files, (b) shows the final dataset we used

Another issue is misclassification. The model may output some images that aren't faces. Figure 4 (a) shows the correct face output and Figure 4 (b) shows the wrong output. We use a script[4] to show pictures in a folder sequentially and delete images based on human input.

(a) (b)

Figure 4: (a) shows the correct face output, (b) shows the correct face output

4. Methods

Our target is generating full body anime characters. Nowadays, the best network architecture for image generation is GAN [1], As we mentioned in class, GAN contains two networks, generator, and discriminator. The Generator tries to generate images and the discriminator tries to classify which image is fake. GAN uses game theory; the generator tries to fool the discriminator and the discriminator tries to classify the image. The loss is defined as

$$\min_{\theta_g}\max_{\theta_d}\left[\mathrm{E}_{x\sim p_{data}}\log D_{\theta_d}(x)+\mathrm{E}_{z\sim p(z)}\log\left(1-D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right]. \quad (1)$$

To train a network to generate high resolution pictures will take lots of time. In one of our experiment, training a network to generate a 256*256 image need around 20 days in 3 GPUs. To accelerate the training process, we consider using a technique proposed in [3] called progressive

[1] https://github.com/opencv/opencv

[2] https://github.com/NineLivesCat/COMP4471_2020Fall/blob/main/script/PNGtoJPEG.py

[3] https://github.com/nagadomi/lbpcascade_animeface

[4] https://github.com/NineLivesCat/COMP4471_2020Fall/blob/main/script/human_filter.py

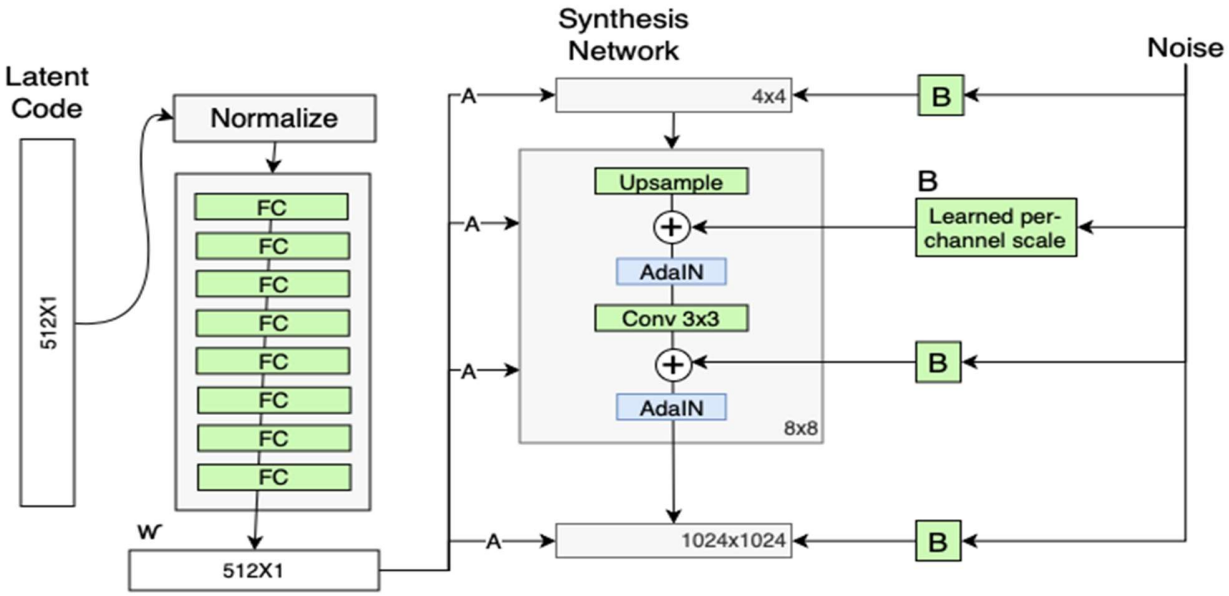


Figure 5: the structure of StyleGAN [5]

growing. Progressive growing is similar to the idea of pretrain and transfer. The idea of pretrain and transfer is training a model in another dataset, fixing some layer of this model, and trying to train the unfix layer in the target dataset. Here progressive growing tries to down samples the dataset to low resolution such as 8*8, then train the image to generate low resolution images. After training is finished in a low-resolution layer, the low-resolution layer that finished training will be fixed. Then the next layer will train based on the up-sampling output of previous layers. By this way, we separated the features, low resolution layers take responsibility for low frequency information, such as body shape. While the higher layers are in charge of details of the image.

Our GAN is based on the StyleGAN [2], which is proposed by Nvidia. We slightly change the model, such as the number of fully connected layers.

According to [2], StyleGAN tries to normalize the latent vector first, then uses 8 fully connected layers to project the latent vector. While the GANs before StyleGAN usually insert the latent vector to the generator. That provides a more accurate way to control the generated image.

After affining the latent vector, the output will be injected to each layer. a layer contains an up-sampling module, and a 3*3 convolution layer. The affined latent vector and noise will be added twice to the image and two AdaIN modules will normalize the image. Here, we have a basic constant layer as the input of the first layer since the first layer cannot get the input from the previous layer.

The noise of StyleGAN also needs some process. The noise will be input to a per-channel scale module and be injected to Synthesis Network together with the affined latent vector. Figure 5 shows the structure of StyleGAN.

The discriminator contains two modules, and its number of layers are kept the same as the generator. First module is 3*3 convolution layer with bias and leaky ReLU. Second module contains a filter to blur image and a 3*3 down sampling layer with bias and leaky ReLU.

5. Experiments

In our project, we try to feed the network with different datasets and tuning different hyperparameters to see the difference in time, iterations, and output results. The base model of our comparison is the StyleGAN model proposed by T. Karras, et. al. [5]. In their training[5], they were using FFHQ dataset, with 900 max iterations, ending at 1024*1024 resolution. And they used 2 days 14 hours with V100*4, though we cannot use their time as the base line. Their sample results are as shown in Figure 6.

Figure 6: The output sample from our reference paper [5].

[5] https://github.com/taki0112/StyleGAN-Tensorflow

At first, we used the Azur Lane dataset. As introduced previously, the first dataset we used in the first place is not curated properly. We set the hyperparameters to be 1045 max interactions (because there are 1045 pictures in the dataset), from 8 to 256 resolution. We chose the final resolution to be 256 because it won't consume too much time in tuning yet also provide a clear visual for human eyes. However, because we ran it on my PC, which is 1060 6G VRAM with 16G RAM, it was out of memory at the start of 32 resolution. The results were shown in milestones already, so we won't put it here. Its training time was around 5 hours.

In the second run and third run, we were still using the bad Azur Lane dataset, running on school GPU server with 2080 Ti*4 from now on. In the second run, we ran it from 8 to 256 resolution, with 1045 max iterations. Its training time is similar to the base line, nearly 3 days, and it generated 236,250 samples under 256 resolution. Two chosen samples are shown in Figure 7 at top. In the third run, we tried to turn off the progressive generation. So, the resolution became from 256 to 256. The max iteration was still 1045. This time, the network tended to generate a tremendous amount of 9.19 million samples. And by estimation, that would consume more than 20 days. So, we had to stop the training process at 1.2 million, which already took us 3 days. Another two chosen samples are shown in Figure 7 at bottom.

In these samples, we can see that due to the problem of the dataset, the jagged outline was perfectly learned by the network, and it is shown in nearly all the samples except some collapse points. Besides, the body shape and the color tone were both learned obviously. However, all the details such as face, and hands were not learned at all. These details are either too blurry or presented as melted colors. What's more, there is not much difference in using or not using progressive generation, by comparing the results. We decided to use the progressive generation all the time in later runs for saving time.

Figure 7: second run at top, from 8 to 256, and third run at bottom, from 256 to 256

In the fourth run and fifth run, we corrected our dataset as introduced in the Data section. We included Girl's Frontline as well Azur Lane in our dataset because they are clean and high resolution. In the fourth run, we used the dataset of all the curated faces from Azur Lane and Girl's Frontline. This run was used to compare with the results of the baseline model, because they were also using a dataset of faces like ours. We ran it from 16 to 256 resolution, with 1044 max iterations (1044 faces after careful selection). Because we skipped the resolution of 8, it generated 352,500 samples, 116,250 more samples than the second run, under 256 resolution, using also nearly 3 days. In the fifth run, we used the dataset of the full body of Azur Lane and Girl's Frontline, with a total of 1392 samples and max iterations. We also ran it from 16 to 256 resolution. By estimation, it will use a little more than 3 days, but we stop it at the same time as the fourth run finished.

As shown in Figure 8, the top two samples are chosen from the fourth run and the bottom two are from the fifth run. A big difference can be seen from these two results. The faces have much clearer contour and details than the full body results. Therefore, we doubt that those melted color phenomena are due to the lack of depth in the network. Besides, some similar features are still learned in these two runs as the second and the third run. What's more, hairs and eyes look reasonable and decent in the fourth run.

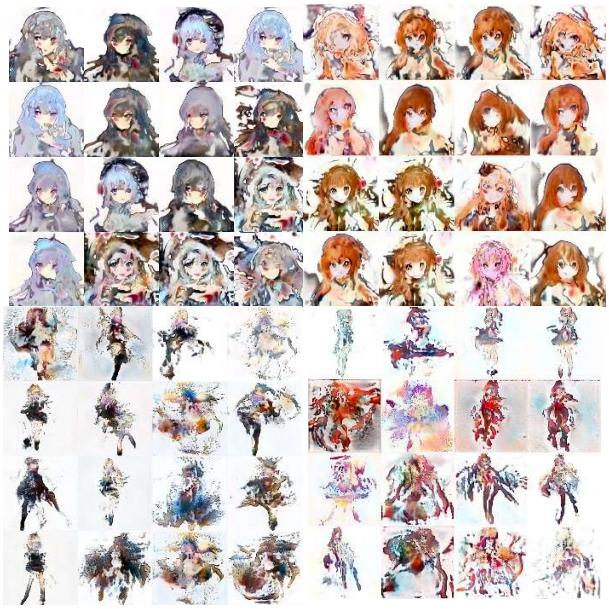


Figure 8: fourth run samples at top, from 16 to 256, and fifth run samples at bottom, from 16to 256.

A summary of all the hyperparameters and results are summarized in Table 1.

Models	Base model	Second run	Third run	Fourth run	Fifth run
Max iteration	900	1045	1045	1044	1392
Dataset	FFHQ	Full body (1045)	Full body (1045)	Faces (1044)	Full body (1392)
Time	2 days 14 hours	≈ 3 days	> 20 days	≈ 3 days	> 3 days
Samples	N/A	236 K	9.19 M	352.5 K	470 K
Results	Perfect	Rough feature	Rough feature	Major feature	Rough feature

Table 1: A summary of all the hyperparameters and results

A problem that we are not sure is that the results from the fourth run at 128 resolution looks even better than the result at 256 resolution. The melted color phenomena are less severe than higher resolution. This deepens our doubt on the lack of depth of the network. Two samples are chosen from the fourth run at 128 resolution, as shown in Figure 9.

Figure 9: samples from the fourth run at 128 resolution.

What's more, we also generated uncurated samples in Figure 10, style mixing in Figure 11, and truncation trick in Figure 12, in comparison with the baseline model and tried to look inside how our model performance is. As shown below, our model indeed accomplishes the style mixing and truncation trick, and this proves that the network indeed learns something but just not enough.

Figure 10: Uncurated samples from the result of the fourth run.

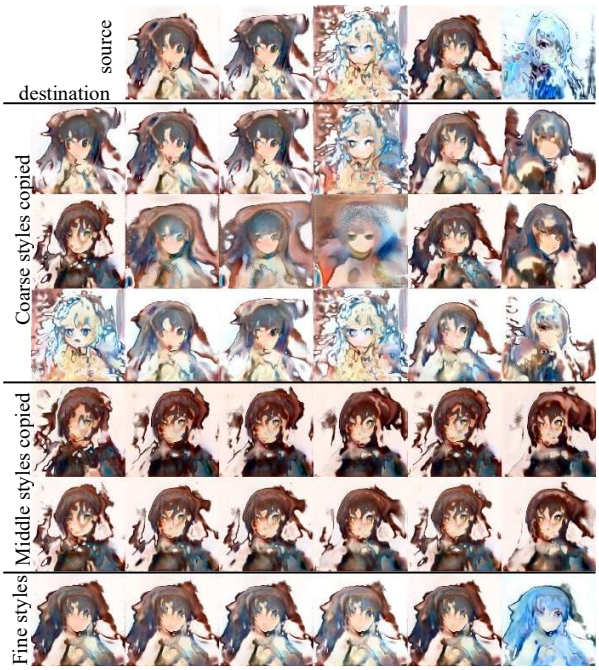


Figure 11: Style mixing from the result of the fourth run.

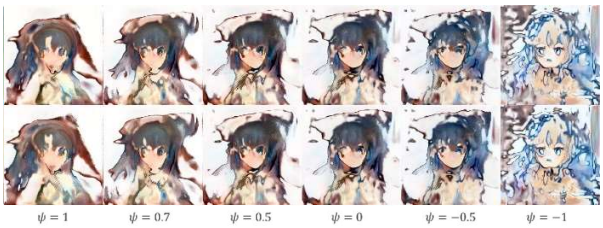


Figure 12: Truncation trick from the result of the fourth run.

6. Conclusion

We explore the automatic creation of the full-body anime characters in this work. By combining clean dataset and several practicable GAN training strategies, we build a model which can generate well recognizable full-body images of anime characters.

There still remain some issues for us for further investigations. One direction is how to improve the GAN model to solve the melted color phenomena on high resolution. Also, quantitative evaluating methods under this scenario should be analyzed.

Another direction is how to improve the final resolution of generated images. Super-resolution seems to be a possible strategy, but the model needs to be more carefully designed and tested.

Reference

[1] I. J. Goodfellow et al., “Generative Adversarial Networks,” arXiv [stat.ML], 2014.

[2] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, “Analyzing and improving the image quality of StyleGAN,” arXiv [cs.CV], 2019.

[3] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive growing of GANs for improved quality, stability, and variation,” arXiv [cs.NE], 2017.

[4] H. Zhang et al., “StackGAN++: Realistic image synthesis with Stacked Generative Adversarial Networks,” arXiv [cs.CV], 2017.

[5] T. Karras, S. Laine, and T. Aila, “A style-based generator architecture for generative adversarial networks,” arXiv [cs.NE], 2018.

[6] Y. Jin, J. Zhang, M. Li, Y. Tian, H. Zhu, and Z. Fang, “Towards the automatic Anime characters creation with generative adversarial networks,” arXiv [cs.CV], 2017.

[7] K. Hamada, K. Tachibana, T. Li, H. Honda, and Y. Uchida, “Full-body high-resolution Anime generation with Progressive Structure-conditional Generative Adversarial Networks,” arXiv [cs.CV], 2018.

[8] “Manga109,” Manga109.org. [Online]. Available: http://www.manga109.org/en/. [Accessed: 08-Nov-2020].

[9] “- nico-opendata,” Nico-opendata.jp. [Online]. Available: https://nico-opendata.jp/en/seigadata/index.html. [Accessed: 08-Nov-2020].

[10] Anonymous, The Danbooru Community, and &. G. Branwen, “Danbooru2019.” 13-Jan-2020.