IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Facial Image Generation of Anime characters

[1]Yogita T Hambir, [2]Pranav Bhaskar, [3]Rohit Kumar, [4]Sourabh Kumar, [5]Tirth Patel

Dept. of Computer Engineering
Army Institute of Technology, Pune, India

Abstract: Back in 2014, Ian Goodfellow dreamt up the idea of Generative Adversarial Net-works (GANs), since then a lot of work has been done in the field of Automatic facial image generation. There are already some papers regarding the anime character generation using different GAN algorithms with good results but not so efficient. In this work, we explored the StyleGAN2 for training our clean and well-suited data-set and obtained an efficient and promising result.For public convenience, we built a website that will allow users to obtain required anime character by giving few characteristics.

Index Terms **– Deep Learning, Generative Model, Anime**

I. INTRODUCTION

The automated generation of cartoon/anime characters allows for the development of custom characters without the need for technical skills. It will easily provide cartoon designers or anime character designers with their custom design. It will save a lot of time. A clean data set from the anime characters database and STYLEGAN2 model is used in order to obtain the promising result. Generative Adversarial Networks (GANs) offers a very powerful unsupervised learning. It consists of a system of two competing neural networks and is able to analyse, capture and copy variation within a data set. Some applications of anime character generator includes manga, magazines, anime series etc.

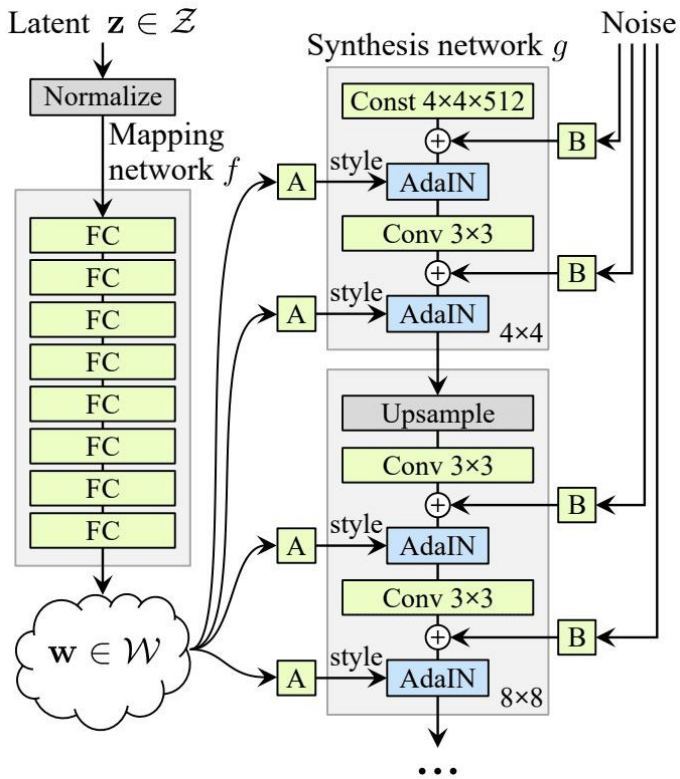


Figure 1. StyleGAN Generator Model Architecture
Taken from: A Style-Based Generator Architecture for Generative Adversarial Networks.

II. LITERATURE SURVEY

GAN since their introduction in 2014 have shown outstanding results in various fields from image generation to feature transfer. It can be broken down into three parts :

Generative : To build a generative model, which is nothing but a probabilistic model that explains how data is produced.

Adversarial : The teaching of a model takes place in a competitive setting.

Networks : Deep neural networks are used for training purposes.

We used STYLEGAN2 which came into picture in 2018 when a STYLEGAN paper got published by NVIDIA. This proposed an architecture that allows them to control various aspects in the obtained samples. It also follows the Progressive GAN concept, which entails training the networks at smaller resolution at initial, and then larger layers are progressively introduced after the network has stabilized. The preparation time is cut in half and the training is more consistent as a result of this. StyleGAN enhances the results by including a mapping network that encodes the input vectors into an intermediate latent space, and then uses different aspects to monitor the various levels of information.

III. PROPOSED SYSTEM

The amount of effort invested in the pre-processing of data sets in the previous systems was less. The images were often face cropped. Face cropping removed other important features from the image too as can be seen in the image. Similarly, including a feature vector for the generation of image can save a lot of effort in guessing latent space afterwards when expecting a specific kind of face.

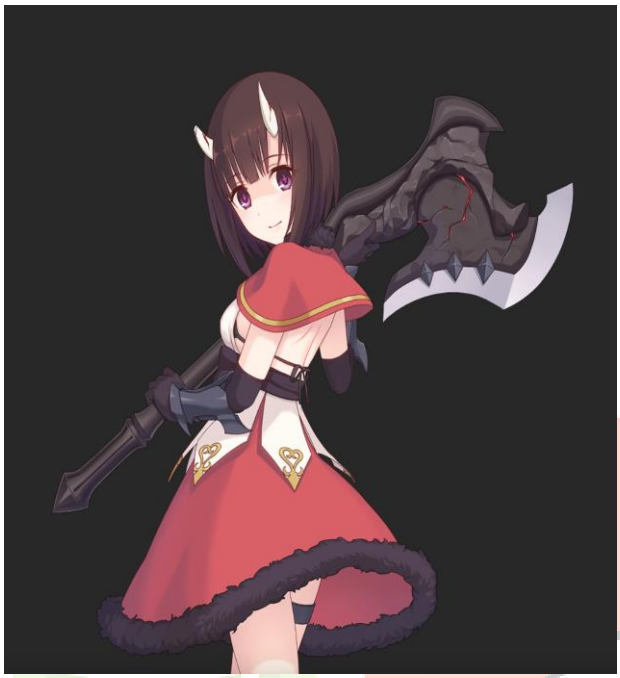
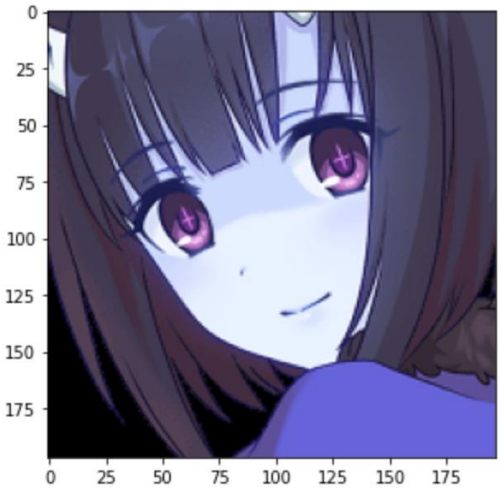
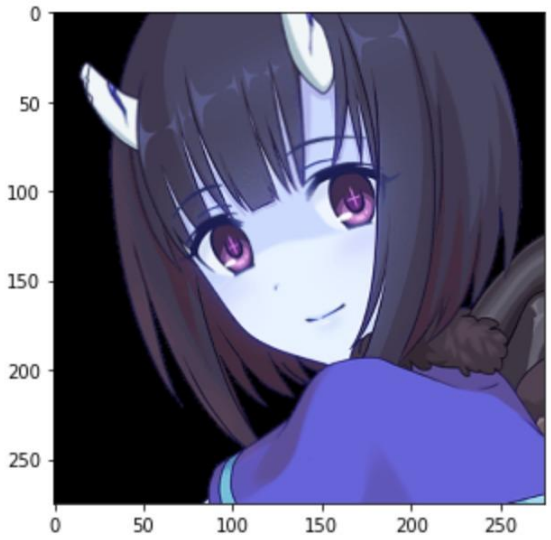


Figure 2. Image from dataset




```
[(0.945797860622406, '1girl'),
 (0.9155818819999695, 'purple eyes'),
 (0.8582026362419128, 'solo'),
 (0.5727720856666565, 'face'),
 (0.5217634439468384, 'short hair'),
 (0.4180621802806854, 'smile'),
 (0.3357243537902832, 'brown hair'),
 (0.3179219365119934, 'black hair'),
 (0.30333322286605835, 'blush'),
 (0.280547559261322, 'looking at viewer'),
 (0.24123570322990417, 'long hair'),
 (0.13788694143295288, 'scarf'),
 (0.1253368854522705, 'bust'),
 (0.1157379150390625, 'lips'),
 (0.10436880588531494, 'ribbon'),
 (0.10212647914886475, 'simple background')]
```

Figure 3. Cropped image without margin (left) and generated tags for image without margin (right)



```
[(0.9683310985565186, '1girl'),
 (0.8615649342536926, 'solo'),
 (0.8422595262527466, 'purple eyes'),
 (0.777560293674469, 'animal ears'),
 (0.6165587306022644, 'smile'),
 (0.6097004413604736, 'short hair'),
 (0.47649165987968445, 'brown hair'),
 (0.3829805552959442, 'looking at viewer'),
 (0.3799143135547638, 'black hair'),
 (0.3471345901489258, 'face'),
 (0.3389146029949188, 'blush'),
 (0.3025951385498047, 'cat ears'),
 (0.15116798877716064, 'bust'),
 (0.14828965067863464, 'simple background'),
 (0.12834909558296204, 'long hair')]
```

Figure 4. Cropped image with margin (left) and generated tags for image with margin (right)

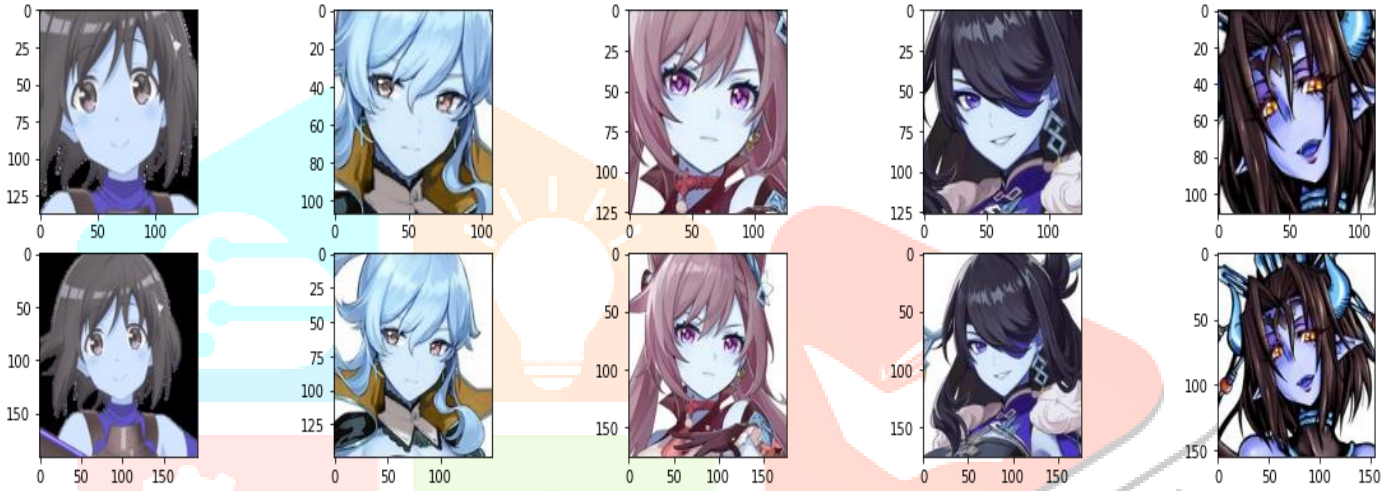


Figure 5. Images cropped without margin (top) miss few features such as hair ornaments which are visible in images with margin (bottom)

IV. Training Requirements

A. Data

The minimum size for a data-set depends on the domain's complexity and whether transfer learning is used or not. The default settings of StyleGAN yield a 1024 px generator with 27.2M parameters, which is a broad model and can theoretically consume a large amount of images (millions), so there is not too much like that. In reality, a minimum of 5000 appears to be required for training and generating decent quality anime faces from scratch. For learning a particular character while using the anime face style GAN.

B. Compute

GPUs with 11 GB VRAM are required to suit appropriate mini-batch sizes. That only trains n = 4 at 512px, and moving below which it will be even slower. So, 1080ti Nvidia and upper versions would be fine. (Open-CL/AMD is reportedly working on running StyleGAN models, and one active training report is available.)

C. Data preparation

The most challenging aspect of running StyleGAN is to accurately plan the data-set. Unlike other GAN implementations like we do with PyTorch, StyleGAN does not support reading a file path as an input; it can only read its specific .tfrecord format that stores each image at each acceptable decision as raw arrays. Therefore, the input files should be uniformly converted by the special data-set tool.py to the .tfrecord format, and will take up 19 more discs.

V. ALGORITHMIC EXPOSURE

A. Generator architecture

G	Params	OutputShape	WeightShape
latents_in	-	(?, 512)	-
labels_in	-	(?, 59)	-
lod	-	()	-
dlatent_avg	-	(512,)	-
G_mapping/latents_in	-	(?, 512)	-
G_mapping/labels_in	-	(?, 59)	-
G_mapping/LabelConcat	30208	(?, 1024)	(59, 512)
G_mapping/Normalize	-	(?, 1024)	-
G_mapping/Dense0	524800	(?, 512)	(1024, 512)
G_mapping/Dense1	262656	(?, 512)	(512, 512)
G_mapping/Dense2	262656	(?, 512)	(512, 512)
G_mapping/Dense3	262656	(?, 512)	(512, 512)
G_mapping/Dense4	262656	(?, 512)	(512, 512)
G_mapping/Dense5	262656	(?, 512)	(512, 512)
G_mapping/Dense6	262656	(?, 512)	(512, 512)
G_mapping/Dense7	262656	(?, 512)	(512, 512)
G_mapping/Broadcast	-	(?, 14, 512)	-
G_mapping/dlatents_out	-	(?, 14, 512)	-
Truncation/Lerp	-	(?, 14, 512)	-
G_synthesis/dlatents_in	-	(?, 14, 512)	-
G_synthesis/4x4/Const	8192	(?, 512, 4, 4)	(1, 512, 4, 4)
G_synthesis/4x4/Conv	2622465	(?, 512, 4, 4)	(3, 3, 512, 512)
G_synthesis/4x4/ToRGB	264195	(?, 3, 4, 4)	(1, 1, 512, 3)
G_synthesis/8x8/Conv0_up	2622465	(?, 512, 8, 8)	(3, 3, 512, 512)
G_synthesis/8x8/Conv1	2622465	(?, 512, 8, 8)	(3, 3, 512, 512)
G_synthesis/8x8/Upsample	-	(?, 3, 8, 8)	-
G_synthesis/8x8/ToRGB	264195	(?, 3, 8, 8)	(1, 1, 512, 3)
G_synthesis/16x16/Conv0_up	2622465	(?, 512, 16, 16)	(3, 3, 512, 512)
G_synthesis/16x16/Conv1	2622465	(?, 512, 16, 16)	(3, 3, 512, 512)
G_synthesis/16x16/Upsample	-	(?, 3, 16, 16)	-
G_synthesis/16x16/ToRGB	264195	(?, 3, 16, 16)	(1, 1, 512, 3)
G_synthesis/32x32/Conv0_up	2622465	(?, 512, 32, 32)	(3, 3, 512, 512)
G_synthesis/32x32/Conv1	2622465	(?, 512, 32, 32)	(3, 3, 512, 512)
G_synthesis/32x32/Upsample	-	(?, 3, 32, 32)	-
G_synthesis/32x32/ToRGB	264195	(?, 3, 32, 32)	(1, 1, 512, 3)
G_synthesis/64x64/Conv0_up	1442561	(?, 256, 64, 64)	(3, 3, 512, 256)
G_synthesis/64x64/Conv1	721409	(?, 256, 64, 64)	(3, 3, 256, 256)
G_synthesis/64x64/Upsample	-	(?, 3, 64, 64)	-
G_synthesis/64x64/ToRGB	132099	(?, 3, 64, 64)	(1, 1, 256, 3)
G_synthesis/128x128/Conv0_up	426369	(?, 128, 128, 128)	(3, 3, 256, 128)
G_synthesis/128x128/Conv1	213249	(?, 128, 128, 128)	(3, 3, 128, 128)
G_synthesis/128x128/Upsample	-	(?, 3, 128, 128)	-
G_synthesis/128x128/ToRGB	66051	(?, 3, 128, 128)	(1, 1, 128, 3)
G_synthesis/256x256/Conv0_up	139457	(?, 64, 256, 256)	(3, 3, 128, 64)
G_synthesis/256x256/Conv1	69761	(?, 64, 256, 256)	(3, 3, 64, 64)
G_synthesis/256x256/Upsample	-	(?, 3, 256, 256)	-
G_synthesis/256x256/ToRGB	33027	(?, 3, 256, 256)	(1, 1, 64, 3)
G_synthesis/images_out	-	(?, 3, 256, 256)	-
G_synthesis/noise0	-	(1, 1, 4, 4)	-
G_synthesis/noise1	-	(1, 1, 8, 8)	-
G_synthesis/noise2	-	(1, 1, 8, 8)	-
G_synthesis/noise3	-	(1, 1, 16, 16)	-
G_synthesis/noise4	-	(1, 1, 16, 16)	-
G_synthesis/noise5	-	(1, 1, 32, 32)	-
G_synthesis/noise6	-	(1, 1, 32, 32)	-
G_synthesis/noise7	-	(1, 1, 64, 64)	-
G_synthesis/noise8	-	(1, 1, 64, 64)	-
G_synthesis/noise9	-	(1, 1, 128, 128)	-
G_synthesis/noise10	-	(1, 1, 128, 128)	-
G_synthesis/noise11	-	(1, 1, 256, 256)	-
G_synthesis/noise12	-	(1, 1, 256, 256)	-
images_out	-	(?, 3, 256, 256)	-
---	---	---	---
Total	25059810		

B. Discriminator architecture

D	Params	OutputShape	WeightShape
images_in	-	(?, 3, 256, 256)	-
labels_in	-	(?, 59)	-
256x256/FromRGB	256	(?, 64, 256, 256)	(1, 1, 3, 64)
256x256/Conv0	36928	(?, 64, 256, 256)	(3, 3, 64, 64)
256x256/Conv1_down	73856	(?, 128, 128, 128)	(3, 3, 64, 128)
256x256/Skip	8192	(?, 128, 128, 128)	(1, 1, 64, 128)
128x128/Conv0	147584	(?, 128, 128, 128)	(3, 3, 128, 128)
128x128/Conv1_down	295168	(?, 256, 64, 64)	(3, 3, 128, 256)
128x128/Skip	32768	(?, 256, 64, 64)	(1, 1, 128, 256)
64x64/Conv0	590080	(?, 256, 64, 64)	(3, 3, 256, 256)
64x64/Conv1_down	1180160	(?, 512, 32, 32)	(3, 3, 256, 512)
64x64/Skip	131072	(?, 512, 32, 32)	(1, 1, 256, 512)
32x32/Conv0	2359808	(?, 512, 32, 32)	(3, 3, 512, 512)
32x32/Conv1_down	2359808	(?, 512, 16, 16)	(3, 3, 512, 512)
32x32/Skip	262144	(?, 512, 16, 16)	(1, 1, 512, 512)
16x16/Conv0	2359808	(?, 512, 16, 16)	(3, 3, 512, 512)
16x16/Conv1_down	2359808	(?, 512, 8, 8)	(3, 3, 512, 512)
16x16/Skip	262144	(?, 512, 8, 8)	(1, 1, 512, 512)
8x8/Conv0	2359808	(?, 512, 8, 8)	(3, 3, 512, 512)
8x8/Conv1_down	2359808	(?, 512, 4, 4)	(3, 3, 512, 512)
8x8/Skip	262144	(?, 512, 4, 4)	(1, 1, 512, 512)
4x4/MinibatchStddev	-	(?, 513, 4, 4)	-
4x4/Conv	2364416	(?, 512, 4, 4)	(3, 3, 513, 512)
4x4/Dense0	4194816	(?, 512)	(8192, 512)
Output	30267	(?, 1)	(512, 59)
scores_out	-	(?, 1)	-
---	---	---	---
Total	24030843		

VI. RESULT

After passing the images in the dataset through preprocessing and extracting their feature vectors, these are the selected (59/500) features which can be used as labels and be appended to the input vector(randomly generated).

impLabels = [’pink hair’, ’blush’, ’open mouth’, ’hair ornament’, ’short hair’, ’looking at viewer’, ’flower’, ’aqua eyes’, ’long hair’, ’blue eyes’, ’collarbone’, ’school uniform’, ’green eyes’, ’braid’, ’hair clip’, ’red eyes’, ’blue hair’, ’ribbon’, ’bow’, ’hair ribbon’, ’pink eyes’, ’silver hair’, ’white hair’, ’red hair’, ’twin tails’, ’black hair’, ’tears’, ’hair flower’, ’smile’, ’blonde hair’, ’choker’, ’bare shoulders’, ’gloves’, ’hair bow’, ’brown hair’, ’brown eyes’, ’yellow eyes’, ’orange hair’, ’animal ears’, ’jewelry’, ’cleavage’, ’bangs’, ’pointy ears’, ’earrings’, ’purple eyes’, ’purple hair’, ’hairband’, ’green hair’, ’hat’, ’cat ears’, ’:d’, ’glasses’, ’fang’, ’lips’, ’necktie’, ’black eyes’, ’ponytail’, ’tongue’, ’blunt bangs’]

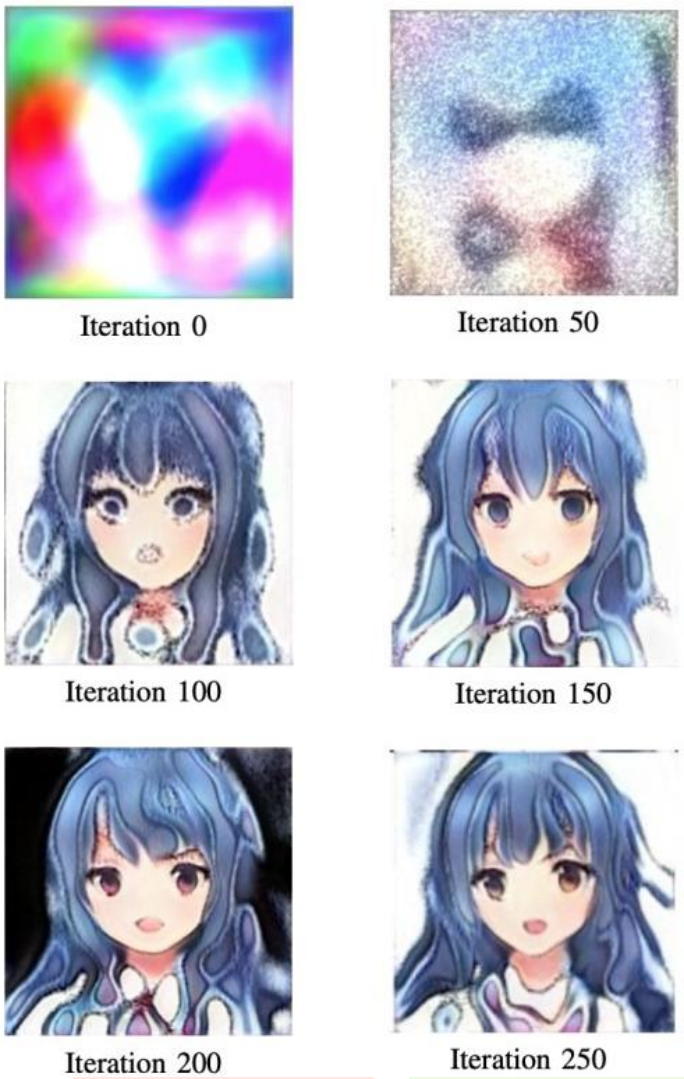


Figure 8. Results of image generated in different iterations

VII. Conclusion

We have proposed a simple system which will help consumers to generate their customized anime characters for their respective applications.

References

[1] Ian Goodfellow, Generative adversarial nets, 2014.
[2] Karras, Tero and Timo Aila, A style-based generator architecture for generative adversarial networks, 2019.
[3] Phillip, Image-to-image translation with conditional adversarial networks, 2017.
[4] Zhu, Jun-Yan, Unpaired image-to-image translation using cycle-consistent adversarial networks, 2017.
[5] Salimans Tim, Improved techniques for training gans, 2016.
[6] Jin Yanghua, Towards the automatic anime characters creation with generative adversarial networks, 2017.
[7] Arjovsky, Martin, Soumith Chintala, and Léon Bottou, Wasserstein gan, 2017.
[8] Miyato Takeru, Spectral normalization for generative adversarial networks, 2018.
[9] Zhang Han, Self-attention generative adversarial networks, 2018.
[10] Berthelot, David, Thomas Schumm, and Luke Metz, Began: Boundary equilibrium generative adversarial networks, 2017.
[11] Lucic Mario, Are gans created equal? a large-scale study, 2018.
[12] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. "The unreasonable effectiveness of deep features as a perceptual metric", 2018.
[13] Martin Arjovsky and Léon Bottou, "Towards principled methods for training generative adversarial networks", 2017.
[14] tdrussell, "https://github.com/tdrussell/IllustrationGAN", 2016.
[15] Mattya, chainer-dcgan, "https://github. com/mattya/chainer-DCGAN", 2015.
[16] Hiroshiba, Girl friend factory, http://qiita.com/Hiroshiba/items/ d5749d8896613e6f0b48,2016
[17] Jie Lei, "https://github.com/ jayleicn/animeGAN", 2017.
[18] Mattya, chainer-gan-lib, "https://github.com/pfnet-research/chainer-gan-lib", 2017.
[19] Yanghua Jin, Jiakai Zhang, "Towards the High-quality Anime Characters Generation with Generative Adversarial Networks", 2017.