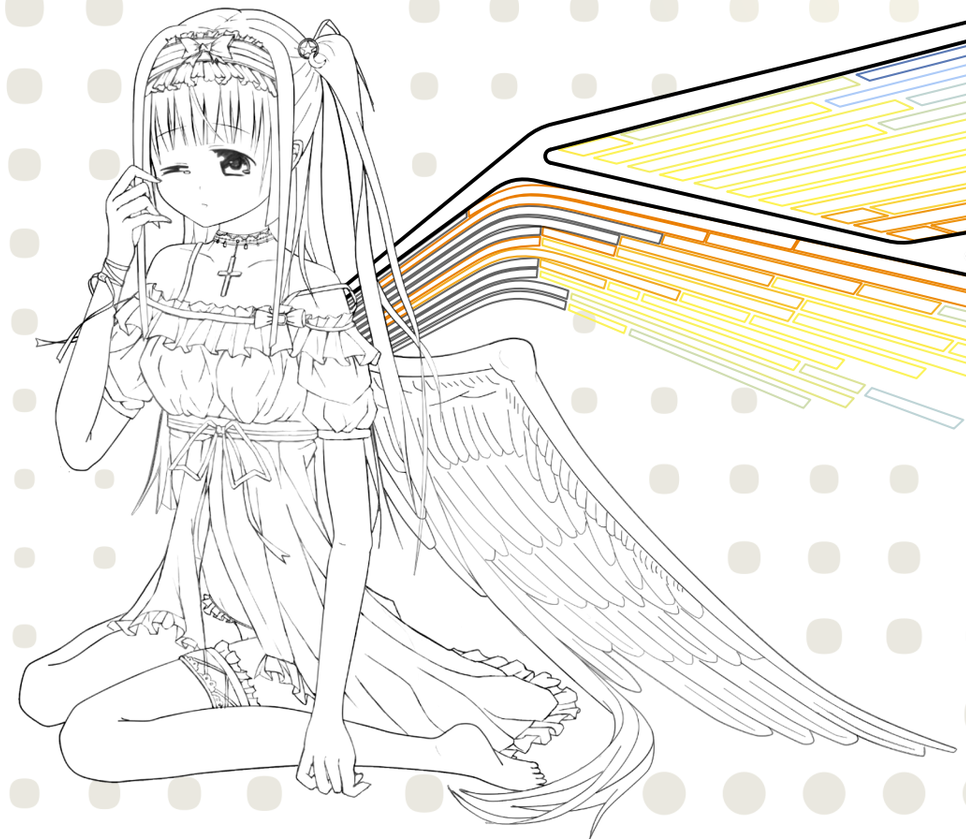
Create Anime Characters With A.I.



2017夏コミ 三日目東ウ05a

Girls Manifold

Create Anime Characters with A.I. !

Yanghua Jin
School of Computer Science
Fudan University
yanghuajin94@gmail.com

Jiakai Zhang
School of Computer Science
Carnegie Mellon University
zhangjk95@gmail.com

Minjun Li
School of Computer Science
Fudan Univerisity
li.akizora@gmail.com

Yingtao Tian
Department of Computer Science
Stony Brook University
alan.yt.tian@gmail.com

Huachun Zhu
School of Mathematics
Fudan Univerisity
tim.zhc@gmail.com

Zhihao Fang
Department of Architecture
Tongji Univerisity
fangzhihao126@gmail.com

Abstract

Automatic generation of facial images has been well studied after the Generative Adversarial Network(GAN) came out. There exists some attempts applying the GAN model to the problem of generating facial images of anime characters, but none of the existing work gives a promising result. In this work, we explore the training of GAN models specialized on an anime facial image dataset. We address the issue from both the data and the model aspect, by collecting a more clean, well-suited dataset and leverage proper, empirical application of DRAGAN. With quantitative analysis and case studies we demonstrate that our efforts lead to a stable and high-quality model. Moreover, to assist people with anime character design, we build a website[1] with our pre-trained model available online, which makes the model easily accessible to general public.

1 Introduction

We all love anime characters and are tempted to create our custom ones. However, it takes tremendous efforts to master the skill of drawing, after which we are first capable of designing our own characters. To bridge this gap, the automatic generation of anime characters offers an opportunity to bring a custom character into existence without professional skill. Besides the benefits for a non-specialist, a professional creator may take advantages of the automatic generation for inspiration on animation and game character design; a Doujin RPG developer may employ copyright-free facial images to reduce designing costs in game production.

Existing literature provides several attempts for generation facial images of anime characters. Among them are Mattya[16] and Rezoolab[24] who first explored the generation of anime character faces right after the appearance of DCGAN[22]. Later, Hirobashi[8] proposed the conditional generation model for anime character faces. Also, codes are made available online focusing on anime faces generation such as IllustrationGAN[28] and AnimeGAN[13].

[1] http://make.girls.moe

コミックマーケット 92 (Comiket 92) (Summer 2017), 東京ビッグサイト.

However, since results from these works are blurred and distorted on an untrivial frequency, it still remains a challenge to generate industry-standard facial images for anime characters.

In this report, we propose a model that produces anime faces at high quality with promising rate of success. Our contribution can be described as three-fold: A clean dataset, which we collected from Getchu, a suitable GAN model, based on DRAGAN, and our approach to train a GAN from images without tags, which can be leveraged as a general approach to training supervised or conditional model without tag data.

2 Related Works

Generative Adversarial Network (GAN) [5], proposed by Goodfellow et al., shows impressive results in image generation [22], image transfer[9], super-resolution[12] and many other generation tasks. The essence of GAN can be summarized 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.

Although the training process is quiet simple, optimizing such models often lead to *mode collapse*, in which the generator will always produce the same image. To train GANs stably, Metz et al. [18] suggests rendering Discriminator omniscient whenever necessary. By learning a loss function to separate generated samples from their real examples, LS-GAN[21] focuses on improving poor generation result and thus avoids mode collapse. More detailed discussion on the difficulty in training GAN will be in Section 4.2.

Many variants of GAN have been proposed for generating images. Radford et al. [22] applied convolutional neural network in GAN to generate images from latent vector inputs. Instead of generating images from latent vectors, serval methods use the same adversarial idea for generating images with more meaningful input. Mirza & Osindero et al. introduced Conditional Generative Adversarial Nets [19] using the image class label as a conditional input to generate MNIST numbers in particular class. Reed et al. [23] further employed encoded text as input to produce images that match the text description. Instead of only feeding conditional information as the input, Odena et al. proposed ACGAN[20], which also train the discriminator as an auxiliary classifier to predict the condition input.

3 Image Data Preparation

It is well understood that image dataset in high quality is essential, if not most important, to the success of image generation. Web services hosting images such as Danbooru[2] and Safebooru[3], commonly known as image boards, provide access to a large number of images enough for training image generation models. Previous works mentioned above all base their approaches on images crawled from one of these image boards, but their datasets suffer from high inter-image variance and noise. We hypothesize that it is due to the fact that image boards allow uploading of images highly different in style, domain, and quality, and believe that it is responsible for a non-trivial portion of quality gaps between the generation of real people faces and anime character faces. In order to bridge such a gap, we propose to use a more consisting, clean, high-quality dataset, and in this section we introduce our method of building such a dataset.

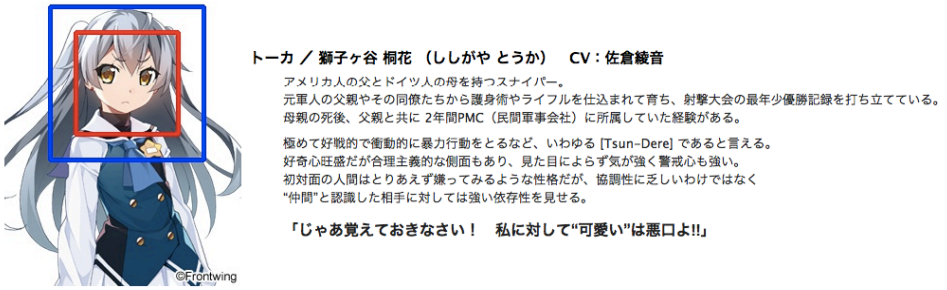


Figure 1: Sample Getchu page and the detection result,`http://www.getchu.com/soft.phtml?id=933144`. Red line indicate the original bounding box and blue line indicate the scaled bounding box. Copyright: Frontwing, 2017

3.1 Image Collection

Getchu [4] is a website providing information and selling of Japanese games, for which there are character introduction sections with standing pictures (立ち絵). Figure 1 shows one sample character introduction from the site. These images are diverse enough since they are created by illustrators with different styles for games in a diverse sets of theme, yet consisting since they are all belonging to domain of character images, are in descent quality, and are properly clipped/aligned due to the nature of illustration purpose. Because of these properties, they are suitable for our task.

Our collection of images consists of the following steps. First we execute a SQL query on ErogameScape's Web SQL API page[5] to get the Getchu page link for each game. The SQL query we used is described in Appendix 8.1. Then we download images and apply lbpcascade animeface [6], an anime character face detector, to each image and get bounding box for faces. We observe that the default estimated bounding box is too close to the face to capture complete character information that includes hair length and hair style, so we zoom out the bounding box by a rate of 1.5x. The difference is shown in Figure 1. Finally, from 42000 face images in total from the face detector, we manually check all anime face images and remove about 4% false positive and undesired images.

3.2 Tag Estimation

The task of generating images with customization requires categorical metadata of images as priors. Images crawled from image boards are accompanied by user created tags which can serve as such priors, as shown in previous works. However, Getchu does not provide such metadata about their images, so to overcome this limitation, we propose to use a pre-trained model for (noisy) estimations of tags.

We use Illustration2Vec[26], a CNN-based tool for estimating tags of anime illustrations[7] for our purpose. Given an anime image, this network can predict probabilities of belonging to 512 kinds of general attributes (tags) such as "smile" and "weapon", among which we select 34 related tags suitable for our task. We show the selected tags and the number of dataset images corresponded to each estimated tag in Table 1. For set of tags with mutual exclusivity (e.g. hair color, eye color), we choose the one with maximum probability from the network as the estimated tag. For orthogonal tags (e.g. "smile", "open mouth", "blush"), we use 0.25 as the threshold and estimate each attribute's presence independently.

[2] `danbooru.donmai.us`
[3] `safebooru.org`
[4] `www.getchu.com`
[5] `http://erogamescape.dyndns.org/~ap2/ero/toukei_kaiseki/sql_for_erogamer_form.php`
[6] `https://github.com/nagadomi/lbpcascade_animeface`
[7] Pre-trained model available on `http://illustration2vec.net/`

blonde hair	brown hair	black hair	blue hair	pink hair	purple hair	green hair
4991	6659	4842	3289	2486	2972	1115
red hair	silver hair	white hair	orange hair	aqua hair	gray hair	long hair
2417	987	573	699	168	57	16562
short hair	twintails	drill hair	ponytail	blush	smile	open mouth
1403	5360	1683	8861	4926	5583	4192
hat	ribbon	glasses	blue eyes	red eyes	brown eyes	green eyes
1403	5360	1683	8861	4926	5583	4192
purple eyes	yellow eyes	pink eyes	aqua eyes	black eyes	orange eyes	
4442	1700	319	193	990	49	

Table 1: Number of dataset images for each tag

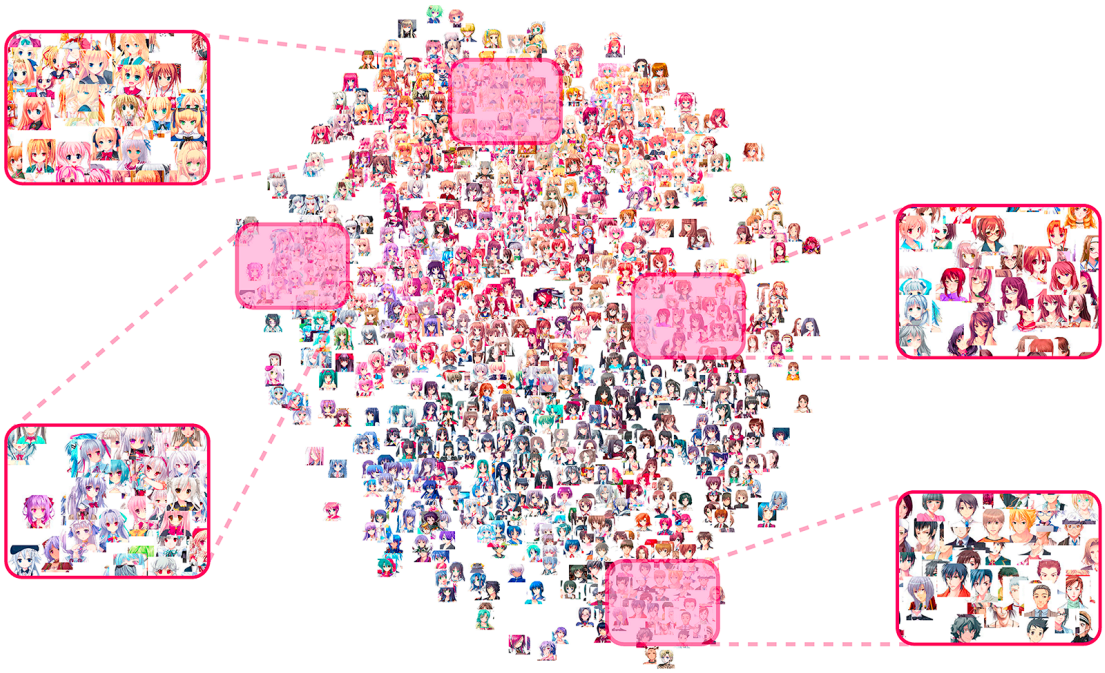


Figure 2: t-SNE visualization of 1500 dataset images

3.3 Visualization

We would like to show the image preparation and the performance of tag estimation through visualization. As an approximation, we apply the Illustration2Vec feature extractor, which largely shares architecture and weights with Illustration2Vec tag estimator, on each image for a 4096-dimension feature vector, and project feature vectors onto a 2D space using t-SNE[15]. Figure 2 shows the t-SNE result of 1500 images sampled from the dataset. We observe that character images with similar visual attributes are placed closely. Due to the shared weights, we believe this also indicates the good performance in tag estimator.

4 Generative Adversarial Network

4.1 Vanilla GAN

Generative Adversarial Networks proposed by Goodfellow et at.[5] are implicit generative models. It proves to be an effective and efficient way to generate highly photo-realistic images in an unsupervised and likelihood-free manner[22]. GAN uses a generator network G to generate samples from P_G. This is done by transforming a latent noise variable $z \sim P_{noise}$ into a sample $G(z)$. The original GAN uses a min-max game strategy to train the generator G, imposing another network D to distinguish samples from G and real samples. Formally, the objective of GAN can be expressed as

$$\min_G \max_D \mathcal{L}(D, G) = \mathbb{E}_{x \sim P_{data}}[\log D(x)] + \mathbb{E}_{x \sim P_{noise}}[\log(1 - D(G(z)))] \,.$$

In this formula, the discriminator D try to maximize the output confidence score from real samples. Meanwhile, it also minimizes the output confidence score from fake samples generated by G. On contrast, the aim of G is to maximize the D evaluated score for its outputs, which can be viewed as deceiving D.

4.2 Improved training of GAN

Despite the impressive results of GAN, it is notoriously hard to train properly GAN. [1] showed the P_G and P_{data} may have non-overlap supports, so the Jensen-Shannon Divergence in the original GAN objective is constantly 0, which leads to instability. [3] argued that there may exist no equilibrium in the game between generator and discriminator. One possible remedy is to use integral probability metric(IPM) based methods instead, e.g. Wasserstein distance[2], Kernel MMD[14], Cramer distance[4]. Some recent GAN variants suggest using gradient penalty to stabilize GAN training[6, 4, 25, 11]. Mattya[17] compared several recent GAN variants under the same network architecture and measures their performance under the same metric.

Here, we use DRAGAN proposed by Kodali et al.[11] as the basic GAN model. As Mattya[17] shows, DRAGAN can give presumable results compare to other GANs, and it has the least computation cost among those GAN variants. Compare with Wasserstein GAN and its variants, DRAGAN can be trained under the simultaneous gradient descent setting, which make the training much faster. In our experiments, we also find it is very stable under several network architectures, we successfully train the DRAGAN with a SRResNet[12]-like generator, model details will be discussed in Section 5.1.

The implementation of DRAGAN is thus straightforward: we only need to sample some points in local regions around real images and force those samples to have norm 1 gradients with respect to the discriminator outputs. This can be done by adding a gradient penalty term to the generator loss. The flexibility of DRAGAN enables it to replace DCGAN in any GAN related tasks.

4.3 GAN with labels

Incorporating label information is important in our task to provide user a way to control the generator. Our utilization of the label information is inspired by ACGAN[20]: The generator G receive random noise z along with a 34-dimension vector c indicate the corresponding

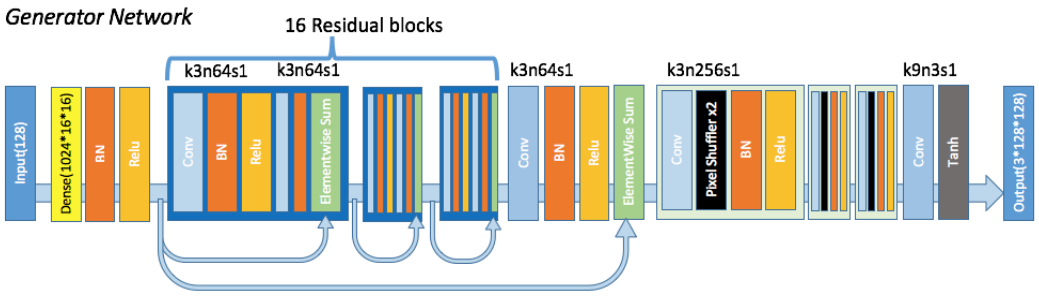


Figure 3: Generator Architecture

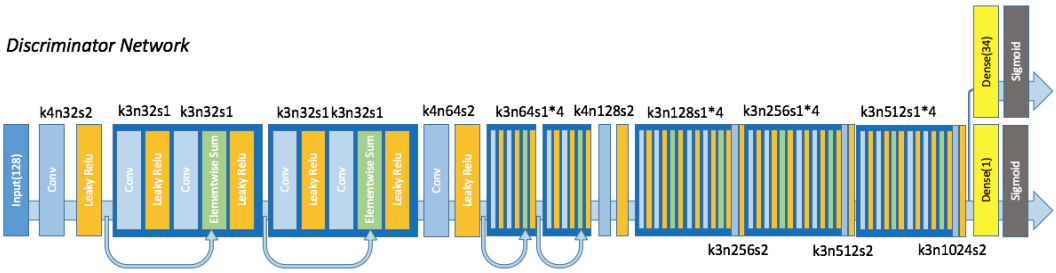


Figure 4: Discriminator Architecture

attribute conditions. We add a multi-label classifier on the top of discriminator network, which try to predict the assigned tags for the input images.

In detail, the loss is described as following:

$$
\begin{aligned}
\mathcal{L}_{adv}(D) &= -\mathbb{E}_{x\sim P_{data}}[\log D(x)] - \mathbb{E}_{x\sim P_{noise}, c\sim P_{cond}}[\log(1 - D(G(z,c)))] \\
\mathcal{L}_{cls}(D) &= \mathbb{E}_{x\sim P_{data}}[\log P_D[label_x|x]] + \mathbb{E}_{x\sim P_{noise}, c\sim P_{cond}}[\log(P_D[c|G(z,c)])] \\
\mathcal{L}_{gp}(D) &= \mathbb{E}_{\hat{x}\sim P_{perturbed_data}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2] \\
\mathcal{L}_{adv}(G) &= \mathbb{E}_{x\sim P_{noise}, c\sim P_{cond}}[\log(D(G(z,c)))] \\
\mathcal{L}_{cls}(G) &= \mathbb{E}_{x\sim P_{noise}, c\sim P_{cond}}[\log(P_D[c|G(z,c)])] \\
\mathcal{L}(D) &= \mathcal{L}_{cls}(D) + \lambda_{adv}\mathcal{L}_{adv}(D) + \lambda_{gp}\mathcal{L}_{gp}(D) \\
\mathcal{L}(G) &= \lambda_{adv}\mathcal{L}_{adv}(G) + \mathcal{L}_{cls}(G)
\end{aligned}
$$

where P_{cond} indicates the prior distribution of assigned tags. $\lambda_{adv}, \lambda_{gp}$ are balance factors for the adversarial loss and gradient penalty respectively.

5 Experiments

5.1 Training details

5.1.1 Model architecture

The generator's architecture is shown in Figure 3, which is a modification from SRResNet[12]. The model contains 16 ResBlocks and uses 3 sub-pixel CNN[27] for feature map upscaling. Figure 4 shows the discriminator architecture, which contains 10 Resblocks in total. All batch normalization layers are removed in the discriminator, since it would bring correlations within the mini-batch, which is undesired for the computation of the gradient

norm. We add an extra fully-connected layer to the last convolution layer as the attribute classifier. All weights are initialized from a Gaussian distribution with mean 0 and standard deviation 0.02.

5.1.2 Hyperparameters

We find that the model achieve best performance with λ_{adv} equaling to the number of attributes, as Zhou et al.[29] gives a detailed analysis of the gradient in the condition of ACGAN. Here, we set λ_{adv} to 34 and λ_{gp} to 0.5 in all experiments. All models are optimized using Adam optimizer[10] with β_1 equaling 0.5. We use a batch size of 64 in the training procedure. The learning rate is initialized to 0.0002 and exponentially decease after 50000 iterations of training.

5.1.3 Model Training

Figure 5: Sample images in different release years.

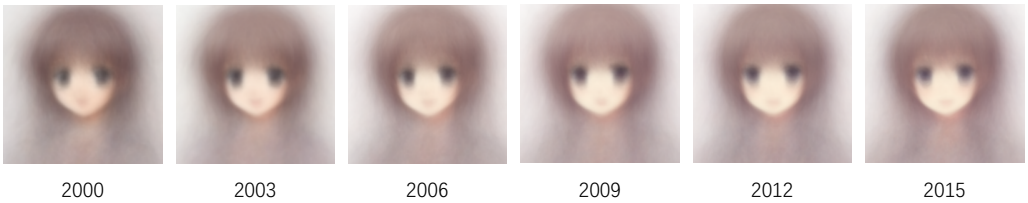


Figure 6: Average images from 1000 samples in different release years.

The technology of making game characters and CGs is evolving continuously, therefore the release year of the game plays an important role for the visual aspect of image quality. As we can see in the Figure 5, characters before 2003 look old-fashioned, while characters in the

recent games is cuter and have better visual quality. Appendix 8.2 shows the distribution of images in our dataset.

We train our GAN model using only images from games released after 2005 and with scaling all training images to a resolution of 128*128 pixels. This gives 31255 training images in total.

On the conditional generation of images, the prior distribution of labels P_{cond} is critical, especially when labels are not evenly distributed. In our case, there are only 49 training images assigned with the attribute "orange eyes" while 8861 images are assigned with the attribute "blue eyes".

But we don't take this in to account in the training stage. To sample related attributes for the noise, we use the following strategy. For the hair and the eye color, we randomly select one possible color with uniform distribution. For other attributes, we set each label independently with a probability of 0.25.

5.2 Generated Results

Figure 7 shows images generated from our model. Different from the training stage, we set the probability of each label based on the corresponding empirical distribution in the training dataset here.

By fixing the random noise part and sampling random conditions, the model can generate images have similar major visual features (e.g. face shapes, face directions). Figure 8 is an example of that. It is also an evidence of the generalization ability of visual concepts learnt from corresponding labels, indicating that our model can avoids the brute-force memorization of training samples.

Another phenomenon we empirically observed is that the random noise part heavily inference the quality of the final result. Some noise vector can give good samples no matter what conditioned on, while some other noise vectors are easier to produce distorted images.

As Table 1 states, labels are not evenly distributed in our training dataset, which results that some combinations of attributes cannot give good images. In Figure 9, (a)(b) are generated with well learned attributes like "blonde hair", "blue eyes". On contrast, (c)(d) are associated with "glasses", "drill hair", which is not well learned because of the insufficiency of corresponding training images. All characters in (a)(b) appear to be attractive, but most characters in (c)(d) are distorted.

5.3 Quantitative Analysis

5.3.1 Attribute Precision

blonde hair 1.00	brown hair 1.00	black hair 1.00	blue hair 0.70	pink hair 0.80	purple hair 0.75	green hair 0.90
red hair 0.95	silver hair 0.85	white hair 0.60	orange hair 0.65	aqua hair 1.00	gray hair 0.35	long hair 1.00
short hair 1.00	twintails 0.60	drill hair 0.20	ponytail 0.45	blush 1.00	smile 0.95	open mouth 0.95
hat 0.15	ribbon 0.85	glasses 0.45	blue eyes 1.00	red eyes 1.00	brown eyes 1.00	green eyes 1.00
purple eyes 0.95	yellow eyes 1.00	pink eyes 0.60	aqua eyes 1.00	black eyes 0.80	orange eyes 0.85	

Table 2: Precision of each label

To evaluate how each tag affect the output result, we measure the precision of the output result when the certain label is assigned. With each target, we fix the target label to true, and sample other labels in random. For each label, 20 images are drawn from the generator.

Figure 7: Generated samples

Then we manually check generated results and judge whether output images behave the fixed attribute we assigned. Table 2 shows the evaluation result. From the table we can see that compared with shape attributes(e.g. “hat”, “glasses”), color attributes are easier to learn. Notice that the boundary between similar colors like “white hair”, “silver hair”, “gray hair ” is not clear enough,. Sometimes people may have troubles to classify those confusing colors. This phenomenon lead to low precision scores for those attributes in our test.

Surprisingly, some rare color attributes like “orange eyes”, “aqua hair”, “aqua eyes” have a relative high precisions even though samples containing those attributes are less than 1% in the training dataset. We believe visual concepts related to colors are simple enough for the generator to get well learned with a extremely small number of training samples.

On contrast, complex attribute like “hat”, “glasses”, “drill hair” are worst behaved attributes in our experiments. When conditioned on those labels, generated images are often distorted and difficult to identify. Although there are about 5% training samples assigned with those attributes, the complicated visual concept they implied are far more accessible for the generator to get well learned.

Figure 8: Generated images with fixed noise part and random attributes

5.3.2 FID Evaluation

One possible quantitative evaluation method for GAN model is Fréchet Inception Distance(FID) proposed by Heusel et al.[7]. To calculate the FID, they use a pre-trained CNN(Inception model) to extract vision-relevant features from both real and fake samples. The real feature distribution and the fake feature distribution are approximated with two guassian distributions. Then, they calculate The Fréchet distance(Wasserstein-2 distance) between two guassian distributions and serve the results as a measurement of the model quality.

The Inception model trained on ImageNet is not suitable for extracting features of anime-style illustrations, since there is no such images in the original training dataset. Here, we replace the model with Illustration2vec feature extractor model for better measurement of visual similarities between generated images and real images.

To evaluate the FID score for our model, we sample 12800 images from real dataset, then generate a fake sample by using the corresponding conditions for each samples real images. After that we feed all images to the Illustation2vec feature extractor and get a 4096-dimension

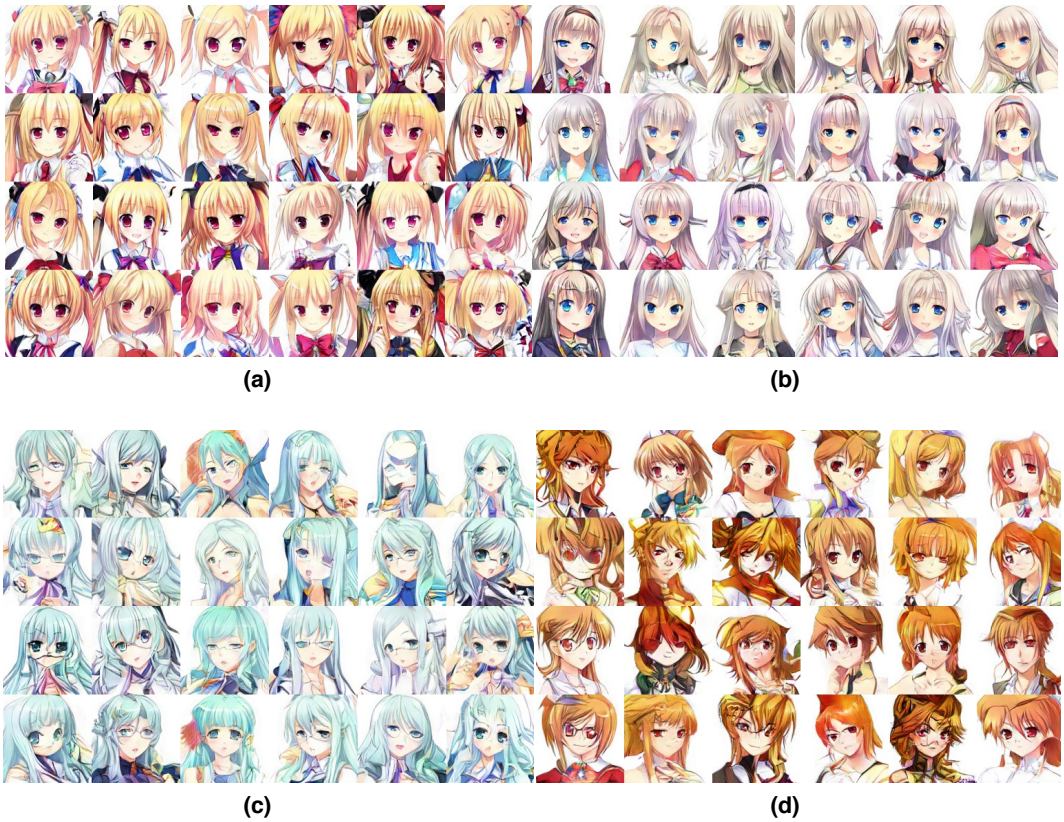


Figure 9: Generated images under fixed conditions. (a) blonde hair, twintails, blush, smile, ribbon, red eyes (b) silver hair, long hair, blush, smile, open mouth, blue eyes (c) aqua hair, long hair, drill hair, open mouth, glasses, aqua eyes (d) orange hair, ponytail, hat, glasses, red eyes, orange eyes

Model	Average FID	MaxFID-MinFID
DCGAN Generator+DRAGAN	5974.96	85.63
Our Model	4607.56	122.96

Table 3: FID of our model and baseline model

feature vector for each image. FID is calculated between the collection of feature vectors from real samples and that from fake samples.

For each model, we repeat this process for 5 times and measure the average score of 5 FID calculation trails. Table 3 shows the result comparing our model with the baseline model. We observe that SRResNet based model can achieve better FID performance evenly with less weight parameters.

5.4 Website Interface

In order to make our model more accessible, we build a website interface[8] for open access. We impose WebDNN[9] and convert the trained Chainer model to the WebAssembly based Javascript model. The web application is built with React.js.

[8] `http://make.girls.moe`
[9] `https://mil-tokyo.github.io/webdnn/`

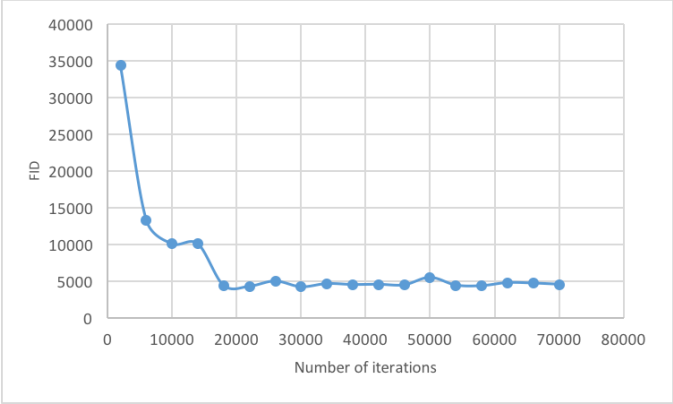


Figure 10: FID decrease and converge to a certain value during the training procedure

Keeping the size of generator model small would be a great benefit when hosting a web browser based deep learning service. This is because user are required to download the model before the computation every time, bigger model results much more downloading time which will affect the user experience. Replacing the DCGAN generator by SRResNet generator can make the model 4x smaller, so the model downloading time can be reduced by a large margin.

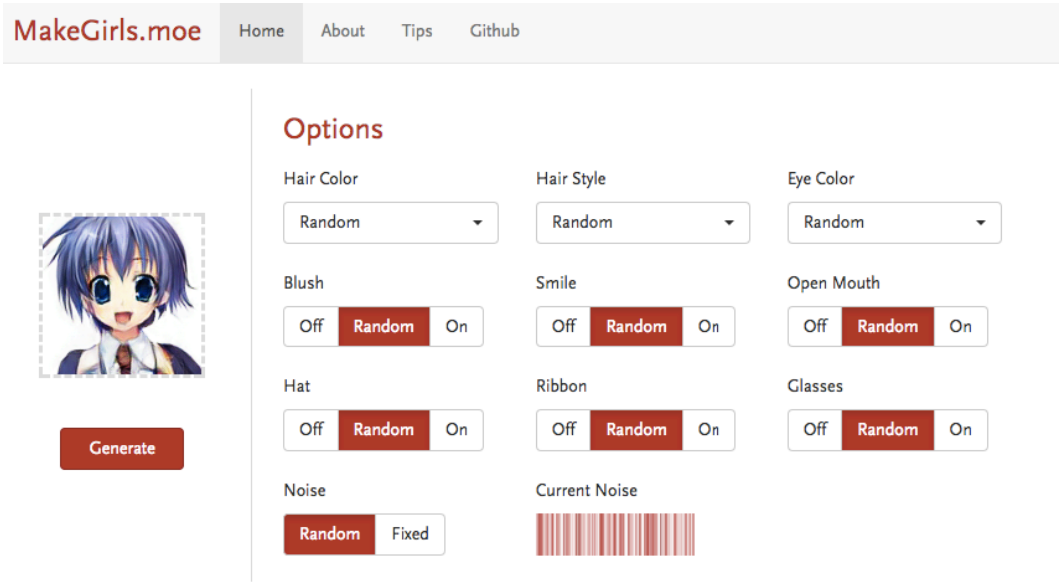


Figure 11: Our website

Users can manually assign attributes to the generator, and all unassigned attributes will be sampled based on empirical distribution of the training dataset. All computations are done on the client side. It takes about 6 $\sim$ 7s to generate one images on average, the detailed performance test is discussed in Appendix 8.4.

5.5 Super-Resolution

As Appendix 8.2 shows, the resolution of available training images are not high, making generation of high resolution facial images from the GAN model directly is difficult. Here

we try to build another Super-Resolution network specifically for generation extra high resolution animation style images to overcome the limitation.

Image (b) in Figure 12 shows the 2x upscaled image with waifu2x[10], which is blurred. [12] gives a comparison of GAN-based super-resolution model and traditional MSE-based super-resolution model. Their result shows that GAN based models can bring more high-frequency details to the upscaled images than MSE only models. This is preferred in our situations, so we choose to implement SRGAN[12] as our super-resolution model.

Image (c) (d) show our attempts of training a SRGAN. (c) is trained with a low adversarial loss weight and (d) is trained with high adversarial loss weight. We observe that as the weight of the adversarial loss increasing, the upscaled image looks sharper. However, this would bring more undesired artifacts to output images.

We observe that the visual quality of anime-style images are more sensitive to extra artifacts than real photos. We hypothesize that it is due to the fact that color/texture patterns in anime-style images are much simpler and clearer than real photos, any artifacts would largely damage the color/texture patterns and make the result looks dirty and messy. This obstacle limit the usage of GAN in our super-resolution networks.

Discouragingly, we failed to find a model balanced well between the sharpness level and the artifact strengths, so we choose not add the super-resolution model to our website for now, and leave the exploration of anime image focused super resolution for future work.

Figure 12: Results of 2x super-resolution. (a) The original image. (b) Result from waifu2x (c)SRGAN with lower adversarial loss (d) SRGAN with higher adversarial loss

6 Conclusion

We explore the automatic creation of the anime characters in this work. By combining a clean dataset and several practicable GAN training strategies, we successfully build a model

[10] `http://waifu2x.udp.jp/index.ja.html`

which can generate realistic facial images of anime characters. We publish this report along with an easy-to-use website service.

There still remain some issues for us for further investigations. One direction is how to improve the GAN model when class labels in the training data are not evenly distributed. Also, quantitative evaluating methods under this scenario should be analyzed, as FID only gives measurement when the prior distribution of sampled labels equals to the empirical labels distribution in the training dataset. This would lead to a measure bias when labels in the training dataset are unbalanced.

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

We hope our work would stimulate more studies on generative modeling of anime-style images and eventually help both amateurs and professionals design and create new anime characters.

7 Acknowledgement

This report is published as a Doujinshi in Comiket 92, summer 2017, with the booth number 三日目東ウ 05a.

The work is done when Yanghua Jin works as a part-time engineer in Preferred Networks, Japan. Special thanks to Eiichi Matsumoto, Taizan Yonetsuji, Saito Masaki, Kosuke Nakago from Preferred Networks for insightful directions and discussions.

The cover illustration is created by Zhihao Fang and Jiakai Zhang helps create the website.

References

[1] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[3] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and equilibrium in generative adversarial nets (gans). *arXiv preprint arXiv:1703.00573*, 2017.

[4] Marc G Bellemare, Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer, and Rémi Munos. The cramer distance as a solution to biased wasserstein gradients. *arXiv preprint arXiv:1705.10743*, 2017.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[7] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *arXiv preprint arXiv:1706.08500*, 2017.

[8] Hirobashi. Girl friend factory - 機械学習で彼女を創る -. `http://qiita.com/Hiroshiba/items/d5749d8896613e6f0b48`, 2016.

[9] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

[10] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. How to train your dragan. *arXiv preprint arXiv:1705.07215*, 2017.

[12] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint arXiv:1609.04802*, 2016.

[13] Jie Lei. Animegan. `https://github.com/jayleicn/animeGAN`, 2017.

[14] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. *arXiv preprint arXiv:1705.08584*, 2017.

[15] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[16] Mattya. chainer-dcgan. `https://github.com/mattya/chainer-DCGAN`, 2015.

[17] Mattya. chainer-gan-lib. `https://github.com/pfnet-research/chainer-gan-lib`, 2017.

[18] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

[19] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[20] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. *arXiv preprint arXiv:1610.09585*, 2016.

[21] Guo-Jun Qi. Loss-sensitive generative adversarial networks on lipschitz densities. *arXiv preprint arXiv:1701.06264*, 2017.

[22] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[23] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[24] Rezoolab. Chainer を使ってコンピュータにイラストを描かせる. `http://qiita.com/rezoolab/items/5cc96b6d31153e0c86bc`, 2015.

[25] Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Stabilizing training of generative adversarial networks through regularization. *arXiv preprint arXiv:1705.09367*, 2017.

[26] Masaki Saito and Yusuke Matsui. Illustration2vec: a semantic vector representation of illustrations. In *SIGGRAPH Asia 2015 Technical Briefs*, page 5. ACM, 2015.

[27] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1874–1883, 2016.

[28] tdrussell. Illustrationgan. `https://github.com/tdrussell/IllustrationGAN`, 2016.

[29] Zhiming Zhou, Shu Rong, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. Generative adversarial nets with labeled data by activation maximization. *arXiv preprint arXiv:1703.02000*, 2017.

8 Appendix

8.1 SQL query on ErogameScape

```
SELECT g.id, g.gamename, g.sellday,
      'www.getchu.com/soft.phtml?id=' || g.comike as links
FROM gamelist g
WHERE g.comike is NOT NULL
ORDER BY g.sellday
```

8.2 Dataset images distribution

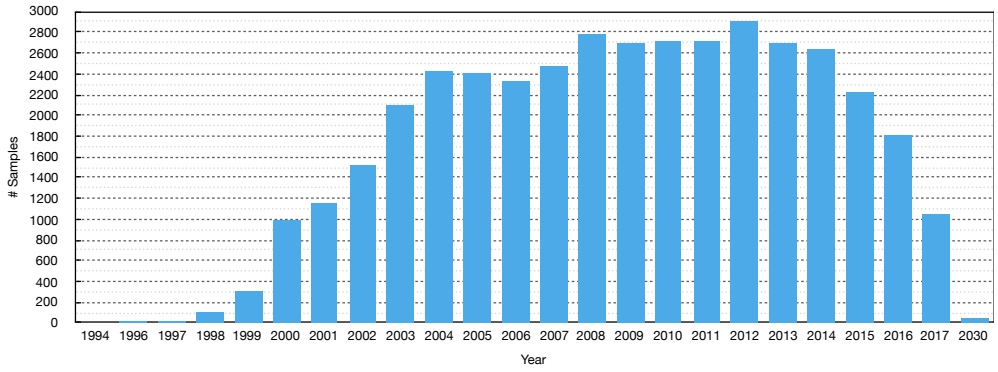


Figure 13: Available dataset images by release years. Note that year=2030 means release year undetermined

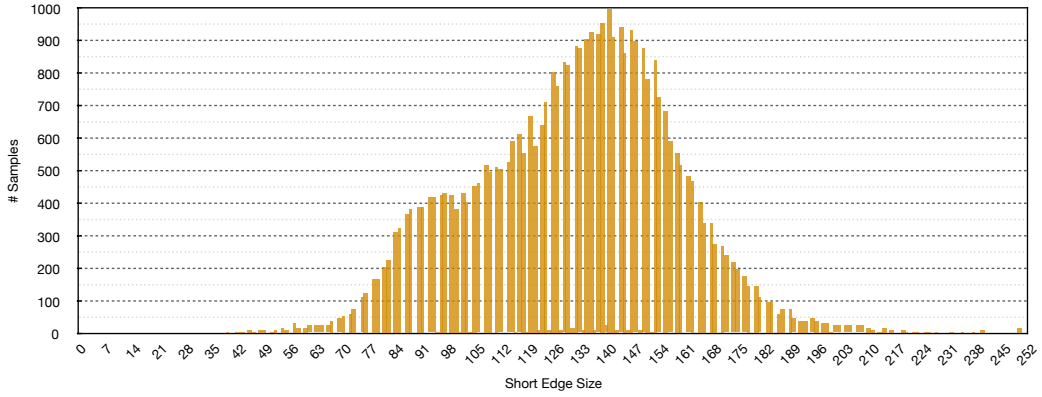


Figure 14: Available dataset images by short edge size.

8.3 Interpolation between random generated samples

Figure 15: Samples in the first column and the last columns are randomly generated under different combinations of conditions. Although label controlling variables are assigned with discrete values in the training stage, the result shows that those discrete attributes are still meaningful under the continuous setting.

8.4 Approximate inference time

Processor	Operation System	Web Browser	Execution Time (s)
I7-6700HQ	macOS Sierra	Chrome 59.0	5.55
I7-6700HQ	macOS Sierra	Safari 10.1	5.60
I5-5250U	macOS Sierra	Chrome 60.0	7.86
I5-5250U	macOS Sierra	Safari 10.1	8.68
I5-5250U	macOS Sierra	Safari Technology Preview 33*	<0.10
I5-5250U	macOS Sierra	Firefox 34	6.01
I3-3320	Ubuntu 16.04	Chromium 59.0	53.61
I3-3320	Ubuntu 16.04	Firefox 54.0	4.36
iPhone 7 Plus	iOS 10	Chrome	4.82
iPhone 7 Plus	iOS 10	Safari	3.33
iPhone 6s Plus	iOS 10	Chrome	6.47
iPhone 6s Plus	iOS 10	Safari	6.23
iPhone 6 Plus	iOS 10	Safari	11.55

Table 4: Approximate inference time on several different environments, note that for Safari Technology Preview, the computation is done by WebGPU ,while for other browsers, the computation is done by WebAssembly. We can see that firefox is better optimized with WebAssembly and faster than other browsers