Towards the High-quality Anime Characters Generation with Generative Adversarial Networks

Yanghua Jin[1] Jiakai Zhang[2] Minjun Li[1] Yingtao Tian[3] Huachun Zhu[4]
[1]School of Computer Science, Fudan University
[2]School of Computer Science, Carnegie Mellon University
[3]Department of Computer Science, Stony Brook University
[4]School of Mathematics, Fudan University
[14]{jinyh13,minjunli13,zhuhc14}@fudan.edu.cn [2]jiakaiz1@andrew.cmu.edu
[3]yittian@cs.stonybrook.edu

1 Introduction

The automatic generation of anime characters offers an opportunity to bring a custom character into existence without professional skill. Besides, professionals may also take advantages of the automatic generation for inspiration on animation and game character design. however results from existing models [15, 18, 8, 22, 12] on anime image generation are blurred and distorted on an non-trivial frequency, thus generating industry-standard facial images for anime characters remains a challenge. In this paper, we propose a model that produces anime faces at high quality with promising rate of success with three-fold contributions: A clean dataset from Getchu, a suitable DRAGAN[10]-based SRResNet[11]-like GAN model, and our general approach to training conditional model from image with estimated tags as conditions. We also make available a public accessible web interface.

2 Dataset Construction and Generative Model

We propose to use a consistent, clean, high-quality anime dataset collected from from Getchu, a website for Japanese games (process detailed in Appendix A). The generation of images with customization requires categorical metadata of images in priors as attributes along with noise. Since Getchu does not provide such metadata, we use Illustration2Vec[20], a CNN-based tool for anime illustrations that serves as attribute estimation. We show the details of attribute estimation with its statistics and visualization in Appendix B.

Generative Adversarial Networks (GAN) [5] are implicit generative models leading to the impressive results. However it is notoriously hard to train properly GAN and several methods have been proposed to address this issue [1, 3, 2, 13, 4, 6, 4, 19, 10, 16]. Here, we use DRAGAN [10] as GAN model which are fast and stable under several network architecture. With it we successfully train the DRAGAN with a SRResNet-like generator. Inspired by ACGAN [17], we utilize the attributions by feeding them along with noise vector and add a multi-label classifier on the top of discriminator for reconstructing the attributions. The network architecture, loss function and training details can be found in Appendix C.

3 Generated Images

We highlights some high-resolution (256 by 256) images generated from our model. Details of protocol and more examples can be found in Appendix D.

Submitted to Machine Learning for Creativity and Design, NIPS 2017 Workshop, Long Beach, California, USA.

Randomly Generated Examples: Figure 1 shows images generated from our model where both noise and attributes are randomly sampled.
Generated Examples with Fixed Noise: In Figure 2 we show that by fixing the random noise part and sampling random attribution, the model generates images have similar major visual features like face shapes and directions, an evidence of the generalization ability.
Generated Examples with Fixed attribution: In Figure 3 we show generated images from fixed attribution and randomly noise. The model here generates images with desired attributions but with different, variant visual features.
Interpolation between images: In Figure 4 we show the interpolation between two sets of randomly selected features. It shows that attributes, like noise are meaningful under the continuous setting.

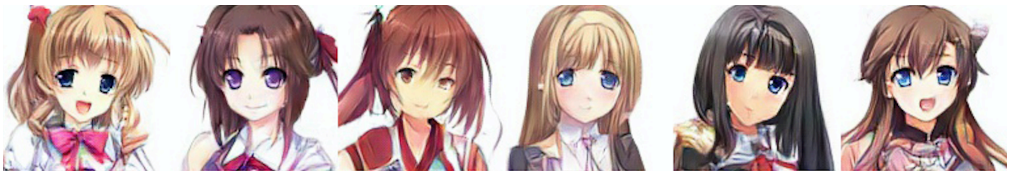

Figure 1: Generated samples with random prior.

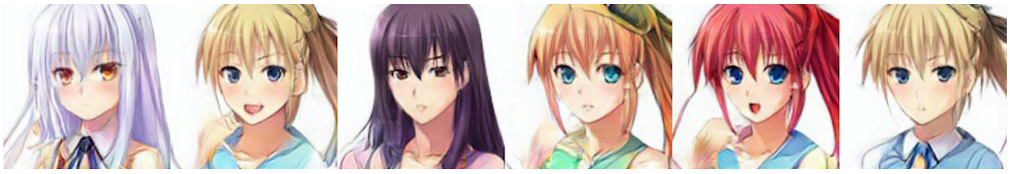

Figure 2: Generated images with fixed noise part and random attributes.

Figure 3: Generated images with fixed conditions (silver hair, long hair, blush, smile, open mouth, blue eyes) and random noise part.

Figure 4: Interpolation between images.

4 Analysis, Public Accessible Interface and Acknowledgement

For quantitively evaluating our generate examples, we conduct *attribute precision* (Appendix E), *FID evaluation* (Appendix F) and *nearest training examples* (Appendix G) that shows the superiority of our model over DCGAN baseline. In order to make our model more accessible, we build a website interface[1] for open access (Appendix I).

This paper was first presented as a Doujinshi in Comiket 92, summer 2017 (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.

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

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] Hiroshiba. Girl friend factory. `http://qiita.com/Hiroshiba/items/d5749d8896613e6f0b48`, 2016.

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

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

[11] 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.

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

[13] 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.

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

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

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

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

[18] Rezoolab. Make illustration on computer with chainer. `http://qiita.com/rezoolab/items/5cc96b6d31153e0c86bc`, 2015.

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

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

[21] 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.

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

[23] 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.

Appendix

A Image Collection

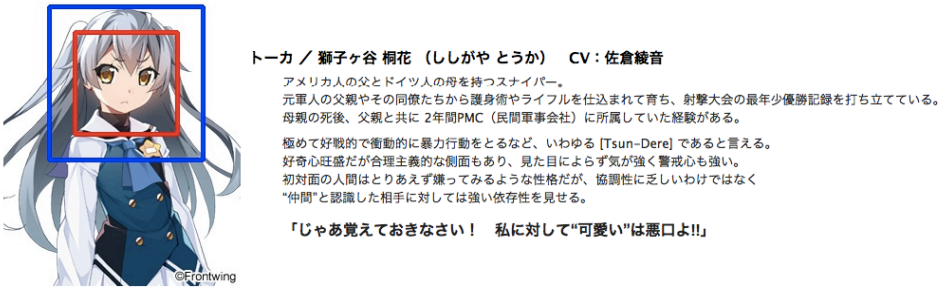

Figure 5: Sample Getchu page and the detection result (`http://www.getchu.com/soft.phtml?id=933144`). Red line indicate the original bounding box and blue line indicate the scaled bounding box. Copyright: Frontwing, 2017

Getchu[2] is a website providing information and selling of Japanese games, for which there are character introduction sections with standing pictures (立ち絵). Figure 5 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 the following SQL query on ErogameScape's Web SQL API page[3] to get the Getchu page link for each game:

```
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
```

Then we download images following returned list of urls from the SQL query, and apply lbpcascade animeface[4], 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 attributesincluding hair length and hair style, so we zoom out the bounding box by a rate of 1.5x. The difference can be observed in Figure 5. 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.

B Tag Estimation

To overcome the limitation that images collected from Getchu are without any tag, we use Illustration2Vec[20], a pre-trained[5] CNN-based tool for (noisy) estimating tags of anime illustrations. 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

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

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

Table 1: Number of dataset images for each tag

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.

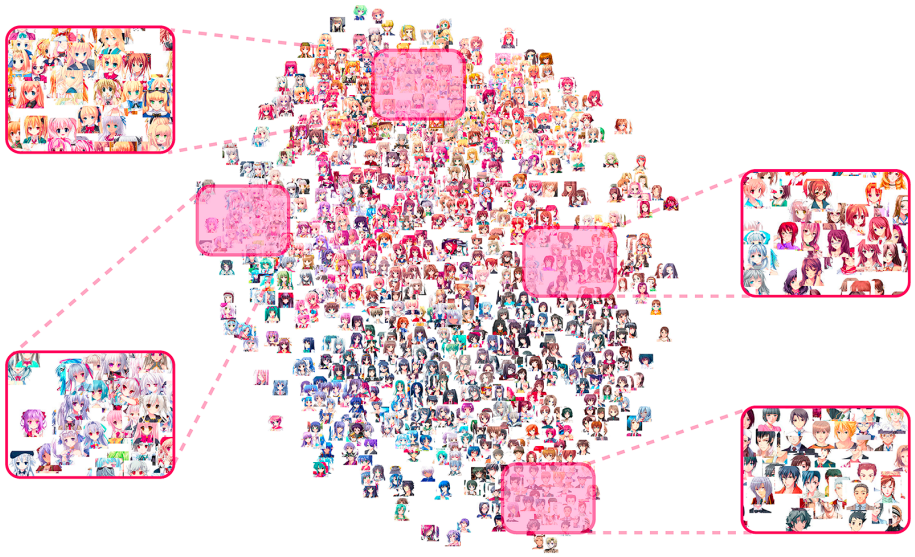

Figure 6: t-SNE visualization of 1500 dataset images. A clustering in terms of similar attributes can be observed in close-up views.

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[14]. Figure 6 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.

C Model Details

C.1 Network Architecture

The generator’s architecture is shown in Figure 7, which is a modification from SRResNet[11]. The model contains 16 ResBlocks and uses 3 sub-pixel CNN[21] for feature map upscaling. Figure 8 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.

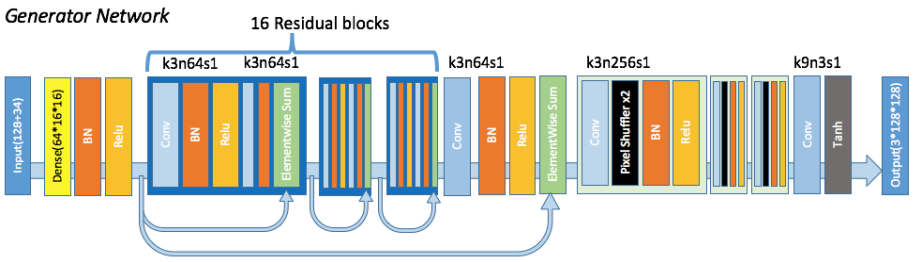

Figure 7: Generator Architecture

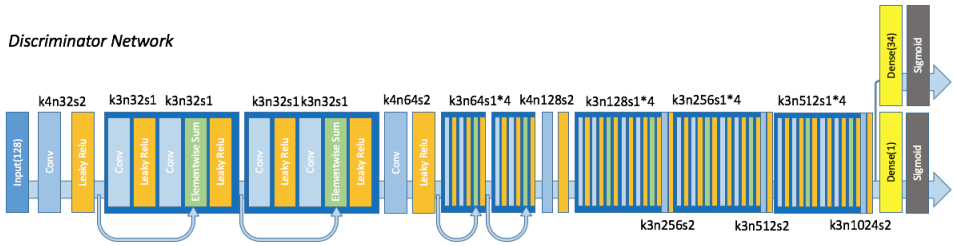

Figure 8: Discriminator Architecture

C.2 Loss Function

The loss function 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.

C.3 Training details

We find that the model achieve best performance with λ_{adv} equaling to the number of attributes, as Zhou et al.[23] 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[9] 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.

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.

we use the following simple strategy to sample related attributes for the noise. For the categorial attributes (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.

D More Generated Images

Figure 9 shows more images generated from our model. Figure 10 is an example of fixing the random noise part and sampling random attributes, where the model can generate images have similar major visual features. In sampling random attributes, we use the following simple strategy: For the categorial attributes (hair and the eye color), we randomly select one possible attribute uniformly. For other attributes, we set each label independently with a probability of 0.25.

Figure 9: Generated samples with random noise and attributes.

We empirically observe 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 11, (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.

We also show more interpolations in Figure 12. 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.

Figure 10: Generated images with fixed noise and random attributes.

E 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. 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

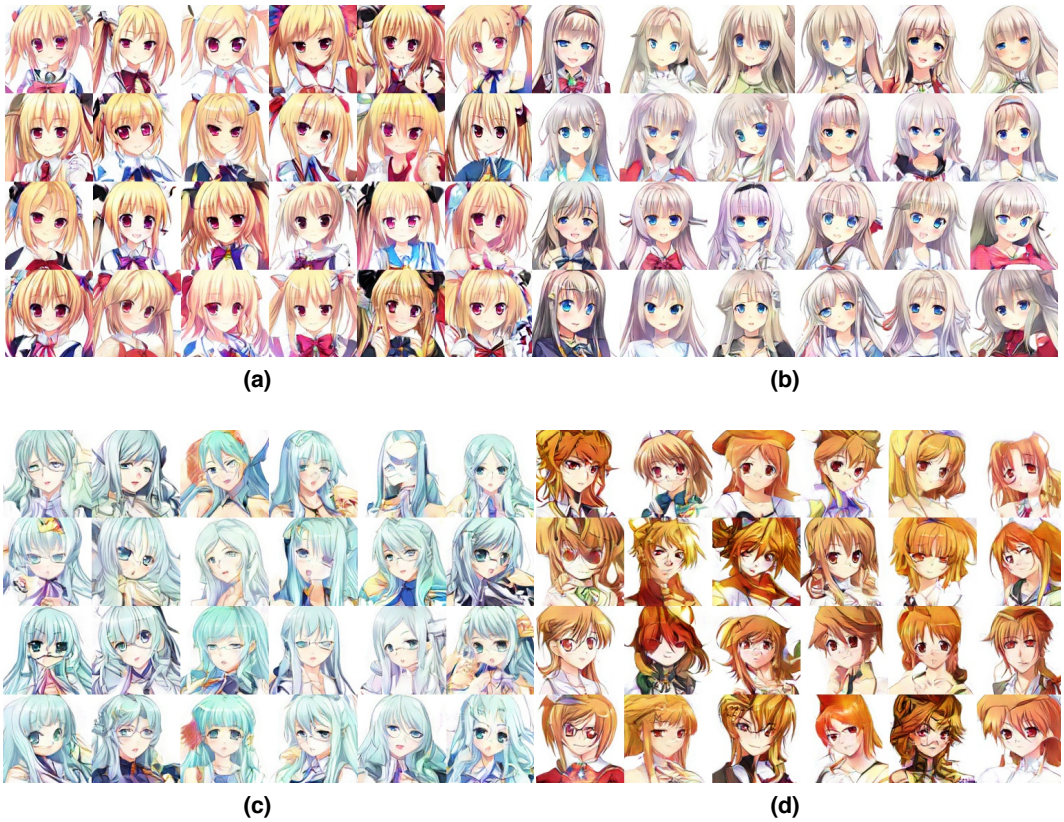

Figure 11: Generated images with random noise and following fixed attributes: (a) blonde hair, twintails, blush, smile, ribbon, red eyes (b) silver hair, long hair, blush, smile, open mouth, blue eyes (c) aqua hair, long hair, drill hair, open mouth, glasses, aqua eyes (d) orange hair, ponytail, hat, glasses, red eyes, orange eyes

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.

F FID Evaluation

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

Figure 12: Interpolations where samples in the first column and the last columns are randomly generated under different combinations of conditions and samples between them are result of interpolated latent points.

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 Inception model with Illustration2vec feature extractor model for better measurement of visual similarities between generated images and real images.

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

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 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 our model can achieve better FID performance evenly with less weight parameters. Also in Figure 13 we trace the FID change in training.

G Nearest Image in Training Set

We would like to know whether our model learns to cheat by generating images in the training set. In Figure 14 we show randomly generated examples and their nearest images in training set. It can be shown that our model learns to produce images with new visual features unseen in the training set, while incorporating the abstract concepts in terms of attributes.

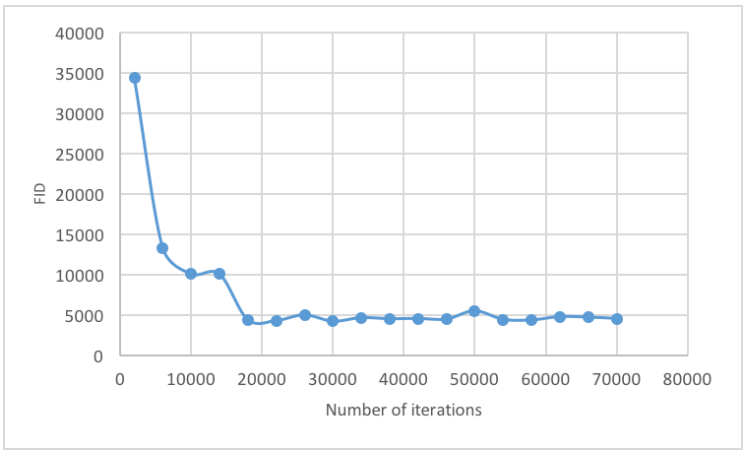

Figure 13: FID decrease and converge to a certain value during the training procedure

Figure 14: Generated images and their nearest image in training set. In each row, on the left we show a randomly generated images, and on the right we show a list of its nearest images in the training set, measured by decreasing L2 distance on features from Illustration2Vec feature extractor. Our model learns to incorporate abstract concepts in terms of attributes (hair style, hair color, etc.) into new, unseen visual features (facial expressions, etc.)

H Evolution of Anime Characters

As an extra experiment, we add the releasing year information as another attribute to the model. Since the main stream of popular anime character styles is continuous evolving, adding year label can help the model catch the prevalent style in each year. Predicting the future of anime character styles is also possible by adjusting the corresponding input value. We made two videos for better demonstrating the result[6] [7].

Figure 15: The leftmost column indicates generated images conditioned on year 2003, and the rightmost column indicates generated images conditioned on year 2017.

I Public Accessible Interface

We impose WebDNN[8] and convert the trained Chainer model to the WebAssembly based Javascript model. WebGL and WebGPU based computation models are also accessible when clients meet requirements. We observe that the inference procedure can be more than 100 times faster by enabling GPU acceleration. The web application is built with React.js.

[6] https://www.youtube.com/watch?v=WR8XnX6W8Bk
[7] https://www.youtube.com/watch?v=dClVF_X5PMU
[8] https://mil-tokyo.github.io/webdnn/

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.

We details the running time in Table 4. From which we believe that the our support for multiple client-side browsers generally results in acceptable running time and allows smooth user experience, while we also make use of state-of-the-art technology (WebGPU) when it is applicable as a proof-of-concept for next-generation user experience.

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	Firefox 34	6.01
Intel HD Graphics 6000	macOS Sierra	Safari 11.0(WebGPU)	<0.10
Intel HD Graphics 6000	macOS Sierra	Chrome 60.0 (WebGL)	0.42
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.