Artificial Anime Character Design: An Application of Generative Adversarial Networks (GANs)

University of Pennsylvania | CIS 519 - Applied Machine Learning

Roy Wu wuroy@seas.upenn.edu
Riley Xu rileyx@sas.upenn.edu
Tianhao Lu tianhlu@seas.upenn.edu

Abstract

Generative Adversarial Networks (GANs) are novel deep-learning architectures that excel in creating artificial images. GANs have been successfully applied to many types of images, most notably human faces, but each category comes with unique challenges. We focus on using GAN to generate artificial anime character faces. We implement three variations of GAN - DCGAN, LSGAN, and StyleGAN, and trained on a data set of 20,000 images. The models were evaluated both qualitatively and with the Frechet Inception Distance.

1. Introduction

Generative Adversarial Network (GAN) is a framework for Deep Learning models to generate superficial data mimicking a training distribution (Goodfellow et al., 2014). Facebook's AI research director Yann LeCun called adversarial training "the most interesting idea in the last 10 years of ML" (LeCun, 2016). GANs have been successfully applied to many image categories, including objects as complicated as human faces, such as in ThisPersonDoesNotExist.com (Wang, 2019). There are many other attempts to apply GANs on paintings, objects, and even Pokémon. In this work, we train and generate images of character faces from Japanese anime.

Anime is an incredibly popular media format around the world. However, coming up with new character designs takes tremendous effort and skill. GANs offer the opportunity to create new and custom designs without extensive input from a professional. This may help anime studios dramatically shorten development times, provide inspiration to professional and hobbyist character designers, and with sufficient progress may even be able to take over some aspects of the animation.

There exist few attempts applying the GAN model to the problem of generating facial images of anime characters. DRAGAN gives a promising result (Jin et al., 2017); however, other attempts tend to be limited to blog posts and personal github repos (Mattya, 2016; Jayleicn, 2017; Branwen, 2019), and the methodology and data samples are not well described by these sources. Moreover, the results of these projects are frequently defective, with effects such as asymmetric facial features and having large swaths that make them look like water-color paintings. Most disappointingly, these authors only generated female character faces.

In this work, we seek to improve in the generation of anime character faces. In the following sections, we first describe the data set that we use. Then, we detail three GAN model architectures and show the results. Finally, we compare and quantify the differences between the architectures.

2. Data Collection & Preprocessing

Previous works mentioned above have datasets that suffer from high variance and noise, leading to results that were not too promising. Our goal for the data preparation is to find a clean, high-quality dataset and preprocess it in such a way that it can be used seamlessly with PyTorch.

Our dataset consists of 21551 unlabeled anime faces obtained from Kaggle (Rakshit, 2019). This is a cleaner version of a dataset originally on Github (Chao, 2019) where the images are fetched from Getchu.com and then cropped using the anime face detection algorithm (Nagadomi, 2011).

The images are generally high-quality, but a few suffer from bad crops. Almost all images have white or otherwise minimal background. The range in art styles, pose, expressions, and hair is sufficiently variant. Notably, this data set does contain male characters, albeit at a smaller proportion, and our networks are trained to generate male faces unlike previous works.

Figure 1. Sample images in the Kaggle dataset.

We transformed and loaded the dataset using Pytorch's ImageFolder and DataLoader classes. We removed as many of the bad samples as possible, mainly poor crops. Additionally, all images are resized to 64x64 for further convenience.

3. Generative Adversarial Network

GAN uses a generator network G to generate fake samples by mapping a latent space vector z sampled from a normal distribution into a sample $G(z)$. GAN uses a min-max game where discriminator network D tries to maximize its accuracy in classifying real and fake data and G tries to minimize the probability that D will predict its outputs are fake (Goodfellow et al., 2014). The objective can be formally expressed as:

$$\min_G \max_D V(D,G) = E_x[\log D(x)] + E_z[\log(1 - D(G(z)))]$$

The generator and discriminator train off of each other, enabling a GAN to learn unsupervised. In general, the training of a GAN occurs as below:

1. Create a standard normally distributed vector z, with arbitrary feature length.
2. Zero out gradient for G, generate fake image, and calculate L_G, where:

$$L_G = \frac{1}{n}\sum_{i=0}^{n} L_{CE}(D(G(z)), 1)$$

3. Backpropogate L_G and step up the optimizer
4. Zero out gradient for D and calculate L_D, where:

$$L_D = \frac{1}{2n}\sum_{i=0}^{n} L_{CE}(D(X_i), 1) + L_{CE}(D(G(z)), 0)$$

5. Backpropogate L_D and step up optimizer

While all GANs have the same basic structure detailed above, there exist many subtleties that result in very different implementations. We explore three implementations: Deep Convolutional, Least Squares, and StyleGAN.

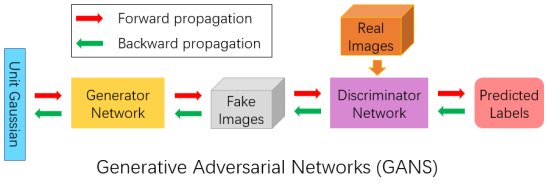

Figure 2. Overview of GAN process and data flow.

4. Deep Convolutional GAN (DCGAN)

Our base model architecture is inspired by Deep Convolutional GAN (DCGAN) (Radford et al., 2015). The discriminator is made up of strided convolution layers, 2d batch norm layers, LeakyReLUs, and outputs the final probability through a Sigmoid activation function. The generator consists of a series of strided two dimensional convolutional transpose layers, each followed by a 2d batch norm layer and a ReLU activation. The outputs go through a tanh function.

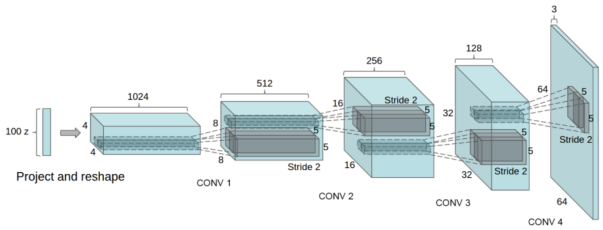

Figure 3. Generator architecture (Radford et al., 2015). It projects latent vector z to a 64x64 RBG image through a series of fractionally-strided convolutions

4.1. Training and Hyperparameter Tuning

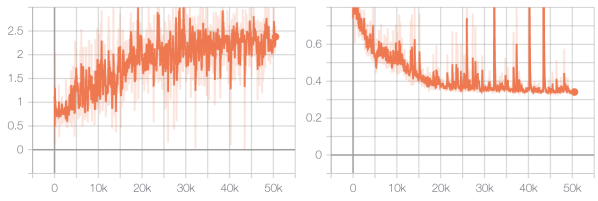

Figure 4. DCGAN L_G (left) and L_D (right) during training.

Figure 4 shows L_G and L_D throughout our training. Some things we watched out for during training:

1. We want to prevent the L_D from going to 0 (this represents a failure mode of training).
2. Both L_D and L_G should not decrease monotonically (they should oscillate around a certain loss), with L_G increasing as a whole and L_D decreasing as a whole.

3. Display sample generated images every a certain number of iterations to watch for mode collapse.

We used a batch size of 64 and a feature length of 100 for vector z. For both the G and D, we use Adam optimizer because it combines the best properties of the AdaGrad and RMSProp algorithms, which can handle sparse gradients on noisy problems. Since D learns much faster than G, we set the learning rate for optimizer for G at 0.0005 and optimizer for D at 0.0001. Additionally, we used fuzzy labels - instead of a hard 0 or 1 as labels for the discriminator, we set them to 0.1 and 0.9 to make the discriminator weaker.

4.2. Results

Uncurated sample images from our DCGAN model are shown in Figure 5. The model was trained on 150 epochs. We find these results quite promising, because most of the generated images are convincing. However, some mistakes are made by the model, such as twisted and broken faces, asymmetric eyes, and non-uniformity in hair. The flaws urge us to try new models.

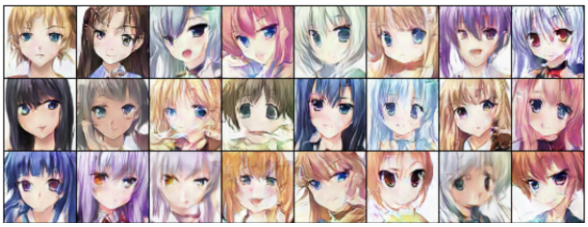

Figure 5. Uncurated DCGAN generated samples.

5. Least Squares GAN (LSGAN)

A large problem with DCGAN is that it can suffer from vanishing gradients. A variation called Least Squares GAN (LSGAN) attempts to combat the vanishing gradients by using a loss function that provides smooth and non-saturating gradient in discriminator D (Mao et al., 2016). Formally, the objective functions can be expressed as such:

$$\min_D V_{LS}(D) = \frac{1}{2}E_x[(D(x)-1)^2] + \frac{1}{2}E_z[(D(G(z)))^2]$$
$$\min_G V_{LS}(G) = \frac{1}{2}E_z[(D(G(z))-1)^2]$$

5.1. Training and Hyperparameter Tuning

The model architecture we used for LSGAN is exactly the same as that of DCGAN, with a few minor changes. First, we removed the sigmoid activation function at the end of D. Moreover, we adjusted the L_G and L_D functions to follow the objective functions described above. We also use the same hyperparameters as we did for DCGAN.

5.2. Results

Uncurated samples are shown in Figure 6, trained on 150 epochs. As expected, LSGAN performs more stable during the learning process. Unfortunately, our generated samples appear to have more artifacts and less variation.

Figure 6. Uncurated LSGAN generated samples.

6. StyleGAN

StyleGAN and StyleGAN2 are new state-of-the-art GAN architectures developed by a team from NVIDIA (Karras et al., 2018; 2019). These architectures address many of the shortcomings of previous GAN implementations and dramatically increase image quality.

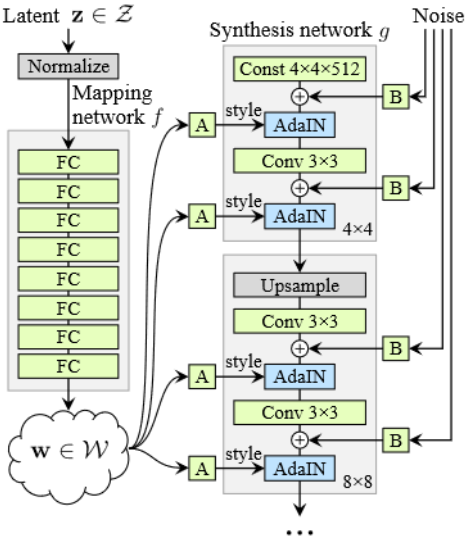

Figure 7. StyleGAN architecture (Karras et al., 2018).

StyleGAN uses a progressive architecture (Karras et al., 2017). Both the generator and discriminator are first trained only on 4x4 pixel images. After sufficient progress, a new layer block is added to both networks to handle 8x8 images. This doubling procedure continues until reaching the desired image size. This progression allows the generator to learn fine features after already having learned coarse features, significantly easing the training.

Figure 8. Uncurated StyleGAN generated samples; random (left) and style mixing (right).

Another key innovation is the adoption of techniques from style-transfer learning. Instead of inputting the latent vector directly into the first layer of the convolution, there is a separate mapping network that processes the latent vector into an intermediate latent vector. This is then transformed into a style vector y_i, via a learned affine transformation, and fed into the synthesis network between each convolution layer. The style transfer is done via adaptive instance normalization (AdaIN):

$$\text{AdaIN}(x, y_i) = y_{i,s}\frac{x - \bar{x}}{\sigma(x)} + y_{i,b}.$$

Above, $y_i = (y_{i,s}, y_{i,b})$ is the style scale and bias in layer i, and x is the input activations from the previous layer.

6.1. Results

Using a baseline StyleGAN architecture implemented in PyTorch, we achieved remarkable results after 150 epochs of training. Uncurated sample fakes are shown in Figure 8. Artifacts rarely appear, and there is huge variation between samples. By eye we are generally unable to distinguish fake from real images.

6.2. Mixing Regularization

A large benefit of separating out the mapping network and using AdaIN in StyleGAN is the ability to mix styles. Specifically, midway through generation, the latent vector can be swapped. This is done during training as a form of regularization, to force independence between styles in adjacent layers. Style mixing can also be done manually when generating fakes to manually alter high-level features, such as hair color or pose. An example array of mixing styles is shown on the right in Figure 8.

7. Quantifying Results

A common metric to quantify the quality of generated images is the Frechet Inception Distance (FID) (Heusel et al., 2017). FID is an improvement on the Inception Score (Salimans et al., 2016) and calculates the similarity between two sets of images, unsupervised. As with Inception Score, FID uses the Inception V3 model, a trained classifier on 1000 objects (Google, 2015). However, instead of using the output of the Inception V3 model, FID calculates the Frechet distance of the activations in the coding layer. Because classifier networks tend to learn general features, using the coding layer generalizes the Inception Score to arbitrary image sets with good effect.

Table 1. FID calculations for various GAN implementations.

GAN	FID
DCGAN	64.446
LSGAN	149.210
StyleGAN	44.704

We calculated the FIDs comparing generated images from each GAN implementation with the original data set. All implementations were trained on 150 epochs. The results are summarized in Table 1; note lower FID scores are better. StyleGAN outperformed DCGAN, while LSGAN seriously underperformed. These observations corroborate our qualitative assessments.

8. Conclusion and Future Work

In this work, we explored the artificial creation of the anime characters using GANs. By extracting a clean data set and introducing several practical training strategies, we showed that DCGANs can produce convincing fakes. We then implemented StyleGAN, a major improvement on DCGAN both qualitatively and by FID score. Additionally, StyleGAN can be trivially used to mix styles. This work has demonstrated that GANs can be readily applied to the anime industry, allowing for streamlined design of new characters.

With the power of style mixing, many possibilities exist for future developments. Assuming a data set with labels, the model can learn to generate specific features on-demand, such as hair color or pose. Other directions are to improve the final resolution of generated images, or to generate full-body designs.

9. Acknowledgement

We acknowledge NVIDIA for their open-source StyleGAN implementation (Karras et al., 2019), Google for the Inception V3 model (Google, 2015), and mseitzer for the FID calculation code(Mseitzer, 2020). We would also like to thank Eric Eaton, Dinesh Jayaraman, and all the TAs for putting together an exciting class.

Our code can be found at the following GitHub link: https://github.com/roynwu/Artificial-Anime-Character-Design/

References

Branwen, Gwern. Making anime faces with stylegan. 2019. URL https://www.gwern.net/Faces.

Chao, Brian. Anime-face-dataset, 2019. URL https://github.com/Mckinsey666/Anime-Face-Dataset.

Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial networks, 2014.

Google. Tensorflow hub, 2015. URL http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz.

Heusel, Martin, Ramsauer, Hubert, Unterthiner, Thomas, Nessler, Bernhard, Klambauer, Günter, and Hochreiter, Sepp. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017. URL http://arxiv.org/abs/1706.08500.

Jayleicn. animegan, 2017. URL https://github.com/jayleicn/animeGAN.

Jin, Yanghua, Zhang, Jiakai, Li, Minjun, Tian, Yingtao, Zhu, Huachun, and Fang, Zhihao. Towards the automatic anime characters creation with generative adversarial networks. *CoRR*, abs/1708.05509, 2017. URL http://arxiv.org/abs/1708.05509.

Karras, Tero, Aila, Timo, Laine, Samuli, and Lehtinen, Jaakko. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017. URL http://arxiv.org/abs/1710.10196.

Karras, Tero, Laine, Samuli, and Aila, Timo. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018. URL http://arxiv.org/abs/1812.04948.

Karras, Tero, Laine, Samuli, Aittala, Miika, Hellsten, Janne, Lehtinen, Jaakko, and Aila, Timo. Analyzing and improving the image quality of stylegan, 2019.

LeCun, Yann. What are some recent and potentially upcoming breakthroughs in deep learning?, 2016. URL https://www.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-deep-learning/answer/Yann-LeCun.

Mao, Xudong, Li, Qing, Xie, Haoran, Lau, Raymond Y. K., Wang, Zhen, and Smolley, Stephen Paul. Least squares generative adversarial networks, 2016.

Mattya. chainer-dcgan, 2016. URL https://github.com/mattya/chainer-DCGAN.

Mseitzer. Pytorch-fid, 2020. URL https://github.com/mseitzer/pytorch-fid.

Nagadomi. A face detector for anime/manga using opencv, 2011. URL https://github.com/nagadomi/lbpcascade_animeface.

Radford, Alec, Metz, Luke, and Chintala, Soumith. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

Rakshit, Soumik. Anime faces, 2019. URL https://www.kaggle.com/soumikrakshit/anime-faces.

Salimans, Tim, Goodfellow, Ian J., Zaremba, Wojciech, Cheung, Vicki, Radford, Alec, and Chen, Xi. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016. URL http://arxiv.org/abs/1606.03498.

Wang, Philip. Thispersondoesnotexist, 2019. URL https://www.thispersondoesnotexist.com/.