GANime Faces - An Undergraduate Study in Generating Anime Faces from an Adversarial Network

Ivana Jovasevic
Engineering Science
Simon Fraser University
Burnaby, BC
ijovasev@sfu.ca

Ryan Lui
Engineering Science
Simon Fraser University
Burnaby, BC
rclui@sfu.ca

Sami Ma
Computing Science
Simon Fraser University
Burnaby, BC
masamim@sfu.ca

Adnan Syed
Engineering Science
Simon Fraser University
Burnaby, BC
ssyed@sfu.ca

Greyson Wang
Engineering Science
Simon Fraser University
Burnaby, BC
greysonw@sfu.ca

Abstract

This paper explores the use of Generative Adversarial Networks (GANs) to create anime character faces. The research done is based on the Keras-GAN-AnimeFace-Character repository by forcecore [1]. After running the default model, we changed various hyperparameters to see its effects on image generation, including batch size, learning rates, and types of optimizers. We also ran the animeface-character-dataset through different models to observe the output.

1 Introduction

Machine learning is a vast, constantly growing research area with many different models and approaches for solving problems. In this class we have learned about discriminative models, which are used to model the decision boundary in a classification problem, and generative models, which are used to model the actual distribution of a class, and can generate synthetic example data (ch4 slides p73). The purpose of this paper is to investigate the use of both of these models in a Generative Adversarial Network (GAN).

GANs were first proposed by Ian Goodfellow et al. as "a new framework for estimating generative models via an adversarial process" [2]. The network is comprised of a discriminative model that is trained to classify between 'real' training data and 'fake' generated data, and a generative model that is trained to generate the data to trick the discriminative model. This method offers a solution to the problem of poorly estimated generative models. By forcing the networks to compete, the generative model is driven to improve its output until the 'fake' generated data is indistinguishable from the 'real' training data [2]. Deep Convolutional GANs in particular have been proven to be one of the most successful network designs, utilizing convolutional layers from convolutional neural networks (CNN) that have proved successful in computer vision [3].

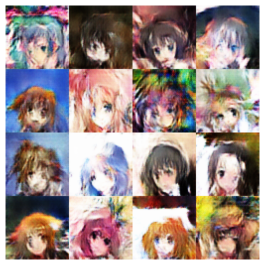
In this paper, we will explore the use of a GAN to generate anime character faces. The inspiration for this topic stems from a personal interest in anime, and the availability of the data to train the model. Our paper is based on the model developed by forcecore [1] and was trained with data from animeface-character-dataset [4]. In this paper, we explore the outputs from the Keras-

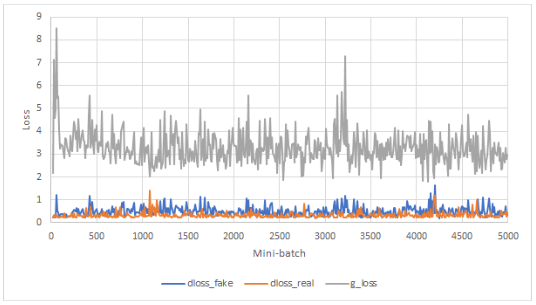
GAN-AnimeFace-Character model along with other real face GAN models, and discuss a variety of experiments we performed on these models in an attempt to improve the results.

2 Approach

The Keras-GAN-AnimeFace-Character model is a GAN developed using Keras [1]. The model is trained on a batch size of 64 using 64 by 64 pixel images. Loss is calculated using the Adam optimizer. Figure 1a shows some example training images from the animeface-character-dataset, and Figure 1b and 1c show the output of the unmodified model and its loss function, respectively.

(a) Training Images


(b) Output at batch 5000



(c) Loss function of discriminator and generator

Figure 1: Results from running the default Keras-GAN-AnimeFace-Character model.

Our approach for this paper is to perform a set of experiments on the default model and observe the outputs in hopes of improving the results. The attempted experiments are listed below, and discussed in further detail in the Experiments section of the paper:

1. Making simple changes to the default model, such as modifying batch size and flipping label assignments.
2. Using a variety of strategies in an attempt to optimize the code, including modifying the learning rates, using the Adam and AdaMax optimizers, and applying Stochastic Gradient Descent (SGD).
3. Running different models on the animeface-character-dataset to compare results to the original model.

3 Experiments

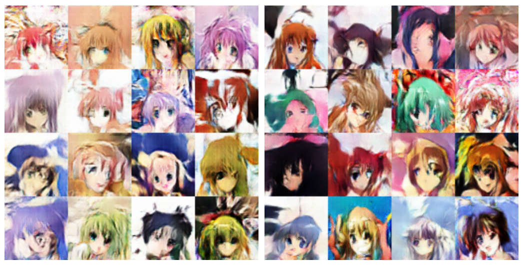
3.1 Experiment: changing batch size and flipping labels

After running the Keras-GAN-AnimeFace-Character model unmodified, our first experiment was to make minor changes based on researched tips for training GANs [5]. We ran the model using different batch sizes (16, 32, 64 and 128), to evaluate the significance of how many epochs were necessary to train the model. The results from the smallest and largest batch size run can be seen in Figure 2 below.

By default, the model ran with a batch size of 64. By modifying the batch size, we observed that a smaller batch size of 16 ran much faster, and the resulting images appear to be about as good. On the contrary, running the model with a larger batch size of 128 failed at around 60000 mini-batches, as can be seen in Figure 3, and the resulting images do not appear to have improved. d_loss0 indicates the loss of the discriminator when trained with fake images, and d_loss1 is the loss when trained with real images.

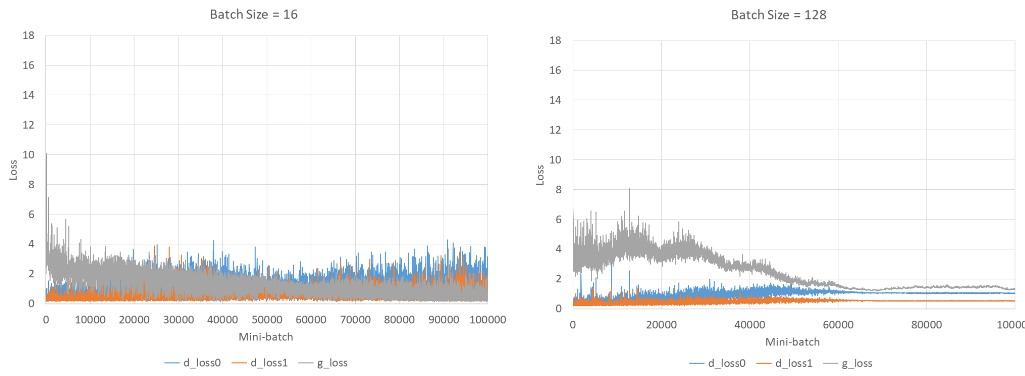
An additional modification we made was flipping the labels for generated and real images, based on the idea that this would help with gradient flow in early iterations of training [5].

The default labels are soft label where the real would be some number close to 0 and the fake would be some number close to 1. We flipped the labels such that the real images were close to 1 and fake



(a) Batch size 16 (b) Batch size 128

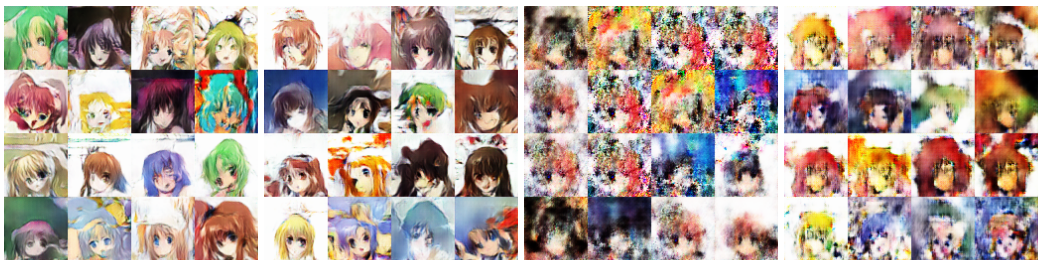
Figure 2: Generated images from running the model with various batch sizes



(a) Batch size 16 (b) Batch size 128

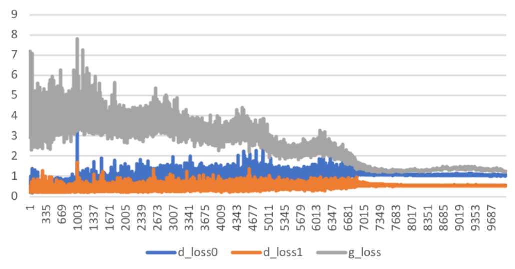
Figure 3: Loss graphs from running the model with various batch sizes

images were close to 0. The idea was that it might help the cases where we might get stuck at a local minimum. With this in mind, we ran the GAN and took a look at the results. The resulting pictures from flipping the labels looked cleaner and more of a face than the default settings. 30,000 was the best of both, but the flipped labels images had eyes that were easily distinguishable. At 100,000, they both looked bad and the images were duplicating because of the overfitting. Even at 100,000, the flipped labels had less noise, less duplicate pictures and still somewhat resembled a face. It seemed like an initial success; however, taking a looking at the loss graph between the two, this was not the case. Overall, the loss of the flipped labels were higher and less stable than the default. We did not get a chance to examine why this was the case; however, something we could have tried is flipping the labels occasionally, rather than all the time, to see if that can negate the effects of the poor loss while retaining the benefits of a cleaner image.

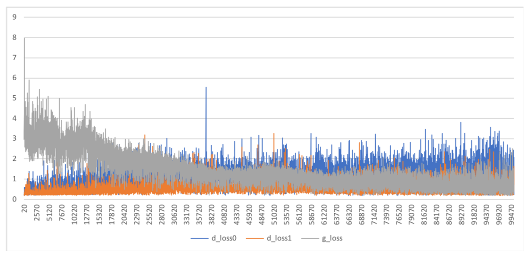


(a) Default at 30,000 (b) Flipped at 30,000 (c) Default at 100,000 (d) Flipped at 100,000

Figure 4: Generated images at various mini-batches with and without flipped classifiers



(a) Default model

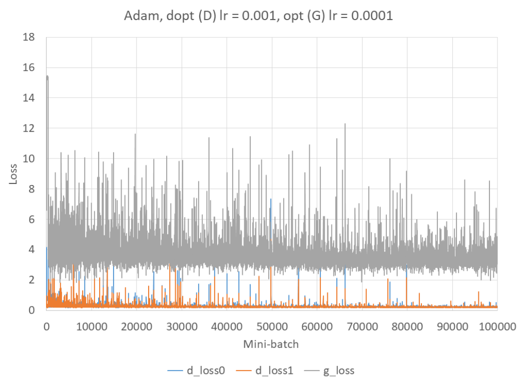


(b) Flipped labels model

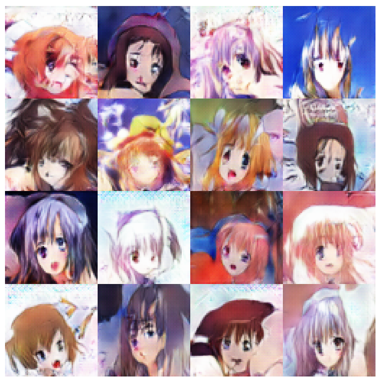
Figure 5: Loss graphs from running the model with and without flipped classifiers

3.2 Experiment: changing optimizers and their learning rates

Some of the other modifications we made included using different optimizers, as suggested by Soumith et al [6]. In the default model run, we noticed that the learning began to fail after 70,000 mini-batches, with the generator beginning to generate extremely noisy images. Adam was the default optimizer that was used, and as such, we first opted to try different values for the discriminator learning rate. In our first attempt, we made the discriminator learning rate larger, from 0.0002 to 0.001, while keeping all other parameters constant. This change eliminated the failure we saw. We believe that the having the learning rate too small for the discriminator made it easier for the generator to "outsmart" the discriminator. Figure 6 outlines the results.



(a) Loss

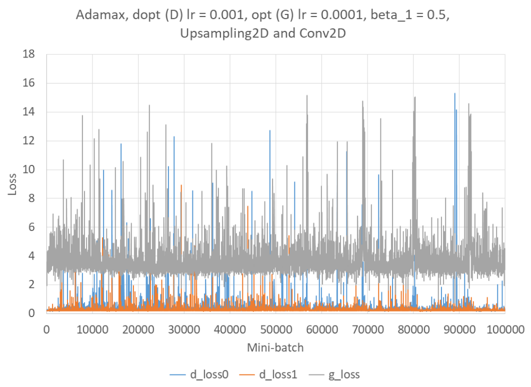


(b) Generated Image

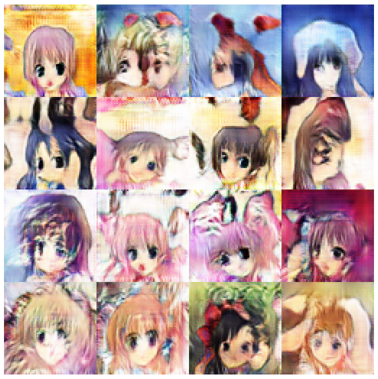
Figure 6: Results from running the model using the Adam optimizer

Next, we modified the generator layers to use 2D upsampling followed by 2D convolution in place of the current 2D transposed convolution from the original model. 2D transposed convolution was noted to exhibit artifacts in the images [7] so by forcing upsampling before convolution, we hoped that this would reduce the noise in the generator at 70,000 mini-batches. This proved to be true, and the losses for both generator and discriminator continually fluctuated.

We also tried using AdaMax and standard stochastic gradient descent, both of which did not succeed. In the first case, we modified the GAN to use the AdaMax optimizer with various values for the learning rate, but the resulting generated images had noticeably more artifacts. This was reflected in the loss graph as shown in Figure 7. In the stochastic gradient descent case, we opted to change the discriminator to SGD while keeping the generator on the Adam optimizer. This led to an early failure, where the generator loss was lower than that of the discriminator, and it was reflected with garbage images.



(a) Loss



(b) Generated Image

Figure 7: Results from running the model using the AdaMax optimizer

3.3 Experiment: changing the model - BEGAN

Our final set of experiments were to change the model we were using, and observe the output for the same animeface-character-dataset.

We considered that anime faces are similar to real faces, and there would be more work done on real face GANs, so we decided to search for repositories on them. One repository is a TensorFlow implementation based on a paper about Boundary Equilibrium Generative Adversarial Networks (BEGAN) [8]. It focused on balancing the generator and discriminator and did not pretrain the discriminator [9]. Using the same 14,000 image dataset (animeface-character-dataset), the images had to be resized to 128x128 with black bar fillings in order to feed them into the network. Preliminary training using the default parameters of the repository led to seemingly overfitting from the generator side. The generator loss was continuously lower than the discriminator loss, and the generated images had repeated faces. The training progressively got worse, where it was stopped at 199,500 mini-batches (Batch size 16) with the result shown in Figure 8.

The faces generated looked like an anime face, but it is hard to say they are good. In addition, the overfitting prevents further training. We noticed that the fake and real images that the discriminator uses for training are both blurred, probably to keep the discriminator from overpowering the generator too much. The overall problems could be that the parameters need tuning or that the entire architecture of balancing the networks works against itself.

To prevent overfitting, we tried changing the learning rates (LR) of the networks. The original learning rates were both 8e-5. We lowered the generator LR to 1e-6 and the discriminator LR to 4e-5 in the hopes that the discriminator would learn faster than the generator. The batch size was also lowered to 4 to speed up training. Lowering the LRs in this way ended in failure as the generated faces started with something that could be faces to just blobs, as seen in Figure 9

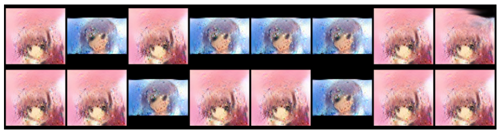


Figure 8: Image at batch 199,500

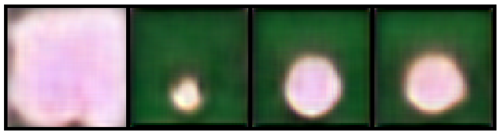
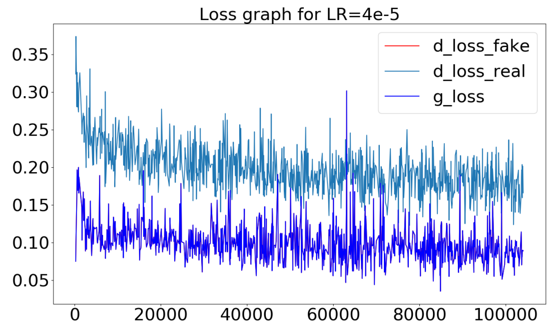


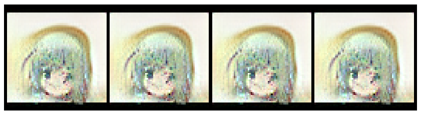
Figure 9: Image at batch 69,000

The generator loss was still constantly below the discriminator, making the generator still overpowering and overfitting the discriminator.

The next experiment was to keep the LRs the same but slightly lower at 4e-5 as suggested by one of the repository issues [10]. The results may have been slightly better, but the same overfitting problem occured. The generator loss is still below the discriminator loss as seen in Figure 10.



(a) Loss



(b) Generated Image

Figure 10: Results from running the BEGAN model with LR = 4e-5

More time would need to be allocated to investigate the overfitting problem. This GAN has the potential to create good anime faces. The problem is that the main part of BEGAN is balancing the two networks, which could work against it when one network, namely the generator, is already incredibly overpowering the other.

3.4 Experiment: Regularization and limiting training

In this section, we expand on the experiments done with Deep Convolutional Generative Adversarial Networks by Taehoon Kim [11], who modified the original concept by updating the generator network twice for every discriminator update in order to avoid fast convergence. While this initially produced good results after 50 epochs, by 100 epochs the generator was producing much worse images and the loss value of the discriminator rapidly converged towards zero.

Our first approach was to stop training the discriminator network when its loss value is much smaller than the generator loss, with the goal of allowing the generator to catch up. We found that stopping training when the ratio of the loss is greater than 10x produced results that were significantly better until 200 epochs. A difference much smaller and much larger than 10x caused the generator to simply produce mostly identical images, as shown in Figure 11 below.

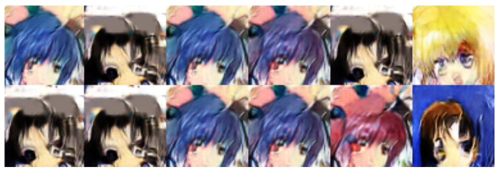


Figure 11: Identical images from DCGAN

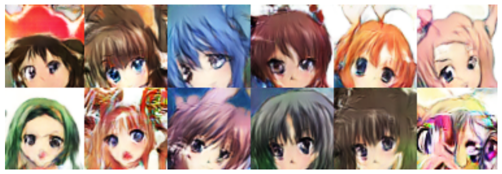


Figure 12: Best output from DCGAN

Combining the original method of updating the generator twice and our method produced comparable results until 160 epoches, but it unfortunately caused the generator to produce mainly identical images afterwards.

Applying L2 regularization had no noticeable effect on the results when applied to the original implementation of the DCGAN. However, when applied to our modified DCGAN, this also caused the generator to continuously produce mainly identical images.

4 Conclusion

The purpose of this paper was to research the use of GANs to generate anime character faces. By basing our findings off of a pre-existing model, we were able to observe which modifications and optimizations we could make to improve the generated anime images. We investigated further by running the same dataset through Real Face GAN models, and were able to produce some definitive character faces using these. This project allowed us to become familiar with the inner workings of Generative Adversarial Networks, and explore their capabilities for image generation. Applications of such networks could potentially be in manga drawing, such as lessening the workload of artists, or providing a starting base in the form of ”idea generation”.

5 Contributions

1. Adnan Syed was in charge of researching various tips and tricks to improve how GANs run, and these were implemented and documented by himself and Ivana Jovasevic.
2. Ryan Lui was in charge of running various optimizations on the Keras model and observing the changes.
3. Greyson Wang and Sami Ma ran the animeface-character-dataset through Real Face GAN models and researched these results in comparison to the Keras-GAN-AnimeFace-Character model.
4. Ivana Jovasevic and Ryan Lui were responsible for formatting and finalizing this paper.

References

[1] forcecore. forcecore/keras-gan-animeface-character: Gan example for keras. `https://github.com/forcecore/Keras-GAN-Animeface-Character`, 2017.

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. `https://arxiv.org/abs/1406.2661`, 2014.

[3] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. `https://arxiv.org/abs/1511.06434`, 2015.

[4] C. Nagadomi. animeface-character-dataset. `http://www.nurs.or.jp/~nagadomi/animeface-character-dataset/`, 2013.

[5] Utkarsh Desai. Keep calm and train a gan. pitfalls and tips on training generative adversarial networks. `https://medium.com/@utk.is.here/edd529764aa9`, 2018.

[6] Soumith Chintala, Emily Denton, Martin Arjovsky, and Michael Mathieu. soumith/ganhacks: How to train a gan? `https://github.com/soumith/ganhacks`, 2016.

[7] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. `https://distill.pub/2016/deconv-checkerboard/`, 2016.

[8] Taehoon Kim. carpedm20/began-tensorflow: Tensorflow implementation of began. `https://github.com/carpedm20/BEGAN-tensorflow`, 2017.

[9] David Berthelot, Thomas Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks. `https://arxiv.org/abs/1703.10717`, 2017.

[10] ProgramItUp. carpedm20/began-tensorflow: Unable to reproduce results: Generator output has only two face types. `https://github.com/carpedm20/BEGAN-tensorflow/issues/30`, 2017.

[11] Taehoon Kim. carpedm20/dcgan-tensorflow: A tensorflow implementation of dcgan. `https://github.com/carpedm20/DCGAN-tensorflow`, 2017.