VAE and GANs in Generating Anime Faces

Jingyu Yao, Yuran Zhu
CPSC 553 Final Project
Yale University
jingyu.yao@yale.edu, yuran.zhu@yale.edu

December 5, 2021

Abstract

In this project, we discover the abilities of variational autoencoders(VAE) and two different kinds of generative adversarial network(GAN) to experiment on their abilities to generate anime faces similar to realistic artistic anime drawings. Our neural networks(NNs) are based on PS3 but builds further by adding convolutional layers to enhance the NNs' generative capabilities. We examine the results produced by VAE and GAN and conclude that their generative capabilities and characteristics remain unchanged to what we have seen during our MNIST experiment. While VAE allows us to generate anime faces that generally resembles real drawings, its results are blurry and could not yield sharp images. On the other hand, both GAN and WGAN allows us to generate sharper images that looks similar to real drawings, but are very sensitive to parameter tuning. We have outlined a few points for GAN parameter tuning that we have not noticed during out MNIST experiment.

1 Introduction

A various degree of computer vision and machine learning techniques use generative models as a necessary component nowadays. Anime characters are popular among younger generations and we all are tempted to create our custom "waifus"; however, it requires significant amounts of artistic expertise to make successful anime drawings. To bridge this gap, we will harvest the power of generative models to automatically generate anime characters. This offers an opportunity to create custom characters without professional art skills. Besides the benefits for anime lovers, even professional creators might take advantage of automatic anime face generation for inspirations on new character design. In this project, we aim to take a level up from MNIST in PS3 and try to generate anime faces using VAE and GAN. We build upon code in PS3 by incorporating convolutional layers in the models to allow the generative models to handle images with more channels and pixels. Our dataset is sourced from https://www.kaggle.com/splcher/animefacedataset, which contains over 60

thousand real anime face images. Our final goal is to create fake anime faces drawings that are as realistic, defined as being sharp and undistorted, as possible to real drawings.

2 Background

Variational Autoencoder(VAE) and Generative Adversarial Network(GAN) are generative models that are able to generate new content. These models have become really popular in the machine learning community due to its interesting applications such as generating synthetic training data, creating arts, image-to-image translation, etc. In the academic scene, most generative model researches begins with the MNIST dataset, such as [1] and [3]. Usually these studies would continue to test the limits of their models on a more complicated MNIST-like dataset called the fashion MNIST[4], which is definitely more challenging and offers insights on how models like VAE and GAN would perform in a real world. Our projects similarly intend to experiment VAE and GAN on real-life datasets, using anime faces as our subject here. Similar work as been performed that shows convolutional GAN can produce high-resolution anime faces almost indistinguishable to real drawings[2]. There are no convolutional VAE researches on anime face datasets yet, but we aim to experiment with it as well as a learning experience.

3 Methods and Results

3.1 Preprocessing

We included 63,632 images and resized them to the same size of 64 by 64 pixels for each image. For all our anime face neural networks, we used a input size of (3, 64, 64), in which there are 3 channels, and each channel has pixel size of 64 by 64. For the Mnist dataset, the images are reshaped to (1, 64, 64).

3.2 VAE on Anime Faces

3.2.1 Model

We improved on the VAE in PS3 by incorporating convolutional layers in the VAE encoder and decoder. Our model can be seen in the summary below.

Batch size = 32
Optimizer = Adam, learning rate = 0.001
VAE loss = binary cross entropy + KL divergence

```
VAE:
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
```

```
            Conv2d-1          [-1, 32, 31, 31]           1,568
              ReLU-2          [-1, 32, 31, 31]               0
            Conv2d-3          [-1, 64, 14, 14]          32,832
              ReLU-4          [-1, 64, 14, 14]               0
            Conv2d-5           [-1, 128, 6, 6]         131,200
              ReLU-6           [-1, 128, 6, 6]               0
            Conv2d-7           [-1, 256, 2, 2]         524,544
              ReLU-8           [-1, 256, 2, 2]               0
           Flatten-9                [-1, 1024]               0
      Linear(fc1)-10                   [-1, 32]          32,800
      Linear(fc2)-11                   [-1, 32]          32,800
      Linear(fc3)-12                [-1, 1024]          33,792
        UnFlatten-13          [-1, 1024, 1, 1]               0
  ConvTranspose2d-14           [-1, 128, 5, 5]       3,276,928
             ReLU-15           [-1, 128, 5, 5]               0
  ConvTranspose2d-16          [-1, 64, 13, 13]         204,864
             ReLU-17          [-1, 64, 13, 13]               0
  ConvTranspose2d-18          [-1, 32, 30, 30]          73,760
             ReLU-19          [-1, 32, 30, 30]               0
  ConvTranspose2d-20           [-1, 3, 64, 64]           3,459
          Sigmoid-21           [-1, 3, 64, 64]               0
================================================================
Total params: 4,348,547
Trainable params: 4,348,547
Non-trainable params: 0
----------------------------------------------------------------
```

We use fc1 and fc2 for reparameterization and fc3 for decoding. Our model follows a convolutional model of 3-32-64-128-256 channels for encoding and 1024-128-64-32-3 channels for decoding. Our latent dimensions is 32. VAE usually use a smaller latent dimension, but for an anime image the information of input is too hard to pass through a small bottleneck. This results in a mean and blurry output. We hope that by using a bigger latent vector, VAE can get higher reconstruction quality.

3.2.2 Results

The VAE results are blurry, but does show an the network's effort in trying to understand the images. We show two sample batches of decoded images using real images as input.

According to [2], these reconstructions are pretty similar to the researchers' "mean images". For our loss function, we find out that the reconstruction loss and the KL loss need to work together for VAE to produce good results. We experimented with MSE and BCE, and found only BCE in producing a successful result.

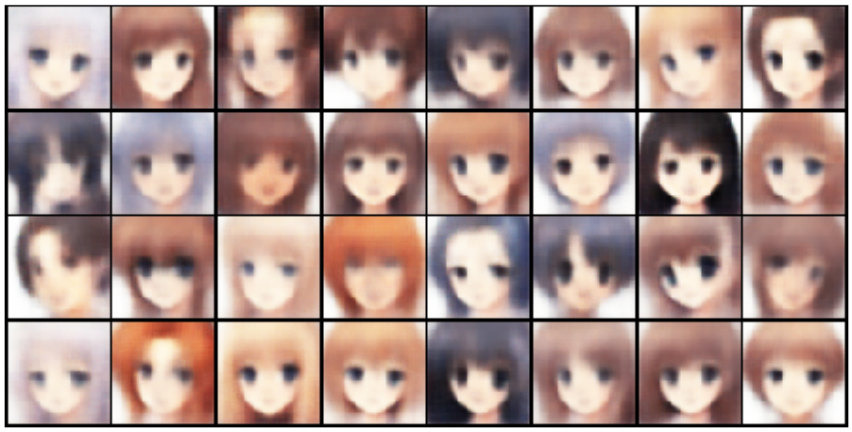

Figure 1: VAE Sample Output Batch 1

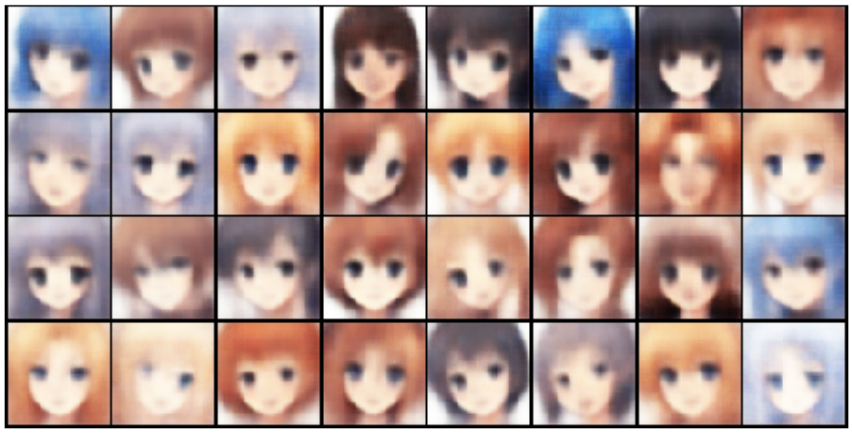

Figure 2: VAE Sample Output Batch 2

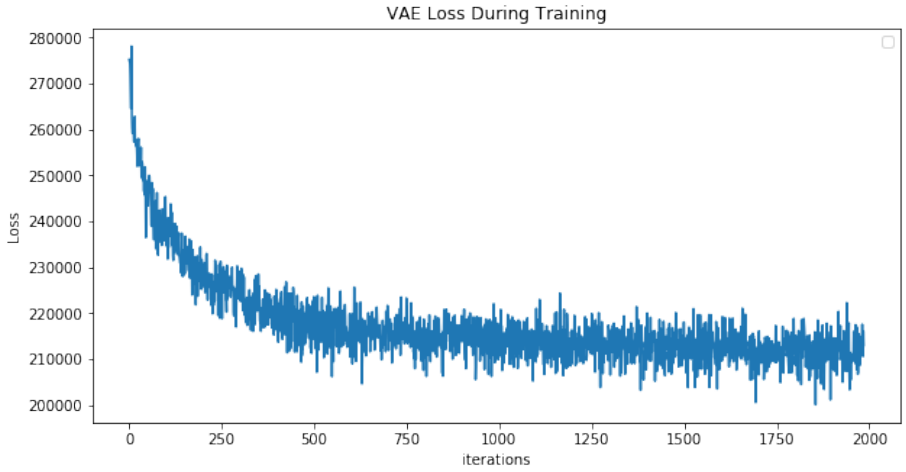

Figure 3: VAE Training Loss

3.3 DCGAN on MNIST and Anime Faces

According to [2], Deep Convolutional Generative Adversarial Networks(DCGAN), shows many promise in generating anime faces. We aim to create a deep convolutional GAN. We first experimented with MNIST on the DCGAN to ensure its feature dimensions could produce a reasonable reconstruction, and then proceeded with the anime face dataset. We will illustrate each of our results here.

3.3.1 MNIST DCGAN Model

Batch size = 128
Generator Optimizer = Adam, learning rate = 0.0002, beta = (0.5, 0.99)
Discriminator Optimizer = Adam, learning rate = 0.0002, beta = (0.5, 0.99)
GAN loss = binary cross entropy

We used an generator input size of (100, 1, 1), followed by convolational layers of size 256-128-64-32-1, with discriminator being the vice versa. Each con2d layer is followed by a batch normalization layer which is essential for 2d-rgb images. Different from PS3, we used Relu instead of leaky Relu for the generator. Although [2] says a leaky Relu in both the generator and discriminator will increase performance, we have not found much success in that. We used a kernel size of (4, 4) and stride of (2, 2) across most convolutional layers.

```
Generator:
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
```

```
   ConvTranspose2d-1            [-1, 256, 4, 4]         409,600
       BatchNorm2d-2            [-1, 256, 4, 4]             512
              ReLU-3            [-1, 256, 4, 4]               0
   ConvTranspose2d-4            [-1, 128, 8, 8]         524,288
       BatchNorm2d-5            [-1, 128, 8, 8]             256
              ReLU-6            [-1, 128, 8, 8]               0
   ConvTranspose2d-7           [-1, 64, 16, 16]         131,072
       BatchNorm2d-8           [-1, 64, 16, 16]             128
              ReLU-9           [-1, 64, 16, 16]               0
  ConvTranspose2d-10           [-1, 32, 32, 32]          32,768
      BatchNorm2d-11           [-1, 32, 32, 32]              64
             ReLU-12           [-1, 32, 32, 32]               0
  ConvTranspose2d-13            [-1, 1, 64, 64]             512
             Tanh-14            [-1, 1, 64, 64]               0
================================================================
Total params: 1,099,200
Trainable params: 1,099,200
Non-trainable params: 0
----------------------------------------------------------------

Discriminator:
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 32, 32]             512
         LeakyReLU-2           [-1, 32, 32, 32]               0
            Conv2d-3           [-1, 64, 16, 16]          32,768
       BatchNorm2d-4           [-1, 64, 16, 16]             128
         LeakyReLU-5           [-1, 64, 16, 16]               0
            Conv2d-6            [-1, 128, 8, 8]         131,072
       BatchNorm2d-7            [-1, 128, 8, 8]             256
         LeakyReLU-8            [-1, 128, 8, 8]               0
            Conv2d-9            [-1, 256, 4, 4]         524,288
      BatchNorm2d-10            [-1, 256, 4, 4]             512
        LeakyReLU-11            [-1, 256, 4, 4]               0
           Conv2d-12              [-1, 1, 1, 1]           4,096
          Sigmoid-13              [-1, 1, 1, 1]               0
================================================================
Total params: 693,632
Trainable params: 693,632
Non-trainable params: 0
----------------------------------------------------------------
```

3.3.2 MNIST DCGAN Results

DCGAN trained on MNist better than our GAN in PS3, which uses linear layers. The advantages are shown at that there are less noise produced in the fake images, and looks basically the same as real images.

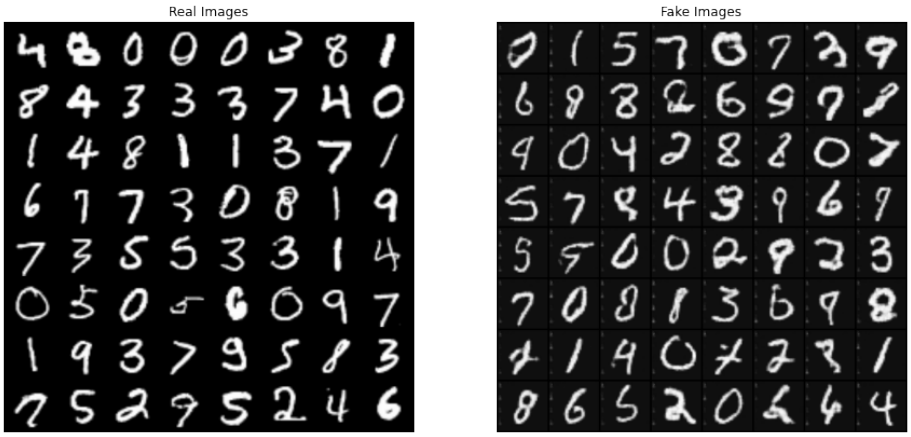

Figure 4: DCGAN Mnist Results

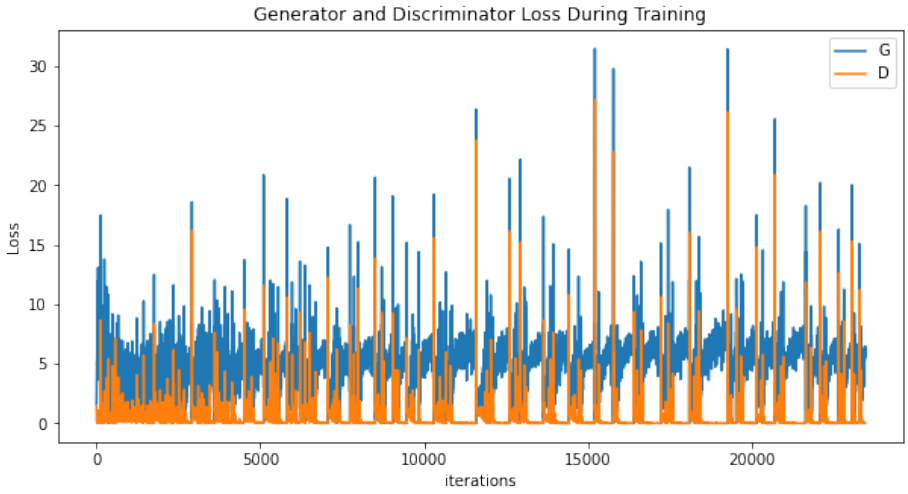

Figure 5: DCGAN Mnist Training Loss

Our results show that our DCGAN model is generally capable of image generation. We modified this model to accept three channels to create our DCGAN model for anime face generation.

3.3.3 Anime Face DCGAN Model

Batch size = 128
Generator Optimizer = Adam, learning rate = 0.0002, beta = (0.5, 0.99)
Discriminator Optimizer = Adam, learning rate = 0.0002, beta = (0.5, 0.99)
GAN loss = binary cross entropy

Our anime face DCGAN model builds on the Mnist GAN by using 3 channels and using larger feature maps. We used an generator input size of (100, 1, 1), followed by convolutional layers of size 512-256-128-64-3, with discriminator being the vice versa. The rest of the configurations remains the same as the Mnist model. We used a kernel size of (4, 4) and stride of (2, 2) across most convolutional layers.

```
Generator:
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1            [-1, 512, 4, 4]         819,200
       BatchNorm2d-2            [-1, 512, 4, 4]           1,024
              ReLU-3            [-1, 512, 4, 4]               0
   ConvTranspose2d-4            [-1, 256, 8, 8]       2,097,152
       BatchNorm2d-5            [-1, 256, 8, 8]             512
              ReLU-6            [-1, 256, 8, 8]               0
   ConvTranspose2d-7          [-1, 128, 16, 16]         524,288
       BatchNorm2d-8          [-1, 128, 16, 16]             256
              ReLU-9          [-1, 128, 16, 16]               0
  ConvTranspose2d-10           [-1, 64, 32, 32]         131,072
      BatchNorm2d-11           [-1, 64, 32, 32]             128
             ReLU-12           [-1, 64, 32, 32]               0
  ConvTranspose2d-13            [-1, 3, 64, 64]           3,072
             Tanh-14            [-1, 3, 64, 64]               0
================================================================
Total params: 3,576,704
Trainable params: 3,576,704
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 3.00
Params size (MB): 13.64
Estimated Total Size (MB): 16.64
----------------------------------------------------------------

Discriminator:
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
```

```
         Conv2d-1          [-1, 64, 32, 32]           3,072
    LeakyReLU-2            [-1, 64, 32, 32]               0
         Conv2d-3          [-1, 128, 16, 16]         131,072
    BatchNorm2d-4          [-1, 128, 16, 16]             256
    LeakyReLU-5            [-1, 128, 16, 16]               0
         Conv2d-6            [-1, 256, 8, 8]         524,288
    BatchNorm2d-7            [-1, 256, 8, 8]             512
    LeakyReLU-8              [-1, 256, 8, 8]               0
         Conv2d-9            [-1, 512, 4, 4]       2,097,152
    BatchNorm2d-10           [-1, 512, 4, 4]           1,024
    LeakyReLU-11             [-1, 512, 4, 4]               0
         Conv2d-12             [-1, 1, 1, 1]           8,192
        Sigmoid-13             [-1, 1, 1, 1]               0
================================================================
Total params: 2,765,568
Trainable params: 2,765,568
Non-trainable params: 0
----------------------------------------------------------------
```

3.3.4 Anime Face DCGAN Results

The DCGAN was able to produce much sharper images than those for VAE. A sample batch of the trained generator using random uniform noise can be seen below.
We also show a few samples of the generator training in progress here.

Figure 6: DCGAN Anime Face Final Result

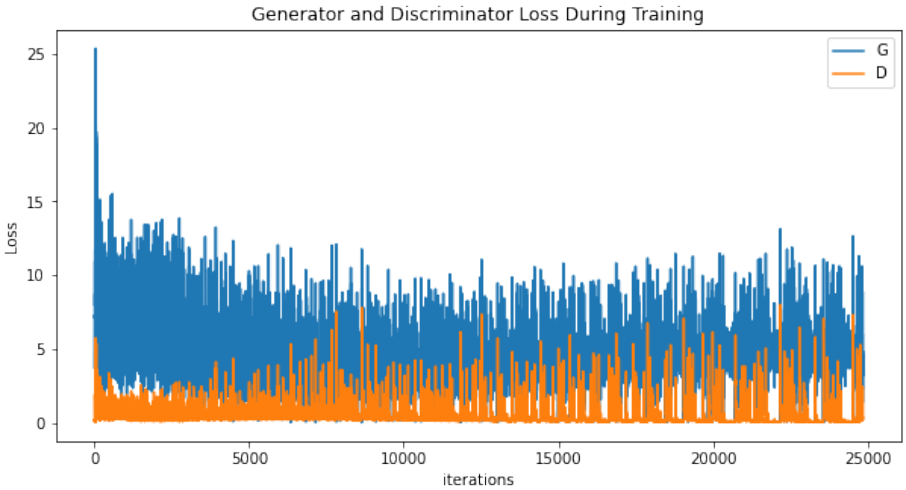

Figure 7: DCGAN Anime Face Training Loss

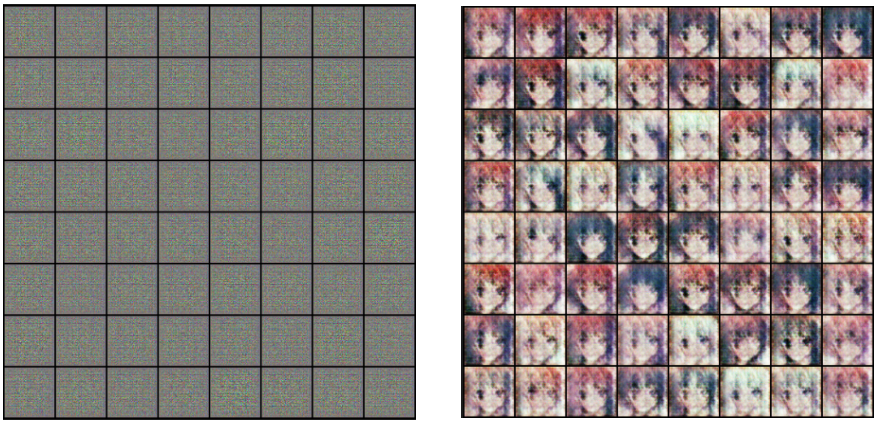

Figure 8: DCGAN Anime Face 0(left), 250(right) Batches Trained

Figure 9: DCGAN Anime Face 1000(left), 2000(right) Batches Trained

Our results show that the DCGAN model is successful in creating images quite similar to real drawings that are sharp and realistic. We do notice some image distortions among the final results. Distortion is actually one of the major issues we were dealing with when trying to craft the model. We will illustrate on this in the next subsection.
From this experiment we conclude that a DCGAN model with a sufficiently large number of feature map can produce realistic anime faces.

3.3.5 Other DCGAN Model Used

Our previous model used about exactly the same GAN structure as our PS3 GAN with leaky Relu in both the generator and the discriminator, and included dropout layers in the discriminator after each relu activation layer. The results shows a large amount of distortion in the images produced by this DCGAN model. A sample output is shown below. Compared to our best DCGAN model, it uses a smaller number of feature maps.
We have learned from these experiments that GANs are very sensitive to parameters and the layers used, and careful parameter tuning is required for a GAN to work as intended.

```
Generator:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 4096)              413696
batch_normalization (BatchNo (None, 4096)              16384
leaky_re_lu (LeakyReLU)      (None, 4096)              0
reshape (Reshape)            (None, 4, 4, 256)         0
conv2d_transpose (Conv2DTran (None, 8, 8, 128)         524416
batch_normalization_1 (Batch (None, 8, 8, 128)         512
```

```
leaky_re_lu_1 (LeakyReLU)    (None, 8, 8, 128)         0
conv2d_transpose_1 (Conv2DTr (None, 16, 16, 64)        131136
batch_normalization_2 (Batch (None, 16, 16, 64)        256
leaky_re_lu_2 (LeakyReLU)    (None, 16, 16, 64)        0
conv2d_transpose_2 (Conv2DTr (None, 32, 32, 32)        32800
batch_normalization_3 (Batch (None, 32, 32, 32)        128
leaky_re_lu_3 (LeakyReLU)    (None, 32, 32, 32)        0
conv2d_transpose_3 (Conv2DTr (None, 64, 64, 3)         1539
activation (Activation)      (None, 64, 64, 3)         0
=================================================================
Total params: 1,120,867
Trainable params: 1,112,227
Non-trainable params: 8,640
_________________________________________________________________

Discriminator:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
leaky_re_lu_4 (LeakyReLU)    (None, 32, 32, 32)        0
dropout (Dropout)            (None, 32, 32, 32)        0
conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496
zero_padding2d (ZeroPadding2 (None, 17, 17, 64)        0
batch_normalization_4 (Batch (None, 17, 17, 64)        256
leaky_re_lu_5 (LeakyReLU)    (None, 17, 17, 64)        0
dropout_1 (Dropout)          (None, 17, 17, 64)        0
conv2d_2 (Conv2D)            (None, 9, 9, 128)         73856
batch_normalization_5 (Batch (None, 9, 9, 128)         512
leaky_re_lu_6 (LeakyReLU)    (None, 9, 9, 128)         0
dropout_2 (Dropout)          (None, 9, 9, 128)         0
conv2d_3 (Conv2D)            (None, 9, 9, 256)         295168
batch_normalization_6 (Batch (None, 9, 9, 256)         1024
leaky_re_lu_7 (LeakyReLU)    (None, 9, 9, 256)         0
dropout_3 (Dropout)          (None, 9, 9, 256)         0
flatten (Flatten)            (None, 20736)             0
dense_1 (Dense)              (None, 1)                 20737
=================================================================
Total params: 410,945
Trainable params: 410,049
Non-trainable params: 896
_________________________________________________________________
```

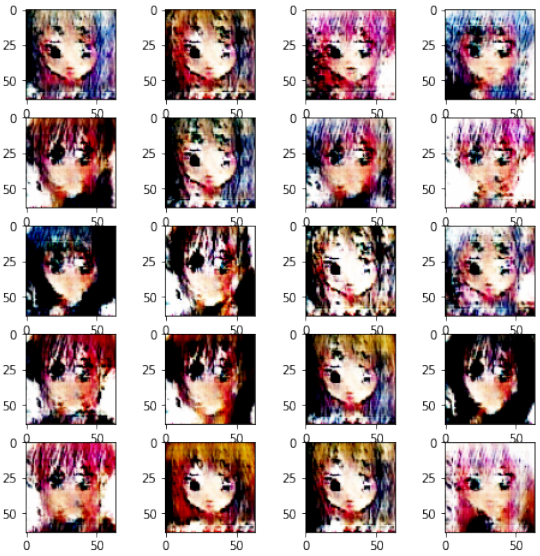

Figure 10: Distorted DCGAN Anime Face Result

3.4 WGAN on Anime Faces

3.4.1 WGAN on Anime Faces Model

Our final experiment was trying to work with the anime dataset using WGAN. The core difference of WGAN and GAN is the lost function, in which WGAN encourages the discriminator to predict a score of how real or fake a given input looks. This turns discriminator from a classifier into a scoring system. The score is maximizing for real examples and minimizing for fake examples. WGAN requires gradient clipping, which we performed by using tensorflow's clip_by_value().

```
for idx, grad in enumerate(gradients_of_discriminator):
    gradients_of_discriminator[idx] = tf.clip_by_value(grad, -0.01,
0.01)
```

Batch size = 256
Epochs = 300

```
Generator:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 131072)            13107200
batch_normalization (BatchNo (None, 131072)            524288
```

```
leaky_re_lu (LeakyReLU)      (None, 65536)            0
reshape (Reshape)            (None, 16, 16, 256)       0
conv2d_transpose (Conv2DTran (None, 16, 16, 128)       3276800
batch_normalization_1 (Batch (None, 16, 16, 128)       1024
leaky_re_lu_1 (LeakyReLU)    (None, 16, 16, 128)       0
conv2d_transpose_1 (Conv2DTr (None, 32, 32, 64)        819200
batch_normalization_2 (Batch (None, 32, 32, 64)        512
leaky_re_lu_2 (LeakyReLU)    (None, 32, 32, 64)        0
conv2d_transpose_2 (Conv2DTr (None, 64, 64, 3)         9600
=================================================================
Total params: 17,738,624
Trainable params: 17,475,712
Non-trainable params: 262,912
_________________________________________________________________

Discriminator:
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 64)        4864
leaky_re_lu_3 (LeakyReLU)    (None, 32, 32, 64)        0
dropout (Dropout)            (None, 32, 32, 64)        0
conv2d_1 (Conv2D)            (None, 16, 16, 128)       204928
leaky_re_lu_4 (LeakyReLU)    (None, 16, 16, 128)       0
dropout_1 (Dropout)          (None, 16, 16, 128)       0
conv2d_2 (Conv2D)            (None, 8, 8, 256)         819456
leaky_re_lu_5 (LeakyReLU)    (None, 8, 8, 256)         0
dropout_2 (Dropout)          (None, 8, 8, 256)         0
flatten (Flatten)            (None, 16384)             0
dense_1 (Dense)              (None, 1)                 16385
=================================================================
Total params: 1,045,633
Trainable params: 1,045,633
Non-trainable params: 0
_________________________________________________________________
```

Cite machinelearningmastery.com/how-to-code-a-wasserstein-generative-adversarial-network-wgan-from-scratch/ for creating our WGAN using keras.

3.4.2 WGAN on Anime Faces Model Results

Overall the WGAN results are not as good as our fine tuned DCGAN model. Although it is better than our first few DCGAN models, its final output still shows some distortions in the faces generated. Furthermore, it has a general blurriness in the generated faces. This is likely due to Wasserstein loss as a loss function calculates the average score for real or fake images. This is reflected in this sample output below.

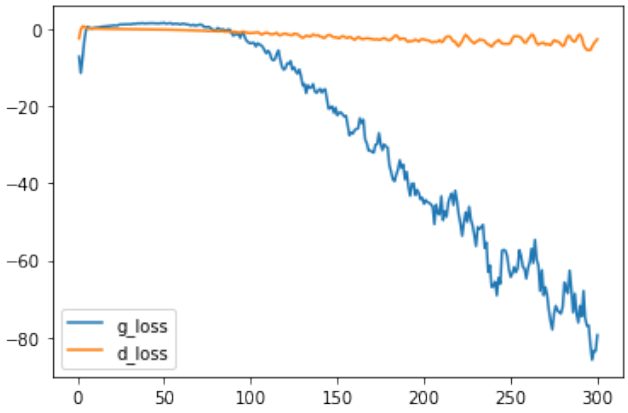

Figure 11: WGAN Training Loss

Another issue is that the loss seems to suggest that the training is not even complete after 300 epochs, as it continues to decrease, yet it is already taking a very long time.

Overall, we can see that while WGAN can grasp the general structure of the anime faces, many details are missing or incorrect, such as the months. The color scheme also indicates that we are merely getting an average color–the hairs and eyes in real drawings are usually drawn with all kinds of colors, the average of all these colors naturally results in the hair and eyes being brown or black, while the face remains generally white. WGAN's images are sharper than VAE's, but it cannot produce realistic images as DCGAN does.

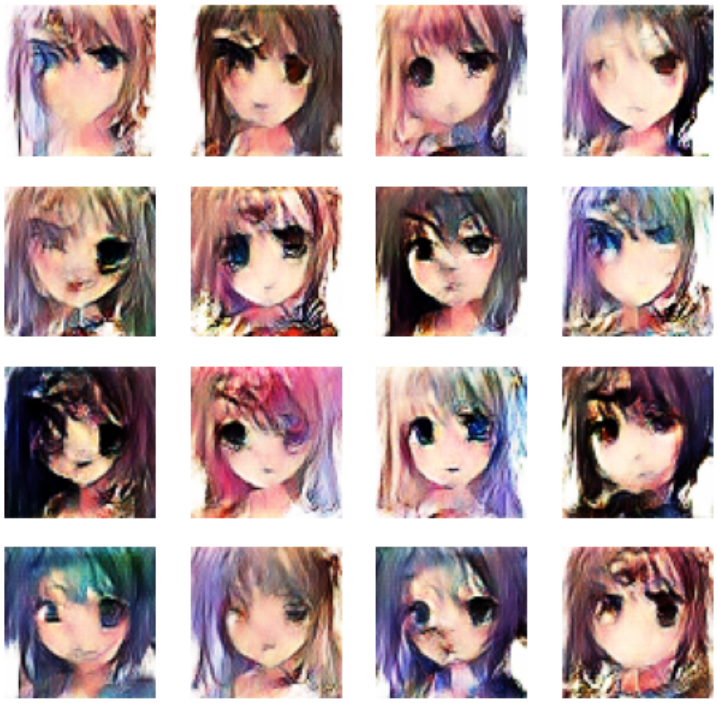

Figure 12: WGAN Sample output

4 Conclusion

Our project concluded with our tuned DCGAN model yielding the best anime face reconstructions. We achieved this model by starting with our PS3 GAN model, incorporating convolutional layers, using relu in the generator, and increasing the number of feature map in the conv2d layers. Other minor parameter tuning are also involved. Our initial DCGAN model struggled from creating distorted anime faces, by we managed to reduce it later. We also tried VAE and WGAN for reconstructing anime faces. VAE generates anime faces that generally resembles an anime face, but is too blurry to be realistic. WGAN is capable of producing fine anime faces, but contains some distortion in the generated images and performs worse than our tuned DCGAN model.

We conclude that we have successfully achieved our initial goal of creating anime faces by using generative models.

5 Alternatives and Future Work

There are mainly two directions that this project can be carried further:
1. The faces generated by our best DCGAN model still shows a little distortion in some parts of the generated faces. The results aren't that realistic. The simple solution for this seems straightforward–more parameter tuning. We might be able to get better results by using larger networks and more epochs. However, according to [2], using variations of GAN such as DRAGAN or StyleGAN seems to be the better solution that offers possibility better computation efficiency and higher performance limits. [2] used DRAGAN to create realistic anime faces that are exactly indistinguishable to real drawings. Sample projects at https://github.com/NVlabs/stylegan2, the official tensorflow implementation of StyleGAN also shows similar capabilities of generating anime faces very similar to real ones. Their output offers extremely fluent outputs without any distortions in the generated images.

We did not find much DRAGAN articles to learn from, and the StyleGAN official implementation only supports Tensorflow 1.x at the moment. Nevertheless, we think these should be the alternatives and future directions if this project is to be carried further.

2. Another direction this project can go is generating anime faces of a particular style, but we need a dataset with anime face labels to do this. We also have not found datasets yet that offers only anime faces in a particular style, such as all yellow/red hair anime faces. However, if possible, we should be able to incorporate GAN with a dataset that contains style labels. In [2], researchers are able to create fixed style anime faces, such as same face shape or same hair color, using fixed noise but different labels. If we train our GAN on different labels, we should be able to create anime faces with the desired hair/face style we wish.

Figure 13: Showcase of our best anime face generations

Figure 14: Examples of [2]'s high resolution faces generated using DRAGAN

References

[1] Keyang Cheng et al. "An analysis of generative adversarial networks and variants for image synthesis on MNIST dataset". In: *Multimedia Tools and Applications* 79.19 (2020), pp. 13725–13752.

[2] Yanghua Jin et al. "Towards the automatic anime characters creation with generative adversarial networks". In: *arXiv preprint arXiv:1708.05509* (2017).

[3] Timothée Lesort et al. "Generative models from the perspective of continual learning". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–8.

[4] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms". In: *arXiv preprint arXiv:1708.07747* (2017).