Animation Faces Generation: Using of Customized DCGAN and Optimization in Face Generation

Group 3: Pengyu WU, Pavan Reddy, Yaxin Zhuang

I. Introduction

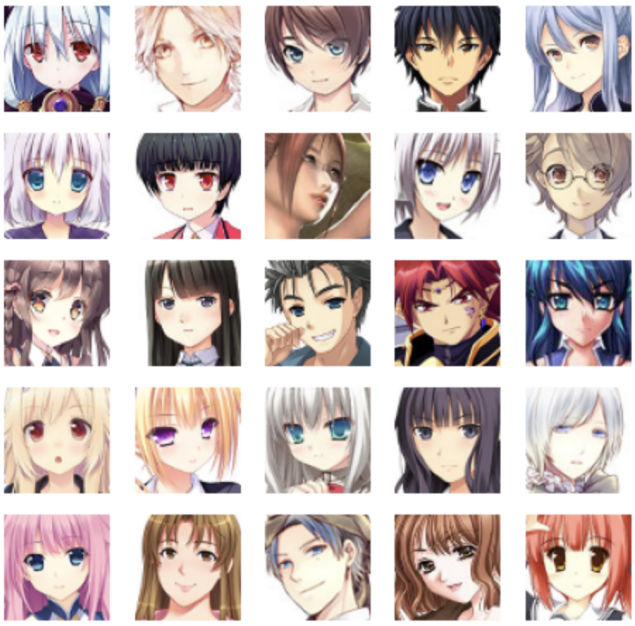
In this project, we applied optimization techniques on DCGAN by modifying model architecture to perform better face generation on Anime Faces data. We analyze our model by looking at discriminator loss, generator loss, and the quality of the resulting image.

This paper is organized as follows, First, we describe our dataset and show the results of Exploratory Data Analysis. Next, we provide a description of the model, then we methodology in terms of model development and optimization, and discuss the comparison and analysis of results. Finally, we summarize our findings and discuss areas of growth. Our main goal is to gain a deep understanding of DCGAN, and our main contribution is to see how we can improve the quality of result images in the context of face generation.

As a brief result, our baseline model performed not very well; the faces it generated are fuzzy, smudgy, and not convincing with missing eyes, and mouths, and some of them can not be recognized as faces. And with the modification of model by changing activation function to ReLU in generator, and adding batch normalization to discriminator. The resulting image becomes more clear and more convincing.

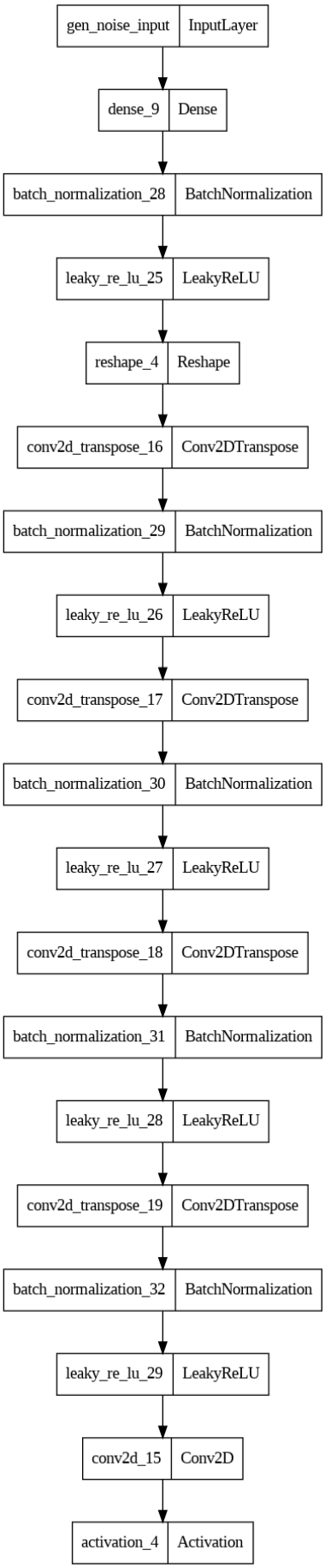
II. Description of Dataset

We aim to use anime faces images obtained from Kaggle. This is a dataset consisting of 21551 anime faces scraped from www.getchu.com, which are then cropped using the anime face detection algorithm in https://github.com/nagadomi/lbpcascade_animeface. All images are resized to 64 * 64 for the sake of convenience. Please also cite the two sources when using this dataset.



III. Description of Model

The baseline model implemented in this project is a customized DCGAN from a GitHub titled “nikhilroxtomar/DCGAN-on-Anime-Faces”. It follows the typical GANs architecture but CNN is used to replace the multilayer perceptron. The convolution layer is used to discriminate the image in the discriminator, and the deconvolution layer is used to generate the image in the generator. The specific structure of the baseline model is as follows: the input layer is followed by a batch normalization layer, and the reshaping layer is used to normalize the preliminary data; then a Conv2DTranspose layer, and a batch normalization layer is followed by a LeakyReLU activation layer, and the Tanh activation function is used in the output layer of the generator. The structure of the discriminator is a Conv2D layer (2D convolution layer) with LeakyReLU activation function, a dropout layer. These layers form a group and three groups are added. Finally, a flattening layer and a fully connected layer are used to flatten the data and output the probability of whether it is sample data or generated data. The figure below is shown the structure of the baseline model.


gen_noise_input
InputLayer
dense_9
Dense
batch_normalization_28
BatchNormalization
leaky_re_lu_25
LeakyReLU
reshape_4
Reshape
conv2d_transpose_16
Conv2DTranspose
batch_normalization_29
BatchNormalization
leaky_re_lu_26
LeakyReLU
conv2d_transpose_17
Conv2DTranspose
batch_normalization_30
BatchNormalization
leaky_re_lu_27
LeakyReLU
conv2d_transpose_18
Conv2DTranspose
batch_normalization_31
BatchNormalization
leaky_re_lu_28
LeakyReLU
conv2d_transpose_19
Conv2DTranspose
batch_normalization_32
BatchNormalization
leaky_re_lu_29
LeakyReLU
conv2d_15
Conv2D
activation_4
Activation

We will implement different methods to find the model structure that will produce better results. The optimized model structure is similar to the baseline with changing activation function to ReLU in generator, and adding batch normalization to discriminator.

IV. Experimental setup

Training:
Model that we are using is a baseline model with typical GANs structure consisting of generator and discriminator. We look at output of generator and discriminator, experimenting with learning rate, batch size, loss function, and optimizer.

Performance:
The loss function we used is Binary Cross-Entropy loss with label_smoothing of 0.1, and the optimizer is Adam optimizer with learning rate of 0.0002. We obtaining two loss from the discriminator and one loss from generator, the way we obtained them are freezing the Generator and train the Discriminator once on generated samples to get d1_loss, freeze the Generator and train the Discriminator once on real samples to get d2_loss, and freeze the Discriminator and train the Generator with Latent vector to get g_loss.The goal is to minimize generator loss and maximize discriminator loss. If g loss is small, it means the generator is able to produce very realistic images. If d loss is high, it means it is not able to classify images into real or fake, meaning the generator is doing a good job.

V. Methodology

1. **Result from baseline model:**

2. **Adjust the learning rate:** we adjusted the learning rate for both the generator and discriminator optimizers to find a better balance between convergence speed and stability. The quality of generated images is lower than our baseline visually when the learning rate is 0.0001 and 0.0005.

(LR: 0.0001)

(LR:0.0005)

3. **Use a different optimizer:** two different optimizers was applied in our baseline model, RMSprop and SGD. RMSprop: This optimizer has been shown to work well in deep learning, and is sometimes used for GANs as well. It also adapts the learning rate based on the gradient of the loss function but uses a moving average of the squared gradients to normalize the learning rate.

SGD: This is the classic stochastic gradient descent optimizer, and can also be used for GANs. It updates the model parameters based on the gradient of the loss function for each batch of training data.

4. **Use a different loss function:** The binary cross-entropy loss function used in our baseline model, and we experimented with different loss functions for better results, such as MeanSquaredError and Hinge Loss. MeanSquaredError: calculates the mean squared difference between the target and predicted values. It is commonly used in image generation tasks.

Hinge loss function: is used in the context of the WGAN-GP architecture. It encourages the discriminator to output high values for real samples and low values for fake samples. When using MeanSquaredError loss function, the model did not output better quality images.

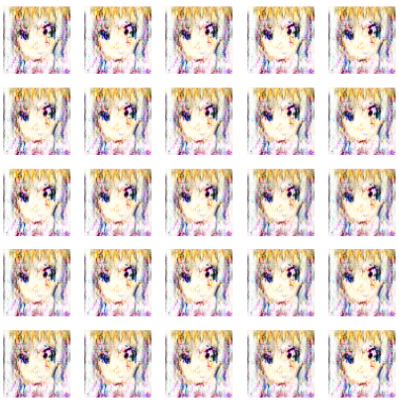
5. **BatchNormalization in discriminator:** Added batch normalization in the discriminator can help improve the stability of the training process by normalizing the activations of the previous layer, reducing the internal covariate shift, which can make it easier for the network to learn and converge to a good solution. Additionally, batch normalization can help prevent overfitting and improve the generalization performance of the discriminator. And the model with batch normalization in the discriminator has better performance on the result.

6. **Change activation function in generator:** In the baseline model, it has a LeakyReLU activation function for each layer of the generator. But the model with the ReLU activation function has achieved better image quality for the result. Because the generator is responsible for generating realistic images from random noise, it needs to be able to produce diverse and meaningful features. In order to do so, the ReLU activation functions can be more effective than LeakyReLU in producing sparse and diverse features, which can lead to more diverse and realistic images. Additionally, ReLU can help the generator produce sharper and more detailed images, which is important for generating high-quality images that are indistinguishable from real images.

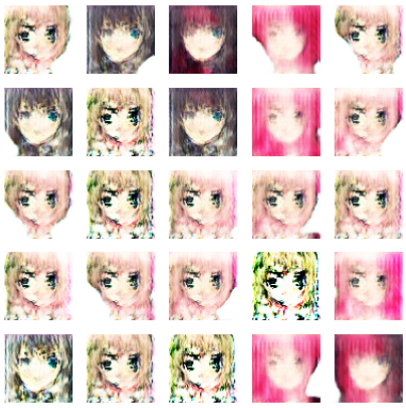
- The result below shows the result of the modified model structure with more layers, batch normalization in the discriminator, and ReLU activation function in the generator. The model performance is clearly improved as shown below.

7. **LeakyRelu Value:** A variety of LeakyRelu values were tried. The default was 0.2. A Leaky ReLu of 0.5 produced slightly better results, but the images appeared more blurred. A LeakyRelu of 0.1 did not produce any better results, while a LeakyRelu of 0.05 led to mode collapse.

0.05



0.1

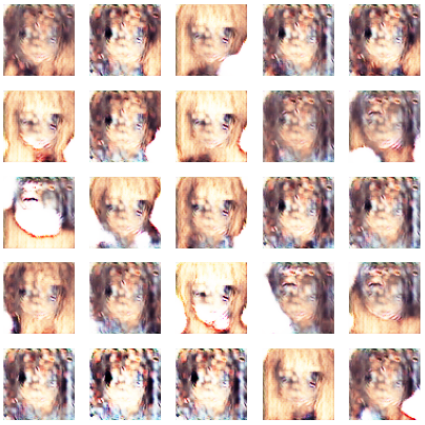


0.5 0.2

8. **Adding noise in discriminator:** And while taking look at the result image from original code I notice the generated faces are smudgy, fuzzy and not convincing, and easily collapsing. Therefore, I did a search on how to stable the generating process of DCGAN, and from paper *Improved Techniques for Training GANs*, authors suggest add Gaussian noise to the output of each layer of the discriminator, and I added it to discriminator for final project. According to the paper, adding noise to the discriminator can help stabilize the training process and improve the performance of the the network by forcing network to learn more robust features that are not affected by small variations in the input data, preventing mode collapse, and improving stability by adding noise to the discriminator's inputs can make it more difficult for the discriminator to distinguish between real and fake images, which can improve the stability of the training process and prevent the discriminator from becoming too dominant and overpowering the generator. But the result of adding noise training on anime face data performs not very well.

    - Result of adding noise after first input layer

- Result of adding noise after each layer

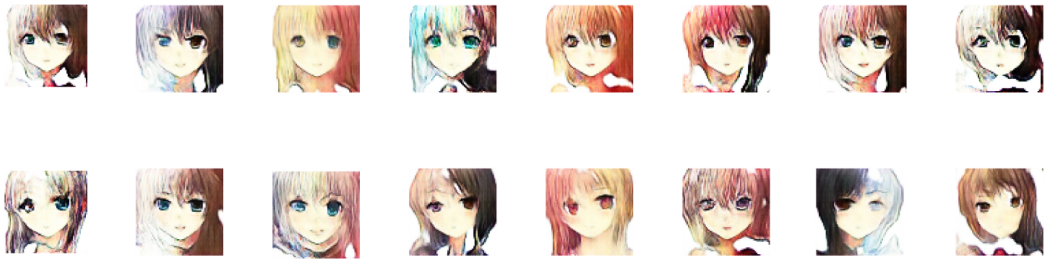


9. **Changing Kernel Size:** The kernel size in the original code was 5 for the conv and deconv layers. Changing the kernel size to 3 yielded better images but merged features since it failed to capture features well. This is expected since a smaller kernel cannot capture many features. A kernel size of 7 yielded all white images and any higher size also yielded white images.

10. **Progressive Growing Kernels:** An approach of progressively increasing the kernel size was tried, wherein, the first layer had a kernel size of 3, the second layer had a kernel size of 5 and so on. This approach seemed to yield better results and the loss minimization of the generator was much faster. Although it significantly increased the training time.

11. **Adding more Filters and increasing the number of epochs:** after experiencing different methods of optimization, the generated images did not have a significant improvement except for changing the LeakyReLU activation function to the ReLU activation function. We added more filters and increased the number of epochs based on it. On the generator layer, the filters changed to 64 from 32, on the discriminator layer the filters changed to 128 from 64. And the number of epochs increased to 200 from 100. However, the model did not generate better images.

VI. Result

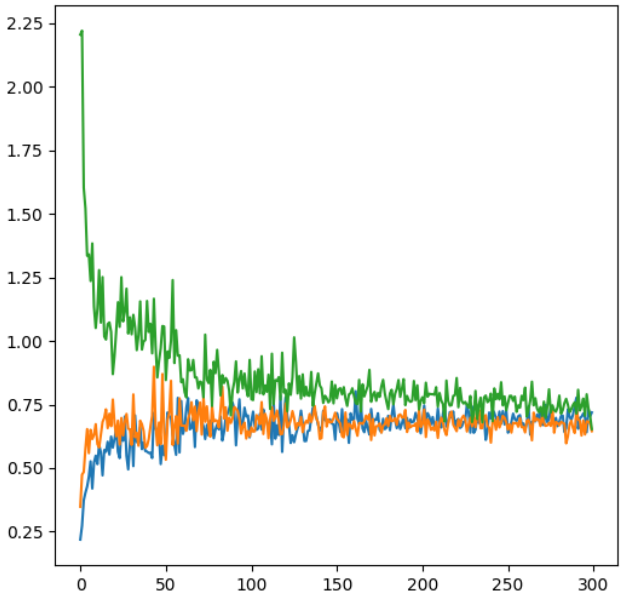


The images on the left are the ones from the baseline model and the ones on the right are from the most optimized model. The results on the bottom are the best images from the optimized model with hyperparameter tuning, trained for 300 epochs. The images from the bottom are more clear and less fuzzy.

Increasing the number of filters to 64 and 128 in the generator and discriminator also produced better results.

Training the model for longer also produced better results

The generator loss initially started at 2.25 and converged at around 0.7 at epoch 300. The discriminator loss was around 0.2 initially and converged at around 0.7.



VII. Summary and Conclusion

In conclusion, this project is mainly on performing anime face generation tasks on anime face data using GANs based on replacing forward feed layers with deconv and conv layers. We fine-tuned the model by adjusting model architecture, and some hyper-parameters like batch size, kernel size, and learning rate.

The above techniques were employed to improve the performance of a GAN model: adjusting the learning rate, using different optimizers such as RMSprop, SGD, and Adam, exploring different loss functions such as MeanSquaredError and Hinge Loss, and adding more filters and increasing the number of epochs. However, none of these modifications resulted in substantial improvements in the generated images, except when the LeakyReLU activation function was replaced with ReLU, and batch normalization was added to the discriminator.

For the improvement of this project, If we have more time we would like to focus on three parts, the first is to do more research on loss function, and the second is to improve the data preprocessing, the third is hyperparameter turning. For the data, it might have some bad cropping results or some non-human faces so we should do more data exploration and also try to find out how to correctly add Gaussian noise to the discriminator. For the loss function and hyperparameters, we should do more research and implementation on it.

VIII. Reference

https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/
https://github.com/hwalsuklee/tensorflow-generative-model-collections
https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks/
https://www.hindawi.com/journals/misy/2022/9005552/
https://arxiv.org/pdf/1801.09195.pdf
https://github.com/nikhilroxtomar/DCGAN-on-Anime-Faces/blob/master/gan.py
https://arxiv.org/abs/1606.03498

https://pyimagesearch.com/2020/11/16/gans-with-keras-and-tensorflow/
https://proceedings.neurips.cc/paper_files/paper/2018/file/90365351ccc7437a1309dc64e4db32a3-Paper.pdf
https://paperswithcode.com/method/label-smoothing
https://arxiv.org/pdf/1511.06434v2.pdf
https://www.kaggle.com/datasets/soumikrakshit/anime-faces