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, the resulting image becomes more clear and more convincing.

I mainly take responsible for tuning hypermeters, including adjust the learning rate, use a different loss function, use a different optimizer, adding more filters and increase the number of training epochs.

Work and results in detail

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 learning rate is 0.0001 and 0.0005.

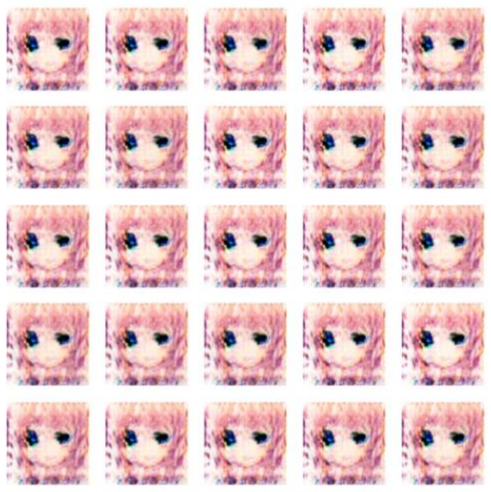
(LR: 0.0001)

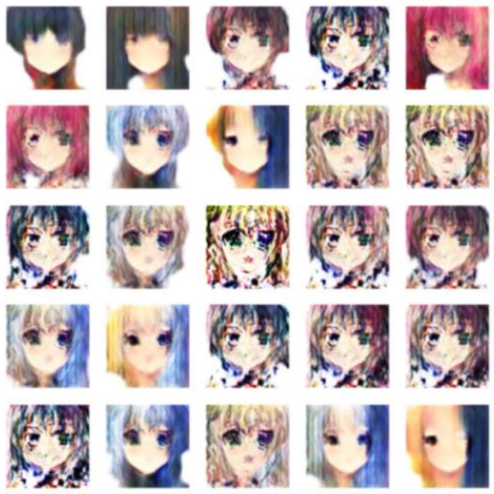
(LR:0.0005)

Use a different optimizer: Two optimizers, RMSprop and SGD, were tested. **RMSprop** is a commonly used optimizer in deep learning and is also applied in GANs. It adjusts the learning rate based on the loss function gradient and normalizes the learning rate by utilizing a moving average of the squared gradients. The image produced by the RMSprop optimizer are presented below, indicating that the model did not exhibit significant enhancement.

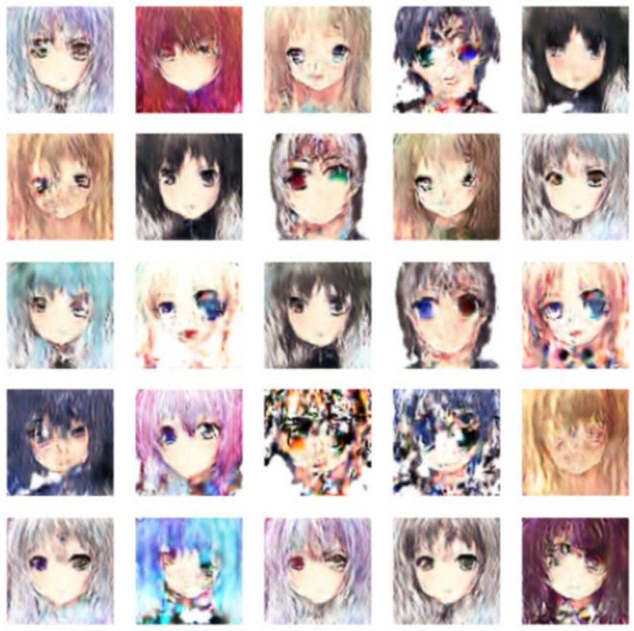
SGD: The stochastic gradient descent (SGD) optimizer is a traditional technique that can be applied to GANs. It updates the model parameters for each batch of training data by utilizing the gradient of the loss function. The resulting image produced by SGD is superior to that generated by the RMSprop optimizer but is not significantly different from the one produced by Adam.

Use a different loss function: Our baseline model employed the binary cross-entropy loss function, and we explored other loss functions, including MeanSquaredError and Hinge Loss, to improve performance. MeanSquaredError measures the mean squared difference between predicted and target values and is typically utilized in image generation tasks. On the other hand, the Hinge loss function is implemented in the WGAN-GP architecture to prompt the discriminator to produce high values for real samples and low values for fake samples. However, when we employed the MeanSquaredError or Hinge loss function, the model did not generate higher-quality images.


(MeanSquaredError)


(Hinge Loss Function)

Adding more Filters and increase the number of epochs: Although we attempted various optimization techniques, we did not observe any substantial improvements in the generated images, except when we replaced the LeakyReLU activation function with ReLU. We augmented the model by adding more filters and doubling the number of epochs, with 64 filters in the generator layer and 128 filters in the discriminator layer and increasing the number of epochs to 200 from 100. Nevertheless, these modifications did not result in higher-quality images.



Summary

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.

Calculate the percentage of the code that you found or copied from the internet
The total number of line of the py. file is 430, and I modified 10 of lines and added another 18 lines of my own code. The percentage would be
(430-10)/(430+18)= 93.73%

References

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