Project 1: Implementation of DCGAN and SA-GAN

Yizhan Wu
Department of Computer Science Engineering
University at Buffalo
Buffalo, NY 14260
yizhanwu@buffalo.edu

Abstract

The purpose of this project is about learning a generative model from which samples can be generated. The task of this project is to generate sample from distribution, and measure models' performance by calculating Frechet Inception Score, as well as loss of generator and discriminator. To achieve the task, we will be implementing two types of Generative Adversarial Networks in this project: Deep Convolution GAN and Self-Attention GAN, using CIFAR-10 dataset.

1 Introduction

1.1 Project overview

This project introduces us to an important deep learning topic: Generative Adversarial Network. We will be implementing two types of Generative Adversarial Networks in this project: Deep Convolution GAN and Self-Attention GAN, using CIFAR-10 dataset.

1.2 Introduction on the tasks

There are two main parts of the project:

1. Implementing DCGAN – Deep Convolution GAN.
2. Implementing SA-GAN – Self-Attention GAN.

To measure model performance, we will be calculating Frechet Inception Score over generated images and testing set from CIFAR-10 dataset.

1.3 Introduction on the dataset used

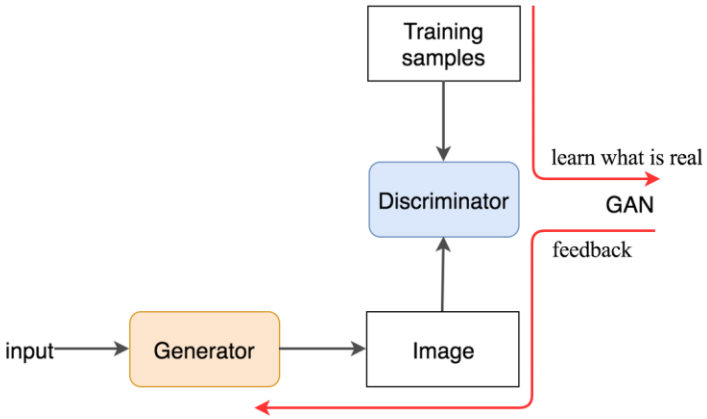
The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. Each image has an image size of 32*32.

2 Theory basis

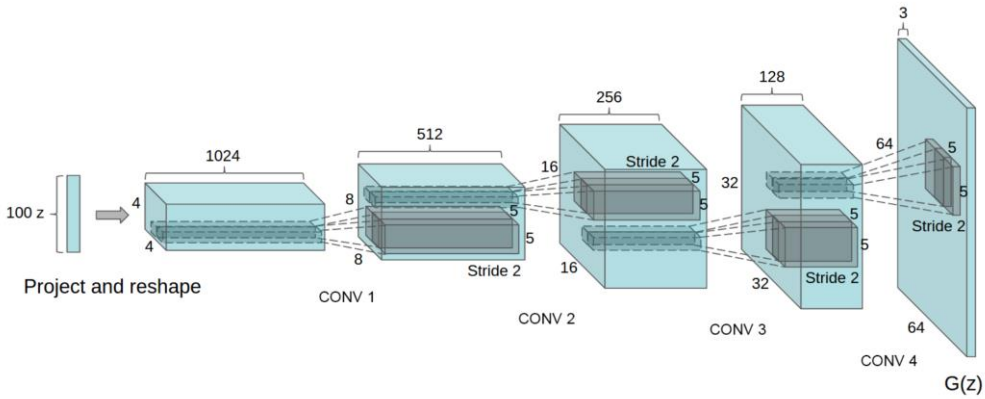
Below are some theories basis used throughout this project. These bases were mainly used in understanding and establishing models as well as tuning hyper-parameters.

2.1 Generative Adversarial Networks

Basic GAN architecture:



Basic DCGAN consists of two neural networks: a discriminator (D) and a generator (G) which compete with each other making each other stronger at the same time. In Deep Convolutional GANs (DCGANs) in which G and D are both based on deep convolution neural network.



Generator performs multiple transposed convolutions to upsample z to generate the image x. We can say it's the deep learning classifier in the reverse direction. By training with real images and generated images, GAN builds a discriminator to learn what features make images real. Then the same discriminator will provide feedback to the generator to make it better.

The discriminator looks at real images (training samples) and generated images separately. It distinguishes whether the input image to the discriminator is real or generated. We train both networks in alternating steps, and eventually the generator will create images that the discriminator cannot tell the difference.

2.2 Self-Attention GANs

Self-attention mechanism:

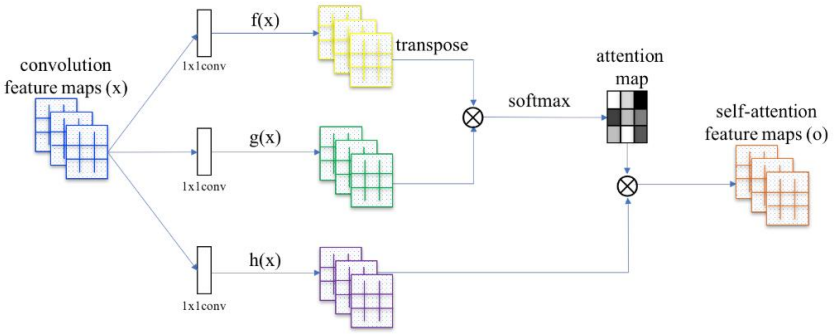


Figure 2: The proposed self-attention mechanism. The $\otimes$ denotes matrix multiplication. The softmax operation is performed on each row.

To mitigate the issues of learning being prevented from long-term dependencies, self-attention module is introduced in DCGANs. Self-attention exhibits a better balance between the ability to model long-range dependencies and the computational and statistical efficiency.

The idea for self-attention module is to give the generator information from a broader feature space, not only the convolutional kernel range. By doing so, the generator can create samples with fine-detail resolution.

To alleviate mode collapse issue, GANs are trained with Wasserstein Loss.

2.3 Wasserstein loss

Algorithm for the Wasserstein GAN:

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

In a standard GAN, the discriminator has a sigmoid output, representing the probability that samples are real or generated. In Wasserstein GANs, however, the output is linear with no activation function! Instead of being constrained to [0, 1], the discriminator wants to make the distance between its output for real and generated samples as large as possible. The most natural way to achieve this is to label generated samples -1 and real samples 1, instead of the 0 and 1 used in normal GANs, so that multiplying the outputs by the labels will give you the loss immediately. Note that the nature of this loss means that it can be (and frequently will be) less than 0.

3 Code implementation

Below I'll explain the tasks completed in my implementations.

3.1 What have been implemented?

Building models of DCGAN and SA-GAN.

Generator model and discriminator model was implemented using my own code, so is Self-Attention module, Wasserstein loss calculation, and FID evaluation metric calculation.

Dropout is added to discriminator to improve GAN performance.

Spectral Normalization is applied on generator and discriminator. Existing implementation method was used.

(https://github.com/IShengFang/SpectralNormalizationKeras/blob/master/SpectralNormalizationKeras.py)

3.2 What can be expected from the output?

DCGAN:

Generator will generate 10,000 images over more than 500,000 iterations (1000 epochs with batch size of 64). Images will be saved to a local folder. FID score is calculated, as well as generator loss and discriminator loss over epochs. Binary cross entropy loss function is used.

Output looks like this:

```
Epoch: 970/1000, Discriminator loss: 0.506115, Generator loss: 3.137685
FID: 57.414
Epoch: 971/1000, Discriminator loss: 0.628891, Generator loss: 2.450305
FID: 97.622
Epoch: 972/1000, Discriminator loss: 0.688544, Generator loss: 2.716472
FID: 219.056
Epoch: 973/1000, Discriminator loss: 0.699170, Generator loss: 2.019724
FID: 38.286
Epoch: 974/1000, Discriminator loss: 0.761890, Generator loss: 2.621065
FID: 77.244
Epoch: 975/1000, Discriminator loss: 0.710665, Generator loss: 2.186087
FID: 134.417
Epoch: 976/1000, Discriminator loss: 0.789739, Generator loss: 2.685663
FID: 48.611
```

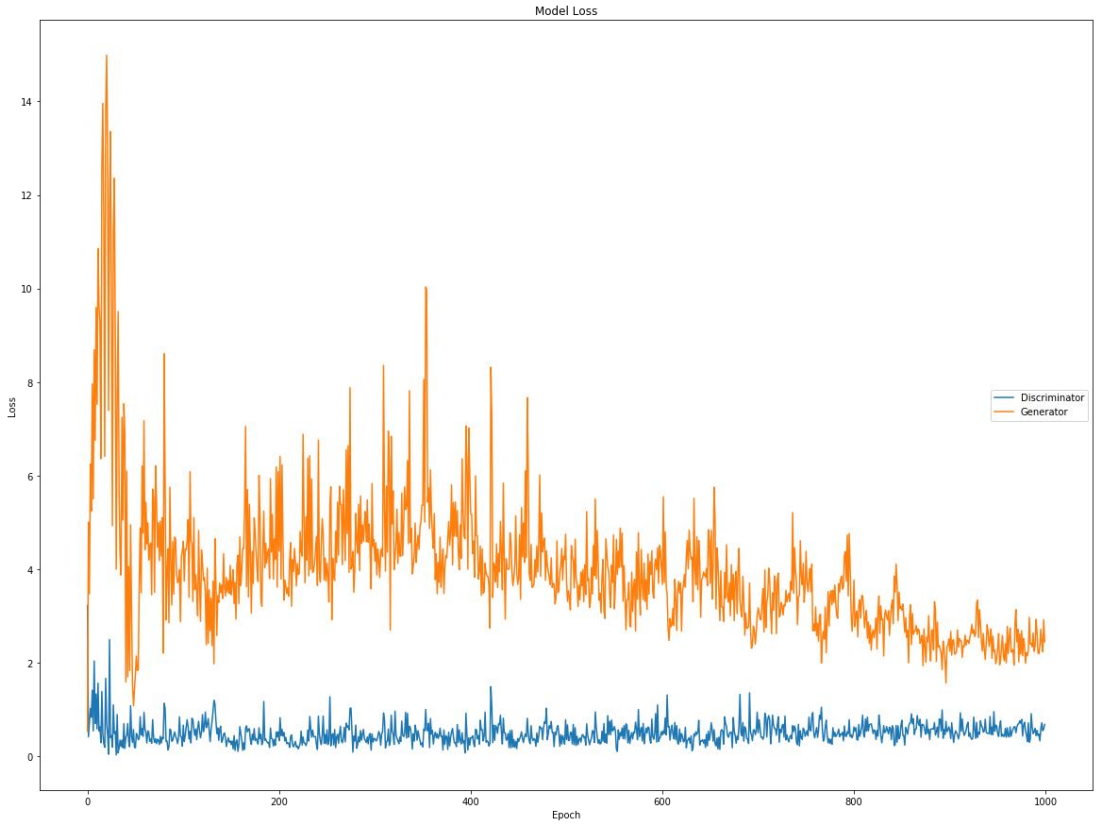
SA-GAN:

Self-Attention module is applied to the generator and discriminator, and spectral normalization is applied to the discriminator. Wasserstein loss function is used.

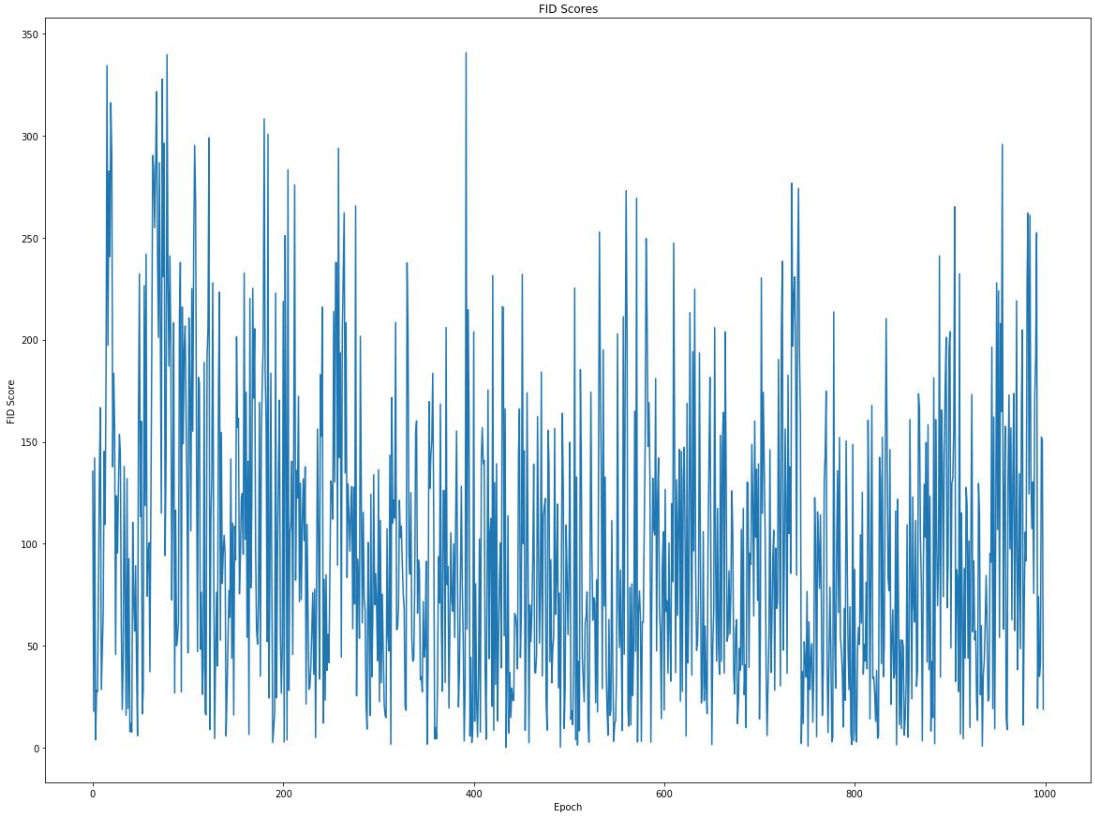
Results from both GAN models have their individual inadequacies, but unfortunately, due to long training times and insufficient project time, I wasn't able to fine tune the model and make it perfect. Possible reasons and analysis for certain inadequacies will be discussed in the next section.

4 Results and graphs

4.1 DCGAN



Generator loss and discriminator loss over epochs.



FID scores over epochs.

Images generated.

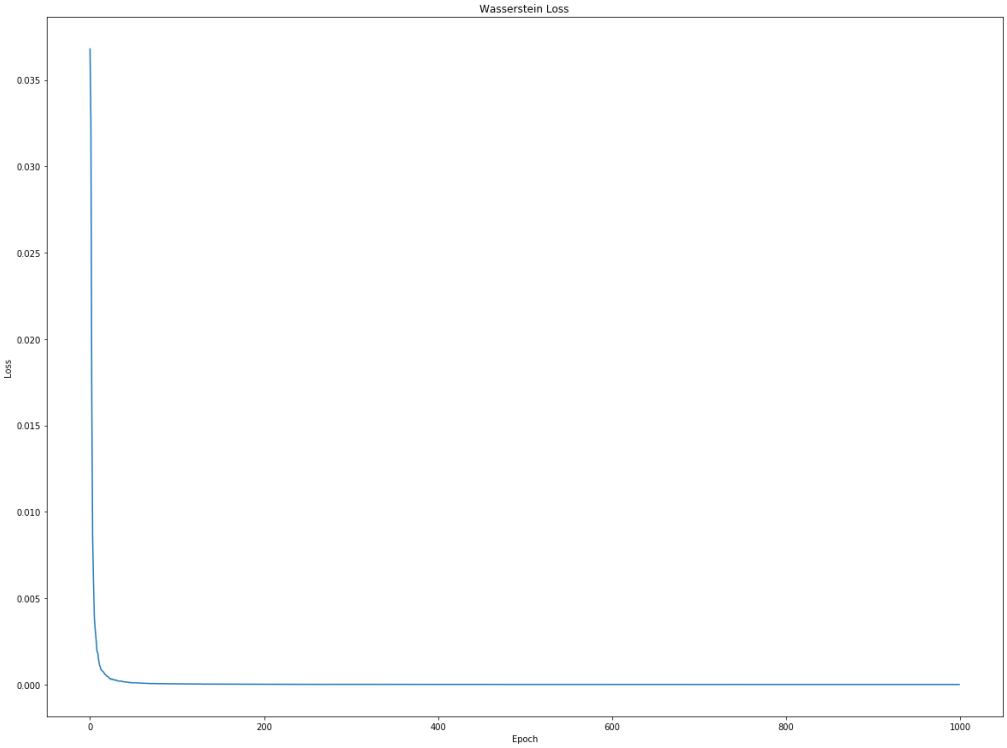
We are able to see from the FID score over epochs graph that FID score fluctuates a lot over iterations. It does show a vague tendency to be decreasing over time though, which holds true theory wise.

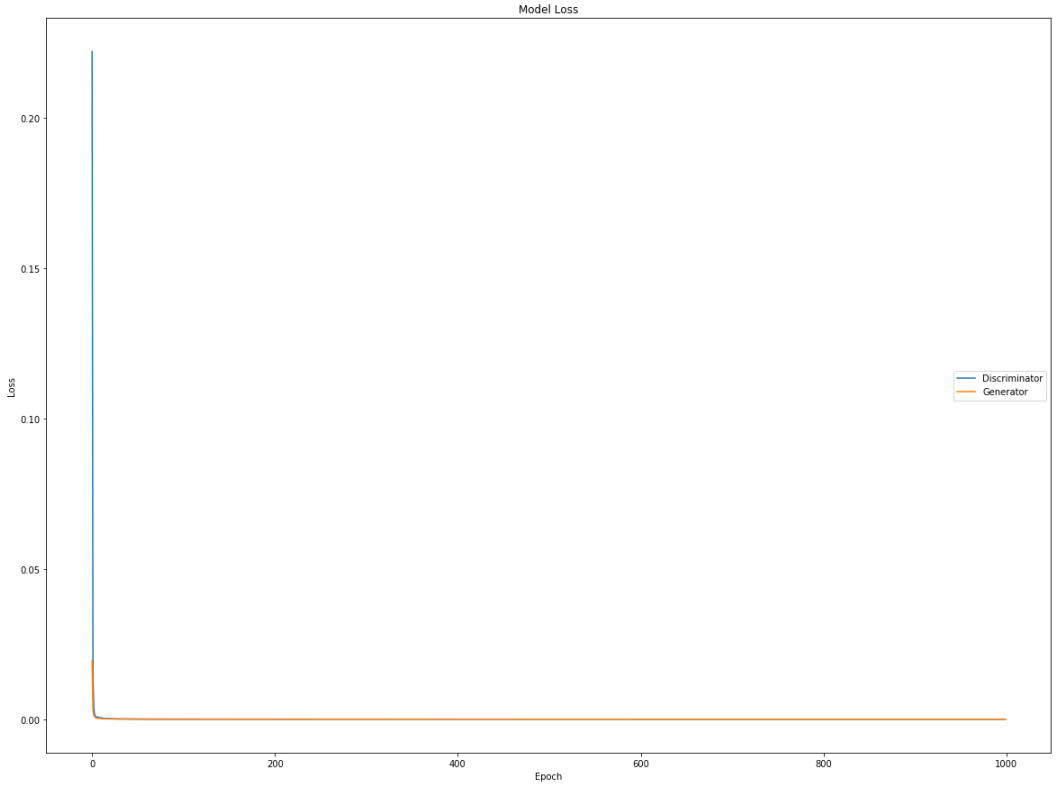
Calculation shows: FID mean = 125.40, standard deviation = 102.38, and the lowest FID score is 8.920, which is fairly decent in terms of DCGAN.

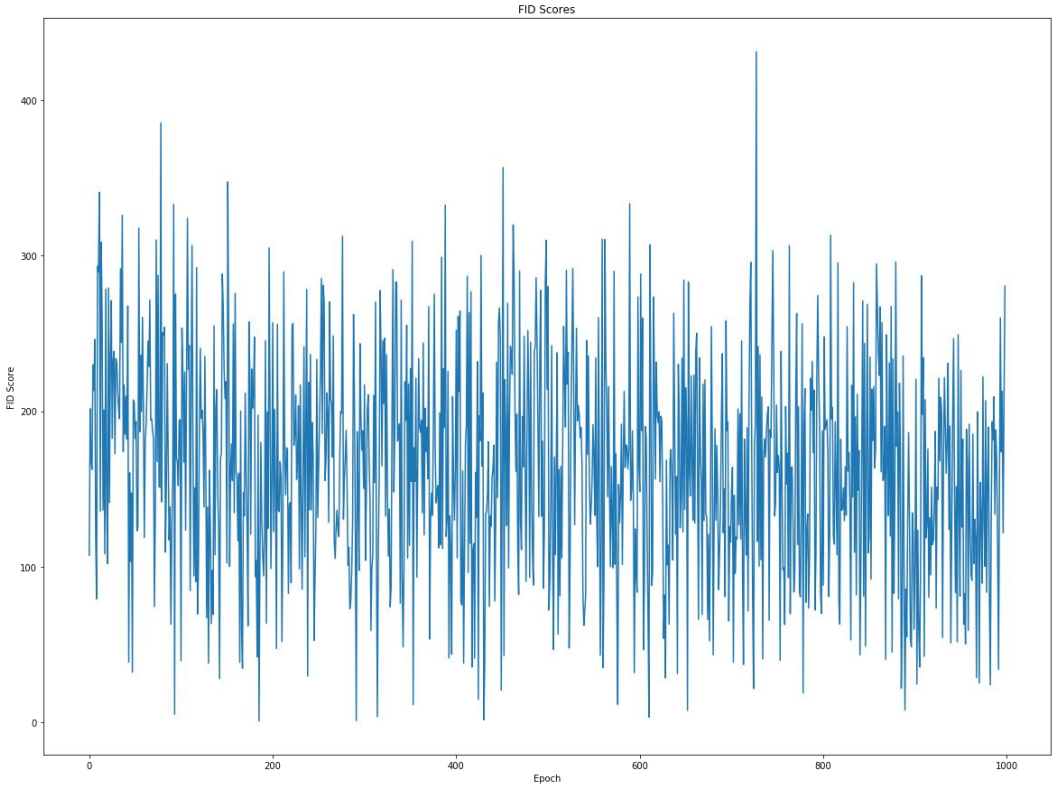
However, the model loss graph showed something more interesting. Discriminator loss was able to stabilize very early, but generator loss remained high for a long time, and bounced between 2-6 later. It implied that discriminator is overpowering generator here, which in turn causing FID score to show high fluctuations. I did try adding dropout to discriminator, but that didn't show too much of an improvement.

I would need more time to fine tune some hyper-parameters in order to make the model better, however due to long training times, it wasn't possible. I'm happy with the current results.

4.2 SA-GAN







FID scores over epochs.

FID score in SA-GAN model showed lower standard deviation (70.77), but higher mean value (164.81). However, the lowest FID score is also lower, 0.945, which is expected, since self-

attention module will balance better. Both losses showed consistency, however, FID scores still indicates that model is not improving in learning overtime, since FID scores fluctuates a lot.

5 Conclusion

5.1 What I learned from the project

From building models and implementing self-attention modules to learning FID evaluation metric and analyzing results, I learned a lot in this project. Time was an issue, since long training times made tuning parameters very hard, and thus yielded less optimal results, however, coding and learning about all the fundamentals behind GANs broadened my knowledge of deep learning, and made my problem analyzing skills better. Given more time, I could definitely make my model more optimal and generate better images.