Contents

[Introduction 1](#_Toc127700359)

[Language and Libraries 1](#_Toc127700360)

[The Dataset 1](#_Toc127700361)

[The Generator 4](#_Toc127700362)

[The Discriminator 4](#_Toc127700363)

[The GAN Class 5](#_Toc127700364)

[Constructor 5](#_Toc127700365)

[Training 6](#_Toc127700366)

[Mode Collapse Score 7](#_Toc127700367)

[Frechet Inception Distance 7](#_Toc127700368)

[Results 8](#_Toc127700369)

[Deploying the GAN model 8](#_Toc127700370)

In this chapter I will be discussing the implementation of the GAM model. In particular, the structures of the generator and discriminator, the loss functions and optimization algorithm and the training phase. In addition, I will be also talking about the deployment in a flask web application.

# Introduction

The project’s goal is to develop an application that produces images using the Gan model. As stated before the Gan model consists of two models which are the generator and discriminator. The models will be trained using a dataset of fine art images and after the training phase, the generator should be able to synthesize new art images that even us humans would think that it is real.

# Language and Libraries

As both the generator and discriminator models will be deep learning models, it would be ideal to use Python as the main development language to due to its major success when it comes to AI applications. In particular, I will be using tensorflow 2 library for implementing the deep learning models. This is also because of the GPU support offered by tensorflow for deep learning applications. It is well known that deep learning models are highly parallel, hence using only the CPU in the training phase will result in an extremely slow training phase. Hence it is important to train on the GPU. I have a machine equipped with RTX 2060 GPU which is a good GPU for such applications.

# The Dataset

In terms of the dataset, I chose the Best Artwork of All Time dataset from Kaggle (<https://www.kaggle.com/datasets/ikarus777/best-artworks-of-all-time>). This dataset contains a collection 8,683 art images created by the 50 most influential artists of all times including Van Gogh, Michel Angelo and much more. Below are some of the painting from the dataset.

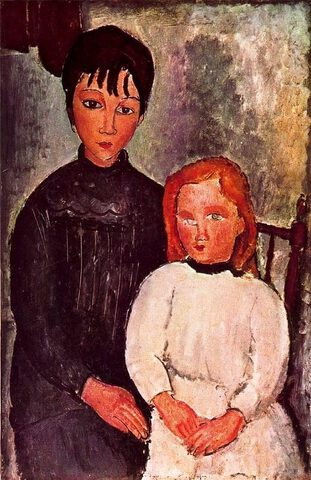


Figure A painting by Alfred Sisley from the dataset

Figure A painting by Amedeo Modigliani



Figure A painting by Caravaggio from the dataset

# 

# The Generator

As mentioned in the previous chapters, the aim of the generator is to generate a fake art image that resembles the samples in the dataset. As a structure, the generator takes as input 100 random numbers generated from a random normal distribution and will output an images of size pixels. This input is known in the literature as the latent space. After that this input is connected to a dense layer of neurons followed by a batch normalization layer and Relu activation layer. Batch normalization is widely used in convolution networks to stabilize the values fed to the following layers and relu activation is a non-linear activation that is also widely used in convolution networks. Relu uses the following simple formula . After that, the resulting layer will be reshaped to a volume of width and high of 8 and a depth of 256. In order to achieve the final output size, the generator uses 4 transpose convolution layers. Each transpose convolution will double the width and height in the feature map using a stride of 2 and reduce the depth based on the number of convolutions. This will results in the following feature maps ( , ) Each transpose convolution is followed by batch normalization and relu activation layers except for the last layer which uses tanh activation to generate the final output. This will generate pixel values in the range (-1,1). We can apply simple normalization step to make the values in the range (0,1).

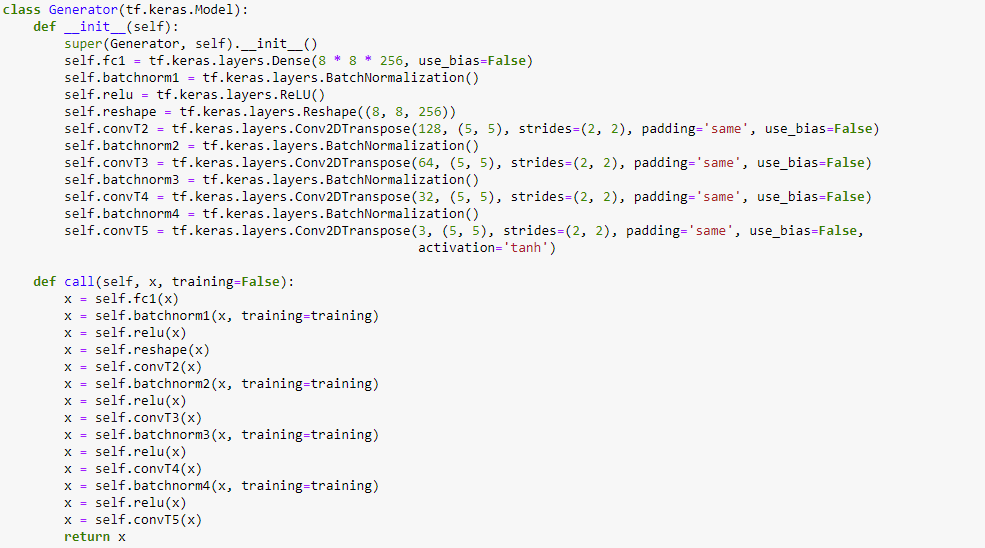


Figure : Generator implementation

# The Discriminator

The discriminator, on the other hand, will take as input an image of size and outputs a value between 0 and 1 corresponding to the probability whether the images is real. The pixel values in the input image are normalized to the range (-1,1) so that it is consistent with the generator output values. This can be achieved using a number of convolution layers where each convolution layer will half the depth and width of the corresponding input feature map. Each convolution is followed by a batch normalization layer and leaky ReLU activation. Leaky ReLu is widely used in the discriminator in the literature as it enhances the training process. The final layers in the discriminator is a flatten layer that will convert the feature map into a list of neurons and the a dense layer with single neuron and sigmoid activation that will give a single output between 0 and 1 corresponding to the probability of the image being real. Below is the discriminator code.

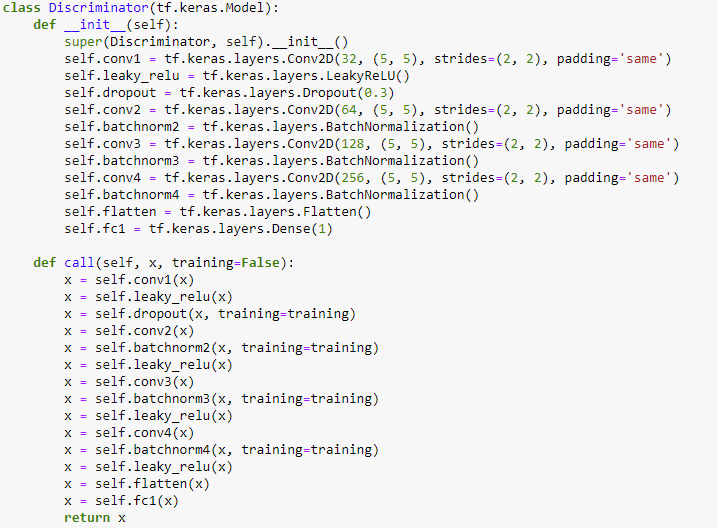


Figure Discriminator implementation

# The GAN Class

In the GAN class, we define the loss functions, the training function and the optimization algorithm in addition to the dataset. The GAN constructor take as input the generator and discriminator models. It also defines the important GAN metric to assess the generated image quality. The two metrics that we implemented is the mode collapse metric and Frechet Inception Distance (FID) score.

## Constructor

In the constructor, we initialize the important GAN variables such as the generator and discriminator, the optimizers for the generator and discriminator, the dataset and the base model used for FID score. This is shown in the code below.

Text

Description automatically generated

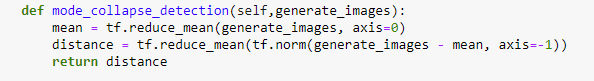
## Training

In the training method, we run a number of epochs where in each epoch we pass over the training dataset in minibatches. In each training iteration, we fetch a training minibatch of the training images. The minibatch size is set to 128 images/minibatch. After that the generator generates 128 fake images. The discriminator then will compute the probabilities for the generator images and the real images in the current minibatch. These results will then allow us to compute the loss functions for the generator and discriminator. The generator loss is basically the mean of the negative log of generated images probabilities (computed by the discriminator). In a typical situation, the probabilities for the generated images should be equal to 1 and hence the loss function becomes 0. For the discriminator, the loss function is the mean of the negative log of real images probabilities plus the mean of the negative log of 1 subtracted by the generated image probability (probability of the image being fake). The final step in the training iteration is to compute the gradients for the generator and discriminator. These gradients are computed using the backpropagation algotihtm based on the loss function for the generator and discriminator. After that, we update the weights of generator and discriminator based on these gradients and the Adam optimization algorithm. Below is the code for the training method.



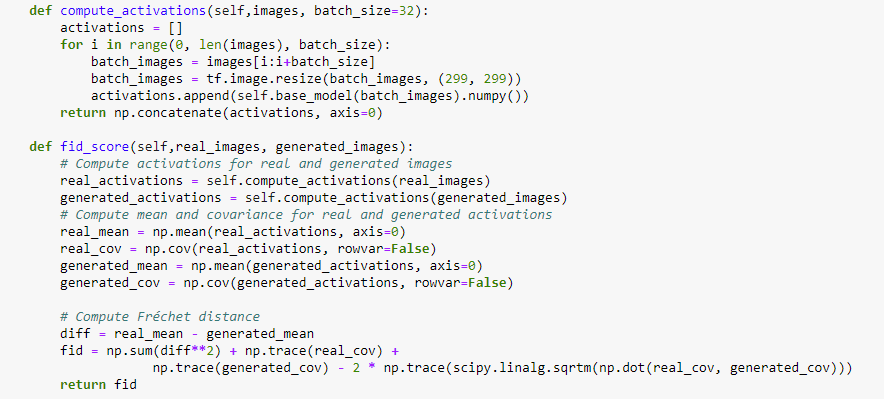
## Mode Collapse Score

The mode collapse score is used to measure the variability in the generated images. The mode collapse is a failure type in GAN in which the generator generates new images that are very similar to one another. This happens for many reasons including lack of variability in the training samples. The code of the Mode collapse is depicted below.



## Frechet Inception Distance

The FID distance is used to assess the quality of the generated images. This is achieved by using a pretrained Inception-v3 model and using that model to compute the activations for the real and generated images. After that, the distance between the distribution of activations is computed. The code for FID score is provided below.



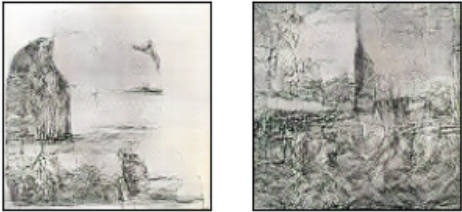
# Results

In this section, I will show some of the results after training the GAN model using 500 epochs. In terms of the FID score, I noticed that the score value went down from 400 to around 200. The generated images were normalized so that pixel values are in the range (0,1). The mode collapse score went down from 0.44 to 0.4. While the difference in the mode collapse score does not seam significant, the generated imaged were very divergent and different to one another. The images below show some of the generated images from the GAN model in the final epochs.



A picture containing text, painting

Description automatically generated



A picture containing text, indoor, mammal, painting

Description automatically generated

Figure Images generated from the generator.

The results look divergent, however, tend to be more of abstract and surrealistic nature. This may suggest that training the GAN to generate very detailed paintings may need much larger dataset.

# Deploying the GAN model

For deploying the GAN model I will be using Flask. Flask is a Python framework for developing web applications. The code for my flasks web application is provided below.

Text

Description automatically generated

The interface of my web app is very simple. It has a single generate button that once pressed direct the user to the generated image.

Graphical user interface, text, application

Description automatically generated