Generative Adversarial Network

DCGAN, CGAN, AutoEncoder.

Frechet Inception Distance Inception Score

Abstract

This project involves the implementation of different types of Generative Adversarial Networks, DCGAN, CGAN, Adversarial AutoEncoder, and evaluation of their performances using frechet Inception Distance score (FIDs) and Inception Score (IS) using Fashion MNIST dataset.

1 Introduction

Generative Adversarial Networks (GAN) are an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset.

In this project, I have implemented different types of Generative adversarial network, and evaluated their performances using Frechet Inception Distance score and Inception score. First of all, I trained the DCGAN and save the generator as **.h5**,I generated the images and saved in **.csv** file, then I trained the Classifier and saved it. At last I calculated the FIDs and IS using pre-trained model, and generated images.

All the implementation is done by using python keras libarary on Fashion MNIST data set and an autoencoder with Pokemon dataset. The models that I used, in this project, to evaluate the performances are given below.

2 Deep Convolutional Generative Adversarial Network (DCGAN)

The GAN model architecture involves two sub-models: a generator model for generating new examples and a discriminator model for classifying whether generated examples are real, from the domain, or fake, generated by the generator model.
During training process, the both sub-models are trained separately. When training the Generator, only Generator loss is used to update the weight the loss function of generator.After training the generator, the discriminator is trained using only the loss of discriminator and and updating the weights. The Loss Function is below.

$$V(D,G) = E_x log(D(X)) + E_z log(1 - D(G(Z)))$$

The DCGAN, in-particular, Involves the following basic struture.

- Replacing any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator)
- Using batchnorm in both the generator and the discriminator.
- Removing fully connected hidden layers for deeper architectures.
- Using ReLU activation in generator for all layers except for the output, which uses tanh.
- Using LeakyReLU activation in the discriminator for all layer

The DCGAN, that is implemented can be seen in the figure, I trained this Model and generated images. After Evaluating the performance of model, I saved 1000 images into **.csv** file, to ease the evaluation procedure. Then these images are used to calculate the Frechet Inception Distance score and Inception Score. The model architectures and images that are obtained after training with random noise can be seen in the following figures. The model was trained upto **100 Epochs**.

Model: "generator"

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 12544)	1254400
batch_normalization_3 (BatchNormalization)	(None, 12544)	50176
leaky_re_lu_5 (LeakyReLU)	(None, 12544)	0
reshape_1 (Reshape)	(None, 7, 7, 256)	0
conv2d_transpose_3 (Conv2DTranspose)	(None, 7, 7, 128)	819200
batch_normalization_4 (BatchNormalization)	(None, 7, 7, 128)	512
leaky_re_lu_6 (LeakyReLU)	(None, 7, 7, 128)	0
conv2d_transpose_4 (Conv2DTranspose)	(None, 14, 14, 64)	204800
batch_normalization_5 (BatchNormalization)	(None, 14, 14, 64)	256
leaky_re_lu_7 (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_transpose_5 (Conv2DTranspose)	(None, 28, 28, 1)	1600

Total params: 2,330,944
Trainable params: 2,305,472
Non-trainable params: 25,472

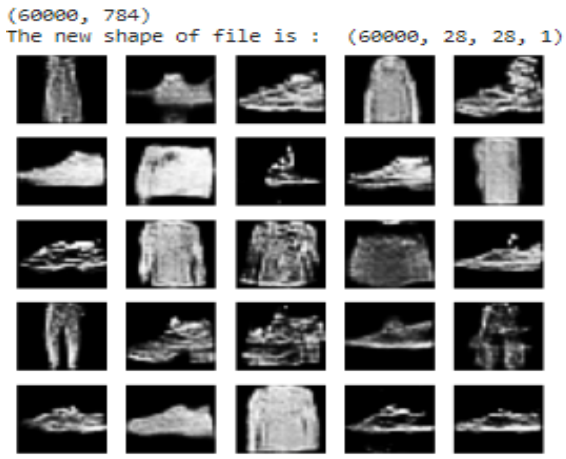
(a)

Model: "discriminator"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 64)	1664
leaky_re_lu_3 (LeakyReLU)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_1 (Conv2D)	(None, 7, 7, 128)	204928
leaky_re_lu_4 (LeakyReLU)	(None, 7, 7, 128)	0
dropout_1 (Dropout)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 1)	6273

Total params: 212,865
Trainable params: 212,865
Non-trainable params: 0

(b)



(c)

Figure 1: (a)Generator (b) Discriminator (c) Generated Images

3 Adversarial AutoEncoder AE

Autoencoder networks teach themselves how to compress data from the input layer into a shorter code, and then uncompress that code into whatever format best matches the original input. I have implemented the **Denoising AE**, which uses a partially corrupted input to learn how to recover the original undistorted input, and try to generated or denoise those set of data. Fashion MNIST dataset is used to train this model.

For this purpose, I need to train AE with two kind of input to the net, noised images and denoised images, AE tries to regenerate the data as close to original. I add some noise to Fashion MNIST dataset with the factor of **0.25** and also used the original data to train AE. The architecture of autoencoder used for this project can be seen into figure 2.

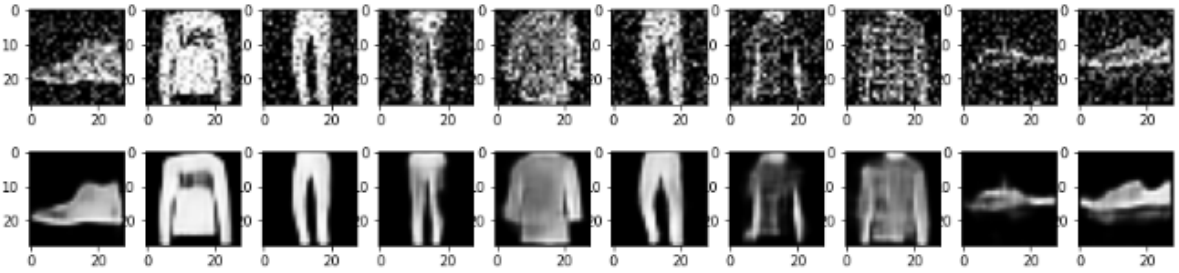
After training the model, I saved those generated images into **.csv** file to evaluated the FIDs and IS. Following figure shows the images regenrated by AE by giving the noised images as input to the trained net.

I also tried the AE on the **Pokemon** dataset with colour images and the model was able to successfully learned the latent space of the orignal data. The results can be seen in the following

```
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_140 (Conv2D)         (None, 28, 28, 32)        320

 max_pooling2d_58 (MaxPoolin  (None, 14, 14, 32)       0
 g2D)

 conv2d_141 (Conv2D)         (None, 14, 14, 16)        4624

 max_pooling2d_59 (MaxPoolin  (None, 7, 7, 16)         0
 g2D)

 conv2d_142 (Conv2D)         (None, 7, 7, 7)           1015

 conv2d_143 (Conv2D)         (None, 7, 7, 7)           448

 up_sampling2d_57 (UpSamplin  (None, 14, 14, 7)        0
 g2D)

 conv2d_144 (Conv2D)         (None, 14, 14, 16)        1024

 up_sampling2d_58 (UpSamplin  (None, 28, 28, 16)       0
 g2D)

 conv2d_145 (Conv2D)         (None, 28, 28, 32)        4640

 conv2d_146 (Conv2D)         (None, 28, 28, 1)         289

=================================================================
Total params: 12,360
Trainable params: 12,360
Non-trainable params: 0
_________________________________________________________________
```

(a)

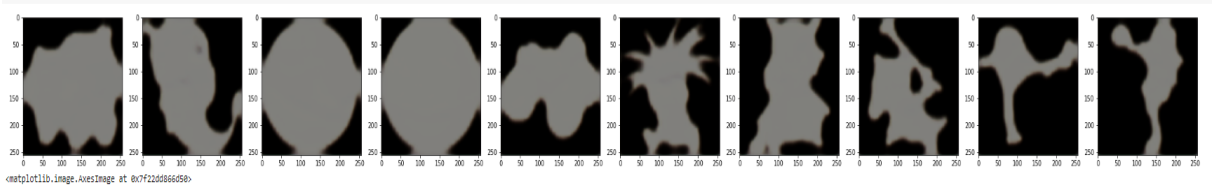
Figure 2: (a)AutoEncoder



(a)

Figure 3: (a)AutoEncoder Generated Images

picture.



(a)

Figure 4: (a)AutoEncoder Generated Images

4 Conditional Generative Adversarial Network (CGAN)

The Basic Architecture of CGAN is a bit different with GAN, starting with the discriminator model, a new second input is defined that takes an integer for the class label of the image. This has the effect of making the input image conditional on the provided class label.

The class label is then passed through an Embedding layer with the size of 50. This means that each of the 10 classes for the Fashion MNIST dataset (0 through 9) will map to a different 50-element vector representation that will be learned by the discriminator model.Next, the generator model must be updated to take the class label. This has the effect of making the point in the latent space conditional on the provided class label.

This model has been taken from a **github** repository and I tired to evaluate the performance through FIDs and IS.

(a)

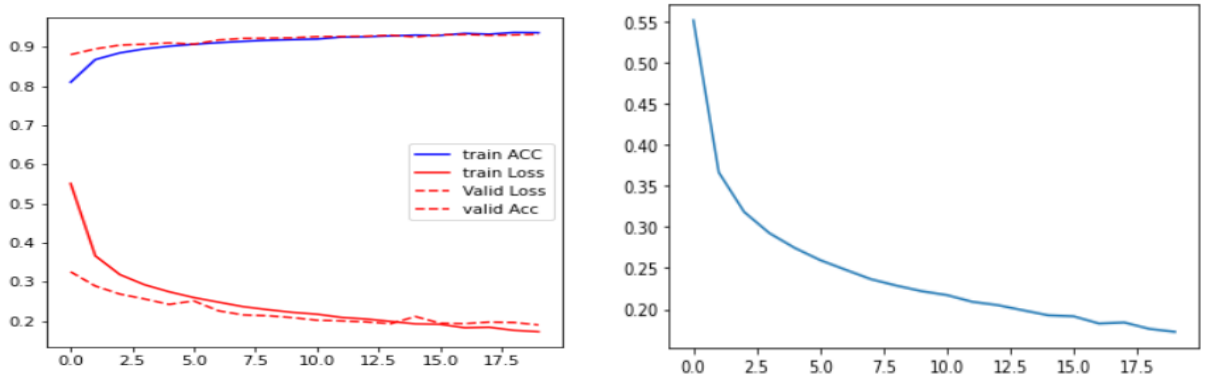
Figure 5: (a)AutoEncoder Generated Images

5 Evaluation Metrics & Performance

Unlike other deep learning neural network models that are trained with a loss function until convergence, a GAN generator model is trained using a second model called a discriminator that learns to classify images as real or generated. Both the generator and discriminator model are trained together to maintain an equilibrium. Instead, a suite of qualitative and quantitative techniques have been developed to assess the performance of a GAN model based on the quality and diversity of the generated synthetic images. To evualuate the performances we need a pre-trained classifier, that is used for feature computation.

Classification Model

Classification model is used to classify the images and it is required to computed the Frechet Inception Distance and Inception Score. For this purpose I have trained the model, suggested in the project guidelines, using fashion MNIST dataset, and both Frechet Distance Score and Inception score is evualted using the same model. I trained the model and save this model as **.h5** format, along with weights. The is used this model to evaluate the performance of GAN models and the architecture can be seen in figure 7. The performance of the model is as follows.



(a) (b)

Figure 6: (a)Training and Validation Accuracy and loss (b) Loss

conv2d (Conv2D)	(None, 28, 28, 32)	320
activation (Activation)	(None, 28, 28, 32)	0
batch_normalization (BatchNormalization)	(None, 28, 28, 32)	128
conv2d_1 (Conv2D)	(None, 28, 28, 32)	9248
activation_1 (Activation)	(None, 28, 28, 32)	0
batch_normalization_1 (BatchNormalization)	(None, 28, 28, 32)	128
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
dropout (Dropout)	(None, 14, 14, 32)	0
conv2d_2 (Conv2D)	(None, 14, 14, 64)	18496
activation_2 (Activation)	(None, 14, 14, 64)	0
batch_normalization_2 (BatchNormalization)	(None, 14, 14, 64)	256
conv2d_3 (Conv2D)	(None, 14, 14, 64)	36928
activation_3 (Activation)	(None, 14, 14, 64)	0
batch_normalization_3 (BatchNormalization)	(None, 14, 14, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
dropout_1 (Dropout)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 512)	1606144
activation_4 (Activation)	(None, 512)	0
batch_normalization_4 (BatchNormalization)	(None, 512)	2048
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 10)	5130

(a)

Figure 7: (a)Classifier

There are alot of evaluation methods that can be used to check the performance of the GAN. Following are two methods that are used, in this project, to evaluate the performance of nets on Real Images and Fake Images. The architecture of the whole Net can be seen in the following picture. Fashion MNIST dataset that is used to train this classification model.

5.1 Frechet Inception Distance score (FIDs):

The FID score is a metric for evaluating the quality of generated images and specifically developed to evaluate the performance of generative adversarial networks. FIDs is calculated by first loading a pre-trained model by removing the output layer of model and the output is taken as the activations from the last pooling layer, a global spatial pooling layer. A feature vector is then predicted for a collection of real images from the problem domain to provide a reference for how real images are represented. Feature vectors can then be calculated for synthetic images.

The result is the collection of two feature vectors for real and generated images. The FIDs is then calculated using the following equation.

$$FID = ||u_r – u_g||^2 + Tr(\sum r + \sum g – 2(\sum r \sum g)^1/2) \quad (1)$$

Where as μ_r and μ_g are the mean of real images and generated images.

For the purpose of computing FIDs, I have loaded the pretained model, **FID_classifier.h5**, and computed the feature vector. To obtained the feature vector,I need to remove some of the last layer of the model ,I used **model.pop()** function to remove the last layers, as the last layer is not required for FID computation. After getting feature vector, I used the FID formula to compute the score using both **REAL** and **Generated** images.

FIDs evaluates the quality of the generated images. Low score corresponds to high quality of generated images.

The obtained FID score for DCGAN, Adversarial AutoEncoder and CGAN are below in the table.

As we can see from the table, FID score obtained for DCGAN is 46.1373 and it is very close to the result obtained according to paper, while it also perform very well on the images generated by

Name	Real	Generated
DCGAN	0	46.1373
AutoEncoder	0	76.3007
CGAN	0	587.1269

Table 1: FID measured on Fashion MNIST

AutoEncoder. During testing, it was also seen that the performance also matters on how many layers we are going to remove from model.

5.2 Inception Score

The IS is an objective metric for evaluating the quality of generated images, specifically synthetic images output by generative adversarial network models. The inception score involves using a pre-trained deep learning neural network model for image classification to classify the generated images.

In particular the IS measure the probability of the image belonging to each class is predicted. These predictions are then summarized into the inception score.

The score seeks to capture two properties of a collection of generated images:

- **Image Quality** check the images look like to which specific object
- **Image Diversity** how much the generated images has the variety.

The inception score has a lowest value of 1.0 and a highest value of the number of classes supported by the classification model. The classification model that I used was trained on Fashion MNIST dataset and it has 10 classes. So the upper score should be 10. The Higher the IS score, the better the generated images. The inception score can be obtained by following equation.

$$IS(G) = exp(E_x D_K L(p(y \mid x) \| p(y)))$$

To compute the Inception score, I have used the same model, that has been trained earlier and also used for FIDs. Unlikely FIDs, the Inception Score is calculated separately on Real and Fake or generated data. The score that is calculated for DCGAN, CGAN and AutoEncoder can be seen in the following table.

Name	Real	Generated
DCGAN	9.3368	5.8469
AutoEncoder	5.0531	4.0411
CGAN	9.4358	6.9150

Table 2: Inception Score measured on Fashion MNIST

The results for DCGAN and CGAN are astonishing while for Autoencoder generated data the model did not peroform very well and the scores are bit low.

References

[1] https://machinelearningmastery.com/

[2] https://keras.io/examples/generative/conditional_gan

[3] https://github.com/eriklindernoren/Keras-GAN/ blob/master/cgan/cgan.py