Deep Convolutional Generative Adversarial Network for Generating Fake Faces

Nikit Naresh Gokhale
Department of Computer and Information Sciences
University of Strathclyde, Glasgow, United Kingdom
nikit.gokhale.2022@uni.strath.ac.uk

***Abstract*—An intriguing new development in machine learning is the use of Generative Adversarial Networks (GAN). To put it simply, GANs are generative models that generate new data instances that are similar to your training data. GANs have demonstrated a viable solution in producing images and videos with realistic appearance. Introduction of Convolution Neural Network (CNN) in GAN have produced new class of deep generative networks for unsupervised learning name Deep Convolutional Generative Adversarial Network (DCGAN). Training DCGAN includes a process of improving the generation of implausible images which are not easy to identify as computer generated image. Experiments with various datasets demonstrate the learning ability of the generator and discriminator in producing a good overall result.**

Keywords—adversarial, convolutional, deep generative, discriminator, generator, images, training

I. Introduction

Generative Adversarial Network (GAN)[1] are types of Deep generative models based on 'Game Theory". GANs are a combination of two deep learning models for generating plausible data when fed with random data distribution. The parts are: **generator** and **discriminator.** The job of the generator is to learn and produce data which the discriminator takes in as input and tells if the input came from real or fake data. Deep Convolution Generative Adversarial Network (DCGAN)[2] is a multi-layer GAN which introduces a class of Convolution Neural Network (CNN) into the generator's and discriminator's network. This paper uses the work from Radford[2] and applies the techniques suggested by Salimans[3] to improve the training of DCGAN using the Anime Face[4] dataset and CelebA[5] dataset for improving the learning performance and the sample generation. Training DCGAN is a tough job, requiring us to find the Nash Equilibrium of the minimax game which generator model and the discriminator model are playing including the high dimensional parameters of the images given as an input for training GANs, however, seeking a Nash Equilibrium may make the algorithm fail to converge; so GANs are typically trained using gradient descent techniques which are intended to encourage convergence of the GANs game.[3] The objective of this paper is to find certain sets of hyperparameters which can be used for generating nearly realistic and vivid images as possible using the DCGAN model. All codes and hyperparameter can be found at https://www.kaggle.com/nikitgoku/dcgan-implementation-pytorch this is referenced with the work of [6].

II. Methodology

A. GAN Structure

The generator and discriminator are both neural networks using CNN architecture removing the fully connected layers and removing the pooling layers with strided convolution and fractional-strided convolution network. The first model; generator takes uniform noise distribution as input for the first layer which is designed to map the latent space vector to data-space. This is to create an image the same size as the training image which is of size 1x28x28. This is achieved through using the transpose of the convolution layer also called as *deconvolution*. The model uses three deconvolution layers, each layer followed with *Batch Normalisation* and *ReLU* activation. Batch normalisation is used to stabilise the learning by normalising the input to each unit for having zero mean and variance. This promotes the gradient to flow deeper into the model.[2] The output layer of the generator uses TanH function which makes sure that the output of the generator matches with the input data range of [-1,1].

On the other hand, the discriminator uses normal convolution layer which takes image as an input and outputs a binary classification of the image being real or fake. The architecture of discriminator also has three convolution layers, each layer followed with batch normalisation for the same purpose of being used in the generator, however it uses *LeakyReLU* as an activation function which makes sure higher resolution modelling is maintained.[2] To produce a binary classification result a *sigmoid activation* function is assigned to the last layer of the discriminator to produce probability like values between 0 and 1.

B. Training inputs

The model used two types of datasets: Anime Face[4] dataset and CelebA[5] seen in Figure 1 and Figure 2 respectively. To make sure that the pixel values for the images are in the range of tanh activation function [-1,1], normalizing the pixel values is needed which was done by resizing and cropping the images to 64x64 pixels with a mean and standard

deviation of 0.5 for each channel, which makes sure that discriminator training is more convenient. For generator input we typically use a tensor of random numbers with a latent shape of 128x1x1 which will be converted into an image tensor of shape 64x64x3. They are referred as *fake images*. The training images from the dataset will also be fed to the generator referred to as *real images.*

Figure 1: Anime Faces dataset

Figure 2: CelebA Faces Dataset

C. Optimizers

In order to achieve proper convergence of GAN while training gradient descent techniques is used here. A similar fashioned stochastic gradient descent is achieved by Adam Optimisation used on both generator and discriminator. The learning rate in the Adam optimiser defaults to 0.001, but as suggested using a learning rate of 0.0002 instead makes sure that more details will be taught to the network and hence requiring more iteration.

The learning process is highly affected with the ***beta1*** and ***beta2*** parameters, as explained [7], beta1 and beta2 parameters controls the acceleration of the learning process. So, using 0.5 and 0.999 for beta1 and beta2 respectively are found to be reasonable.

D. Weight Initialization

As suggested in [2], all the weights are initialized randomly from a zero-centred normal distribution (mean=0) with standards deviation of 0.02. All the weights are applied to the convolutional and deconvolutional layer and a lot to the batch normalisation layers.

E. Training and Hyperparameter Optimisation

Training a GAN or DCGAN to be specific is a tough job. With so many tightly bound hyperparameters, with a very small tweak in the parameters can lead to the whole model collapsing. Referring to the original paper from Goodfellow[1], work from Radford[2], and information provided in Salimans'[3] work, the training process for the model is carried out.

A) Training the Discriminator

The goal of the discriminator is to maximise the equation (1);

$$\log(D(x))+\log(1-D(G(z))) \quad (1)$$

bearing in mind the goal of correctly classifying the given input as 0 or 1 (real or fake). Here the concept of minibatch training is used where batches of real images from the training set is passed, loss is calculated and gradient is calculated in backward pass. When the minibatch processing is avoided, the discriminator processes each sample separately, the gradient of the discriminator will become more unstable leading to the collapse of generator model. The batch size provided in the code is equal to 128 which is a very steady state.

The first part of the discriminator training is to pass this batch of real samples to the discriminator, calculate the former part of the loss equation (1) and then calculate the gradient in the backward pass. In the second part, we pass a batch of fake samples generated by generator to the discriminator, calculate the latter part of the loss equation (1), calculate the gradients with the backward pass, now, call the Adam optimizer for the discriminator

B) Training the Generator

The goal for the generator is to maximise the equation (2);

$$\log(D(G(z))) \quad (2)$$

We use the discriminator as part of the loss function. Computing generators' loss using real labels which we do this because we want to fool the discriminator, computing gradients in backward pass and updating parameters using Adam optimiser, so that the generator gets better at generating false images which fools the discriminator.

We now report the losses calculated by the discriminator and the generator.

C) Hyperparameter optimisation

The training process requires a very sound use of hyperparameters and tuning them is a great challenge in itself. Training our model required a lot of iteration and have to run the training process for several of epochs, however beta1, beta2 and learning rate plays a big part in the tuning of DCGAN as well. Not denying the role of training iteration, but even if we run our training for several of iteration and epochs the result could be the same, but tuning the optimizer to give better results is very important. So, in response to that, the training epochs are kept the same (**epochs=20**), but the hyperparameters for the optimisers are being updated to get a better result. It is recommended that the value for *beta1* should be in the range of **0.2** to **0.5**, as having higher value for beta1 results in instability.[2] Keeping in statement to [2], the experiment with *slope* of the *leakyReLU* was carried out between **0.06** to **0.2**. Starting point of the experiment started with the *batch size* of **32** which went up to **128**. A point to note is that the *batch size* and the learning rate are linked, so if the batch size is small then the gradients will become more unstable and would need to reduce the learning rate. Experimenting with the *kernel size* and *strides* during the initial execution didn't seem having much of an impact, so it was fixed to **4** and **2** respectively. Learning rate worked fine for the value of **0.0002.**

Final sets of hyperparameter are as follows:

- Batch size during training 128
- Latent vector size 100
- Number of epochs 20
- Adam Optimiser; beta1=0.5 and beta2=0.992
- Learning rate 0.0002
- LeakyReLU slope 0.2
- Kernel Size 4 and strides 2

III. Results

As for the result, the visual ailment for the generated images were improved from the initial experiment. The sample generation experiment on the Anime Faces datasets and CelebA dataset was carried. Following is the summarisation for the sample generation

A. *Anime Faces dataset*

The Anime Faces dataset contains 63,632 images. Experiment was performed by randomly picking fraction of these images, considering the batch size. Here we will look at three different results, first we visualise the discriminator and generator losses during training seen in Figure 3 and then we visualise the generated images seen in Figure 4.

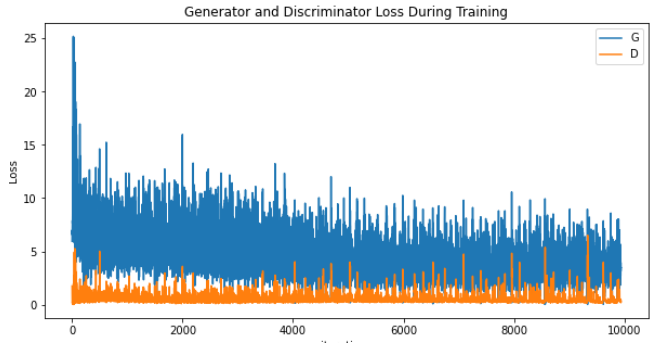

Figure 3: Discriminator and Generator Loss for Anime Faces.

Figure 4: Generated Anime Face Images .

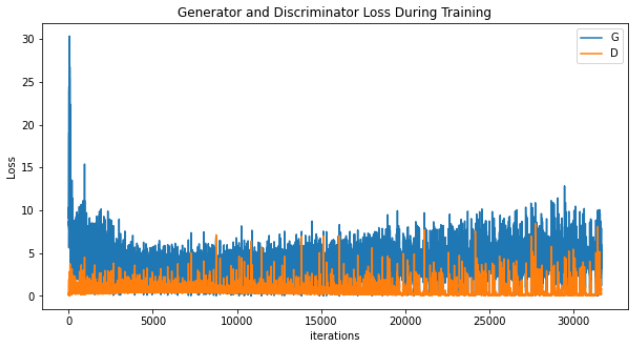

Figure 5: Discriminator and Generator Loss for CelebA.

Figure 6: Generated Celebrity Images.

The loss changes over time and we expect the generator loss to reduce over time, without the discriminator loss getting too high. Both the losses seem to be oscillating but gradually decreasing overtime.

B. *CelebA Dataset*

The CelebA Face dataset t contains 202,599 number of face images of various celebrities. The same experiment was performed by randomly picking fraction of these images, considering the batch size. Here we will look at three different results, first we visualise the discriminator and generator losses during training seen in

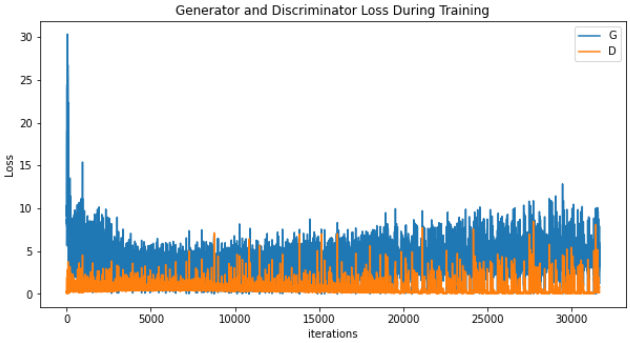

Figure 5 and then we visualise the generated images seen in Figure 6.

IV. Comparison Study

The training on the Anime Faces datasets seems to generating more implausible images with the generator and discriminator loss converging with a proper descent whereas, training on the CelebA image dataset seems to produce good images at certain point but with improper feature recognition and incorrectly combined features to make a proper person. Large dataset can be one of the reasons for the DCGAN to give generally bad result with this requiring more epochs for training as opposed to the Anime Face dataset which contains less images and finds these hyperparameter nearly optimal.

V. Future Work

There have been various studies conducted in finding the optimal hyperparameter values to make the GAN more and more stable with many experiments focussing on re-modelling the generator and discriminator, mathematical studies to improve the training. There have been various formats of GAN developed and are being researched upon for instance, semi-supervised learning[1] to improve the performance of the discriminator when limited data is fed to it. Contribution made to this field makes it more and more interesting in hope of producing more robust system. One can approximately update all the condition and improve the value of the deep generative networks.

References

[1] G Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron C., and Bengio, Yoshua. Generative adversarial nets. NIPS, 2014.

[2] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." arXiv, Jan. 07, 2016. doi: 10.48550/arXiv.1511.06434.

[3] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved Techniques for Training GANs." arXiv, Jun. 10, 2016. doi: 10.48550/arXiv.1606.03498.

[4] "Anime Face Dataset." https://www.kaggle.com/datasets/splcher/animefacedataset (accessed Nov. 27, 2022).

[5] "CelebA Dataset." https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html (accessed Nov. 27, 2022).

[6] "Google Colaboratory." https://colab.research.google.com/github/MicPie/DepthFirstLearning/blob/master/InfoGAN/DCGAN_MNIST_v5.ipynb#scrollTo=qE--CmntVG5o (accessed Nov. 27, 2022).

[7] "(3) DCGAN Hyperparameter Tuning: Part 1 | LinkedIn." https://www.linkedin.com/pulse/dcgan-hyperparameter-tuning-thiago-abreu-da-silva/ (accessed Nov. 27, 2022).