Face Model Generator using Deep Learning

A PROJECT REPORT

Submitted by

BL.EN.U4CSE17092 Nudurupati Prudhvi Raju

BL.EN.U4CSE17105 R Ganesh Phanindra

BL.EN.U4CSE17112 Sai Krishna C R

BL.EN.U4CSE17132 Thania Vivek

in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

AMRITA SCHOOL OF ENGINEERING, BANGALORE

AMRITA VISHWA VIDYAPEETHAM

BANGALORE 560 035

May - 2021

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ENGINEERING, BANGALORE, 560035

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"Face Model Generator using Deep Learning"** submitted by

BL.EN.U4CSE17092	Nudurupati Prudhvi Raju
BL.EN.U4CSE17105	R Ganesh Phanindra
BL.EN.U4CSE17112	Sai Krishna C R
BL.EN.U4CSE17132	Thania Vivek

in partial fulfillment of the requirements for the award of the **Degree Bachelor of Technology** in "**COMPUTER SCIENCE AND ENGINEERING** "is a bonafide record of the work carried out under my guidance and supervision at Amrita School of Engineering, Bangalore.

Ms. Jyotsna C.	Dr. Sriram Devanathan
Assistant Professor	Chairperson,
Computer Science and Engineering	Computer Science and Engineering

This project report was evaluated by us on 26th May 2021.

EXAMINER1

Ms. Sangita Khare

EXAMINER2

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without mention of people who made it possible, and whose constant encouragement and guidance have been a source of inspiration throughout the course of this project work.

We offer our sincere pranams at the lotus feet of **"AMMA", MATA AMRITANANDAMAYI DEVI** who showered her blessing upon us throughout the course of this project work.

We owe our gratitude to **Br. Viswamrita Chaitanya Swamiji**, Director, Amrita School of Engineering, Bangalore.

We thank **Dr. Sriram Devanathan**, Principal, Amrita School of Engineering, Bangalore for his support and inspiration.

It is a great pleasure to express our gratitude and indebtedness to our project guide **Ms. Jyotsna C.**, Assistant Professor, Department of Computer Science and Engineering, Amrita School of Engineering, Bangalore for her valuable guidance, encouragement, moral support and affection throughout the project work.

We express our heartfelt thanks to **Dr. Sriram Devanathan,** Chairperson, Department of Computer Science and Engineering, Amrita School of Engineering, Bangalore for his encouragement and support during the course of work.

We would like to express our gratitude to project panel members for their suggestions, encouragement and moral support during the process of project work. Finally, we are forever grateful to our parents, who have loved, supported and encouraged us in all our endeavours.

ABSTRACT

In recent years, computer vision systems have embraced learning with networks. Unsupervised learning with convolutional neural networks, on the other hand, has gotten less coverage. We hope that this research will contribute to closing the space between the performance of convolutional networks in machine learning. We demonstrate that deep convolutional generative adversarial networks (DCGANs), a class of convolutional neural networks with certain architectural constraints, are for unsupervised learning.

Convolutional networks aid in the discovery of deep correlation within a picture, i.e., spatial correlation. This means that DCGAN is a better choice for image/video data, while GANs can be thought of as a general idea from which DCGANs and a variety of other architectures (CGAN, CycleGAN, StarGAN, and so on) have been created. GANs can be thought of as a two-player (Generator and Discriminator) non-cooperative game in which each player tries to reduce the cost function of the other.

The compelling proof is produced showing that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator by training on various image datasets. We'll also use the learned features of novel tasks to show how they can be used as general image representations.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 - INTRODUCTION

General Adversarial Networks (GANs) are a type of generative model that allows us to produce an entire image in parallel. This network uses unsupervised learning. GANs use differentiable performance represented by a neural network(Generator Network), like most different sorts of generative models. DCGANs are like GANs, however, they use convolutional networks instead of the fully connected networks which are utilized in Vanilla GANs. Deep convolutional generative adversarial networks and deep learning were used to design and create a face generation model. A collection of networks and deep learning are used to build the facial model. In the fields of games and animations, face templates are widely used.

Fig 1.1 Application in-game.

One of the applications of DCGAN is where the actor Keanu Reeves's face model from the movie John Wick was used in a game called Cyberpunk 2077. The face model is shown in fig 1.1.

Fig 1.2 Application in anime characters.

One another famous application is where characters for a new anime are basically created from feature extraction of characters from an old anime. An example of this is shown in fig 1.2.

Companies have made headlines for unintentional diversity-boosting efforts, such as the University of Wisconsin-use Madison's of Photoshop to Photoshop a black man into an all-white throng in an undergraduate booklet, or the superimposing of women into group images of men.

Fig 1.3. Face images. Top 3 images are fake while bottom 3 are real.

Fake faces that look realistic are shown in fig 1.3. These pictures are generates using Artificial Intelligence with chipmakers. There are still some forms of model instability which can be improved with better diversity in the dataset and increased samples. The images are from a website that uses AI to guess which images are fake and which are real.

As we have seen every application, now we move on to some statistical information, as why DCGAN are used.

- GANs, in general, can be defined as a generative model that lets us generate a whole image in parallel.
- Along with several other kinds of generative models, GANs uses a differentiable function represented by a neural network as a Generator Network.
- DCGANs are very similar to GANs but specifically focuses on using deep convolutional networks in place of fully-connected networks used in Vanilla GANs.
- GANs have lot of applications such as Generating examples for Image Datasets,Generate New Human Poses, Face Aging

Using artificial intelligence i.e, AI, a software developer of a kind has constructed and built a famous website that generates synthetic mixed faces (AI). Using the technologies developed by the company Nvidia generates a new lifelike image every time the page is refreshed. In fig 1.3 the new website which is created by the software developer is shown.

Fig 1.4 Fake faces used in dating apps

There are many applications regarding this fake generator. There are more applications where fake faces are being used. Firms are offering diverse photos where they can be used in chatbots and dating apps. Many deals have been made for fake face models with clients.

One of the examples is dating apps where they require women's faces as shown in fig 1.4. Companies require diverse photos. So, one solution is fake people.

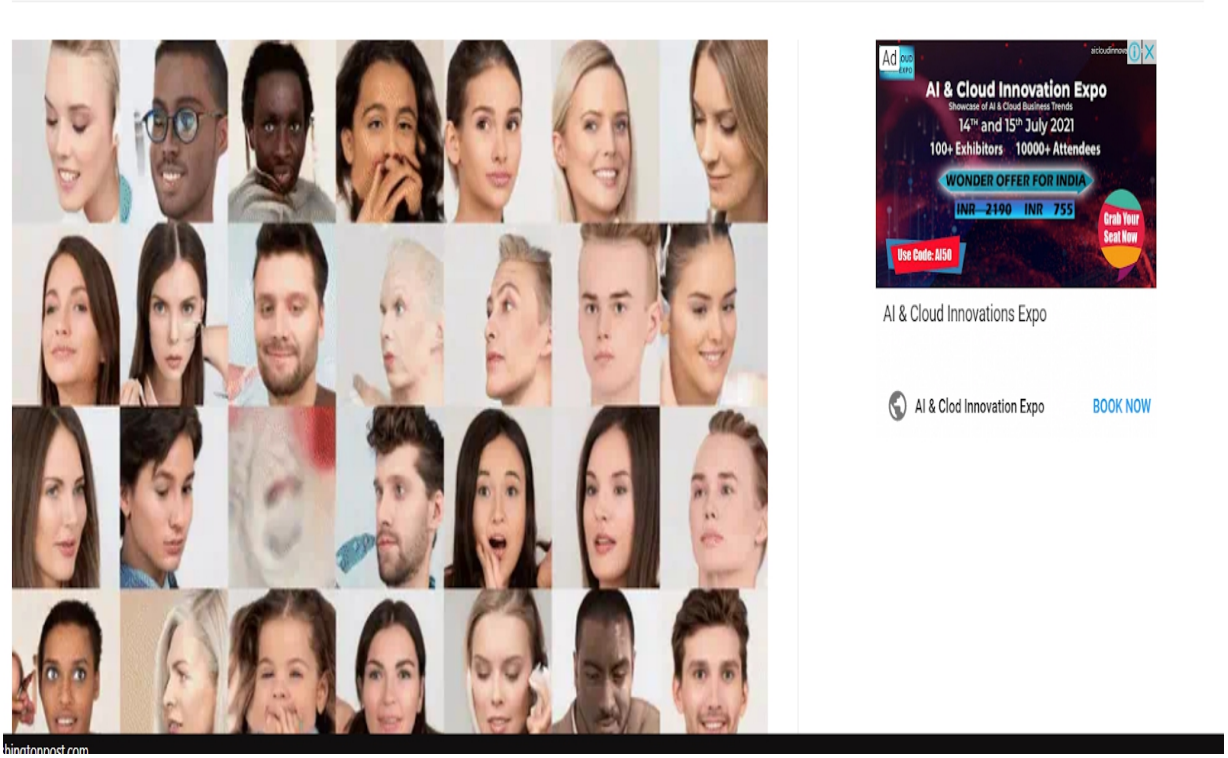

Fig 1.5. Fake images being provided for advertisements

Icons8, an IT company, is offering to sell diverse photos for marketing brochures and dating apps that intend to use the images in a chatbot. They are providing worry-free diversified photos i.e, models on-demand using Artificial Intelligence.

It also has a filter option where they could filter the photos which are fake by all means. The company offered solutions to the clients and they

also provide more diversity and the bias is reduced by including many background noises.

Adversarial algorithms are being used a lot in the security and defence fields. These fake faces are widely used in espionage, that is in spy work. These models help preserve the spy's identity by creating faces that don't exist.

Icons8, an IT company, is offering to sell diverse photos for marketing brochures and dating apps that intend to use the images in a chatbot.

CHAPTER 2 – LITERATURE REVIEW

The resolution and clarity of images generated by generative methods, particularly generative adversarial networks, has lately improved dramatically. Despite recent efforts, the generators continue to operate as black boxes, and understanding of key parts of the picture synthesis process, such as the origin of stochastic features, is still unclear. The latent space's features are similarly poorly known, and the often exhibited latent space interpolations offer no empirical way to compare different generators. The generator starts with a learnt constant input and alters the image's type at each convolution layer depending on the latent coding, allowing it to alter the strength of visual characteristics at various scales directly.

Traditional GAN models lack control over the style of synthetic images they produce. The StyleGAN model's architecture adds various levels of detail, the GAN model introduces control over the style of generated images. When the StyleGAN architecture was used to create synthetic human faces, it produced impressive results [1]. A stable set of architectures for training generative adversarial networks was obtained, which show evidence that adversarial networks learn good representations of images for supervised learning and generative modelling. An open problem is extending this framework to other domains such as video and audio. There are still some forms of model instability that can be improved with better diversity in the dataset and increased samples.

In data-driven unrestricted generative picture modelling, the style-based GAN architecture (StyleGAN) produces cutting-edge outcomes. The architecture's characteristic artefacts are exposed and analysed and changes in the model architecture and training methods have been suggested. The generator normalisation, especially, is improved and regularised to support excellent conditioning in the mapping from latent codes to pictures. In order to enhance image quality, this path length regularizer has the advantage of making the generator much simpler to reverse. This allows you to accurately attribute a produced picture to a certain network [6].

DCGANs have been known to achieve acceptable precision in the end, they are susceptible to instability during training. The RDCGAN, which was inspired by Mode Regularized GANs, shows that adding more feature maps to a model not only improves its efficiency as a representation learning method for supervised tasks but also stabilises the mode in GANs. It has been discovered that using reconstruction error can produce a meaningful error curve that corresponds to the quality of the produced images [2]. For supervised learning and generative modelling, the paper proposes a more stable set of architectures for training generative adversarial networks, and we show that adversarial networks learn good representations of pictures. It was discovered that as models are trained longer, a subset of filters might collapse into a single oscillating mode, indicating that there are forms of model instability.

Class-imbalanced data frequently contain one or more inadequate classes. In such cases, learning algorithms are frequently skewed toward the bulk

of class occurrences. As a result, before performing a classification job, certain changes must be made to the training algorithm or the data itself. One typical strategy used to boost the prevalence of minority classes and equalize the dataset is data augmentation. Simple augmentation strategies, such as applying an affine modification to the data, may not be sufficient in extreme circumstances and frequently fail to capture the variation contained in the dataset. Generating extra sample data collection is utilised in Multiple Fake Class Generative Adversarial Networks (MFC-GAN). We show that the model can create the necessary minority instances by adding numerous fake classes and over-samples.

Methods for directly shaping the intermediate latent space during training will provide interesting avenues for future work. A new training method for GAN is described. The important concept is to gradually increase the quality of both the generator and the discriminator: beginning with a coarse contrast, we introduce additional layers that mimic more precise features as training goes. This accelerates and stabilises the training, enabling us to create photographs of exceptional quality [3]. Improvement in the microstructure of the images by which better photorealism can be achieved.

The framework for predicting generative models using an adversarial approach where two models are trained at the same time has its own advantages and disadvantages. Adversarial networks may benefit statistically from the generator network not being directly updated with data instances, but rather with gradients passing via the discriminator. This implies that input items are not immediately duplicated into the generator's variables. A further benefit of adversarial networks is that

they can depict very precise, even degenerate distributions, whereas Markov chain approaches require the distribution to be slightly hazy in order for the chains to blend among modes [4]. Novel attributes of the face image generation method with generative adversarial networks called AFGAN are used to solve the problem of generating a facial image with specific attributes.

From the above literature surveys, we understand that GANs are composed of two networks, Generator and discriminator. When the models are both multilayer perceptrons, the adversarial modelling framework is the easiest to use [5]. It is shown how the architecture of a generator changes photorealism. We will build Deep Convolutional networks in place of fully connected networks as they look for spatial correlations.

So DCGAN would likely be more fitting for image type data. Finding the proper parameters for the convolutional layers, as well as the project and reshape layers, is the final obstacle.

CHAPTER 3 – REQUIREMENTS AND SPECIFICATIONS

3.1 Software requirements

- Jupyter Notebook

  Jupyter Notebook is a web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. The uses of this application are data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more. We will be using this application for training and running our deep learning model.

- Unit test

  JUnit inspired the unit test unit testing framework, which has a similar flavour to big unit testing systems in other languages. It allows for test automation, the sharing of the test setup and shutdown code, the grouping of tests into sets, and test independence from the reporting framework.

  The unit test framework in Python is called the unit test, which comes packaged with Python. Unit testing makes your code future proof since you anticipate the cases where your code could potentially fail or produce a bug. Though you cannot predict all of the cases, you still address most of them.

Unit test supports important concepts in an object-oriented way such as:

- ➢ Text fixture which represents preparation needed to perform one or more tests, and any associated cleanup actions. This may involve creating temporary or proxy databases, directories, or starting a server process.
- ➢ A test case is an individual unit of testing that can check for a specific response to a set of inputs. A unit test provides TestCase as a base class that can be used to create new test cases.
- ➢ The test suite, A collection of test cases or other test suites as well. The test suite is used to aggregate tests which should be executed together.
- ➢ Test runner, It's a component that plans the execution of tests and provides the outcome. The runner uses a graphical interface, a textual interface. It also returns a special value to show the results of executed tests.

- PyTorch

  It's a tensor library that is optimized, primarily used for Deep Learning applications that use GPUs and CPUs. It is an open-source machine learning library for Python. It uses the scripting language LuaJIT and an underlying C implementation to provide a wide range of deep learning. PyTorch is developed by the Facebook AI Research team. Pytorch comes under the most used python libraries, others being TensorFlow and Keras.

- Pickle

  For serialising and deserializing Python object structures, the pickle module is used. Pickling, serialisation, flattening, and marshalling are all terms for the process of converting any Python object into byte streams of 0s and 1s. Unpickling is a process that converts the byte stream back into Python objects.

  Advantages of using Pickle Module:

  - ➢ Recursive objects: Pickle always keeps track of the objects which have already been serialized, so later references to the same object won't be serialized again.
  - ➢ Object sharing: This is the same as self-referencing objects; pickle stores the object once, and makes sure all other references point only to the master copy. Shared objects always remain shared, which helps mutable objects as it's very important.
  - ➢ User-defined classes and their instances: Compared to Marshall which doesn't support this pickle can save and restore class instances transparently, but the only condition being the definition of the class should be importable and be present in the same module as the module where the object was stored.

CHAPTER 4 – SYSTEM DESIGN

The Generative Adversarial networks mainly work as a two-man game. The generator network is given a noise vector as input. This network, which consists of deep convolutional networks, changes this noise vector into a face model. Then this “fake face” along with an image from the training set is given to the discriminator. The discriminator's work is to identify which among the two images is the fake one. The broad architecture is shown in fig. 4.1.

The generator is trained to reduce the accuracy of the discriminator and the discriminator is trained to improve its prediction accuracy. This way both the discriminator and the generator are learning from each other to beat each other.

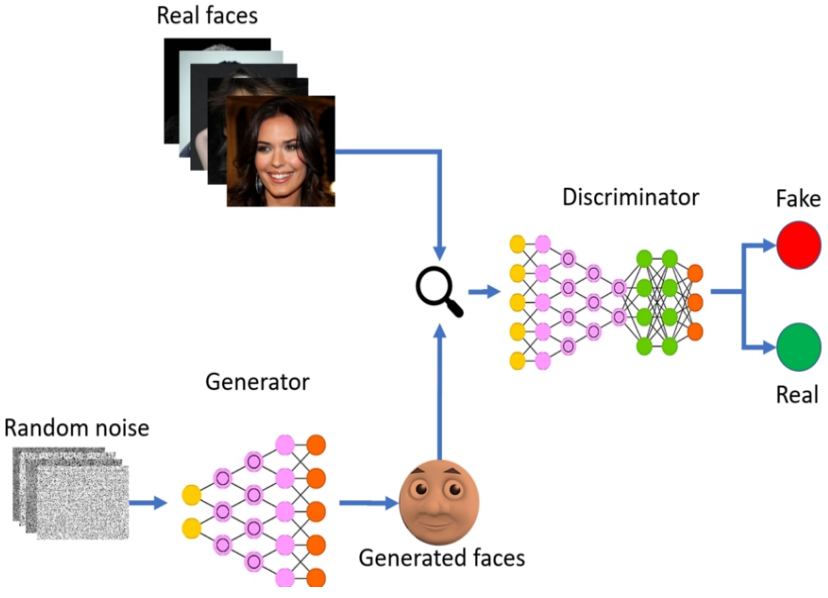

Figure 4.1 system design

4.1 Generator

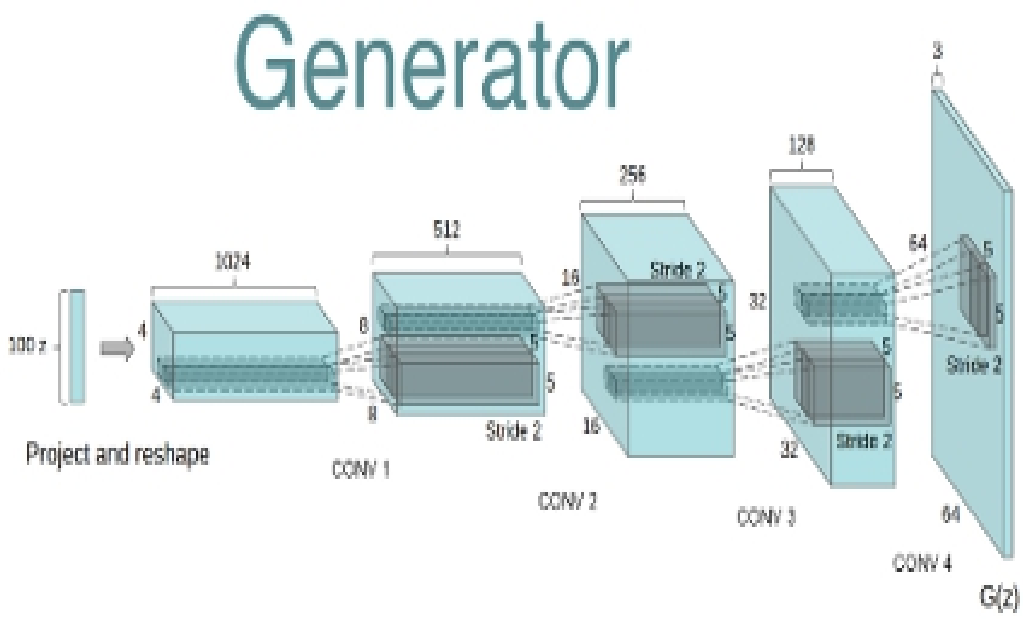

Figure 4.2 Generator Illustration

Random noise is given to the generator network as input and uses a differentiable function to transform and reshape it into a structure that is identical to those images in the dataset (neural network).

Random noise which is used as an input determines the Generator's output. When the models are both multilayer perceptrons, the adversarial modelling framework is the easiest to use. When the Generator Network is applied to a wide range of random input sounds, it generates a wide range of useful output images. The architecture for the generator is shown in fig 4.2.

Every layer the generator has doubled the dimensions, therefore it gets doubled from 1x1 all the way to 32x32. The generator has transpose convolutional layers, after each transpose convolutional layer batch normalization is done except for the last layer which is excluded from it.

The faces which are generated should be of training size 32x32x3. There are 4 layers, all of them are transposed layers. It is defined in such a way that after each layer the width and height become double. We used a leaky ReLU activation function for the hidden units. Feature maps are also calculated.

4.2 Discriminator

Figure 4.3 Discriminator Illustration

A Discriminator network is a classifier network that gives the probability of an image that is true. As a result, The images given to the Discriminator contain 50% real images and 50% fake images generated

from the Generator. The Discriminator's aim is to give true images a probability close to one and false images a probability close to zero.

This is a convolutional classifier, but it doesn't have any MaxPooling layers. Three convolutional layers and a final completely linked layer make up the following architecture, which outputs a digit. This digit determines the genuinity of every image.

The generator, on the other hand, aims to generate fake images to which Discriminator yields a probability close to one. The Discriminator will become more adept at discriminating between true and false images as training progresses. The threshold is the minimum signal level required for the logic pulse to be emitted. As a result, the Generator would be forced to update in order to produce more realistic samples in order to fool the Discriminator The architecture for the discriminator is shown in fig 4.3.

It is basically a classifier that does not have max-pooling layers. It contains convolutional layers and a fully connected layer where normalization happens. optimizers also have been used.

The Discriminator will become more adept at discriminating between true and false images as training progresses. As a result, the Generator would be forced to update in order to produce more realistic samples in order to fool the Discriminator.

Except for the first, each convolution layer is accompanied by normalization which is conducted for every batch. We used a leaky

ReLU activation function for the hidden units. There are inputs and outputs to the discriminator which account for the network class.

We initialise the discriminator module and build the respective architecture and forward propagation of the neural network is done.

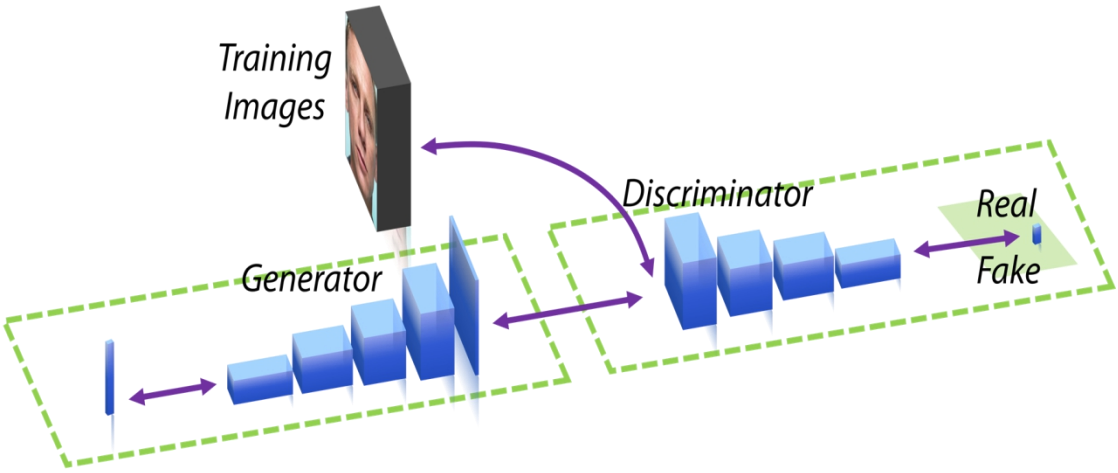

Figure 4.4 Combined Illustration

The working architecture of the model after combining discriminator and generator is shown in fig 4.4. where the generated images are being analyzed in the discriminator and the result is provided.

CHAPTER 5 - IMPLEMENTATION

The implementation phase is divided into several steps, ranging from data loading to identifying and training adversarial networks.

5.1 Libraries

Numpy, problem_unittests, pickle and torch are imported. Unit test cases which is a class in the programme that is prepared for rechecking the test cases which will generate arguments that are useful in testing. pickle is used for serialization and torch is used for importing the deep neural network.

5.2 Data Loader

Our dataset is CelebA dataset, we get the data by cropping them and resizing them into 64*64*3 images of NumPy type. A few images from the dataset are shown in fig 5.1.

get_data function is defined to load the processed data from the images where it has been transformed by resizing, centre cropping and the images are in tensor form.

The images are transformed to centre crop as shown in fig 5.2. From all the batches one batch of training images is obtained and plotted with corresponding labels.

Figure 5.1 Dataset

Figure 5.2 Processed Dataset

The DataLoader returned by our function can shuffle and batch these Tensor images. Dataset also contains a wide range of poses and backgrounds. The batch size is 256 and the image size is 32*32

Below are the steps provided for Data Loading and Transforming:

1. The data transformation is passed into the image folder where the wrapper is used.
2. get_data function is defined to load the processed data from the images where it has been transformed by resizing, centre cropping and the images are in tensor form.
3. After Transforming the images are wrapped into a Python image folder and the whole images are shuffled.
4. After Transforming image data you will load it into a different class.

5.3 Visualization

Data Loading is performed, now we can visualize some sample images. The images are transformed into NumPy type images. A helper display function is created to display the images. From all the batches one batch of training images is obtained and plotted with corresponding labels. The visualization of sample images is shown in fig 5.3

Figure 5.3 Visualisation of sample images

5.4 Preprocessing and scaling

In preprocessing part we are scaling the image ranges, the pixel values are modified within the range of the output activated generator i.e, tanh. So we are re-scaling the images to the feature range

Preprocessing is done, now we can check the scaled range of images which will lie in the range -1 to 1.

Min: tensor(-0.96978)

Max: tensor(0.98691)
which lies between -1 to 1 i.e, feature range.

5.5 Discriminator Network

The Discriminator will become more adept at discriminating between true and false images as training progresses. As a result, the Generator would be forced to update in order to produce more realistic samples in order to fool the Discriminator. This is a convolutional classifier, but it

doesn't have any MaxPooling layers. Three convolutional layers and a final completely linked layer make up the following architecture, which outputs a digit. This digit determines the genuinity of every image.

Except for the first, each convolution layer is accompanied by normalization which is conducted for every batch
We used a leaky ReLU activation function for the hidden units.
There are inputs and outputs to discriminator which account to the network class -

1. input is 32*32 images and 3 dimensions.
2. output is a single digit indicator.

We initialise the discriminator module and build the respective architecture and forward propagation of the neural network is done.

5.6 Generator Network

Random noise which is used as an input determines the Generator's output. When the Generator Network is applied to a wide range of random input sounds, it generates a wide range of useful output images.

In the generator class,

1. input is the length of the vector.
2. output is an image 32*32 and 3 dimensions.

We initialise the generator module and build the respective architecture and forward propagation of the neural network is done.

There are 4 layers, all of them are transposed layers. It is defined in such a way that after each layer the width and height become double. We used leaky ReLU activation function for the hidden units. Feature maps are also calculated.

5.7 Model Architecture

Standard deviation of 0.02 is used to initialise all weights.

Define the models' hyperparameters and use the discriminator and generator classes from the Defining Model section to instantiate them. The bias terms will be set to 0.

Total model is defined here. A representation of the architecture of our model is shown in fig 5.4.

A completely linked layer is followed by four transpose convolution layers with the following architecture. This architecture is designed so that the output of the fourth transpose convolution layer produces a 32X32X3 image.

```
Discriminator(
  (layer_1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
  (layer_2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (layer_3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (fc): Linear(in_features=4096, out_features=1, bias=True)
)

Generator(
  (fc): Linear(in_features=100, out_features=2048, bias=True)
  (layer_1): Sequential(
    (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (layer_2): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (layer_3): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (layer_4): Sequential(
    (0): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
  )
)
```

Figure 5.4 Building the Model

5.8 Training

In the training loss functions, optimizer selection, and eventually the model building are all part of the phase. We trained our model with 40 epochs.

The discriminator and the generator will be trained in turn. To calculate the losses, we'll use the loss function methods that are previously established.

We are training the model on GPU and the model inputs are also moved to GPU. The steps involved in training are

1. store the samples and losses.
2. load the data images.
3. Training the discriminator on real and fake images.
4. Loss statistics.

5.9 Training Loss

Loss from Discriminator:

- Total loss is the sum of fake and real image losses.

Loss from Generator:

- Only the labels will be flipped in the Generator failure. The generator's aim is to persuade the discriminator that the images it generates are genuine.

```
def real_loss(D_out):

    batch_size = D_out.size(0)
    labels = torch.ones(batch_size)
    if train_on_gpu:
        labels = labels.cuda()
    criterion = nn.BCEWithLogitsLoss()
    loss = criterion(D_out.squeeze(),labels)
    return loss

def fake_loss(D_out):

    batch_size = D_out.size(0)
    labels = torch.zeros(batch_size)
    if train_on_gpu:
        labels = labels.cuda()
    criterion =  nn.BCEWithLogitsLoss()
    loss = criterion(D_out.squeeze(),labels)
    return loss
```

Figure 5.4 Losses from Generator and Discriminator

5.10 Optimizers

Optimizers are defined for two networks, we should execute both of them at the same time in order to improve both networks. The optimizer we used is Adam optimizer. Fig 5.5 shows the usage of optimizers for the code. The Adam optimizer is not the same as the traditional stochastic gradient descent algorithm.

```
1 import torch.optim as optim
2
3
4 d_optimizer = optim.Adam(D.parameters(),lr = .0002, betas = [0.5,0.999])
5 g_optimizer = optim.Adam(G.parameters(),lr = .0002, betas = [0.5,0.999])
```

Figure 5.6 Optimizers

For all weight updates, stochastic gradient descent maintains a single learning rate (called alpha), which does not fluctuate during training. Each network weight (parameter) has its own learning rate, which is adjusted individually as learning progresses.

Gradient descent is one of the most widely used optimization methods, and it is by far the most frequent method for optimising neural networks.

Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

CHAPTER 6 – RESULTS AND DISCUSSION

6.1 Training Losses

The training losses for the networks which are generated after each instance are plotted. Fig 6.1 shows the plot of the loss generated.

Figure 6.1 Loss Generation

In the Fig 6.1, training loss fluctuates because of input noise but there is a decrease in the end because it started producing fake images to fool it. So it distinguishes which is original and which is fake. As stated before, the generator's job is to create fake images thereby fooling the discriminator.

6.2 Generated Images

The samples are loaded from the generator network which are generated while training.Fig 6.2 shows the generated fake faces by DCGAN model.

Figure 6.2 Fake Faces generated

Our algorithm was able to create new photos of phoney human faces that were as lifelike as possible. We can also see that all of the photographs are lighter in colour, even the dark faces. This is due to the fact that the CelebA dataset is skewed, as it contains predominantly white "celebrity" faces. DCGAN, on the other hand, successfully generates near-real images from noise.

CHAPTER 7 – CONCLUSION AND FUTURE SCOPE

The model was able to successfully generate fake human images that look similar to real ones. It should be noted that all images, including the brown ones, are lighter in shade. This is because the data set we took is biased as the majority of the images are of celebrities that are white.In the end the model was able to convert noise into realistic human faces using DCGAN successfully.

Our algorithm was able to create new photos of phoney human faces that were as lifelike as possible. We can also see that all of the photographs are lighter in colour, even the dark faces. This is due to the fact that the CelebA dataset is skewed, as it contains predominantly white "celebrity" faces. DCGAN, on the other hand, successfully generates near-real images from noise.

The results can be improved significantly with a more wider range of variety in the dataset. Extending this framework to other domains such as video and audio has not been done.

CHAPTER 8 – REFERENCES

1. Karras, Tero, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4401-4410. 2019.
2. Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." *arXiv preprint arXiv:1511.06434*. 2015.
3. Yuan, Zheng, Jie Zhang, Shiguang Shan, and Xilin Chen. "Attributes Aware Face Generation with Generative Adversarial Networks." In 2020 25th International Conference on Pattern Recognition (ICPR), pp. 1657-1664. IEEE. 2021.
4. Karras, Tero, Timo Aila, Samuli Laine, and Jaakko Lehtinen. "Progressive Growing of GANs for Improved Quality, Stability, and Variation, ICLR 2018." arXiv preprint arXiv:1710.10196. 2018.
5. Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial networks." arXiv preprint arXiv:1406.2661. 2014.
6. Karras, Tero, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. "Analyzing and improving the image quality of stylegan." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8110-8119. 2020.

7. Tariq, Shahroz, et al. "Gan is a friend or foe? a framework to detect various fake face images." *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. 2019.
8. Ali-Gombe, Adamu, Eyad Elyan, and Chrisina Jayne. "Multiple fake classes GAN for data augmentation in face image dataset." *2019 International joint conference on neural networks (IJCNN)*. IEEE, 2019.
9. Marra, Francesco, et al. "Detection of gan-generated fake images over social networks." *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. IEEE, 2018.
10. Huang, Yihao, et al. "FakeLocator: Robust Localization of GAN-Based Face Manipulations." *arXiv preprint arXiv:2001.09598* 3. 2020.
11. Mansourifar, Hadi, and Weidong Shi. "One-shot gan generated fake face detection." *arXiv preprint arXiv:2003.12244* (2020).
12. Yang, Xin, et al. "Exposing gan-synthesized faces using landmark locations." *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*. 2019.