PROJECT REPORT

on

FAKE HUMAN FACE GENERATOR USING GAN

(Generative Adversarial Network)

By

SABA NAZNEEN

(116221539004)

B.SC (MSDs)

Of

OSMANIA UNIVERSITY

Under the guidance of

Mrs. POOJA MAHAJAN

(Lecturer of Data science)

PRIYANKA DEGREE COLLEGE FOR WOMEN

Mehdipatnam Hyderabad-500028

DECLARATION

I , **SABA NAZNEEN**, pursuing **BACHELORS OF DATA SCIENCE,** hereby declare that this project report titled **" Fake Human face generator using GAN"** submitted by us to the **Department of Data science , Osmania University, Hyderabad** is a bonafide work undertaken by us and it is not submitted to any other University or Institute for the award of any degree/diploma/certificate or published any time before.

SIGNATURE OF STUDENT

Priyanka
Degree College for Women

(Affiliated by Osmania University)

@priyanka.wcollege@gmail.com

CERTIFICATION

This is to certify that the project title **fake face generator using GANs (Generative adversarial networks)** submitted in partial fulfillment for the award of **BSC Programme of Department of Data Science, Osmania University, Hyderabad,** was carried out by **SABA NAZNEEN (h.no:116221539004)** Under my Guidance. This has not been submitted to any other University or institute for the award of any degree/diploma/certificate

INTERNAL GUIDE

EXTERNAL

PRINCIPAL

(college stamp)

ACKNOWLEDGEMENT

I would like to thank my project supervisor, **Mrs. POOJA MAHAJAN**, for providing an awful amount of guidance and input throughout the writing of this report.

I feel privileged in extending my earnest obligation to our College principal **Mrs. S.BHAVANI** for providing us this opportunity.

In addition, I'd like to thank my family for the support throughout my final year at university, and for checking over my report.

ABSTRACT

As machine learning is growing rapidly, creating art and images by machine is one of the most trending topics in current time. It has enormous applications in our current day to day life.

The aim of our work is to generate comparatively better real-life fake human faces with low computational power and without any external image classifier.

For that, in this project, we tried to implement our generative adversarial network with two fully connected sequential models, one as a generator and another as a discriminator. Our generator is trainable which gets random data and tries to create fake human faces. On the other hand, our discriminator gets data from the CelebA dataset and tries to detect that the images generated by the generator are fake or real, and gives feedback to the generator. Based on the feedback the generator improves its model and tries to generate more realistic images.

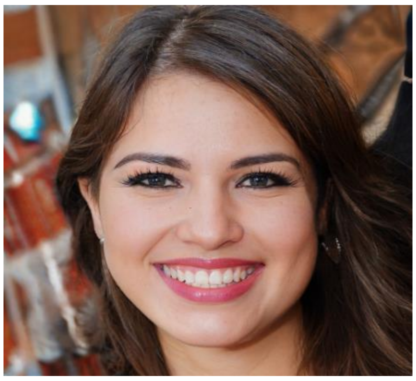
Keywords—Generative adversarial network; fake human faces; generator; discriminator; CelebA dataset

CONTENTS

CHAPTER -1

1.INTRODUCTION

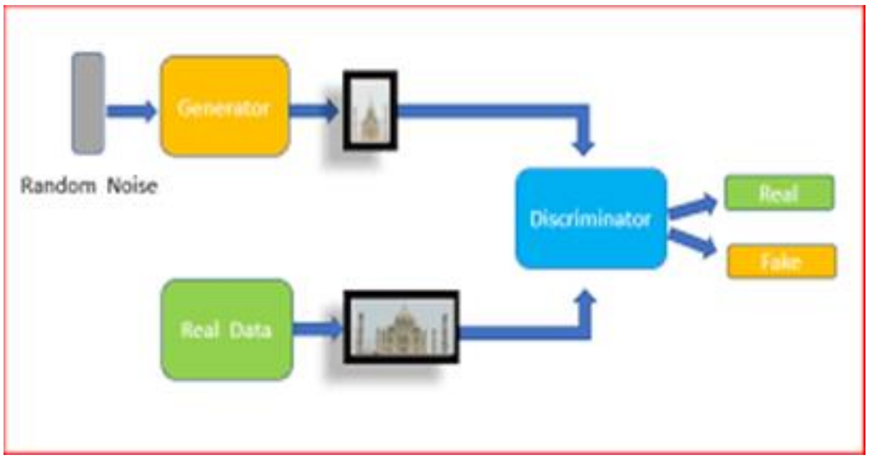
The growth of Generative Adversarial Networks (GANs) is rapid. The continuous advancements in these dual neural network architectures, consisting of a generator model and discriminator, help to stabilize outputs and generate real images, which become almost next to impossible for the human eye to differentiate.



The above image shows the picture of a cheerful woman. Looking at the following image, it would come as a surprise to people that the person does not actually exist. It was generated by GAN, and you can also generate random faces on your own. Each time the generator is run, a completely new realistic face is generated.

Data Scientists use Generative Adversarial Networks (GANs) for a wide range of tasks, with image generation being one of the most common. A particular type of GAN known as DCGAN (Deep Convolutional GAN) has been created specifically for this.

A generative adversarial network (GAN) is a deep learning architecture . It trains two neural networks to compete against each other to generate more authentic new data from a given training dataset. For instance, you can generate new images from an existing image database or original music from a database of songs. A GAN is called *adversarial* because it trains two different networks and pits them against each other. One network generates new data by taking an input data sample and modifying it as much as possible. The other network tries to predict whether the generated data output belongs in the original dataset. The system generates newer, improved versions of fake data values until the predicting network can no longer distinguish fake from original.



How GANs work ?

GANs are typically divided into the following three categories:

- **Generative.** This describes how data is generated in terms of a probabilistic model.
- **Adversarial.** A model is trained in an adversarial setting.
- **Networks.** Deep neural networks can be used as artificial intelligence (AI) algorithms for training purposes.

A GAN typically takes the following steps:

1. The generator outputs an image after accepting random numbers.
2. The discriminator receives this created image in addition to a stream of photos from the real, ground-truth data set.
3. The discriminator inputs both real and fake images and outputs probabilities -- a value between 0 and 1 -- where 1 indicates a prediction of authenticity and 0 indicates a fake.

1.1 PURPOSE :

The goal of GANs is to train two neural networks to compete against each other to generate more authentic new data from a given training dataset. For instance, you can generate new images from an existing image database.

1.2 SCOPE OF THE PROJECT :

GANs have ushered in a new era of generative models, offering unparalleled capabilities in data synthesis. Their ability to generate high-fidelity, realistic data has opened up myriad possibilities across industries, revolutionizing approaches to creativity, research, and development.

1.3 PROPOSED SYSTEM :

For implementing generative adversarial network ,we proposed a generator whose work will be to generate fake faces from the noise and update its own model after receiving the feedback from the discriminator and we also proposed a proper discriminator to identify the real and fake image, so that it give the right feedback, means it says true to real image and false to fake image and based on its feedback the generator will change its model in such a way that eventually it will create almost real like images.

CHAPTER -2

2.LITERATURE REVIEW:

AI applications use GAN networks to develop synthesized images and videos that look like real people.

- Generative adversarial networks for deep generative modelling were introduced by Goodfellow et. al. in the year 2014 which showed that the adversarial training mechanism of GANs proves to be efficient and accurate.

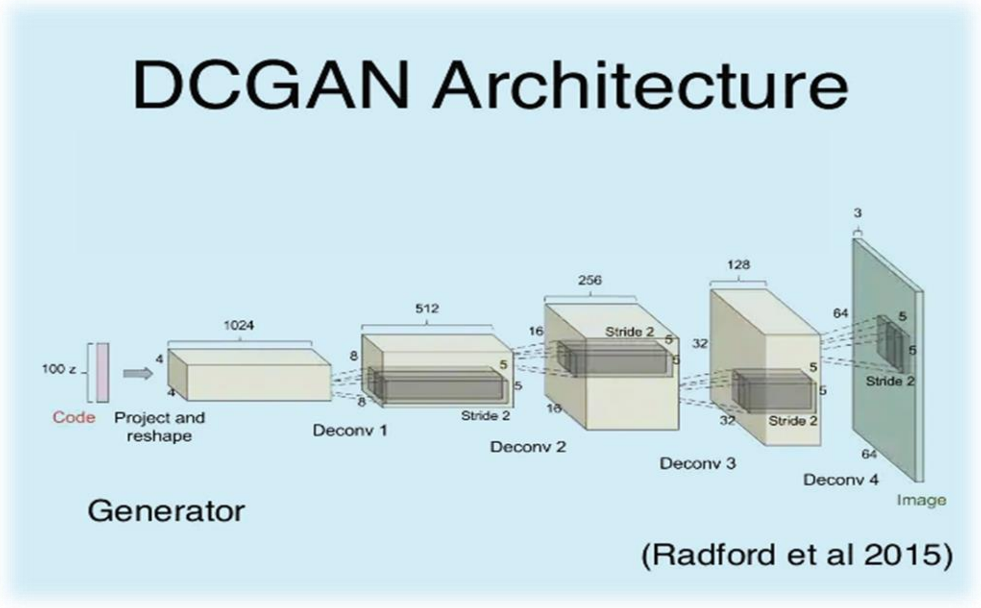
- Wang et al in 2016 tried to simplify the overall process of generative models which gave them more realistic high resolution images as well as highly stable and robust learning procedures. Zhang et al in 2020 discussed an algorithm that can improve the quality of generated images as the quality of the fake images generated by the conventional GAN is limited..

- In recent years, Generative Adversarial Networks have achieved extraordinary results for various applications in many fields especially in image generation because of their ability to create sharp and realistic images.
- While the idea of GAN is simple in theory, it is very difficult to build a model that works. In GAN, there are two deep networks coupled together making backpropagation of gradients twice as challenging.

Deep Convolutional GAN (DCGAN) is one of the models that demonstrated how to build a practical GAN that can learn by itself how to synthesize new images. DCGAN is very similar to *GAN*s but specifically focuses on using deep convolutional networks in place of fully-connected networks used in Vanilla *GAN*s.

Convolutional networks help in finding deep correlation within an image, that is they look for spatial correlation.

DCGAN is one of the popular and successful network design for GAN. It mainly composes of convolution layers without max pooling or fully connected layers. It uses convolutional stride and transposed convolution for the downsampling and the upsampling.



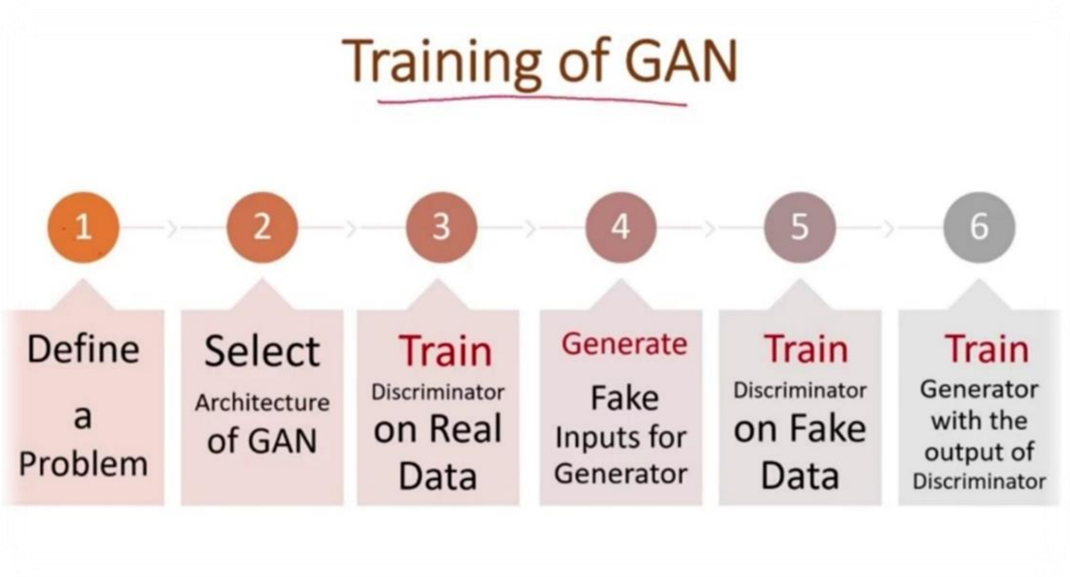
2.1 PROJECT DESCRIPTION :

In our project, we have tried to implement generative adversarial networks in a very classic way with new and latest tools and technologies. we are trying to implement our generative adversarial network to generate fake human faces. Our generator and discriminator will use the keras's classic sequential modeling.

We proposed to use two sequential models, one as a generator and another one as a discriminator to implement our generative adversarial model. For implementing the generator, we have used random noise as input data for the generator. We have also used the Batch Normalization technique and the Reshape module from the keras to resize the data and normalize to train the generator properly. We have used the Leaky ReLU activation function for all the layers except the output layer; in the output layer we have used the tanh activation function for the output layer. We have also used the loss function as the binary cross-entropy for our generator model.

Our generator will receive feedback data from the discriminator and based on the received feedback our generator will be updated. On the other hand, our discriminator is also implemented with the sequential model. However, the main difference between our generators is that after each epoch it does not learn, meaning it will not update or train after each epoch. Our discriminator's main purpose is to identify the real and fake image and give the feedback to the generator, so that we will use the sequential model. In our discriminator's input data we have used both the generator produced data and the real image from the CelebA dataset and it takes both images and tries to identify the real and the fake image.

Here, the model uses a total of four layers. As an input layer we have taken the generated data and the real image data and there are also two hidden dense layers along with an output layer which produce binary values. Other than the output layer we have used the Leaky ReLU activation function in our model and as our output will be classification type, so in the output layer we have used the sigmoid function. For the loss function in our model we have used the binary cross-entropy function.


Training of GAN
1
2
3
4
5
6
Define a Problem
Select Architecture of GAN
Train Discriminator on Real Data
Generate Fake Inputs for Generator
Train Discriminator on Fake Data
Train Generator with the output of Discriminator

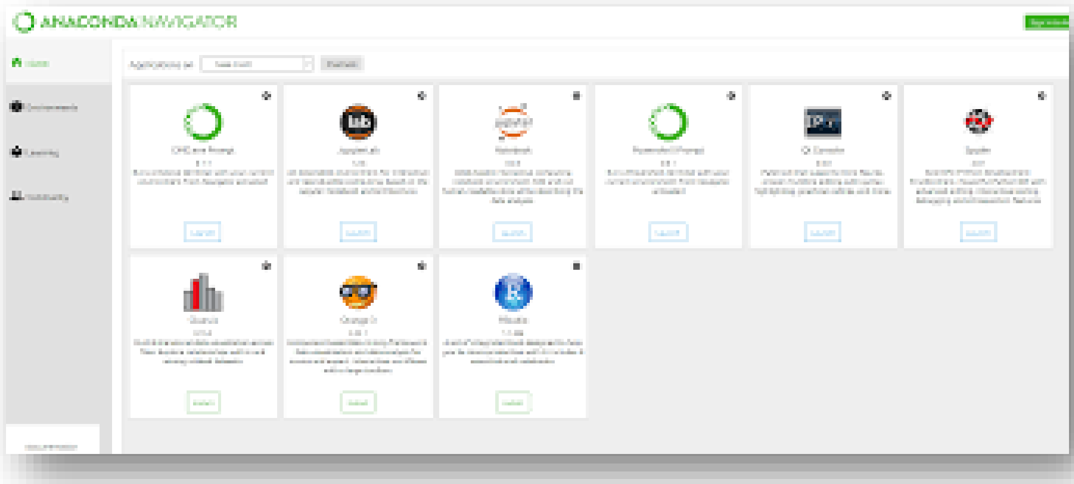
2.2 USER CHARACTERISTICS:

ANACONDA NAVIGATOR :

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda Distribution that allows you to launch applications and manage conda packages, environments, and channels without using command line interface (CLI) commands.

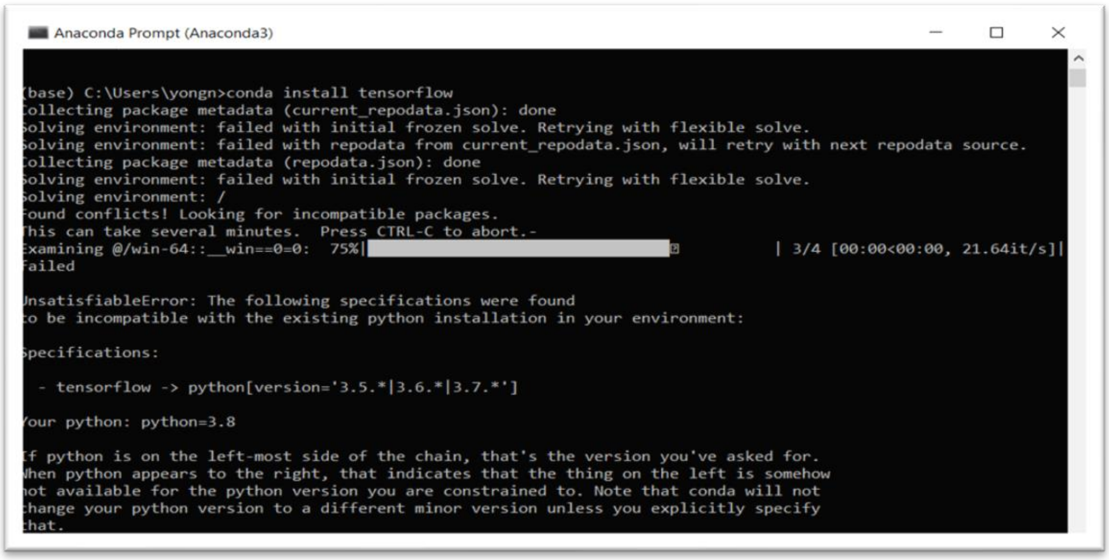
Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.

Navigator is a graphical interface that enables you work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages, and update them –all in navigator.



ANACONDA PROMPT :

Anaconda Prompt is a command-line interface (CLI) that comes with the Anaconda distribution of Python. It provides a way to interact with Anaconda and its associated tools and libraries. Anaconda is a popular distribution for data science and scientific computing, as it includes many pre-installed packages and tools commonly used in these domains.

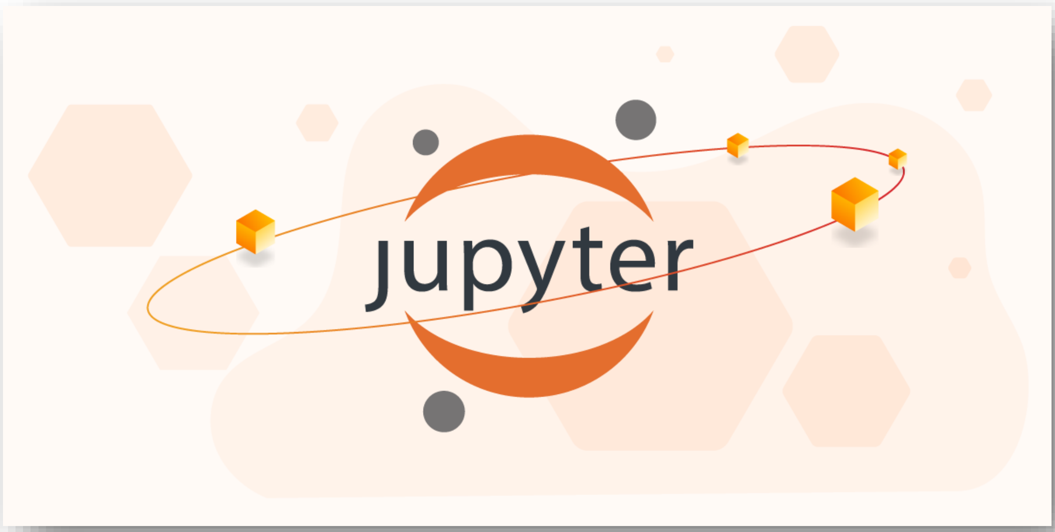


```
(base) C:\Users\yongn>conda install tensorflow
ollecting package metadata (current_repodata.json): done
olving environment: failed with initial frozen solve. Retrying with flexible solve.
olving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
ollecting package metadata (repodata.json): done
olving environment: failed with initial frozen solve. Retrying with flexible solve.
olving environment: /
ound conflicts! Looking for incompatible packages.
his can take several minutes.  Press CTRL-C to abort.-
xamining @/win-64::__win==0=0:  75%|██████████████████████████           | 3/4 [00:00<00:00, 21.64it/s]|
ailed

nsatisfiableError: The following specifications were found
o be incompatible with the existing python installation in your environment:

pecifications:

  - tensorflow -> python[version='3.5.*|3.6.*|3.7.*']

our python: python=3.8

f python is on the left-most side of the chain, that's the version you've asked for.
hen python appears to the right, that indicates that the thing on the left is somehow
ot available for the python version you are constrained to. Note that conda will not
hange your python version to a different minor version unless you explicitly specify
hat.
```

JUPYTER NOTEBOOK :

Jupyter notebooks are especially useful for "showing the work" that your data team has done through a combination of code, markdown, links, and images. They are easy to use and can be run cell by cell to better understand what the code does.

A Jupyter notebook has two components: a front-end web page and a back-end kernel. The front-end web page allows data scientists to enter programming code or text in rectangular "cells." The browser then passes the code to the back-end kernel which runs the code and returns the results.

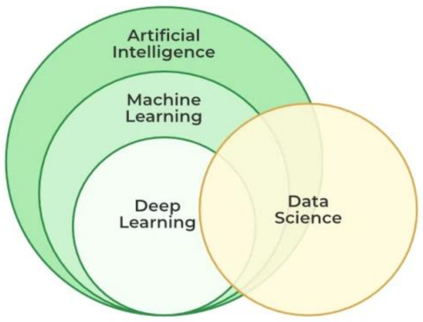


CHAPTER 3

3.BACKGROUND

3.1 DEEP LEARNING:

Deep learning is the branch of machine learning which is based on artificial neural network architecture. An artificial neural network or ANN uses layers of interconnected nodes called neurons that work together to process and learn from the input data.
In a fully connected Deep neural network, there is an input layer and one or more hidden layers connected one after the other. Each neuron receives input from the previous layer neurons or the input layer. The output of one neuron becomes the input to other neurons in the next layer of the network, and this process continues until the final layer produces the output of the network.



Today Deep learning has become one of the most popular and visible areas of machine learning, due to its success in a variety of applications, such as computer vision, natural language processing, and Reinforcement learning.

3.2 HOW DEEP LEARNING WORKS?

Deep learning is a branch of machine learning that is made up of a neural network with three or more layers:

- **Input layer:** Data enters through the input layer.
- **Hidden layers:** Hidden layers process and transport data to other layers.
- **Output layer:** The final result or prediction is made in the output layer.

Neural networks attempt to model human learning by digesting and analyzing massive amounts of information, also known as training data. They perform a given task with that data repeatedly, improving in accuracy each time. It's similar to the way we study and practice to improve skills.

Data Preparation: Deep learning models require labeled data for training. This data is split into training and validation sets.

Architecture Selection: Choose an appropriate neural network architecture (e.g., feedforward, convolutional, or recurrent).

Forward Pass: During training, input data flows through the layers, and predictions are made.

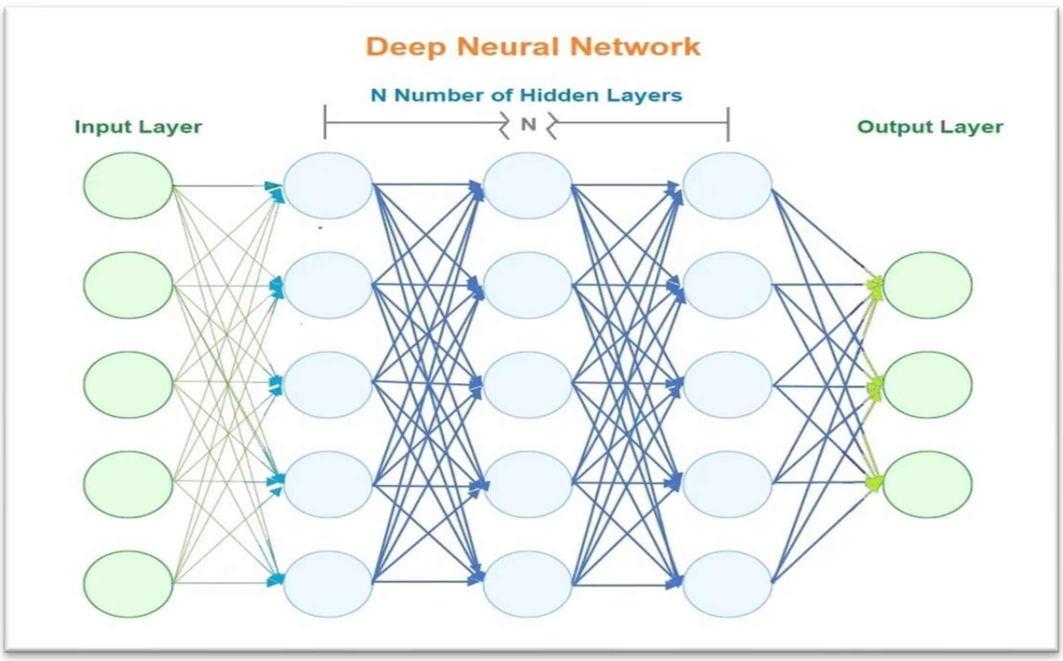
Loss Calculation: The difference between predicted and actual values (the loss) is computed.

Back propagation: The network adjusts its weights and biases to minimize the loss.

Optimization: Algorithms like gradient descent fine-tune the model.

Repeat: This process iterates until the model converges.

Deep learning models can handle complex relationships, making them suitable for tasks like image recognition, natural language understanding, and speech synthesis.



3.3 APPLICATIONS OF DEEP LEARNING:

Image Recognition:

- Identify objects, faces, and scenes in images.
- Applications include self-driving cars, medical imaging, and security surveillance.

Natural Language Processing (NLP):

- Understand and generate human language.
- Used in chatbots, sentiment analysis, and machine translation.

Speech Recognition:

- Convert spoken language into text.
- Powering virtual assistants like Siri and Google Assistant.

Recommendation Systems:

- Personalize content recommendations (e.g., Netflix, Amazon).

Challenges and Future Directions:

- Deep learning requires large labeled datasets and substantial computational resources.
- Researchers are exploring techniques like transfer learning, mechanisms, and explainable AI.

3.4 POPULAR DEEP LEARNING ARCHITECTURES:

Convolutional Neural Networks (CNNs):

- Designed for image analysis.
- Use convolutional layers to extract features from local regions.
- Commonly used in image classification, object detection, and segmentation.

Recurrent Neural Networks (RNNs):

- Process sequences (e.g., time series, text).
- Maintain hidden states to capture context.
- Useful for language modeling, speech recognition, and sentiment analysis.

Generative Adversarial Networks (GANs):

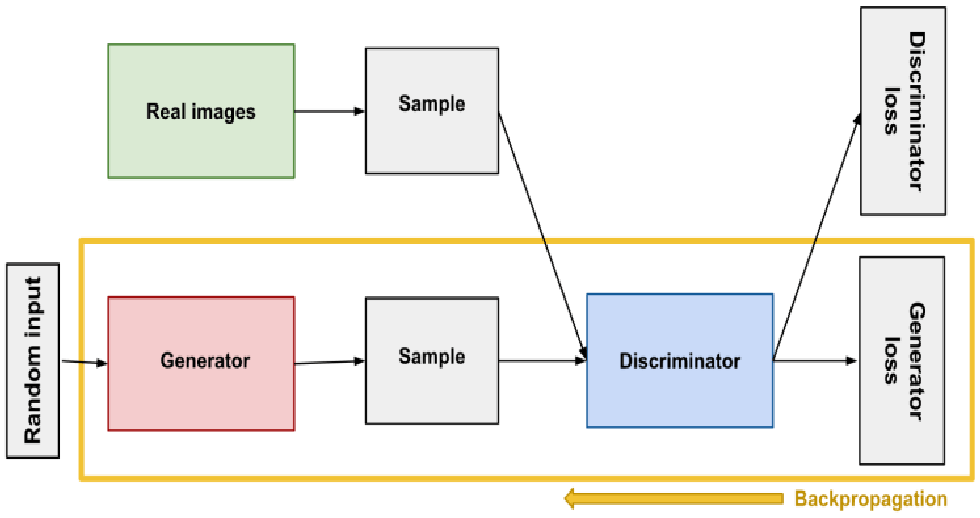
- Consist of a generator and a discriminator.
- Generate realistic data (e.g., images) by learning from real examples.
- Used in art generation, style transfer, and data augmentation.

3.5 WHAT ARE GENERATIVE ADVERSARIAL NETWORKS (GANS)?

Generative adversarial networks are implicit likelihood models that generate data samples from the statistical distribution of the data. They're used to copy variations within the dataset. They use a combination of two networks: generator and discriminator.

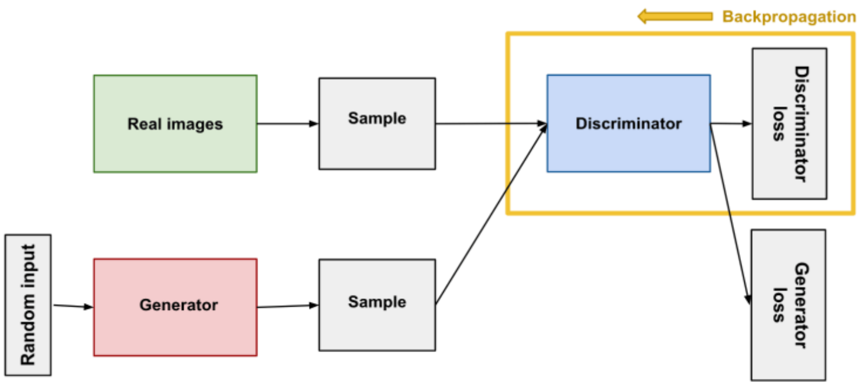
Generator:

A generator network takes a random normal distribution (z), and outputs a generated sample that's close to the original distribution.



Discriminator:

A discriminator tries to evaluate the output generated by the generator with the original sample, and outputs a value between 0 and 1. If the value is close to 0, then the generated sample is fake, and if the value is close to 1 then the generated sample is real.



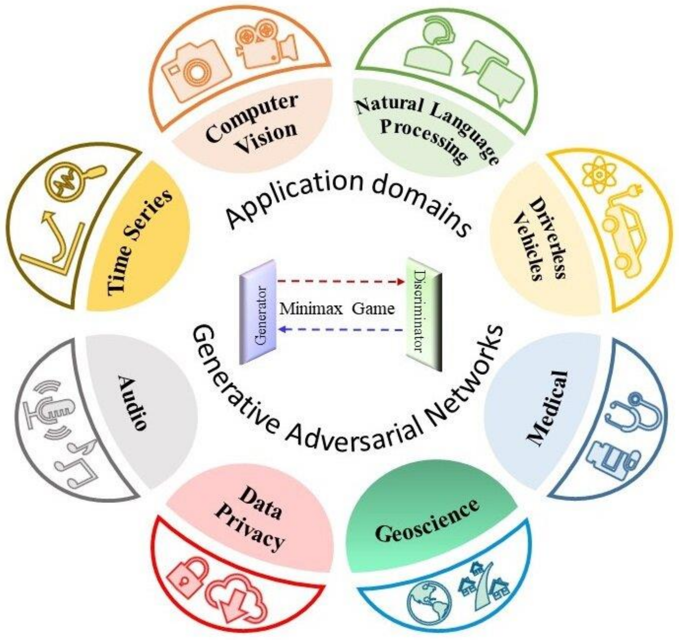
In short, the discriminator's job is to identify whether the generated sample is real or fake by comparing it with the original sample. The generator's job is to fool the discriminator by generating samples that are close to the original sample.

3.6 APPLICATIONS OF GANS :

GANs have a lot of real life applications, some of which are:

- Generate Examples for Image Datasets
  - Generating examples is very handy in medicine or material science, where there's very little data to work with.
- Generate Photographs of Human Faces
  - Video game designers can use this to generate realistic human faces.
- Generate Realistic Photographs
  - Very useful for photographers and videographers.

- Generate Cartoon Characters
    - Artists can use this to create a new character design, or scenes in a cartoon, or even in a video game.
- Image-to-Image Translation
    - Photographers can use these algorithms to convert day into night, summer into winter, etc.
- GANs can be used to simulate a worst-case scenario to optimize risk management in a business.



Other use cases of GAN could be:

- Text-to-Image Translation
- Generate New Human Poses
- Photos to Emojis
- Face Aging
- Super Resolution
- Photo Inpainting
- Clothing Translation
- Video Prediction
- 3D Object Generation

3.7 TYPES OF GANS :

Vanilla GAN :

The Vanilla GAN is the simplest type of GAN made up of the generator and discriminator, where the classification and generation of images is done by the generator and discriminator internally with the use of multi-layer perceptron. The generator captures the data distribution meanwhile, the discriminator tries to find the probability of the input belonging to a certain class, finally the feedback is sent to both the generator and discriminator after calculating the loss function , and hence the effort to minimize the loss comes into picture.

Deep Convolutional GAN (DCGAN):

This is the first GAN where the generator used deep convolutional network , hence generating high resolution and quality images to be differentiated ReLU activation is used in Generator all layers except last one where Tanh activation is used, meanwhile in Discriminator all layers use the Leaky-ReLu activation function. Adam optimizer is used with a learning rate of 0.0002.

Style GAN:

Other GANs focused on improving the discriminator in this case we improve the generator. This GAN generates by taking a reference picture.

CycleGAN:

This GAN is made for Image-to-Image translations, meaning one image to be mapped with another image. For example , if summer and winter are made to undergo the process of Image-Image translation we find a mapping function that could convert summer images into that of winter images and vice versa by adding or removing features according to the mapping function,such that the predicted output and actual output have minimized loss.

Generative Adversarial Text to Image Synthesis :

In this the GANs are capable of finding an image from the dataset that is closest to the text description and generate similar images.

Resolution GAN (SRGAN):

The main purpose of this type of GAN is to make a low resolution picture into a more detailed picture. This is one of the most researched problems in Computer vision.

CHAPTER 4

4.ANALYSIS AND DESIGN

4.1 SYSTEM ANALYSIS

system analysis is a critical process in the development and deployment of deep learning systems. It involves examining all components of the system including hardware, software, data and user interfaces, to ensure that they were together seamlessly and efficiently.

4.1.1 **HARDWARE REQUIREMENTS**

- System: intel 3.10GHZ
- Hard disk: 256GB
- Drivers: NVIDIA GeForce GT 610
- GPU: 2GB
- RAM: 16GB

4.1.2 **SOTWARE REQUIREMENTS**

- Operating system: Windows 10
- IDE: Anaconda Navigator
- Coding Language: Python
- Front End: Jupyter notebook
- Dataset: celebrity faces dataset

4.2 MODULES

Required modules in this project are

- Tensorflow
- Keras
- Numpy
- Os

4.2.1 MODULE DESCRIPTION

Tensorflow:

TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning models. TensorFlow also provides libraries for many other languages, although Python tends to dominate.

TensorFlow can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation)-based simulations.

***Keras*:**

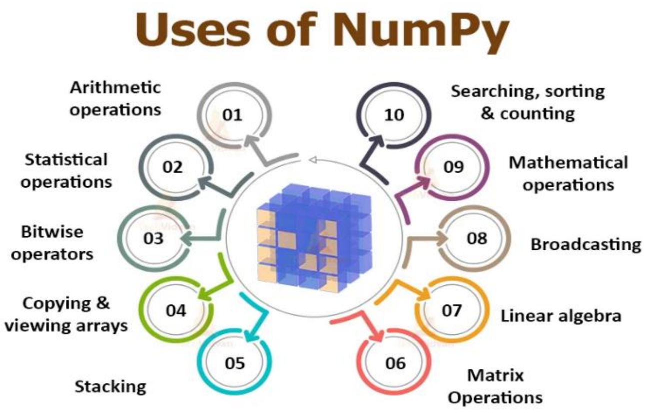
Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation. Keras is relatively easy to learn and work with because it provides a python frontend with a high level of abstraction while having the option of multiple back-ends for computation purposes. Keras allows you to switch between different back ends.

K Keras

A deep learning library

Numpy:

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant.It is an open source project and you can use it freely. NumPy stands for Numerical Python.NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.



Os:

The **OS module in Python** provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality.

Some important functions of the Python os module :

- Handling the Current Working Directory
- Creating a Directory
- Listing out Files and Directories with Python
- Deleting Directory or Files using Python

4.3 DATASET:

We have used the CelebA dataset for our model. In our CelebA dataset there are more than 200k data. At first, we ran our model on this whole dataset but we used the size of images as 48*48. We ran 100000 epochs on the model to get a good result. Normally in generative adversarial models we have to run a lot of epochs to get a good result. Generative adversarial models generally generate new images according to the task given. So, it requires a lot of time to generate the new images. That's why we ran the model 100k epochs.

This dataset is great for training and testing models for face detection, particularly for recognizing facial attributes such as finding people with brown hair, are smiling, or wearing glasses. Images cover large pose variations, background clutter, diverse people, supported by a large number of images and rich annotations.

The dataset can be downloaded from Kaggle. Our objective is to create a model capable of generating realistic **human images that do not exist in reality.**

4.4 PROGRAMMING LANGUAGES

Tensorflow and Keras, which are the two libraries of the Python programming language, have been used in our model. Nowadays Python is the most popular programming language for implementing Machine Learning and Deep Learning models as it has many collections of packages which are very helpful to implement the different Machine Learning and Deep Learning models. Python is also the most preferred language because it is very easy to learn and implement.

Python has many free and open source libraries for Machine Learning and Deep Learning models. Tensorflow is one of them. Keras is an open source software library which supports a Python interface for artificial neural networks. Keras plays the role as an interface for the Tensorflow library.

4.5 MODEL IMPLEMENTATION

Importing the essential libraries:

The starting step of most projects is to import all the essential libraries that we will utilize for the development of the required project. For the construction of this face generation model, we will utilize the TensorFlow and Keras deep learning frameworks for achieving our goals. Apart from these fabulous frameworks, we will also utilize other essential libraries, such as matplotlib, numpy, and OpenCV, for performing visualizations, computations, and graphics-related operations accordingly.

Exploring the data:

In this step, we will load the data into a Dataset format and map the details accordingly. The pre-processing class in the Keras allows us to access the flowing of data from a particular directory so that we can access all the images stored in the particular folder. Ensure that once you extract the celeb dataset containing over 200000 images, you place the extracted directory into another directory titled "Images" or any other folder name of your choice. The dataset will flow from the following directory, and we will assign the image size of 64 x 64 for the computation of the following data

The successful execution of the code cell block should show a result similar to the following statement - "Found 202599 files belonging to 1 classes." You can choose to randomly display an image to check if you have loaded the dataset precisely as planned.

Next step is to create a Generator:

The generator goes the other way: It is the artist who is trying to fool the discriminator. This network consists of 8 convolutional layers. Here first, we take our input, called gen_input and feed it into our first convolutional layer. Each convolutional layer performs a convolution and then performs batch normalization and a leaky ReLu as well. Then, we return the tanh activation function.

Next, create a Discriminator:

The discriminator network consists of convolutional layers the same as the generator. For every layer of the network, we are going to perform a convolution, then we are going to perform batch normalization to make the network faster and more accurate and finally, we are going to perform a Leaky ReLu.

Define a GAN Model:

Next, a GAN model can be defined that combines both the generator model and the discriminator model into one larger model. This larger model will be used to train the model weights in the generator, using the output and error calculated by the discriminator model. The discriminator model is trained separately, and as such, the model weights are marked as not trainable in this larger GAN model to ensure that only the weights of the generator model are updated. This change to the trainability of the discriminator weights only affects when training the combined GAN model, not when training the discriminator standalone.

This larger GAN model takes as input a point in the latent space, uses the generator model to generate an image, which is fed as input to the discriminator model, then output or classified as real or fake.

Since the output of the Discriminator is sigmoid, we use binary cross-entropy for the loss. RMSProp as an optimizer generates more realistic fake images compared to Adam for this case. The learning rate is 0.0001. Weight decay and clip value stabilize learning during the latter part of the training. You have to adjust the decay if you want to adjust the learning rate.

GANs try to replicate a probability distribution. Therefore, we should use loss functions that reflect the distance between the distribution of the data generated by the GAN and the distribution of the real data.

Rather than just having a single loss function, we need to define three: The loss of the generator, the loss of the discriminator when using real images and the loss of the discriminator when using fake images. The sum of the fake image and real image loss is the overall discriminator loss.

Training the GAN model:

Training is the hardest part and since a GAN contains two separately trained networks, its training algorithm must address two complications:

- GANs must juggle two different kinds of training (generator and discriminator).
- GAN convergence is hard to identify.

As the generator improves with training, the discriminator performance gets worse because the discriminator can't easily tell the difference between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy. In effect, the discriminator flips a coin to make its prediction.

This progression poses a problem for convergence of the GAN as a whole: the discriminator feedback gets less meaningful over time. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its quality may collapse

CHAPTER 5

5. Implementation

5.1 CODING:

```
import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

import os
```

```
directory = 'Downloads/proj/Celebrity_Faces_Dataset'

# Get all images from dir

Dataset=tf.keras.preprocessing.image_dataset_from_directory(directory=dir
ectory, color_mode='rgb',   batch_size=128, image_size=(32,32),
label_mode=None,shuffle=True, seed=42)

# Normalize the data

dataset=dataset.map(lambda x: x /255.00)
```

```
#Grid of image

from mpl_toolkits.axes_grid1 import ImageGrid
```

```
it=iter(dataset)

one_batch = next(it).numpy()

images = one_batch[:16,:,:,:]

# show the image

fig = plt.figure(figsize=(10,10))

grid = ImageGrid(fig, 111, nrows_ncols=(4,4),

        axes_pad=0)
```

```
for ax,im in zip(grid, images):
    ax.imshow(im)
plt.show()
```

```
from tensorflow.keras.layers import Conv2D,
LeakyReLU,BatchNormalization, Dropout, Flatten, Dense, Activation,
Reshape
from tensorflow.keras import Model, Input


def discriminator_model():
    disc_input = Input(shape=(32,32,3), name='discriminator_network')
    x=Conv2D(filters=64, kernel_size=3, strides=(2,2),
padding='same')(disc_input)
    x=LeakyReLU()(x)
    x=Dropout(0.2)(x)

    x=Conv2D(filters=128, kernel_size=3, strides=(2,2), padding='same')(x)
    x=LeakyReLU()(x)
    x=Dropout(0.2)(x)

    x=Conv2D(filters=128, kernel_size=3, strides=(2,2), padding='same')(x)
    x=LeakyReLU()(x)
    x=Dropout(0.2)(x)

    x=Conv2D(filters=64, kernel_size=3, strides=(2,2), padding='same')(x)
    x=LeakyReLU()(x)
    x=Dropout(0.2)(x)


    x=Flatten()(x)
    x=Dense(1)(x)


    output = Activation('sigmoid')(x)
    disc=Model(inputs = disc_input, outputs=output)
    return disc
```

```
disc_model = discriminator_model()
disc_model.summary()
```

```
def generator_model(z_dim=100):
    gen_input = Input(shape=(z_dim), name='generator_network')


    x = Dense(8*8*3)(gen_input)
    x = Reshape(target_shape=(8,8,3))(x)
    x = BatchNormalization()(x)
    x = LeakyReLU()(x)


    x = Conv2DTranspose(filters=512, kernel_size=3, strides=(1,1),
padding='same')(x)
    x = BatchNormalization()(x)
    x = LeakyReLU()(x)
```

```
   x = Conv2DTranspose(filters=256, kernel_size=3, strides=(2,2),
padding='same')(x)
   x = BatchNormalization()(x)
   x = LeakyReLU()(x)


   x = Conv2DTranspose(filters=128, kernel_size=3, strides=(1,1),
padding='same')(x)
   x = BatchNormalization()(x)
   x = LeakyReLU()(x)


   x = Conv2DTranspose(filters=128, kernel_size=3, strides=(2,2),
padding='same')(x)
   x = BatchNormalization()(x)
   x = LeakyReLU()(x)


    x = Conv2DTranspose(filters=64, kernel_size=3, strides=(1,1),
padding='same')(x)
   x = BatchNormalization()(x)
   x = LeakyReLU()(x)


   x = Conv2DTranspose(filters=3, kernel_size=3, strides=(1,1),
padding='same')(x)
   fake_images_gen = LeakyReLU()(x)


   model_gen=Model(inputs = gen_input, outputs=fake_images_gen)


   return model_gen
```

```
LATENT_DIM = 32
CHANNELS = 3

def create_generator():
    gen_input = Input(shape=(LATENT_DIM, ))

    x = Dense(128 * 16 * 16)(gen_input)
    x = LeakyReLU()(x)
    x = Reshape((16, 16, 128))(x)

    x = Conv2D(256, 5, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2D(512, 5, padding='same')(x)
    x = LeakyReLU()(x)
    x = Conv2D(512, 5, padding='same')(x)
    x = LeakyReLU()(x)
    x = Conv2D(CHANNELS, 7, activation='tanh', padding='same')(x)


    generator = Model(gen_input, x)
    return generator
```

```
generator = create_generator()
generator.summary()
```

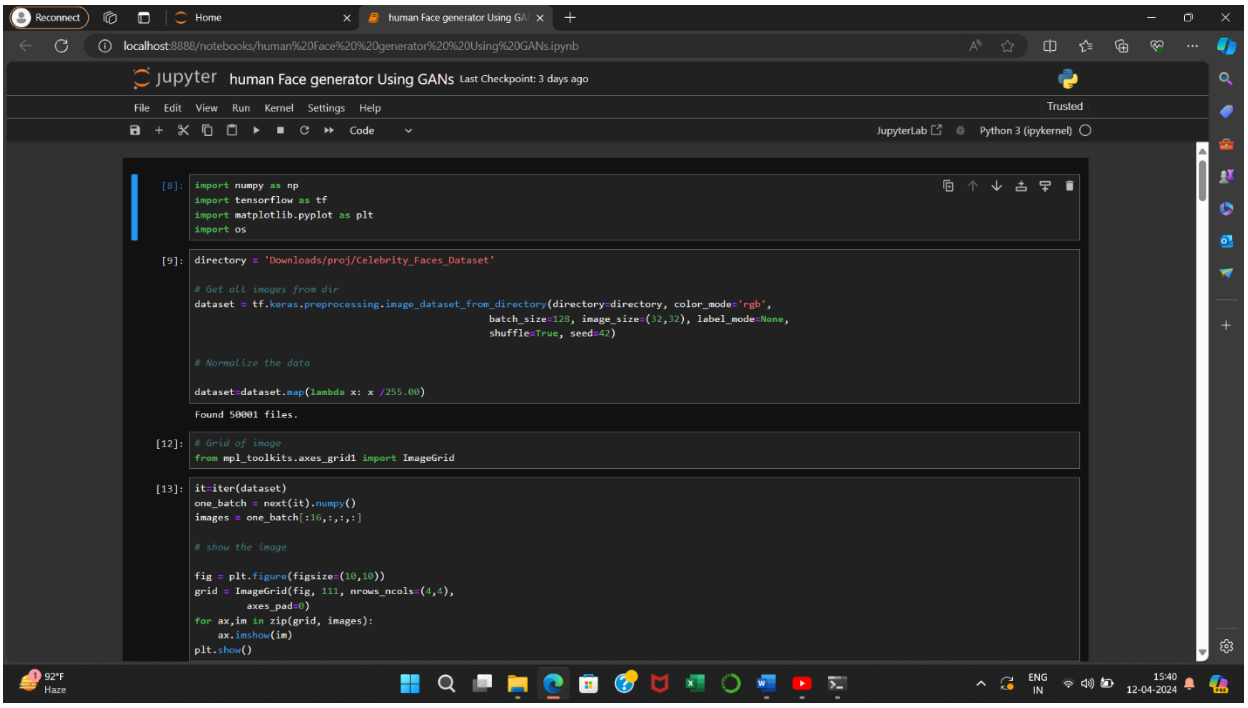
```
disc_model = discriminator_model()
generator =  create_generator()
# Image Classification CNN
# Compile the model that trains disc.
disc_model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['
accuracy'])
```

```
def train_discriminator(batch, batch_size):
    valid = np.ones((batch_size, 1))
    fake = np.zeros((batch_size, 1))
    disc_model.train_on_batch(batch, valid)
    noise = np.random.normal(0,1, (batch_size, z_dim))
    gen_image = gen_model.predict(noise)
    disc_model.train_on_batch(gen_image, fake)
def train_generator(batch_size):
    valid = np.ones((batch_size, 1))
    noise = np.random.normal(0,1, (batch_size, z_dim))
    model.train_on_batch(noise, valid)
```

```
def save_image(fake_image, path):
    image = tf.keras.preprocessing.image.array_to_img(fake_image.numpy())
    tf.keras.preprocessing.image.save_img(path, image)
    return image
def generate_and_save_image(path):
    noise = np.random.normal(0,1, (1, z_dim))
    fake_images = gen_model(noise)
    image = save_image(fake_images[0], path)
    plt.imshow(image)
    plt.show()
```

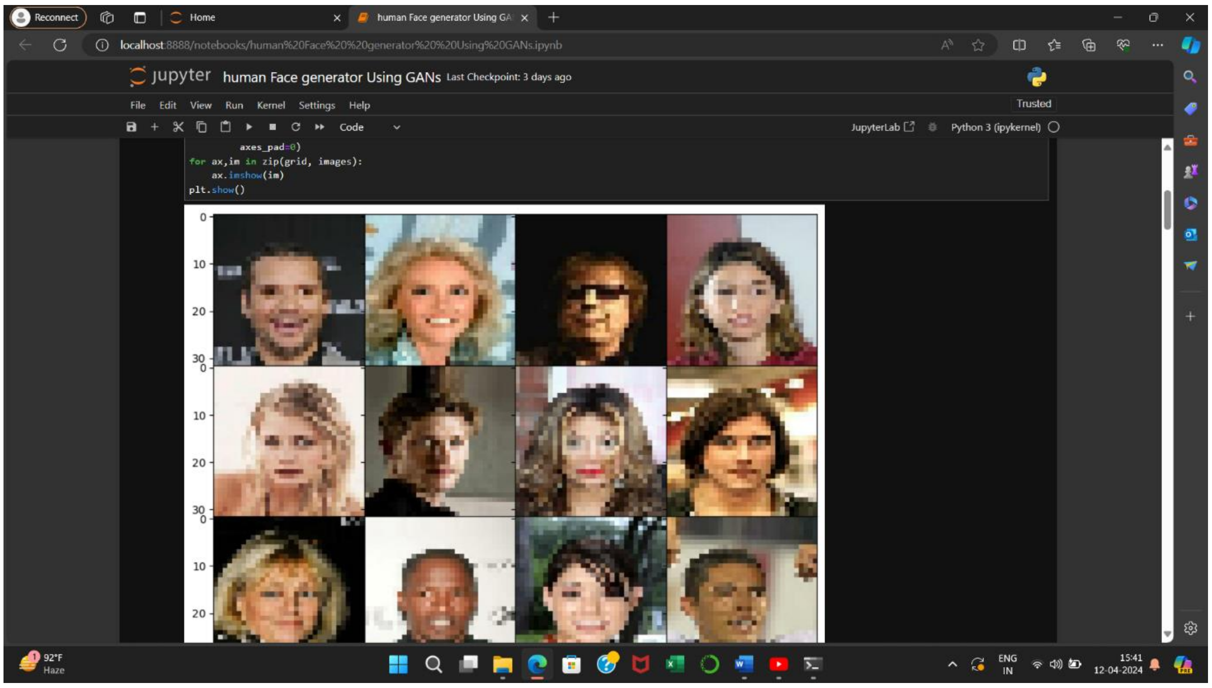
```
for epochs in range(10):
    print(f'No of Epochs-> {epochs}')
    for i,batch in enumerate(dataset):
        train_discriminator(batch, batch.shape[0])
        train_generator(batch.shape[0])
        if i%100==0:
            print(f'Batch Images-> {i}')
            PATH = 'generate_face'
            generate_and_save_image(path =
f'{PATH}/epochs_{epochs}batch_{i}.jpg')
```

5.2 SCREENSHOTS:



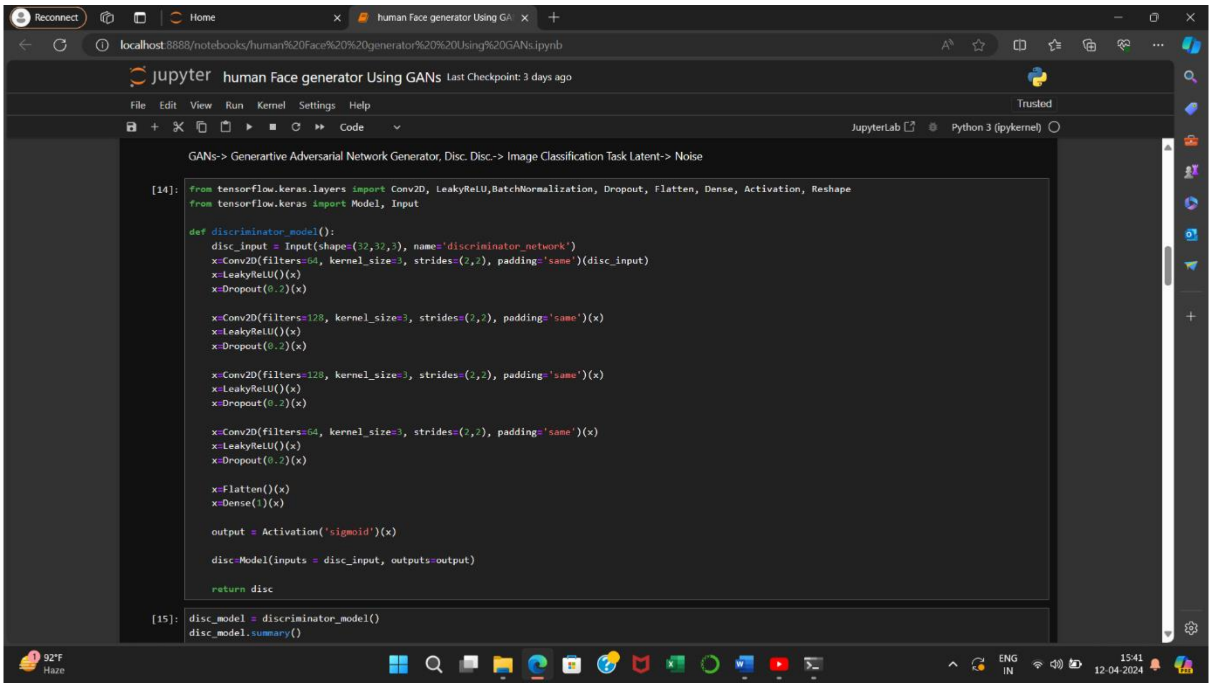
localhost:8888/notebooks/human%20Face%20%20generator%20%20Using%20GANs.ipynb
jupyter human Face generator Using GANs Last Checkpoint: 3 days ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
[8]: import numpy as np
     import tensorflow as tf
     import matplotlib.pyplot as plt
     import os

[9]: directory = 'Downloads/proj/Celebrity_Faces_Dataset'

     # Get all images from dir
     dataset = tf.keras.preprocessing.image_dataset_from_directory(directory=directory, color_mode='rgb',
                                                                   batch_size=128, image_size=(32,32), label_mode=None,
                                                                   shuffle=True, seed=42)

     # Normalize the data

     dataset=dataset.map(lambda x: x /255.00)

     Found 50001 files.

[12]: # Grid of image
      from mpl_toolkits.axes_grid1 import ImageGrid

[13]: it=iter(dataset)
      one_batch = next(it).numpy()
      images = one_batch[:16,:,:,:]

      # show the image

      fig = plt.figure(figsize=(10,10))
      grid = ImageGrid(fig, 111, nrows_ncols=(4,4),
               axes_pad=0)
      for ax,im in zip(grid, images):
          ax.imshow(im)
      plt.show()
```



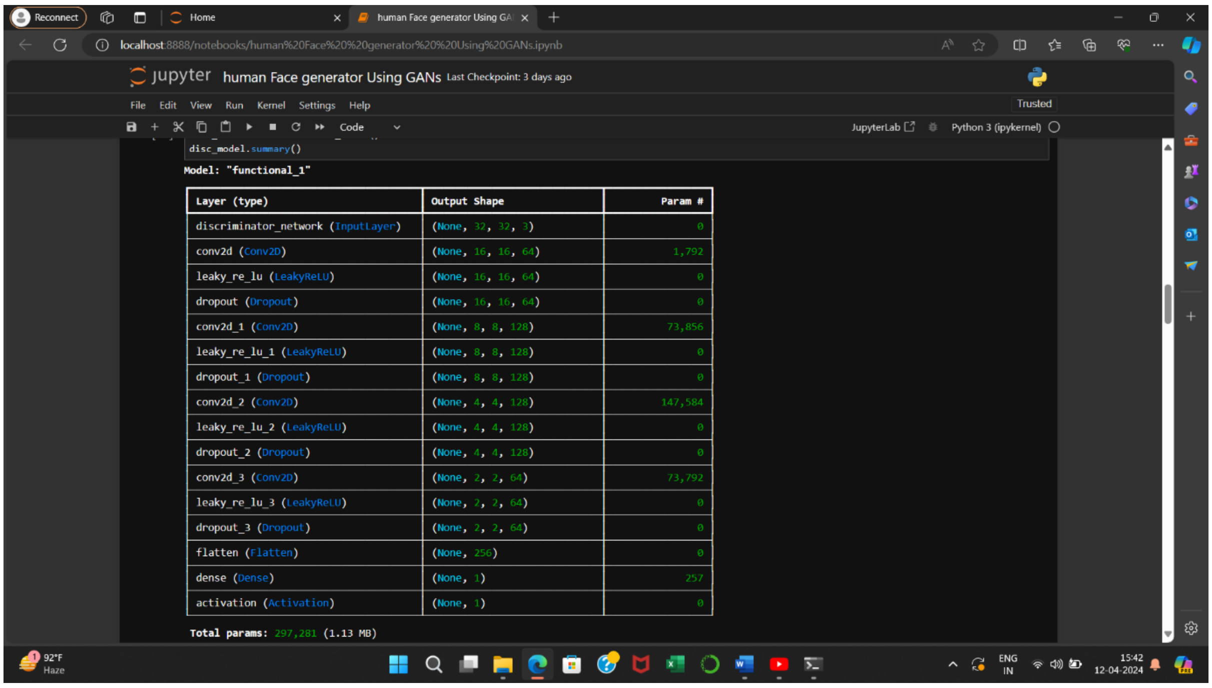
localhost:8888/notebooks/human%20Face%20%20generator%20%20Using%20GANs.ipynb
jupyter human Face generator Using GANs Last Checkpoint: 3 days ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
0
10
20
30

```
               axes_pad=0)
      for ax,im in zip(grid, images):
          ax.imshow(im)
      plt.show()
```

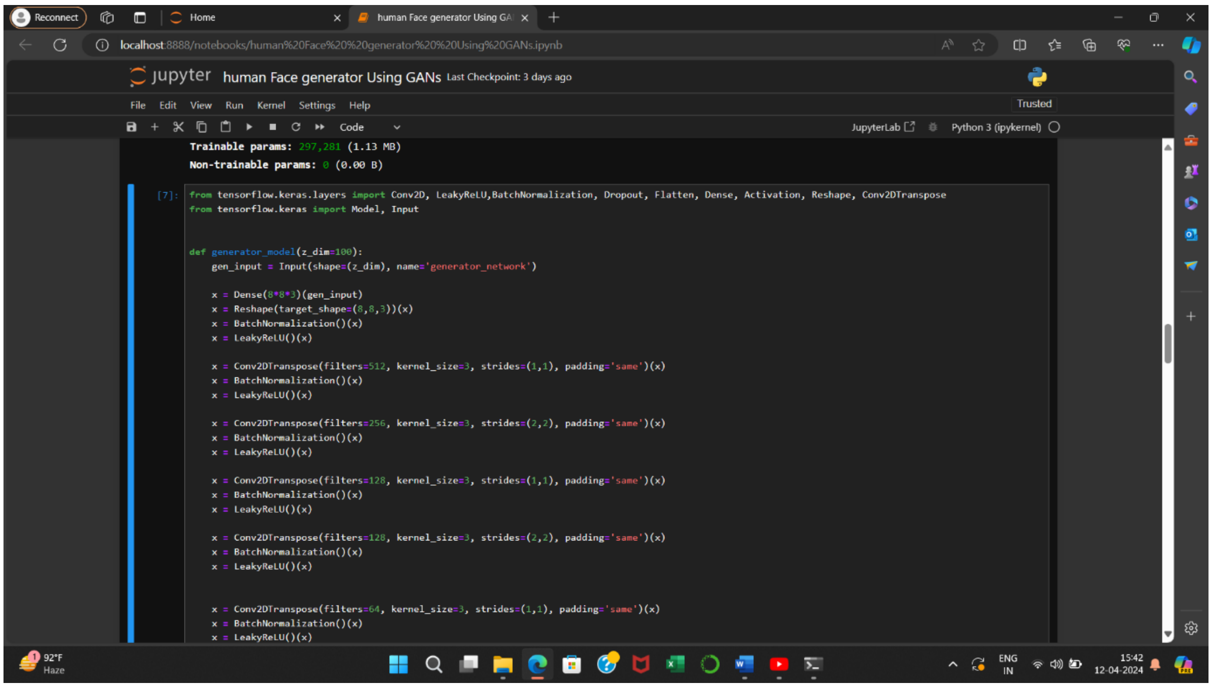

GANs-> Generartive Adversarial Network Generator, Disc. Disc.-> Image Classification Task Latent-> Noise

```
[14]: from tensorflow.keras.layers import Conv2D, LeakyReLU,BatchNormalization, Dropout, Flatten, Dense, Activation, Reshape
      from tensorflow.keras import Model, Input

      def discriminator_model():
          disc_input = Input(shape=(32,32,3), name='discriminator_network')
          x=Conv2D(filters=64, kernel_size=3, strides=(2,2), padding='same')(disc_input)
          x=LeakyReLU()(x)
          x=Dropout(0.2)(x)

          x=Conv2D(filters=128, kernel_size=3, strides=(2,2), padding='same')(x)
          x=LeakyReLU()(x)
          x=Dropout(0.2)(x)

          x=Conv2D(filters=128, kernel_size=3, strides=(2,2), padding='same')(x)
          x=LeakyReLU()(x)
          x=Dropout(0.2)(x)

          x=Conv2D(filters=64, kernel_size=3, strides=(2,2), padding='same')(x)
          x=LeakyReLU()(x)
          x=Dropout(0.2)(x)

          x=Flatten()(x)
          x=Dense(1)(x)

          output = Activation('sigmoid')(x)

          disc=Model(inputs = disc_input, outputs=output)

          return disc
```

```
[15]: disc_model = discriminator_model()
      disc_model.summary()
```


```
disc_model.summary()
```

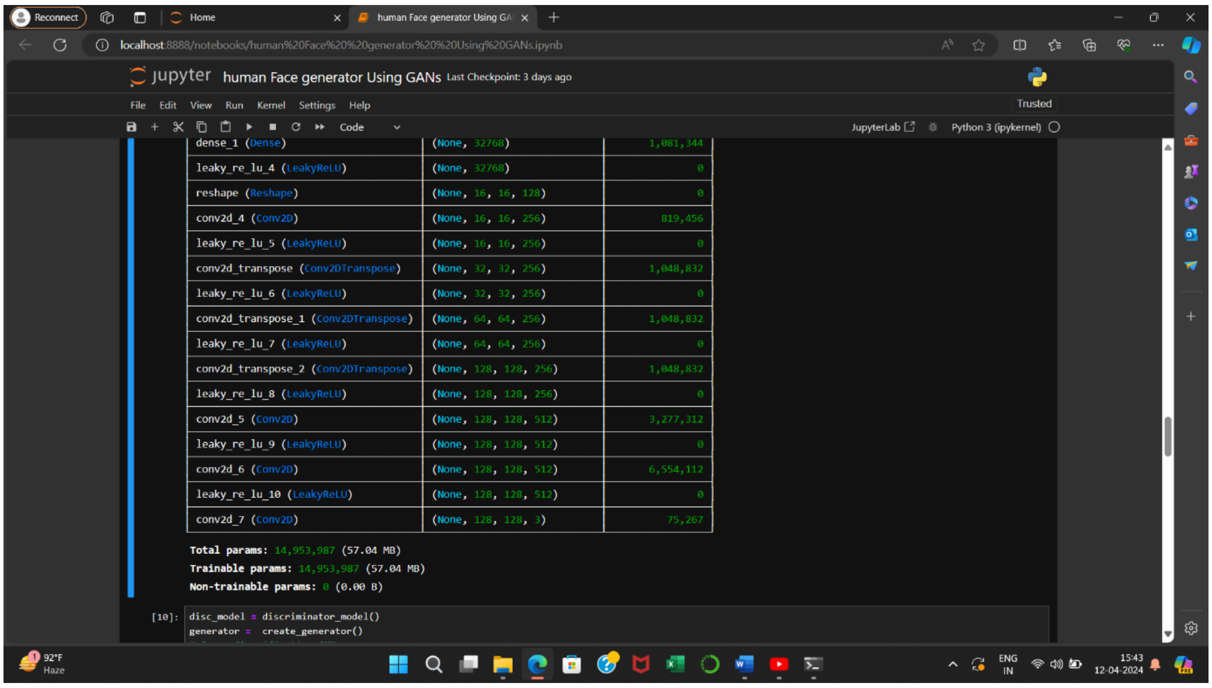
```
Model: "functional_1"
┌─────────────────────────────────────┬────────────────────────┬──────────┐
│ Layer (type)                        │ Output Shape           │  Param # │
├─────────────────────────────────────┼────────────────────────┼──────────┤
│ discriminator_network (InputLayer)  │ (None, 32, 32, 3)      │        0 │
│ conv2d (Conv2D)                     │ (None, 16, 16, 64)     │    1,792 │
│ leaky_re_lu (LeakyReLU)             │ (None, 16, 16, 64)     │        0 │
│ dropout (Dropout)                   │ (None, 16, 16, 64)     │        0 │
│ conv2d_1 (Conv2D)                   │ (None, 8, 8, 128)      │   73,856 │
│ leaky_re_lu_1 (LeakyReLU)           │ (None, 8, 8, 128)      │        0 │
│ dropout_1 (Dropout)                 │ (None, 8, 8, 128)      │        0 │
│ conv2d_2 (Conv2D)                   │ (None, 4, 4, 128)      │  147,584 │
│ leaky_re_lu_2 (LeakyReLU)           │ (None, 4, 4, 128)      │        0 │
│ dropout_2 (Dropout)                 │ (None, 4, 4, 128)      │        0 │
│ conv2d_3 (Conv2D)                   │ (None, 2, 2, 64)       │   73,792 │
│ leaky_re_lu_3 (LeakyReLU)           │ (None, 2, 2, 64)       │        0 │
│ dropout_3 (Dropout)                 │ (None, 2, 2, 64)       │        0 │
│ flatten (Flatten)                   │ (None, 256)            │        0 │
│ dense (Dense)                       │ (None, 1)              │      257 │
│ activation (Activation)             │ (None, 1)              │        0 │
└─────────────────────────────────────┴────────────────────────┴──────────┘
 Total params: 297,281 (1.13 MB)
```



localhost:8888/notebooks/human%20Face%20%20generator%20%20Using%20GANs.ipynb
jupyter human Face generator Using GANs Last Checkpoint: 3 days ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
Trainable params: 297,281 (1.13 MB)
Non-trainable params: 0 (0.00 B)
```

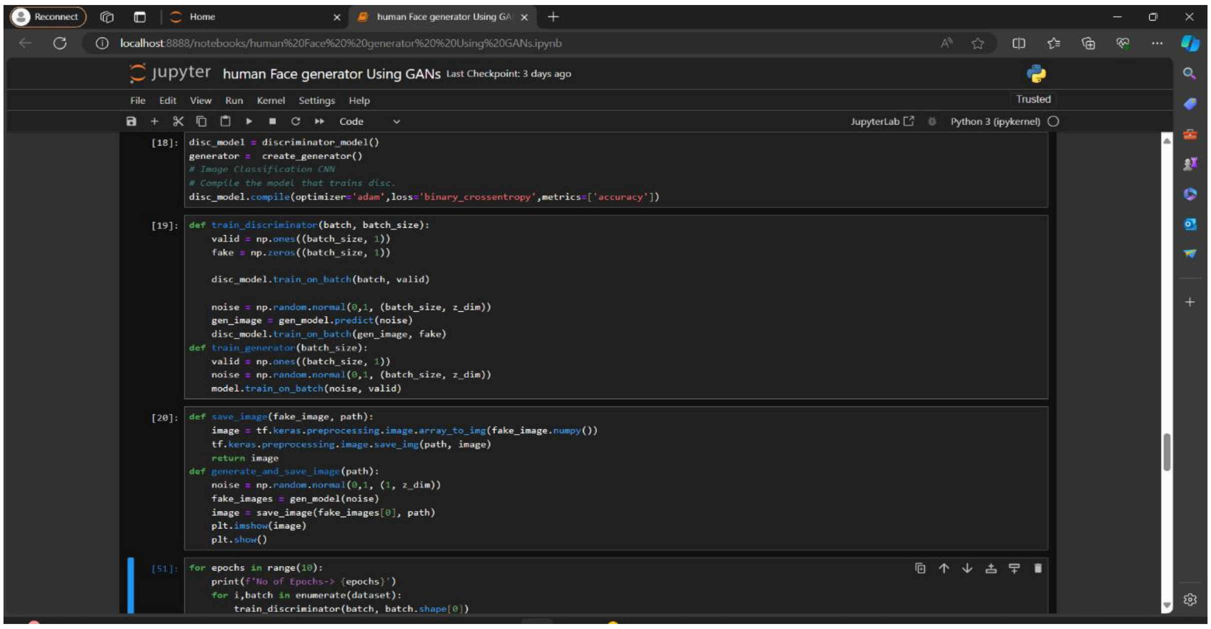
```
[7]: from tensorflow.keras.layers import Conv2D, LeakyReLU,BatchNormalization, Dropout, Flatten, Dense, Activation, Reshape, Conv2DTranspose
     from tensorflow.keras import Model, Input


     def generator_model(z_dim=100):
         gen_input = Input(shape=(z_dim), name='generator_network')

         x = Dense(8*8*3)(gen_input)
         x = Reshape(target_shape=(8,8,3))(x)
         x = BatchNormalization()(x)
         x = LeakyReLU()(x)

         x = Conv2DTranspose(filters=512, kernel_size=3, strides=(1,1), padding='same')(x)
         x = BatchNormalization()(x)
         x = LeakyReLU()(x)

         x = Conv2DTranspose(filters=256, kernel_size=3, strides=(2,2), padding='same')(x)
         x = BatchNormalization()(x)
         x = LeakyReLU()(x)

         x = Conv2DTranspose(filters=128, kernel_size=3, strides=(1,1), padding='same')(x)
         x = BatchNormalization()(x)
         x = LeakyReLU()(x)

         x = Conv2DTranspose(filters=128, kernel_size=3, strides=(2,2), padding='same')(x)
         x = BatchNormalization()(x)
         x = LeakyReLU()(x)


         x = Conv2DTranspose(filters=64, kernel_size=3, strides=(1,1), padding='same')(x)
         x = BatchNormalization()(x)
         x = LeakyReLU()(x)
```



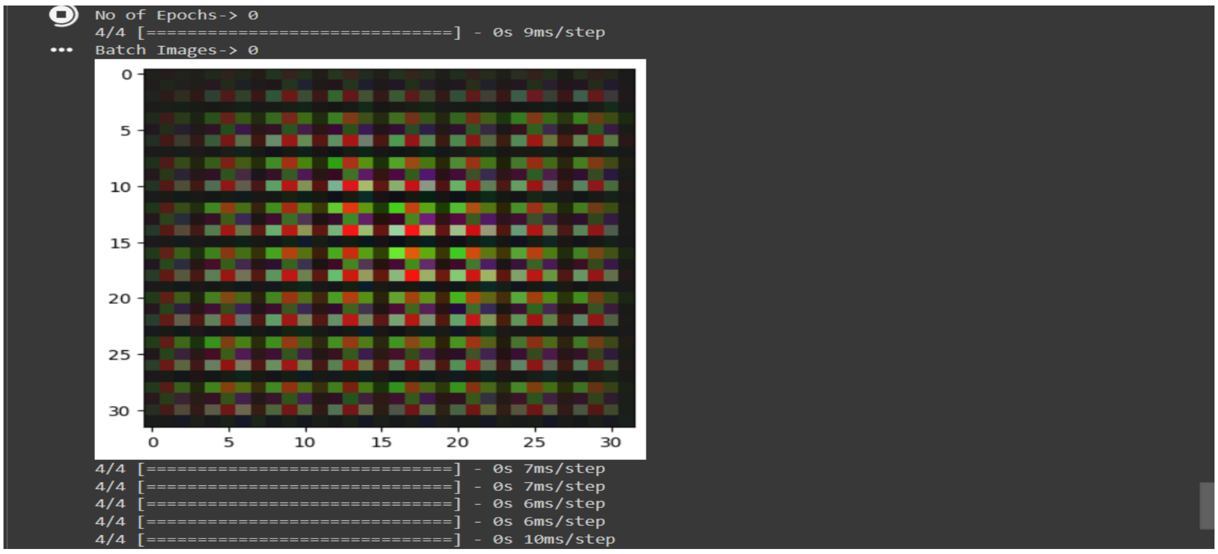
localhost:8888/notebooks/human%20Face%20%20generator%20%20Using%20GANs.ipynb
jupyter human Face generator Using GANs Last Checkpoint: 3 days ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
│ dense_1 (Dense)                  │ (None, 32768)            │     1,081,344 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ leaky_re_lu_4 (LeakyReLU)        │ (None, 32768)            │             0 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ reshape (Reshape)                │ (None, 16, 16, 128)      │             0 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_4 (Conv2D)                │ (None, 16, 16, 256)      │       819,456 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ leaky_re_lu_5 (LeakyReLU)        │ (None, 16, 16, 256)      │             0 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_transpose (Conv2DTranspose) │ (None, 32, 32, 256)    │     1,048,832 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ leaky_re_lu_6 (LeakyReLU)        │ (None, 32, 32, 256)      │             0 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_transpose_1 (Conv2DTranspose) │ (None, 64, 64, 256)  │     1,048,832 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ leaky_re_lu_7 (LeakyReLU)        │ (None, 64, 64, 256)      │             0 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_transpose_2 (Conv2DTranspose) │ (None, 128, 128, 256) │    1,048,832 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ leaky_re_lu_8 (LeakyReLU)        │ (None, 128, 128, 256)    │             0 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_5 (Conv2D)                │ (None, 128, 128, 512)    │     3,277,312 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ leaky_re_lu_9 (LeakyReLU)        │ (None, 128, 128, 512)    │             0 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_6 (Conv2D)                │ (None, 128, 128, 512)    │     6,554,112 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ leaky_re_lu_10 (LeakyReLU)       │ (None, 128, 128, 512)    │             0 │
├──────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_7 (Conv2D)                │ (None, 128, 128, 3)      │        75,267 │
└──────────────────────────────────┴──────────────────────────┴───────────────┘
 Total params: 14,953,987 (57.04 MB)
 Trainable params: 14,953,987 (57.04 MB)
 Non-trainable params: 0 (0.00 B)
```

```
[10]: disc_model = discriminator_model()
      generator =  create_generator()
```

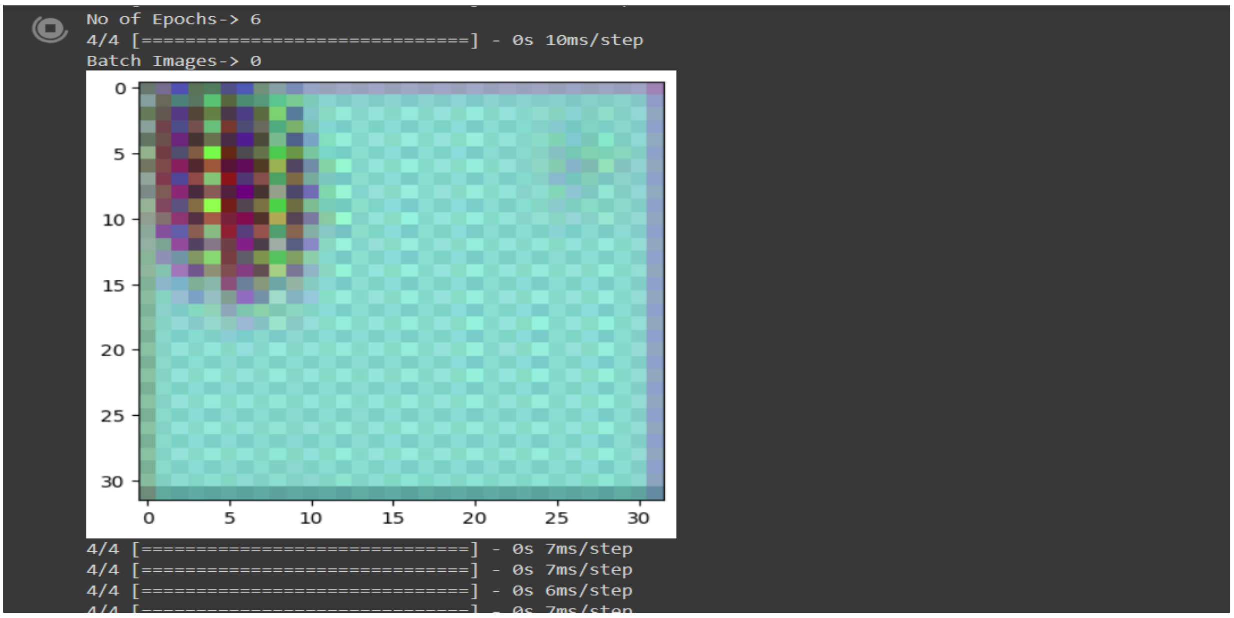

Reconnect
Home
human Face generator Using GA
localhost:8888/notebooks/human%20Face%20%20generator%20%20Using%20GANs.ipynb
jupyter
human Face generator Using GANs
Last Checkpoint: 3 days ago
File
Edit
View
Run
Kernel
Settings
Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
[18]: disc_model = discriminator_model()
      generator =  create_generator()
      # Image Classification CNN
      # Compile the model that trains disc.
      disc_model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

[19]: def train_discriminator(batch, batch_size):
          valid = np.ones((batch_size, 1))
          fake = np.zeros((batch_size, 1))

          disc_model.train_on_batch(batch, valid)

          noise = np.random.normal(0,1, (batch_size, z_dim))
          gen_image = gen_model.predict(noise)
          disc_model.train_on_batch(gen_image, fake)
      def train_generator(batch_size):
          valid = np.ones((batch_size, 1))
          noise = np.random.normal(0,1, (batch_size, z_dim))
          model.train_on_batch(noise, valid)

[20]: def save_image(fake_image, path):
          image = tf.keras.preprocessing.image.array_to_img(fake_image.numpy())
          tf.keras.preprocessing.image.save_img(path, image)
          return image
      def generate_and_save_image(path):
          noise = np.random.normal(0,1, (1, z_dim))
          fake_images = gen_model(noise)
          image = save_image(fake_images[0], path)
          plt.imshow(image)
          plt.show()

[51]: for epochs in range(10):
          print(f'No of Epochs-> {epochs}')
          for i,batch in enumerate(dataset):
              train_discriminator(batch, batch.shape[0])
```



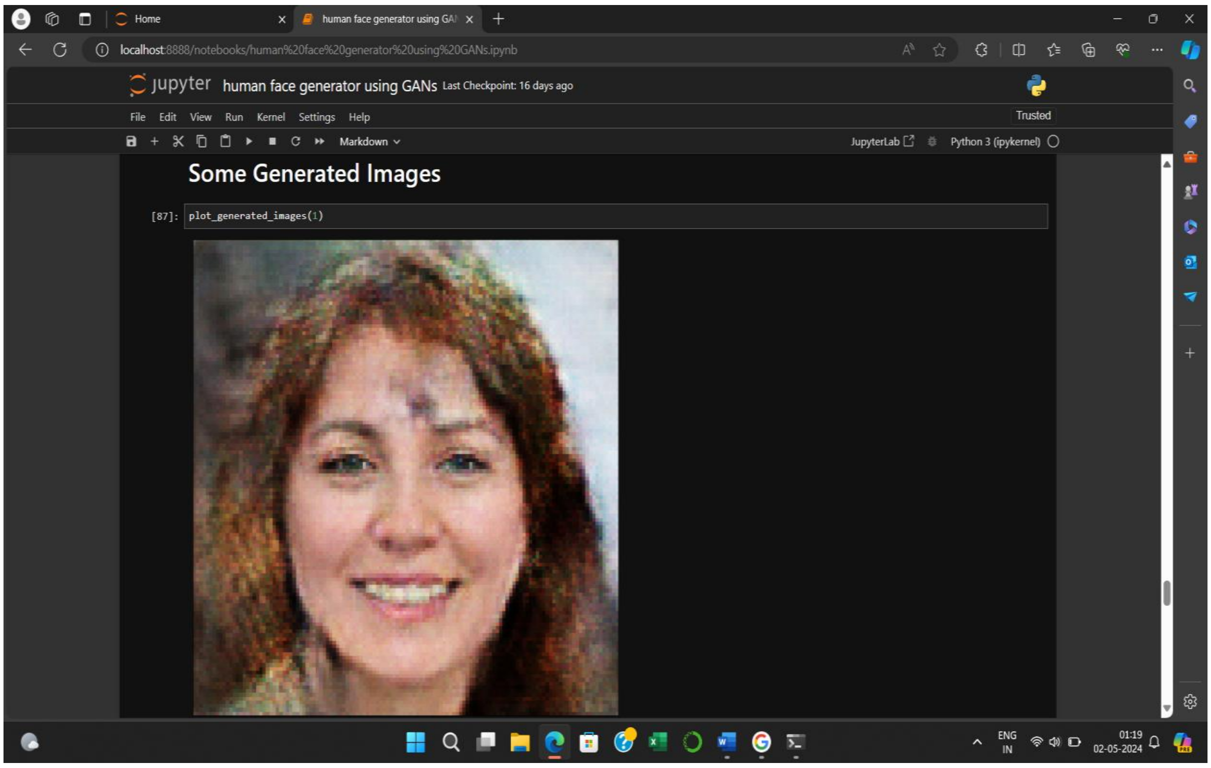
```
No of Epochs-> 0
4/4 [==============================] - 0s 9ms/step
Batch Images-> 0
```

```
4/4 [==============================] - 0s 7ms/step
4/4 [==============================] - 0s 7ms/step
4/4 [==============================] - 0s 6ms/step
4/4 [==============================] - 0s 6ms/step
4/4 [==============================] - 0s 10ms/step
```



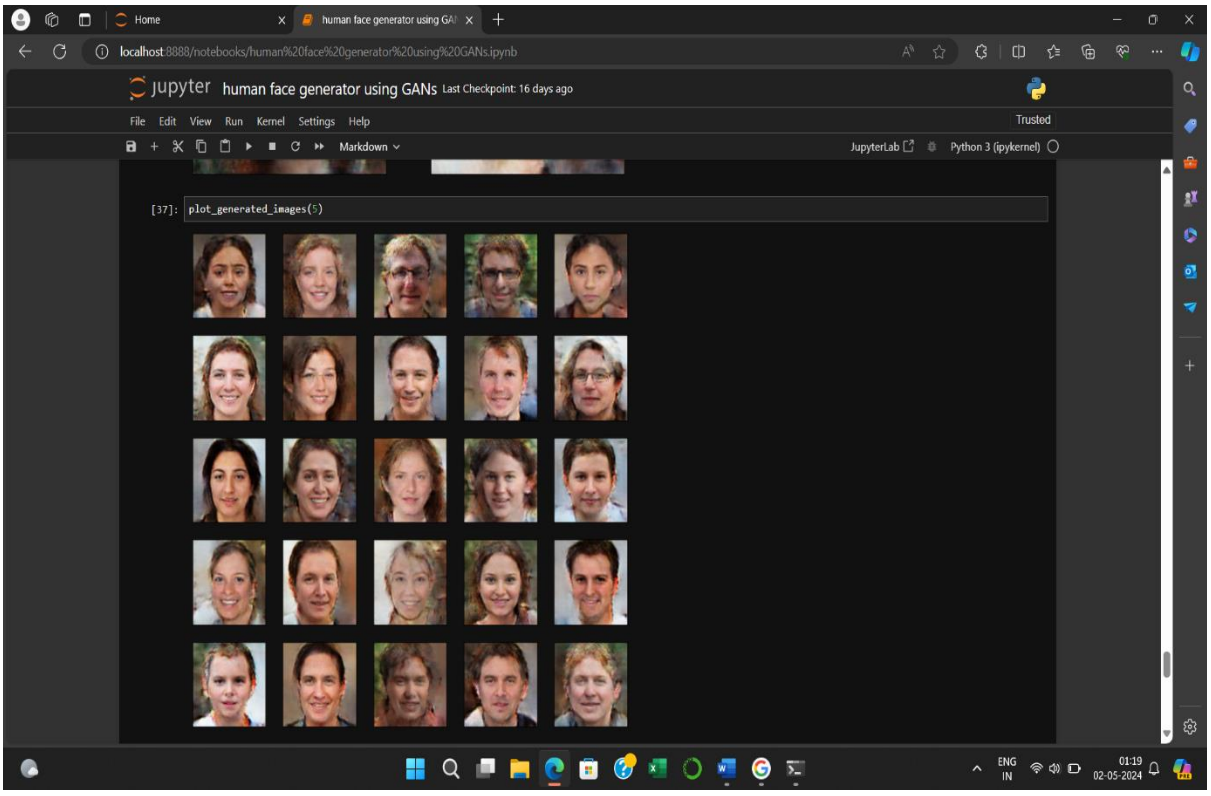
```
No of Epochs-> 6
4/4 [==============================] - 0s 10ms/step
Batch Images-> 0
```

```
4/4 [==============================] - 0s 7ms/step
4/4 [==============================] - 0s 7ms/step
4/4 [==============================] - 0s 6ms/step
4/4 [==============================] - 0s 7ms/step
```



Some Generated Images

```
[87]: plot_generated_images(1)
```



```
[37]: plot_generated_images(5)
```

CHAPTER 6

6.CONCLUSION AND FUTURE ENHANCEMENT

From our result and the loss function, we can see that, we successfully created human faces from random noises using limited computational power. However, the images are not as good as the real life images. Overall, using our low computational power we have tried to generate real life fake human faces. For comparing this model's results and to see how it reacts with different sizes of images we have tested it with three different sizes. At first, we tested it using 48 * 48 sizes. At that time, the model did not give that much good result though it generated fake human faces having noises. And then, to improve the quality we used 100*100 sizes. This time, the model performed quite well, where it gave quite a good result. When we used 100*100 sizes of the generated images were very clear and understandable. To get a better result, we then used 150*150 images. This time the model gave comparatively better results than the previous two models. Each time we ran our model at 100000 epochs.

Fake images can be used by police to track down people who are involved in adultery.

As we know nowadays child and underage pornography is alarming. To catch such people we can use computer generated fake faces, that will be more ethical than using real images .

As we can see nowadays creating fake or edited images is very easy. To identify these kinds of images, we can build a model. For this model to be trained we can use the computer generated images to identify the real and fake images.

We use the fake computer generated images in visual art and the advertising industries. These images have a huge role in computer games.

In computer and mobile games along with play station games, these images can be used.

As augmented reality and virtual reality are growing rapidly, we can use these computer generated fake images.

CHAPTER 7

7.References or bibliography

- https://www.geeksforgeeks.org/introduction-deep-learning/
- https://www.techtarget.com/searchenterpriseai/definition/generative-adversarial-network-GAN
- https://www.technologyreview.com/2018/02/21/145289/the-ganfather-the-man-whos-given-machines-the-gift-of-imagination/
- https://docs.anaconda.com/ae-notebooks/user-guide/basic-tasks/apps/jupyter/index.html
- https://www.tensorflow.org/guide/keras
- https://www.kaggle.com/datasets/jessicali9530/celeba-dataset
- https://blog.paperspace.com/face-generation-with-dcgans/
- https://www.kdnuggets.com/2020/03/generate-realistic-human-face-using-gan.html