Assignment No: 3

TICS-II

BCS 8

Name: **Hussain Ali**

Reg No: **FA21-BCS-066**

1. **Remember (Knowledge Recall)**

- **Task: List and define the key components of a GAN, including the Generator, Discriminator, and the concept of adversarial training.**

1. Generator (G)

Definition:
The **Generator** is a neural network that learns to generate data that mimics the real data distribution (e.g., images, audio, text).

Function:

- Takes **random noise (latent vector)** as input.
- Tries to produce **realistic outputs** that resemble actual data.

Goal: Fool the Discriminator into classifying its outputs as real.

2. Discriminator (D)

Definition:
The **Discriminator** is another neural network that acts as a binary classifier to distinguish between real data (from the dataset) and fake data (from the Generator).

Function:

- Receives either real data or generated data as input.
- Outputs a probability indicating if the data is real or fake.

Goal: Correctly classify real vs. fake data.

3. Adversarial Training

Definition:
Adversarial training is the **training process** in which the Generator and Discriminator are **pitted against each other** in a two-player minimax game.

Function:

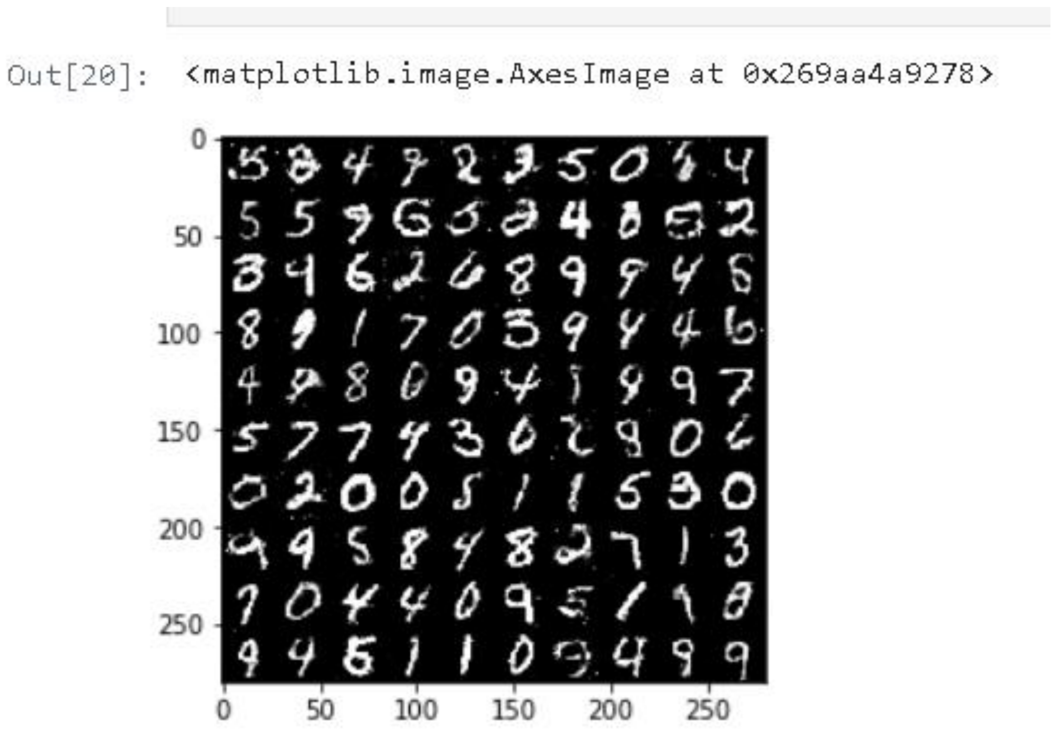
- The **Generator (G)** takes random noise and produces fake data.
- The **Discriminator (D)** receives both real data and fake data, and tries to **classify** them correctly.
- Both networks **update their parameters** during training:
  - **D** gets better at detecting fakes.
  - **G** gets better at creating fakes that **fool D**.

Goal:
To reach a **point of equilibrium** where:

- The **Discriminator can no longer tell** the difference between real and generated data (outputs 0.5 probability for both).
- The **Generator produces data** that is **indistinguishable from real data**.

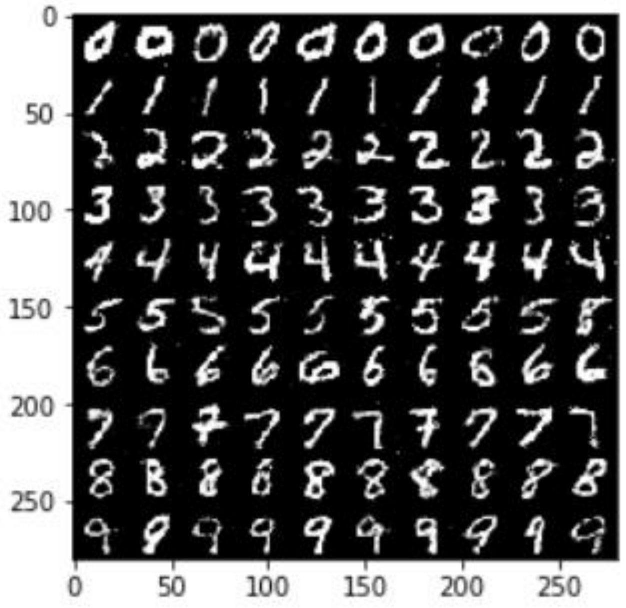
This results in a **well-trained Generator** capable of creating **realistic outputs**.

- **Deliverable: Run all the codes from Section 1 to Section 9 , include Screenshots and short explanation of code**

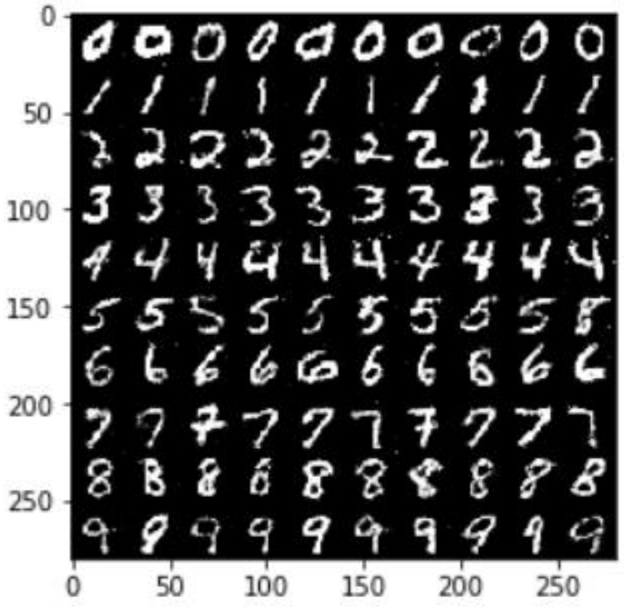


```
Out[20]:  <matplotlib.image.AxesImage at 0x269aa4a9278>
```

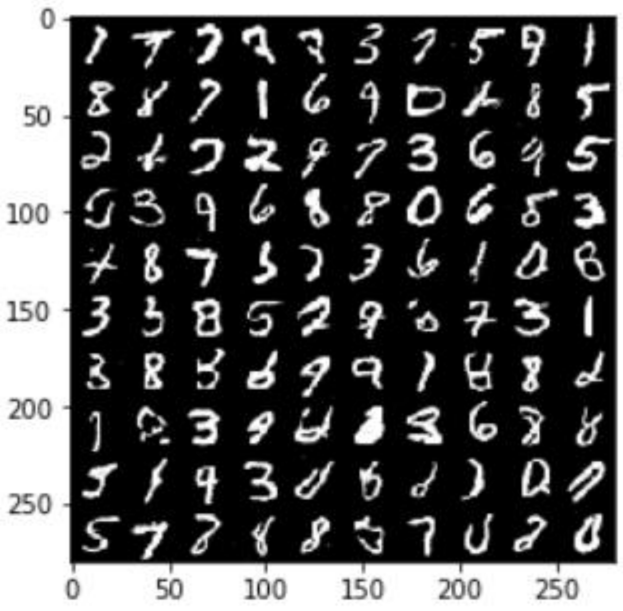
```
Out[20]:  <matplotlib.image.AxesImage at 0x219b2a6c400>
```


0
50
100
150
200
250
0
50
100
150
200
250

```
Out[20]:  <matplotlib.image.AxesImage at 0x219b2a6c400>
```

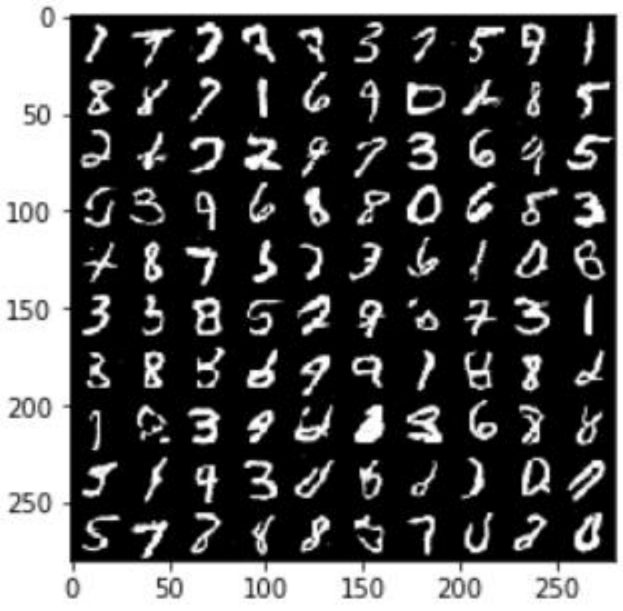

0
50
100
150
200
250
0
50
100
150
200
250

Out[18]:
```
<matplotlib.image.AxesImage at 0x1e152e82630>
```



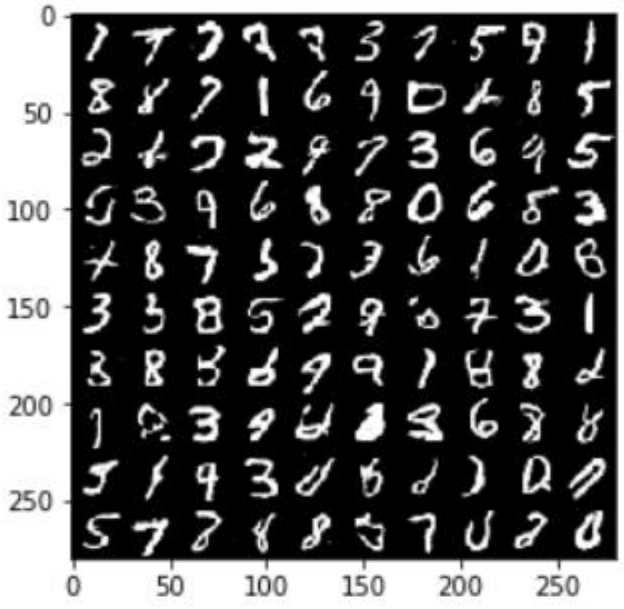
In [ ]:

Out[18]:
```
<matplotlib.image.AxesImage at 0x1e152e82630>
```

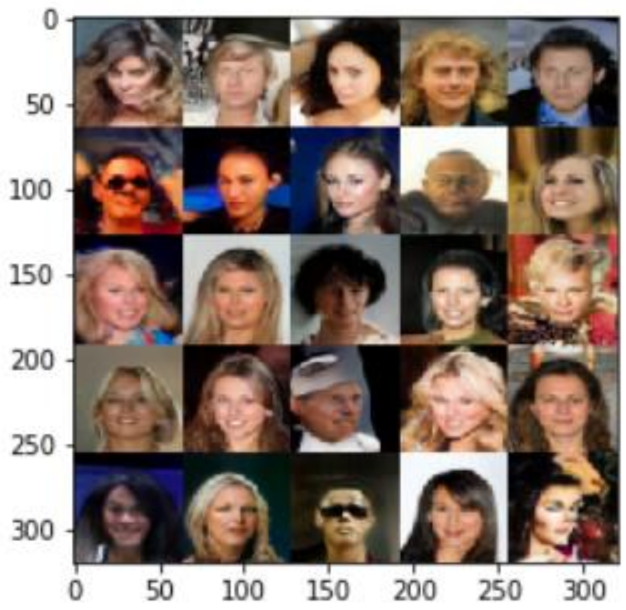


In [ ]:

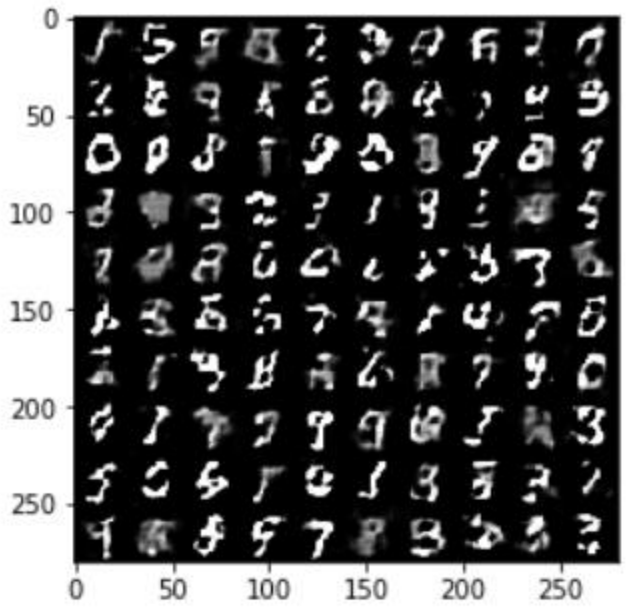
```
Out[18]:  <matplotlib.image.AxesImage at 0x1e152e82630>
```


0
50
100
150
200
250
0
50
100
150
200
250

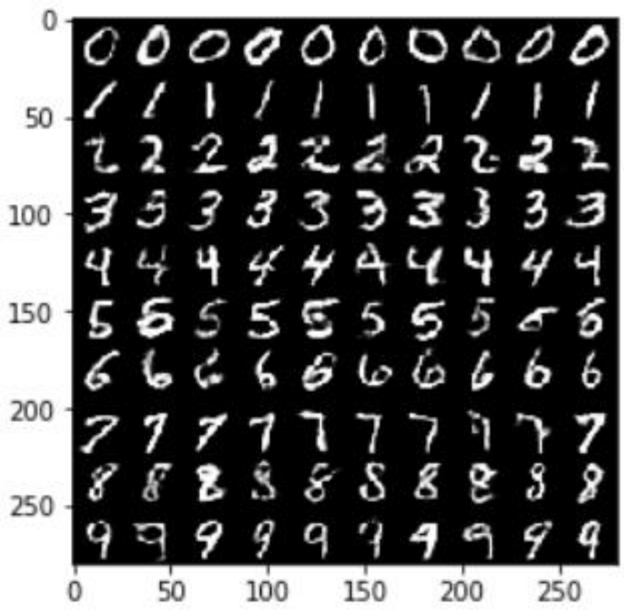
```
Out[42]:  <matplotlib.image.AxesImage at 0x7f7c87d1bc18>
```


0
50
100
150
200
250
300
0
50
100
150
200
250
300

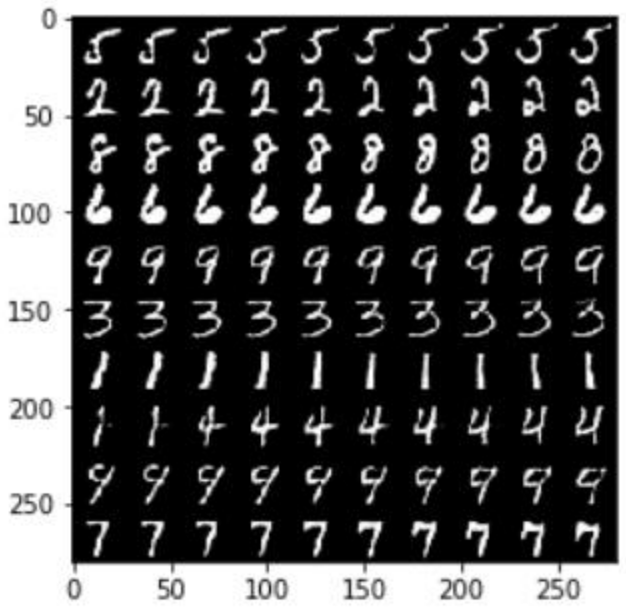
```
Out[20]:  <matplotlib.image.AxesImage at 0x1d4550dc8d0>
```



```
Out[29]:  <matplotlib.image.AxesImage at 0x7f23f053c0f0>
```



```
Out[25]: <matplotlib.image.AxesImage at 0x2342b7f5048>
```



Understand (Comprehension)

- **Task: Explain the process of training a GAN, detailing how the Generator and Discriminator interact during the training phase.**

1. Initialize the Generator and Discriminator

Both networks are initialized with random weights.

- The **generator** takes random noise as input and tries to generate fake data (e.g., images).

- The **discriminator** takes an image and outputs a probability of whether the image is real (from the dataset) or fake (from the generator).

---

2. Prepare Real and Fake Data

- **Real data** is sampled from the dataset (e.g., MNIST images).
- **Fake data** is produced by feeding random noise (and sometimes conditional labels) into the generator.

---

3. Train the Discriminator

- Input both **real images** and **fake images** into the discriminator.
- Label the real images as 1 and fake images as 0.
- Calculate the **discriminator loss** using a binary cross-entropy loss function, comparing the predicted labels to the true labels.
- Backpropagate the loss and update only the discriminator's weights.

At this stage, the discriminator learns to distinguish real images from fake ones.

4. Train the Generator

- Generate fake images from random noise.
- Feed these fake images into the discriminator (without updating the discriminator's weights).
- The goal is to **fool the discriminator**, so label the fake images as 1 (i.e., pretend they are real).
- Compute the **generator loss** based on the discriminator's output and backpropagate the loss through the generator.
- Update only the generator's weights.

This encourages the generator to produce more realistic images.

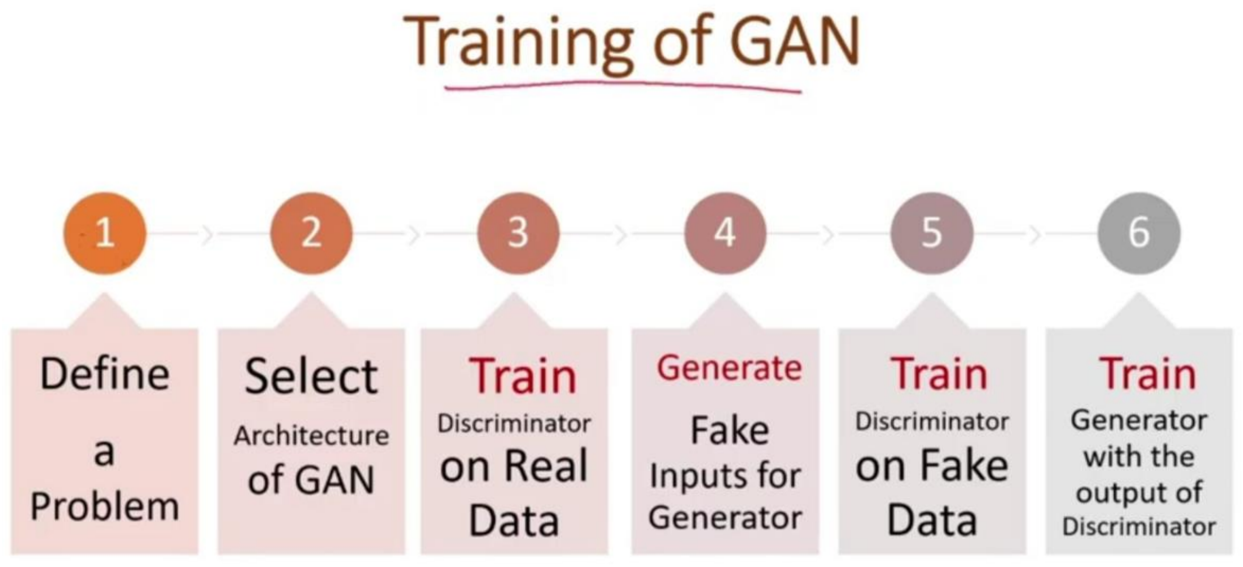
---

5. Repeat

- Alternate between training the discriminator and the generator for several epochs.
- As training progresses:
    - The generator improves its ability to create realistic data.
    - The discriminator gets better at spotting fakes—up to a point.

Ideally, both networks reach a point where the discriminator cannot easily tell real from fake, achieving what is known as a **Nash Equilibrium**.

- **Deliverable: A diagram illustrating the training loop of a GAN, accompanied by a brief explanation of each step.**


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

Step 1: Define a Problem

The problem statement is key to the success of the project so the first step is to define your problem. GANs work with a different set of problems you are aiming so you need to define What you are creating like audio, poem, text, Image is a type of problem.

Step 2: Select Architecture of GAN

There are many different types of GAN Architecture, that we will study further. we have to define which type of GAN architecture we are using.

Step 3: Train Discriminator on Real Dataset

Now Discriminator is trained on a real dataset. It is only having a forward path, no backpropagation is there in the training of the Discriminator in n epochs. And the Data you are providing is without Noise and only contains real images, and for fake images, Discriminator uses instances created by the generator as negative output. Now, what happens at the time of discriminator training.

- It classifies both real and fake data.
- The discriminator loss helps improve its performance and penalize it when it misclassifies real as fake or vice-versa.
- weights of the discriminator are updated through discriminator loss.

Step 4: Train Generator

Provide some fake inputs (noise) for the generator, which will use random noise to generate fake outputs. When you train the Generator, the Discriminator remains idle, and when you train the Discriminator, the Generator stays idle. During generator training, the model takes random noise as input and tries to transform it into meaningful data. Generating meaningful output takes time and runs over many epochs. The steps to train a generator are listed below.

- get random noise and produce a generator output on noise sample
- predict generator output from discriminator as original or fake.
- we calculate discriminator loss.
- perform backpropagation through discriminator, and generator both to calculate gradients.
- Use gradients to update generator weights.

Step 5: Train Discriminator on Fake Data

The Generator will pass the generated samples to the Discriminator, which will predict whether the data is fake or real and provide feedback to the Generator.

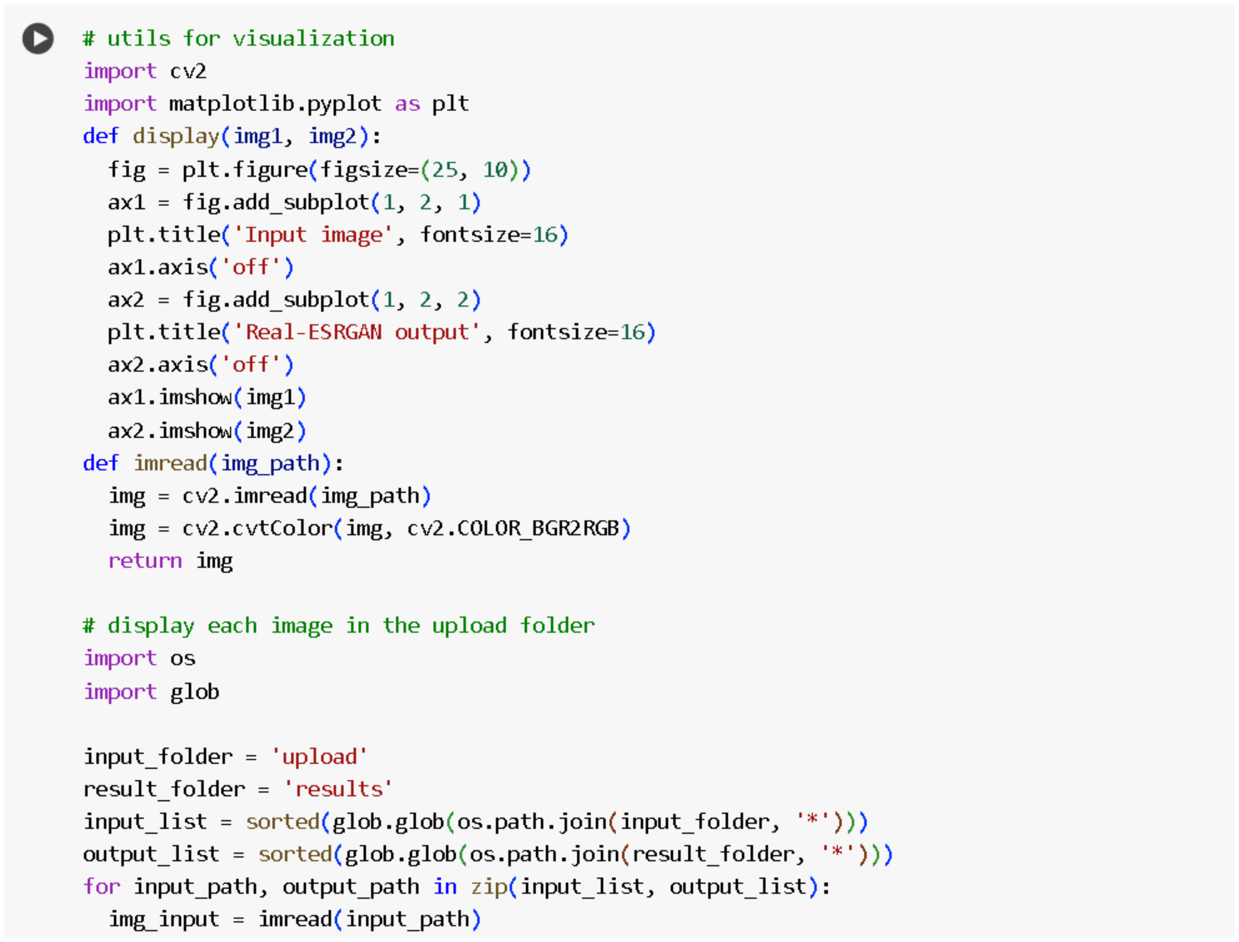
Step 6: Train Generator with the output of Discriminator

Again Generator will be trained on the feedback given by Discriminator and try to improve performance.

This is an iterative process and continues running until the Generator is not successful in making the discriminator fool.

3. Apply (Practical Application)

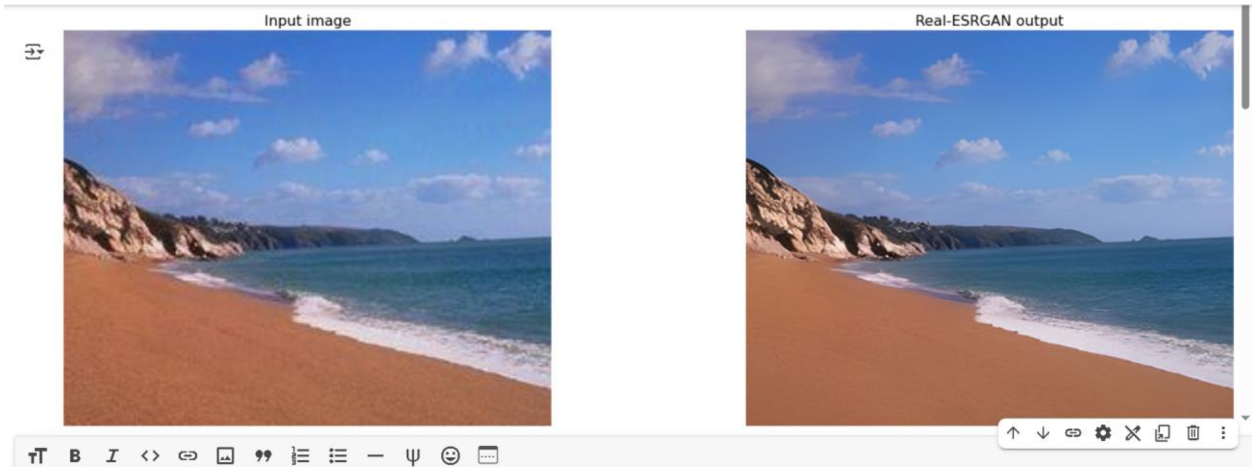
- **Task: Implement as GAN using a framework such as TensorFlow or PyTorch to generate synthetic images and upscale those images using ESRGAN**



```python
# utils for visualization
import cv2
import matplotlib.pyplot as plt
def display(img1, img2):
  fig = plt.figure(figsize=(25, 10))
  ax1 = fig.add_subplot(1, 2, 1)
  plt.title('Input image', fontsize=16)
  ax1.axis('off')
  ax2 = fig.add_subplot(1, 2, 2)
  plt.title('Real-ESRGAN output', fontsize=16)
  ax2.axis('off')
  ax1.imshow(img1)
  ax2.imshow(img2)
def imread(img_path):
  img = cv2.imread(img_path)
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  return img

# display each image in the upload folder
import os
import glob

input_folder = 'upload'
result_folder = 'results'
input_list = sorted(glob.glob(os.path.join(input_folder, '*')))
output_list = sorted(glob.glob(os.path.join(result_folder, '*')))
for input_path, output_path in zip(input_list, output_list):
  img_input = imread(input_path)
```

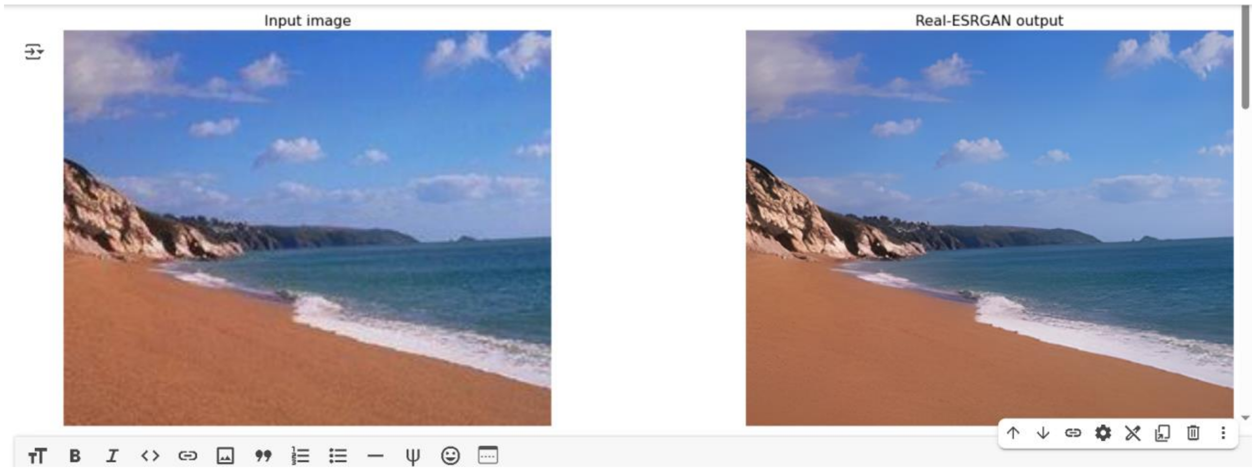


- **Deliverable: A Jupyter notebook or Python script demonstrating the GAN implementation, including code and generated outputs.**

LINK:

https://colab.research.google.com/drive/1k2Zod6kSHEvraybHl50Lys0LerhyTMCo?usp=sharing#scrollTo=nKH0syu9ZAwV

Output:


Input image
Real-ESRGAN output