E Deepak kumar

Final Project

HAND WRITTEN DIGIT RECOGNITION USING GENERATIVE ADVERSARIAL NETWORK

AGENDA

- PROBLEM STATEMENT
- PROJECT OVERVIEW
- WHO ARE THE END USERS ?
- YOUR SOLUTIONAND ITS VALUE PROPOSITION
- MODELLING
- RESULTS

PROBLEM STATEMENT

"Inefficient handwritten text recognition persists due to the scarcity of diverse datasets and the complexity of individual writing styles. To address this, leveraging Generative Adversarial Networks (GANs) offers a promising avenue. Our project aims to harness GANs to generate realistic handwritten characters, facilitating data augmentation for improved model training. By bridging the gap between synthetic and real-world handwritten samples, our approach seeks to enhance the accuracy and robustness of handwritten text recognition systems. Through this research, we aim to revolutionize handwritten text processing, enabling more efficient and accurate recognition across various applications and industries."

PROJECT OVERVIEW

The project aims to tackle the challenge of inefficient handwritten text recognition by leveraging Generative Adversarial Networks (GANs). It addresses the scarcity of diverse datasets and the complexity of individual writing styles by generating realistic handwritten characters through GANs. These generated characters facilitate data augmentation for improved model training, bridging the gap between synthetic and real-world handwritten samples. Ultimately, the goal is to enhance the accuracy and robustness of handwritten text recognition systems, revolutionizing handwritten text processing across various applications and industries.

WHO ARE THE END USERS?

1.Researchers and developers working on handwritten text recognition systems.
2.Companies or organizations implementing handwritten text recognition in their products or services, such as OCR (Optical Character Recognition) software developers.
3.Industries where handwritten text recognition is essential, such as finance (for processing handwritten forms or checks), healthcare (for interpreting handwritten medical records), and logistics (for recognizing handwritten addresses on packages).
4.Individuals who rely on handwritten text recognition tools for personal or professional use, such as students, professionals, or anyone dealing with handwritten documents.

YOUR SOLUTION AND ITS VALUE PROPOSITION

Our solution utilizes Generative Adversarial Networks (GANs) to generate realistic handwritten characters, addressing the scarcity of diverse datasets and the complexity of individual writing styles. By bridging the gap between synthetic and real-world handwritten samples, our approach facilitates data augmentation for improved model training. The value proposition lies in enhancing the accuracy and robustness of handwritten text recognition systems, revolutionizing handwritten text processing across various applications and industries.

THE WOW IN YOUR SOLUTION

The wow factor in our solution lies in its innovative use of Generative Adversarial Networks (GANs) to generate highly realistic handwritten characters. This approach not only addresses the challenges of limited datasets and diverse writing styles but also revolutionizes data augmentation for improved model training. By seamlessly bridging the gap between synthetic and real-world handwritten samples, our solution significantly enhances the accuracy and robustness of handwritten text recognition systems. This breakthrough has the potential to transform handwritten text processing, unlocking new possibilities for efficiency and accuracy across diverse applications and industries.

MODELLIN
(

Import Libraries

```
In [1]:
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
```

Define generator model

```
In [11]:
def generator_model(latent_dim):
  model = tf.keras.Sequential() #Object Creation
  model.add(layers.Dense(128,input_dim = latent_dim, activation = "relu"))
  model.add(layers.Dense(784, activation = "sigmoid"))
  model.add(layers.Reshape((28, 28, 1)))
  return model
```

Define Discriminator model

```
In [12]:
def discriminator_model(img_shape):
  model = tf.keras.Sequential() #Object Creation
  model.add(layers.Flatten(input_shape = img_shape))
  model.add(layers.Dense(128, activation = "relu"))
  model.add(layers.Dense(1, activation = "sigmoid"))
  return model
```

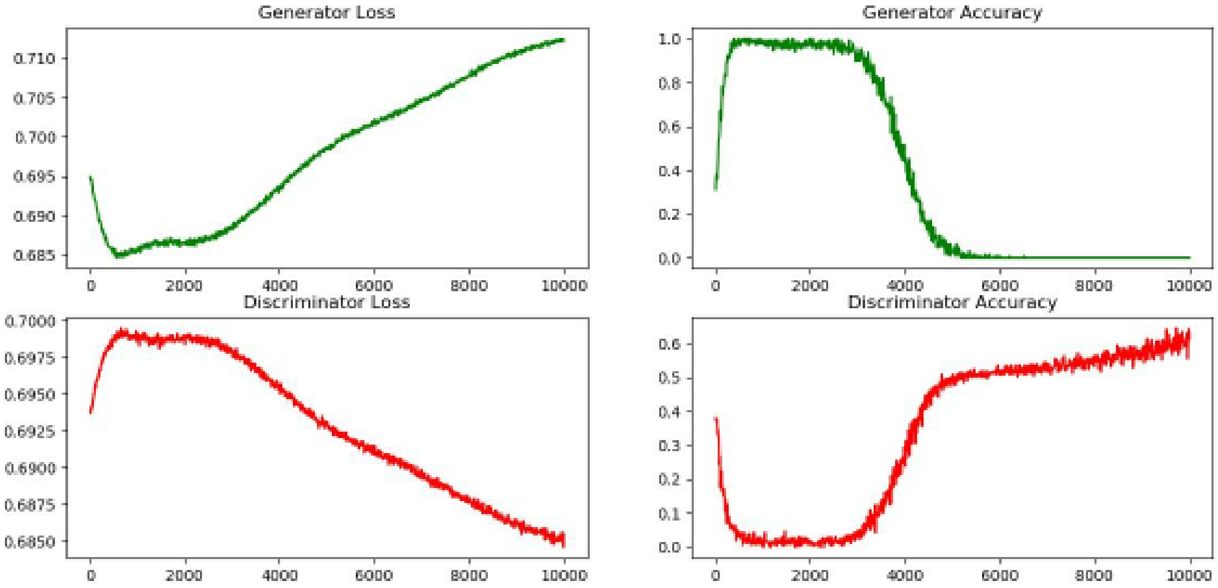
Denine the GAN model for Combining Generator and Discriminator Model

```
In [13]:
def build_gan(generator, discriminator):
  discriminator.trainable = False #Set the discriminator not to the trainable GAN training
```

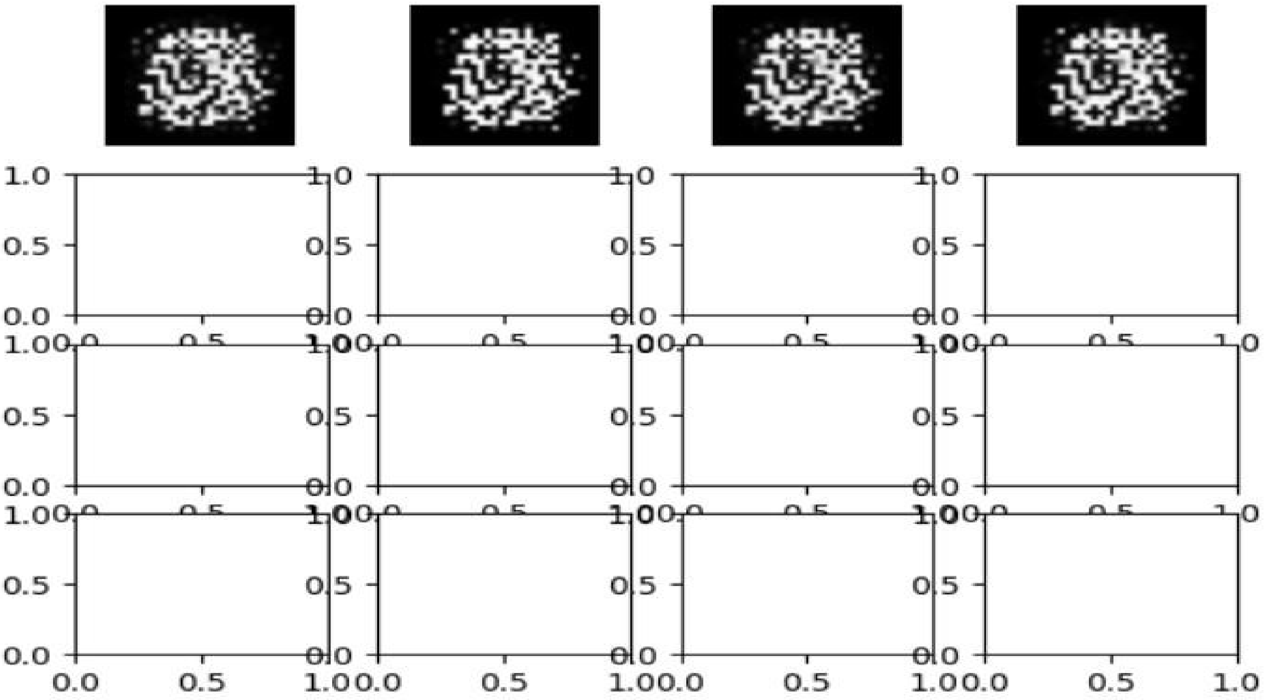
In [29]:

```
epochs = 10000

batch_size = 64

for epoch in range(epochs):
  #Generate random noise as input to the generator
  noise = np.random.normal(0, 1, size=(batch_size, latent_dim))

  # Generate fake images using the generator
  generated_images = generator.predict(noise)

  # Select a random batch of real images from the dataset
  idx = np.random.randint(0, x_train.shape[0], batch_size)
  real_images = x_train [idx]

  # Create labels for the generated and real images
  labels_real = np.ones((batch_size, 1))
  labels_fake = np.zeros((batch_size, 1))

  # Train the discriminator on real and fake images
  d_loss_real =  discriminator.train_on_batch(real_images, labels_real)
  d_loss_fake = discriminator.train_on_batch(generated_images, labels_fake)

  # Calculate the total discriminator loss
  d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

  #Train the generator by fooling the discriminator
  noise = np.random.normal(0, 1, size=(batch_size, latent_dim))
  labels_gan = np.ones((batch_size, 1))
  g_loss = gan.train_on_batch(noise, labels_gan)

  #Print the Output
  if epoch % 100 == 0:
    print(f"Epoch epoch}, Discriminator Loss: {d_loss [0]}, Generator Loss: {g_loss}")

    #Save generated images
    gen_imgs = generator.predict(np.random.normal(0, 1, size=(16, latent_dim)))
    gen_imgs = 0.5 * gen_imgs + 0.5 # Rescale generated images to [0, 1]
    fig, axs = plt.subplots (4, 4)
    count = 0
    for i in range(4):
      for j in range(4):
        axs[i, j].imshow(gen_imgs [count, :, :, 0], cmap='gray')
        axs [i, j].axis('off')
        count += 1
      plt.show()
```

RESULTS



Generator Loss
0.710
0.705
0.700
0.695
0.690
0.685
0
2000
4000
6000
8000
10000
Generator Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
Discriminator Loss
0.7000
0.6975
0.6950
0.6925
0.6900
0.6875
0.6850
Discriminator Accuracy
0.6
0.5
0.4
0.3
0.2
0.1
0.0




```
2/2 [==============================] - 0s 8ms/step
2/2 [==============================] - 0s 7ms/step
2/2 [==============================] - 0s 9ms/step
```