**PRACTICAL NO : 9**

**Deep Generative Models**

**AIM: Set up a generator network to produce samples and a discriminator network to distinguish between real and generated data. (Use a simple small dataset)**

**Code:**

import tensorflow as tf

from tensorflow.keras import layers, models

import numpy as np

import matplotlib.pyplot as plt

# Load the MNIST dataset

(X\_train, \_), (\_, \_) = tf.keras.datasets.mnist.load\_data()

X\_train = X\_train.astype('float32') / 255.0 # Normalize to [0, 1]

X\_train = np.expand\_dims(X\_train, axis=-1) # Add channel dimension (28x28x1)

# Create the generator

def build\_generator():

model = models.Sequential()

model.add(layers.Dense(256, input\_dim=100))

model.add(layers.LeakyReLU(0.2))

model.add(layers.BatchNormalization(momentum=0.8))

model.add(layers.Dense(512))

model.add(layers.LeakyReLU(0.2))

model.add(layers.BatchNormalization(momentum=0.8))

model.add(layers.Dense(1024))

model.add(layers.LeakyReLU(0.2))

model.add(layers.BatchNormalization(momentum=0.8))

model.add(layers.Dense(28 \* 28 \* 1, activation='tanh')) # Output is 28x28 image

model.add(layers.Reshape((28, 28, 1)))

return model

# Create the discriminator

def build\_discriminator():

model = models.Sequential()

model.add(layers.Flatten(input\_shape=(28, 28, 1)))

model.add(layers.Dense(512))

model.add(layers.LeakyReLU(0.2))

model.add(layers.Dense(256))

model.add(layers.LeakyReLU(0.2))

model.add(layers.Dense(1, activation='sigmoid')) # Binary output (real or fake)

return model

# Build the GAN (stacking the generator and discriminator)

def build\_gan(generator, discriminator):

discriminator.trainable = False # Freeze the discriminator during GAN training

model = models.Sequential()

model.add(generator)

model.add(discriminator)

return model

# Instantiate the models

generator = build\_generator()

discriminator = build\_discriminator()

# Compile the discriminator

discriminator.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Compile the GAN

gan = build\_gan(generator, discriminator)

gan.compile(optimizer='adam', loss='binary\_crossentropy')

# Training the GAN

epochs = 10000

batch\_size = 64

half\_batch = batch\_size // 2

# Labels for real and fake images

real\_label = np.ones((half\_batch, 1))

fake\_label = np.zeros((half\_batch, 1))

for epoch in range(epochs):

# Train discriminator with real images

idx = np.random.randint(0, X\_train.shape[0], half\_batch)

real\_images = X\_train[idx]

d\_loss\_real = discriminator.train\_on\_batch(real\_images, real\_label)

# Train discriminator with fake images (generated)

noise = np.random.normal(0, 1, (half\_batch, 100)) # Random noise

fake\_images = generator.predict(noise)

d\_loss\_fake = discriminator.train\_on\_batch(fake\_images, fake\_label)

# Compute the total discriminator loss

d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

# Train the generator (we only train the generator in the GAN model)

noise = np.random.normal(0, 1, (batch\_size, 100)) # Random noise for GAN input

g\_loss = gan.train\_on\_batch(noise, real\_label) # We want the generator to fool the discriminator

# Print the progress

if epoch % 1000 == 0:

print(f"Epoch: {epoch} | D Loss: {d\_loss[0]} | G Loss: {g\_loss}")

# Save generated images at intervals

if epoch % 5000 == 0:

noise = np.random.normal(0, 1, (16, 100))

generated\_images = generator.predict(noise)

fig, axs = plt.subplots(4, 4, figsize=(4, 4))

cnt = 0

for i in range(4):

for j in range(4):

axs[i, j].imshow(generated\_images[cnt, :, :, 0], cmap='gray')

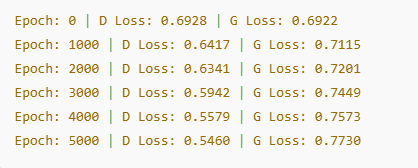
axs[i, j].axis('off')

cnt += 1

plt.savefig(f'gan\_images\_epoch\_{epoch}.png')

plt.close()

OutPut:



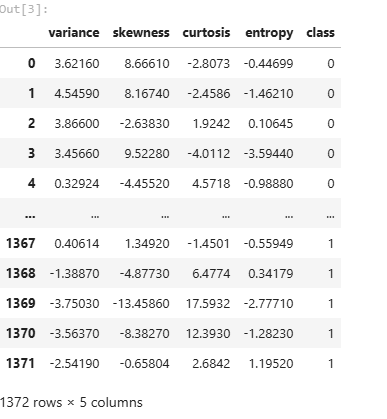
**PRACTICAL NO : 10**

**AIM: Develop an API to deploy your model and perform predictions**

**import** pandas **as** pd

**import** numpy **as** np

df**=**pd**.**read\_csv('BankNote\_Authentication.csv')

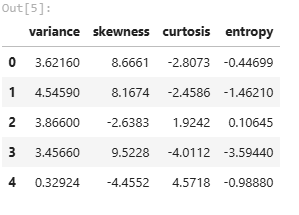
****

*### Independent and Dependent features*

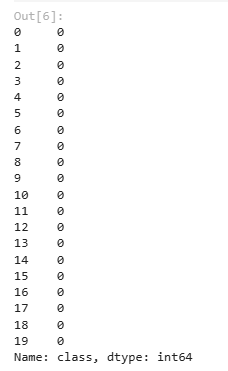
X**=**df**.**iloc[:,:**-**1]

y**=**df**.**iloc[:,**-**1]

X**.**head()

****

y**.**head(20)

****

*### Train Test Split*

**from** sklearn.model\_selection **import** train\_test\_split

X\_train,X\_test,y\_train,y\_test**=**train\_test\_split(X,y,test\_size**=**0.3,random\_state**=**0)

*### Implement Random Forest classifier*

**from** sklearn.ensemble **import** RandomForestClassifier

classifier**=**RandomForestClassifier()

classifier**.**fit(X\_train,y\_train)

RandomForestClassifier()

*## Prediction*

y\_pred**=**classifier**.**predict(X\_test)

*### Check Accuracy*

**from** sklearn.metrics **import** accuracy\_score

score**=**accuracy\_score(y\_test,y\_pred)

score



*### Create a Pickle file using serialization*

**import** pickle

pickle\_out **=** open("classifier.pkl","wb")

pickle**.**dump(classifier, pickle\_out)

pickle\_out**.**close()

**import** numpy **as** np

classifier**.**predict([[2,3,4,1]])

