Ex No: 9 BUILD GENERATIVE ADVERSARIAL NEURAL NETWORK

Aim:

To build a generative adversarial neural network using Keras/TensorFlow.

Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Program:

import tensorflow as tf from tensorflow.keras import layers

import numpy as np

import matplotlib.pyplot as plt def build\_generator(noise\_dim):

model = tf.keras.Sequential()

# Dense layer to project the noise into a larger dimension

model.add(layers.Dense(128, activation='relu', input\_dim=noise\_dim))

# Add more dense layers

model.add(layers.Dense(256, activation='relu'))

model.add(layers.Dense(512, activation='relu'))

# Final layer to output the data (usually using 'tanh' for image generation) model.add(layers.Dense(28 \* 28, activation='tanh')) model.add(layers.Reshape((28, 28))) # Shape output as 28x28 for images like MNIST

return model def build\_discriminator():

model = tf.keras.Sequential()

# Flatten the input image

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

# Add dense layers to classify real/fake model.add(layers.Dense(512, activation='relu'))

model.add(layers.Dense(256, activation='relu'))

# Final layer to output a single probability (real or fake) model.add(layers.Dense(1, activation='sigmoid'))

return model

def build\_gan(generator, discriminator):

model = tf.keras.Sequential() model.add(generator) model.add(discriminator)

return model

# Compile the discriminator discriminator = build\_discriminator()

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

# Build the generator generator = build\_generator(noise\_dim=100) # Compile the GAN (discriminator is untrainable when training the generator) discriminator.trainable = False gan = build\_gan(generator, discriminator) gan.compile(loss='binary\_crossentropy', optimizer='adam')

def train\_gan(generator, discriminator, gan, epochs, batch\_size, noise\_dim):

(X\_train, \_), \_ = tf.keras.datasets.mnist.load\_data() # Use MNIST as example

X\_train = X\_train / 127.5 - 1.0 # Normalize images to [-1, 1]

for epoch in range(epochs):

# Select a random batch of real images idx = np.random.randint(0, X\_train.shape[0], batch\_size) real\_images = X\_train[idx]

# Generate a batch of fake images noise = np.random.normal(0, 1, (batch\_size, noise\_dim)) fake\_images = generator.predict(noise)

# Train the discriminator (real = 1, fake = 0) d\_loss\_real = discriminator.train\_on\_batch(real\_images, np.ones((batch\_size, 1)))

d\_loss\_fake = discriminator.train\_on\_batch(fake\_images, np.zeros((batch\_size, 1)))

# Train the generator (wants discriminator to predict all as real) noise = np.random.normal(0, 1, (batch\_size, noise\_dim))

g\_loss = gan.train\_on\_batch(noise, np.ones((batch\_size, 1)))

# Print progress if epoch % 100 == 0:

print(f"{epoch} [D loss: {0.5 \* np.add(d\_loss\_real, d\_loss\_fake)}] [G loss: {g\_loss}]")

# Optionally save generated samples to visualize progress

train\_gan(generator, discriminator, gan, epochs=1000, batch\_size=64, noise\_dim=100)

def generate\_images(generator, noise\_dim, examples=10): noise = np.random.normal(0, 1, (examples, noise\_dim)) gen\_images = generator.predict(noise)

plt.figure(figsize=(10, 10)) for i in range(examples): plt.subplot(1, 10, i+1) plt.imshow(gen\_images[i], cmap='gray') plt.axis('off')

plt.show()

# Call this function after training to visualize generated images generate\_images(generator, noise\_dim=100)

Output:



Result:

Generative Adversial Neural network has been successfully built.