**Experiment: -10**

**AIM:** Implementing Adversarial Generative Network

Code:

import numpy as np

import tensorflow as tf

from tensorflow.keras.layers import Input, Dense, Reshape, Flatten, Dropout, LeakyReLU

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset

(x\_train, \_), (\_, \_) = mnist.load\_data()

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

x\_train = np.expand\_dims(x\_train, axis=-1)

# Define dimensions

img\_rows, img\_cols, channels = x\_train.shape[1], x\_train.shape[2], x\_train.shape[3]

img\_shape = (img\_rows, img\_cols, channels)

latent\_dim = 100

# Build the generator

def build\_generator():

model = Sequential()

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

model.add(LeakyReLU(alpha=0.2))

model.add(Dense(512))

model.add(LeakyReLU(alpha=0.2))

model.add(Dense(1024))

model.add(LeakyReLU(alpha=0.2))

model.add(Dense(np.prod(img\_shape), activation='tanh'))

model.add(Reshape(img\_shape))

noise = Input(shape=(latent\_dim,))

img = model(noise)

return Model(noise, img)

# Build the discriminator

def build\_discriminator():

model = Sequential()

model.add(Flatten(input\_shape=img\_shape))

model.add(Dense(512))

model.add(LeakyReLU(alpha=0.2))

model.add(Dense(256))

model.add(LeakyReLU(alpha=0.2))

model.add(Dense(1, activation='sigmoid'))

img = Input(shape=img\_shape)

validity = model(img)

return Model(img, validity)

# Compile the discriminator

discriminator = build\_discriminator()

discriminator.compile(loss='binary\_crossentropy', optimizer=Adam(0.0002, 0.5), metrics=['accuracy'])

# Build and compile the generator

generator = build\_generator()

generator.compile(loss='binary\_crossentropy', optimizer=Adam(0.0002, 0.5))

# Create the GAN by stacking the generator and discriminator

z = Input(shape=(latent\_dim,))

img = generator(z)

discriminator.trainable = False

valid = discriminator(img)

combined = Model(z, valid)

combined.compile(loss='binary\_crossentropy', optimizer=Adam(0.0002, 0.5))

# Training the GAN

def train(epochs, batch\_size=128, save\_interval=50):

half\_batch = int(batch\_size / 2)

for epoch in range(epochs):

# Train the discriminator

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

imgs = x\_train[idx]

noise = np.random.normal(0, 1, (half\_batch, latent\_dim))

gen\_imgs = generator.predict(noise)

d\_loss\_real = discriminator.train\_on\_batch(imgs, np.ones((half\_batch, 1)))

d\_loss\_fake = discriminator.train\_on\_batch(gen\_imgs, np.zeros((half\_batch, 1)))

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

# Train the generator

noise = np.random.normal(0, 1, (batch\_size, latent\_dim))

valid\_y = np.array([1] \* batch\_size)

g\_loss = combined.train\_on\_batch(noise, valid\_y)

# Print progress

print(f"{epoch} [D loss: {d\_loss[0]}, acc.: {100\*d\_loss[1]}%] [G loss: {g\_loss}]")

# Save generated images at intervals

if epoch % save\_interval == 0:

save\_imgs(epoch)

def save\_imgs(epoch):

r, c = 5, 5

noise = np.random.normal(0, 1, (r \* c, latent\_dim))

gen\_imgs = generator.predict(noise)

# Rescale images 0 - 1

gen\_imgs = 0.5 \* gen\_imgs + 0.5

fig, axs = plt.subplots(r, c)

cnt = 0

for i in range(r):

for j in range(c):

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

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

cnt += 1

fig.savefig(f"mnist\_{epoch}.png")

plt.close()

# Train the GAN for 10000 epochs with a batch size of 32

train(epochs=10000, batch\_size=32, save\_interval=1000)