Deep Learning-Based Medical Image Synthesis for Data Augmentation and Missing Data Completion.

PROGRAM:

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Dense, LeakyReLU, BatchNormalization, Reshape, Flatten

from tensorflow.keras.optimizers import Adam

# Load the MNIST dataset

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

# Normalize and reshape the data

X\_train = (X\_train.astype(np.float32) - 127.5) / 127.5

X\_train = X\_train.reshape(X\_train.shape[0], 784)

# Generator model

def build\_generator():

generator = Sequential([

Dense(256, input\_dim=100),

LeakyReLU(alpha=0.2),

BatchNormalization(momentum=0.8),

Dense(512),

LeakyReLU(alpha=0.2),

BatchNormalization(momentum=0.8),

Dense(1024),

LeakyReLU(alpha=0.2),

BatchNormalization(momentum=0.8),

Dense(784, activation='tanh')

])

noise = Input(shape=(100,))

img = generator(noise)

return Model(noise, img)

# Discriminator model

def build\_discriminator():

discriminator = Sequential([

Dense(512, input\_dim=784),

LeakyReLU(alpha=0.2),

Dense(256),

LeakyReLU(alpha=0.2),

Dense(1, activation='sigmoid')

])

img = Input(shape=(784,))

validity = discriminator(img)

return Model(img, validity)

# Compile the models

optimizer = Adam(lr=0.0002, beta\_1=0.5)

generator = build\_generator()

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

discriminator = build\_discriminator()

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

# Combined model

z = Input(shape=(100,))

img = generator(z)

validity = discriminator(img)

combined = Model(z, validity)

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

# Train the GAN

epochs = 30000

batch\_size = 32

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

for epoch in range(epochs):

# Select a random half batch of real images

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

imgs = X\_train[idx]

# Generate a half batch of fake images

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

gen\_imgs = generator.predict(noise)

# Train the discriminator

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, 100))

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

# Print progress

print(f"Epoch {epoch}, Discriminator Loss: {d\_loss[0]}, Generator Loss: {g\_loss}")

# Generate some synthetic images

noise = np.random.normal(0, 1, (10, 100))

gen\_imgs = generator.predict(noise)

# Plot the generated images

fig, axs = plt.subplots(2, 5)

count = 0

for i in range(2):

for j in range(5):

axs[i,j].imshow(gen\_imgs[count].reshape(28, 28), cmap='gray')

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

count += 1

plt.show()

Output:

Epoch 0, Discriminator Loss: 0.6931471824645996, Generator Loss: 0.6880357265472412

Epoch 1, Discriminator Loss: 0.6919647455215454, Generator Loss: 0.6743184328079224

Epoch 2, Discriminator Loss: 0.6923722629547119, Generator Loss: 0.658417046546936

...

Epoch 29998, Discriminator Loss: 0.6927782297134399, Generator Loss: 0.6931305522918701

Epoch 29999, Discriminator Loss: 0.6931374073028564, Generator Loss: 0.6931369304656982