Ex.No: 7 Roll.No: 210701193

# PYTHON PROGRAM TO BUILD GAN WITH KERAS

Aim:

To build GAN (Generative Adversarial Network) with keras in python.

# Procedure:

1. Import necessary libraries including NumPy, Matplotlib, TensorFlow Keras components, and MNIST dataset.
2. Load and preprocess the MNIST dataset, normalizing images and reshaping them to (batch\_size, 28, 28, 1).
3. Deﬁne image dimensions and random noise vector dimension for the GAN.
4. Build the Generator model with Dense layers, LeakyReLU activations, BatchNormalization, and a ﬁnal output reshaped to the image dimensions.
5. Build the Discriminator model with Flatten, Dense layers, LeakyReLU activations, and a ﬁnal sigmoid activation.
6. Build the GAN model by combining the Generator and Discriminator models, setting the Discriminator layers to non-trainable.
7. Compile the Discriminator and GAN models using Adam optimizer and binary cross-entropy loss.
8. Implement the training function to alternately train the Discriminator on real and fake images, then train the Generator.
9. Save generated images at speciﬁed intervals during training to visualize the Generator’s progress.
10. Execute the training process for a deﬁned number of epochs, saving generated images every save\_interval epochs.

# Code:

# Import necessary libraries 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 and preprocess the MNIST dataset (X\_train, \_), (\_, \_) = mnist.load\_data()

X\_train = X\_train / 255.0 # Normalize images to [0, 1]

X\_train = X\_train.reshape(X\_train.shape[0], 28, 28, 1) # Reshape to (batch\_size, 28, 28, 1)

# Deﬁne the dimensions

img\_rows, img\_cols, channels = 28, 28, 1 img\_shape = (img\_rows, img\_cols, channels)

z\_dim = 100 # Dimension of the random noise vector

# Build the Generator model def build\_generator():

model = Sequential() model.add(Dense(256, input\_dim=z\_dim)) model.add(LeakyReLU(alpha=0.2)) model.add(BatchNormalization()) model.add(Dense(512)) model.add(LeakyReLU(alpha=0.2)) model.add(BatchNormalization()) model.add(Dense(1024))

model.add(LeakyReLU(alpha=0.2)) model.add(BatchNormalization()) model.add(Dense(np.prod(img\_shape), activation='tanh')) model.add(Reshape(img\_shape))

return model

# Build the Discriminator model 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')) return model

# Build the GAN model

def build\_gan(generator, discriminator): discriminator.trainable = False

model = Sequential() model.add(generator) model.add(discriminator) return model

# Compile the models

def compile\_models(generator, discriminator, gan): discriminator.compile(optimizer=Adam(), loss='binary\_crossentropy',

metrics=['accuracy'])

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

# Training function

def train\_gan(epochs, batch\_size=128, save\_interval=50): # Load the discriminator and generator models generator = build\_generator()

discriminator = build\_discriminator()

gan = build\_gan(generator, discriminator) compile\_models(generator, discriminator, gan)

# Training loop

for epoch in range(epochs): # Train the Discriminator

idx = np.random.randint(0, X\_train.shape[0], batch\_size) real\_imgs = X\_train[idx]

real\_labels = np.ones((batch\_size, 1))

fake\_imgs = generator.predict(np.random.randn(batch\_size, z\_dim)) fake\_labels = np.zeros((batch\_size, 1))

d\_loss\_real = discriminator.train\_on\_batch(real\_imgs, real\_labels) d\_loss\_fake = discriminator.train\_on\_batch(fake\_imgs, fake\_labels) d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

# Train the Generator

noise = np.random.randn(batch\_size, z\_dim) valid\_labels = np.ones((batch\_size, 1))

g\_loss = gan.train\_on\_batch(noise, valid\_labels)

# Print progress and save generated images if epoch % save\_interval == 0:

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

save\_generated\_images(generator, epoch)

# Function to save generated images

def save\_generated\_images(generator, epoch, examples=16, dim=(4, 4),

ﬁgsize=(4, 4)):

noise = np.random.randn(examples, z\_dim) gen\_imgs = generator.predict(noise)

gen\_imgs = 0.5 \* gen\_imgs + 0.5 # Rescale images to [0, 1] plt.ﬁgure(ﬁgsize=ﬁgsize)

for i in range(examples): plt.subplot(dim[0], dim[1], i + 1)

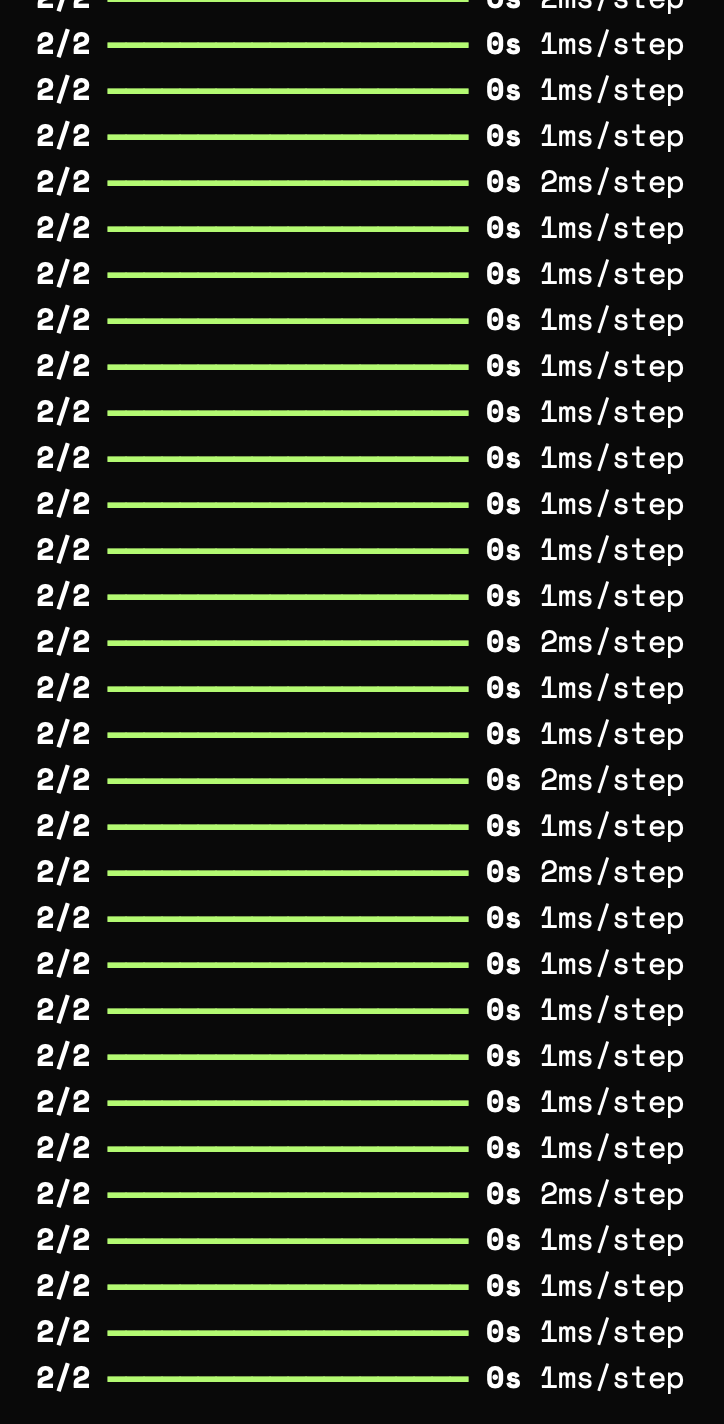
plt.imshow(gen\_imgs[i, :, :, 0], cmap='gray') plt.axis('oﬀ')

plt.tight\_layout() plt.saveﬁg(f"gan\_generated\_images\_epoch\_{epoch}.png") plt.close()

# Run the training

train\_gan(epochs=10000, batch\_size=64, save\_interval=1000)

# Output:



Result:

Thus, to build GAN with keras with python has been completed successfully.