Ex.No: 7 Roll.No: 210701189

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:

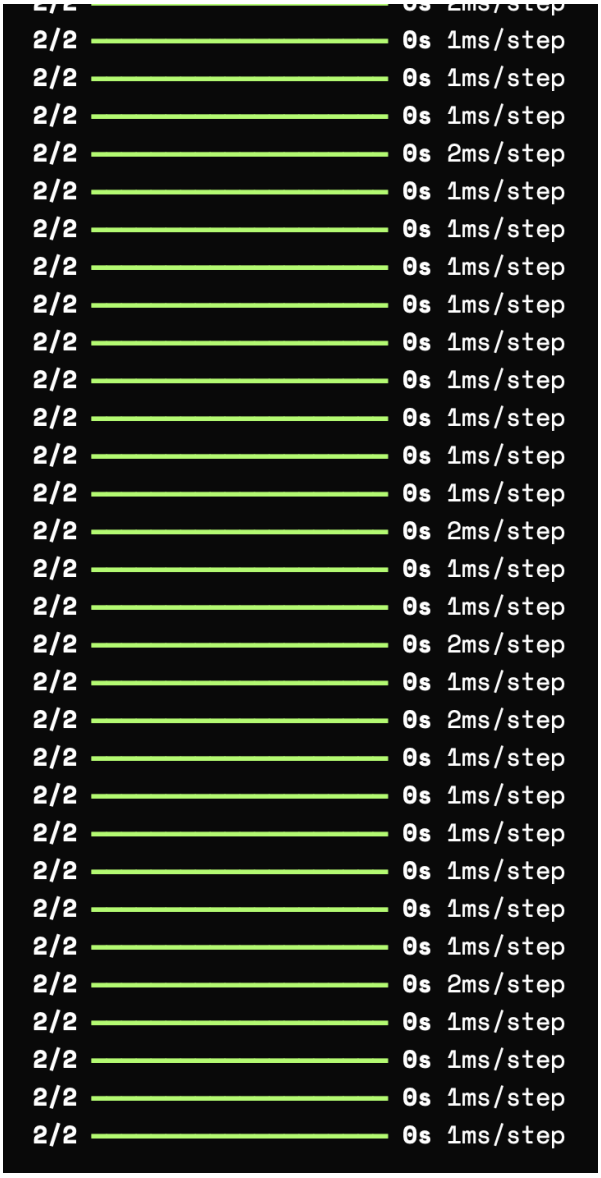
```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, LeakyReLU, BatchNormalization,
Reshape, Flatten
from tensorflow.keras.optimizers 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.