EXP. NO.9 Title: Generative Adversarial Network (GAN) for MNIST Digit Generation

---

Aim:

To build and train a Generative Adversarial Network (GAN) that generates handwritten digits similar to those in the MNIST dataset.

Procedure:

1. **Install and Import Necessary Libraries:**
   - **Install TensorFlow if not already available.**
   - **Import TensorFlow, Keras layers, NumPy, and Matplotlib for building and visualizing the GAN.**
2. **Load and Preprocess the MNIST Dataset:**
   - **Load the MNIST dataset and normalize pixel values to the range [0, 1].**
   - **Reshape the images to include a channel dimension (grayscale format).**
3. **Build the Generator Model:**
   - **Define a sequential model that upsamples a 100-dimensional noise vector into a 28×2828 \times 2828×28 image.**
   - **Use Dense, Reshape, UpSampling2D, Conv2D, BatchNormalization, and ReLU layers.**
   - **Output layer uses a `sigmoid` activation to generate pixel values between 0 and 1.**
4. **Build the Discriminator Model:**
   - **Define a sequential model that downsamples the input image to a binary classification output.**
   - **Use Conv2D, LeakyReLU, Dropout, Flatten, and Dense layers.**

- Output layer uses a `sigmoid` activation function.

5. **Compile the Discriminator:**
   - Use `binary_crossentropy` loss and the Adam optimizer.
6. **Build the Combined GAN Model:**
   - Stack the generator and discriminator models.
   - Freeze the discriminator while training the generator.
   - Compile the GAN with the same loss and optimizer.
7. **Train the GAN:**
   - In each epoch:
     - Sample real images and generate fake images.
     - Train the discriminator on both real and fake images.
     - Train the generator through the combined model, encouraging it to generate more realistic images.
   - Save generated images at regular intervals for visual inspection.

Code

```
# Install TensorFlow

!pip install tensorflow



# Import Libraries

import tensorflow as tf

from tensorflow.keras import layers
```

```
import matplotlib.pyplot as plt

import numpy as np


# Load and Preprocess MNIST Data

(X_train, _), (_, _) = tf.keras.datasets.mnist.load_data()

X_train = X_train / 255.0

X_train = X_train.reshape(-1, 28, 28, 1)


# Build Generator

def build_generator():

    model = tf.keras.Sequential([

        layers.Dense(7*7*256, input_dim=100),

        layers.Reshape((7, 7, 256)),

        layers.UpSampling2D(),

        layers.Conv2D(128, kernel_size=3, padding='same'),

        layers.BatchNormalization(),

        layers.ReLU(),

        layers.UpSampling2D(),

        layers.Conv2D(64, kernel_size=3, padding='same'),

        layers.BatchNormalization(),

        layers.ReLU(),

        layers.Conv2D(1, kernel_size=3, padding='same',
activation='sigmoid')
```

```
    ])
    return model

# Build Discriminator
def build_discriminator():
    model = tf.keras.Sequential([
        layers.Conv2D(64, kernel_size=3, strides=2, padding='same',
input_shape=(28, 28, 1)),
        layers.LeakyReLU(alpha=0.2),
        layers.Dropout(0.3),
        layers.Conv2D(128, kernel_size=3, strides=2, padding='same'),
        layers.LeakyReLU(alpha=0.2),
        layers.Dropout(0.3),
        layers.Flatten(),
        layers.Dense(1, activation='sigmoid')
    ])
    return model

# Compile Discriminator
discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy',
optimizer=tf.keras.optimizers.Adam(0.0002, 0.5), metrics=['accuracy'])
```

```
# Build and Compile GAN

generator = build_generator()

z = layers.Input(shape=(100,))

img = generator(z)

discriminator.trainable = False

valid = discriminator(img)

gan = tf.keras.models.Model(z, valid)

gan.compile(loss='binary_crossentropy',
optimizer=tf.keras.optimizers.Adam(0.0002, 0.5))


# Train the GAN

def train_gan(epochs, batch_size=128, save_interval=500):

    half_batch = batch_size // 2


    for epoch in range(epochs):

        # Train Discriminator

        idx = np.random.randint(0, X_train.shape[0], half_batch)

        real_images = X_train[idx]

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

        fake_images = generator.predict(noise)


        d_loss_real = discriminator.train_on_batch(real_images,
np.ones((half_batch, 1)))
```

```
        d_loss_fake = discriminator.train_on_batch(fake_images,
np.zeros((half_batch, 1)))

        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)


        # Train Generator

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

        g_loss = gan.train_on_batch(noise, np.ones((batch_size, 1)))


        # Print Progress

        if epoch % 100 == 0:

            print(f"{epoch} [D loss: {d_loss[0]} | D accuracy: {100 *
d_loss[1]:.2f}%] [G loss: {g_loss}]")


        # Save Images

        if epoch % save_interval == 0:

            save_generated_images(epoch)


# Function to Save Generated Images

def save_generated_images(epoch, examples=10, dim=(1, 10),
figsize=(10, 1)):

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

    generated_images = generator.predict(noise)

    generated_images = generated_images.reshape(examples, 28, 28)

    plt.figure(figsize=figsize)
```

```
    for i in range(examples):
        plt.subplot(dim[0], dim[1], i + 1)
        plt.imshow(generated_images[i], cmap='gray')
        plt.axis('off')
    plt.tight_layout()
    plt.savefig(f"generated_image_{epoch}.png")
    plt.close()

# Start Training
train_gan(epochs=3000, batch_size=32, save_interval=500)
```

Output:

🖉 Generator Model Summary:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 12544)	1,266,944
reshape (Reshape)	(None, 7, 7, 256)	0
up_sampling2d (UpSampling2D)	(None, 14, 14, 256)	0
conv2d_2 (Conv2D)	(None, 14, 14, 128)	295,040
batch_normalization (BatchNormalization)	(None, 14, 14, 128)	512
re_lu (ReLU)	(None, 14, 14, 128)	0
up_sampling2d_1 (UpSampling2D)	(None, 28, 28, 128)	0
conv2d_3 (Conv2D)	(None, 28, 28, 64)	73,792
batch_normalization_1 (BatchNormalization)	(None, 28, 28, 64)	256
re_lu_1 (ReLU)	(None, 28, 28, 64)	0
conv2d_4 (Conv2D)	(None, 28, 28, 1)	577

Total params: 1,637,121 (6.25 MB)

Trainable params: 1,636,737 (6.24 MB)

Non-trainable params: 384 (1.50 KB)

🔍 Discriminator Model Summary:
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 64)	640
leaky_re_lu (LeakyReLU)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_1 (Conv2D)	(None, 7, 7, 128)	73,856
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 128)	0
dropout_1 (Dropout)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 1)	6,273

Total params: 80,769 (315.50 KB)
Trainable params: 0 (0.00 B)
Non-trainable params: 80,769 (315.50 KB)

👹 GAN (Generator + Discriminator) Model Summary:
Model: "functional_2"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 100)	0
sequential_1 (Sequential)	(None, 28, 28, 1)	1,637,121
sequential (Sequential)	(None, 1)	80,769

Total params: 4,991,366 (19.04 MB)
Trainable params: 1,636,737 (6.24 MB)
Non-trainable params: 81,153 (317.00 KB)
Optimizer params: 3,273,476 (12.49 MB)

Result:

A Generative Adversarial Network was successfully trained on the MNIST dataset. The generator model was able to produce realistic-looking handwritten digit images after several training epochs.