Title: Image Generation on MNIST Dataset Using GAN

Aim:

To build and train a Generative Adversarial Network (GAN) for generating handwritten digit images similar to those in the MNIST dataset.

Procedure:

1. **Load and Preprocess the MNIST Dataset:**
   - **Load the MNIST dataset from TensorFlow's datasets.**
   - **Normalize the images to values between 0 and 1.**
   - **Flatten the images for fully connected layers.**
2. **Define the Generator Model:**
   - **Create a Sequential model with dense layers.**
   - **Input: Random noise vector (size 100).**
   - **Output: Generated image vector (flattened 28x28).**
3. **Define the Discriminator Model:**
   - **Create a Sequential model with dense layers.**
   - **Input: Flattened image.**
   - **Output: Probability of the image being real or fake.**
4. **Compile the Discriminator:**
   - **Use binary cross-entropy loss.**
   - **Use Adam optimizer.**

5. **Build the GAN Model:**
   - **Stack generator and discriminator.**
   - **Freeze discriminator weights during GAN training.**
6. **Train the GAN:**
   - **Alternate training discriminator and generator.**
   - **For discriminator: Train on real and generated (fake) images.**
   - **For generator: Train to fool the discriminator.**
   - **Display generated samples at intervals.**

Corrected Code:

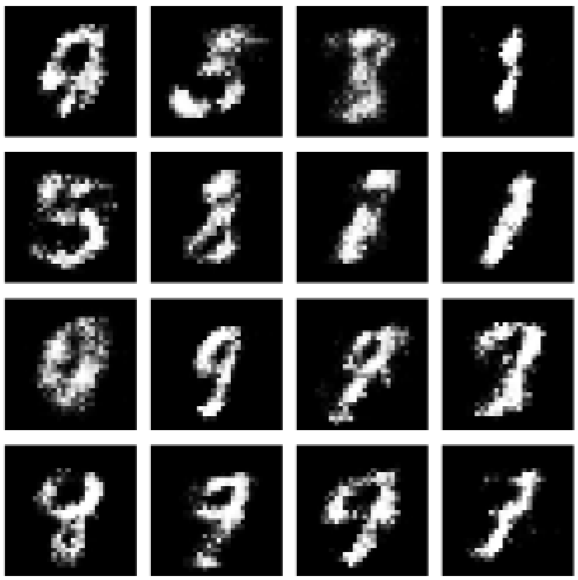
```
import tensorflow as tf

from tensorflow.keras import layers

import numpy as np

import matplotlib.pyplot as plt


# Load and preprocess MNIST dataset

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

x_train = x_train.astype("float32") / 255.0

x_train = x_train.reshape(-1, 784)  # Flatten the images


# Generator Model

def build_generator():

    model = tf.keras.Sequential([
```

```
        layers.Dense(128, activation="relu", input_shape=(100,)),
        layers.Dense(784, activation="sigmoid")
    ])
    return model

# Discriminator Model
def build_discriminator():
    model = tf.keras.Sequential([
        layers.Dense(128, activation="relu", input_shape=(784,)),
        layers.Dense(1, activation="sigmoid")
    ])
    return model

# Build and compile discriminator
discriminator = build_discriminator()
discriminator.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Build GAN by combining generator and discriminator
generator = build_generator()
discriminator.trainable = False

gan_input = tf.keras.Input(shape=(100,))
generated_image = generator(gan_input)
gan_output = discriminator(generated_image)
```

```
gan = tf.keras.Model(gan_input, gan_output)
gan.compile(optimizer='adam', loss='binary_crossentropy')

# Function to display generated images
def show_generated_images(generator, examples=16, dim=(4, 4), figsize=(6, 6)):
    noise = np.random.normal(0, 1, size=(examples, 100))
    generated_images = generator.predict(noise)
    generated_images = generated_images.reshape(-1, 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.show()

# Training Loop
def train_gan(epochs=10000, batch_size=128, display_interval=1000):
    for epoch in range(epochs):
        # Train Discriminator
        idx = np.random.randint(0, x_train.shape[0], batch_size)
        real_images = x_train[idx]

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

```
        fake_images = generator.predict(noise)

        X_combined = np.concatenate([real_images, fake_images])
        y_combined = np.concatenate([np.ones((batch_size, 1)), np.zeros((batch_size, 1))])

        discriminator.trainable = True
        d_loss, d_acc = discriminator.train_on_batch(X_combined, y_combined)

        # Train Generator
        noise = np.random.normal(0, 1, size=(batch_size, 100))
        y_gen = np.ones((batch_size, 1))  # Try to fool the discriminator

        discriminator.trainable = False
        g_loss = gan.train_on_batch(noise, y_gen)

        # Output logs
        if epoch % display_interval == 0 or epoch == epochs - 1:
            print(f"Epoch {epoch} | D Loss: {d_loss:.4f} | D Acc: {d_acc:.4f} | G Loss: {g_loss:.4f}")
            show_generated_images(generator)

# Train the GAN
train_gan(epochs=10000, batch_size=128, display_interval=1000)
```

Expected Output:



Result:

A Generative Adversarial Network (GAN) was successfully built and trained on the MNIST dataset. The generator was able to synthesize realistic handwritten digits over the course of training.