EXPERIMENT 7: Generative Models with GANs: Creating and Training a Generative Adversarial Network

AIM:

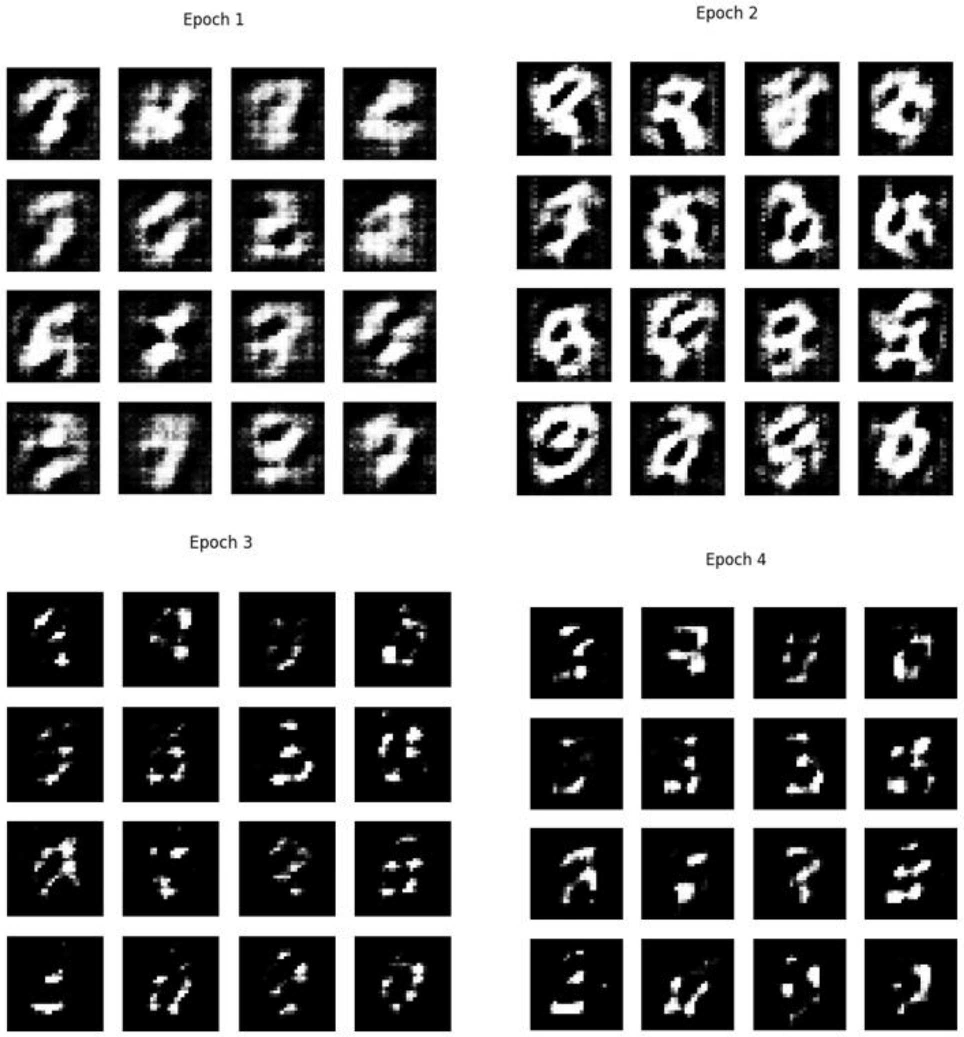
To construct and train a Generative Adversarial Network (GAN) using the TensorFlow/Keras framework. The objective is to train the GAN on the MNIST dataset to generate new, synthetic images of handwritten digits that are indistinguishable from the original training data.

SOURCE CODE:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
import os
(x_train, _), (_, _) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1).astype('float32')
x_train = (x_train - 127.5) / 127.5   # normalize to [-1,1]

latent_dim = 100
def build_generator():
   model = keras.Sequential([
      layers.Dense(7*7*128, use_bias=False, input_shape=(latent_dim,)),
      layers.BatchNormalization(),
      layers.LeakyReLU(),
      layers.Reshape((7, 7, 128)),
      layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding="same", use_bias=False),
      layers.BatchNormalization(),
      layers.LeakyReLU(),
      layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding="same",
                      use_bias=False, activation="tanh")
   ])
```

```
    return model

generator = build_generator()
def build_discriminator():
    model = keras.Sequential([
        layers.Conv2D(64, (5, 5), strides=(2, 2), padding="same", input_shape=[28, 28, 1]),
        layers.LeakyReLU(),
        layers.Dropout(0.3),
        layers.Flatten(),
        layers.Dense(1, activation="sigmoid")
    ])
    return model

discriminator = build_discriminator()
cross_entropy = keras.losses.BinaryCrossentropy(from_logits=False)

def discriminator_loss(real_output, fake_output):
    return (cross_entropy(tf.ones_like(real_output), real_output) +
            cross_entropy(tf.zeros_like(fake_output), fake_output))

def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)

generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
@tf.function
def train_step(images, batch_size, latent_dim):
    noise = tf.random.normal([batch_size, latent_dim])
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)
        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)
        gen_loss = generator_loss(fake_output)

        disc_loss = discriminator_loss(real_output, fake_output)

    generator_optimizer.apply_gradients(zip(gen_tape.gradient(gen_loss,
generator.trainable_variables),

                                            generator.trainable_variables))

    discriminator_optimizer.apply_gradients(zip(disc_tape.gradient(disc_loss,
discriminator.trainable_variables),

                                                discriminator.trainable_variables))

    return gen_loss, disc_loss


def generate_and_save_images(model, epoch, test_input):

    predictions = model(test_input, training=False)

    predictions_rescaled = (predictions * 0.5) + 0.5

    fig = plt.figure(figsize=(6, 6))

    for i in range(predictions.shape[0]):

        plt.subplot(4, 4, i + 1)

        plt.imshow(predictions_rescaled[i, :, :, 0], cmap="gray")

        plt.axis("off")

    plt.suptitle(f"Epoch {epoch}")

    plt.show()

EPOCHS = 5

batch_size = 64

num_examples_to_generate = 16

seed = tf.random.normal([num_examples_to_generate, latent_dim])

train_dataset = tf.data.Dataset.from_tensor_slices(x_train).shuffle(10000).batch(batch_size)

def train(dataset, epochs):

    print("\n--- Beginning Quick GAN Training ---")

    for epoch in range(epochs):

        for image_batch in dataset:

            gen_loss, disc_loss = train_step(image_batch, batch_size, latent_dim)

        print(f"Epoch {epoch + 1}/{epochs} - Generator Loss: {gen_loss:.4f}, Discriminator Loss:
{disc_loss:.4f}")

        generate_and_save_images(generator, epoch + 1, seed)
    print("\n--- Training complete. ---")
    generate_and_save_images(generator, epochs, seed)
train(train_dataset, EPOCHS)
```

OUTPUT:


Epoch 1
Epoch 2
Epoch 3
Epoch 4