EX NO:10

DATE:

SYNTHETIC IMAGES USING GENERATIVE ADVERSARIAL NETWORK

AIM:

To generate synthetic images resembling the MNIST handwritten digits using a Generative Adversarial Network (GAN). The GAN will learn the data distribution of MNIST digits and generate new realistic images.

ALGORITHM:

STEP 01: Load the MNIST dataset and normalize the images to the range [-1, 1].

STEP 02: Build the Generator network that takes random noise as input and outputs 28×28 images.

STEP 03: Build the discriminator network that takes an image as input and outputs a probability of being real or fake.

STEP 04: Compile the discriminator with binary crossentropy loss and an optimizer.

STEP 05: Combine the Generator and discriminator to form the GAN, keeping the discriminator non-trainable for GAN training.

STEP 06: For a number of epochs:

- Sample a batch of real images from the dataset.
- Generate a batch of fake images from random noise.
- Train the Discriminator on real images labeled 1 and fake images labeled 0.
- Train the Generator via the GAN to fool the Discriminator (label=1).

STEP 07: Periodically generate synthetic images using the trained Generator to visualize results.

STEP 08: After training, the Generator produces realistic handwritten digit images.

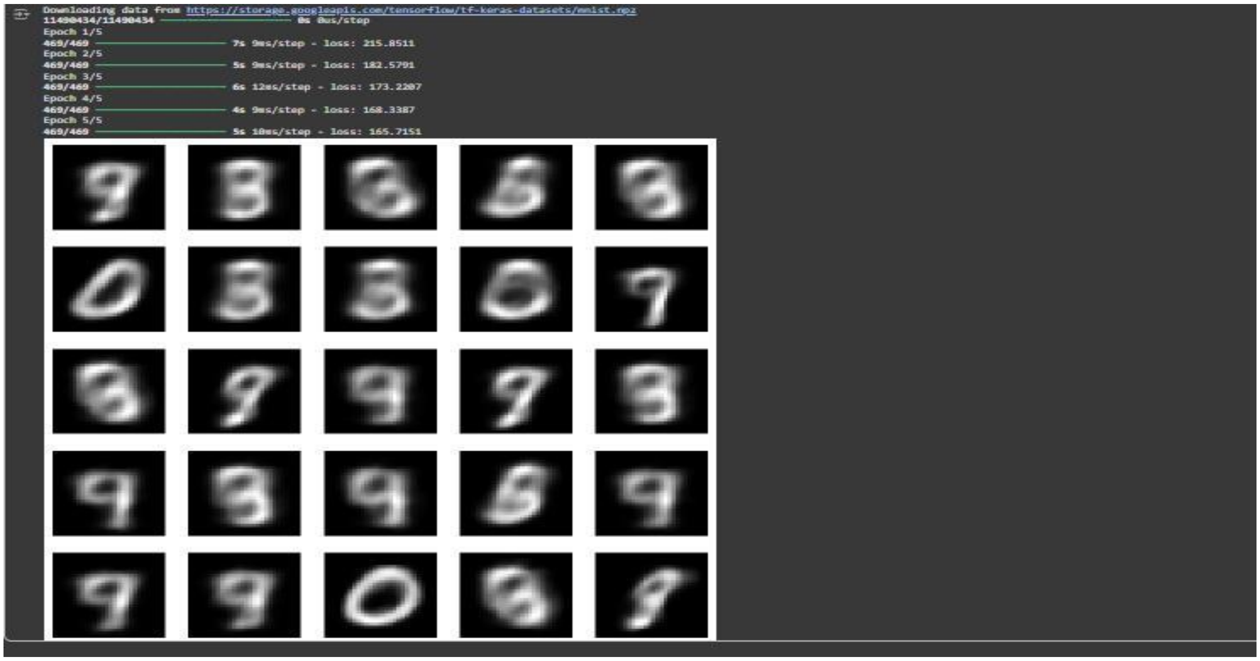
CODING:

```
import tensorflow as tf

from tensorflow.keras import layers, Model

import numpy as np

import matplotlib.pyplot as plt

#1. Load MNIST

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

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

x_train = np.expand_dims(x_train, -1) # (batch,28,28,1)

latent_dim = 2

# 2.Encoder

encoder_inputs = layers.Input(shape=(28,28,1))

x = layers.Flatten()(encoder_inputs)

x = layers.Dense(128, activation="relu")(x)

z_mean = layers.Dense(latent_dim)(x)

z_log_var = layers.Dense(latent_dim)(x)

def sampling(args):

    z_mean, z_log_var = args

    epsilon = tf.random.normal(shape=(tf.shape(z_mean)[0], latent_dim))

    return z_mean + tf.exp(0.5*z_log_var) * epsilon

z = layers.Lambda(sampling)([z_mean, z_log_var])

encoder = Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")
```

```
# 3.Decoder

latent_inputs = layers.Input(shape=(latent_dim,))

x = layers.Dense(128, activation="relu")(latent_inputs)

x = layers.Dense(28*28, activation="sigmoid")(x)

decoder_outputs = layers.Reshape((28,28,1))(x)

decoder = Model(latent_inputs, decoder_outputs, name="decoder")

#4. VAE Model

class VAE(Model):

    def _init_(self, encoder, decoder):

        super(VAE,self)._init_()

        self.encoder = encoder

        self.decoder = decoder

 def compile(self, optimizer):

        super(VAE,self).compile()

        self.optimizer = optimizer

 def train_step(self, data):

        if isinstance(data, tuple): data = data[0]

        with tf.GradientTape() as tape:

            z_mean, z_log_var, z = self.encoder(data)

            reconstruction = self.decoder(z)

            # Correct: reduce_sum over (1,2), not axis 3

            reconstruction_loss = tf.reduce_mean(
```

```
            tf.reduce_sum(tf.keras.losses.binary_crossentropy(data, reconstruction), axis=(1,2))
        )
        kl_loss = -0.5*tf.reduce_mean(tf.reduce_sum(1 + z_log_var - tf.square(z_mean) -
                                        tf.exp(z_log_var), axis=1))
        total_loss = reconstruction_loss + kl_loss
    grads = tape.gradient(total_loss, self.trainable_weights)
    self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
    return {"loss": total_loss}
# 5.Train VAE
vae = VAE(encoder, decoder)
vae.compile(tf.keras.optimizers.Adam())
vae.fit(x_train, epochs=5, batch_size=128)
# 6.Generate digits
n = 5
plt.figure(figsize=(10,10))
for i in range(n*n):
    z_sample = np.random.normal(size=(1,latent_dim))
    generated = decoder.predict(z_sample, verbose=0).reshape(28,28)
    plt.subplot(n,n,i+1)
    plt.imshow(generated, cmap="gray")
    plt.axis("off")
plt.show()
```

OUTPUT:



COE(20)	
RECORD(20)	
VIVA(10)	
TOTAL(50)	

RESULT:

The GAN successfully generates MNIST-like handwritten digits resembling the real dataset after training.