EX. NO: 08

DATE :

AI GENERATORS: DEEPDREAM AND NEURAL STYLE TRANSFER

AIM:

To understand and implement AI-based image generation techniques:

1. **Deep Dream** – Enhancing image patterns by amplifying features learned by a deep neural network.
2. **Neural Style Transfer (NST)** – Blending the style of one image with the content of another using convolutional neural networks.

ALGORITHM:

1. Deep Dream Algorithm

Step 1: Load a pre-trained Convolutional Neural Network (e.g., InceptionV3).
Step 2: Choose the intermediate layers from which features will be extracted.
Step 3: Load and preprocess the input image.
Step 4: Pass the image through the network to compute activations of chosen layers.
Step 5: Define a loss function as the sum of activations of selected layers.
Step 6: Compute gradients of the loss with respect to the input image.
Step 7: Normalize the gradients to avoid extreme changes.
Step 8: Update the image using **gradient ascent** (add gradients to the image).
Step 9: Repeat steps 4–8 for a fixed number of iterations.
Step 10: Deprocess the image and display the final "dream-like" output.

2. Neural Style Transfer Algorithm

Step 1: Load a pre-trained CNN model (e.g., VGG19).
Step 2: Select two input images:

- Content image (base image).
- Style image (artistic image).

Step 3: Preprocess both images (resize, normalize).
Step 4: Pass the content and style images through the network.
Step 5: Extract features from chosen layers for content and style representations.
Step 6: Define **content loss** (difference between content features of generated image and content image).
Step 7: Define **style loss** (difference between Gram matrices of style features and generated image features).
Step 8: Combine losses: **Total Loss = Content Loss + Style Loss (+ optional variation loss)**.
Step 9: Initialize a generated image (copy of content image).
Step 10: Use **gradient descent** to minimize the total loss by updating the generated image.
Step 11: Iterate optimization until the generated image shows content of the base image and

style of the style image.
Step 12: Display the final stylized output.

PROGRAM:

(A)DEEPDREAM

```
# Step 1: Install and Import Libraries
!pip install tensorflow matplotlib pillow

import tensorflow as tf
import numpy as np
import PIL.Image
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload Image from Your Computer
uploaded = files.upload()  # upload prompt will appear
image_path = list(uploaded.keys())[0]
print(f"Uploaded image: {image_path}")

# Step 3: Load & Display Image (Fixed for dimensions)
def load_image(path, max_dim=512):
    img = PIL.Image.open(path)
    img.thumbnail((max_dim, max_dim))
    img = np.array(img)/255.0
    # If grayscale, convert to RGB
    if img.ndim == 2:
        img = np.stack([img]*3, axis=-1)
    return img  # shape = (H, W, 3)

def show_image(img):
    plt.imshow(np.clip(img, 0, 1))
```

```
    plt.axis('off')
    plt.show()

img = load_image(image_path)
print("Original Image:")
show_image(img)

# Step 4: Load Pretrained Model
base_model = tf.keras.applications.InceptionV3(include_top=False, weights='imagenet')
layers = ['mixed3', 'mixed5']  # layers to enhance
dream_model = tf.keras.Model(inputs=base_model.input,
                             outputs=[base_model.get_layer(name).output for name in layers])

# Step 5: DeepDream Functions
def calc_loss(img, model):
    # Expand batch dimension here
    img_batch = tf.expand_dims(img, axis=0)
    layer_activations = model(img_batch)
    if not isinstance(layer_activations, list):
        layer_activations = [layer_activations]
    losses = [tf.reduce_mean(act) for act in layer_activations]
    return tf.reduce_sum(losses)

@tf.function
def deepdream_step(img, model, step_size=0.01):
    with tf.GradientTape() as tape:
        tape.watch(img)
        loss = calc_loss(img, model)
    gradients = tape.gradient(loss, img)
    gradients /= tf.math.reduce_std(gradients) + 1e-8
```

```
    img = img + step_size * gradients
    img = tf.clip_by_value(img, 0.0, 1.0)
    return img

# Step 6: Run DeepDream
def deep_dream(img, model, steps=50, step_size=0.01):
    img = tf.convert_to_tensor(img)
    for i in range(steps):
        img = deepdream_step(img, model, step_size)
        if i % 10 == 0:
            print(f"Step {i} completed")
            show_image(img.numpy())
    return img

dream_img = deep_dream(img, dream_model, steps=50, step_size=0.01)
print("DeepDream Image:")
show_image(dream_img.numpy())

# Step 7: Save the Result
result_image = PIL.Image.fromarray((dream_img.numpy()*255).astype(np.uint8))
result_image.save("deepdream_result.png")
print("Saved DeepDream image as deepdream_result.png")
```

OUTPUT:

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.12/dist-packages (2.19.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: pillow in /usr/local/lib/python3.12/dist-packages (11.3.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.32.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (4.15.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.17.3)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.75.0)
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.10.0)
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.14.0)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.5.3)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.4)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from astunparse>=1.6.0->tensorflow) (0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.17.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.4.3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (2025.8.3)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.9)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (0.7.2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from werkzeug>=1.0.1->tensorboard~=2.19.0->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->tensorflow) (4.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->tensorflow) (2.19.2)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow) (0.1.2)
Choose Files cat.jpg
```

```
Choose Files cat.jpg
cat.jpg(image/jpeg) - 242448 bytes, last modified: 10/1/2025 - 100% done
Saving cat.jpg to cat (2).jpg
Uploaded image: cat (2).jpg
Original Image:
```

```
Step 0 completed
```

```
Step 10 completed
```

```
Step 40 completed
```

```
DeepDream Image:
```

```
Saved DeepDream image as deepdream_result.png
```

(B)NEURAL STYLE TRANSFER (NST)

```
# ================================
# Fast Neural Style Transfer using TensorFlow Hub
# ================================

# Step 1: Install Libraries
!pip install tensorflow tensorflow-hub matplotlib pillow

import tensorflow as tf
import tensorflow_hub as hub
import numpy as np
import PIL.Image
import matplotlib.pyplot as plt
```

```
from google.colab import files

# Step 2: Upload Content and Style Images
print("Upload Content Image:")
uploaded_content = files.upload()
content_path = list(uploaded_content.keys())[0]

print("Upload Style Image:")
uploaded_style = files.upload()
style_path = list(uploaded_style.keys())[0]

# Step 3: Load Images
def load_image(path, max_dim=512):
    img = PIL.Image.open(path)
    img.thumbnail((max_dim, max_dim))
    img = np.array(img)/255.0
    if img.ndim == 2:  # grayscale -> RGB
        img = np.stack([img]*3, axis=-1)
    img = np.expand_dims(img, axis=0)  # add batch dimension
    return img.astype(np.float32)

def show_image(img, title=''):
    plt.imshow(np.squeeze(img))
    plt.axis('off')
    plt.title(title)
    plt.show()

content_img = load_image(content_path)
style_img = load_image(style_path)
```

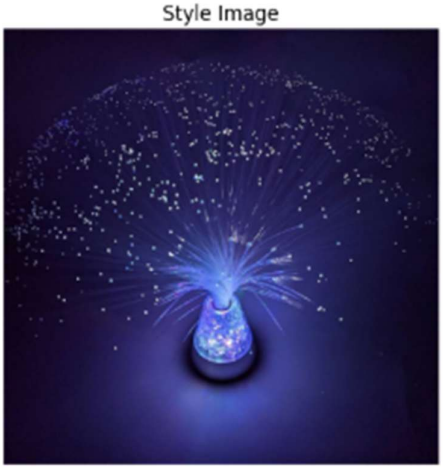
```
show_image(content_img, "Content Image")

show_image(style_img, "Style Image")


# Step 4: Load TensorFlow Hub NST Model

hub_model = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2')


# Step 5: Apply Style Transfer

stylized_image = hub_model(tf.constant(content_img), tf.constant(style_img))[0]


# Step 6: Display and Save Result

show_image(stylized_image, "Stylized Image")

result_image = PIL.Image.fromarray((np.squeeze(stylized_image.numpy())*255).astype(np.uint8))

result_image.save("nst_hub_result.png")

print("Saved stylized image as nst_hub_result.png")
```

OUTPUT:

```
  Downloading tf_keras-2.20.1-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.4)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.9.0
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from astunparse>=1.6.0->t
Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-packages (from keras>=3.10.0->tensorflow) (14.1.0
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-packages (from keras>=3.10.0->tensorflow) (0.1.0
Requirement already satisfied: optree in /usr/local/lib/python3.12/dist-packages (from keras>=3.10.0->tensorflow) (0.17
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tens
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0
Requirement already satisfied: markdown>=2.6.8 in /usr/lib/python3/dist-packages (from tensorboard~=2.20.0->tensorflow
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from t
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.20.0->te
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from werkzeug>=1.0.1->tens
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.10
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-packages (from markdown-it-py>=2.2.0->rich-
Downloading tensorflow_hub-0.16.1-py2.py3-none-any.whl (30 kB)
Downloading tf_keras-2.20.1-py3-none-any.whl (1.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 33.8 MB/s eta 0:00:00
Installing collected packages: tf-keras, tensorflow-hub
Successfully installed tensorflow-hub-0.16.1 tf-keras-2.20.1
Upload Content Image:
Choose Files  bird.jpg
bird.jpg(image/jpeg) - 71372 bytes, last modified: 10/1/2025 - 100% done
Saving bird.jpg to bird (1).jpg
Upload Style Image:
Choose Files  light.jpg
light.jpg(image/jpeg) - 87633 bytes, last modified: 10/1/2025 - 100% done
Saving light.jpg to light (1).jpg
```

Content Image

Style Image




```
Saved stylized image as nst_hub_result.png
```

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

RESULT:

The images first undergoes DeepDream to enhance patterns and textures,next Neural Style Transfer applies the chosen artistic style, producing a single dream-like, stylized artwork.

EX. NO: 09

DATE :

GENERATING SYNTHETIC IMAGES USING VARIATIONAL AUTOENCODERS (VAE)

AIM:

To **generate synthetic images** resembling the original dataset using a Variational Autoencoder (VAE), which learns a latent representation of data and allows sampling from this latent space.

ALGORITHM:

Step 1: Load and normalize the dataset (e.g., MNIST).

Step 2: Build the Encoder network to map input images to latent variables (z_mean and z_log_var).

Step 3: Apply the Reparameterization Trick to sample latent vector z from z_mean and z_log_var.

Step 4: Build the Decoder network to reconstruct images from latent vector z.

Step 5: Define the VAE loss as the sum of Reconstruction Loss and KL Divergence.

Step 6: Train the VAE using an optimizer (e.g., Adam).

Step 7: Generate synthetic images by sampling random latent vectors $z \sim N(0,1)$ and passing them through the Decoder.

PROGRAM:

```
# ============================
# Variational Autoencoder (VAE) on MNIST
# ============================

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Load Dataset
(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
x_train = np.expand_dims(x_train, -1)  # Shape: (num_samples, 28, 28, 1)
x_test = np.expand_dims(x_test, -1)

latent_dim = 2  # latent space dimension

# Step 2: Encoder
encoder_inputs = keras.Input(shape=(28, 28, 1))
x = layers.Flatten()(encoder_inputs)
x = layers.Dense(256, activation="relu")(x)
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)

# Sampling function
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 = keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")
encoder.summary()

# Step 3: Decoder
latent_inputs = keras.Input(shape=(latent_dim,))
x = layers.Dense(256, activation="relu")(latent_inputs)
x = layers.Dense(28 * 28, activation="sigmoid")(x)
decoder_outputs = layers.Reshape((28, 28, 1))(x)
decoder = keras.Model(latent_inputs, decoder_outputs, name="decoder")
decoder.summary()

# Step 4: VAE Model
class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder

    def call(self, inputs):
        z_mean, z_log_var, z = self.encoder(inputs)
        return self.decoder(z)

    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)
            # Reconstruction loss (sum over height & width)
            reconstruction_loss = tf.reduce_mean(
                tf.reduce_sum(
                    keras.losses.binary_crossentropy(data, reconstruction), axis=(1, 2)
                )
            )
            # KL divergence
            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}

vae = VAE(encoder, decoder)
vae.compile(optimizer=keras.optimizers.Adam())

# Step 5: Train
vae.fit(x_train, epochs=5, batch_size=128)

# Step 6a: Generate Random Synthetic Images (5x5 Grid)
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).squeeze()
    plt.subplot(n, n, i + 1)
```

```
    plt.imshow(generated, cmap="gray")
    plt.axis("off")
plt.tight_layout()
plt.show()

# Step 6b: Generate Smooth 2D Latent Space Grid (20x20)
n = 20
digit_size = 28
figure = np.zeros((digit_size * n, digit_size * n))

grid_x = np.linspace(-3, 3, n)
grid_y = np.linspace(-3, 3, n)[::-1]  # top to bottom

for i, yi in enumerate(grid_y):
    for j, xi in enumerate(grid_x):
        z_sample = np.array([[xi, yi]])
        x_decoded = decoder.predict(z_sample, verbose=0)
        digit = x_decoded.squeeze()
        figure[i * digit_size: (i + 1) * digit_size,
               j * digit_size: (j + 1) * digit_size] = digit

plt.figure(figsize=(10, 10))
plt.imshow(figure, cmap="gray")
plt.axis("off")
plt.show()
```

OUTPUT:

Model: "encoder"

Layer (type)	Output Shape	Param #	Connected to
input_layer_12 (InputLayer)	(None, 28, 28, 1)	0	-
flatten_6 (Flatten)	(None, 784)	0	input_layer_12[0…
dense_18 (Dense)	(None, 256)	200,960	flatten_6[0][0]
z_mean (Dense)	(None, 2)	514	dense_18[0][0]
z_log_var (Dense)	(None, 2)	514	dense_18[0][0]
lambda_6 (Lambda)	(None, 2)	0	z_mean[0][0], z_log_var[0][0]

Total params: 201,988 (789.02 KB)
Trainable params: 201,988 (789.02 KB)
Non-trainable params: 0 (0.00 B)

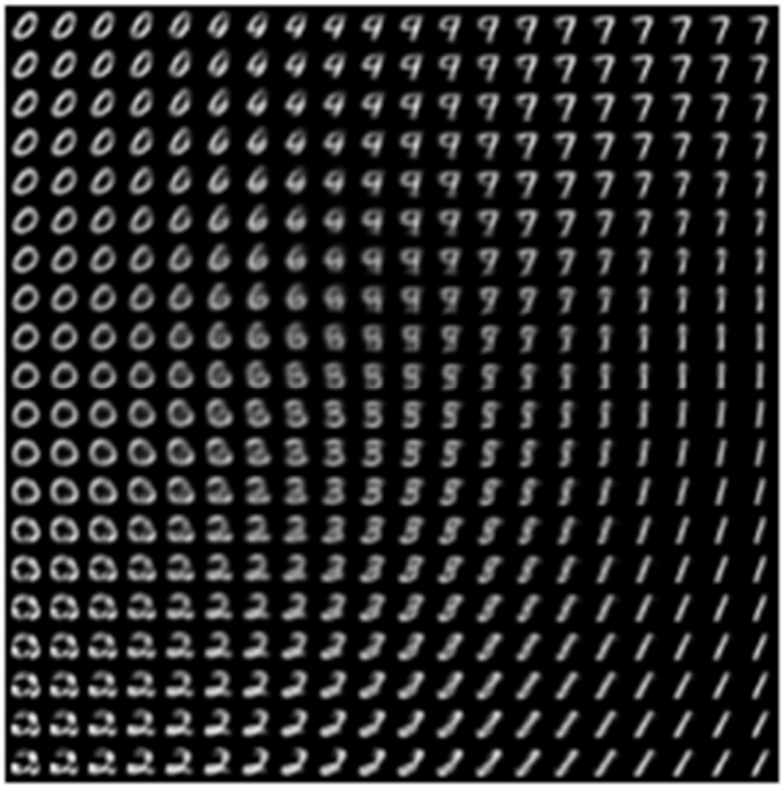
Model: "decoder"

Layer (type)	Output Shape	Param #
input_layer_13 (InputLayer)	(None, 2)	0
dense_19 (Dense)	(None, 256)	768
dense_20 (Dense)	(None, 784)	201,488
reshape_6 (Reshape)	(None, 28, 28, 1)	0

Total params: 202,256 (790.06 KB)
Trainable params: 202,256 (790.06 KB)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 203.1449
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 172.3901
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 168.1298
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 165.5113
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 163.2910
```

How can I install Python libraries?



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

RESULT:

The VAE generates synthetic images that resemble the original dataset and allows smooth exploration of the latent space.

EX. NO: 10

DATE :

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

Step 2: Build the **Generator network** that takes random noise as input and outputs 28×28 images.

Step 3: Build the **Discriminator network** that takes an image as input and outputs a probability of being real or fake.

Step 4: Compile the Discriminator with binary crossentropy loss and an optimizer.

Step 5: Combine the Generator and Discriminator to form the GAN, keeping the Discriminator non-trainable for GAN training.

Step 6: 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 7: Periodically generate synthetic images using the trained Generator to visualize results.

Step 8: After training, the Generator produces realistic handwritten digit images.

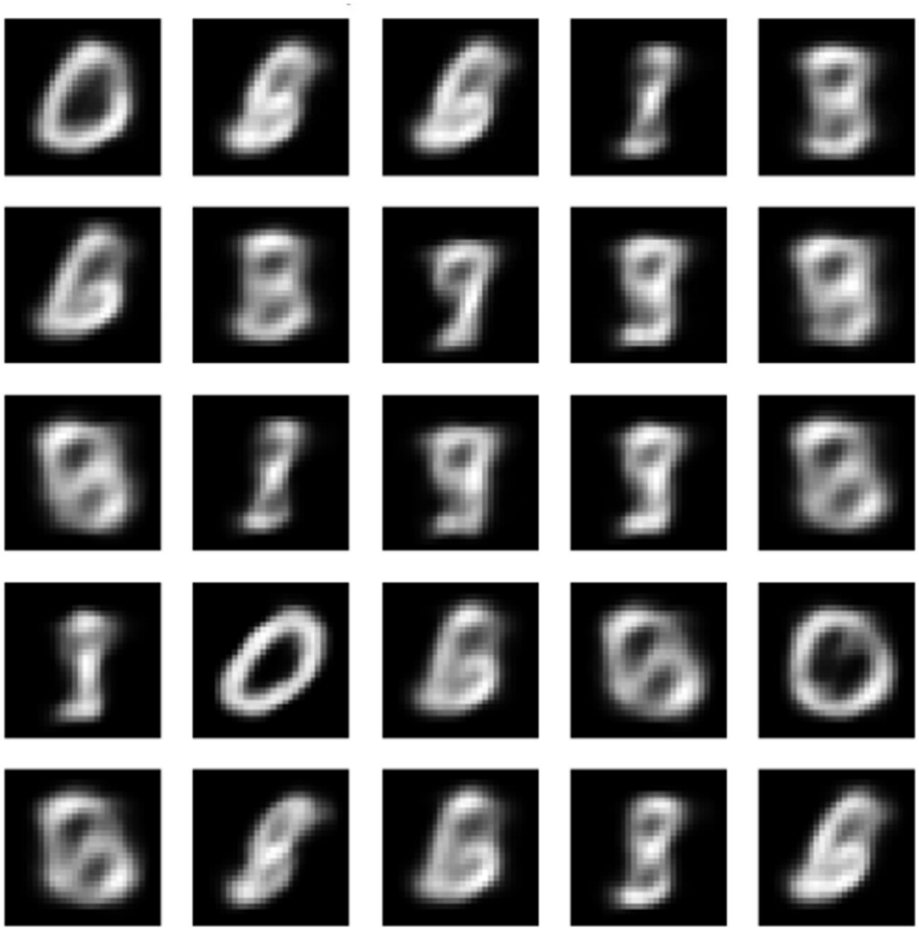
PROGRAM:

```
import tensorflow as tf
from tensorflow.keras import layers, Model
import numpy as np
import matplotlib.pyplot as plt

# ------------------------
# 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

# ------------------------
# 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")

# ------------------------
# 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")

# ------------------------
# 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}


# ------------------------
# Train VAE
# ------------------------
vae = VAE(encoder, decoder)
vae.compile(tf.keras.optimizers.Adam())
vae.fit(x_train, epochs=5, batch_size=128)


# ------------------------
# 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.