# DL-LAB-11-Activities

1. Demonstrate the working of **GAN (Generative Adversarial Network)** for image generation using **MNIST** / Equivalent dataset by implementing **DCGAN** on Keras/TensorFlow.
2. Compare the performance of GAN with VAE for image generation
3. Build GAN for generating images from text (refer to content in the next section)

**Quick References:**

- **Theory: pdfs**
- https://www.tensorflow.org/tutorials/generative/dcgan
- https://www.kaggle.com/code/adusumilligkrishna/building-a-gan-s-in-8-steps-from-scratch

**********

Reference for Exercise no. 3

**Build a GAN to generate image from text**

**Aim:**

Building a Generative Adversarial Network (GAN) to generate images from text involves training two neural networks simultaneously: a generator network and a discriminator network. The generator takes text descriptions as input and generates images, while the discriminator distinguishes between real images and those generated by the generator. Here's a step-by-step process.

**Procedure:**

**1. Prepare the Dataset:**

Obtain a dataset containing pairs of text descriptions and corresponding images.
Preprocess the text descriptions and images as necessary.

**2. Define the Generator Network:**

Design a generator network architecture, typically using a combination of convolutional and deconvolutional layers (also known as transpose convolutional layers).
The input to the generator will be the text descriptions, often encoded as vectors using techniques like Word Embeddings or text-to-image techniques.
The output of the generator will be synthesized images.

**3. Define the Discriminator Network:**

Design a discriminator network architecture, typically using convolutional layers.

The input to the discriminator will be either real images from the dataset or images generated by the generator.
The output of the discriminator will be a probability indicating whether the input image is real or fake.

**4. Define the GAN Model:**

Combine the generator and discriminator networks to form the GAN model.
The generator is trained to generate images that fool the discriminator, while the discriminator is trained to distinguish between real and fake images.

**5. Train the GAN:**

Train the GAN model on the dataset of text-image pairs.
During training, alternate between training the generator to minimize the discriminator's ability to distinguish fake images and training the discriminator to better distinguish between real and fake images.

**6. Evaluate the Performance:**

Evaluate the quality of generated images visually.
Use quantitative metrics such as Inception Score, Frechet Inception Distance (FID), or Precision and Recall to assess the quality of generated images.

**7. Fine-Tuning and Optimization:**

Fine-tune the hyperparameters of the GAN model to improve performance.
Experiment with different network architectures, training strategies, and optimization techniques to enhance the quality of generated images.

**Code snippet:**

Below is a basic outline of how you might implement a GAN for generating images from text using TensorFlow and Keras.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Reshape, Flatten, Conv2D, Conv2DTranspose, Embedding
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy
```

```
# Define Generator Network
generator = Sequential([
    # Input layer for text embedding
    Embedding(input_dim=vocab_size, output_dim=embedding_dim,
input_length=max_seq_length),
    # Additional layers (e.g., LSTM, Dense) to process text
embeddings
    # Reshape or flatten the processed text embeddings to match
the size of image input

    # Transpose convolutional layers to generate image from
processed text embeddings
    Conv2DTranspose(64, (5, 5), strides=(1, 1), padding='same',
activation='relu'),
    Conv2DTranspose(32, (5, 5), strides=(2, 2), padding='same',
activation='relu'),
    Conv2DTranspose(3, (5, 5), strides=(2, 2), padding='same',
activation='sigmoid')
])

# Define Discriminator Network
discriminator = Sequential([
    # Convolutional layers to process images
    Conv2D(32,   (5,   5),   strides=(2,   2),   padding='same',
input_shape=(image_height,     image_width,     image_channels),
activation='relu'),
    Conv2D(64,   (5,   5),   strides=(2,   2),   padding='same',
activation='relu'),
    Flatten(),
    # Dense layers for binary classification (real or fake)
    Dense(1, activation='sigmoid')
])

# Define GAN Model
gan = Sequential([generator, discriminator])

# Compile Discriminator (separately from GAN) with appropriate
optimizer and loss function
discriminator.compile(optimizer=Adam(),
loss=BinaryCrossentropy())

# Freeze Discriminator's weights during GAN training
discriminator.trainable = False
```

```
# Compile GAN with appropriate optimizer and loss function
gan.compile(optimizer=Adam(), loss=BinaryCrossentropy())

# Train GAN
for epoch in range(num_epochs):
    # Sample a batch of text descriptions and corresponding images
from the dataset
    idx = np.random.randint(0, len(images), batch_size)
    real_images = images[idx]
    real_text_descriptions = text_descriptions[idx]

    # Generate fake images using the generator based on the sampled
text descriptions
    fake_images = generator.predict(real_text_descriptions)

    # Concatenate real and fake images and corresponding labels
    x = np.concatenate([real_images, fake_images])
    y      =      np.concatenate([np.ones((batch_size,      1)),
np.zeros((batch_size, 1))])

    # Train the discriminator on the concatenated data
    discriminator_loss = discriminator.train_on_batch(x, y)

    # Sample a batch of text descriptions from the dataset
    idx = np.random.randint(0, len(images), batch_size)
    real_text_descriptions = text_descriptions[idx]

    # Train the GAN by updating the generator's weights while
keeping the discriminator's weights fixed
    misleading_targets = np.ones((batch_size, 1))
    gan_loss     =     gan.train_on_batch(real_text_descriptions,
misleading_targets)

    # Evaluate and visualize generated images periodically
    if epoch % evaluate_every == 0:
        # Generate sample images using a fixed set of text
descriptions
        sample_text_descriptions = fixed_text_descriptions
        sample_images                                          =
generator.predict(sample_text_descriptions)

        # Visualize sample images
     if epoch % visualize_interval == 0:
        visualize_images(fake_images)
```

```
        # Evaluate GAN performance using quantitative metrics
          if epoch % evaluation_interval == 0:
        evaluate_gan_performance(gan,   generator,   test_texts,
test_images)
```

**Explanation:**

- The generator network takes text descriptions as input and generates images.
- The discriminator network takes real images from the dataset and images generated by the generator as input and classifies them as real or fake.
- The GAN model combines the generator and discriminator networks and is trained to optimize both networks simultaneously.
- During training, the generator learns to generate images that resemble real images, while the discriminator learns to distinguish between real and fake images.
- The training loop alternates between training the discriminator and training the GAN. The discriminator is trained to classify real and fake images accurately, while the GAN is trained to generate images that fool the discriminator.
- Evaluation involves visually inspecting the quality of generated images and using quantitative metrics to assess their fidelity to real images.

**********