DEEP LEARNING

INDEX

Sr. No	Practical	Date	Sign
1	Introduction to TensorFlow	28/02/2025	
2	Linear Regression	28/02/2025	
3	Convolutional Neural Networks (Classification)	09/04/2025	
4	Write a program to implement deep learning techniques for image segmentation.	09/04/2025	
5	Write a program to predict a caption for a sample image using LSTM.	23/04/2025	
6	Applying the Autoencoder algorithms for encoding real-world data	26/04/2025	
7	Write a program for character recognition using RNN and compare it with CNN.	26/05/2025	
8	Write a program to develop Autoencoders using MNIST Handwritten Digits	26/05/2025	
9	Demonstrate recurrent neural network that learns to perform sequence analysis for stock price. (google stock price)	27/05/2025	
10	Applying Generative Adversarial Networks for image generation and unsupervised tasks.	27/05/2025	

PRACTICAL 01 : Introduction to TensorFlow

(A)

- **Create tensors with different shapes and data types.**
- **Perform basic operations like addition, subtraction, multiplication, and division on tensors.**
- **Reshape, slice, and index tensors to extract specific elements or sections.**
- **Performing matrix multiplication and finding eigenvectors and eigenvalues using TensorFlow**

CODE:-

```
!pip install tensorflow

import tensorflow as tf

tensorl = tf.constant ([[1, 2], [3, 4]], dtype=tf.float32)
tensor2 = tf.constant ([[5, 6], [7, 8]], dtype=tf.int32)
tensor3 = tf.random.uniform((3, 2, 4), dtype=tf.float64)

add_result = tf.add(tensorl, tf.cast (tensor2, tf.float32))
sub_result = tf.subtract (tensorl, tf.cast(tensor2, tf.float32))
mul_result = tf.multiply(tensorl, tf.cast (tensor2, tf.float32))
div_result = tf.divide (tensorl, tf.cast(tensor2, tf.float32))

reshaped = tf.reshape (tensor3, (6, 4))
sliced = tensor3 [:, 0:2, 1:3]
indexed = tensorl [0, 1]

matmul_result = tf.matmul(tensorl, tf.cast(tensor2, tf.float32))

eigenvalues, eigenvectors = tf.linalg.eigh(tensorl)

print("Tensorl:\n", tensorl)
print("Tensor2:\n", tensor2)
print("Tensor3:\n", tensor3)
print("Addition:\n", add_result)
print("Subtraction:\n", sub_result)
print("Multiplication:\n", mul_result)
print("Division:\n", div_result)
print("Reshaped:\n", reshaped)
print("Sliced:\n", sliced)
print("Indexed:\n", indexed)
print("Matrix Multiplication:\n", matmul_result)
```

```
print("Eigenvalues: \n", eigenvalues)
print("Eigenvectors:\n", eigenvectors)
```

OUTPUT:-

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\91982\Documents\all\Viqar\sem 4\Deep Learning\practical 1.a.py
Tensor1:
 tf.Tensor(
[[1. 2.]
 [3. 4.]], shape=(2, 2), dtype=float32)
Tensor2:
 tf.Tensor(
[[5 6]
 [7 8]], shape=(2, 2), dtype=int32)
Tensor3:
 tf.Tensor(
[[[0.05857522 0.20329664 0.84178525 0.63595551]
  [0.22615016 0.37736725 0.27095421 0.66988444]]

 [[0.89851106 0.56468965 0.29401177 0.05699739]
  [0.3585432  0.52824301 0.1222756  0.57636425]]

 [[0.42800161 0.62674304 0.52656743 0.53927206]
  [0.82661668 0.89376572 0.10340966 0.1662692 ]]], shape=(3, 2, 4), dtype=float64)
Addition:
 tf.Tensor(
[[ 6.  8.]
 [10. 12.]], shape=(2, 2), dtype=float32)
Subtraction:
 tf.Tensor(
[[-4. -4.]
 [-4. -4.]], shape=(2, 2), dtype=float32)
Multiplication:
 tf.Tensor(
[[ 5. 12.]
 [21. 32.]], shape=(2, 2), dtype=float32)
Division:
 tf.Tensor(
[[0.2        0.33333334]
 [0.42857143 0.5       ]], shape=(2, 2), dtype=float32)
Reshaped:
 tf.Tensor(
[[0.05857522 0.20329664 0.84178525 0.63595551]
 [0.22615016 0.37736725 0.27095421 0.66988444]
 [0.89851106 0.56468965 0.29401177 0.05699739]
 [0.3585432  0.52824301 0.1222756  0.57636425]
 [0.42800161 0.62674304 0.52656743 0.53927206]
 [0.82661668 0.89376572 0.10340966 0.1662692 ]], shape=(6, 4), dtype=float64)
Sliced:
 tf.Tensor(
[[[0.20329664 0.84178525]
  [0.37736725 0.27095421]]

 [[0.56468965 0.29401177]
  [0.52824301 0.1222756 ]]

 [[0.62674304 0.52656743]
  [0.89376572 0.10340966]]], shape=(3, 2, 2), dtype=float64)
Indexed:
 tf.Tensor(2.0, shape=(), dtype=float32)
Matrix Multiplication:
 tf.Tensor(
[[19. 22.]
 [43. 50.]], shape=(2, 2), dtype=float32)
Eigenvalues:
 tf.Tensor([-0.8541021  5.854102 ], shape=(2,), dtype=float32)
Eigenvectors:
 tf.Tensor(
[[-0.85065085 -0.5257311 ]
 [ 0.5257311  -0.85065085]], shape=(2, 2), dtype=float32)
>
```

(B) Program to solve the XOR problem.

CODE:-

```
!pip install tensorflow

import tensorflow as tf
import numpy as np

X= np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y=np.array([[0], [1], [1], [0]], dtype=np.float32)

model = tf.keras.Sequential ([
    tf.keras.layers.Dense (16, activation='relu', input_shape=(2,)),
    tf.keras.layers.Dense (1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.fit(X, y, epochs=100, verbose=0)

predictions=model.predict(X)

print("Input XOR Output")
for i in range(4):
  print (f" {X[i]} -> {predictions[i][0]:.4f}")
```

OUTPUT:-

```
====================
[1m1/1[0m [32m  ━
Input XOR Output
 [0. 0.] -> 0.1956
 [0. 1.] -> 0.9568
 [1. 0.] -> 0.9570
 [1. 1.] -> 0.0432
```

PRACTICAL 02 : Linear Regression

- **Implement a simple linear regression model using TensorFlow's lowlevel API (or tf. keras).**
- **Train the model on a toy dataset (e.g., housing prices vs. square footage).**
- **Visualize the loss function and the learned linear relationship.**
- **Make predictions on new data points.**

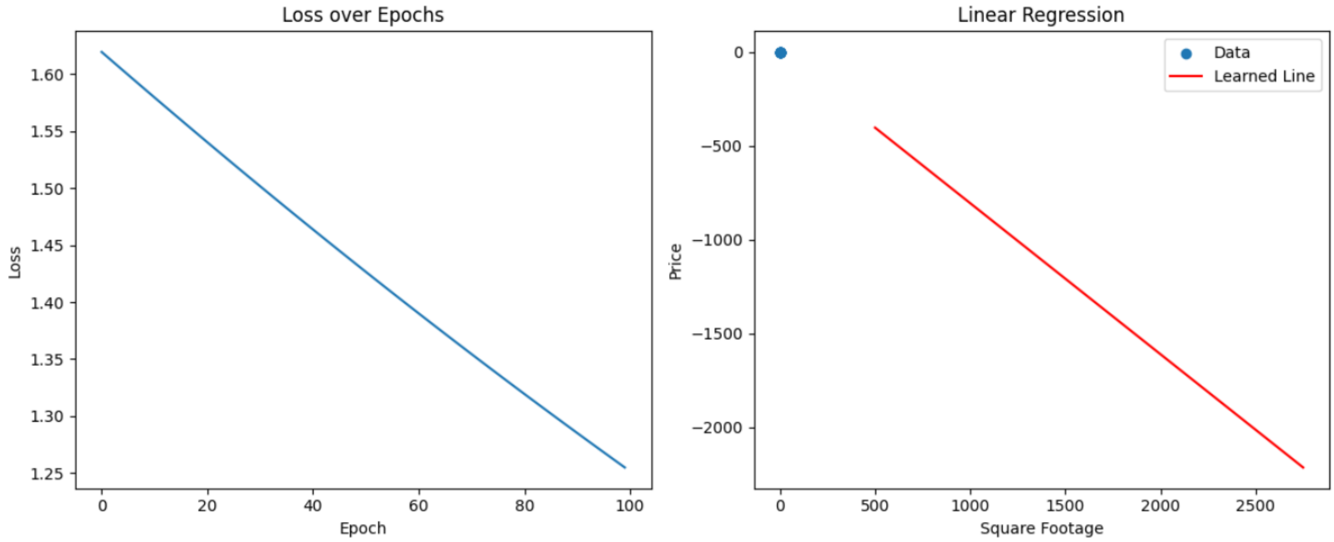
CODE:-

```
!pip install tensorflow
!pip install numpy
!pip install matplotlib

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)
X = np.array([500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500, 2750], dtype=np.float32)
y = np.array([200000, 250000, 300000, 350000, 400000, 450000, 500000, 550000, 600000, 650000],
dtype=np.float32)

X = X/np.max(X)
y = y/np.max(y)

model = tf.keras.Sequential ([
   tf.keras.layers.Input (shape=(1,)),
   tf.keras.layers. Dense (1)
])


model.compile(optimizer='adam', loss='mse')

history = model.fit(X, y, epochs=100, verbose=0)

plt.figure(figsize=(12,5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'])
plt.title('Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')

plt.subplot(1, 2, 2)
plt.scatter (X * np.max(X), y * np.max(y), label='Data')
X_pred = np.linspace (500, 2750, 100)
```

```
y_pred = model.predict(X_pred / np.max (X)) * np.max(y)
plt.plot(X_pred, y_pred, 'r', label='Learned Line')
plt.title('Linear Regression')
plt.xlabel('Square Footage')
plt.ylabel('Price')
plt.legend()

plt.tight_layout()
plt.savefig('linear_regression_plot.png')

X_new = np.array([800, 1200, 1800], dtype=np.float32)
y_new = model.predict (X_new/np.max(X)) * np.max(y)

print("New Predictions:")
for i in range (len (X_new)):
  print (f"Square Footage: {X_new[i]}, Predicted Price: {y_new[i].item():.2f}")
```

OUTPUT:-

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
New Predictions:
Square Footage: 800.0, Predicted Price: -644.43
Square Footage: 1200.0, Predicted Price: -966.69
Square Footage: 1800.0, Predicted Price: -1450.09
```



PRACTICAL 03 : Convolutional Neural Networks (Classification)

A. Implementing deep neural network for performing binary classification task.

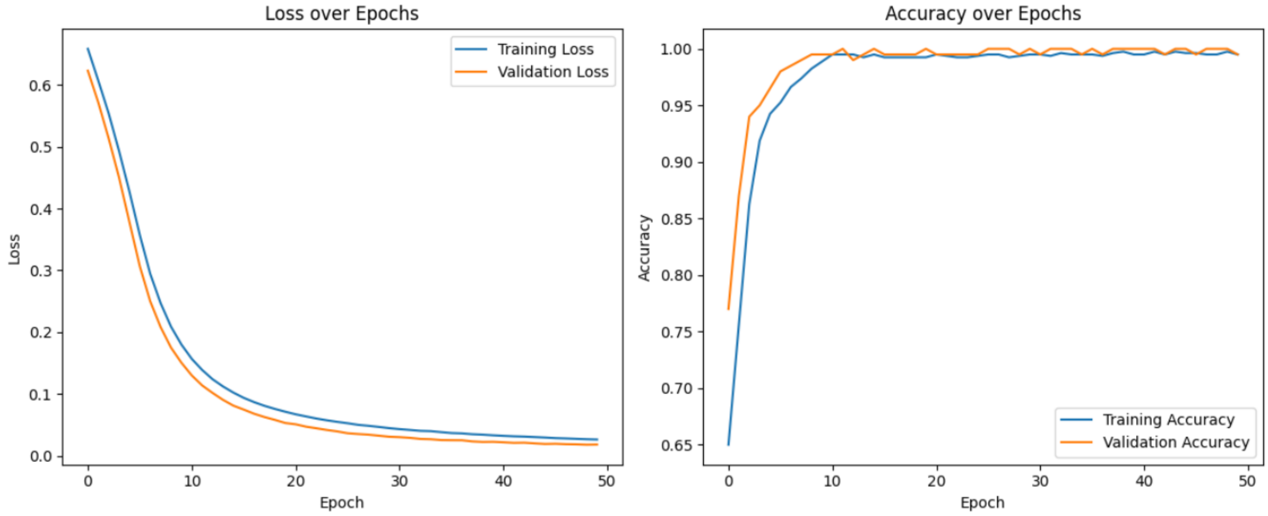
CODE:-

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
X = np.random.randn(1000, 2)
y = (X[:, 0] + X[:, 1] > 0).astype(np.float32).reshape(-1, 1)

model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(2,)),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(X, y, epochs=50, validation_split=0.2, verbose=0)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.savefig('binary_classification_plot.png')

X_test = np.array([[1, 1], [-1, -1], [0, 1]], dtype=np.float32)
```

```
predictions = model.predict(X_test)

print("Binary Classification Predictions:")
for i in range(len(X_test)):
    print(f"Input: {X_test[i]}, Predicted Class: {int(predictions[i] > 0.5)}")
```

OUTPUT:-

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
Binary Classification Predictions:
Input: [1. 1.], Predicted Class: 1
Input: [-1. -1.], Predicted Class: 0
Input: [0. 1.], Predicted Class: 1
/tmp/ipython-input-34-3223021697.py:46: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract
  print(f"Input: {X_test[i]}, Predicted Class: {int(predictions[i] > 0.5)}")
```



B. Using a deep feed-forward network with two hidden layers for performing multiclass classification and predicting the class.

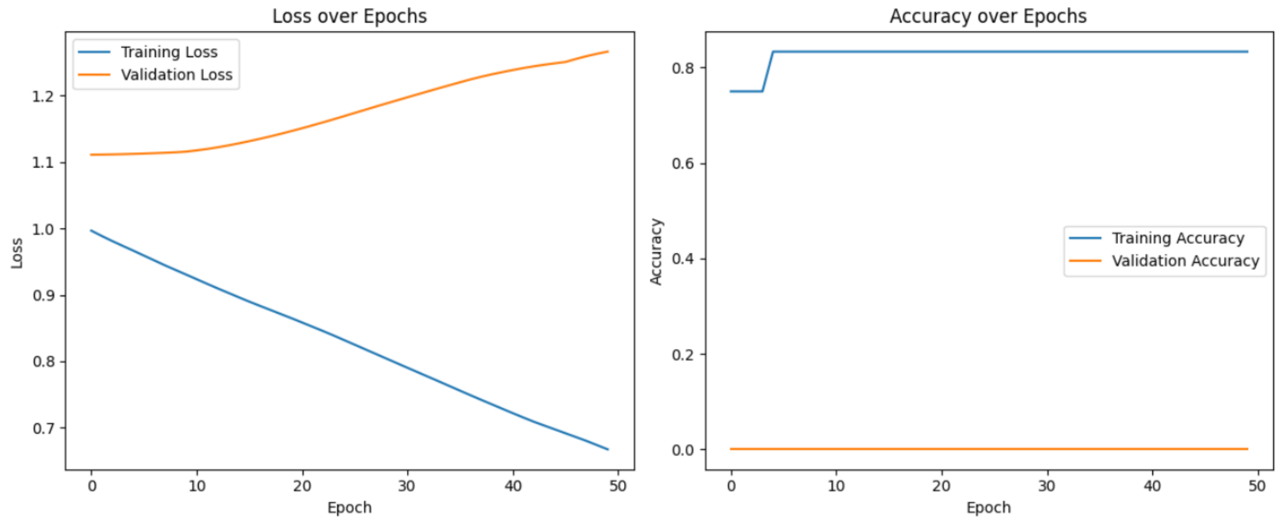
CODE:-

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

np.random.seed(42)
X = np.array([
    [5.1, 3.5, 1.4, 0.2], [4.9, 3.0, 1.4, 0.2], [4.7, 3.2, 1.3, 0.2], [4.6, 3.1, 1.5, 0.2],
    [5.0, 3.6, 1.4, 0.2], [5.4, 3.9, 1.7, 0.4], [4.6, 3.4, 1.4, 0.3], [6.9, 3.1, 4.9, 1.5],
    [5.5, 2.3, 4.0, 1.3], [6.5, 2.8, 4.6, 1.5], [6.3, 3.3, 6.0, 2.5], [5.8, 2.7, 5.1, 1.9],
    [7.5, 3.2, 6.8, 2.4], [6.5, 3.0, 5.8, 2.2]
], dtype=np.float32)

y = np.array([0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2], dtype=np.int32)
y = tf.keras.utils.to_categorical(y)

np.random.shuffle(np.c_[X, y])
X_train = X[:12]
y_train = y[:12]
X_test = X[12:]
y_test = y[12:]

X_mean = np.mean(X_train, axis=0)
X_std = np.std(X_train, axis=0)
X_train = (X_train - X_mean) / X_std
X_test = (X_test - X_mean) / X_std

model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(4,)),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=50, validation_data=(X_test, y_test), verbose=0)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epoch')
```

```
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.savefig("multiclass_classification_plot.png")

predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)
actual_classes = np.argmax(y_test, axis=1)

print("Multiclass Classification Predictions:")
for i in range(len(X_test)):
    print(f"Sample {i+1}: Predicted Class: {predicted_classes[i]}, Actual Class: {actual_classes[i]}")
```

OUTPUT:-

```
WARNING:tensorflow:6 out of the last 9 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x7ec9611d8900> triggere
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step
Multiclass Classification Predictions:
Sample 1: Predicted Class: 1, Actual Class: 2
Sample 2: Predicted Class: 1, Actual Class: 2
```



PRACTICAL 04

Write a program to implement deep learning Techniques for image segmentation.

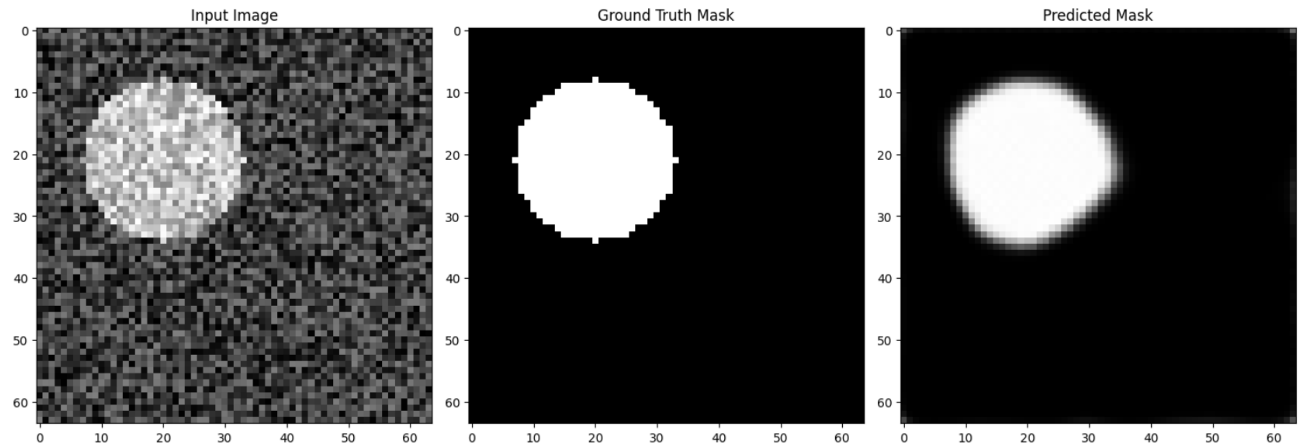
CODE:-

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

def create_toy_dataset(n_samples, img_size=64):
    X = np.zeros((n_samples, img_size, img_size, 1), dtype=np.float32)
    y = np.zeros((n_samples, img_size, img_size, 1), dtype=np.float32)

    for i in range(n_samples):
        img = np.random.random((img_size, img_size, 1)) * 0.5
        mask = np.zeros((img_size, img_size, 1))
        center = np.random.randint(20, 44, 2)
        radius = np.random.randint(10, 20)
        y_grid, x_grid = np.ogrid[:img_size, :img_size]
        mask_region = (x_grid - center[1])**2 + (y_grid - center[0])**2 <= radius**2
        mask[mask_region] = 1.0
        img[mask_region] += 0.5
        X[i] = img / np.max(img)
        y[i] = mask
    return X, y

X_train, y_train = create_toy_dataset(100)
X_test, y_test = create_toy_dataset(10)

def unet_model(input_shape=(64, 64, 1)):
    inputs = tf.keras.layers.Input(input_shape)
    c1 = tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same')(inputs)
    c1 = tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same')(c1)
    p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)

    c2 = tf.keras.layers.Conv2D(32, 3, activation='relu', padding='same')(p1)
    c2 = tf.keras.layers.Conv2D(32, 3, activation='relu', padding='same')(c2)
    p2 = tf.keras.layers.MaxPooling2D((2, 2))(c2)

    c3 = tf.keras.layers.Conv2D(64, 3, activation='relu', padding='same')(p2)
    c3 = tf.keras.layers.Conv2D(64, 3, activation='relu', padding='same')(c3)

    u4 = tf.keras.layers.Conv2DTranspose(32, 3, strides=(2, 2), padding='same')(c3)
    u4 = tf.keras.layers.Concatenate()([u4, c2])
```

```
    c4 = tf.keras.layers.Conv2D(32, 3, activation='relu', padding='same')(u4)
    c4 = tf.keras.layers.Conv2D(32, 3, activation='relu', padding='same')(c4)

    u5 = tf.keras.layers.Conv2DTranspose(16, 3, strides=(2, 2), padding='same')(c4)
    u5 = tf.keras.layers.Concatenate()([u5, c1])
    c5 = tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same')(u5)
    c5 = tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same')(c5)

    outputs = tf.keras.layers.Conv2D(1, 1, activation='sigmoid')(c5)
    return tf.keras.Model(inputs, outputs)

model = unet_model()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test), verbose=0)

plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.imshow(X_test[0, :, :, 0], cmap='gray')
plt.title("Input Image")

plt.subplot(1, 3, 2)
plt.imshow(y_test[0, :, :, 0], cmap='gray')
plt.title("Ground Truth Mask")

pred = model.predict(X_test[:1])[0, :, :, 0]
plt.subplot(1, 3, 3)
plt.imshow(pred, cmap='gray')
plt.title("Predicted Mask")

plt.tight_layout()
plt.savefig("segmentation_result.png")

print("Segmentation Accuracy:", history.history['val_accuracy'][-1])
```

OUTPUT:-

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 265ms/step
Segmentation Accuracy: 0.862841784954071
```



PRACTICAL 05

Write a program to predict a caption for a sample image using LSTM.

CODE:-

```
import torch
import torch.nn as nn
import torchvision.models as models

class EncoderCNN(nn.Module):
    def __init__(self, embed_size):
        super(EncoderCNN, self).__init__()
        resnet = models.resnet18(pretrained=True)
        for param in resnet.parameters():
            param.requires_grad = False  # Freeze backbone
        self.resnet = nn.Sequential(*list(resnet.children())[:-1])  # Remove last FC layer
        self.fc = nn.Linear(resnet.fc.in_features, embed_size)
        self.bn = nn.BatchNorm1d(embed_size, momentum=0.01)

    def forward(self, images):
        features = self.resnet(images)
        if features.dim() == 4:
            features = features.squeeze(-1).squeeze(-1)  # remove H and W if they are 1
        elif features.dim() == 2:
            pass  # already in [batch_size, feature_size] form
        features = self.bn(self.fc(features))
        return features

class DecoderRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers=1):
        super(DecoderRNN, self).__init__()
        self.embed = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, vocab_size)

    def forward(self, features, captions):
        embeddings = self.embed(captions[:, :-1])  # exclude <end>
        inputs = torch.cat((features.unsqueeze(1), embeddings), 1)
        hiddens, _ = self.lstm(inputs)
        outputs = self.fc(hiddens)
        return outputs

    def sample(self, features, max_len=20):
        sampled_ids = []
        inputs = features.unsqueeze(1)
        states = None
        for _ in range(max_len):
```

```
            hiddens, states = self.lstm(inputs, states)
            outputs = self.fc(hiddens.squeeze(1))
            _, predicted = outputs.max(1)
            sampled_ids.append(predicted.item())
            inputs = self.embed(predicted).unsqueeze(1)
            if predicted.item() == vocab['<end>']:
                break
        return sampled_ids



vocab = {'<pad>': 0, '<start>': 1, '<end>': 2, 'a': 3, 'dog': 4, 'on': 5, 'grass': 6}
inv_vocab = {v: k for k, v in vocab.items()}
vocab_size = len(vocab)

from PIL import Image
from torchvision import transforms

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

def load_image(image_path):
    image = Image.open(image_path).convert("RGB")
    image = transform(image).unsqueeze(0)
    return image

# Initialize
embed_size = 256
hidden_size = 512
encoder = EncoderCNN(embed_size)
decoder = DecoderRNN(embed_size, hidden_size, vocab_size)
encoder.eval()
decoder.eval()

# Load Image
img_path = "dog.jpg"  # replace with actual image
image_tensor = load_image(img_path)

# Forward Pass
with torch.no_grad():
    feature = encoder(image_tensor)
    sampled_ids = decoder.sample(feature)
    caption = [inv_vocab.get(word_id, "") for word_id in sampled_ids]
    result = ' '.join(caption).replace('<start>', '').replace('<end>', '')
    print("Predicted Caption:", result)
```

OUTPUT:-

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretraine
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a w
  warnings.warn(msg)
Predicted Caption: dog
```

PRACTICAL 06

Applying the Autoencoder algorithms for encoding real-world data

CODE:-

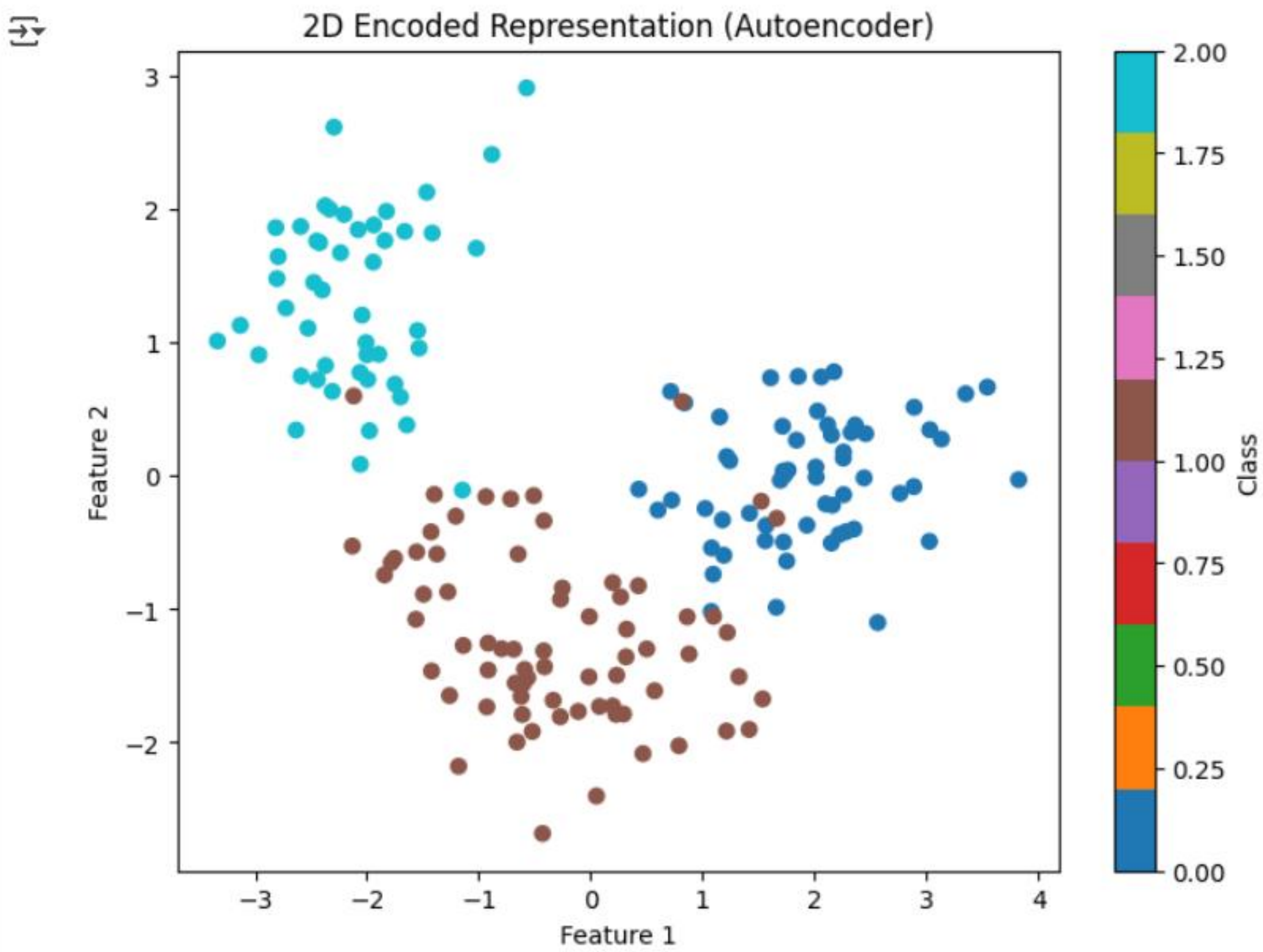
```
from sklearn.datasets import load_wine
import pandas as pd
import torch
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import StandardScaler

# Load dataset
data = load_wine()
df = pd.DataFrame(data.data, columns=data.feature_names)

# Normalize data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df)

# Convert to PyTorch Tensor
X_tensor = torch.tensor(X_scaled, dtype=torch.float32)

# DataLoader
dataset = TensorDataset(X_tensor, X_tensor)
loader = DataLoader(dataset, batch_size=32, shuffle=True)


import torch.nn as nn

class Autoencoder(nn.Module):
    def __init__(self, input_dim, encoding_dim):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 16),
            nn.ReLU(),
            nn.Linear(16, encoding_dim),
        )
        self.decoder = nn.Sequential(
            nn.Linear(encoding_dim, 16),
            nn.ReLU(),
            nn.Linear(16, input_dim),
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
```

```
input_dim = X_tensor.shape[1]
encoding_dim = 3  # dimensionality of latent space

model = Autoencoder(input_dim, encoding_dim)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

# Training loop
for epoch in range(50):
    total_loss = 0
    for batch_X, _ in loader:
        output = model(batch_X)
        loss = criterion(output, batch_X)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    if (epoch+1) % 10 == 0:
        print(f"Epoch {epoch+1}, Loss: {total_loss:.4f}")
```

```
Epoch 10, Loss: 5.0724
Epoch 20, Loss: 3.4669
Epoch 30, Loss: 2.7795
Epoch 40, Loss: 2.7278
Epoch 50, Loss: 2.4950
```

```
import matplotlib.pyplot as plt

model.eval()
with torch.no_grad():
    encoded_data = model.encoder(X_tensor).numpy()

# Plot 2D projection
plt.figure(figsize=(8, 6))
plt.scatter(encoded_data[:, 0], encoded_data[:, 1], c=data.target, cmap='tab10')
plt.title("2D Encoded Representation (Autoencoder)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.colorbar(label="Class")
plt.show()
```

OUTPUT:-


2D Encoded Representation (Autoencoder)
Feature 1
Feature 2
Class
−3
−2
−1
0
1
2
3
4
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

PRACTICAL 07

Write a program for character recognition using RNN and compare it with CNN.

CODE:-

```
[1] !pip install extra-keras-datasets
```

```
Collecting extra-keras-datasets
  Downloading extra_keras_datasets-1.2.0-py3-none-any.whl.metadata (982 bytes)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from extra-keras-datasets) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from extra-keras-datasets) (1.15.3)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from extra-keras-datasets) (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (from extra-keras-datasets) (1.6.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->extra-keras-datasets) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->extra-keras-datasets) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->extra-keras-datasets) (2025.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->extra-keras-datasets) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->extra-keras-datasets) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas->extra-keras-datasets) (1.17.0)
Downloading extra_keras_datasets-1.2.0-py3-none-any.whl (12 kB)
Installing collected packages: extra-keras-datasets
Successfully installed extra-keras-datasets-1.2.0
```

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Reshape, Flatten
from tensorflow.keras.utils import to_categorical

# Load MNIST digits dataset (0–9)
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# RNN Model
model_rnn = Sequential([
    Reshape((28, 28), input_shape=(28, 28)),
    SimpleRNN(128, activation='tanh'),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

model_rnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model_rnn.fit(x_train, y_train, epochs=5, validation_split=0.1)
print("RNN Accuracy:", model_rnn.evaluate(x_test, y_test)[1])
```

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D

model_cnn = Sequential([
    Reshape((28, 28, 1), input_shape=(28, 28)),
    Conv2D(32, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

model_cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model_cnn.fit(x_train, y_train, epochs=5, validation_split=0.1)
cnn_score = model_cnn.evaluate(x_test, y_test)
print("CNN Test Accuracy:", cnn_score[1])
```

OUTPUT:-

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/reshape.py:39: UserWarning: Do not pass an `input_shape`
  super().__init__(**kwargs)
Epoch 1/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 19s 10ms/step - accuracy: 0.8159 - loss: 0.5873 - val_accuracy: 0.9498 - val_loss: 0.1725
Epoch 2/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 16s 9ms/step - accuracy: 0.9366 - loss: 0.2100 - val_accuracy: 0.9643 - val_loss: 0.1238
Epoch 3/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 20s 9ms/step - accuracy: 0.9484 - loss: 0.1780 - val_accuracy: 0.9620 - val_loss: 0.1340
Epoch 4/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 17s 10ms/step - accuracy: 0.9540 - loss: 0.1567 - val_accuracy: 0.9677 - val_loss: 0.1167
Epoch 5/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 15s 9ms/step - accuracy: 0.9567 - loss: 0.1450 - val_accuracy: 0.9622 - val_loss: 0.1431
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9393 - loss: 0.2162
RNN Accuracy: 0.9505000114440918
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/reshape.py:39: UserWarning: Do not pass an `input_shape`
  super().__init__(**kwargs)
Epoch 1/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 27ms/step - accuracy: 0.9059 - loss: 0.3179 - val_accuracy: 0.9860 - val_loss: 0.0451
Epoch 2/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 80s 26ms/step - accuracy: 0.9853 - loss: 0.0461 - val_accuracy: 0.9877 - val_loss: 0.0406
Epoch 3/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 45s 26ms/step - accuracy: 0.9907 - loss: 0.0296 - val_accuracy: 0.9900 - val_loss: 0.0382
Epoch 4/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 43s 26ms/step - accuracy: 0.9932 - loss: 0.0213 - val_accuracy: 0.9913 - val_loss: 0.0318
Epoch 5/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 83s 26ms/step - accuracy: 0.9957 - loss: 0.0143 - val_accuracy: 0.9892 - val_loss: 0.0412
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9870 - loss: 0.0431
CNN Test Accuracy: 0.9887999892234802
```

PRACTICAL 08

Write a program to develop Autoencoders using MNIST Handwritten Digits.

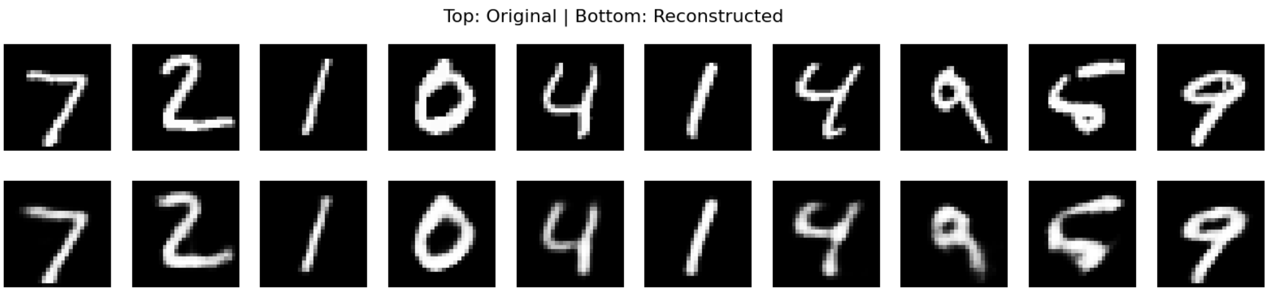
CODE:-

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist

# Load MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize and flatten
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), -1))  # shape: (60000, 784)
x_test = x_test.reshape((len(x_test), -1))     # shape: (10000, 784)

# Autoencoder architecture
input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)  # Latent space

decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)  # Output layer

autoencoder = Model(input_img, decoded)

# Compile model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train model
autoencoder.fit(x_train, x_train,
                epochs=20,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - loss: 0.3360 - val_loss: 0.1673
Epoch 2/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.1588 - val_loss: 0.1375
Epoch 3/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - loss: 0.1355 - val_loss: 0.1243
Epoch 4/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 18ms/step - loss: 0.1239 - val_loss: 0.1178
Epoch 5/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - loss: 0.1181 - val_loss: 0.1131
Epoch 6/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.1130 - val_loss: 0.1088
Epoch 7/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step - loss: 0.1091 - val_loss: 0.1057
Epoch 8/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.1063 - val_loss: 0.1035
Epoch 9/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - loss: 0.1040 - val_loss: 0.1012
Epoch 10/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 19ms/step - loss: 0.1020 - val_loss: 0.1000
Epoch 11/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.1004 - val_loss: 0.0986
Epoch 12/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0989 - val_loss: 0.0974
Epoch 13/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 19ms/step - loss: 0.0978 - val_loss: 0.0955
Epoch 14/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0965 - val_loss: 0.0947
Epoch 15/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.0953 - val_loss: 0.0940
Epoch 16/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step - loss: 0.0945 - val_loss: 0.0932
Epoch 17/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.0938 - val_loss: 0.0927
Epoch 18/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - loss: 0.0930 - val_loss: 0.0920
Epoch 19/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step - loss: 0.0927 - val_loss: 0.0910
Epoch 20/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 0.0919 - val_loss: 0.0906
<keras.src.callbacks.history.History at 0x7eddc4309ed0>
```

```python
# Predict and visualize
decoded_imgs = autoencoder.predict(x_test)

n = 10  # Number of digits to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.axis('off')

    # Reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.axis('off')

plt.suptitle("Top: Original | Bottom: Reconstructed", fontsize=16)
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

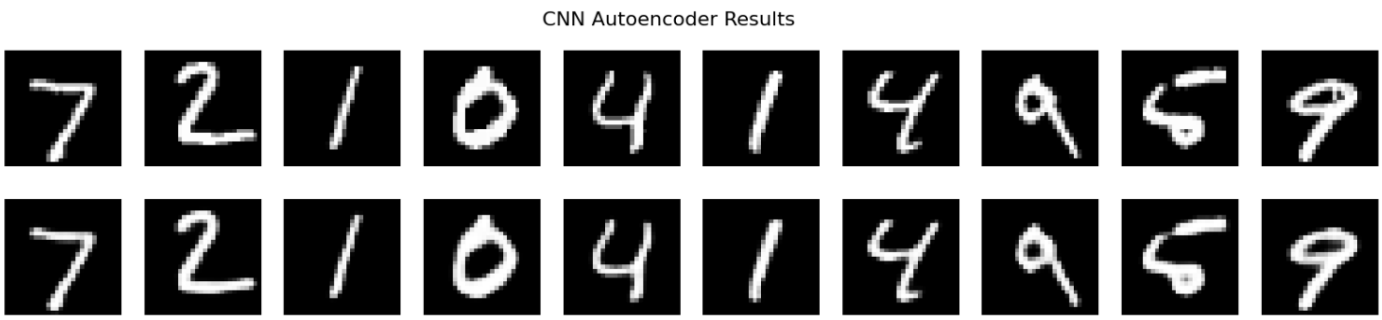


```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D

(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train[..., np.newaxis]  # shape: (60000, 28, 28, 1)
x_test = x_test[..., np.newaxis]

input_img = Input(shape=(28, 28, 1))

# Encoder
x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

# Decoder
x = Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder_cnn = Model(input_img, decoded)
autoencoder_cnn.compile(optimizer='adam', loss='binary_crossentropy')

# Train
autoencoder_cnn.fit(x_train, x_train,
                    epochs=10,
                    batch_size=128,
                    shuffle=True,
                    validation_data=(x_test, x_test))

# Visualize
decoded_imgs_cnn = autoencoder_cnn.predict(x_test)

plt.figure(figsize=(20, 4))
for i in range(n):
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.axis('off')

    ax = plt.subplot(2, n, i + 1 + n)
```

```
    plt.imshow(decoded_imgs_cnn[i].reshape(28, 28), cmap='gray')
    plt.axis('off')

plt.suptitle("CNN Autoencoder Results", fontsize=16)
plt.show()
```

OUTPUT:-

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 101s 210ms/step - loss: 0.2259 - val_loss: 0.0812
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 99s 211ms/step - loss: 0.0802 - val_loss: 0.0753
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 103s 220ms/step - loss: 0.0754 - val_loss: 0.0732
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 142s 220ms/step - loss: 0.0731 - val_loss: 0.0714
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 143s 222ms/step - loss: 0.0718 - val_loss: 0.0705
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 134s 204ms/step - loss: 0.0709 - val_loss: 0.0702
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 97s 206ms/step - loss: 0.0703 - val_loss: 0.0697
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 144s 211ms/step - loss: 0.0699 - val_loss: 0.0690
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 140s 208ms/step - loss: 0.0695 - val_loss: 0.0686
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 141s 207ms/step - loss: 0.0689 - val_loss: 0.0682
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step
```



PRACTICAL 09

Demonstrate recurrent neural network that learns to perform sequence analysis for stock price. (google stock price).

CODE:-

```
import yfinance as yf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense


# Download GOOGL stock data (last 5 years)
data = yf.download('GOOGL', start='2018-01-01', end='2023-12-31')
closing_prices = data['Close'].values.reshape(-1, 1)

# Normalize to [0, 1] range
scaler = MinMaxScaler (feature_range=(0, 1))
scaled_prices = scaler.fit_transform(closing_prices)
```

```
/tmp/ipython-input-10-1111475963.py:2: FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download('GOOGL', start='2018-01-01', end='2023-12-31')
[*********************100%***********************]  1 of 1 completed
```

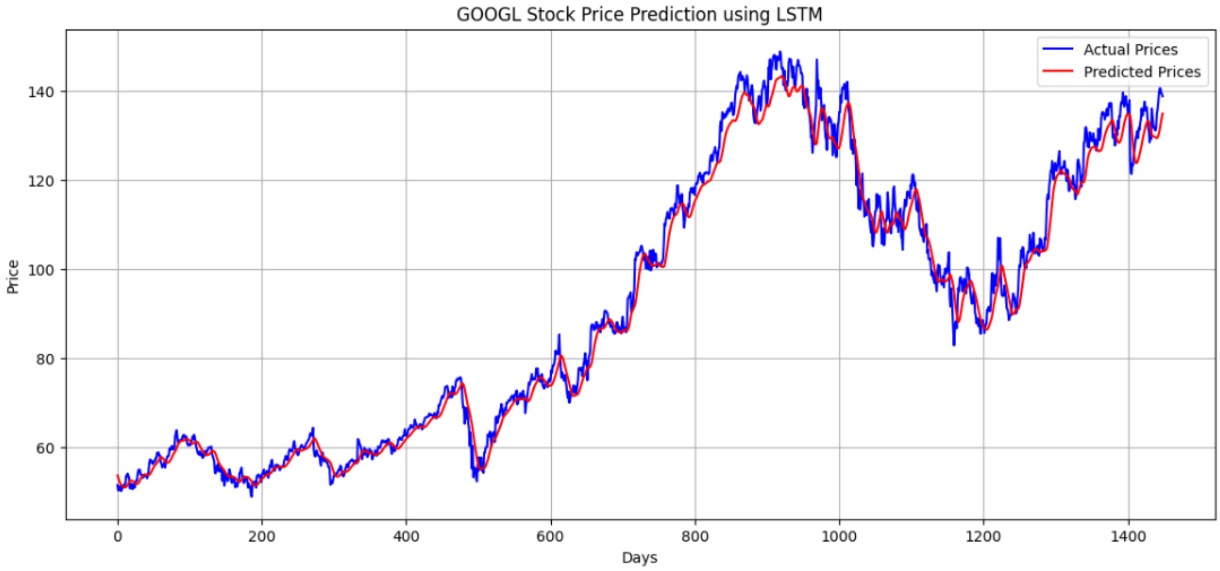
```
sequence_length = 60
X = []
y = []

for i in range(sequence_length, len(scaled_prices)):
  X.append(scaled_prices [i - sequence_length:i, 0])
  y.append(scaled_prices [i, 0])

X, y= np.array (X), np.array(y)
X = np.reshape (X, (X.shape[0], X.shape[1], 1)) # (samples, time steps, features)




model = Sequential ([
LSTM(units=50, return_sequences=True, input_shape=(X.shape[1], 1)),
LSTM(units=50),
Dense (1)
])
```

```
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X, y, epochs=10, batch_size=32)
```

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: User
  super().__init__(**kwargs)
46/46 ━━━━━━━━━━━━━━━━━━━━ 5s 46ms/step - loss: 0.0677
Epoch 2/10
46/46 ━━━━━━━━━━━━━━━━━━━━ 3s 62ms/step - loss: 0.0017
Epoch 3/10
46/46 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 0.0015
Epoch 4/10
46/46 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 0.0016
Epoch 5/10
46/46 ━━━━━━━━━━━━━━━━━━━━ 3s 45ms/step - loss: 0.0015
Epoch 6/10
46/46 ━━━━━━━━━━━━━━━━━━━━ 3s 49ms/step - loss: 0.0014
Epoch 7/10
46/46 ━━━━━━━━━━━━━━━━━━━━ 3s 64ms/step - loss: 0.0014
Epoch 8/10
46/46 ━━━━━━━━━━━━━━━━━━━━ 4s 47ms/step - loss: 0.0013
Epoch 9/10
46/46 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 0.0012
Epoch 10/10
46/46 ━━━━━━━━━━━━━━━━━━━━ 3s 46ms/step - loss: 0.0014
<keras.src.callbacks.history.History at 0x7eddebb9e110>
```

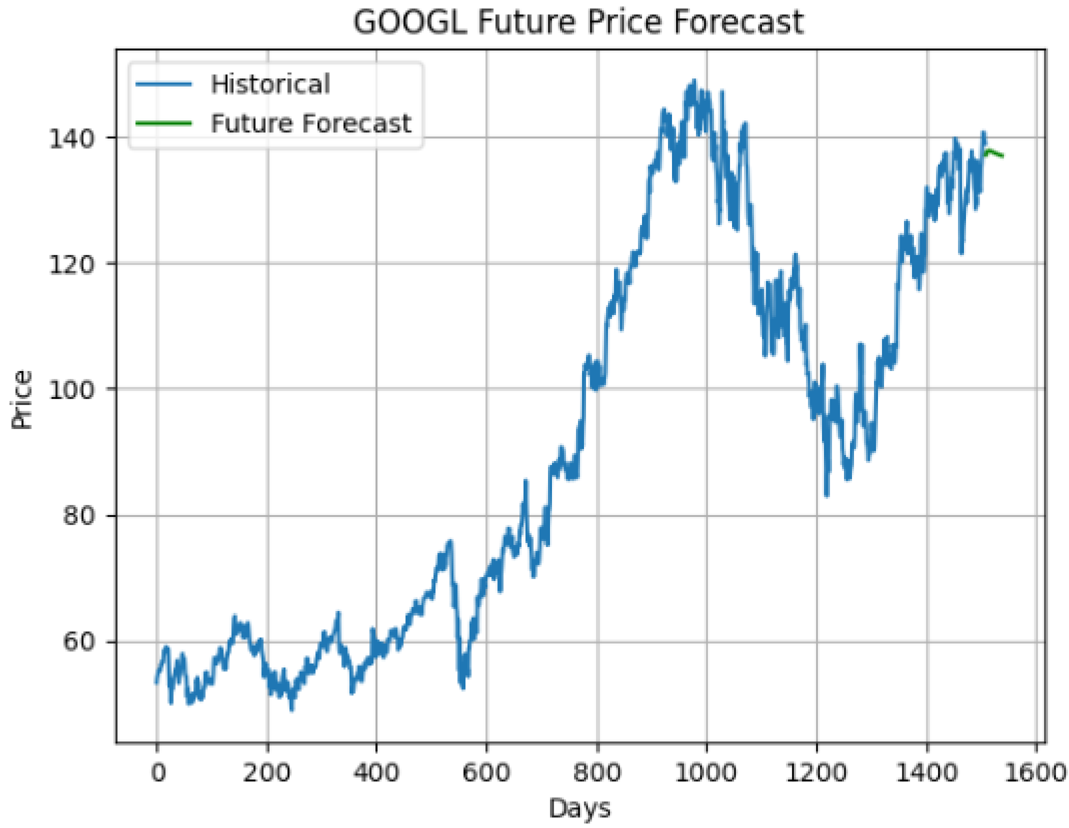
```
# Predict on training data to visualize fit
predicted_prices = model.predict(X)
predicted_prices = scaler.inverse_transform (predicted_prices)

# Inverse transform real prices
actual_prices = scaler.inverse_transform(y.reshape(-1, 1))

# Plot
plt.figure(figsize=(14, 6))
plt.plot(actual_prices, label="Actual Prices", color='blue')
plt.plot(predicted_prices, label="Predicted Prices", color='red')
plt.title("GOOGL Stock Price Prediction using LSTM")
plt.xlabel("Days")
plt.ylabel("Price")
plt.legend()
plt.grid(True)
plt.show()
```

46/46 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step




```
future_steps = 30
last_sequence = scaled_prices[-60:].reshape(1, 60, 1)
future_predictions = []

for _ in range(future_steps):
    next_price = model.predict(last_sequence)[0, 0]
    future_predictions.append(next_price)
    last_sequence = np.append(last_sequence[:, 1:, :], [[[next_price]]], axis=1)

# Inverse transform predicted values
future_prices = scaler.inverse_transform(np.array(future_predictions).reshape(-1, 1))

# Plot
plt.plot(range(len(closing_prices)), closing_prices, label="Historical")
plt.plot(range(len(closing_prices), len(closing_prices) + future_steps), future_prices, label="Future Forecast", color='green')
plt.title("GOOGL Future Price Forecast")
plt.xlabel("Days")
plt.ylabel("Price")
plt.legend()
plt.grid(True)
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
```



PRACTICAL 10

Applying Generative Adversarial Networks for image generation and unsupervised tasks.

CODE:-

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten, Reshape, Conv2D, Conv2DTranspose, LeakyReLU,
BatchNormalization
from tensorflow.keras.models import Sequential


(X_train, _), (_, _) = tf.keras.datasets.mnist.load_data()
X_train = (X_train.astype('float32') - 127.5) / 127.5 # Normalize to [-1, 1]
X_train = np.expand_dims (X_train, axis=-1) # (60000, 28, 28, 1)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

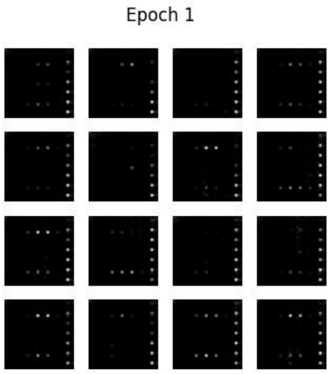
```
def build_generator():
    model = Sequential([
        Dense(7*7*256, use_bias=False, input_shape=(100,)),
        BatchNormalization(),
        LeakyReLU(),
        Reshape((7, 7, 256)),

        Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False),
        BatchNormalization(),
        LeakyReLU(),

        Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False),
        BatchNormalization(),
        LeakyReLU(),

        Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh')
    ])
    return model



def build_discriminator():
    model = Sequential([
        Conv2D(64, (5, 5), strides=(2, 2), padding='same',
               input_shape=[28, 28, 1]),
        LeakyReLU(),
```

```
        tf.keras.layers.Dropout(0.3),

        Conv2D(128, (5, 5), strides=(2, 2), padding='same'),
        LeakyReLU(),
        tf.keras.layers.Dropout(0.3),

        Flatten(),
        Dense(1)
    ])
    return model



cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

# Loss functions
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    return real_loss + fake_loss

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

# Models and optimizers
generator = build_generator()
discriminator = build_discriminator()
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)

@tf.function
def train_step(images):
    noise = tf.random.normal([batch_size, 100])

    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)

    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
```

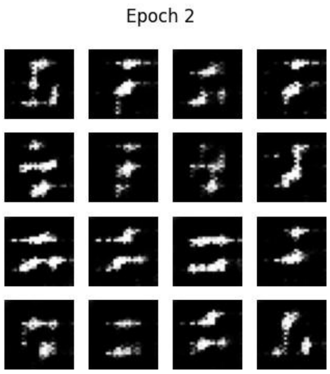
```
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
discriminator.trainable_variables))




train(train_dataset, epochs=30)
```

OUTPUT:-

```
2025-05-30 12:38:30.679578: I tensorflow/core/framework/local_rendezvous.cc:407] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
Epoch 1 completed in 152.71s
```

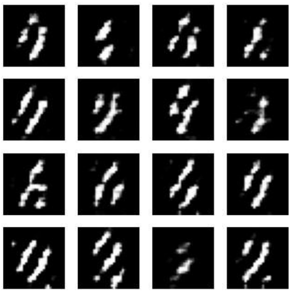



```
2025-05-30 12:41:14.820032: I tensorflow/core/framework/local_rendezvous.cc:407] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
Epoch 2 completed in 163.87s
```



```
Epoch 3 completed in 171.14s
```

Epoch 3



```
2025-05-30 12:46:52.118694: I tensorflow/core/framework/local_rendezvous.cc:407] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
Epoch 4 completed in 165.51s
```

Epoch 4

```
Epoch 5 completed in 165.59s
```

Epoch 5

```
Epoch 6 completed in 166.30s
```

Epoch 6

```
Epoch 7 completed in 157.21s
```

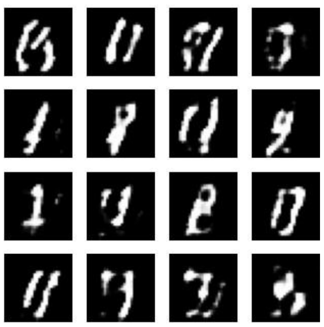
Epoch 7

```
2025-05-30 12:57:38.957773: I tensorflow/core/framework/local_rendezvous.cc:407] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
Epoch 8 completed in 156.82s
```

Epoch 8

```
Epoch 9 completed in 160.47s
```

Epoch 9



```
Epoch 10 completed in 165.25s
```

Epoch 10