Practical No. 1

Aim: Implementing advanced deep learning algorithms such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) using Python libraries like TensorFlow or PyTorch.

Code:

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
# Step 1: Load and Prepare the IMDb Dataset
max_features = 10000  # Use the top 10,000 most frequent words
maxlen = 100  # Limit each review to 100 words
# Load the dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
# Pad sequences to ensure all inputs are the same length
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
# Step 2: Define the RNN Model
model = Sequential([
    Embedding(max_features, 32, input_length=maxlen),  # Embedding layer
    SimpleRNN(32, activation='relu'),                  # RNN layer
    Dense(1, activation='sigmoid')                     # Output layer
])
# Step 3: Compile the Model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Step 4: Train the Model
model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.2)
# Step 5: Evaluate the Model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.2f}")
```

Output:

```
2025-01-04 15:49:17.701017: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use avail
able CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - accuracy: 0.5614 - loss: 0.6728 - val_accuracy: 0.7116 - val_loss: 0.5599
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.8028 - loss: 0.4442 - val_accuracy: 0.8270 - val_loss: 0.3967
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.8935 - loss: 0.2607 - val_accuracy: 0.8320 - val_loss: 0.3981
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.9266 - loss: 0.1943 - val_accuracy: 0.8398 - val_loss: 0.4102
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.9525 - loss: 0.1421 - val_accuracy: 0.8254 - val_loss: 0.4433
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8247 - loss: 0.4381
Test Accuracy: 0.83
```

Practical No. 2

Aim: Building a natural language processing (NLP) model for sentiment analysis or text classification.

Code:

```
from transformers import pipeline
# Load the pre-trained sentiment-analysis pipeline
sentiment_analyzer = pipeline('sentiment-analysis')
# Example texts to classify
texts = [
    "I love this product, it's amazing!",
    "This is the worst service I've ever had.",
    "I'm so happy with my purchase, highly recommend!",
    "I'm not satisfied at all with this experience."
]
# Function to analyze sentiment
def analyze_sentiment(texts):
    for text in texts:
        result = sentiment_analyzer(text)
        label = result[0]['label']
        score = result[0]['score']
        print(f"Text: {text}\nSentiment: {label} (Confidence: {score:.2f})\n")
# Call the function to classify sentiments
analyze_sentiment(texts)
```

Output:

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f (https://huggingface.co/di
stilbert/distilbert-base-uncased-finetuned-sst-2-english).
Using a pipeline without specifying a model name and revision in production is not recommended.
Text: I love this product, it's amazing!
Sentiment: POSITIVE (Confidence: 1.00)

Text: This is the worst service I've ever had.
Sentiment: NEGATIVE (Confidence: 1.00)

Text: I'm so happy with my purchase, highly recommend!
Sentiment: POSITIVE (Confidence: 1.00)

Text: I'm not satisfied at all with this experience.
Sentiment: NEGATIVE (Confidence: 1.00)
```

Practical No. 3

Aim: Creating a chatbot using advanced techniques like transformer models

Code:

```
from transformers import pipeline
# Step 1: Load a Pre-trained Transformer Model
chatbot = pipeline("text-generation", model="microsoft/DialoGPT-medium")

# Step 2: Start a Chat Session
print("Chatbot: Hello! I'm here to chat with you. Type 'exit' to end the conversation.")

# Step 3: Loop for Chatting
while True:
    user_input = input("You: ")
        if user_input.lower() == "exit":
        print("Chatbot: Goodbye!")
        break
        # Generate a Response
    response = chatbot(user_input, max_length=50, num_return_sequences=1)
    print("Chatbot:", response[0]['generated_text'])
```

Output:

```
Chatbot: Hello! I'm here to chat with you. Type 'exit' to end the conversation.
You: Hi
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Def
aulting to 'longest_first' truncation strategy. If you encode pairs of sequences (GLUE-style) with the tokenizer you can select this strategy more precisely by provid
ing a specific strategy to `truncation`.
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
Chatbot: Hi and welcome to the party!
You:
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's `attention_mask` to obtain relia
ble results.
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
The attention mask is not set and cannot be inferred from input because pad token is same as eos token. As a consequence, you may observe unexpected behavior. Please
pass your input's `attention_mask` to obtain reliable results.
Chatbot:
You: Hi
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
Chatbot: Hi and welcome to the party!
You:
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's `attention_mask` to obtain relia
ble results.
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
Chatbot:
You: How are you
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
Chatbot: How are you irl?
You:
```

Practical No. 4

Aim: Developing a recommendation system using collaborative filtering or deep learning approaches.

Code:

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import StandardScaler
# Step 1: Load dataset
df = pd.read_csv('C:/Users//Desktop/ml-latest-small/ratings.csv') # Assuming columns: userId, movieId, rating
df = pd.read_csv('C:\\Users\\Admin\\OneDrive\\Desktop\\pract\\ratings.csv')
# Step 2: Create user-item interaction matrix
interaction_matrix = df.pivot(index='userId', columns='movieId', values='rating').fillna(0)
# Step 3: Normalize the data (optional but helps with similarity calculation)
scaler = StandardScaler(with_mean=False)
interaction_matrix_scaled = scaler.fit_transform(interaction_matrix)
# Step 4: Compute user-user similarity
user_similarity = cosine_similarity(interaction_matrix_scaled)
user_similarity_df = pd.DataFrame(user_similarity, index=interaction_matrix.index,
columns=interaction_matrix.index)
# Step 5: Generate recommendations
def recommend(user_id, k=5):
    # Find similar users
    similar_users = user_similarity_df[user_id].sort_values(ascending=False)[1:k+1]
    # Collect weighted ratings from similar users
    similar_users_ratings = interaction_matrix.loc[similar_users.index]
    weighted_ratings = similar_users_ratings.T.dot(similar_users)
    # Exclude movies already rated by the user
    user_rated = interaction_matrix.loc[user_id]
    recommendations = weighted_ratings[user_rated == 0].sort_values(ascending=False).head(k)
    return recommendations.index.tolist()
# Example: Recommend movies for user ID
user_id = int(input("Enter your input"))
recommendations = recommend(user_id)
print(f"Recommendations for User {user_id}: {recommendations}")
```

Output:

```
Enter your input: 2
Recommendations for User 2: [2959, 527, 1246, 116797, 7153]
```

Practical No. 5

Aim: Implementing a computer vision project, such as <u>object detection</u> or image segmentation.

Requirements: Python: 3.8.10

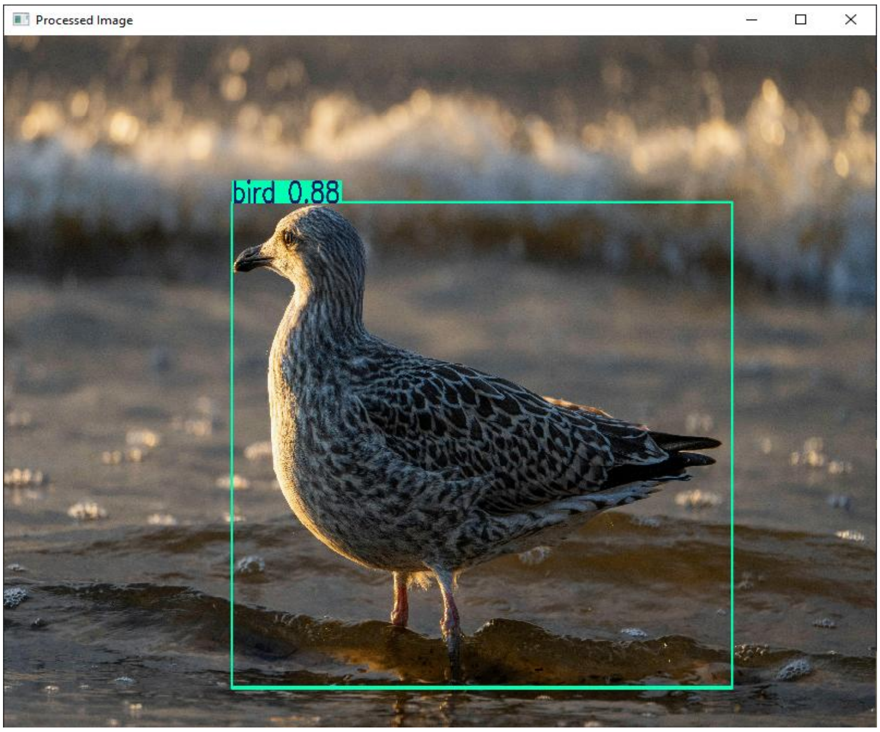
Code:

```
from ultralytics import YOLO
import cv2
# Load the YOLO model
model = YOLO("C:/Users/Desktop/Yolo-Weights/yolov8n.pt")
# Process the image without automatically showing it
results = model("C:/Users//Desktop//p5/imgs/1.jpg")
# If results is a list, access the first element (which should contain the image)
image = results[0].plot()  # Plot the results (draw bounding boxes, etc.)
# Resize the image to a suitable size before displaying
resized_image = cv2.resize(image, (800, 800))  # Adjust 800x800 to your preferred size
# Create the OpenCV window with a normal resizing option
cv2.namedWindow("Processed Image", cv2.WINDOW_NORMAL)
# Resize the window to match the size of the image
cv2.resizeWindow("Processed Image", resized_image.shape[1], resized_image.shape[0])
# Display the resized image in the window
cv2.imshow("Processed Image", resized_image)
# Wait for a key press and close the window
cv2.waitKey(0)
cv2.destroyAllWindows()
```

model = YOLO("C:/Users/Admin/Desktop/Yolo-Weights/yolov8n.pt")

results = model("C:\\Users\\Admin\\OneDrive\\Desktop\\pract\\dog.jpg")

Output



Practical No. 6

Aim: Applying reinforcement learning algorithms to solve complex decision-making problems.

Code:

```
import numpy as np
import random
# Define the environment
grid_size = 3  # Smaller grid
goal_state = (2, 2)
obstacle_state = (1, 1)  # Single obstacle
actions = ['up', 'down', 'left', 'right']
action_to_delta = {
    'up': (-1, 0),
    'down': (1, 0),
    'left': (0, -1),
    'right': (0, 1)
}
# Initialize Q-table (simple 3D array for states and actions)
q_table = np.zeros((grid_size, grid_size, len(actions)))
# Parameters
alpha = 0.1  # Learning rate
gamma = 0.9  # Discount factor
epsilon = 1.0  # Exploration rate
epsilon_decay = 0.99
min_epsilon = 0.1
episodes = 200  # Fewer episodes
# Reward function
def get_reward(state):
    if state == goal_state:
        return 10  # Reward for reaching the goal
    elif state == obstacle_state:
        return -10  # Penalty for hitting the obstacle
    return -1  # Step penalty
# Check if the new state is valid
def is_valid_state(state):
    return 0 <= state[0] < grid_size and 0 <= state[1] < grid_size and state != obstacle_state
# Main Q-learning loop
for episode in range(episodes):
    state = (0, 0)  # Start at the top-left corner
    total_reward = 0
    while state != goal_state:
        # Choose an action (epsilon-greedy strategy)
        if random.uniform(0, 1) < epsilon:
            action = random.choice(actions)  # Explore
        else:
            action = actions[np.argmax(q_table[state[0], state[1]])]  # Exploit best action
```

```
        # Perform the action
        delta = action_to_delta[action]
        next_state = (state[0] + delta[0], state[1] + delta[1])
        # Stay in the same state if the move is invalid
        if not is_valid_state(next_state):
            next_state = state
        # Get reward and update Q-table
        reward = get_reward(next_state)
        total_reward += reward
        best_next_action = np.max(q_table[next_state[0], next_state[1]])
        q_table[state[0], state[1], actions.index(action)] += alpha * (
            reward + gamma * best_next_action - q_table[state[0], state[1], actions.index(action)]
        )
        # Update state
        state = next_state
    # Decay epsilon
    epsilon = max(min_epsilon, epsilon * epsilon_decay)
    print(f"Episode {episode + 1}: Total Reward = {total_reward}")
# Display the learned policy
policy = np.full((grid_size, grid_size), ' ')
for i in range(grid_size):
    for j in range(grid_size):
        if (i, j) == goal_state:
            policy[i, j] = 'G'  # Goal
        elif (i, j) == obstacle_state:
            policy[i, j] = 'X'  # Obstacle
        else:
            best_action = np.argmax(q_table[i, j])
            policy[i, j] = actions[best_action][0].upper()  # First letter of the best action
print("Learned Policy:")
print(policy)
```

Output:

```
Episode 1: Total Reward = 2
Episode 2: Total Reward = -2
Episode 3: Total Reward = 0
Episode 4: Total Reward = -26
Episode 5: Total Reward = -74
Episode 6: Total Reward = -22
Episode 7: Total Reward = -5
Episode 8: Total Reward = -4
Episode 9: Total Reward = -31
Episode 10: Total Reward = -20
Episode 11: Total Reward = -6
Episode 12: Total Reward = -3
Episode 13: Total Reward = 0
Episode 14: Total Reward = -10
Episode 15: Total Reward = 0
Episode 16: Total Reward = -39
Episode 17: Total Reward = 6
Episode 18: Total Reward = 2
Episode 19: Total Reward = 5
Episode 20: Total Reward = 2
Episode 21: Total Reward = 2
Episode 22: Total Reward = 0
Episode 23: Total Reward = -8
Episode 24: Total Reward = -27
Episode 25: Total Reward = -3
Episode 26: Total Reward = 2
Episode 27: Total Reward = -8
Episode 28: Total Reward = 5
Episode 29: Total Reward = 0
Episode 30: Total Reward = 7
Episode 31: Total Reward = -5
Episode 32: Total Reward = 7
Episode 33: Total Reward = 0
Episode 34: Total Reward = -1
Episode 35: Total Reward = -1
Episode 36: Total Reward = -1
Episode 37: Total Reward = -7
Episode 38: Total Reward = 3
Episode 39: Total Reward = 5
Episode 40: Total Reward = -1
Episode 41: Total Reward = -11
Episode 42: Total Reward = -1
Episode 43: Total Reward = 4
Episode 44: Total Reward = -4
Episode 45: Total Reward = -4
Episode 46: Total Reward = 2
Episode 47: Total Reward = 0
Episode 48: Total Reward = 0
Episode 49: Total Reward = -4
Episode 50: Total Reward = 6
Episode 51: Total Reward = 5
Episode 52: Total Reward = 1
Episode 53: Total Reward = 0
Episode 54: Total Reward = 4
Episode 55: Total Reward = 5
Episode 56: Total Reward = 2
Episode 57: Total Reward = 5
Episode 58: Total Reward = -2
Episode 59: Total Reward = 2
Episode 60: Total Reward = 5
Episode 61: Total Reward = 5
Episode 62: Total Reward = 3
Episode 63: Total Reward = 2
Episode 64: Total Reward = 1
Episode 65: Total Reward = 3
Episode 66: Total Reward = 7
Episode 67: Total Reward = 7
Episode 68: Total Reward = 6
Episode 69: Total Reward = 7
Episode 70: Total Reward = 7
Episode 71: Total Reward = 1
Episode 72: Total Reward = 5
Episode 73: Total Reward = 4
Episode 74: Total Reward = 7
Episode 75: Total Reward = -1
Episode 76: Total Reward = 3
Episode 77: Total Reward = 0
Episode 78: Total Reward = 1
Episode 79: Total Reward = 7
Episode 80: Total Reward = 0
Episode 81: Total Reward = 7
Episode 82: Total Reward = 5
Episode 83: Total Reward = 2
Episode 84: Total Reward = 2
Episode 85: Total Reward = 6
Episode 86: Total Reward = 6
Episode 87: Total Reward = 7
Episode 88: Total Reward = 3
Episode 89: Total Reward = 7
Episode 90: Total Reward = 2
Episode 91: Total Reward = 6
Episode 92: Total Reward = 7
Episode 93: Total Reward = 7
Episode 94: Total Reward = 6
Episode 95: Total Reward = 3
Episode 96: Total Reward = 6
Episode 97: Total Reward = 7
Episode 98: Total Reward = 1
Episode 99: Total Reward = 5
Episode 100: Total Reward = 7
Episode 101: Total Reward = 6
Episode 102: Total Reward = 5
Episode 103: Total Reward = 7
Episode 104: Total Reward = 6
Episode 105: Total Reward = 7
Episode 106: Total Reward = 7
Episode 107: Total Reward = 6
Episode 108: Total Reward = 7
Episode 109: Total Reward = 7
Episode 110: Total Reward = 7
Episode 111: Total Reward = 7
Episode 112: Total Reward = 4
Episode 113: Total Reward = 7
Episode 114: Total Reward = 3
Episode 115: Total Reward = 7
Episode 116: Total Reward = 6
Episode 117: Total Reward = 7
Episode 118: Total Reward = 6
Episode 119: Total Reward = 7
Episode 120: Total Reward = 3
Episode 121: Total Reward = 7
Episode 122: Total Reward = 7
Episode 123: Total Reward = 6
Episode 124: Total Reward = 5
Episode 125: Total Reward = 7
Episode 126: Total Reward = 7
Episode 127: Total Reward = 7
Episode 128: Total Reward = 7
Episode 129: Total Reward = 6
Episode 130: Total Reward = 7
Episode 131: Total Reward = 7
Episode 132: Total Reward = 6
Episode 133: Total Reward = 7
Episode 134: Total Reward = 5
Episode 135: Total Reward = 7
Episode 136: Total Reward = 7
Episode 137: Total Reward = 7
Episode 138: Total Reward = 7
Episode 139: Total Reward = 6
Episode 140: Total Reward = 3
Episode 141: Total Reward = 6
Episode 142: Total Reward = 7
Episode 143: Total Reward = 7
Episode 144: Total Reward = 7
Episode 145: Total Reward = 6
Episode 146: Total Reward = 7
Episode 147: Total Reward = 4
Episode 148: Total Reward = 7
Episode 149: Total Reward = 4
Episode 150: Total Reward = 7
Episode 151: Total Reward = 7
Episode 152: Total Reward = 6
Episode 153: Total Reward = 4
Episode 154: Total Reward = 4
Episode 155: Total Reward = 5
Episode 156: Total Reward = 7
Episode 157: Total Reward = 6
Episode 158: Total Reward = 6
Episode 159: Total Reward = 6
Episode 160: Total Reward = 7
Episode 161: Total Reward = 6
Episode 162: Total Reward = 4
Episode 163: Total Reward = 7
Episode 164: Total Reward = 7
Episode 165: Total Reward = 5
Episode 166: Total Reward = 1
Episode 167: Total Reward = 7
Episode 168: Total Reward = 6
Episode 169: Total Reward = 7
Episode 170: Total Reward = 7
Episode 171: Total Reward = 7
Episode 172: Total Reward = 6
Episode 173: Total Reward = 7
Episode 174: Total Reward = 6
Episode 175: Total Reward = 5
Episode 176: Total Reward = 7
Episode 177: Total Reward = 7
Episode 178: Total Reward = 7
Episode 179: Total Reward = 7
Episode 180: Total Reward = 6
Episode 181: Total Reward = 7
Episode 182: Total Reward = 4
Episode 183: Total Reward = 7
Episode 184: Total Reward = 6
Episode 185: Total Reward = 7
Episode 186: Total Reward = 3
Episode 187: Total Reward = 7
Episode 188: Total Reward = 7
Episode 189: Total Reward = 7
Episode 190: Total Reward = 7
Episode 191: Total Reward = 3
Episode 192: Total Reward = 7
Episode 193: Total Reward = 7
Episode 194: Total Reward = 7
Episode 195: Total Reward = 7
Episode 196: Total Reward = 7
Episode 197: Total Reward = 7
Episode 198: Total Reward = 6
Episode 199: Total Reward = 7
Episode 200: Total Reward = 7
Learned Policy:
[['D' 'R' 'D']
 ['D' 'X' 'D']
 ['R' 'R' 'G']]
```

Practical No. 7

Aim: Utilizing transfer learning to improve model performance on limited datasets

Code:

```
import tensorflow as tf
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
# Parameters
IMG_SIZE = 128  # Smaller image size for faster computation
BATCH_SIZE = 16  # Reduced batch size to save memory
EPOCHS = 2       # Fewer epochs for quicker training
LEARNING_RATE = 0.001
# Load and Preprocess MNIST Dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
# Use only a subset of the data (e.g., 10,000 samples for training)
x_train, y_train = x_train[:10000], y_train[:10000]
x_test, y_test = x_test[:2000], y_test[:2000]
# Preprocessing function
def preprocess(image, label):
    image = tf.image.resize(tf.expand_dims(image, axis=-1), (IMG_SIZE, IMG_SIZE)) / 255.0
    image = tf.image.grayscale_to_rgb(image)  # Convert grayscale to RGB
    label = tf.one_hot(label, depth=10)       # One-hot encode labels
    return image, label
# Create TensorFlow datasets
train_dataset = (
    tf.data.Dataset.from_tensor_slices((x_train, y_train))
    .map(preprocess)
    .batch(BATCH_SIZE)
    .prefetch(tf.data.AUTOTUNE)
)
test_dataset = (
    tf.data.Dataset.from_tensor_slices((x_test, y_test))
    .map(preprocess)
    .batch(BATCH_SIZE)
    .prefetch(tf.data.AUTOTUNE)
)
# Load the smaller pre-trained MobileNet model
base_model = MobileNet(weights="imagenet", include_top=False, input_shape=(IMG_SIZE, IMG_SIZE, 3))
# Freeze the base model
base_model.trainable = False
# Add custom layers on top
x = base_model.output
x = GlobalAveragePooling2D()(x)  # Reduce dimensions
x = Dropout(0.3)(x)              # Dropout for regularization
```

```
predictions = Dense(10, activation="softmax")(x)  # Output layer for 10 classes
# Create the full model
model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
model.compile(optimizer=Adam(learning_rate=LEARNING_RATE),
              loss="categorical_crossentropy",
              metrics=["accuracy"])
# Train the model
history = model.fit(
    train_dataset,
    validation_data=test_dataset,
    epochs=EPOCHS
)
# Evaluate the model on the test dataset
evaluation = model.evaluate(test_dataset, verbose=1)
# Print the evaluation metrics
print(f"Test Loss: {evaluation[0]:.4f}")
print(f"Test Accuracy: {evaluation[1]:.4f}")
```

Output:

```
2025-01-05 15:43:00.551839: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU inst
ructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate c
ompiler flags.
Epoch 1/2
625/625 [==============================] - 85s 132ms/step - loss: 0.4097 - accuracy: 0.8680 - val_loss: 0.1673 - val_accuracy: 0.9485
Epoch 2/2
625/625 [==============================] - ETA: 0s - loss: 0.1525 - accuracy: 0.9528
```

Practical No. 8

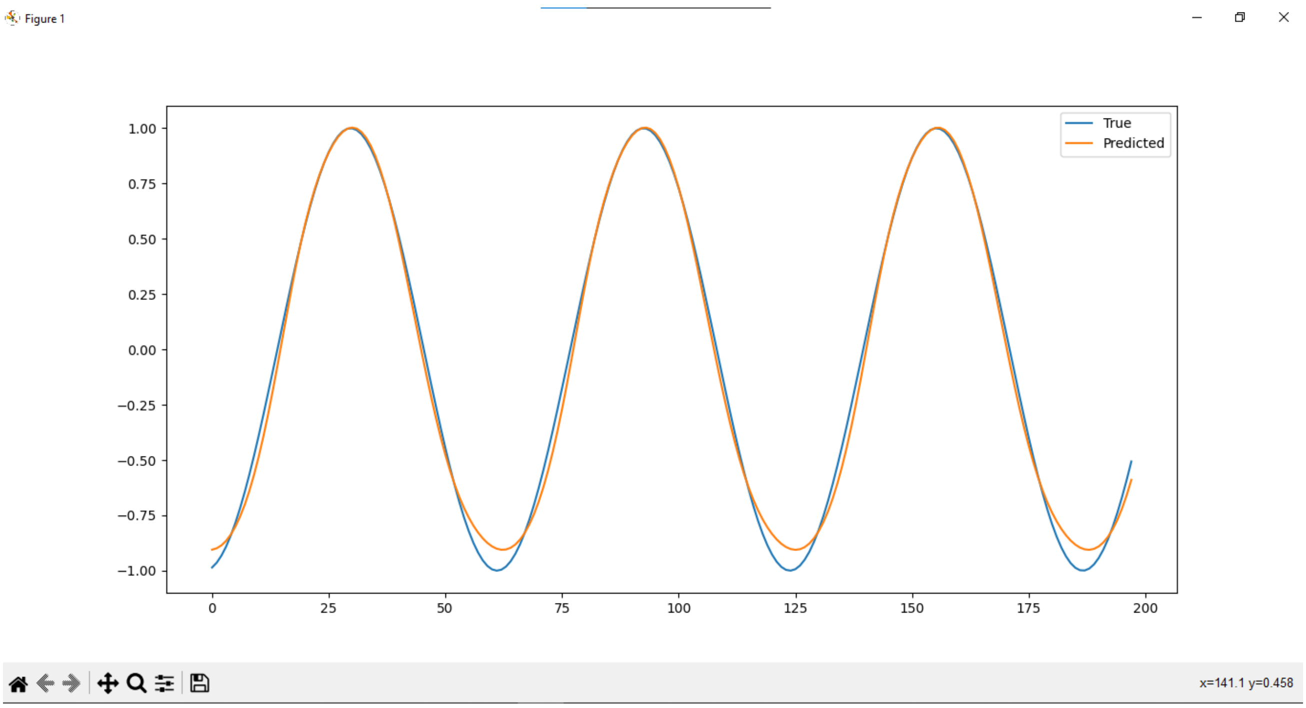
Aim: Building a deep learning model for <u>time series forecasting</u> or anomaly detection.

Code:

```
# time series
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
# 1. Prepare and normalize data
data = pd.DataFrame(np.sin(np.linspace(0, 100, 1000)), columns=['value'])
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)
# 2. Create dataset for LSTM
X, y = [], []
for i in range(len(scaled_data) - 10):
    X.append(scaled_data[i:i+10, 0])
    y.append(scaled_data[i+10, 0])
X, y = np.array(X), np.array(y)
X = X.reshape(X.shape[0], X.shape[1], 1)
# 3. Split data into train and test
train_size = int(len(X) * 0.8)
X_train, X_test, y_train, y_test = X[:train_size], X[train_size:], y[:train_size], y[train_size:]
# 4. Build and train model
model = Sequential([LSTM(50, input_shape=(X_train.shape[1], 1)), Dense(1)])
model.compile(optimizer='adam', loss='mse')
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)
# 5. Predict and plot
predictions = scaler.inverse_transform(model.predict(X_test))
y_test = scaler.inverse_transform(y_test.reshape(-1, 1))
plt.plot(y_test, label='True')
plt.plot(predictions, label='Predicted')
plt.legend()
plt.show()
```

Output:

```
7/7 [==============================] - 1s 5ms/step
```



Practical No. 9

Aim: Implementing a machine learning pipeline for automated feature engineering and model selection.

many changes refer the pract files

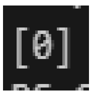
Code:

```
import pickle
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest,chi2
from sklearn.tree import DecisionTreeClassifier
df = pd.read_csv('C:/Users//Desktop/p10/train.csv')
df.drop(columns=['PassengerId','Name','Ticket','Cabin'],inplace=True)
# Step 1 -> train/test/split
X_train,X_test,y_train,y_test = train_test_split(df.drop(columns=['Survived']), df['Survived'], test_size=0.2,
random_state=42)
X_train.head()
y_train.sample(5)
# imputation transformer
trf1 = ColumnTransformer([
    ('impute_age',SimpleImputer(),[2]),
    ('impute_embarked',SimpleImputer(strategy='most_frequent'),[6])
],remainder='passthrough')
# one hot encoding
trf2 = ColumnTransformer([
    ('ohe_sex_embarked',OneHotEncoder(sparse=False,handle_unknown='ignore'),[1,6])
],remainder='passthrough')
# Scaling
trf3 = ColumnTransformer([
    ('scale',MinMaxScaler(),slice(0,10))
])
# Feature selection
trf4 = SelectKBest(score_func=chi2,k=8)
# train the model
trf5 = DecisionTreeClassifier()
pipe = Pipeline([
    ('trf1',trf1),
    ('trf2',trf2),
    ('trf3',trf3),
    ('trf4',trf4),
    ('trf5',trf5)
```

```
])
# train
pipe.fit(X_train,y_train)
pipe.named_steps
# Display Pipeline
from sklearn import set_config
set_config(display='diagram')
# Predict
y_pred = pipe.predict(X_test)
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
# cross validation using cross_val_score
from sklearn.model_selection import cross_val_score
cross_val_score(pipe, X_train, y_train, cv=5, scoring='accuracy').mean()
from sklearn.model_selection import GridSearchCV
# Corrected parameter grid
params = {
        'trf5__max_depth': [1, 2, 3, 4, 5, None]
}
grid = GridSearchCV(pipe, params, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)
grid.best_score_
grid.best_params_
# export
pickle.dump(pipe,open('C:/Users/Desktop/p10/pipe.pkl','wb'))
```

predict.py

```
import pickle
import numpy as np
import pandas as pd
pipe = pickle.load(open('C:/Users/Desktops/pipe.pkl','rb'))
# Assume user input
test_input2 = np.array([2, 'male', 31.0, 0, 0, 10.5, 'S'],dtype=object).reshape(1,7)
#  Adding a new row to the dataframe
# test_input2 = np.vstack([
#     test_input2,
#     np.array([12, 'female', 47.0, 0, 0, 54.3, 'C'], dtype=object).reshape(1, 7),
#     np.array([3, 'male', 23.0, 0, 0, 12.3, 'S'], dtype=object).reshape(1, 7)
# ])
columns = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
test_input2_df = pd.DataFrame(test_input2, columns=columns)
# Assume user input
print(pipe.predict(test_input2_df))
```

Output: 

Remark: [0] or [1] like “This person will survive” or “This person won’t survive” but that would require a bit of change in code.

Practical No. 10

Aim: Using advanced optimization techniques like evolutionary algorithms or Bayesian optimization for hyperparameter tuning.

Code:

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from skopt import BayesSearchCV
from skopt.space import Real, Integer
# Load dataset
data = load_iris()
X, y = data.data, data.target
# Define the model
model = RandomForestClassifier(random_state=42)
# Define the search space for hyperparameters
param_space = {
    'n_estimators': Integer(10, 200),
     # Number of trees
    'max_depth': Integer(1, 20),
    # Maximum depth of a tree
    'min_samples_split': Real(0.01, 0.3),
     # Minimum fraction of samples required to split
    'min_samples_leaf': Integer(1, 10),
    # Minimum samples at a leaf node
    'max_features': Real(0.1, 1.0),
   # Fraction of features to consider for split
}
# Bayesian Optimization with Cross-Validation
opt = BayesSearchCV(
    estimator=model,
    search_spaces=param_space,
    n_iter=50,  # Number of parameter settings to try
    cv=5,       # Number of cross-validation folds
    n_jobs=-1,  # Use all processors
    random_state=42
)
```

```
# Perform the optimization
opt.fit(X, y)
# Results
print("Best Parameters:", opt.best_params_)print("Best Cross-Validation Score:", opt.best_score_)
```

Output:

```
Best Parameters: OrderedDict([('max_depth', 20), ('max_features', 0.6369563862688051), ('min_samples_leaf', 1), ('min_samples_split', 0.12954654430286391), ('n_estimators', 10)])
Best Cross-Validation Score: 0.9666666666666668
PS C:\Users\RPIMS\Desktop\AAI>
```