ARTIFICIAL INTELLIGENCE

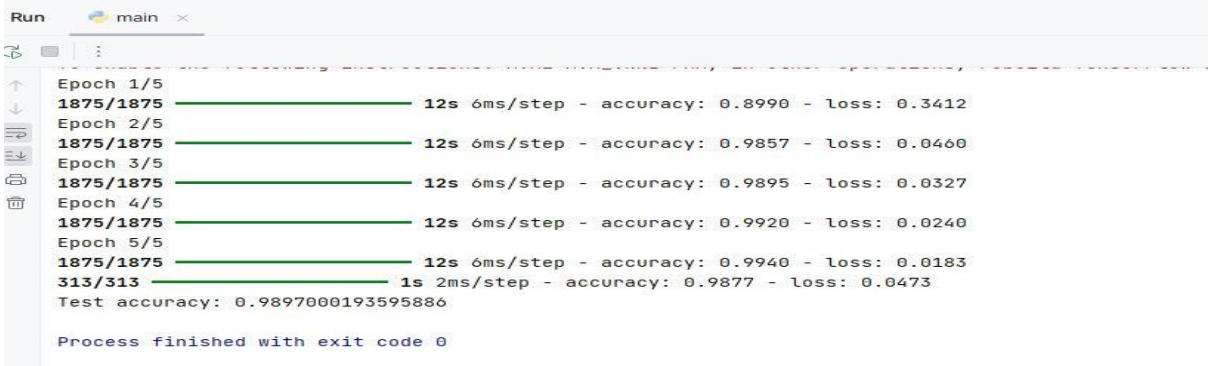
TABLE OF CONTENTS

Practical 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.

```
import tensorflow as tf

from tensorflow.keras import layers, models

(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()

train_images = train_images.reshape((train_images.shape[0], 28, 28, 1))

test_images = test_images.reshape((test_images.shape[0], 28, 28, 1))

train_images, test_images = train_images / 255.0, test_images / 255.0

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),

layers.MaxPooling2D((2, 2)),layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),layers.Conv2D(64, (3, 3), activation='relu'),layers.Flatten(),

layers.Dense(64, activation='relu'), layers.Dense(10, activation='softmax')])

model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy']
) model.fit(train_images, train_labels, epochs=5)

test_loss, test_acc = model.evaluate(test_images, test_labels)

print(f'Test accuracy: {test_acc}')
```

OUTPUT:



Practical 2

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

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, classification_report

data = { 'text': [

    'I love this product, it is amazing!',

    'This is the worst purchase I have ever made.',

    'I am very happy with the service.',

    'I will never buy from this store again.',

    'The quality is excellent!',

    'Terrible experience, very disappointed.',

    'Absolutely fantastic! Highly recommend it.',

    'Not good at all, I want a refund.',

    'Very satisfied with my order.',

    'The item broke after one use, awful.'   ],   'label': ['positive', 'negative', 'positive', 'negative', 'positive', 'negative', 'positive', 'negative', 'positive', 'negative']

}df = pd.DataFrame(data)

X_train, X_test, y_train, y_test = train_test_split( df['text'], df['label'], test_size=0.2, random_state=42, stratify=df['label']

)vectorizer = CountVectorizer()

X_train_vectors = vectorizer.fit_transform(X_train)

X_test_vectors = vectorizer.transform(X_test)

model = MultinomialNB()model.fit(X_train_vectors, y_train)
```

```
y_pred = model.predict(X_test_vectors)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print('Classification Report:')
print(classification_report(y_test, y_pred, zero_division=0))
custom_reviews = [
    "This product exceeded my expectations!",
    "Worst service ever, completely dissatisfied.",
    "The experience was okay, not great, not terrible.",
    "Highly recommend this to everyone!"
]custom_review_vectors = vectorizer.transform(custom_reviews)
custom_predictions = model.predict(custom_review_vectors)
for review, sentiment in zip(custom_reviews, custom_predictions):
 print(f"Review: \"{review}\" -> Sentiment: {sentiment}")
```

OUTPUT:

```
============== RESTART: D:/Janu_mscIT_P2/AAI-practs/(NLP) model.py =============
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

    negative       1.00      1.00      1.00         1
    positive       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2

Review: "This product exceeded my expectations!" -> Sentiment: positive
Review: "Worst service ever, completely dissatisfied." -> Sentiment: negative
Review: "The experience was okay, not great, not terrible." -> Sentiment: negati
ve
Review: "Highly recommend this to everyone!" -> Sentiment: positive
```

Practical 3

Aim- Creating a chatbot using advanced techniques like transformer models.

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer,import torch,import os

os.environ["HF_HOME"] = "./hf_cache"

model_name = "microsoft/DialoGPT-medium"

try:

    model = GPT2LMHeadModel.from_pretrained(model_name)

    tokenizer = GPT2Tokenizer.from_pretrained(model_name)

except Exception as e:

    print(f"Error loading model: {e}")

    exit()

def chat_with_bot(input_text):

    try:

        inputs = tokenizer.encode(input_text + tokenizer.eos_token, return_tensors="pt")

        attention_mask = torch.ones(inputs.shape, dtype=torch.long)outputs = model.generate(
inputs,max_length=100,pad_token_id=tokenizer.eos_token_id,

attention_mask=attention_mask,              no_repeat_ngram_size=2,

  temperature=0.7,              top_p=0.9,  top_k=50, do_sample=True,          )

Decode the response back into text and filter it

        response = tokenizer.decode(outputs[:, inputs.shape[-1]:][0], skip_special_tokens=True)

        return filter_response(response)

    except Exception as e:

        return f"Error during response generation: {e}"

def filter_response(response):

    if len(response.strip()) < 5:  # Adjusted to allow shorter valid responses

        return "Could you say that again? I'm not sure I understood."

elif "?" in response:  # If response is a question, it might be a valid conversational turn
```

```
return response return response

if __name__ == "__main__":

    print("Chatbot: Hello! Type 'exit' to end the conversation.")

  while True:  user_input = input("You: ")

        if user_input.lower() == "exit":

            print("Chatbot: Goodbye!") break

        response = chat_with_bot(user_input)

        print(f"Chatbot: {response}")
```

OUTPUT:

```
Chatbot: Hello! Type 'exit' to end the conversation.
You: Hello,how are you?
Chatbot: Hi, I'm good thanks!
You: i am also fine
Chatbot: okay im in the room now
```

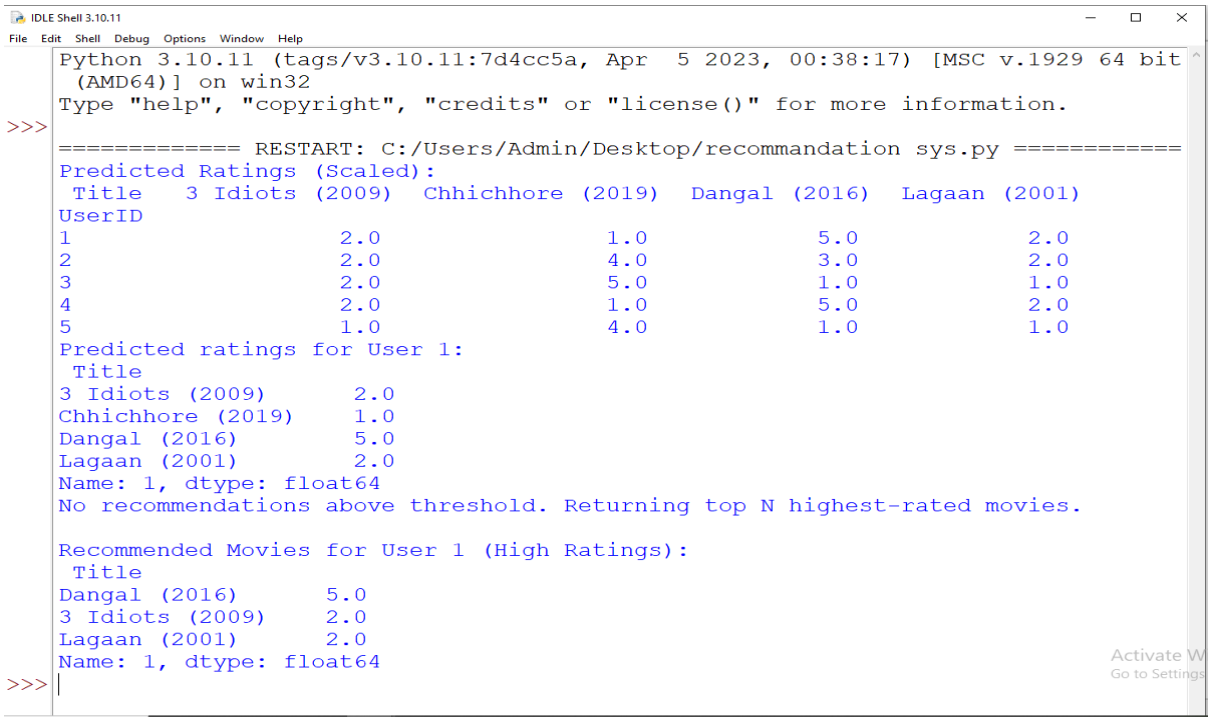
Practical 4

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

```
import pandas as pd

import numpy as np

from sklearn.decomposition import TruncatedSVD

data = {

    "UserID": [1, 1, 2, 2, 3, 3, 4, 4, 5, 5],

    "MovieID": [101, 102, 101, 103, 102, 103, 101, 104, 103, 104],

    "Rating": [5, 4, 3, 4, 2, 5, 5, 3, 4, 2]}movies = {

    "MovieID": [101, 102, 103, 104],

    "Title": ["Dangal (2016)", "3 Idiots (2009)", "Chhichhore (2019)", "Lagaan (2001)"]}

ratings = pd.DataFrame(data)

movies = pd.DataFrame(movies)

ratings = pd.merge(ratings, movies, on="MovieID")

user_item_matrix = ratings.pivot_table(index="UserID", columns="Title",
values="Rating").fillna(0)

svd = TruncatedSVD(n_components=2)  # Use a smaller number of components for
simplicity

user_factors = svd.fit_transform(user_item_matrix)

item_factors = svd.components_

predicted_ratings = np.dot(user_factors, item_factors)

min_rating, max_rating = predicted_ratings.min(), predicted_ratings.max()

predicted_ratings = 1 + 4 * (predicted_ratings - min_rating) / (max_rating - min_rating)

predicted_df = pd.DataFrame(predicted_ratings, index=user_item_matrix.index,
columns=user_item_matrix.columns)

predicted_df = predicted_df.round()

print("Predicted Ratings (Scaled):\n", predicted_df)
```

```
def recommend_movies(user_id, user_item_matrix, predicted_df, top_n=3, threshold=2):
    user_ratings = predicted_df.loc[user_id]
    rated_movies = user_item_matrix.loc[user_id] > 0
    recommendations = user_ratings[~rated_movies]
    print(f"Predicted ratings for User {user_id}:\n", user_ratings)
    recommendations = recommendations[recommendations > threshold].sort_values(ascending=False)
  if recommendations.empty:
        print("No recommendations above threshold. Returning top N highest-rated movies.")
        recommendations = user_ratings.sort_values(ascending=False)
 return recommendations.head(top_n)
recommendations = recommend_movies(1, user_item_matrix, predicted_df, top_n=3, threshold=2)
print("\nRecommended Movies for User 1 (High Ratings):\n", recommendations)
```

OUTPUT:

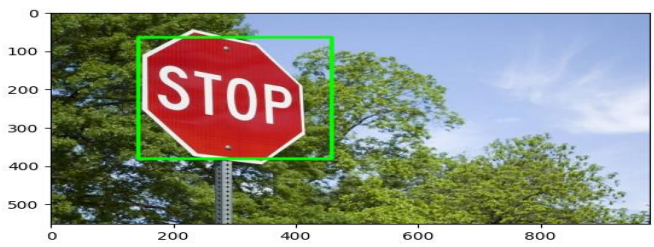


Practical 5

Aim- Implementing a computer vision project, such as object detection or image segmentation.

```
import cv2
from matplotlib import pyplot as plt
image_path = r"C:\Users\Janhavi\Downloads\stop.jpg"
cascade_path = r"C:\Users\Janhavi\Downloads\stop_data.xml"
img = cv2.imread(image_path)
if img is None:
   raise FileNotFoundError("Image file could not be loaded. Check the path.")
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
stop_data = cv2.CascadeClassifier(cascade_path)
if stop_data.empty():
    raise FileNotFoundError("Cascade file could not be loaded. Check the path.")
for (x, y, w, h) in stop_data.detectMultiScale(img_gray, minSize=(20, 20)):
 cv2.rectangle(img_rgb, (x, y), (x + h, y + w), (0, 255, 0), 5)
plt.imshow(img_rgb)
plt.show()
```

OUTPUT:



Practical 6

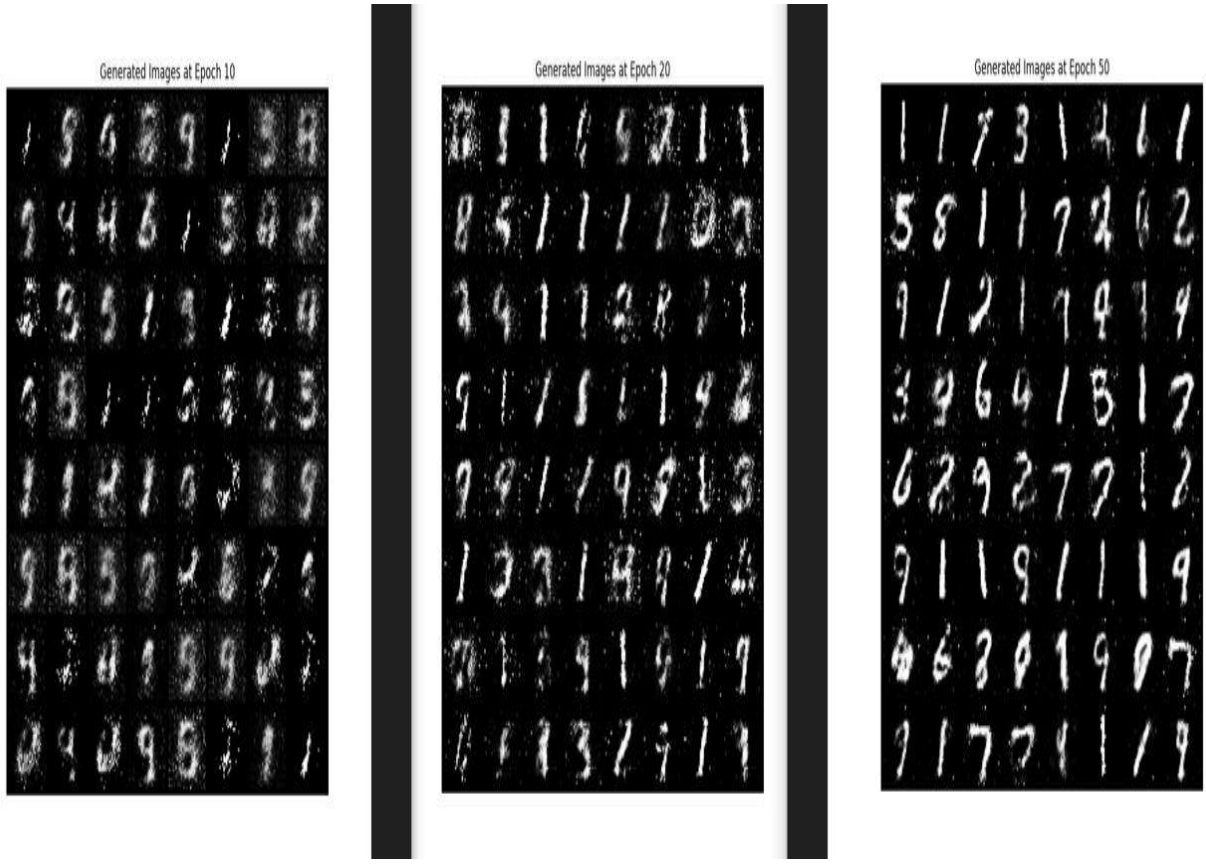
Aim- Training a generative adversarial network (GAN) for generating realistic images.

```
import torch

import torch.nn as nn

import torch.optim as optim

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

import torchvision.utils as vutils

import matplotlib.pyplot as plt

class Generator(nn.Module):

    def __init__(self, input_size, output_size):

        super(Generator, self).__init__() self.model = nn.Sequential(nn.Linear(input_size, 128),

            nn.ReLU(),nn.Linear(128, 256),nn.BatchNorm1d(256), nn.ReLU(),nn.Linear(256,
512), nn.BatchNorm1d(512), nn.ReLU(),nn.Linear(512, output_size), nn.Tanh()        )

    def forward(self, x):

        return self.model(x)

class Discriminator(nn.Module):

    def __init__(self, input_size):

        super(Discriminator, self).__init__() self.model = nn.Sequential( nn.Linear(input_size,
512), nn.LeakyReLU(0.2), nn.Linear(512, 256), nn.LeakyReLU(0.2), nn.Linear(256, 1),
nn.Sigmoid()                    )

 def forward(self, x):

  return self.model(x)latent_size = 100 image_size = 28 * 28  batch_size = 64

epochs = 50 lr = 0.0002

transform = transforms.Compose([

    transforms.ToTensor(),

    transforms.Normalize(mean=[0.5], std=[0.5])])

dataset = datasets.MNIST(root="./data", train=True, transform=transform, download=True)
```

```
data_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

generator = Generator(input_size=latent_size, output_size=image_size)

discriminator = Discriminator(input_size=image_size)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

generator.to(device) discriminator.to(device)

criterion = nn.BCELoss()

optimizer_g = optim.Adam(generator.parameters(), lr=lr)

optimizer_d = optim.Adam(discriminator.parameters(), lr=lr)

for epoch in range(epochs):

    for batch_idx, (real_images, _) in enumerate(data_loader):

        real_images = real_images.view(-1, image_size).to(device)  batch_size =
real_images.size(0) real_labels = torch.ones(batch_size, 1).to(device)fake_labels =
torch.zeros(batch_size, 1).to(device)outputs = discriminator(real_images)d_loss_real =
criterion(outputs, real_labels)z = torch.randn(batch_size, latent_size).to(device)fake_images
= generator(z)outputs = discriminator(fake_images.detach())d_loss_fake = criterion(outputs,
fake_labels)d_loss = d_loss_real + d_loss_fake
optimizer_d.zero_grad()d_loss.backward()optimizer_d.step()

  z = torch.randn(batch_size, latent_size).to(device)fake_images = generator(z)

  outputs = discriminator(fake_images) g_loss = criterion(outputs, real_labels)

optimizer_g.zero_grad()g_loss.backward()optimizer_g.step()

print(f"Epoch [{epoch + 1}/{epochs}]  D Loss: {d_loss.item():.4f}  G Loss:
{g_loss.item():.4f}")

    if (epoch + 1) % 10 == 0:

        with torch.no_grad():

            z = torch.randn(64, latent_size).to(device)

            fake_images = generator(z).view(-1, 1, 28, 28).cpu()

            print(f"Epoch {epoch + 1}: Generated Images Mean Pixel Value =
{fake_images.mean().item():.4f}")

 grid = vutils.make_grid(fake_images, nrow=8, normalize=True)

            plt.figure(figsize=(8, 8))
```

```
        plt.imshow(grid.permute(1, 2, 0).numpy(), cmap="gray")
        plt.title(f"Generated Images at Epoch {epoch + 1}")
        plt.axis("off")
        filename = f"generated_images_epoch_{epoch + 1}.png"
        plt.savefig(filename)
        plt.close()
print(f"Images saved to {filename}")
```

OUTPUT:



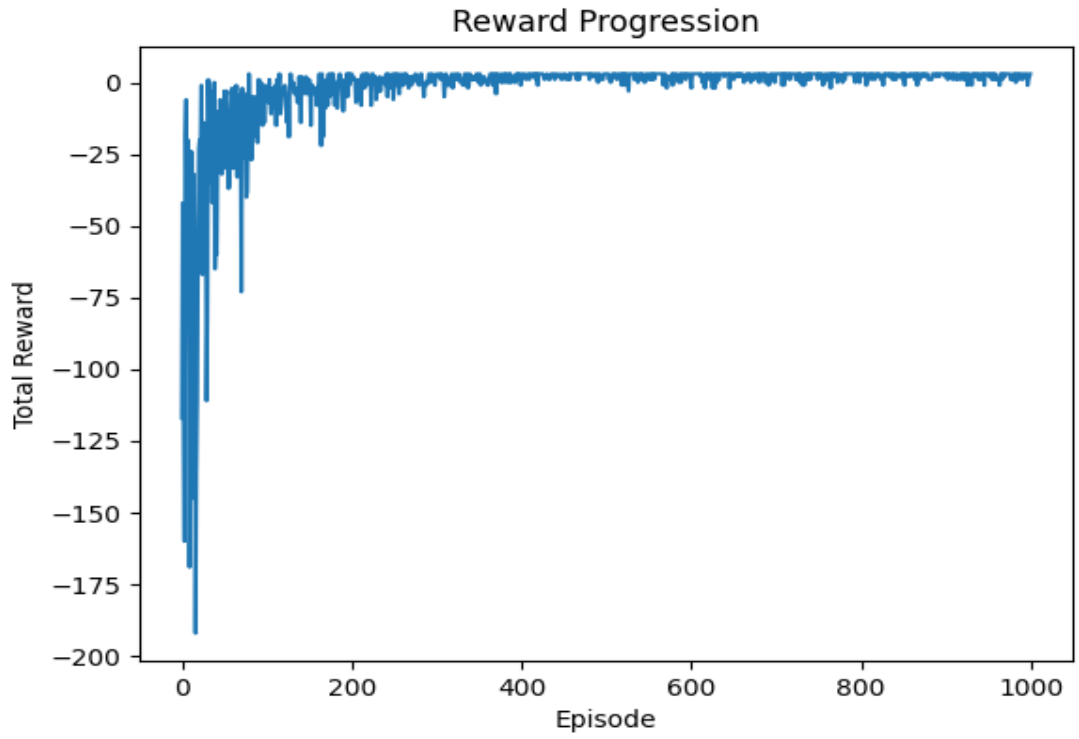
Practical 7

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

```
import numpy as np
import matplotlib.pyplot as plt
class GridWorld:
    def __init__(self, size, start, goal, obstacles):
        self.size = size self.start = start self.goal = goal self.obstacles = obstacles self.reset()
   def reset(self):
        self.state = self.start  return self.state def step(self, action):
        x, y = self.state
        if action == 0:  # Up
            next_state = (max(x - 1, 0), y)
        elif action == 1:  # Down
            next_state = (min(x + 1, self.size - 1), y)
        elif action == 2:  # Left
            next_state = (x, max(y - 1, 0))
        elif action == 3:  # Right
            next_state = (x, min(y + 1, self.size - 1))
        if next_state in self.obstacles:
            next_state = self.state  reward = -1  done = False
 if next_state == self.goal:
            reward = 10   done = True
self.state = next_state
        return next_state, reward, done
start = (0, 0)  goal = (4, 4)  obstacles = [(2, 2), (3, 2)]
env = GridWorld(size, start, goal, obstacles)
```

```
actions = 4  q_table = np.zeros((size, size, actions))  episodes = 1000 alpha = 0.1

gamma = 0.9   epsilon = 1.0   epsilon_decay = 0.995 min_epsilon = 0.1

rewards = []

for episode in range(episodes): state = env.reset()      total_reward = 0 while not done:

      x, y = state if np.random.uniform(0, 1) < epsilon: action = np.random.choice(actions)  # Explore          else: action = np.argmax(q_table[x, y])

 next_state, reward, done = env.step(action)  next_x, next_y = next_state

plt.plot(rewards)  plt.xlabel('Episode')  plt.ylabel('Total Reward')

plt.title('Reward Progression') plt.show()

print("Learned Q-values:")

for i in range(size):

   for j in range(size):

      print(f"({i}, {j}): {q_table[i, j]}")
```

OUTPUT:



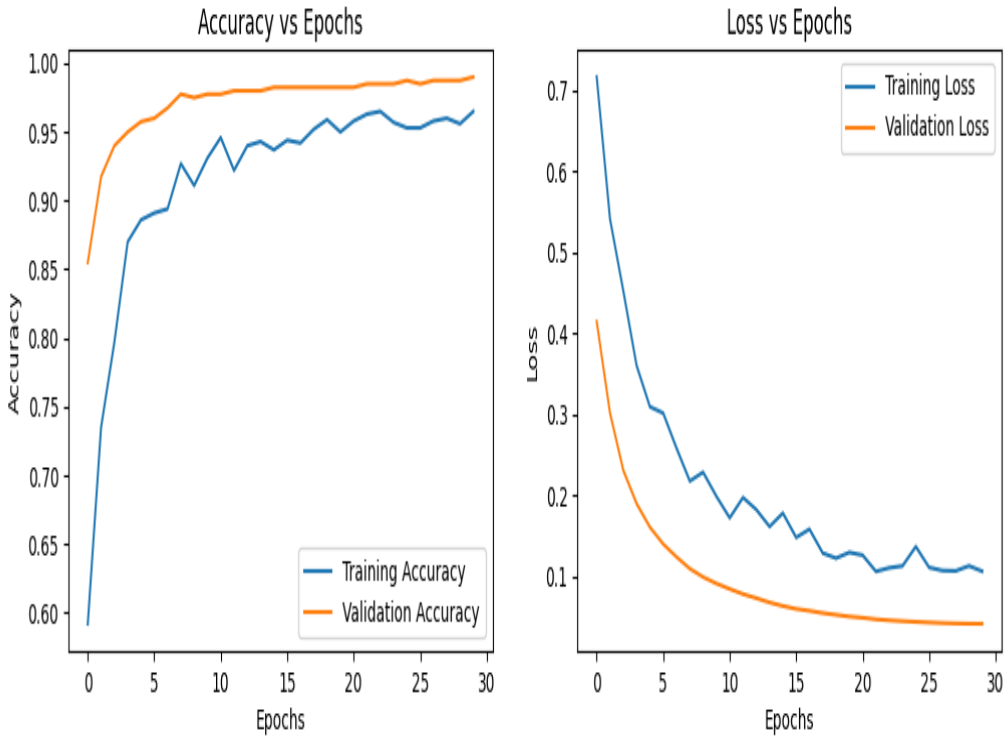
Practical 8

Aim- Utilizing transfer learning to improve model performance on limited datasets.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from PIL import Image
import matplotlib.pyplot as plt
train_dir = r'C:\Users\Janhavi\Downloads\cats_dogs_light\train'
validation_dir = r'C:\Users\Janhavi\Downloads\cats_dogs_light\test'
print(f"Train directory exists: {os.path.isdir(train_dir)}")
print(f"Validation directory exists: {os.path.isdir(validation_dir)}")
print(f"Found {len(os.listdir(train_dir))} subdirectories in the train directory.")
print(f"Found {len(os.listdir(validation_dir))} subdirectories in the validation directory.")
def check_images(directory):
    print(f"Checking files in {directory}...")
    for subdir in os.listdir(directory):
        subdir_path = os.path.join(directory, subdir)
        if os.path.isdir(subdir_path):
            for file_name in os.listdir(subdir_path):
                file_path = os.path.join(subdir_path, file_name)
for layer in base_model.layers[:-10]:  # Unfreeze all but the last 10 layers
    layer.trainable = False model = Sequential([  base_model,
GlobalAveragePooling2D(),Dense(64, activation='relu'),  # Reduced neurons for simplicity
    Dropout(0.5),  # Dropout to prevent overfitting
    Dense(1, activation='sigmoid')  # Binary classification])
model.compile(
```

```
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),  # Use lower learning rate
    loss='binary_crossentropy',    metrics=['accuracy'] )
train_generator = train_datagen.flow_from_directory(train_dir, target_size=(224, 224),
batch_size=32, class_mode='binary'  # Binary classification (cats vs dogs)history = model.fit(
    train_generator,    validation_data=validation_generator, epochs=30,
callbacks=[early_stopping, lr_scheduler]plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1) plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy') plt.ylabel('Accuracy')
plt.legend()plt.title('Accuracy vs Epochs') plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss') plt.xlabel('Epochs')
plt.ylabel('Loss') plt.legend()
plt.title('Loss vs Epochs')
plt.show()
```

OUTPUT:



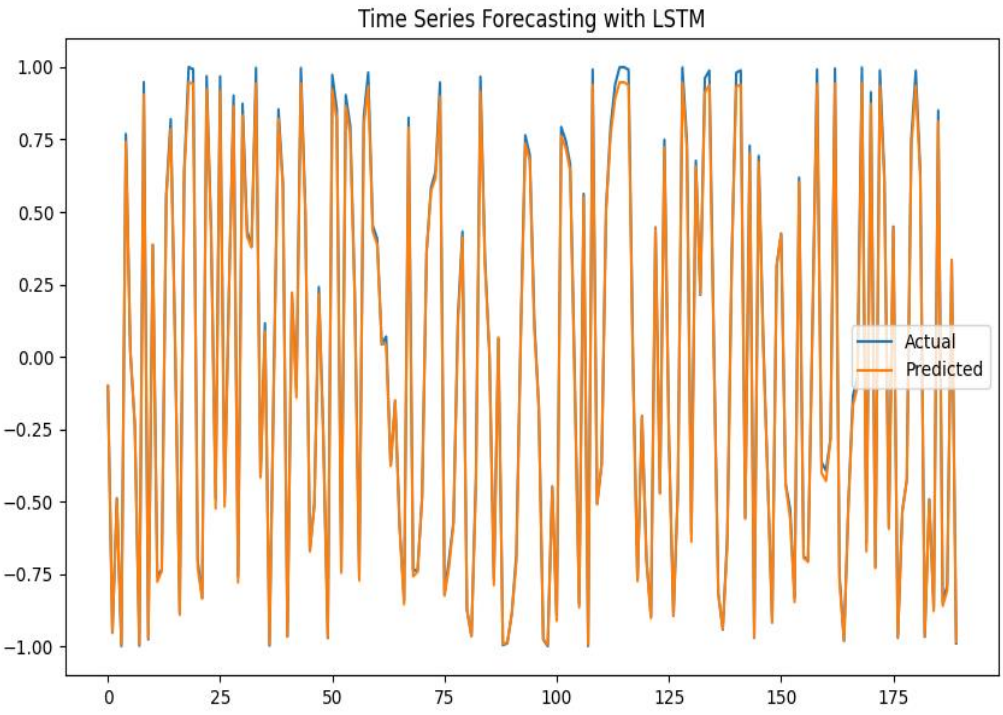
Practical 9

Aim- Building a deep learning model for time series forecasting or anomaly detection.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split

def generate_sine_wave(length):

    x = np.linspace(0, 50, length)  y = np.sin(x) return y

def create_dataset(data, look_back):

    X, y = [], [] for i in range(len(data) - look_back):X.append(data[i:i + look_back])
y.append(data[i + look_back]) return np.array(X), np.array(y)

TIME_STEPS = 50  # Look-back window size EPOCHS = 20 BATCH_SIZE = 32

data = generate_sine_wave(1000) scaler = MinMaxScaler(feature_range=(0, 1))

scaled_data = scaler.fit_transform(data.reshape(-1, 1)).flatten()

X, y = create_dataset(scaled_data, TIME_STEPS)

X = np.expand_dims(X, axis=-1)  # Add a channel dimension for LSTM

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = Sequential([LSTM(50, return_sequences=True, input_shape=(TIME_STEPS, 1)),

Dropout(0.2), LSTM(50), Dropout(0.2), Dense(1)])model.compile(optimizer='adam',
loss='mean_squared_error')

history = model.fit(X_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE,
validation_data=(X_test, y_test))

train_loss = model.evaluate(X_train, y_train, verbose=0)

test_loss = model.evaluate(X_test, y_test, verbose=0)

print(f"Train Loss: {train_loss:.4f}, Test Loss: {test_loss:.4f}")
```

```
predicted = model.predict(X_test)

predicted = scaler.inverse_transform(predicted)

y_test_actual = scaler.inverse_transform(y_test.reshape(-1, 1))

plt.figure(figsize=(10, 6))

plt.plot(y_test_actual, label='Actual')

plt.plot(predicted, label='Predicted')

plt.legend()

plt.title("Time Series Forecasting with LSTM")

plt.show()
```

OUTPUT:



Practical 10

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

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
import lightgbm as lgb
import matplotlib.pyplot as plt
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
data = pd.read_csv(url)
data = data.drop(columns=['Name', 'Ticket', 'Cabin'])  # Drop irrelevant columns
data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})  # Encode 'Sex' feature
data = pd.get_dummies(data, drop_first=True)  # One-hot encoding for 'Embarked'
X = data.drop(columns='Survived')
y = data['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # Handle missing values
    ('scaler', StandardScaler())  # Feature scaling])
categorical_features = X.select_dtypes(include=['object']).columns
    ('encoder', 'passthrough')  # Use 'passthrough' to handle categorical features in pd.get_dummies outside the pipeline
])preprocessor = ColumnTransformer( transformers=[('num', numeric_transformer, numeric_features), ('cat', categorical_transformer, categorical_features)])
```

```
models = [
    ('Logistic Regression', LogisticRegression(max_iter=1000)),
    ('Random Forest', RandomForestClassifier()),
    ('SVM', SVC()),    ('XGBoost', xgb.XGBClassifier()),    ('LightGBM',
lgb.LGBMClassifier())]
    cv_score = cross_val_score(pipeline, X_train, y_train, cv=5, scoring='accuracy')
    mean_score = np.mean(cv_score)
if hasattr(best_pipeline.named_steps['classifier'], 'feature_importances_'):
    importances = best_pipeline.named_steps['classifier'].feature_importances_
    indices = np.argsort(importances)[::-1]
    plt.figure(figsize=(10, 6))
    plt.title(f"Feature Importances for {best_model}")
    plt.bar(range(X_train.shape[1]), importances[indices], align='center')
    plt.xticks(range(X_train.shape[1]), X_train.columns[indices], rotation=90)
    plt.show()
```

OUTPUT:

```
Training Logistic Regression...
Logistic Regression Mean Cross-Validation Score: 0.7913
Training Random Forest...
Random Forest Mean Cross-Validation Score: 0.8058
Training SVM...
SVM Mean Cross-Validation Score: 0.8170
Training XGBoost...
XGBoost Mean Cross-Validation Score: 0.7978
Training LightGBM...
[LightGBM] [Info] Number of positive: 184, number of negative: 314
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000175 seconds.
```

Practical 11

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

```
import os
os.environ["OPTUNA_NO_COLOR"] = "1"  # Disable colored output
import optuna
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
data = load_iris()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
def objective(trial):
    # Define hyperparameter search space
    n_estimators = trial.suggest_int("n_estimators", 10, 200)
    max_depth = trial.suggest_int("max_depth", 2, 20)
    min_samples_split = trial.suggest_int("min_samples_split", 2, 20)
    min_samples_leaf = trial.suggest_int("min_samples_leaf", 1, 10)
    clf = RandomForestClassifier(
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,
        random_state=42, )
    score = cross_val_score(clf, X_train, y_train, cv=5, scoring="accuracy").mean()
    return score
```

```
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=50)
print("Best hyperparameters:", study.best_params)
best_params = study.best_params
best_model = RandomForestClassifier(**best_params, random_state=42)
best_model.fit(X_train, y_train)
accuracy = best_model.score(X_test, y_test)
print(f"Accuracy with best hyperparameters: {accuracy:.2f}")
```

OUTPUT:

```
meters: {'n_estimators': 154, 'max_depth': 6, 'min_samples_split': 5, 'min_sampl
es_leaf': 3}. Best is trial 1 with value: 0.9583333333333334.[0m
[32m[I 2024-11-22 19:24:57,858][0m Trial 49 finished with value: 0.94166666666
66667 and parameters: {'n_estimators': 144, 'max_depth': 9, 'min_samples_split':
 6, 'min_samples_leaf': 4}. Best is trial 1 with value: 0.9583333333333334.[0m
Best hyperparameters: {'n_estimators': 92, 'max_depth': 20, 'min_samples_split':
 15, 'min_samples_leaf': 3}
Accuracy with best hyperparameters: 1.00
>>>
```

Practical 12

Aim- Use Python libraries such as GPT-2 or textgenrnn to train generative models on a corpus of text data and generate new text based on the patterns it has learned.

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer

model_name = "gpt2"  # You can change to a larger version like 'gpt2-medium' if needed

tokenizer = GPT2Tokenizer.from_pretrained(model_name)

model = GPT2LMHeadModel.from_pretrained(model_name)

tokenizer.pad_token = tokenizer.eos_token

prompt = "Once upon a time in a faraway land"

inputs = tokenizer.encode(prompt, return_tensors="pt")

outputs = model.generate(inputs, max_length=50,

    num_return_sequences=5, do_sample=True,    top_k=50,  top_p=0.95,

     attention_mask=None,   pad_token_id=tokenizer.eos_token_id)  # Set pad_token_id to eos_token_id

print("\nGenerated Texts:") for i, output in enumerate(outputs, start=1): print(f"{i}: {tokenizer.decode(output, skip_special_tokens=True)}")
```

OUTPUT:

```
Generated Texts:
1: Once upon a time in a faraway land, when in my travels the wind and fire are
both so keen they break down to pieces, yet in a vast place a stone is broken, t
hough it cannot break. When in a faraway land,
2: Once upon a time in a faraway land there is a mighty earthquake, and all that
 can be seen is that there is an enormous and massive earth that is completely c
overed with molten lava, and there is still a great amount of energy left over,
3: Once upon a time in a faraway land in the South, he stood on a ledge on which
 his body was resting. He saw in its depths a woman with no name or name of any
kind with his wife. And he took her out to
4: Once upon a time in a faraway land, a spirit spoke to the inhabitants and tol
d them to stop. The inhabitants thought they had been tricked by the spirit and
were prepared to fight it for their lives with the hope of having the spirit out
 of
5: Once upon a time in a faraway land, there existed a civilization of humans an
d elves. One of their basic needs was to produce food, so that humans could get
by without relying on agriculture for their livelihood. Many of the human races
of the
>>>
```