Practical Journal

ADVANCED ARTIFICIAL INTELLIGENCE

A Practical Report

Submitted in partial fulfillment of the

Requirements for the award of the

Degree

MASTER OF SCIENCE (INFORMATION TECHNOLOGY)

Part 2 – SEM III

Submitted by

Mr. Awan Imran maknojia

Seat No: 1313174

DEPARTMENT OF INFORMATION TECHNOLOGY

VALIA C.L COLLEGE OF COMMERCE & VALIA L.C COLLEGE OF ARTS CES ROAD D.N NAGAR

(Affiliated to University of Mumbai)

MUMBAI, 400053

MAHARASHTRA

2024-2025

VALIA C.L COLLEGE OF COMMERCE & VALIA L.C COLLEGE OF ARTS CES ROAD D.N NAGAR

(Affiliated to University of

Mumbai) Mumbai-Maharashtra-

400053

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the practical report of **ADVANCED ARTIFICIAL INTELLIGENCE** is bonafide work of **Awan Imran maknojia** bearing Seat No: **1313179** submitted in partial fulfilment of the requirements for the award of degree of MASTER OF SCIENCE in INFORMATION TECHNOLOGY from University of Mumbai.

Internal Guide Coordinator

External Examiner

Date: College Seal

INDEX

Practical 1

AIM:-

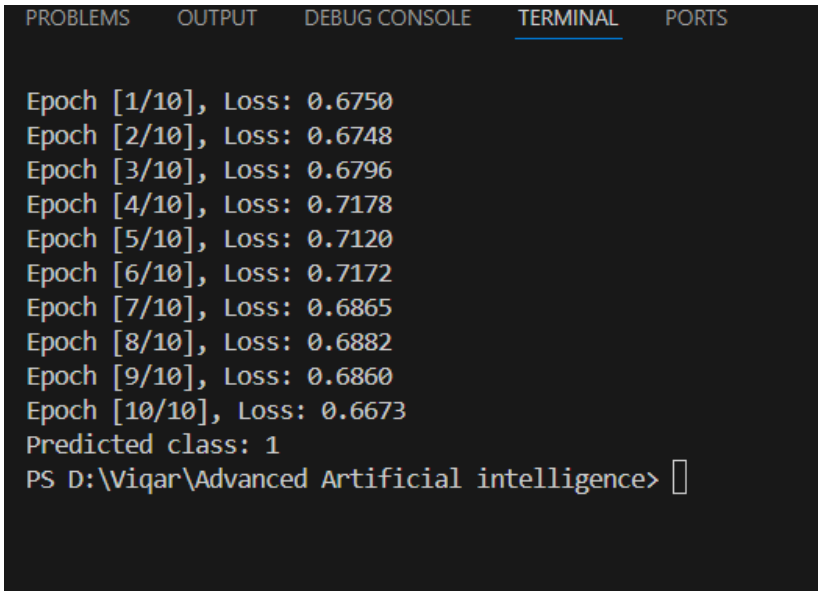
Implementing advanced deep learning algorithms such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) using Python libraries like TensorFlow or PyTorch.

CODE:-

```
import torch
import torch.nn as nn
import torch.optim as
optim
from torch.utils.data import DataLoader,
Dataset import numpy as np

class SequenceDataset(Dataset):
    def __init__(self, num_sequences=1000, sequence_length=10,
        num_classes=2): self.num_sequences = num_sequences
        self.sequence_length = sequence_length
        self.num_classes = num_classes
        self.data = np.random.randn(num_sequences, sequence_length, 1)
        self.labels = np.random.randint(0, num_classes, num_sequences)

    def __len__(self):
        return self.num_sequences

    def __getitem__(self, idx):
        sequence = self.data[idx]
        label = self.labels[idx]
        return torch.tensor(sequence,
dtype=torch.float32), torch.tensor(label,
dtype=torch.long)

class LSTMClassifier(nn.Module):
    def __init__(self, input_size=1, hidden_size=64, num_layers=2,
num_classes=2):
        super(LSTMClassifier, self). init ()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0),
self.hidden_size).to(x.device)
        c0 = torch.zeros(self.num_layers, x.size(0),
```

```
        out = out[:, -1,
        :] out =
        self.fc(out)
        return out

input_size = 1
hidden_size = 64
num_layers = 2
num_classes = 2
num_epochs = 10
batch_size = 32
learning_rate = 0.001

dataset = SequenceDataset()
train_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

device = torch.device("cuda" if torch.cuda.is_available() else
"cpu") model = LSTMClassifier(input_size, hidden_size, num_layers,
num_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    for sequences, labels in train_loader:
        sequences, labels = sequences.to(device), labels.to(device)
        outputs = model(sequences)
        loss = criterion(outputs,
        labels) optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

with torch.no_grad():
    test_sequence = torch.randn(1, dataset.sequence_length,
input_size).to(device)
    prediction = model(test_sequence)
    predicted_class = torch.argmax(prediction,
    dim=1).item() print(f'Predicted class:
```

OUTPUT:-


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Epoch [1/10], Loss: 0.6750
Epoch [2/10], Loss: 0.6748
Epoch [3/10], Loss: 0.6796
Epoch [4/10], Loss: 0.7178
Epoch [5/10], Loss: 0.7120
Epoch [6/10], Loss: 0.7172
Epoch [7/10], Loss: 0.6865
Epoch [8/10], Loss: 0.6882
Epoch [9/10], Loss: 0.6860
Epoch [10/10], Loss: 0.6673
Predicted class: 1
PS D:\Viqar\Advanced Artificial intelligence>

Practical 2

AIM:-

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

PREREQUISITES:-

pip install scikit-learn pandas
pip install scikit-learn pandas datasets

CODE:-

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer from
sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score from datasets
import load_dataset

dataset = load_dataset("imdb") df =
pd.DataFrame({
        'text': dataset['train']['text'],
        'label': dataset['train']['label']
})

X_train, X_test, y_train, y_test = train_test_split(df['text'], df['label'], test_size=0.2, random_state=42)
vectorizer = TfidfVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train) model =
LogisticRegression(max_iter=1000) model.fit(X_train_vectorized,
y_train) X_test_vectorized = vectorizer.transform(X_test) predictions =
model.predict(X_test_vectorized) accuracy = accuracy_score(y_test,
predictions) print(f'Accuracy: {accuracy * 100:.2f}%')
new_text = ["I did not like this at all."] new_vectorized =
vectorizer.transform(new_text) prediction =
model.predict(new_vectorized)
print(f'Prediction for "{new_text[0]}": {"positive" if prediction[0] == 1 else "negative"}')
```

OUTPUT:-

```
vanced Artificial Intelligence\Practical 2.py
Accuracy: 88.44%
Prediction for "I did not like this at all.": negative
```

Practical 3

AIM:-

Creating a chatbot using advanced techniques like transformer models.

PREREQUISITE:-
pip install transformers torch

CODE:-

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("microsoft/DialoGPT-
medium") model =
AutoModelForCausalLM.from_pretrained("microsoft/DialoGPT-medium")

def chat():
    chat_history_ids = None
    print("Chatbot: Hi! I'm a chatbot. Type 'quit' to
    exit.") while True:
        user_input = input("User: ")
        if user_input.lower() ==
            'quit': break
        new_input_ids = tokenizer.encode(user_input +
tokenizer.eos_token, return_tensors='pt')

        # Create attention mask
        if chat_history_ids is None:
            bot_input_ids = new_input_ids
            attention_mask = torch.ones(bot_input_ids.shape,
        dtype=torch.int) else:
            bot_input_ids = torch.cat([chat_history_ids, new_input_ids],
            dim=-
1)
            attention_mask = torch.ones(bot_input_ids.shape,
            dtype=torch.int)

        chat_history_ids = model.generate(bot_input_ids,
                                          max_length=1000,
                                          pad_token_id=tokenizer.eos_token_
                                          id
,
                                          temperature=0.9
                                          ,
                                          do_sample=True,
                                          top_k=50,
                                          top_p=0.95,
                                          attention_mask=attention_mask)
```

```
chat()
```

OUTPUT:-


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Chatbot: Hi! I'm a chatbot. Type 'quit' to exit.
User: Hii
Chatbot: Hey there!
User: how are you
Chatbot: Good, yourself?
User: fine
Chatbot: that's nice.
User: thank you
Chatbot: I'm glad.
User: quit

Practical 4

AIM:

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

PREQUISITE:-

pip install tensorflow numpy pandas

CODE:-

```
import pandas as pd

# https://files.grouplens.org/datasets/movielens/ml-
25m.zip movies = pd.read_csv("movies.csv")

movies.head()

import re

def clean_title(title):
    title = re.sub("[^a-zA-Z0-9 ]", "",
    title) return title

movies["clean_title"] =

movies["title"].apply(clean_title) movies

from sklearn.feature_extraction.text import
TfidfVectorizer vectorizer =
TfidfVectorizer(ngram_range=(1,2))

tfidf = vectorizer.fit_transform(movies["clean_title"])

from sklearn.metrics.pairwise import
cosine_similarity import numpy as np

def search(title):
    title = clean_title(title)
    query_vec = vectorizer.transform([title])
    similarity = cosine_similarity(query_vec, tfidf).flatten()
    indices = np.argpartition(similarity, -5)[-5:]
    results = movies.iloc[indices].iloc[::-

    1] return results

# pip install ipywidgets
#jupyter labextension install @jupyter-widgets/jupyterlab-manager

import ipywidgets as widgets
from IPython.display import display
```

```
movie_input = widgets.Text( value='Toy Story',
        description='Movie Title:', disabled=False
)
movie_list = widgets.Output()

def on_type(data): with
        movie_list:
                movie_list.clear_output() title =
                data["new"]
                if len(title) > 5: display(search(title))

movie_input.observe(on_type, names='value')
```

```
display(movie_input, movie_list)

movie_id = 89745

#def find_similar_movies(movie_id):
movie = movies[movies["movieId"] ==

movie_id] ratings = pd.read_csv("ratings.csv")

ratings.dtypes

similar_users = ratings[(ratings["movieId"] == movie_id) & (ratings["rating"]
> 4)]["userId"].unique()

similar_user_recs = ratings[(ratings["userId"].isin(similar_users)) & (rat- ings["rating"] > 4)]["movieId"]

similar_user_recs = similar_user_recs.value_counts() / len(similar_users)

similar_user_recs = similar_user_recs[similar_user_recs > .10]

all_users = ratings[(ratings["movieId"].isin(similar_user_recs.index)) & (rat- ings["rating"] > 4)]

all_user_recs = all_users["movieId"].value_counts() / len(all_us- ers["userId"].unique())

rec_percentages = pd.concat([similar_user_recs, all_user_recs], axis=1) rec_percentages.columns =
["similar", "all"]

rec_percentages
```

```
rec_percentages["score"] = rec_percentages["similar"] / rec_percentages["all"]

rec_percentages = rec_percentages.sort_values("score", ascending=False)

rec_percentages.head(10).merge(movies, left_index=True, right_on="movieId")

def find_similar_movies(movie_id):
    similar_users = ratings[(ratings["movieId"] == movie_id) & (ratings["rat- ing"] > 4)]
    ["userId"].unique() similar_user_recs = ratings[(ratings["userId"].isin(similar_users)) &
    (ratings["rating"] > 4)]["movieId"] similar_user_recs = similar_user_recs.value_counts() /
    len(similar_users)

    similar_user_recs = similar_user_recs[similar_user_recs > .10] all_users =
    ratings[(ratings["movieId"].isin(similar_user_recs.index)) &
(ratings["rating"] > 4)]
    all_user_recs = all_users["movieId"].value_counts() / len(all_us- ers["userId"].unique())
    rec_percentages = pd.concat([similar_user_recs, all_user_recs], axis=1) rec_percentages.columns =
    ["similar", "all"]

    rec_percentages["score"] = rec_percentages["similar"] / rec_percent-
    ages["all"] rec_percentages = rec_percentages.sort_values("score",
    ascending=False) return rec_percentages.head(10).merge(movies,
    left_index=True,
right_on="movieId")[["score", "title", "genres"]]

import ipywidgets as widgets
from IPython.display import display

movie_name_input = widgets.Text( value='Toy
    Story', description='Movie Title:',
    disabled=False
)
recommendation_list = widgets.Output()

def on_type(data):
    with recommendation_list:
        recommendation_list.clear_output() title =
        data["new"]
        if len(title) > 5:
            results = search(title)
            movie_id = results.iloc[0]["movieId"]
            display(find_similar_movies(movie_id)
            )

movie_name_input.observe(on_type,  names='value')
```

```
display(movie_name_input, recommendation_list)
```

OUTPUT:-

Movie Title: Toy Story

	score	title	genres
3021	18.841924	Toy Story 2 (1999)	Adventure|Animation|Children|Comedy|Fantasy
2264	8.210086	Bug's Life, A (1998)	Adventure|Animation|Children|Comedy
2669	6.868954	Iron Giant, The (1999)	Adventure|Animation|Children|Drama|Sci-Fi
14813	6.503216	Toy Story 3 (2010)	Adventure|Animation|Children|Comedy|Fantasy|IMAX
3650	6.272875	Chicken Run (2000)	Animation|Children|Comedy
1992	5.531892	Little Mermaid, The (1989)	Animation|Children|Comedy|Musical|Romance
1818	5.362941	Mulan (1998)	Adventure|Animation|Children|Comedy|Drama|Musi...
2895	5.349396	Who Framed Roger Rabbit? (1988)	Adventure|Animation|Children|Comedy|Crime|Fant...
0	5.287943	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
3082	5.283613	Galaxy Quest (1999)	Adventure|Comedy|Sci-Fi

Practical 5

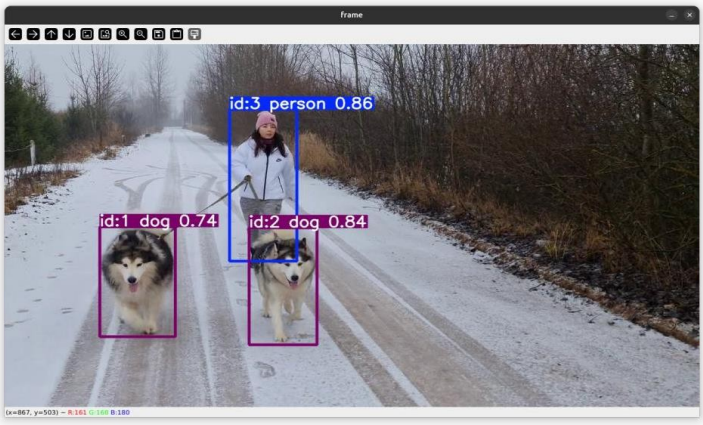
AIM:
-

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

PREQUISITE:-
pip install tensorflow numpy pandas
CODE:-

```
from ultralytics import YOLO

import cv2


# load yolov8 model
model = YOLO('yolov8n.pt')

# load video
video_path = './test.mp4'
cap = cv2.VideoCapture(video_path)

ret = True
# read frames
while ret:
ret, frame = cap.read()

if ret:

# detect
objects # track
objects
results = model.track(frame, persist=True)

# plot results
#
cv2.rectangle
# cv2.putText
frame_ = results[0].plot()

# visualize
cv2.imshow('frame', frame_)
if cv2.waitKey(25) & 0xFF == ord('q'): break
```

OUTPUT:-



Practical 6

AIM:

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

PREQUISITE:-

pip install tensorflow numpy pandas

CODE:-

```
import torch
import random
import numpy as
np
from collections import deque
from game import SnakeGameAI, Direction, Point
from model import Linear_QNet, QTrainer
from helper import plot

MAX_MEMORY = 100_000
BATCH_SIZE = 1000
LR = 0.001

class Agent:

    def __init__(self):
        self.n_games = 0
        self.epsilon = 0 # randomness
        self.gamma = 0.9 # discount rate
        self.memory = deque(maxlen=MAX_MEMORY) # popleft()
        self.model = Linear_QNet(11, 256, 3)
        self.trainer = QTrainer(self.model, lr=LR, gamma=self.gamma)


    def get_state(self, game):
        head = game.snake[0]
        point_l = Point(head.x - 20, head.y)
        point_r = Point(head.x + 20, head.y)
        point_u = Point(head.x, head.y - 20)
        point_d = Point(head.x, head.y +
        20)

        dir_l = game.direction == Direction.LEFT
        dir_r = game.direction == Direction.RIGHT
        dir_u = game.direction == Direction.UP
        dir_d = game.direction ==
        Direction.DOWN

        state = [
            # Danger straight
            (dir_r and game.is_collision(point_r)) or
            (dir_l and game.is_collision(point_l)) or
            (dir_u and game.is_collision(point_u)) or
            (dir_d and game.is_collision(point_d)),
```

```
        (dir_d and game.is_collision(point_l))
        or (dir_l and
        game.is_collision(point_u)) or (dir_r
        and game.is_collision(point_d)),

        # Danger left
        (dir_d and game.is_collision(point_r))
        or (dir_u and
        game.is_collision(point_l)) or (dir_r and
        game.is_collision(point_u)) or (dir_l
        and game.is_collision(point_d)),

        # Move direction
        dir_l,
        dir_r,
        dir_u,
        dir_d,

        # Food location
        game.food.x < game.head.x,  # food left
        game.food.x > game.head.x,  # food right
        game.food.y < game.head.y,  # food up
        game.food.y > game.head.y # food down
        ]

    return np.array(state, dtype=int)

def remember(self, state, action, reward, next_state, done):
    self.memory.append((state, action, reward, next_state, done)) # popleft if MAX_MEMORY is reached

def train_long_memory(self):
    if len(self.memory) > BATCH_SIZE:
        mini_sample = random.sample(self.memory, BATCH_SIZE) # list of tuples
    else:
        mini_sample = self.memory

    states, actions, rewards, next_states, dones = zip(*mini_sample)
    self.trainer.train_step(states, actions, rewards, next_states, dones) #for
    state, action, reward, nexrt_state, done in mini_sample:
    #    self.trainer.train_step(state, action, reward, next_state, done)

def train_short_memory(self, state, action, reward, next_state, done):
    self.trainer.train_step(state, action, reward, next_state, done)

def get_action(self, state):
    # random moves: tradeoff exploration / exploitation
    self.epsilon = 80 - self.n_games
    final_move = [0,0,0]
    if random.randint(0, 200) < self.epsilon:
        move = random.randint(0, 2)
        final_move[move] =
    1 else:
        state0 = torch.tensor(state, dtype=torch.float) prediction
        = self.model(state0)
        move = torch.argmax(prediction).item()
```

```
            final_move[move] = 1

        return final_move


def train():
    plot_scores = []
    plot_mean_scores =
    [] total_score = 0
    record = 0
    agent =
    Agent() game
    =
    SnakeGameAI() while
    True:
        # get old state
        state_old = agent.get_state(game)

        # get move
        final_move = agent.get_action(state_old)

        # perform move and get new state
        reward, done, score =
        game.play_step(final_move) state_new =
        agent.get_state(game)

        # train short memory
        agent.train_short_memory(state_old, final_move, reward, state_new, done)

        # remember
        agent.remember(state_old, final_move, reward, state_new, done)

        if done:
            # train long memory, plot
            result game.reset()
            agent.n_games += 1
            agent.train_long_memory()

            if score > record:
                record = score
                agent.model.save()

            print('Game', agent.n_games, 'Score', score, 'Record:', record)

            plot_scores.append(score)
            total_score += score
            mean_score = total_score /
            agent.n_games
            plot_mean_scores.append(mean_score)
            plot(plot_scores, plot_mean_scores)


if __name__ == '__main__':
    train()
```

OUTPUT:-

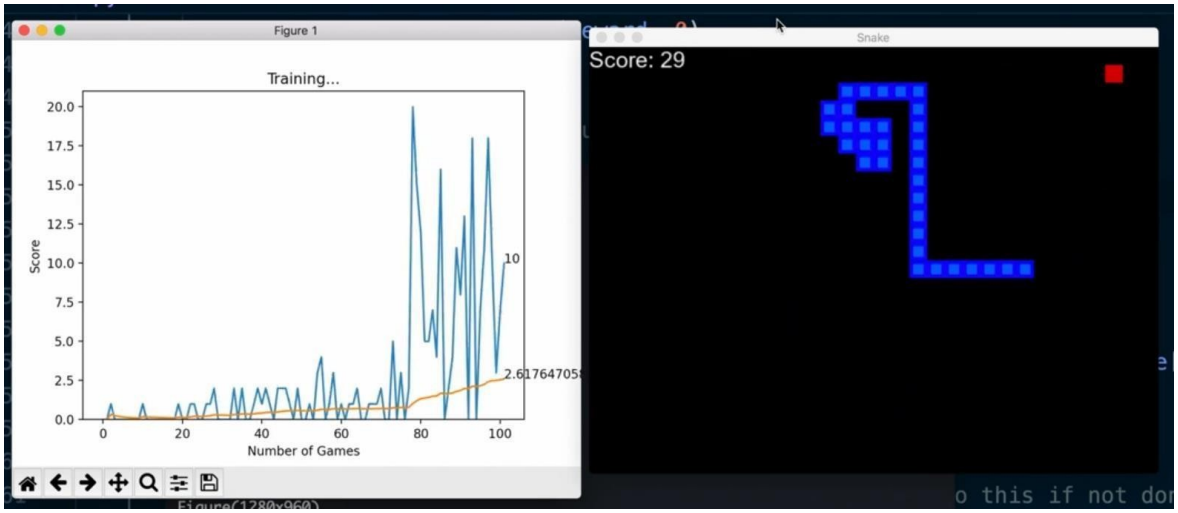


Figure 1
Training...
Score
Number of Games
0
20
40
60
80
100
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
10
Snake
Score: 29

Practical 7

AIM:

Utilizing transfer learning to improve model performance on limited datasets.

PREQUISITE:-

pip install tensorflow numpy pandas

CODE:-

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy

mean = np.array([0.5, 0.5, 0.5])
std = np.array([0.25, 0.25, 0.25])

data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean, std)
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean, std)
    ]),
}

data_dir = 'data/hymenoptera_data'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                              shufle=True, num_workers=0)
               for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(class_names)
```

```
def imshow(inp, title):
    """Imshow for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    inp = std * inp + mean
    inp = np.clip(inp, 0,
    1) plt.imshow(inp)
    plt.title(title)
    plt.show()


# Get a batch of training data
inputs, classes = next(iter(dataloaders['train']))

# Make a grid from batch
out = torchvision.utils.make_grid(inputs)

imshow(out, title=[class_names[x] for x in

classes])

def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs
        - 1)) print('-' * 10)

        # Each epoch has a training and validation
        phase for phase in ['train', 'val']:
            if phase == 'train':
                model.train() # Set model to training mode
            else:
                model.eval()  # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'): outputs
                    = model(inputs)
                    _, preds = torch.max(outputs,
                    1) loss = criterion(outputs,
                    labels)

                    # backward + optimize only if in training
                    phase if phase == 'train':
                        optimizer.zero_grad()
```

```
                loss.backward()
                optimizer.step()

            # statistics
            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds ==
            labels.data)

        if phase == 'train':
            scheduler.step()

        epoch_loss = running_loss / dataset_sizes[phase]
        epoch_acc = running_corrects.double() / dataset_sizes[phase]

        print('{} Loss: {:.4f} Acc:
            {:.4f}'.format( phase, epoch_loss,
            epoch_acc))

        # deep copy the model
        if phase == 'val' and epoch_acc > best_acc:
            best_acc = epoch_acc
            best_model_wts = copy.deepcopy(model.state_dict())

    print()

  time_elapsed = time.time() - since
  print('Training complete in {:.0f}m
     {:.0f}s'.format( time_elapsed // 60,
     time_elapsed % 60))
  print('Best val Acc: {:4f}'.format(best_acc))

  # load best model weights
  model.load_state_dict(best_model_wts)
  return model


#### Finetuning the convnet ####
# Load a pretrained model and reset final fully connected layer.

model = models.resnet18(pretrained=True)
num_ftrs = model.fc.in_features
# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to nn.Linear(num_ftrs, len(class_names)).
model.fc = nn.Linear(num_ftrs, 2)

model = model.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer = optim.SGD(model.parameters(),
lr=0.001)

# StepLR Decays the learning rate of each parameter group by gamma every step_size epochs #
Decay LR by a factor of 0.1 every 7 epochs
# Learning rate scheduling should be applied after optimizer's update #
```
e.g., you should write your code this way:

```
step_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)

model = train_model(model, criterion, optimizer, step_lr_scheduler, num_epochs=25)
model_conv = torchvision.models.resnet18(pretrained=True)
for param in model_conv.parameters():
   param.requires_grad = False

num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

model_conv =

model_conv.to(device) criterion =

nn.CrossEntropyLoss()
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)
```

OUTPUT:

```
Epoch 23/24
----------
train Loss: 0.3284 Acc: 0.8730
val Loss: 0.1729 Acc: 0.9477

Epoch 24/24
----------
train Loss: 0.2955 Acc: 0.8934
val Loss: 0.1615 Acc: 0.9477

Training complete in 1m 28s
Best val Acc: 0.954248

Process finished with exit code 0
```

Practical 8

AIM:

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

PREQUISITE:-

pip install tensorflow numpy pandas

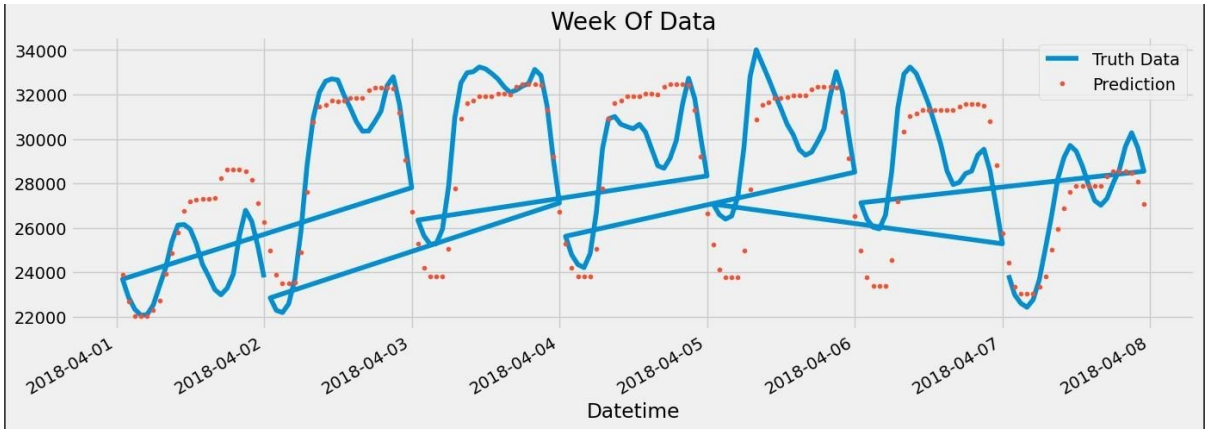
CODE:-

```
import pandas as
pd import numpy as
np
import matplotlib.pyplot as
plt import seaborn as sns

import xgboost as xgb
from sklearn.metrics import
mean_squared_error color_pal =
sns.color_palette()
plt.style.use('fivethirtyeight')

df =
pd.read_csv('/content/hourly-energy-consumption/PJME_hourly.csv'
) df = df.set_index('Datetime')
df.index = pd.to_datetime(df.index)

df.plot(style='.',
        figsize=(15, 5),
        color=color_pal[0],
        title='PJME Energy Use in
        MW')
plt.show()

# Train / Test Split

train = df.loc[df.index < '01-01-
2015'] test = df.loc[df.index >= '01-
01-2015']

fig, ax = plt.subplots(figsize=(15, 5))
train.plot(ax=ax, label='Training Set', title='Data Train/Test Split')
test.plot(ax=ax, label='Test Set')
ax.axvline('01-01-2015', color='black',
ls='--') ax.legend(['Training Set', 'Test
Set']) plt.show()

df.loc[(df.index > '01-01-2010') & (df.index < '01-08-2010')] \
    .plot(figsize=(15, 5), title='Week Of
Data') plt.show()
```

```
        Create time series features based on time series index. """
        df = df.copy()
        df['hour'] = df.index.hour df['dayofweek'] =
        df.index.dayofweek df['quarter'] =
        df.index.quarter df['month'] = df.index.month
        df['year'] = df.index.year df['dayofyear'] =
        df.index.dayofyear df['dayofmonth'] =
        df.index.day
        df['weekofyear'] = df.index.isocalendar().week return df

df = create_features(df)

# Visualize our Feature / Target Relationship

fig, ax = plt.subplots(figsize=(10, 8)) sns.boxplot(data=df, x='hour',
y='PJME_MW') ax.set_title('MW by Hour')
plt.show()

fig, ax = plt.subplots(figsize=(10, 8))
sns.boxplot(data=df, x='month', y='PJME_MW', palette='Blues') ax.set_title('MW by
Month')
plt.show()

# Create our Model

train = create_features(train) test =
create_features(test)

FEATURES = ['dayofyear', 'hour', 'dayofweek', 'quarter', 'month', 'year'] TARGET = 'PJME_MW'

X_train = train[FEATURES] y_train =
train[TARGET]

X_test = test[FEATURES] y_test =
test[TARGET]

reg = xgb.XGBRegressor(base_score=0.5, booster='gbtree',
                       n_estimators=1000,
                       early_stopping_rounds=50,
                       objective='reg:linear', max_depth=3,
                       learning_rate=0.01)
reg.fit(X_train, y_train,
```

```
        eval_set=[(X_train, y_train), (X_test,
        y_test)], verbose=100)

# Feature Importance

fi = pd.DataFrame(data=reg.feature_importances_,
            index=reg.feature_names_in_,
            columns=['importance'])
fi.sort_values('importance').plot(kind='barh', title='Feature
Importance') plt.show()

# Forecast on Test

test['prediction'] = reg.predict(X_test)
df = df.merge(test[['prediction']], how='left', left_index=True,
right_in- dex=True)
ax = df[['PJME_MW']].plot(figsize=(15,
5)) df['prediction'].plot(ax=ax,
style='.') plt.legend(['Truth Data',
'Predictions']) ax.set_title('Raw Dat
and Prediction') plt.show()

ax = df.loc[(df.index > '04-01-2018') & (df.index < '04-08-2018')]
['PJME_MW']
\
    .plot(figsize=(15, 5), title='Week Of Data')
df.loc[(df.index > '04-01-2018') & (df.index < '04-08-2018')]
['prediction'] \
    .plot(style='.')
plt.legend(['Truth
Data','Prediction']) plt.show()

# Score (RMSE)

score = np.sqrt(mean_squared_error(test['PJME_MW'],
test['prediction'])) print(f'RMSE Score on Test set: {score:0.2f}')

# Calculate Error
- Look at the worst and best predicted days

test['error'] = np.abs(test[TARGET] - test['prediction'])
test['date'] = test.index.date
test.groupby(['date'])
```

OUTPUT:-


Week Of Data
34000
32000
30000
28000
26000
24000
22000
2018-04-01
2018-04-02
2018-04-03
2018-04-04
2018-04-05
2018-04-06
2018-04-07
2018-04-08
Datetime
Truth Data
Prediction

Practical 9

AIM:

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

PREQUISITE:-
pip install tensorflow numpy pandas

CODE:-

```
!pip install evalml

### Loading The Dataset
- We can also read the dataset from csv
- then convert to datatable

import evalml
X, y = evalml.demos.load_breast_cancer()
X_train, X_test, y_train, y_test = evalml.preprocessing.split_data(X,
y, prob- lem_type='binary')

X_train.head()

### Running the AutoML to select the best algorithm

import evalml
evalml.problem_types.ProblemTypes.all_problem_types

from evalml.automl import AutoMLSearch
automl = AutoMLSearch(X_train=X_train, y_train=y_train,
problem_type='binary') automl.search()

automl.rankings

### Getting The Best Pipeline

automl.best_pipeline

best_pipeline=automl.best_pipeline

### Let's Check the detailed desscription

automl.describe_pipeline(automl.rankings.iloc[0]["id"])

### Evaluate on hold out data
best_pipeline.score(X_test, y_test,
objectives=["auc","f1","Precision","Re- call"])

### We can also optimize for a problem specific objective
```

```
automl_auc = AutoMLSearch(X_train=X_train, y_train=y_train,
                          problem_type='binary',
                          objective='auc',
                          additional_objectives=['f1', 'precision'],
                          max_batches=1,
                          optimize_thresholds=True)

automl_auc.search()

automl_auc.rankings

automl_auc.describe_pipeline(automl_auc.rankings.iloc[0]

["id"]) best_pipeline_auc = automl_auc.best_pipeline

# get the score on holdout data
best_pipeline_auc.score(X_test, y_test,
objectives=["auc"])

best_pipeline.save("model.pkl")

#### Loading the Model

check_model=automl.load('model.pk

l')
```

OUTPUT:-

	id	pipeline_name	search_order	ranking_score	mean_cv_score	standard_deviation_cv_score	percent_better_than_baseline	high_variance_cv	parameters
0	6	Logistic Regression Classifier w/ Label Encode...	6	0.111823	0.111823	0.033800	99.169626	False	{'Label Encoder': {'positive_label': None}, 'I...
1	4	Elastic Net Classifier w/ Label Encoder + Impu...	4	0.114153	0.114153	0.031724	99.152325	False	{'Label Encoder': {'positive_label': None}, 'I...
2	1	Random Forest Classifier w/ Label Encoder + Im...	1	0.126770	0.126770	0.035172	99.058640	False	{'Label Encoder': {'positive_label': None}, 'I...
3	3	Extra Trees Classifier w/ Label Encoder + Impu...	3	0.150551	0.150551	0.034153	98.882044	False	{'Label Encoder': {'positive_label': None}, 'I...
4	5	XGBoost Classifier w/ Label Encoder + Imputer ...	5	0.150885	0.150885	0.046259	98.879567	False	{'Label Encoder': {'positive_label': None}, 'I...
5	2	LightGBM Classifier w/ Label Encoder + Imputer...	2	0.194301	0.194301	0.052339	98.557171	False	{'Label Encoder': {'positive_label': None}, 'I...
6	0	Mode Baseline Binary Classification Pipeline	0	13.466641	13.466641	0.086133	0.000000	False	{'Label Encoder': {'positive_label': None}, 'B...

```
*****************************************************************************************
INFO:evalml.pipelines.pipeline_base.describe:
*****************************************************************************************
* Logistic Regression Classifier w/ Label Encoder + Imputer + Standard Scaler + Select Columns Transformer *
INFO:evalml.pipelines.pipeline_base.describe:* Logistic Regression Classifier w/ Label Encoder + Imputer + Standard Scaler + Select Columns Transformer *
*****************************************************************************************
INFO:evalml.pipelines.pipeline_base.describe:*****************************************************************************************

INFO:evalml.pipelines.pipeline_base.describe:
Problem Type: binary
INFO:evalml.pipelines.pipeline_base.describe:Problem Type: binary
Model Family: Linear
```

Practical 10

AIM:

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

PREQUISITE:-

pip install tensorflow numpy pandas

CODE:-

```
# Startup Google
CoLab try:
    import google.colab
    COLAB = True
    print("Note: using Google
CoLab") except:
    print("Note: not using Google CoLab")
    COLAB = False

# Nicely formatted time string


def hms_string(sec_elapsed):
    h = int(sec_elapsed / (60 * 60))
    m = int((sec_elapsed % (60 * 60)) /
    60) s = sec_elapsed % 60
    return "{}:{:>02}:{:>05.2f}".format(h, m, s)


# Make use of a GPU or MPS (Apple) if one is available.  (see
module 3.2) import torch
has_mps = torch.backends.mps.is_built()
device = "mps" if has_mps else "cuda" if torch.cuda.is_available() else
"cpu" print(f"Using device: {device}")

from scipy.stats import zscore
import pandas as pd
import
logging
import os
logging.disable(logging.WARNING)


# Read the data
set df =
pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/jh-simple-dataset.csv",
    na_values=['NA', '?'])

# Generate dummies for
```

```
df.drop('job', axis=1, inplace=True)

# Generate dummies for area df
= pd.concat(
        [df, pd.get_dummies(df['area'], prefix="area", dtype=int)], axis=1) df.drop('area', axis=1, inplace=True)

# Missing values for income med =
df['income'].median()
df['income'] = df['income'].fillna(med)

# Standardize ranges
df['income'] = zscore(df['income'])
df['aspect'] = zscore(df['aspect']) df['save_rate'] =
zscore(df['save_rate']) df['age'] = zscore(df['age'])
df['subscriptions'] = zscore(df['subscriptions'])

# Convert to numpy - Classification
x_columns = df.columns.drop('product').drop('id') x =
df[x_columns].values
dummies = pd.get_dummies(df['product'])  # Classification products =
dummies.columns
y = dummies.values

import pandas as pd import
numpy as np import time
import statistics
from scipy.stats import zscore from sklearn
import metrics
from sklearn.model_selection import StratifiedShuffleSplit import torch
import torch.nn as nn import
torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# Convert data to PyTorch tensors
x_tensor = torch.FloatTensor(x).to(device)
y_tensor = torch.LongTensor(np.argmax(y, axis=1)).to( device)  # Convert one-
        hot to index


class NeuralNetwork(nn.Module):
        def __init__(self, input_dim, dropout, neuronPct, neuronShrink): super(NeuralNetwork, self).__init__()

                layers = []
                neuronCount = int(neuronPct * 5000)
```

```
        layer = 0

        prev_count = input_dim
        while neuronCount > 25 and layer < 10:
            layers.append(nn.Linear(prev_count, neuronCount)) prev_count =
            neuronCount layers.append(nn.PReLU())
            layers.append(nn.Dropout(dropout))
            neuronCount = int(neuronCount * neuronShrink) layer += 1

        layers.append(nn.Linear(prev_count, y.shape[1])) layers.append(nn.Softmax(dim=1))
        self.model = nn.Sequential(*layers)

    def forward(self, x): return
        self.model(x)

# Create and print the model
model = NeuralNetwork(x.shape[1], 0.2, 0.1, 0.25).to(device) print(model)

# Evaluation function SPLITS = 2
EPOCHS = 500
PATIENCE = 10

def  evaluate_network(learning_rate=1e-3,dropout=0.2,
                      neuronPct=0.1, neuronShrink=0.25):

    boot = StratifiedShuffleSplit(n_splits=SPLITS, test_size=0.1) mean_benchmark = []
    epochs_needed = []

    for train, test in boot.split(x, np.argmax(y, axis=1)): x_train =
        x_tensor[train]
        y_train = y_tensor[train] x_test =
        x_tensor[test] y_test = y_tensor[test]

        model = NeuralNetwork(x.shape[1],
                              dropout=dropout,  neuronPct=neuronPct,
                              neuronShrink=neuronShrink).to(device)
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(model.parameters(), lr=learning_rate)

        dataset_train = TensorDataset(x_train, y_train)
```

```
            loader_train = DataLoader(dataset_train, batch_size=32, shuffle=True)

            best_loss = float('inf') patience_counter = 0

            for epoch in range(EPOCHS):
                  model.train()
                  for batch_x, batch_y in loader_train: optimizer.zero_grad()
                        outputs = model(batch_x)
                        loss = criterion(outputs, batch_y) loss.backward()
                        optimizer.step()

                  model.eval()
                  with torch.no_grad(): outputs_test =
                        model(x_test)
                        val_loss = criterion(outputs_test, y_test).item()

                   if val_loss < best_loss: best_loss
                        = val_loss patience_counter
                        = 0
                  else:
                        patience_counter += 1

                  if patience_counter >= PATIENCE:
                        epochs_needed.append(epoch) break

            # Evaluate
            withtorch.no_grad(): model.eval()
                  # Move predictions to CPU for evaluation pred =
                  model(x_test).cpu().numpy() y_compare =
                  y_test.cpu().numpy()
                  score = metrics.log_loss(y_compare, pred)
                  mean_benchmark.append(score)

      return -

statistics.mean(mean_benchmark)

print(evaluate_network(learning_rate=1e-3,
                              dropout=0.2,
                        neuronPct=0.1,
                        neuronShrink=0.25)
                        )

print(evaluate_network( dropout=0.2,
      neuronPct=0.1,
```

```
    neuronShrink=0.25))
!pip install bayesian-optimization

from bayes_opt import
BayesianOptimization import time
import warnings
warnings.filterwarnings("ignore",
category=RuntimeWarning) pbounds = {'dropout': (0.0,
0.499),
          'learning_rate': (0.0, 0.1),
          'neuronPct': (0.01, 1),
          'neuronShrink': (0.01, 1)
          }

optimizer = BayesianOptimization(
    f=evaluate_network,
    pbounds=pbounds,
    verbose=2,  # verbose = 1 prints only when a
    maximum # is observed, verbose = 0 is silent
    random_state=1,
)
start_time = time.time()
optimizer.maximize(init_points=10,
n_iter=20,) time_took = time.time() -
start_time

print(f"Total runtime: {hms_string(time_took)}")
```

OUTPUT:-

iter	target	dropout	learni...	neuronPct	neuron...
1	-8.29	0.2081	0.07203	0.01011	0.3093
2	-8.29	0.07323	0.009234	0.1944	0.3521
3	-12.71	0.198	0.05388	0.425	0.6884
4	-8.29	0.102	0.08781	0.03711	0.6738
5	-12.95	0.2082	0.05587	0.149	0.2061
6	-8.29	0.3996	0.09683	0.3203	0.6954
7	-8.29	0.4373	0.08946	0.09419	0.04866
8	-9.167	0.08475	0.08781	0.1074	0.4269
9	-9.167	0.478	0.05332	0.695	0.3224
10	-11.88	0.3426	0.08346	0.02811	0.7526
11	-12.0	0.4915	0.07209	0.7089	0.323
12	-9.167	0.2208	0.04135	0.5523	0.7468

Practical 11

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.

PREQUISITE:-

pip install tensorflow numpy pandas

CODE:-

```
!pip install transformers # Installing the transformers library

import transformers # transformers library
import torch # PyTorch, we are using PyTorch as our library

# We are going to load in GPT-2 using the transformers library gpt_tokenizer =
transformers.GPT2Tokenizer.from_pretrained('gpt2-large') # Loading in model now...
gpt_model = transformers.GPT2LMHeadModel.from_pretrained('gpt2-large') # Takes a while
to run...

## Making a function that will generate text for us ##
def gen_text(prompt_text, tokenizer, model, n_seqs=1, max_length=25): # n_seqs is the
    number of sentences to generate
    # max_length is the maximum length of the sentence
    encoded_prompt = tokenizer.encode(prompt_text, add_special_tokens=False, re- turn_tensors="pt")
    # We are encoding the text using the gpt tokenizer. The return tensors are of type "pt"
    # since we are using PyTorch, not tensorflow output_sequences =
    model.generate(
            input_ids=encoded_prompt,
            max_length=max_length+len(encoded_prompt), # The model has to generate
            something, # so we add the length of the original sequence to max_length
            temperature=1.0,
            top_k=0,
            top_p=0.9,
            repetition_penalty=1.2, # To ensure that we dont get repeated phrases do_sample=True,
            num_return_sequences=n_seqs
    ) # We feed the encoded input into the model. ## Getting the
    output ##
    if len(output_sequences.shape) > 2:
        output_sequences.squeeze_() # the _ indicates that the operation will be done in-place
    generated_sequences = []
    for generated_sequence_idx, generated_sequence in enumerate(output_se-
        quences): generated_sequence = generated_sequence.tolist()
```

```
    text = tokenizer.decode(generated_sequence)
    total_sequence = (
        prompt_text + text[len(tokenizer.decode(encoded_prompt[0],
clean_up_tokenization_spaces=True, )) :]
    )
    generated_sequences.append(total_sequence)
  return generated_sequences

# Lots of syntax errors, but now we can test our model
## One important note: in our function, on line 5, make sure that
# return_tensor is return_tensors, otherwise you will get an
error like # this:
#####
#  Another  important  note:  on  line  27  of  the  function,
instead    of    #    clear_up_tokenization_spaces,    write
clean_up_tokenization_spaces ####
gen_text("Legolas  and  Gimli  advanced  on  the  orcs,  raising  their
weapons with a harrowing war cry",gpt_tokenizer,gpt_model)

# Sequence length was too small, lets increase it
gen_text("Legolas and Gimli advanced on the orcs, raising their weapons
with a harrowing war cry",
        gpt_tokenizer,
        gpt_model,
        max_length=100)
# Will take some time......

# We can demostrate n_seqs here
gen_text("Legolas and Gimli advanced on the orcs, raising their weapons
with a harrowing war cry",
        gpt_tokenizer,
        gpt_model,
        max_length=40,
        n_seqs=3) # Will take even longer....
```

OUTPUT:-

```
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 reliable results.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
["Legolas and Gimli advanced on the orcs, raising their weapons with a harrowing war cry.\n\nAll four fell dead; because of them, many things happened, some nearsighted and out of focus',
 'Legolas and Gimli advanced on the orcs, raising their weapons with a harrowing war cry as they shouted. Behind them came tattered wood paneling leaking from around a ruined pool of red liquid,-',
 "Legolas and Gimli advanced on the orcs, raising their weapons with a harrowing war cry. Glorfindel looked at his hobbit companion expectantly, while Rohan's Master said a short prayer",
 'Legolas and Gimli advanced on the orcs, raising their weapons with a harrowing war cry. But they were quickly halted by lowing orc knights atop stilts on The Door to Mordor. Two of',
 'Legolas and Gimli advanced on the orcs, raising their weapons with a harrowing war cry. Their words were without rhyme or reason:\n\n"It is important to take this new information into consideration',
 "Legolas and Gimli advanced on the orcs, raising their weapons with a harrowing war cry. They did not bring the fight to an end in front of King Theoden's camp until the last orc"]
```

Practical 12

AIM:

Experiment with neural networks like GANS (Generative Adversarial Networks) using Python libraries like TensorFlow or PyTorch to generate new images based on a dataset of images.

PREQUISITE:-

pip install tensorflow numpy pandas

CODE:-

Setup

```
import tensorflow as

tf tf.__version_

# To generate GIFs
!pip install imageio
!pip install git+https://github.com/tensorflow/docs

import glob
import
imageio
import matplotlib.pyplot as
plt import numpy as np
import os
import
PIL
from tensorflow.keras import
layers import time

from IPython import display

### Load and prepare the dataset



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

train_images = train_images.reshape(train_images.shape[0], 28, 28,
1).astype('float32')
train_images = (train_images - 127.5) / 127.5 # Normalize the images
to [-1, 1]

BUFFER_SIZE = 60000
BATCH_SIZE = 256

# Batch and shuffle the data
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuf-
```

```
### The Generator



def make_generator_model(): model =
        tf.keras.Sequential()
        model.add(layers.Dense(7*7*256, use_bias=False,
        input_shape=(100,))) model.add(layers.BatchNormalization())
        model.add(layers.LeakyReLU())

        model.add(layers.Reshape((7, 7, 256)))
        assert model.output_shape == (None, 7, 7, 256)  # Note: None is the batch size

        model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), pad- ding='same', use_bias=False))
        assert model.output_shape == (None, 7, 7, 128)
        model.add(layers.BatchNormalization()) model.add(layers.LeakyReLU())

        model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), pad- ding='same', use_bias=False))
        assert model.output_shape == (None, 14, 14, 64)
        model.add(layers.BatchNormalization()) model.add(layers.LeakyReLU())

        model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), pad- ding='same', use_bias=False,
activation='tanh'))
        assert model.output_shape == (None, 28, 28, 1)

        return model

generator = make_generator_model()

noise = tf.random.normal([1, 100])
generated_image = generator(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')

### The Discriminator


def make_discriminator_model(): model =
        tf.keras.Sequential()
        model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                                                input_shape=[28, 28, 1]))
        model.add(layers.LeakyReLU())
```

```
        model.add(layers.Dropout(0.3))

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

        model.add(layers.Flatten()) model.add(layers.Dense(1))

        return model

discriminator = make_discriminator_model() decision =
discriminator(generated_image) print (decision)

## Define the loss and optimizers



# This method returns a helper function to compute cross entropy loss cross_entropy =
tf.keras.losses.BinaryCrossentropy(from_logits=True)

### Discriminator loss


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) total_loss = real_loss + fake_loss
        return total_loss

### Generator loss


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

generator_optimizer = tf.keras.optimizers.Adam(1e-4) discriminator_optimizer
= tf.keras.optimizers.Adam(1e-4)

### Save checkpoints

checkpoint_dir = './training_checkpoints' checkpoint_prefix =
os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                 discriminator_optimizer=discriminator_opti-
mizer,
                                 generator=generator,
                                 discriminator=discriminator)
```

```
## Define the training loop


EPOCHS = 50
noise_dim = 100
num_examples_to_generate = 16

# You will reuse this seed overtime (so it's easier) # to visualize
progress in the animated GIF)
seed = tf.random.normal([num_examples_to_generate, noise_dim])

# Notice the use of `tf.function`
# This annotation causes the function to be "compiled". @tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    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.traina- ble_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discrimina- tor.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator, genera- tor.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
discriminator.trainable_variables))

def train(dataset, epochs): for epoch in
  range(epochs):
    start = time.time()

    for image_batch in dataset:
        train_step(image_batch)

    # Produce images for the GIF as you go
    display.clear_output(wait=True)
    generate_and_save_images(generator,
                             epoch + 1, seed)
```

```
      # Save the model every 15 epochs if (epoch +
      1) % 15 == 0:
          checkpoint.save(file_prefix = checkpoint_prefix)

      print ('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-

  start)) # Generate after the final epoch
  display.clear_output(wait=True)
  generate_and_save_images(generator
  ,
                                       epochs,
                                       seed)
```

Generate and save images

```
def generate_and_save_images(model, epoch, test_input): # Notice
  `training` is set to False.
  # This is so all layers run in inference mode (batchnorm). predictions
  = model(test_input, training=False)

  fig = plt.figure(figsize=(4, 4))

  for i in range(predictions.shape[0]): plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray') plt.axis('off')

  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch)) plt.show()

## Train the model


train(train_dataset, EPOCHS)

Restore the latest checkpoint.

checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir)

) ## Create a GIF


# Display a single image using the epoch number def
display_image(epoch_no):
  return PIL.Image.open('image_at_epoch_{:04d}.png'.format(epoch_no))

display_image(EPOCHS)
```

```
anim_file = 'dcgan.gif'

with imageio.get_writer(anim_file, mode='I') as
  writer: filenames = glob.glob('image*.png')
  filenames =
  sorted(filenames) for
  filename in filenames:
    image = imageio.imread(filename)
    writer.append_data(image)
  image = imageio.imread(filename)
  writer.append_data(image)

import tensorflow_docs.vis.embed as embed
embed.embed_file(anim_file)
```

)

OUTPUT:-