Name: Siddhesh Santosh More **Seat No:** 1313160
Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

INDEX

Practical No	Practicals	Date	Sign
1.	Implementing advanced deep learning algorithms such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) using Python libraries like TensorFlow or PyTorch.		
2.	Building a natural language processing (NLP) model for sentiment analysis or text classification.		
3.	Creating a chatbot using advanced techniques like transformer models.		
4.	Developing a recommendation system using collaborative filtering or deep learning approaches.		
5.	Implementing a computer vision project, such as object detection or image segmentation.		
6.	Training a generative adversarial network (GAN) for generating realistic images.		
7.	Building a deep learning model for time series forecasting or anomaly detection.		
8.	Utilizing transfer learning to improve model performance on limited datasets.		
9.	Using advanced optimization techniques like evolutionary algorithms or Bayesian optimization for hyperparameter tuning.		
10.	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.		

Name: Siddhesh Santosh More **Seat No:** 1313160
Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

PRACTICAL NO. 1

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

Step-by-Step Implementation for CNN (TensorFlow)
1. Install Dependencies:

```
pip install tensorflow matplotlib
```

2. Python Code for CNN (TensorFlow):

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
import matplotlib.pyplot as plt

# Step 1: Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Step 2: Normalize the data
x_train = x_train / 255.0
x_test = x_test / 255.0

# Step 3: One-hot encode the labels
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Step 4: Define the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    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')
```

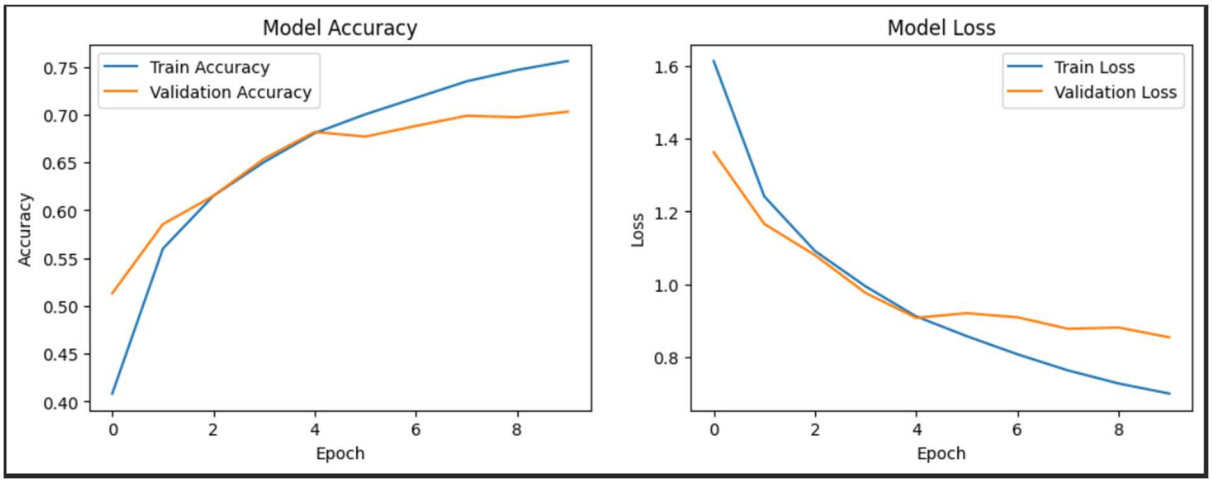
```
])

# Step 5: Compile the model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Step 6: Train the model
history = model.fit(
    x_train,
    y_train,
    epochs=10,
    batch_size=64,
    validation_data=(x_test, y_test)
)

# Step 7: Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"Test Accuracy: {test_acc:.4f}")

# Step 8: Plot training history
plt.figure(figsize=(12, 4))

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

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

plt.show()
```

Name: Siddhesh Santosh More **Seat No:** 1313160

Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

OUTPUT:

```
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 70s 87ms/step - accuracy: 0.3101 - loss: 1.8590 - val_accuracy: 0.5133 - val_loss: 1.3626
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 82s 87ms/step - accuracy: 0.5457 - loss: 1.2822 - val_accuracy: 0.5855 - val_loss: 1.1658
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 66s 84ms/step - accuracy: 0.6068 - loss: 1.1148 - val_accuracy: 0.6152 - val_loss: 1.0804
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 66s 84ms/step - accuracy: 0.6480 - loss: 1.0038 - val_accuracy: 0.6540 - val_loss: 0.9765
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 66s 84ms/step - accuracy: 0.6798 - loss: 0.9153 - val_accuracy: 0.6820 - val_loss: 0.9083
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 84s 87ms/step - accuracy: 0.7019 - loss: 0.8550 - val_accuracy: 0.6771 - val_loss: 0.9210
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 67s 86ms/step - accuracy: 0.7208 - loss: 0.8030 - val_accuracy: 0.6883 - val_loss: 0.9097
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 69s 88ms/step - accuracy: 0.7390 - loss: 0.7560 - val_accuracy: 0.6989 - val_loss: 0.8781
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 66s 84ms/step - accuracy: 0.7481 - loss: 0.7211 - val_accuracy: 0.6974 - val_loss: 0.8816
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 66s 84ms/step - accuracy: 0.7602 - loss: 0.6920 - val_accuracy: 0.7032 - val_loss: 0.8547
313/313 - 5s - 16ms/step - accuracy: 0.7032 - loss: 0.8547
Test Accuracy: 0.7032
```



Step-by-Step Implementation for RNN (PyTorch)

1. Install Dependencies:

pip install torch torchvision torchtext matplotlib

```
Installing collected packages: torchtext
Successfully installed torchtext-0.18.0
```

2. Python Code for RNN (PyTorch):

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import matplotlib.pyplot as plt
```

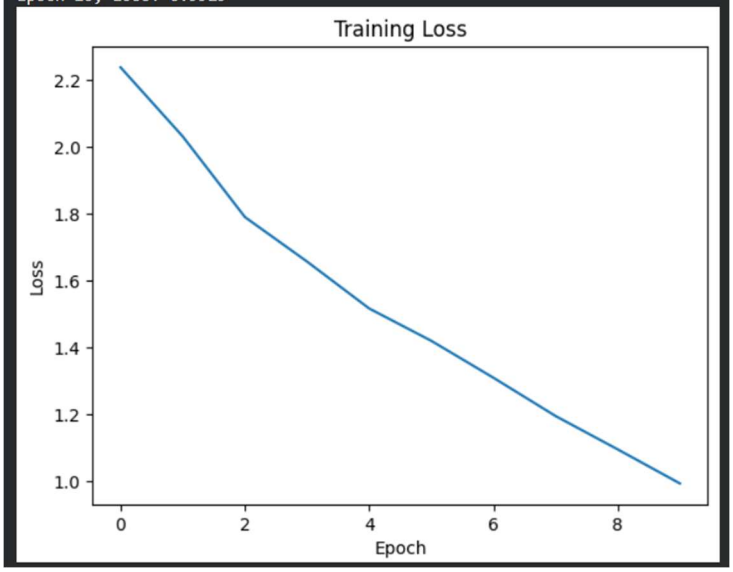
```
# ----------------------------
# Simple Text Dataset (IMDB-like)
# ----------------------------
texts = [
    "this movie was amazing",
    "i loved the film",
    "fantastic acting and story",
    "this movie was terrible",
    "i hated the film",
    "worst movie ever"
]

labels = [1, 1, 1, 0, 0, 0]  # 1 = positive, 0 = negative

# ----------------------------
# Build Vocabulary
# ----------------------------
def tokenize(text):
    return text.lower().split()

vocab = {"<pad>": 0}
for text in texts:
    for token in tokenize(text):
        if token not in vocab:
            vocab[token] = len(vocab)

PAD_IDX = vocab["<pad>"]

# ----------------------------
# Dataset Class
# ----------------------------
class TextDataset(Dataset):
    def __init__(self, texts, labels, vocab):
        self.texts = texts
        self.labels = labels
        self.vocab = vocab

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
```

```
        tokens = tokenize(self.texts[idx])
        encoded = [self.vocab[token] for token in tokens]
        return torch.tensor(encoded), torch.tensor(self.labels[idx])

# ----------------------------
# Collate Function (Padding)
# ----------------------------
def collate_batch(batch):
    texts, labels = zip(*batch)
    texts = nn.utils.rnn.pad_sequence(texts, batch_first=True, padding_value=PAD_IDX)
    return texts, torch.tensor(labels)

dataset = TextDataset(texts, labels, vocab)
loader = DataLoader(dataset, batch_size=2, shuffle=True, collate_fn=collate_batch)

# ----------------------------
# RNN Model
# ----------------------------
class RNNModel(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=PAD_IDX)
        self.rnn = nn.RNN(embed_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, 2)

    def forward(self, x):
        x = self.embedding(x)
        _, h = self.rnn(x)
        return self.fc(h.squeeze(0))

model = RNNModel(len(vocab), 32, 64)

# ----------------------------
# Training Setup
# ----------------------------
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

losses = []

# ----------------------------
# Train Model
```

```
# ----------------------------
for epoch in range(10):
    total_loss = 0
    for texts_batch, labels_batch in loader:
        optimizer.zero_grad()
        outputs = model(texts_batch)
        loss = criterion(outputs, labels_batch)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    losses.append(total_loss)
    print(f"Epoch {epoch+1}, Loss: {total_loss:.4f}")

# ----------------------------
# Plot Loss
# ----------------------------
plt.plot(losses)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training Loss")
plt.show()
```

OUTPUT:

```
Epoch 1, Loss: 2.2369
Epoch 2, Loss: 2.0304
Epoch 3, Loss: 1.7895
Epoch 4, Loss: 1.6564
Epoch 5, Loss: 1.5161
Epoch 6, Loss: 1.4194
Epoch 7, Loss: 1.3089
Epoch 8, Loss: 1.1945
Epoch 9, Loss: 1.0948
Epoch 10, Loss: 0.9929
```



Name: Siddhesh Santosh More **Seat No:** 1313160
Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

PRACTICAL NO. 2

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

Step-01:
To Install Jupyter Notebook using command line

Step-02
Install required library files
!pip install pandas scikit-learn nltk

Step-03 To run the script

```
import pandas as pd
import nltk
import string

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

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

# Download NLTK data
nltk.download('stopwords', quiet=True)
nltk.download('punkt', quiet=True)
nltk.download('punkt_tab', quiet=True)

# ----------------------------
# Dataset
# ----------------------------
data = {
    "text": [
        "I love this product! It's amazing and works perfectly.",
        "The quality is terrible, I'm very disappointed.",
        "Good value for money, happy with my purchase.",
        "Awful experience, would not recommend it to anyone.",
        "Decent, but could be improved in some areas."
    ],
    "label": ["positive", "negative", "positive", "negative", "neutral"]
}
```

```
# Create DataFrame
df = pd.DataFrame(data)
print("Dataset:\n", df)

# ----------------------------
# Text Preprocessing Function
# ----------------------------
stop_words = set(stopwords.words("english"))

def preprocess_text(text):
    text = text.lower()
    tokens = word_tokenize(text)
    tokens = [word for word in tokens if word not in string.punctuation]
    tokens = [word for word in tokens if word not in stop_words]
    return " ".join(tokens)

# Apply preprocessing
df["cleaned_text"] = df["text"].apply(preprocess_text)

# ----------------------------
# Feature Extraction
# ----------------------------
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df["cleaned_text"])
y = df["label"]

# ----------------------------
# Train-Test Split
# ----------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# ----------------------------
# Model Training
# ----------------------------
model = MultinomialNB()
model.fit(X_train, y_train)

# ----------------------------
# Evaluation
```

```
# ----------------------------
y_pred = model.predict(X_test)

print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

# ----------------------------
# Sentiment Prediction Function
# ----------------------------
def predict_sentiment(text):
    processed_text = preprocess_text(text)
    vectorized_text = vectorizer.transform([processed_text])
    return model.predict(vectorized_text)[0]

# Test prediction
new_text = "This is the best product I've ever bought!"
print("\nNew Text:", new_text)
print("Predicted Sentiment:", predict_sentiment(new_text))
```

OUTPUT :

```
Dataset:
                                                text     label
0  I love this product! It's amazing and works pe...  positive
1    The quality is terrible, I'm very disappointed.  negative
2      Good value for money, happy with my purchase.  positive
3  Awful experience, would not recommend it to an...  negative
4       Decent, but could be improved in some areas.   neutral

Accuracy: 0.0

Classification Report:

              precision    recall  f1-score   support

    negative       0.00      0.00      0.00       1.0
    positive       0.00      0.00      0.00       0.0

    accuracy                           0.00       1.0
   macro avg       0.00      0.00      0.00       1.0
weighted avg       0.00      0.00      0.00       1.0


New Text: This is the best product I've ever bought!
Predicted Sentiment: positive
```

Name: Siddhesh Santosh More **Seat No:** 1313160
Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

PRACTICAL NO. 3

Creating a chatbot using advanced techniques like transformer models.

Step -01: Install required library

```
!pip install transformers torch
```

Step-02:

Code:

```
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch
# Load the model and tokenizer for DialoGPT
model_name = "microsoft/DialoGPT-medium"  # Options: small, medium, large
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)
# Function for chatbot interaction
def chatbot_response(prompt, chat_history_ids=None):
    # Tokenize the input prompt
    input_ids = tokenizer.encode(prompt + tokenizer.eos_token, return_tensors="pt")
    # Append to chat history
    bot_input_ids = (
        torch.cat([chat_history_ids, input_ids], dim=-1)
        if chat_history_ids is not None
        else input_ids
    )
    # Generate response using the model
    chat_history_ids = model.generate(
        bot_input_ids,
        max_length=1000,
        pad_token_id=tokenizer.eos_token_id,
        top_k=50,
        top_p=0.95,
        temperature=0.7,
        do_sample=True,
    )
    # Decode and return the response
    response = tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0],
skip_special_tokens=True)
    return response, chat_history_ids
# Start the chatbot
print("Chatbot: Hello! I am a chatbot. How can I help you today?")
chat_history = None  # To maintain context in conversation
```

```
while True:
    user_input = input("You: ")
    # Exit condition
    if user_input.lower() in ["exit", "quit", "bye"]:
        print("Chatbot: Goodbye! Have a great day!")
        break
    # Get response from chatbot
    response, chat_history = chatbot_response(user_input, chat_history)
    print(f"Chatbot: {response}")
```

OUTPUT:

```
Chatbot: Hello! I am a chatbot. How can I help you today?
You:  hi how are you
The attention mask is not set and cannot be inferred from input beca
havior. Please pass your input's `attention_mask` to obtain reliable
Chatbot: im good how about you
You:  im also fine
Chatbot: what do you want to talk about
You:  can you suugest any book
Chatbot: im about to go on a trip to a barbeque
You:  ok bye
Chatbot: good bye
You:  good afternoon
Chatbot: good afternoon
You:  are you busy
Chatbot: idk, maybe
You:  bye
Chatbot: Goodbye! Have a great day!
```

Name: Siddhesh Santosh More **Seat No:** 1313160
Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

PRACTICAL NO. 4

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

SOURCE CODE:
Step 1: Install Required Libraries
pip install tensorflow numpy pandas matplotlib scikit-learn

Step 2: Download the Dataset
Download the MovieLens 100K dataset from grouplens.org/datasets/movielens. Extract the dataset into a folder.
Alternatively, the code below assumes that the u.data file is in the ml-100k folder.

Step 3: Python Code for the Recommendation System

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

from sklearn.model_selection import train_test_split

# ------------------------------------
# Step 1: Load and preprocess dataset
# ------------------------------------
# Download the MovieLens 100K dataset if not already present
!wget -nc http://files.grouplens.org/datasets/movielens/ml-100k.zip -P .
!unzip -o ml-100k.zip -d .

file_path = "ml-100k/u.data"
column_names = ['user_id', 'item_id', 'rating', 'timestamp']

data = pd.read_csv(file_path, sep='\t', names=column_names)

# Normalize IDs to start from 0
data['user_id'] -= 1
data['item_id'] -= 1

num_users = data['user_id'].max() + 1
num_items = data['item_id'].max() + 1

print(f"Number of users: {num_users}, Number of items: {num_items}")
```

```
# ----------------------------------
# Step 2: Train-test split
# ----------------------------------
train_data, test_data = train_test_split(
    data, test_size=0.2, random_state=42
)

# ----------------------------------
# Step 3: TensorFlow dataset creation
# ----------------------------------
def create_tf_dataset(df):
    users = tf.constant(df['user_id'].values, dtype=tf.int32)
    items = tf.constant(df['item_id'].values, dtype=tf.int32)
    ratings = tf.constant(df['rating'].values, dtype=tf.float32)

    return tf.data.Dataset.from_tensor_slices(
        ((users, items), ratings)
    ).shuffle(1024).batch(32)

train_dataset = create_tf_dataset(train_data)
test_dataset = create_tf_dataset(test_data)

# ----------------------------------
# Step 4: Matrix Factorization Model
# ----------------------------------
class MatrixFactorizationModel(tf.keras.Model):
    def __init__(self, num_users, num_items, embedding_dim=50):
        super().__init__()
        self.user_embedding = tf.keras.layers.Embedding(
            num_users, embedding_dim
        )
        self.item_embedding = tf.keras.layers.Embedding(
            num_items, embedding_dim
        )

    def call(self, inputs):
        user_vector = self.user_embedding(inputs[0])
        item_vector = self.item_embedding(inputs[1])
        return tf.reduce_sum(user_vector * item_vector, axis=1)

model = MatrixFactorizationModel(num_users, num_items)
```

```
# ------------------------------------
# Step 5: Compile model
# ------------------------------------
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
    loss="mse",
    metrics=["mae"]
)

# ------------------------------------
# Step 6: Train model
# ------------------------------------
history = model.fit(
    train_dataset,
    validation_data=test_dataset,
    epochs=10
)

# ------------------------------------
# Step 7: Evaluate model
# ------------------------------------
test_loss, test_mae = model.evaluate(test_dataset)
print(f"Test Loss: {test_loss:.4f}, Test MAE: {test_mae:.4f}")

# ------------------------------------
# Step 8: Plot training history
# ------------------------------------
plt.figure(figsize=(12, 4))

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

plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label="Train MAE")
plt.plot(history.history['val_mae'], label="Validation MAE")
plt.xlabel("Epoch")
```

```
plt.ylabel("Mean Absolute Error")
plt.title("MAE over Epochs")
plt.legend()
plt.show()
# ------------------------------------
# Step 9: Recommendation Function
# ------------------------------------
def recommend(user_id, top_k=5):
    user_tensor = tf.constant([user_id] * num_items, dtype=tf.int32)
    item_tensor = tf.constant(np.arange(num_items), dtype=tf.int32)

    predictions = model.predict((user_tensor, item_tensor), verbose=0)
    top_items = np.argsort(-predictions)[:top_k]
    return top_items
# Example recommendation
user_id = 0
recommended_items = recommend(user_id)

print(f"Recommended items for user {user_id}: {recommended_items}")
```

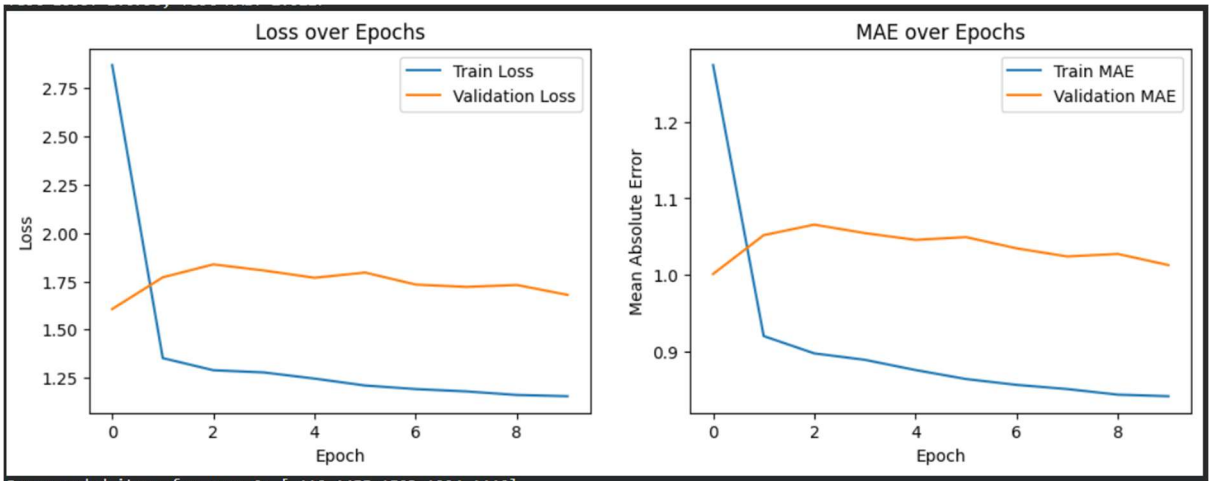
OUTPUT:

```
--2026-01-26 08:29:54--  http://files.grouplens.org/datasets/movielens/ml-100k.zip
Resolving files.grouplens.org (files.grouplens.org)... 128.101.96.204
Connecting to files.grouplens.org (files.grouplens.org)|128.101.96.204|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://files.grouplens.org/datasets/movielens/ml-100k.zip [following]
--2026-01-26 08:29:55--  https://files.grouplens.org/datasets/movielens/ml-100k.zip
Connecting to files.grouplens.org (files.grouplens.org)|128.101.96.204|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4924029 (4.7M) [application/zip]
Saving to: ‘./ml-100k.zip’

ml-100k.zip         100%[===================>]   4.70M  3.53MB/s    in 1.3s

2026-01-26 08:29:57 (3.53 MB/s) - ‘./ml-100k.zip’ saved [4924029/4924029]

Archive:  ml-100k.zip
   creating: ./ml-100k/
  inflating: ./ml-100k/allbut.pl
  inflating: ./ml-100k/mku.sh
  inflating: ./ml-100k/README
  inflating: ./ml-100k/u.data
  inflating: ./ml-100k/u.genre
  inflating: ./ml-100k/u.info
  inflating: ./ml-100k/u.item
  inflating: ./ml-100k/u.occupation
  inflating: ./ml-100k/u.user
  inflating: ./ml-100k/u1.base
  inflating: ./ml-100k/u1.test
  inflating: ./ml-100k/u2.base
  inflating: ./ml-100k/u2.test
  inflating: ./ml-100k/u3.base
  inflating: ./ml-100k/u3.test
  inflating: ./ml-100k/u4.base
  inflating: ./ml-100k/u4.test
  inflating: ./ml-100k/u5.base
```

Name: Siddhesh Santosh More **Seat No:** 1313160

Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

```
Number of users: 943, Number of items: 1682
Epoch 1/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - loss: 5.6853 - mae: 1.8783 - val_loss: 1.6062 - val_mae: 1.0010
Epoch 2/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - loss: 1.4463 - mae: 0.9481 - val_loss: 1.7704 - val_mae: 1.0519
Epoch 3/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - loss: 1.2955 - mae: 0.9010 - val_loss: 1.8372 - val_mae: 1.0656
Epoch 4/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 1.2882 - mae: 0.8934 - val_loss: 1.8056 - val_mae: 1.0546
Epoch 5/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 8s 2ms/step - loss: 1.2508 - mae: 0.8771 - val_loss: 1.7681 - val_mae: 1.0457
Epoch 6/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - loss: 1.2029 - mae: 0.8619 - val_loss: 1.7949 - val_mae: 1.0493
Epoch 7/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - loss: 1.1993 - mae: 0.8579 - val_loss: 1.7329 - val_mae: 1.0346
Epoch 8/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 1.1795 - mae: 0.8519 - val_loss: 1.7207 - val_mae: 1.0240
Epoch 9/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - loss: 1.1820 - mae: 0.8483 - val_loss: 1.7308 - val_mae: 1.0273
Epoch 10/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - loss: 1.1583 - mae: 0.8415 - val_loss: 1.6796 - val_mae: 1.0127
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 982us/step - loss: 1.6623 - mae: 1.0053
Test Loss: 1.6796, Test MAE: 1.0127
```




```
Recommended items for user 0: [ 118 1455 1593 1294 1446]
```

Name: Siddhesh Santosh More **Seat No:** 1313160
Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

PRACTICAL NO. 5

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

SOURCE CODE:
Step-01 Install required libraries
!pip install torch torchvision numpy opencv-python matplotlib ultralytics

Code:

```
from ultralytics import YOLO
import cv2
import matplotlib.pyplot as plt

# Load a pre-trained YOLOv8 model
model = YOLO("yolov8n.pt")  # lightweight & recommended

# Image path (CHANGE THIS)
image_path = r"/content/ash.jpg"

# Read and convert image
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Perform object detection
results = model.predict(image, conf=0.5)
# Visualize results
annotated_image = results[0].plot()

plt.imshow(annotated_image)
plt.axis("off")
plt.show()
```

OUTPUT:

Name: Siddhesh Santosh More **Seat No:** 1313160
Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

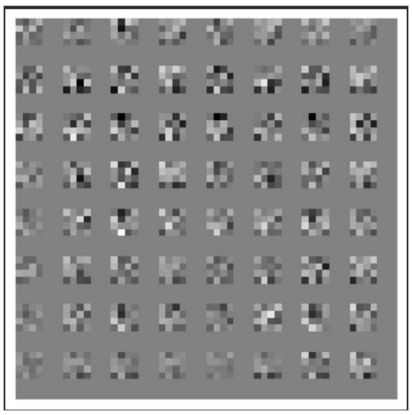
PRACTICAL NO. 6

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

SOURCE CODE:

```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
# Define the generator model
def build_generator():
    model = tf.keras.Sequential([
        layers.Dense(128, activation="relu", input_shape=(100,)),
        layers.Reshape((4, 4, 8)),
        layers.Conv2DTranspose(
            64, (4, 4), strides=(2, 2), padding="same", activation="relu"
        ),
        layers.Conv2DTranspose(
            1, (4, 4), strides=(7, 7), padding="same", activation="sigmoid"
        )
    ])
    return model
# Create generator
generator = build_generator()
# Generate random noise
noise = tf.random.normal([1, 100])
# Generate image
generated_image = generator(noise)
# Visualize the generated image
plt.imshow(generated_image[0, :, :, 0], cmap="gray")
plt.axis("off")
plt.show()
```

OUTPUT :



Name: Siddhesh Santosh More **Seat No:** 1313160
Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

PRACTICAL NO. 7

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

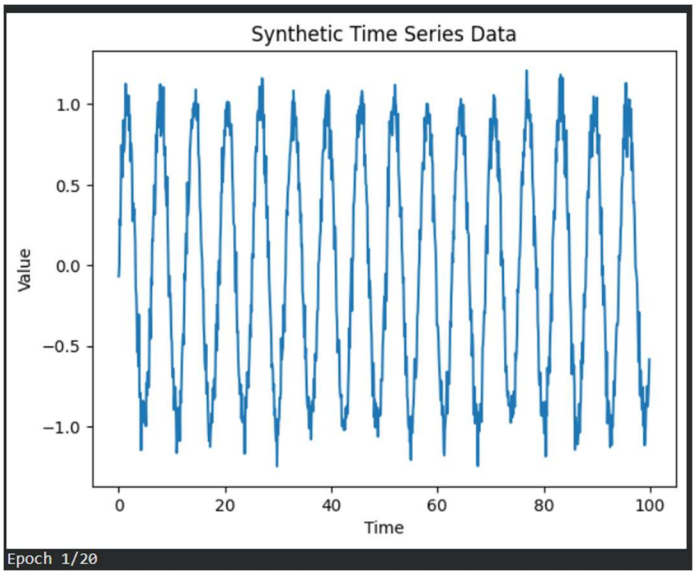
Step-01 Install required libraries
!pip install numpy pandas matplotlib tensorflow

Step-02:
Code:

```
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, Dropout

time = np.arange(0, 100, 0.1)
data = np.sin(time) + np.random.normal(0, 0.1, len(time))  # Sine wave with noise
plt.plot(time, data)
plt.title("Synthetic Time Series Data")
plt.xlabel("Time")
plt.ylabel("Value")
plt.show()

# Convert to a DataFrame
df = pd.DataFrame(data, columns=["value"])

# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
df["value"] = scaler.fit_transform(df[["value"]])

def create_sequences(data, sequence_length):
    sequences = []
    labels = []
    for i in range(len(data) - sequence_length):
        seq = data[i:i + sequence_length]
        label = data[i + sequence_length]
        sequences.append(seq)
        labels.append(label)
    return np.array(sequences), np.array(labels)

# Create sequences
```

```
sequence_length = 50
data_values = df["value"].values
X, y = create_sequences(data_values, sequence_length)

# Reshape for LSTM input (samples, timesteps, features)
X = X.reshape(X.shape[0], X.shape[1], 1)

# Define the LSTM model
model = Sequential([
    LSTM(50, activation="relu", input_shape=(sequence_length, 1), return_sequences=True),
    Dropout(0.2),
    LSTM(50, activation="relu", return_sequences=False),
    Dropout(0.2),
    Dense(1)
])
model.compile(optimizer="adam", loss="mean_squared_error")

# Train the model
history = model.fit(X, y, epochs=20, batch_size=32, validation_split=0.2)

# Plot training loss
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
plt.title("Training and Validation Loss")
plt.show()
```

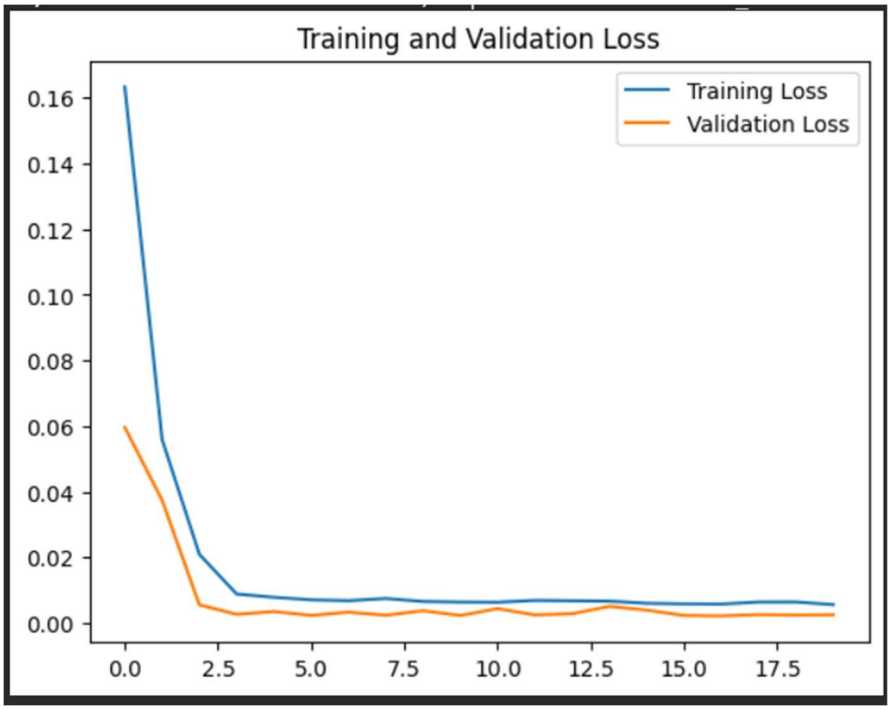
OUTPUT :



Name: Siddhesh Santosh More **Seat No:** 1313160

Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199: Use
  super().__init__(**kwargs)
24/24 ━━━━━━━━━━━━━━━━━━━━ 9s 136ms/step - loss: 0.2421 - val_loss: 0.0596
Epoch 2/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 162ms/step - loss: 0.0639 - val_loss: 0.0374
Epoch 3/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - loss: 0.0298 - val_loss: 0.0055
Epoch 4/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 149ms/step - loss: 0.0106 - val_loss: 0.0027
Epoch 5/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 71ms/step - loss: 0.0076 - val_loss: 0.0035
Epoch 6/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 69ms/step - loss: 0.0069 - val_loss: 0.0023
Epoch 7/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - loss: 0.0065 - val_loss: 0.0033
Epoch 8/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 73ms/step - loss: 0.0073 - val_loss: 0.0024
Epoch 9/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 139ms/step - loss: 0.0063 - val_loss: 0.0037
Epoch 10/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - loss: 0.0066 - val_loss: 0.0023
Epoch 11/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step - loss: 0.0064 - val_loss: 0.0044
Epoch 12/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - loss: 0.0068 - val_loss: 0.0025
Epoch 13/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step - loss: 0.0067 - val_loss: 0.0028
Epoch 14/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 64ms/step - loss: 0.0072 - val_loss: 0.0051
Epoch 15/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 143ms/step - loss: 0.0067 - val_loss: 0.0039
Epoch 16/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 78ms/step - loss: 0.0059 - val_loss: 0.0023
Epoch 17/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - loss: 0.0053 - val_loss: 0.0022
Epoch 18/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 78ms/step - loss: 0.0066 - val_loss: 0.0025
Epoch 19/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 79ms/step - loss: 0.0058 - val_loss: 0.0024
Epoch 20/20
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 99ms/step - loss: 0.0058 - val_loss: 0.0025
```



Name: Siddhesh Santosh More **Seat No:** 1313160
Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

PRACTICAL NO. 8

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

# Download and extract the dataset
!wget -nc https://download.pytorch.org/tutorial/hymenoptera_data.zip -P .
!unzip -n hymenoptera_data.zip -d .

# Mean and standard deviation for normalization
mean = np.array([0.5, 0.5, 0.5])
std = np.array([0.25, 0.25, 0.25])

# Data transformations
```

```
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)
    ]),
}

# Dataset directory
data_dir = "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,
        shuffle=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 configuration
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print("Classes:", class_names)

# Function to show images
def imshow(inp, title=None):
    inp = inp.numpy().transpose((1, 2, 0))
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.axis("off")
    plt.show()

# Display a batch of training images
inputs, classes = next(iter(dataloaders["train"]))
```

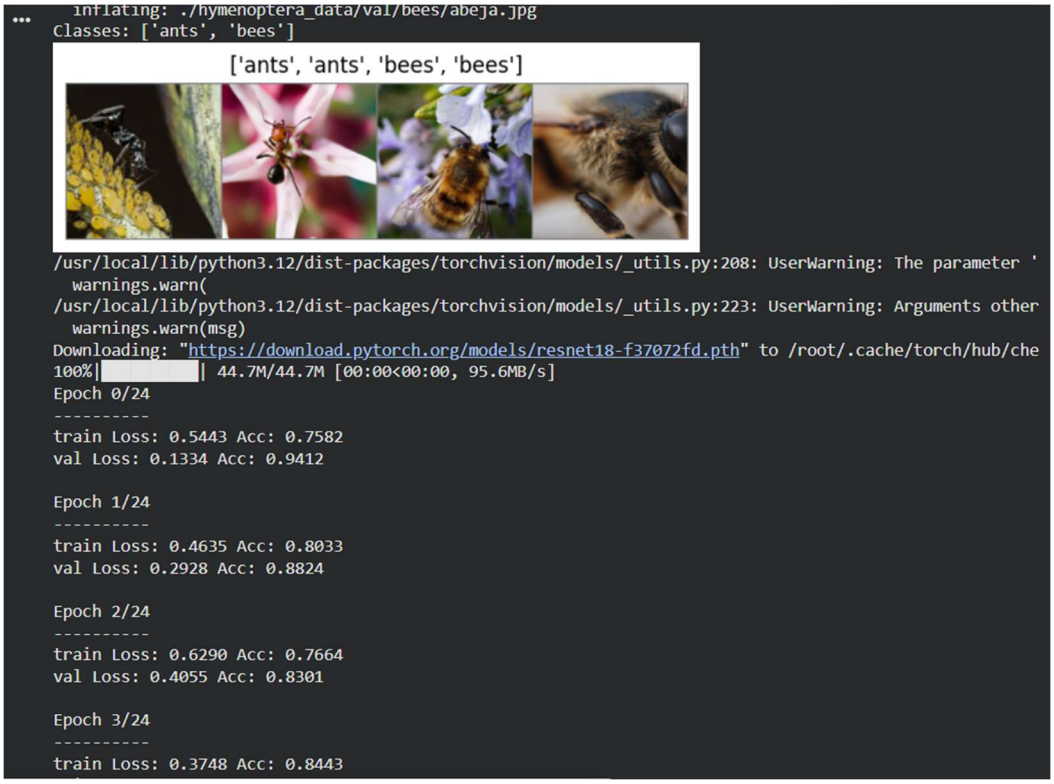
```
out = torchvision.utils.make_grid(inputs)
imshow(out, title=[class_names[x] for x in classes])


# Training function
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(f"Epoch {epoch}/{num_epochs - 1}")
        print("-" * 10)

        for phase in ["train", "val"]:
            if phase == "train":
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

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

                optimizer.zero_grad()
```

```
            with torch.set_grad_enabled(phase == "train"):
                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
                loss = criterion(outputs, labels)

                if phase == "train":
                    loss.backward()
                    optimizer.step()

            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(f"{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}")

        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(f"Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s")
    print(f"Best val Acc: {best_acc:.4f}")

    model.load_state_dict(best_model_wts)
    return model

# ------------------------------
# FINETUNING THE CONVNET
# ------------------------------
model = models.resnet18(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)

model = train_model(model, criterion, optimizer, scheduler, num_epochs=25)

# ------------------------------
# FIXED FEATURE EXTRACTOR
# ------------------------------
model_conv = 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)
scheduler_conv = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)


model_conv = train_model(
    model_conv,
    criterion,
    optimizer_conv,
    scheduler_conv,
    num_epochs=25
)
```

OUTPUT:



Name: Siddhesh Santosh More **Seat No:** 1313160
Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

PRACTICAL NO. 9

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

Step-01 Install required libraries
pip install numpy scikit-learn scikit-optimize

Step-02
Code:

```
# Import necessary libraries
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

from skopt import BayesSearchCV
from skopt.space import Real, Integer

# Load dataset (Iris dataset)
data = load_iris()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Define the hyperparameter search space for RandomForestClassifier
param_space = {
    "n_estimators": Integer(10, 200),          # Number of trees
    "max_depth": Integer(1, 50),               # Maximum depth of trees
    "min_samples_split": Integer(2, 10),       # Minimum samples to split a node
    "min_samples_leaf": Integer(1, 10),        # Minimum samples at leaf node
    "max_features": Real(0.1, 1.0)             # Proportion of features used
}

# Initialize the RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
```

```
# Initialize Bayesian optimizer with cross-validation
opt = BayesSearchCV(
    estimator=rf,
    search_spaces=param_space,
    n_iter=50,
    random_state=42,
    cv=5,
    n_jobs=-1,
    verbose=1
)

# Perform the hyperparameter search
opt.fit(X_train, y_train)

# Print the best hyperparameters found
print("Best Hyperparameters:", opt.best_params_)

# Evaluate the model with the best hyperparameters
best_rf = opt.best_estimator_
y_pred = best_rf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the optimized model: {accuracy:.4f}")
```

OUTPUT:

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

Name: Siddhesh Santosh More **Seat No:** 1313160
Program : MSc.IT Part II (Sem III) **Subject :** Advanced Artificial Intelligence

PRACTICAL NO. 10

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.

Step-01 Install required libraries
pip install transformers torch
Step-02
Code:

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer
import torch
# Load the GPT-2 model and tokenizer
# You can use "gpt2-medium" or "gpt2-large" for better performance
model_name = "gpt2"
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
model = GPT2LMHeadModel.from_pretrained(model_name)
# Move model to CPU (safe default)
model.eval()
def generate_text(prompt, max_length=50, temperature=0.7, top_p=0.9):
    input_ids = tokenizer.encode(prompt, return_tensors="pt")
    # Generate text
    output = model.generate(
        input_ids=input_ids,
        max_length=max_length,
        temperature=temperature,
        top_p=top_p,
        do_sample=True,
        pad_token_id=tokenizer.eos_token_id
    )
    # Decode and return generated text
    return tokenizer.decode(output[0], skip_special_tokens=True)
# Input prompt
prompt = "My college is at"
# Generate text
generated_text = generate_text(prompt, max_length=100)
print("Generated Text:\n")
print(generated_text)
```

OUTPUT:

```
Generated Text:

My college is at a place that has been very supportive of the students, but that has a lot of the problems with the current situation. They don't know how to deal with these issues and they have to deal with the people who are doing these things. I think they're also a little bit confused about how to deal with the issues.

"I think they're a little bit confused about how to deal with the issues. I think they're a little bit confused about how to deal
```