A PRACTICAL REPORT

ON

ADVANCE ARTIFICIAL INTELLIGENCE

SUBMITTED BY

Mr. Malpura Mohd Azharuddin Mohd Rafiq

Submitted in fulfillment of the requirements for qualifying
M.Sc.IT Part-2 Semester-3 Examination 2025-2026

University of Mumbai

Department of Information Technology

University of Mumbai

Institute of Distance and Open Learning (IDOL)

Dr. Shankardayal Sharma bhavan, Vidyanagari, Santacruz(E)

PCP CENTER: RIZVI COLLEGE, BANDRA (W)

Certificate

*This is to certify that **ADVANCE ARTIFICIAL INTELLIGENCE** performed at RIZVI COLLEGE, BANDRA (W) by Mr. **Malpura Mohd Azharuddin Mohd Rafiq** holding Seat No/Application No. 8163020 /6780 studying Masters of Science in Information Technology Semester – 3 has been satisfactorily completed as prescribed by the University of Mumbai, during the year 2025 – 2026.*

Subject In-Charge **Coordinator In-Charge** **External Examiner**

INDEX

SR NO.	Practical List	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.	Applying reinforcement learning algorithms to solve complex decision-making problems.		
8.	Utilizing transfer learning to improve model performance on limited datasets.		
9.	Building a deep learning model for time series forecasting or anomaly detection.		
10.	Implementing a machine learning pipeline for automated feature engineering and model selection.		
11.	Using advanced optimization techniques like evolutionary algorithms or Bayesian optimization for hyperparameter tuning.		
12.	Deploying a machine learning model in a production environment using containerization and cloud services.		

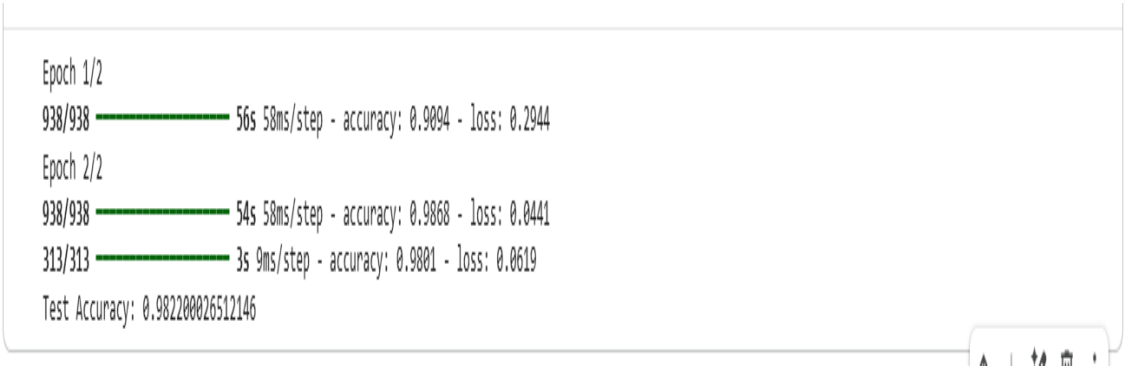
13.	Use Python libraries such as GPT-2 or textgenrnn to train generative models on a corpus of text data and generate new text based on the patterns it has learned.		
14.	Experiment with neural networks like GANs (Generative Adversarial Networks) using Python libraries like TensorFlow or PyTorch to generate new images based on a dataset of images.		

Practical No.1

Aim:Implementing advanced deep learning algorithms such as CNN and RNN using python libraries like tensorflow and pytorch.

A.Code:

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

# Load dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocessing
x_train = x_train.reshape(-1, 28, 28, 1) / 255.0
x_test = x_test.reshape(-1, 28, 28, 1) / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# CNN Model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    Flatten(),v
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

# Train
model.fit(x_train, y_train, epochs=2, batch_size=64)

# Evaluate
loss, accuracy = model.evaluate(x_test, y_test)
print("Test Accuracy:", accuracy)
```



Practical No.1

B.Code:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Create dummy sequence data
X = np.random.rand(1000, 10, 1)   # 1000 samples, 10 time steps
y = np.random.randint(0, 2, 1000) # Binary labels

# RNN Model
model = Sequential([
    LSTM(64, input_shape=(10,1)),
    Dense(1, activation='sigmoid')
])

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

# Train
model.fit(X, y, epochs=5, batch_size=32)

# Test prediction
prediction = model.predict(X[:1])
print("Prediction:", prediction)
```

```
Epoch 1/5
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sec
  super().__init__(**kwargs)
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.4776 - loss: 0.6949
Epoch 2/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5369 - loss: 0.6911
Epoch 3/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4903 - loss: 0.6944
Epoch 4/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5077 - loss: 0.6935
Epoch 5/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5170 - loss: 0.6927
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 181ms/step
Prediction: [[0.52220577]]
```

Practical No.2

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

Code:

```
import pandas as pd
import numpy as np
import nltk
import re

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, classification_report
data = {
    "text": [
        "I love this product",
        "This is the worst experience",
        "Amazing quality and service",
        "I hate this item",
        "Very happy with the purchase",
        "Terrible and disappointing"
    ],
    "label": [1, 0, 1, 0, 1, 0]  # 1 = Positive, 0 = Negative
}

df = pd.DataFrame(data)
def clean_text(text):
    text = text.lower()
    text = re.sub(r"[^a-z\s]", "", text)
    return text

df["clean_text"] = df["text"].apply(clean_text)

X_train, X_test, y_train, y_test = train_test_split(
    df["clean_text"], df["label"], test_size=0.2, random_state=42
)
vectorizer = TfidfVectorizer(max_features=3000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
model = LogisticRegression()
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
def predict_sentiment(text):
    text = clean_text(text)
    vector = vectorizer.transform([text])
    prediction = model.predict(vector)
    return "Positive" if prediction[0] == 1 else "Negative"

print(predict_sentiment("The movie was fantastic"))
print(predict_sentiment("I am very disappointed"))
```

```
••• Accuracy: 0.5
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       0.00      0.00      0.00         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

Positive
Positive
Negative
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels w
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels w
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels w
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Practical No 3

Creating a chatbot using advanced techniques like transformer models.

```
!pip install transformers torch
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

# Load model and tokenizer
model_name = "microsoft/DialoGPT-medium"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

chat_history_ids = None

print("Chatbot is ready! Type 'quit' to exit.\n")

while True:
    user_input = input("You: ")
    if user_input.lower() == "quit":
        break

    # Encode user input
    new_input_ids = tokenizer.encode(
        user_input + tokenizer.eos_token, return_tensors="pt"
    )

    # Append to conversation history
    bot_input_ids = (
        torch.cat([chat_history_ids, new_input_ids], dim=-1)
        if chat_history_ids is not None
        else new_input_ids
    )

    # Generate response
    chat_history_ids = model.generate(
        bot_input_ids,
        max_length=1000,
        pad_token_id=tokenizer.eos_token_id
    )

    # Decode response
    response = tokenizer.decode(
        chat_history_ids[:, bot_input_ids.shape[-1]:][0],
        skip_special_tokens=True
    )

    print("Bot:", response)
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages (2.9.
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from transformers)
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from transformer
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from transfo
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from transformer
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from trans
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from transformers)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages (
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (from tran
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-packages (from transformers
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.12/dist-packages (fr
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch) (75.2
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch) (1
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec>=0.8.5 in /usr/local/lib/python3.12/dist-packages (from torch) (2
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from hu
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from symp
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2-
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requ
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requ
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:104: UserWarning:
Error while fetching `HF_TOKEN` secret value from your vault: 'Requesting secret HF_TOKEN timed out. Se
```

```
tokenizer_config.json:   0%|          | 0.00/614 [00:00<?, ?B/s]
vocab.json: 0.00B [00:00, ?B/s]
merges.txt: 0.00B [00:00, ?B/s]
config.json:   0%|          | 0.00/642 [00:00<?, ?B/s]
WARNING:torchao.kernel.intmm:Warning: Detected no triton, on systems without Triton certain kernels wil
pytorch_model.bin:   0%|          | 0.00/863M [00:00<?, ?B/s]
model.safetensors:   0%|          | 0.00/863M [00:00<?, ?B/s]
generation_config.json:   0%|          | 0.00/124 [00:00<?, ?B/s]
Chatbot is ready! Type 'quit' to exit.

The attention mask is not set and cannot be inferred from input because pad token is same as eos token.
Bot: Hey there!
You: how are you
Bot: I'm good, how are you?
```

Practical No.4

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

A.Code:

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity

data = pd.read_csv("/rating.csv")

user_item_matrix = data.pivot_table(
    index='user_id',
    columns='movie_id',
    values='rating'
).fillna(0)

user_similarity = cosine_similarity(user_item_matrix)

similarity_df = pd.DataFrame(
    user_similarity,
    index=user_item_matrix.index,
    columns=user_item_matrix.index
)

def recommend(user_id, top_n=3):
    similar_users =
similarity_df[user_id].sort_values(ascending=False)[1:]
    scores =
user_item_matrix.loc[similar_users.index].T.dot(similar_users)
    return scores.sort_values(ascending=False).head(top_n)

print("Recommended movies for User 1:")
print(recommend(1))
```

Output:-

```
Recommended movies for User 1:
movie_id
104    5.270463
101    3.457424
103    3.162278
dtype: float64
```

Practical No.4

B.Code:

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Embedding, Flatten, Dense,
Concatenate
from tensorflow.keras.models import Model

num_users = 5
num_items = 10
embedding_size = 8

user_input = Input(shape=(1,))
item_input = Input(shape=(1,))

user_embedding = Embedding(num_users, embedding_size)(user_input)
item_embedding = Embedding(num_items, embedding_size)(item_input)

user_vec = Flatten()(user_embedding)
item_vec = Flatten()(item_embedding)

concat = Concatenate()([user_vec, item_vec])

dense = Dense(64, activation='relu')(concat)
dense = Dense(32, activation='relu')(dense)

output = Dense(1)(dense)

model = Model([user_input, item_input], output)
model.compile(optimizer='adam', loss='mse')

model.summary()
```

Model: "functional_3"

Layer (type)	Output Shape	Param #	Connected to
input_layer_3 (InputLayer)	(None, 1)	0	-
input_layer_4 (InputLayer)	(None, 1)	0	-
embedding (Embedding)	(None, 1, 8)	40	input_layer_3[0]…
embedding_1 (Embedding)	(None, 1, 8)	80	input_layer_4[0]…
flatten_2 (Flatten)	(None, 8)	0	embedding[0][0]
flatten_3 (Flatten)	(None, 8)	0	embedding_1[0][0]
concatenate (Concatenate)	(None, 16)	0	flatten_2[0][0], flatten_3[0][0]
dense_5 (Dense)	(None, 64)	1,088	concatenate[0][0]
dense_6 (Dense)	(None, 32)	2,080	dense_5[0][0]
dense_7 (Dense)	(None, 1)	33	dense_6[0][0]

Total params: 3,321 (12.97 KB)
Trainable params: 3,321 (12.97 KB)
Non-trainable params: 0 (0.00 B)

Practical No 5

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

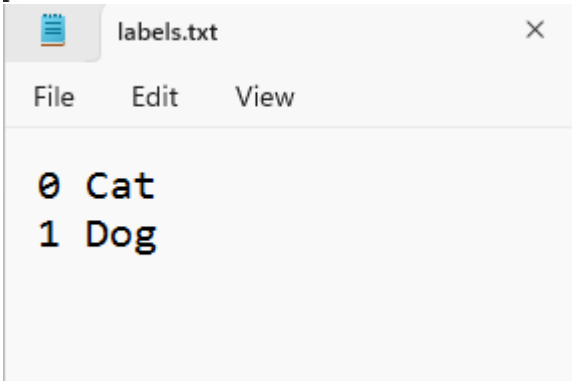
Code:-

```
import tensorflow as tf

import numpy as np

import cv2

from PIL import Image


# ==========================

# CONFIGURATION

# ==========================

MODEL_PATH = "/content/model_unquant.tflite"

LABELS_PATH = "/content/labels.txt"

IMAGE_PATH = "/content/c6.jpg"   # <-- change this


CONFIDENCE_THRESHOLD = 0.5


# ==========================

# LOAD LABELS

# ==========================

with open(LABELS_PATH, "r") as f:

    labels = [line.strip() for line in f.readlines()]


# ==========================

# LOAD TFLITE MODEL

# ==========================

interpreter = tf.lite.Interpreter(model_path=MODEL_PATH)

interpreter.allocate_tensors()


input_details = interpreter.get_input_details()

output_details = interpreter.get_output_details()
```

```
input_height = input_details[0]['shape'][1]
input_width = input_details[0]['shape'][2]

# ==========================
# LOAD & PREPROCESS IMAGE
# ==========================
image = Image.open(IMAGE_PATH).convert("RGB")
original_width, original_height = image.size

image_resized = image.resize((input_width, input_height))
input_data = np.expand_dims(image_resized, axis=0)

# Normalize if model expects float
if input_details[0]['dtype'] == np.float32:
    input_data = np.float32(input_data) / 255.0

# ==========================
# RUN INFERENCE
# ==========================
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()

# ==========================
# GET OUTPUTS
# ==========================
# For classification models, the output_details typically has one
element
# representing the probabilities for each class.
predictions = interpreter.get_tensor(output_details[0]['index'])[0] #
Get the single output array

# Get the class with the highest probability
```

```
class_id = np.argmax(predictions)
confidence = predictions[class_id]

# ==========================
# DRAW DETECTIONS (modified for classification)
# ==========================
image_np = np.array(image)

if confidence >= CONFIDENCE_THRESHOLD:
    label = labels[class_id]

    # Display the predicted label and confidence. Adjust position as
needed.
    cv2.putText(
        image_np,
        f"Class: {label} ({confidence:.2f})",
        (10, 30), # Position for text
        cv2.FONT_HERSHEY_SIMPLEX,
        0.8,
        (0, 255, 0),
        2
    )

# ==========================
# SHOW RESULT
# ==========================
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 8))
plt.imshow(image_np)
plt.axis("off")
plt.show()
```

Requirement:-



Practical No 6

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

Code:-

Importing Packages

```
import os
import time
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
import argparse
from IPython import display
import matplotlib.pyplot as plt
# %matplotlib inline
from tensorflow import keras
```

#we construct the argument parser

```
parser = argparse.ArgumentParser()
parser.add_argument("--n_epochs", type=int, default=200, help="number
of epochs of training")
parser.add_argument("--batch_size", type=int, default=128, help="size
of the batches")
parser.add_argument("--lr", type=float, default=2e-4, help="adam:
learning rate")
parser.add_argument("--b1", type=float, default=0.5, help="adam: decay
of first order momentum of gradient")
parser.add_argument("--b2", type=float, default=0.999, help="adam:
decay of first order momentum of gradient")
parser.add_argument("--latent_dim", type=int, default=100,
help="dimension of the latent space (generator's input)")
parser.add_argument("--image_dim", type=int, default=784, help="image
size")
parser.add_argument("--channels", type=int, default=1, help="image
channels")
args = parser.parse_args("")
```

Then load the data and perform Data preprocessing

```
(x_train, y_train), (x_test, y_test) =
tf.keras.datasets.fashion_mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], 28, 28,
1).astype('float32')
x_train = (x_train - 127.5) / 127.5 # Normalize the images to [-1, 1]
# Batch and shuffle the data
train_dataset = tf.data.Dataset.from_tensor_slices(x_train).\
shuffle(60000).batch(args.batch_size)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-
32768/29515 [==============================] - 0s 0us/step
40960/29515 [==========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-i
16384/5148 [================================================================================
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-i
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
```

Creating Generator Network

```
def generator(image_dim):
  inputs = keras.Input(shape=(100,), name='input_layer')
  x = layers.Dense(128,
kernel_initializer=tf.keras.initializers.he_uniform,
name='dense_1')(inputs)
  #print(x.dtype)
  x = layers.LeakyReLU(0.2, name='leaky_relu_1')(x)
  x = layers.Dense(256,
kernel_initializer=tf.keras.initializers.he_uniform, name='dense_2')(x)
  x = layers.BatchNormalization(momentum=0.1,  epsilon=0.8,
name='bn_1')(x)
  x = layers.LeakyReLU(0.2, name='leaky_relu_2')(x)
  x = layers.Dense(512,
kernel_initializer=tf.keras.initializers.he_uniform, name='dense_3')(x)
  x = layers.BatchNormalization(momentum=0.1,  epsilon=0.8,
name='bn_2')(x)
  x = layers.LeakyReLU(0.2, name='leaky_relu_3')(x)
  x = layers.Dense(1024,
kernel_initializer=tf.keras.initializers.he_uniform,  name='dense_4')(x
)
  x = layers.BatchNormalization(momentum=0.1,  epsilon=0.8,
name='bn_3')(x)
  x = layers.LeakyReLU(0.2, name='leaky_relu_4')(x)
  x = layers.Dense(image_dim,
kernel_initializer=tf.keras.initializers.he_uniform,
activation='tanh',  name='dense_5')(x)
  outputs = tf.reshape(x, [-1, 28, 28, 1], name='Reshape_Layer')
  model = tf.keras.Model(inputs, outputs, name="Generator")
  return model
generator = generator(args.image_dim)
generator.summary()
```

```
Model: "Generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_layer (InputLayer)     [(None, 100)]             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               12928
_________________________________________________________________
leaky_relu_1 (LeakyReLU)     (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               33024
_________________________________________________________________
bn_1 (BatchNormalization)    (None, 256)               1024
_________________________________________________________________
leaky_relu_2 (LeakyReLU)     (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               131584
_________________________________________________________________
bn_2 (BatchNormalization)    (None, 512)               2048
_________________________________________________________________
leaky_relu_3 (LeakyReLU)     (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              525312
_________________________________________________________________
bn 3 (BatchNormalization)    (None, 1024)              4096
```

```
leaky_relu_3 (LeakyReLU)     (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              525312
_________________________________________________________________
bn_3 (BatchNormalization)    (None, 1024)              4096
_________________________________________________________________
leaky_relu_4 (LeakyReLU)     (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 784)               803600
_________________________________________________________________
tf.reshape (TFOpLambda)      (None, 28, 28, 1)         0
=================================================================
Total params: 1,513,616
Trainable params: 1,510,032
Non-trainable params: 3,584
```

Creating Discriminator Network

```
def discriminator():
  inputs = keras.Input(shape=(28,28,1), name='input_layer')
  input = tf.reshape(inputs, [-1, 784], name='reshape_layer')
  x = layers.Dense(512,
kernel_initializer=tf.keras.initializers.he_uniform,
name='dense_1')(input)
  x = layers.LeakyReLU(0.2, name='leaky_relu_1')(x)
  x = layers.Dense(256,
kernel_initializer=tf.keras.initializers.he_uniform, name='dense_2')(x)
```

```
  x = layers.LeakyReLU(0.2, name='leaky_relu_2')(x)
  outputs = layers.Dense(1,
kernel_initializer=tf.keras.initializers.he_uniform,
activation='sigmoid', name='dense_3') (x)
  model = tf.keras.Model(inputs, outputs, name="Discriminator")
  return model
discriminator = discriminator()
discriminator.summary()
```

```
Model: "Discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_layer (InputLayer)     [(None, 28, 28, 1)]       0
_________________________________________________________________
tf.reshape_1 (TFOpLambda)    (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
leaky_relu_1 (LeakyReLU)     (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_relu_2 (LeakyReLU)     (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 257
=================================================================
Total params: 533,505
Trainable params: 533,505
Non-trainable params: 0
_________________________________________________________________
```

Loss Function

```
binary_cross_entropy = tf.keras.losses.BinaryCrossentropy()
```

Generator Loss

```
def generator_loss(fake_output):
  gen_loss = binary_cross_entropy(tf.ones_like(fake_output),
fake_output)
  #print(gen_loss)
  return gen_loss
```

Discriminator Loss

```
def discriminator_loss(real_output, fake_output):
  real_loss = binary_cross_entropy(tf.ones_like(real_output),
real_output)
  fake_loss = binary_cross_entropy(tf.zeros_like(fake_output),
fake_output)
  total_loss = real_loss + fake_loss
  #print(total_loss)
  return total_loss
```

Optimizers

```
generator_optimizer = tf.keras.optimizers.Adam(learning_rate = args.lr,
beta_1 = args.b1, beta_2 = args.b2 )
discriminator_optimizer = tf.keras.optimizers.Adam(learning_rate =
args.lr, beta_1 = args.b1, beta_2 = args.b2 )
```

Training Loop (all the functions combined for training GAN)

```
@tf.function
def train_step(images):
  noise = tf.random.normal([args.batch_size, args.latent_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_gen = gen_tape.gradient(gen_loss,
generator.trainable_variables) # computing the gradients


  gradients_of_disc = disc_tape.gradient(disc_loss,
discriminator.trainable_variables) # computing the gradients


  generator_optimizer.apply_gradients(zip(gradients_of_gen,
generator.trainable_variables)) # updating generator parameter


  discriminator_optimizer.apply_gradients(zip(gradients_of_disc,discrim
inator.trainable_variables)) # updating discriminator parameter
```

Final training

```
# We will reuse this seed overtime to visualize progress
num_examples_to_generate = 25
seed = tf.random.normal([num_examples_to_generate, args.latent_dim])

!mkdir tensor
import os
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint =
tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                 discriminator_optimizer=discriminator_
optimizer,
                                 generator=generator,
```

```
                                 discriminator=discriminator)
def train(dataset, epochs):
  for epoch in range(epochs):
    start = time.time()
    i = 0
    D_loss_list, G_loss_list = [], []
    for image_batch in dataset:
      i += 1
      train_step(image_batch)

    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)
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)
    #print(predictions.shape)
    fig = plt.figure(figsize=(4,4))

    for i in range(predictions.shape[0]):
        plt.subplot(5, 5, i+1)
        pred = (predictions[i, :, :, 0] + 1) * 127.5
        pred = np.array(pred)
        plt.imshow(pred.astype(np.uint8), cmap='gray')
        plt.axis('off')

    plt.savefig('tensor/image_at_epoch_{:d}.png'.format(epoch))
    plt.show()
train(train_dataset, args.n_epochs)
```

•••

Practical No.7

Aim:Applying Reinforcement learning algorithms to solve complex decision making problem.

Code:

```
import numpy as np
# import gym # The 'gym' library is outdated and has API changes with
newer NumPy versions.
import gymnasium as gym # Use 'gymnasium' as the maintained
replacement.

# Create environment
# env = gym.make('CartPole-v1') # Old gym API
env = gym.make('CartPole-v1', render_mode='rgb_array') # Use gymnasium.
'render_mode' can be 'human' for display.

# Q-learning parameters
alpha = 0.1
gamma = 0.9
epsilon = 0.1

# --- State Discretization for CartPole-v1 ---
# CartPole-v1 has a continuous observation space. For Q-learning, this
needs to be discretized.
# We define bins for each of the 4 observation dimensions.

# Number of discrete bins for each observation dimension
# (Cart Position, Cart Velocity, Pole Angle, Pole Angular Velocity)
num_bins_per_dimension = (10, 10, 10, 10)

# Define the ranges for each observation dimension.
# Using env.observation_space.low and env.observation_space.high
provides the official bounds.
# For velocity, which has -Inf to Inf, we'll use reasonable practical
bounds.
pos_space = np.linspace(env.observation_space.low[0],
env.observation_space.high[0], num_bins_per_dimension[0] + 1)[1:-1]
vel_space = np.linspace(-3, 3, num_bins_per_dimension[1] + 1)[1:-1] #
Approximated velocity bounds, can be tuned
angle_space = np.linspace(env.observation_space.low[2],
env.observation_space.high[2], num_bins_per_dimension[2] + 1)[1:-1]
angle_vel_space = np.linspace(-3, 3, num_bins_per_dimension[3] + 1)[1:-
1] # Approximated angular velocity bounds, can be tuned

# Q-table initialization
# The size of the Q-table will be (num_bins_pos, num_bins_vel,
num_bins_angle, num_bins_angle_vel, num_actions)
q_table = np.zeros((*num_bins_per_dimension, env.action_space.n))

# Helper function to discretize the continuous state
def discretize_state(state_array):
    pos_idx = np.digitize(state_array[0], pos_space)
    vel_idx = np.digitize(state_array[1], vel_space)
    angle_idx = np.digitize(state_array[2], angle_space)
```

```
    angle_vel_idx = np.digitize(state_array[3], angle_vel_space)
    return (pos_idx, vel_idx, angle_idx, angle_vel_idx) # Return a
tuple of integers


# Q-learning loop
for episode in range(1000):
    # Initialize state using gymnasium's new reset API
    raw_state, _ = env.reset()
    state = discretize_state(raw_state) # Discretize the initial state

    done = False
    rewards = 0.0

    while not done:
        # Choose action using epsilon-greedy policy
        if np.random.rand() < epsilon:
            action = env.action_space.sample()
        else:
  # Use the discretized state tuple to index the q_table
            action = np.argmax(q_table[state])

   # Take action and observe next state and reward using gymnasium's
new step API
        raw_next_state, reward, terminated, truncated, _ =
env.step(action)
        next_state = discretize_state(raw_next_state) # Discretize the
next state
        done = terminated or truncated # 'done' is true if the episode
is terminated OR truncated

        # Update Q-table
        # Use the discretized state and next_state tuples for indexing
        q_table[state + (action,)] += alpha * (reward + gamma *
np.max(q_table[next_state]) - q_table[state + (action,)])

        # Update state and rewards
        state = next_state
        rewards += reward

    print(f'Episode {episode+1}, Reward: {rewards}')

env.close() # It's good practice to close the environment when done.
```

```
Episode 1, Reward: 12.0
Episode 2, Reward: 10.0
Episode 3, Reward: 10.0
Episode 4, Reward: 11.0
Episode 5, Reward: 11.0
Episode 6, Reward: 9.0
Episode 7, Reward: 11.0
Episode 8, Reward: 11.0
Episode 9, Reward: 11.0
Episode 10, Reward: 11.0
Episode 11, Reward: 10.0
Episode 12, Reward: 10.0
Episode 13, Reward: 8.0
Episode 14, Reward: 10.0
Episode 15, Reward: 10.0
Episode 16, Reward: 12.0
Episode 17, Reward: 10.0
Episode 18, Reward: 8.0
Episode 19, Reward: 12.0
Episode 20, Reward: 9.0
Episode 21, Reward: 12.0
Episode 22, Reward: 9.0
Episode 23, Reward: 11.0
Episode 24, Reward: 12.0
Episode 25, Reward: 11.0
Episode 26, Reward: 10.0
Episode 27, Reward: 10.0
Episode 28, Reward: 10.0
Episode 29, Reward: 12.0
Episode 30, Reward: 11.0
Episode 31, Reward: 11.0
Episode 32, Reward: 10.0
Episode 33, Reward: 9.0
Episode 34, Reward: 10.0
Episode 35, Reward: 9.0
Episode 36, Reward: 11.0
Episode 37, Reward: 11.0
Episode 38, Reward: 10.0
Episode 39, Reward: 10.0
Episode 40, Reward: 11.0
```

Practical No 8

AIM: Utilizing transfer learning to improve model performance on limited datasets.

```
!pip install tensorflow numpy matplotlib

# Load and preprocess data
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMG_SIZE = (224, 224)
BATCH_SIZE = 32

datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)

train_data = datagen.flow_from_directory(
    "data/",
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    subset="training"
)

val_data = datagen.flow_from_directory(
    "data/",
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    subset="validation"
)

# Load pretrained model (feature extractor)
base_model = tf.keras.applications.ResNet50(
    weights="imagenet",
    include_top=False,
    input_shape=(224, 224, 3)
)

base_model.trainable = False  # freeze pretrained layers

# Build transfer learning model
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(train_data.num_classes, activation="softmax")
])

model.compile(
    optimizer="adam",
```

```
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

model.summary()

# Train the model
history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=10
)

# Fine-tuning (optional but powerful)
# Unfreeze top layers for fine-tuning
base_model.trainable = True

for layer in base_model.layers[:-30]:
    layer.trainable = False

model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-5),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

model.fit(
    train_data,
    validation_data=val_data,
    epochs=5
)
```

Output:-

```
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: markupsafe>=2.1.1 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/d
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-package
Found 8 images belonging to 2 classes.
Found 2 images belonging to 2 classes.
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 2048)	0
dense_2 (Dense)	(None, 128)	262,272

```
Found 2 images belonging to 2 classes.
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 2048)	0
dense_2 (Dense)	(None, 128)	262,272
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 2)	258

Total params: 23,850,242 (90.98 MB)
Trainable params: 262,530 (1.00 MB)
Non-trainable params: 23,587,712 (89.98 MB)

```
Epoch 1/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 13s 13s/step - accuracy: 0.5000 - loss: 0.9024 - val_accuracy: 0.5000 - val_loss: 0.8643
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.6250 - loss: 0.9689 - val_accuracy: 0.5000 - val_loss: 0.9235
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.3750 - loss: 0.9287 - val_accuracy: 0.5000 - val_loss: 0.8548
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.6250 - loss: 0.7848 - val_accuracy: 0.5000 - val_loss: 0.7224
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.7500 - loss: 0.6417 - val_accuracy: 0.5000 - val_loss: 0.6947
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 0.6250 - loss: 0.6556 - val_accuracy: 0.5000 - val_loss: 0.7255
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 0.7500 - loss: 0.6915 - val_accuracy: 0.5000 - val_loss: 0.7517
Epoch 8/10
```

Practical No 9

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

```
# Imports & Setup
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
# Prepare Time Series Data
# Synthetic time series data
np.random.seed(42)
t = np.arange(0, 500)
data = np.sin(0.04 * t) + np.random.normal(0, 0.1, len(t))

# Scale data
scaler = MinMaxScaler()
data = scaler.fit_transform(data.reshape(-1, 1))

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

SEQ_LEN = 30
X, y = create_sequences(data, SEQ_LEN)

# Train-test split
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

#LSTM Forecasting Model
model = Sequential([
    LSTM(64, return_sequences=True, input_shape=(SEQ_LEN, 1)),
    LSTM(32),
    Dense(1)
])

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

model.fit(
    X_train,
    y_train,
    epochs=20,
    batch_size=32,
    validation_split=0.1
)

# Make Forecasts
```

```
predictions = model.predict(X_test)
predictions = scaler.inverse_transform(predictions)
y_test = scaler.inverse_transform(y_test)

print("Sample predictions:", predictions[:5].flatten())
```

Output:-

```
Epoch 1/20
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarn
  super().__init__(**kwargs)
11/11 ━━━━━━━━━━━━━━━━━━━━ 4s 72ms/step - loss: 0.1653 - val_loss: 0.0746
Epoch 2/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0223 - val_loss: 0.0076
Epoch 3/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - loss: 0.0170 - val_loss: 0.0291
Epoch 4/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 0.0136 - val_loss: 0.0087
Epoch 5/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - loss: 0.0109 - val_loss: 0.0158
Epoch 6/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 53ms/step - loss: 0.0084 - val_loss: 0.0083
Epoch 7/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - loss: 0.0072 - val_loss: 0.0087
Epoch 8/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0057 - val_loss: 0.0054
Epoch 9/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - loss: 0.0035 - val_loss: 0.0024
Epoch 10/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - loss: 0.0022 - val_loss: 0.0027
Epoch 11/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - loss: 0.0023 - val_loss: 0.0025
Epoch 12/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0019 - val_loss: 0.0024
Epoch 13/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.0019 - val_loss: 0.0033
Epoch 14/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0019 - val_loss: 0.0023
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0019 - val_loss: 0.0024
Epoch 13/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.0019 - val_loss: 0.0033
Epoch 14/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0019 - val_loss: 0.0023
Epoch 15/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.0018 - val_loss: 0.0027
Epoch 16/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - loss: 0.0019 - val_loss: 0.0026
Epoch 17/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0018 - val_loss: 0.0025
Epoch 18/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0017 - val_loss: 0.0028
Epoch 19/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 0.0022 - val_loss: 0.0029
Epoch 20/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0021 - val_loss: 0.0024
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 157ms/step
Sample predictions: [-0.5356201  -0.5732988  -0.6091801  -0.63954824 -0.6677947 ]
```

Practical NO 10

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

Code:-

```
pip install scikit-learn featuretools optuna pandas
```

```
Requirement already satisfied: sqlalchemy>=1.4.2 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: PyYAML in /usr/local/lib/python3.12/dist-packages (from optu
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: Mako in /usr/local/lib/python3.12/dist-packages (from alembi
Requirement already satisfied: typing-extensions>=4.12 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from py
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: importlib-resources>=5.10.0 in /usr/local/lib/python3.12/dis
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.12/dist-packages
Downloading featuretools-1.31.0-py3-none-any.whl (587 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 587.9/587.9 kB 6.2 MB/s eta 0:00:00
Downloading optuna-4.6.0-py3-none-any.whl (404 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 404.7/404.7 kB 9.4 MB/s eta 0:00:00
Downloading woodwork-0.31.0-py3-none-any.whl (215 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 215.2/215.2 kB 11.7 MB/s eta 0:00:00
Downloading colorlog-6.10.1-py3-none-any.whl (11 kB)
Installing collected packages: colorlog, woodwork, optuna, featuretools
Successfully installed colorlog-6.10.1 featuretools-1.31.0 optuna-4.6.0 woodwork-0.31.0
```

```
import pandas as pd
import optuna
import featuretools as ft

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier

# ------------------------------
# Load dataset
# ------------------------------
X, y = load_breast_cancer(return_X_y=True, as_frame=True)
X["target"] = y

# ------------------------------
# Automated Feature Engineering
# ------------------------------
es = ft.EntitySet(id="data")
es = es.add_dataframe(
    dataframe_name="features",
    dataframe=X,
    index="index",
    make_index=True
)
```

```
feature_matrix, feature_defs = ft.dfs(
    entityset=es,
    target_dataframe_name="features",
    max_depth=2
)

X_fe = feature_matrix.drop(columns=["target"])
y = feature_matrix["target"]

# ----------------------------
# Objective function for Optuna
# ----------------------------
def objective(trial):
    model_name = trial.suggest_categorical(
        "model", ["logistic", "random_forest", "gradient_boost"]
    )

    if model_name == "logistic":
        model = LogisticRegression(
            C=trial.suggest_float("C", 1e-3, 10, log=True),
            max_iter=1000
        )

    elif model_name == "random_forest":
        model = RandomForestClassifier(
            n_estimators=trial.suggest_int("n_estimators", 50, 300),
            max_depth=trial.suggest_int("max_depth", 2, 20)
        )

    else:
        model = GradientBoostingClassifier(
            n_estimators=trial.suggest_int("n_estimators", 50, 300),
            learning_rate=trial.suggest_float("learning_rate", 0.01,
0.3)
        )

    pipeline = Pipeline([
        ("scaler", StandardScaler()),
        ("model", model)
    ])

    score = cross_val_score(
        pipeline,
        X_fe,
        y,
        cv=5,
        scoring="accuracy"
    ).mean()

    return score

# ----------------------------
# Run Automated Model Selection
# ----------------------------
```

```
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=40)

# ------------------------------
# Results
# ------------------------------
print("Best Accuracy:", study.best_value)
print("Best Pipeline Configuration:")
print(study.best_params)
```

```
[I 2026-01-04 02:55:22,187] A new study created in memory with name: no-name-47298873
[I 2026-01-04 02:55:34,763] Trial 0 finished with value: 0.9596180717279925 and parameters: {'model': 'gradi
[I 2026-01-04 02:55:39,726] Trial 1 finished with value: 0.9648657040832168 and parameters: {'model': 'randc
[I 2026-01-04 02:55:47,619] Trial 2 finished with value: 0.9666356155876418 and parameters: {'model': 'gradi
[I 2026-01-04 02:55:57,589] Trial 3 finished with value: 0.9578481602235678 and parameters: {'model': 'gradi
[I 2026-01-04 02:55:59,158] Trial 4 finished with value: 0.9631268436578171 and parameters: {'model': 'randc
[I 2026-01-04 02:55:59,243] Trial 5 finished with value: 0.9560782487191428 and parameters: {'model': 'logis
[I 2026-01-04 02:56:02,311] Trial 6 finished with value: 0.9560937742586555 and parameters: {'model': 'randc
[I 2026-01-04 02:56:11,784] Trial 7 finished with value: 0.9648657040832169 and parameters: {'model': 'gradi
[I 2026-01-04 02:56:18,381] Trial 8 finished with value: 0.9560937742586555 and parameters: {'model': 'gradi
[I 2026-01-04 02:56:21,887] Trial 9 finished with value: 0.9648657040832168 and parameters: {'model': 'randc
[I 2026-01-04 02:56:21,984] Trial 10 finished with value: 0.9701599130569788 and parameters: {'model': 'logi
[I 2026-01-04 02:56:22,079] Trial 11 finished with value: 0.9701599130569788 and parameters: {'model': 'logi
[I 2026-01-04 02:56:22,179] Trial 12 finished with value: 0.9701599130569788 and parameters: {'model': 'logi
[I 2026-01-04 02:56:22,283] Trial 13 finished with value: 0.9701599130569788 and parameters: {'model': 'logi
[I 2026-01-04 02:56:22,368] Trial 14 finished with value: 0.9806862288464524 and parameters: {'model': 'logi
[I 2026-01-04 02:56:22,452] Trial 15 finished with value: 0.9771774569166277 and parameters: {'model': 'logi
[I 2026-01-04 02:56:22,536] Trial 16 finished with value: 0.9771774569166277 and parameters: {'model': 'logi
[I 2026-01-04 02:56:22,612] Trial 17 finished with value: 0.9771619313771154 and parameters: {'model': 'logi
[I 2026-01-04 02:56:22,694] Trial 18 finished with value: 0.9806862288464524 and parameters: {'model': 'logi
[I 2026-01-04 02:56:22,770] Trial 19 finished with value: 0.9279925477410339 and parameters: {'model': 'logi
[I 2026-01-04 02:56:22,892] Trial 20 finished with value: 0.9771774569166279 and parameters: {'model': 'logi
[I 2026-01-04 02:56:22,979] Trial 21 finished with value: 0.9806862288464524 and parameters: {'model': 'logi
[I 2026-01-04 02:56:23,061] Trial 22 finished with value: 0.9771774569166279 and parameters: {'model': 'logi
[I 2026-01-04 02:56:23,148] Trial 23 finished with value: 0.9806862288464524 and parameters: {'model': 'logi
[I 2026-01-04 02:56:23,224] Trial 24 finished with value: 0.9718987734823784 and parameters: {'model': 'logi
[I 2026-01-04 02:56:23,311] Trial 25 finished with value: 0.9806862288464524 and parameters: {'model': 'logi
[I 2026-01-04 02:56:23,548] Trial 28 finished with value: 0.9771774569166277 and para
[I 2026-01-04 02:56:24,325] Trial 29 finished with value: 0.9560627231796305 and parameters: {'model':
[I 2026-01-04 02:56:26,764] Trial 30 finished with value: 0.9613724576929048 and parameters: {'model':
[I 2026-01-04 02:56:26,980] Trial 31 finished with value: 0.9806862288464524 and parameters: {'model':
[I 2026-01-04 02:56:27,152] Trial 32 finished with value: 0.9771774569166279 and parameters: {'model':
[I 2026-01-04 02:56:27,246] Trial 33 finished with value: 0.9771619313771154 and parameters: {'model':
[I 2026-01-04 02:56:27,364] Trial 34 finished with value: 0.9806862288464524 and parameters: {'model':
[I 2026-01-04 02:56:29,740] Trial 35 finished with value: 0.9525694767893184 and parameters: {'model':
[I 2026-01-04 02:56:33,643] Trial 36 finished with value: 0.9613569321533924 and parameters: {'model':
[I 2026-01-04 02:56:33,724] Trial 37 finished with value: 0.9771774569166279 and parameters: {'model':
[I 2026-01-04 02:56:33,808] Trial 38 finished with value: 0.9771774569166279 and parameters: {'model':
[I 2026-01-04 02:56:36,111] Trial 39 finished with value: 0.9631113181183044 and parameters: {'model':
Best Accuracy: 0.9806862288464524
Best Pipeline Configuration:
{'model': 'logistic', 'C': 0.6587789085958529}
```

Practical No:11

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

Code:

```
pip install optuna scikit-learn
```

```
Collecting optuna
  Downloading optuna-4.6.0-py3-none-any.whl.metadata (17 kB)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: alembic>=1.5.0 in /usr/local/lib/python3.12/dist-packa
Collecting colorlog (from optuna)
  Downloading colorlog-6.10.1-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: sqlalchemy>=1.4.2 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: PyYAML in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist
Requirement already satisfied: Mako in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: typing-extensions>=4.12 in /usr/local/lib/python3.12/d
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.12/dist-pa
Downloading optuna-4.6.0-py3-none-any.whl (404 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 404.7/404.7 kB 4.4 MB/s eta 0:00:00
Downloading colorlog-6.10.1-py3-none-any.whl (11 kB)
Installing collected packages: colorlog, optuna
```

```
import optuna
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

# Load dataset
X, y = datasets.load_iris(return_X_y=True)

# Objective function for Bayesian Optimization
def objective(trial):
    C = trial.suggest_float("C", 1e-3, 1e3, log=True)
    gamma = trial.suggest_float("gamma", 1e-4, 1e1, log=True)
    kernel = trial.suggest_categorical("kernel", ["rbf", "poly",
"sigmoid"])

    model = SVC(
        C=C,
        gamma=gamma,
        kernel=kernel
    )

    score = cross_val_score(model, X, y, cv=5,
scoring="accuracy").mean()
    return score

# Create study
study = optuna.create_study(direction="maximize")
```

```
# Run Bayesian Optimization
study.optimize(objective, n_trials=30)

# Best results
print("Best Accuracy:", study.best_value)
print("Best Hyperparameters:")
print(study.best_params)
```

```
[I 2026-01-04 02:52:41,271] A new study created in memory with name: no-name-4aceb52f-0c6f-4056-bc14-5dbc684
[I 2026-01-04 02:52:41,302] Trial 0 finished with value: 0.9333333333333333 and parameters: {'C': 76.5647982
[I 2026-01-04 02:52:41,997] Trial 1 finished with value: 0.9400000000000001 and parameters: {'C': 777.207368
[I 2026-01-04 02:52:42,027] Trial 2 finished with value: 0.9133333333333334 and parameters: {'C': 0.0040104
[I 2026-01-04 02:52:42,055] Trial 3 finished with value: 0.9133333333333334 and parameters: {'C': 2.8795110
[I 2026-01-04 02:52:42,085] Trial 4 finished with value: 0.9133333333333334 and parameters: {'C': 4.5594725
[I 2026-01-04 02:52:42,106] Trial 5 finished with value: 0.9800000000000001 and parameters: {'C': 540.338295
[I 2026-01-04 02:52:42,131] Trial 6 finished with value: 0.3333333333333333 and parameters: {'C': 1.1520204
[I 2026-01-04 02:52:42,314] Trial 7 finished with value: 0.9466666666666667 and parameters: {'C': 0.0093126
[I 2026-01-04 02:52:42,415] Trial 8 finished with value: 0.96 and parameters: {'C': 0.007917928060705154, 'g
[I 2026-01-04 02:52:42,436] Trial 9 finished with value: 0.9666666666666666 and parameters: {'C': 45.2734548
[I 2026-01-04 02:52:42,472] Trial 10 finished with value: 0.9133333333333334 and parameters: {'C': 0.1064698
[I 2026-01-04 02:52:42,500] Trial 11 finished with value: 0.9733333333333334 and parameters: {'C': 89.258292
[I 2026-01-04 02:52:42,529] Trial 12 finished with value: 0.9666666666666666 and parameters: {'C': 963.25331
[I 2026-01-04 02:52:42,556] Trial 13 finished with value: 0.9666666666666666 and parameters: {'C': 65.332751
[I 2026-01-04 02:52:42,587] Trial 14 finished with value: 0.9533333333333334 and parameters: {'C': 19.192455
[I 2026-01-04 02:52:42,616] Trial 15 finished with value: 0.9666666666666666 and parameters: {'C': 287.15627
[I 2026-01-04 02:52:42,653] Trial 16 finished with value: 0.9133333333333334 and parameters: {'C': 0.176447
[I 2026-01-04 02:52:42,684] Trial 17 finished with value: 0.9733333333333334 and parameters: {'C': 10.094818
[I 2026-01-04 02:52:42,718] Trial 18 finished with value: 0.29333333333333333 and parameters: {'C': 194.2844
[I 2026-01-04 02:52:42,753] Trial 19 finished with value: 0.7466666666666667 and parameters: {'C': 0.1345846
[I 2026-01-04 02:52:42,783] Trial 20 finished with value: 0.9666666666666666 and parameters: {'C': 208.98437
[I 2026-01-04 02:52:42,813] Trial 21 finished with value: 0.96 and parameters: {'C': 12.864230528294085, 'ga
[I 2026-01-04 02:52:42,842] Trial 22 finished with value: 0.9733333333333334 and parameters: {'C': 11.740207
[I 2026-01-04 02:52:42,873] Trial 23 finished with value: 0.9533333333333334 and parameters: {'C': 34.775658
[I 2026-01-04 02:52:42,903] Trial 24 finished with value: 0.9666666666666668 and parameters: {'C': 157.0374
[I 2026-01-04 02:52:42,933] Trial 25 finished with value: 0.9866666666666667 and parameters: {'C': 4.372349
[I 2026-01-04 02:52:42,964] Trial 26 finished with value: 0.96 and parameters: {'C': 0.4997566121772906, 'ga
[I 2026-01-04 02:52:42,994] Trial 27 finished with value: 0.9733333333333334 and parameters: {'C': 455.3256
[I 2026-01-04 02:52:43,029] Trial 28 finished with value: 0.7466666666666667 and parameters: {'C': 3.627624
[I 2026-01-04 02:52:43,066] Trial 29 finished with value: 0.13333333333333333 and parameters: {'C': 85.3914
Best Accuracy: 0.9866666666666667
Best Hyperparameters:
{'C': 4.372349253752519, 'gamma': 0.11181174530451622, 'kernel': 'rbf'}
```

Practical No 12

Deploying a machine learning model in a production environment using containerization and cloud services

Trained Model
↓
REST API (FastAPI)
↓
Docker Container
↓
Cloud Service (AWS / GCP / Azure)
↓
Public Endpoint

- Step 1: Train & Save the Model

Example using scikit-learn.

python
Copy code

```
# train.py
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
import joblib

X, y = load_iris(return_X_y=True)

model = RandomForestClassifier()
model.fit(X, y)

joblib.dump(model, "model.joblib")
```

- Step 2: Create an Inference API (FastAPI)

python
Copy code

```
# app.py
from fastapi import FastAPI
import joblib
import numpy as np

app = FastAPI()
model = joblib.load("model.joblib")

@app.get("/")
def health_check():
    return {"status": "Model is running"}

@app.post("/predict")
def predict(features: list):
    data = np.array(features).reshape(1, -1)
    prediction = model.predict(data)
```

```
    return {"prediction": int(prediction[0])}
```

🔹 Step 3: Define Dependencies

Create requirements.txt:

```
nginx
Copy code
fastapi
uvicorn
scikit-learn
joblib
numpy
```

🔹 Step 4: Containerize with Docker

Dockerfile

dockerfile

```
Copy code
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY app.py model.joblib ./

EXPOSE 8000

CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

🔹 Step 5: Build & Run Locally

bash

```
Copy code
docker build -t ml-model-api .
docker run -p 8000:8000 ml-model-api
```

Test:

bash

```
Copy code
curl -X POST "http://localhost:8000/predict" \
    -H "Content-Type: application/json" \
    -d "[5.1, 3.5, 1.4, 0.2]"
```

🔹 Step 6: Push to Cloud (Example: AWS)

1️⃣ Push to Amazon ECR

bash

```
Copy code
aws ecr create-repository --repository-name ml-model-api
docker tag ml-model-api:latest <ECR_URI>
docker push <ECR_URI>
```

Practical No 13

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.

1 Install libraries

```
pip install transformers datasets torch
```

---

2 Prepare your text data

Create a file called `train.txt`:

```
Once upon a time in a distant land, there lived a wise king.
He ruled his kingdom with fairness and courage.
...
```

---

3 Fine-tune GPT-2 on your text

```
from transformers import (
    GPT2Tokenizer,
    GPT2LMHeadModel,
    Trainer,
    TrainingArguments,
    DataCollatorForLanguageModeling
)
from datasets import load_dataset

# Load dataset
dataset = load_dataset("text", data_files={"train": "train.txt"})

# Load tokenizer and model
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
tokenizer.pad_token = tokenizer.eos_token
model = GPT2LMHeadModel.from_pretrained("gpt2")

# Tokenize text
def tokenize(example):
    return tokenizer(example["text"], truncation=True,
padding="max_length", max_length=128)

tokenized_data = dataset.map(tokenize, batched=True,
remove_columns=["text"])

# Training setup
training_args = TrainingArguments(
    output_dir="./gpt2-textgen",
    overwrite_output_dir=True,
    num_train_epochs=3,
    per_device_train_batch_size=2,
    save_steps=500,
    logging_steps=100
)
```

```
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm=False
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_data["train"],
    data_collator=data_collator
)

trainer.train()
```

---

4 Generate new text

```
prompt = "Once upon a time"
inputs = tokenizer(prompt, return_tensors="pt")

output = model.generate(
    **inputs,
    max_length=100,
    temperature=0.8,
    top_k=50,
    top_p=0.95,
    do_sample=True
)

print(tokenizer.decode(output[0], skip_special_tokens=True))
```

Practical No:14

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

Code:

1 Install dependencies

```
pip install torch torchvision matplotlib
```

---

2 Imports & Setup

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torchvision.utils import save_image
```

---

3 Load Image Dataset

```
transform = transforms.Compose([
    transforms.Resize(28),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

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

dataloader = torch.utils.data.DataLoader(
    dataset,
    batch_size=64,
    shuffle=True
)

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

---

4 Define Generator

```
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(100, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
```

```
            nn.ReLU(),
            nn.Linear(512, 784),
            nn.Tanh()
        )

    def forward(self, z):
        return self.model(z)
```

---

5 Define Discriminator

```
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(784, 512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

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

---

6 Initialize Models & Optimizers

```
generator = Generator().to(device)
discriminator = Discriminator().to(device)

criterion = nn.BCELoss()
opt_G = optim.Adam(generator.parameters(), lr=0.0002)
opt_D = optim.Adam(discriminator.parameters(), lr=0.0002)
```

---

7 Training Loop

```
epochs = 20

for epoch in range(epochs):
    for real_imgs, _ in dataloader:
        real_imgs = real_imgs.view(real_imgs.size(0), -1).to(device)

        # Labels
        real_labels = torch.ones(real_imgs.size(0), 1).to(device)
        fake_labels = torch.zeros(real_imgs.size(0), 1).to(device)

        # ---------------------
        # Train Discriminator
        # ---------------------
        z = torch.randn(real_imgs.size(0), 100).to(device)
        fake_imgs = generator(z)

        real_loss = criterion(discriminator(real_imgs), real_labels)
```

```
        fake_loss = criterion(discriminator(fake_imgs.detach()),
fake_labels)
        d_loss = real_loss + fake_loss

        opt_D.zero_grad()
        d_loss.backward()
        opt_D.step()

        # -----------------
        # Train Generator
        # -----------------
        z = torch.randn(real_imgs.size(0), 100).to(device)
        fake_imgs = generator(z)
        g_loss = criterion(discriminator(fake_imgs), real_labels)

        opt_G.zero_grad()
        g_loss.backward()
        opt_G.step()

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

---

8 Generate New Images

```
z = torch.randn(64, 100).to(device)
generated_images = generator(z).view(-1, 1, 28, 28)

save_image(generated_images, "generated.png", nrow=8, normalize=True)
```

output:-

```
...  Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages (2.9.0+cpu)
     Requirement already satisfied: torchvision in /usr/local/lib/python3.12/dist-packages (0.24.0+cpu)
     Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
     Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch) (3.20.0)
     Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.12/dist-packages (from to
     Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch) (75.2.0)
     Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch) (1.14.0
     Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch) (3.6.
     Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch) (3.1.6)
     Requirement already satisfied: fsspec>=0.8.5 in /usr/local/lib/python3.12/dist-packages (from torch) (2025.3
     Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from torchvision) (2.0.2)
     Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.12/dist-packages (from torchv
     Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib)
     Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.
     Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib
     Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib
     Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib)
     Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib)
     Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplot
     Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.
     Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.
     Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torc
     100%|██████████| 9.91M/9.91M [00:00<00:00, 19.7MB/s]
     100%|██████████| 28.9k/28.9k [00:00<00:00, 487kB/s]
     100%|██████████| 1.65M/1.65M [00:00<00:00, 4.33MB/s]
     100%|██████████| 4.54k/4.54k [00:00<00:00, 5.34MB/s]
     Epoch [1/20]  D Loss: 0.7095  G Loss: 4.5900
     Epoch [2/20]  D Loss: 0.3953  G Loss: 2.9186
     Epoch [3/20]  D Loss: 0.2607  G Loss: 3.0564
```

```
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local
100%|██████████| 9.91M/9.91M [00:00<00:00, 19.7MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 487kB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 4.33MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 5.34MB/s]
Epoch [1/20]  D Loss: 0.7095  G Loss: 4.5900
Epoch [2/20]  D Loss: 0.3953  G Loss: 2.9186
Epoch [3/20]  D Loss: 0.2607  G Loss: 3.0564
Epoch [4/20]  D Loss: 0.2781  G Loss: 3.3582
Epoch [5/20]  D Loss: 0.0340  G Loss: 6.6301
Epoch [6/20]  D Loss: 0.1037  G Loss: 4.4536
Epoch [7/20]  D Loss: 0.0633  G Loss: 4.8496
Epoch [8/20]  D Loss: 0.0417  G Loss: 4.9818
Epoch [9/20]  D Loss: 0.2752  G Loss: 4.7137
Epoch [10/20]  D Loss: 0.4584  G Loss: 3.1011
Epoch [11/20]  D Loss: 0.2059  G Loss: 4.8855
Epoch [12/20]  D Loss: 0.3046  G Loss: 5.4943
Epoch [13/20]  D Loss: 0.8135  G Loss: 4.1702
Epoch [14/20]  D Loss: 0.4547  G Loss: 5.0504
Epoch [15/20]  D Loss: 0.1550  G Loss: 3.0624
Epoch [16/20]  D Loss: 0.3790  G Loss: 4.9360
Epoch [17/20]  D Loss: 0.2307  G Loss: 4.1096
Epoch [18/20]  D Loss: 0.3086  G Loss: 3.7646
Epoch [19/20]  D Loss: 0.3493  G Loss: 1.9690
Epoch [20/20]  D Loss: 0.3589  G Loss: 2.8671
```