**Assignment-7**

**Statistics, Machine Learning, Deep Learning**

1. Write a Python program that computes the value of the Gaussian distribution at a given vector X. Hence, plot the effect of varying mean and variance to the normal distribution.

Ans. import numpy as np

import matplotlib.pyplot as plt

def gaussian(X, mean, variance):

coefficient = 1 / np.sqrt(2 \* np.pi \* variance)

exponent = np.exp(-((X - mean) \*\* 2) / (2 \* variance))

return coefficient \* exponent

# Define a range of X values

X = np.linspace(-10, 10, 1000)

# Plotting the Gaussian distributions

mean\_values = [0, 0, 0]

variance\_values = [0.5, 1, 2]

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

for mean, variance in zip(mean\_values, variance\_values):

plt.plot(X, gaussian(X, mean, variance), label=f"Mean = {mean}, Variance = {variance}")

plt.title("Gaussian Distribution")

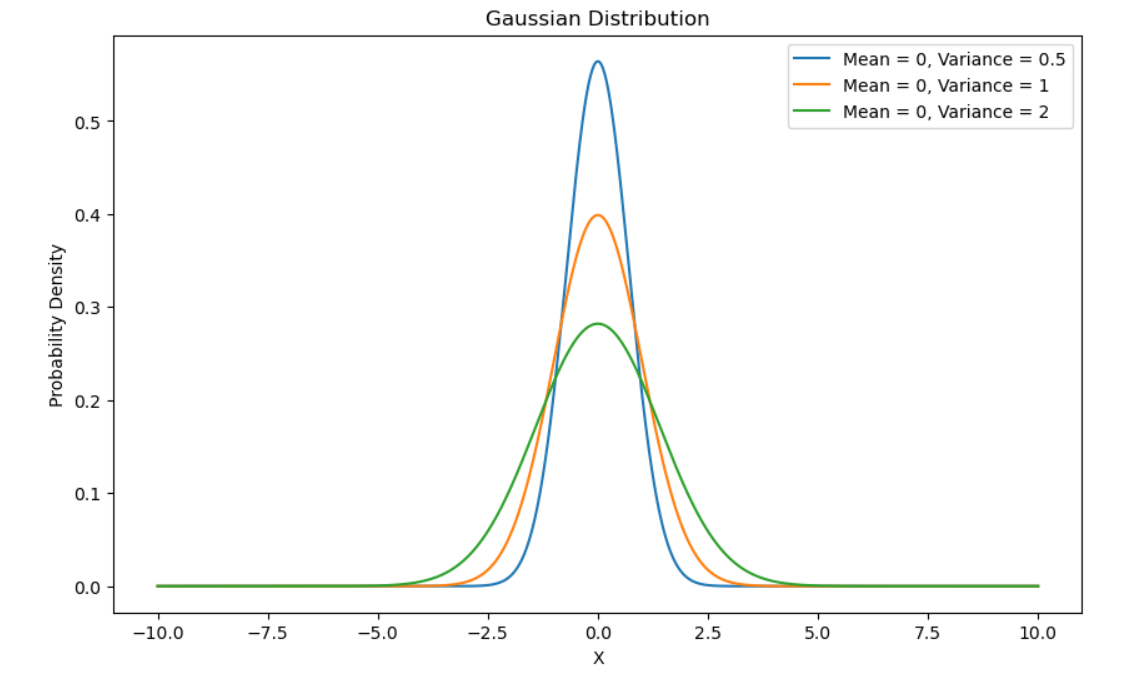
plt.xlabel("X")

plt.ylabel("Probability Density")

plt.legend()

plt.show()

output:



2. Write a python program to implement linear regression.

Ans.

import numpy as np

import matplotlib.pyplot as plt

# Linear Regression using Normal Equation

def linear\_regression(X, y):

X\_b = np.c\_[np.ones((X.shape[0], 1)), X] # Add bias term (intercept)

theta = np.linalg.inv(X\_b.T @ X\_b) @ X\_b.T @ y

return theta

# Sample Data

X = 2 \* np.random.rand(100, 1)

y = 4 + 3 \* X + np.random.randn(100, 1)

# Model

theta = linear\_regression(X, y)

print(f"Model parameters: {theta}")

# Recompute X\_b for plotting

X\_b = np.c\_[np.ones((X.shape[0], 1)), X]

# Plotting

plt.scatter(X, y)

plt.plot(X, X\_b @ theta, color="red")

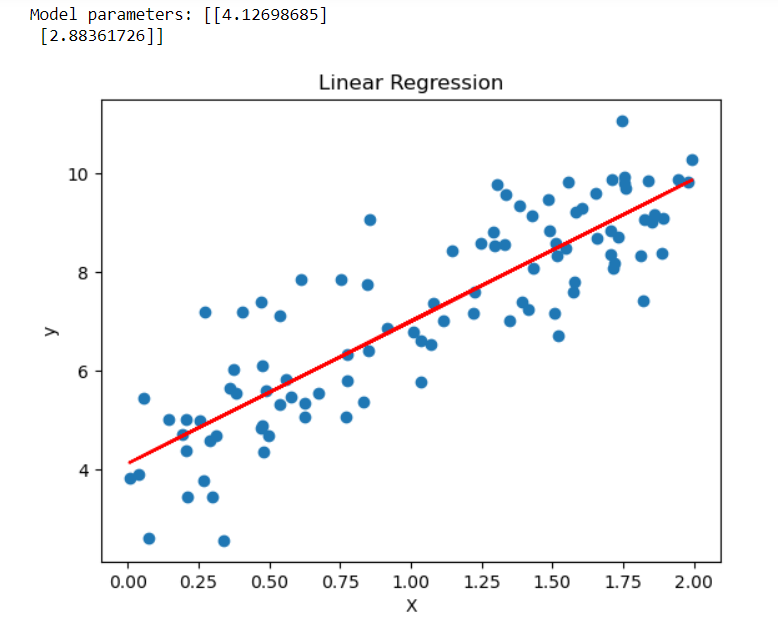
plt.title("Linear Regression")

plt.xlabel("X")

plt.ylabel("y")

plt.show()

output:



3. Write a python program to implement gradient descent.

Ans.

import numpy as np

def gradient\_descent(X, y, learning\_rate=0.01, n\_iterations=1000):

m = X.shape[0]

X\_b = np.c\_[np.ones((m, 1)), X]

theta = np.random.randn(2, 1)

for iteration in range(n\_iterations):

gradients = 2/m \* X\_b.T @ (X\_b @ theta - y)

theta -= learning\_rate \* gradients

return theta

# Sample Data

X = 2 \* np.random.rand(100, 1)

y = 4 + 3 \* X + np.random.randn(100, 1)

# Gradient Descent

theta = gradient\_descent(X, y)

print(f"Model parameters: {theta}")

output:



4. Write a python program to classify different flower images using MLP.

Ans.

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import classification\_report

# Load Dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# MLP Classifier

mlp = MLPClassifier(hidden\_layer\_sizes=(10, 10), max\_iter=1000, random\_state=42)

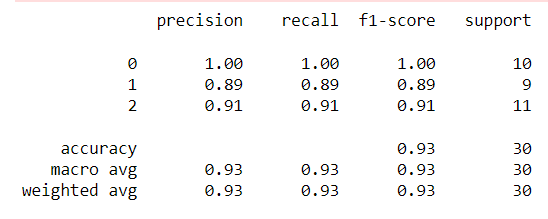
mlp.fit(X\_train, y\_train)

# Evaluation

y\_pred = mlp.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

output:



5. Write a python program to classify different flower images using the SVM classifier.

Ans.

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import classification\_report

# Load Dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# SVM Classifier

svm = SVC(kernel='linear', random\_state=42)

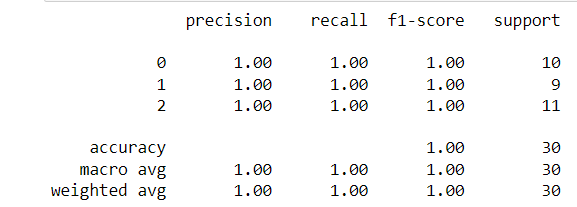
svm.fit(X\_train, y\_train)

# Evaluation

y\_pred = svm.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

output:



6. Write a python program to classify different flower images using CNN.

Ans.

import tensorflow as tf

from tensorflow.keras import layers, models

from sklearn.datasets import load\_iris

# Load Dataset (Example with Iris dataset)

iris = load\_iris()

X = iris.data.reshape(-1, 4, 1) # Reshape for Conv1D

y = iris.target

# CNN Model using Conv1D

model = models.Sequential([

layers.Conv1D(32, 2, activation='relu', input\_shape=(4, 1)),

layers.MaxPooling1D(1),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(3, activation='softmax')

])

# Compile and Train

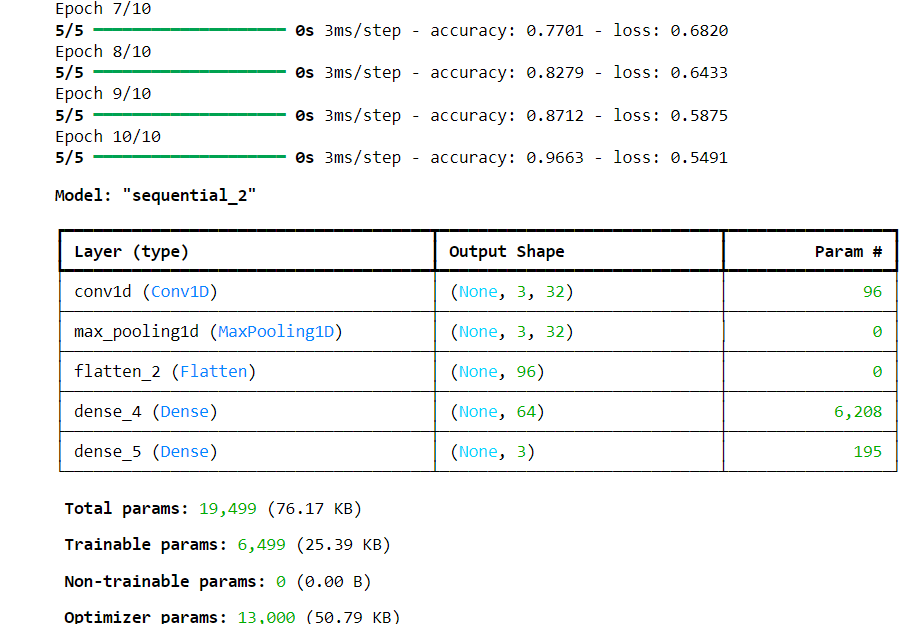
model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.fit(X, y, epochs=10)

# This will print a summary of your model architecture

model.summary()

output:



7. Write a python program to classify different handwritten character images using the SVM classifier.

Ans.

from sklearn import datasets, svm, metrics

from sklearn.model\_selection import train\_test\_split

# Load Dataset

digits = datasets.load\_digits()

X = digits.images.reshape((len(digits.images), -1))

y = digits.target

# Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=42)

# SVM Classifier

classifier = svm.SVC(gamma=0.001)

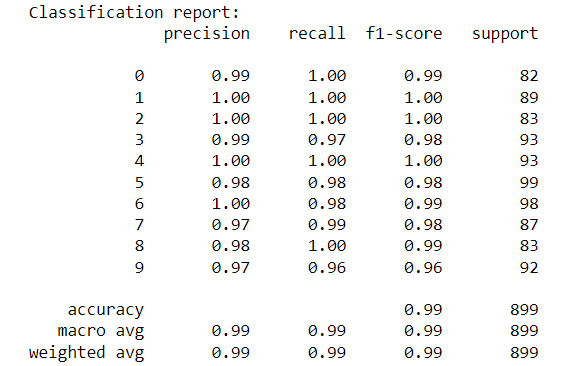
classifier.fit(X\_train, y\_train)

# Predict and Evaluate

y\_pred = classifier.predict(X\_test)

print(f"Classification report:\n{metrics.classification\_report(y\_test, y\_pred)}")

output:



8. Write a python program to classify different face images using CNN.

Ans. import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import cifar10

# Load Dataset (Example with CIFAR-10)

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()

# 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')

])

# Compile and Train

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

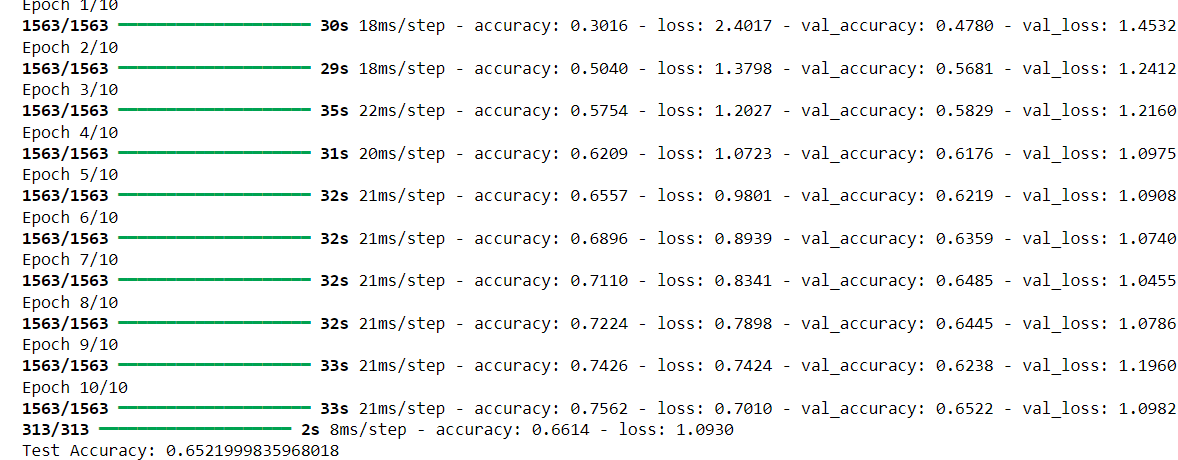
model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test))

# Evaluate

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f"Test Accuracy: {accuracy}")

output:



9. Write a python program to identify a person from the walking style (gait recognition) using convolutional recurrent neural network.

Ans.

import numpy as np

import tensorflow as tf

from tensorflow.keras import layers, models

# Hyperparameters and dataset specifications

num\_samples = 100 # Number of gait sequences

timesteps = 30 # Number of frames per sequence

image\_height, image\_width = 64, 64

num\_channels = 1 # Grayscale images

num\_classes = 10 # Number of people in the dataset

# Randomly generated dataset (replace with actual data loading)

# In a real scenario, load and preprocess your dataset accordingly

X\_train = np.random.rand(num\_samples, timesteps, image\_height, image\_width, num\_channels)

y\_train = np.random.randint(0, num\_classes, num\_samples)

X\_test = np.random.rand(num\_samples, timesteps, image\_height, image\_width, num\_channels)

y\_test = np.random.randint(0, num\_classes, num\_samples)

# Define the CRNN model

model = models.Sequential([

# Convolutional layers to extract spatial features from each frame

layers.TimeDistributed(layers.Conv2D(32, (3, 3), activation='relu'), input\_shape=(timesteps, image\_height, image\_width, num\_channels)),

layers.TimeDistributed(layers.MaxPooling2D((2, 2))),

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

layers.TimeDistributed(layers.MaxPooling2D((2, 2))),

layers.TimeDistributed(layers.Conv2D(128, (3, 3), activation='relu')),

layers.TimeDistributed(layers.MaxPooling2D((2, 2))),

layers.TimeDistributed(layers.Flatten()),

# LSTM layer to capture temporal dependencies across frames

layers.LSTM(128, return\_sequences=False),

layers.Dropout(0.5),

# Fully connected layer and output layer

layers.Dense(64, activation='relu'),

layers.Dense(num\_classes, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Print model summary

model.summary()

# Train the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=16, validation\_split=0.2)

# Evaluate the model

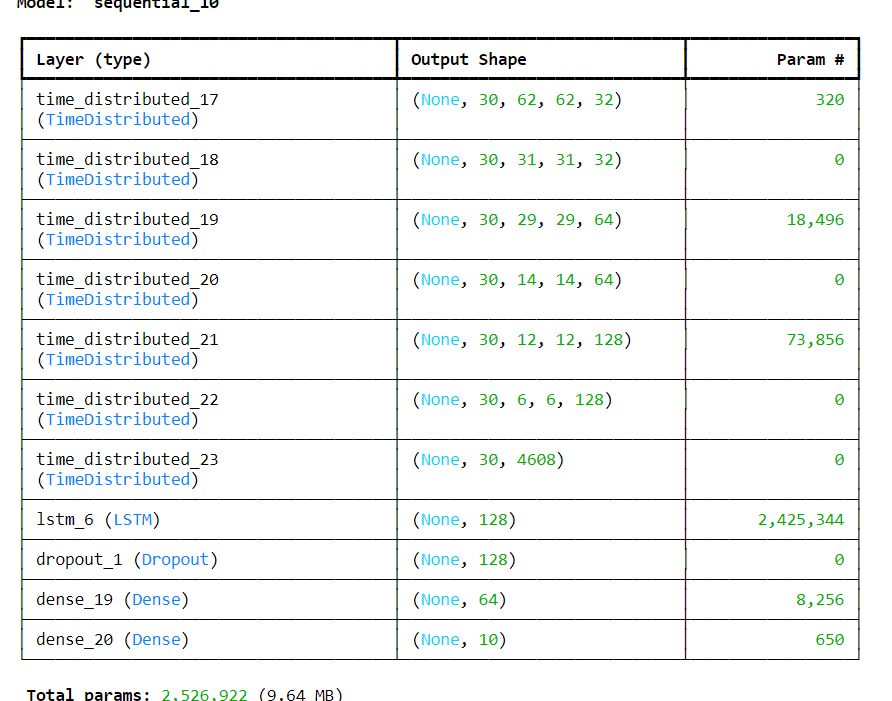
test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

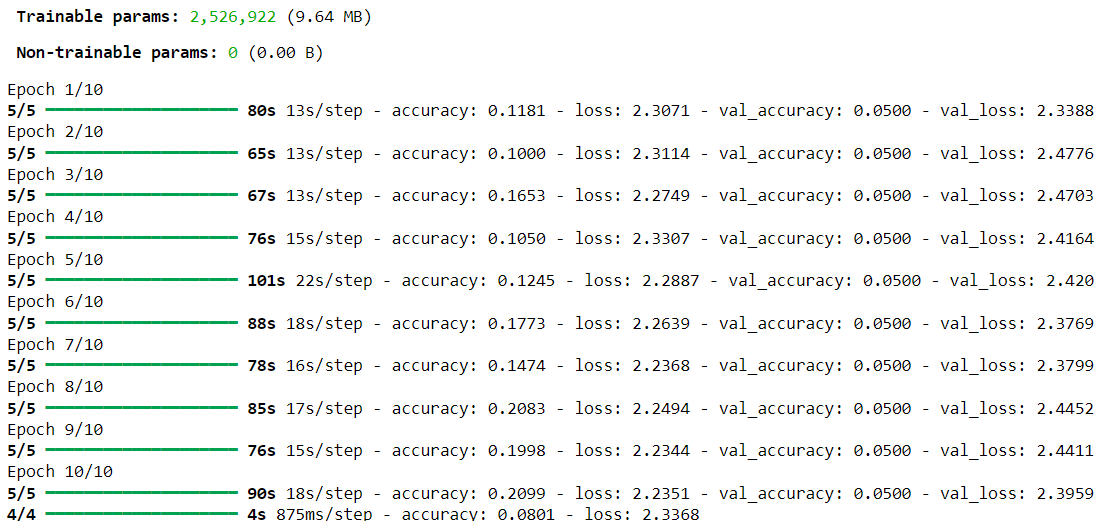
print(f"Test Accuracy: {test\_acc \* 100:.2f}%")

# Save the model

model.save('gait\_recognition\_model.h5')

output:







10. Write a python program to classify breast cancer from histopathological images using VGG-16 and DenseNet-201 CNN architectures.

Ans.

import numpy as np

from PIL import Image

import os

def create\_random\_image(size, save\_path):

img = Image.fromarray(np.random.randint(0, 256, size=(size[1], size[0], 3), dtype=np.uint8))

img.save(save\_path)

# Parameters

img\_size = (224, 224)

num\_images = 100

# Create directories

os.makedirs('/content/synthetic/train/class1', exist\_ok=True)

os.makedirs('/content/synthetic/train/class2', exist\_ok=True)

# Generate random images

for i in range(num\_images):

create\_random\_image(img\_size, f'/content/synthetic/train/class1/image\_{i}.jpg')

create\_random\_image(img\_size, f'/content/synthetic/train/class2/image\_{i}.jpg')

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train\_dir = '/content/synthetic/train'

# Data augmentation and preprocessing

train\_datagen = ImageDataGenerator(

rescale=1./255,

rotation\_range=40,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

fill\_mode='nearest'

)

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=(224, 224),

batch\_size=32,

class\_mode='binary' # Use 'categorical' if more than two classes

)

import tensorflow as tf

# VGG-16 Model

def build\_vgg16\_model():

base\_model = tf.keras.applications.VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

x = base\_model.output

x = tf.keras.layers.Flatten()(x)

x = tf.keras.layers.Dense(256, activation='relu')(x)

x = tf.keras.layers.Dropout(0.5)(x)

predictions = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.models.Model(inputs=base\_model.input, outputs=predictions)

for layer in base\_model.layers:

layer.trainable = False

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

return model

# DenseNet-201 Model

def build\_densenet201\_model():

base\_model = tf.keras.applications.DenseNet201(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

x = base\_model.output

x = tf.keras.layers.Flatten()(x)

x = tf.keras.layers.Dense(256, activation='relu')(x)

x = tf.keras.layers.Dropout(0.5)(x)

predictions = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.models.Model(inputs=base\_model.input, outputs=predictions)

for layer in base\_model.layers:

layer.trainable = False

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

return model

# Train VGG-16

vgg16\_model = build\_vgg16\_model()

vgg16\_history = vgg16\_model.fit(

train\_generator,

steps\_per\_epoch=train\_generator.samples // 32,

epochs=10

)

# Train DenseNet-201

densenet201\_model = build\_densenet201\_model()

densenet201\_history = densenet201\_model.fit(

train\_generator,

steps\_per\_epoch=train\_generator.samples // 32,

epochs=10

)

import matplotlib.pyplot as plt

# Plot VGG-16 Training History

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.plot(vgg16\_history.history['accuracy'], label='accuracy')

plt.plot(vgg16\_history.history['loss'], label='loss')

plt.title('VGG-16 Training History')

plt.xlabel('Epoch')

plt.ylabel('Value')

plt.legend()

# Plot DenseNet-201 Training History

plt.subplot(1, 2, 2)

plt.plot(densenet201\_history.history['accuracy'], label='accuracy')

plt.plot(densenet201\_history.history['loss'], label='loss')

plt.title('DenseNet-201 Training History')

plt.xlabel('Epoch')

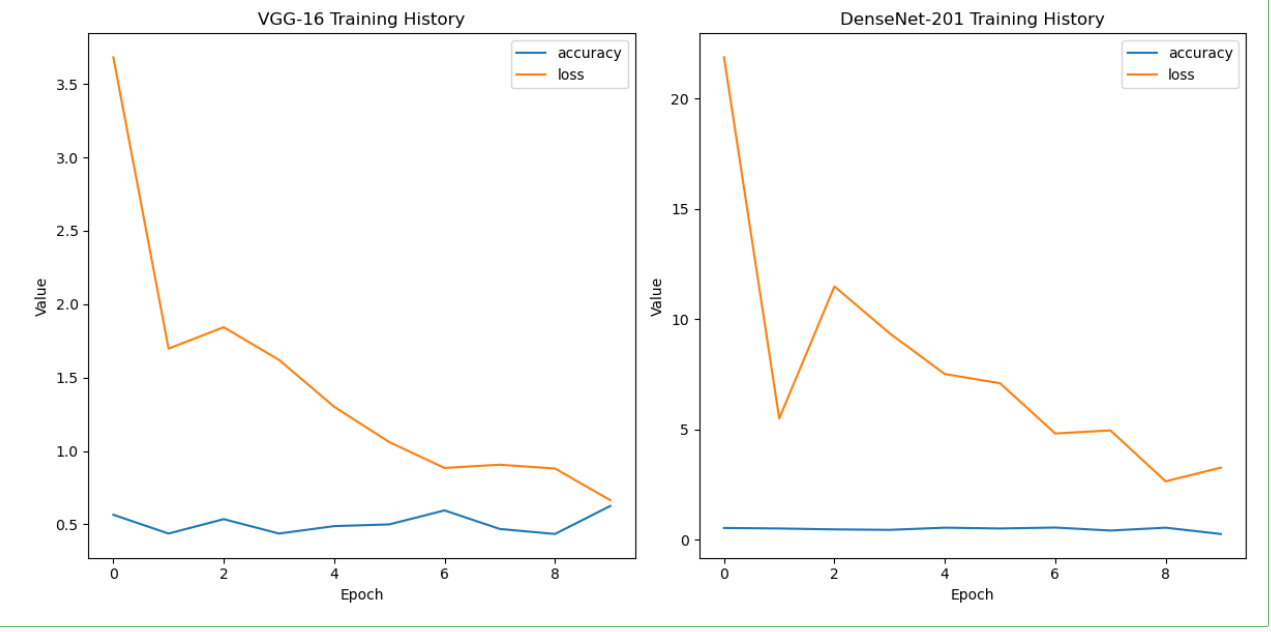
plt.ylabel('Value')

plt.legend()

plt.tight\_layout()

plt.show()

output:



11. Given are two one-dimensional arrays A & B, which are sorted in ascending order. Write a Java program to merge them into single sorted array C that contains every item from arrays A & B, in ascending order.

Ans.

def merge\_sorted\_arrays(A, B):

merged\_array = []

i = j = 0

while i < len(A) and j < len(B):

if A[i] < B[j]:

merged\_array.append(A[i])

i += 1

else:

merged\_array.append(B[j])

j += 1

# Append remaining elements from A or B

while i < len(A):

merged\_array.append(A[i])

i += 1

while j < len(B):

merged\_array.append(B[j])

j += 1

return merged\_array

# Example usage

A = [1, 3, 5, 7]

B = [2, 4, 6, 8]

C = merge\_sorted\_arrays(A, B)

print("Merged array:", C)

output:



12.Write a Java program to show 0-arguments constructor.

Ans. class ZeroArgumentsConstructor:

def \_\_init\_\_(self):

self.message = "This is a zero-arguments constructor"

def show\_message(self):

print(self.message)

# Example usage

obj = ZeroArgumentsConstructor()

obj.show\_message()

output:



13.Write a Java program to show parameterized constructor.

Ans.

class ParameterizedConstructor:

def \_\_init\_\_(self, value):

self.value = value

def show\_value(self):

print(f"Value: {self.value}")

# Example usage

obj = ParameterizedConstructor(10)

obj.show\_value()

output:



14. Write a Java program to show constructor overloading.

Ans.

class ConstructorOverloading:

def \_\_init\_\_(self, value1=0, value2=0):

self.value1 = value1

self.value2 = value2

def show\_values(self):

print(f"Value1: {self.value1}, Value2: {self.value2}")

# Example usage

obj1 = ConstructorOverloading()

obj1.show\_values()

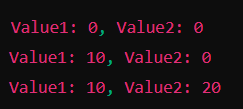
obj2 = ConstructorOverloading(10)

obj2.show\_values()

obj3 = ConstructorOverloading(10, 20)

obj3.show\_values()

output:



15. Write a class, Grader, which has an instance variable, score, an appropriate constructor and appropriate methods.

A method, letterGrade() that returns the letter grade as O/E/A/B/C/F. Now write a demo class to test the Grader class by reading a score from the user, using it to create a Grader object after validating that the value is not negative and is not greater then 100. Finally, call the letterGrade() method to get and print the grade.

Ans. class Grader:

def \_\_init\_\_(self, score):

if score < 0 or score > 100:

raise ValueError("Score must be between 0 and 100")

self.score = score

def letter\_grade(self):

if 90 <= self.score <= 100:

return 'O'

elif 80 <= self.score < 90:

return 'E'

elif 70 <= self.score < 80:

return 'A'

elif 60 <= self.score < 70:

return 'B'

elif 50 <= self.score < 60:

return 'C'

else:

return 'F'

# Demo class to test Grader

def demo\_grader():

score = int(input("Enter the score (0-100): "))

try:

grader = Grader(score)

print("Letter Grade:", grader.letter\_grade())

except ValueError as e:

print(e)

# Uncomment to test

# demo\_grader()

Output:



16. Write a class, Commission, which has an instance variable, sales; an appropriate constructor; and a method, commission() that returns the commission.

Now write a demo class to test the Commission class by reading a sale from the user, using it to create a Commission object after validating that the value is not negative. Finally, call the commission() method to get and print the commission. If the sales are negative, your demo should print the message “Invalid Input”.

Ans.

class Commission:

def \_\_init\_\_(self, sales):

if sales < 0:

raise ValueError("Sales cannot be negative")

self.sales = sales

def commission(self):

# Assuming a fixed commission rate of 10% for this example

return self.sales \* 0.10

# Demo class to test Commission

def demo\_commission():

sales = float(input("Enter the sales amount: "))

try:

commission\_calculator = Commission(sales)

print(f"Commission: ${commission\_calculator.commission():.2f}")

except ValueError as e:

print("Invalid Input")

output:

