**Lab-7**

Q1. 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.

Code: import numpy as np

import matplotlib.pyplot as plt

def gaussian\_distribution(x, mean, variance):

    return (1 / np.sqrt(2 \* np.pi \* variance)) \* np.exp(-((x - mean) \*\* 2) / (2 \* variance))

x\_values = np.linspace(-10, 10, 1000)

means = [0, 0, 0]

variances = [1, 2, 3]

for mean, variance in zip(means, variances):

    y\_values = gaussian\_distribution(x\_values, mean, variance)

    plt.plot(x\_values, y\_values, label=f"Mean: {mean}, Variance: {variance}")

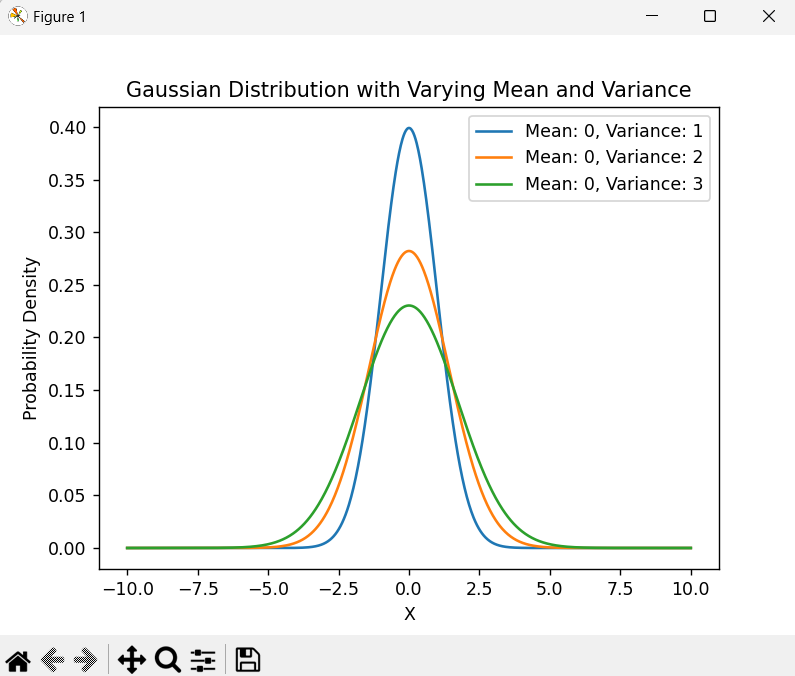
plt.title("Gaussian Distribution with Varying Mean and Variance")

plt.xlabel("X")

plt.ylabel("Probability Density")

plt.legend()

plt.show()



Q2. Write a python program to implement linear regression.

Code: import numpy as np

import matplotlib.pyplot as plt

def linear\_regression(X, y):

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

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

    return theta\_best

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

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

theta = linear\_regression(X, y)

X\_new = np.array([[0], [2]])

X\_new\_b = np.c\_[np.ones((2, 1)), X\_new]

y\_predict = X\_new\_b.dot(theta)

plt.plot(X\_new, y\_predict, "r-")

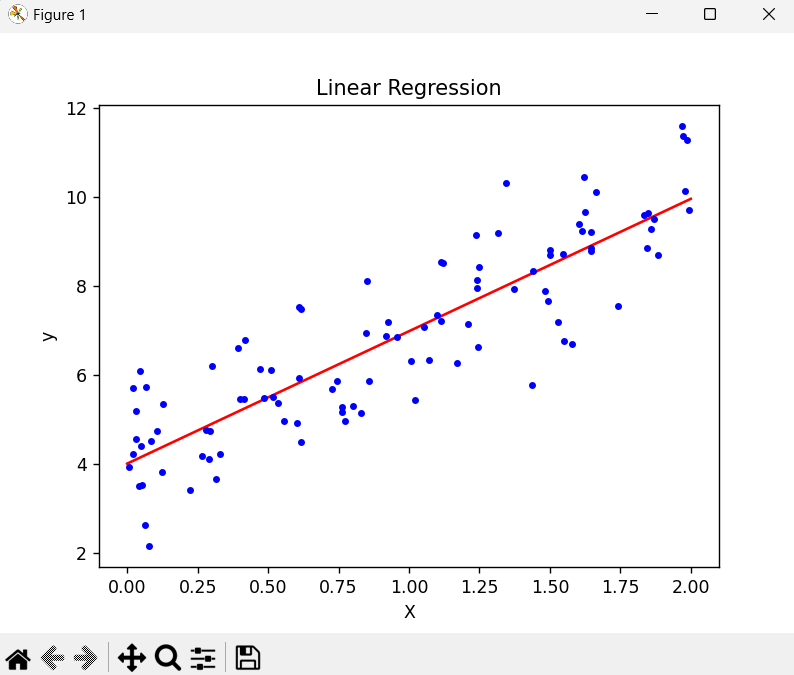
plt.plot(X, y, "b.")

plt.xlabel("X")

plt.ylabel("y")

plt.title("Linear Regression")

plt.show()



Q3. Write a python program to implement gradient descent.

Code: import numpy as np

def gradient\_descent(X, y, learning\_rate=0.1, n\_iterations=1000):

    m = len(y)

    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.dot(X\_b.dot(theta) - y)

        theta = theta - learning\_rate \* gradients

    return theta

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

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

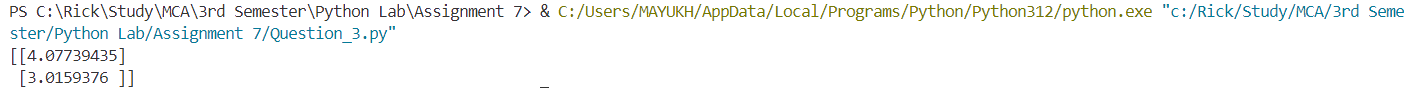
theta = gradient\_descent(X, y)

X\_new = np.array([[0], [2]])

X\_new\_b = np.c\_[np.ones((2, 1)), X\_new]

y\_predict = X\_new\_b.dot(theta)

print(theta)



Q4. Write a python program to classify different flower images using MLP.

Code: import numpy as np

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

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

X\_train = X\_train / 255.0

X\_test = X\_test / 255.0

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

model = models.Sequential()

model.add(layers.Flatten(input\_shape=(32, 32, 3)))

model.add(layers.Dense(512, activation='relu'))

model.add(layers.Dense(256, activation='relu'))

model.add(layers.Dense(128, activation='relu'))

model.add(layers.Dense(10, activation='softmax'))

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

model.fit(X\_train, y\_train, epochs=10, batch\_size=64, validation\_split=0.2)

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

print(test\_acc)



Q5. Write a python program to classify different flower images using the SVM classifier.

Code: import numpy as np

from sklearn import datasets

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

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import classification\_report, accuracy\_score

data = datasets.load\_iris()

X = data.data

y = data.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

model = SVC(kernel='linear')

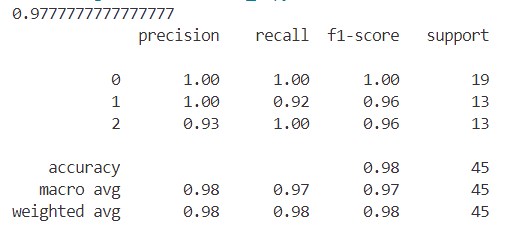
model.fit(X\_train, y\_train)

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

accuracy = accuracy\_score(y\_test, y\_pred)

print(accuracy)

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



Q6. Write a python program to classify different flower images using CNN.

Code: import tensorflow as tf

from tensorflow.keras import layers, models

dataset\_path = r'C:\Rick\Study\MCA\3rd Semester\Python Lab\Assignment 5\jpg'

train\_ds = tf.keras.preprocessing.image\_dataset\_from\_directory(

    dataset\_path,

    validation\_split=0.2,

    subset="training",

    seed=123,

    image\_size=(150, 150),

    batch\_size=32

)

val\_ds = tf.keras.preprocessing.image\_dataset\_from\_directory(

    dataset\_path,

    validation\_split=0.2,

    subset="validation",

    seed=123,

    image\_size=(150, 150),

    batch\_size=32

)

model = models.Sequential([

    layers.Input(shape=(150, 150, 3)),

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

    layers.MaxPooling2D((2, 2)),

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

    layers.MaxPooling2D((2, 2)),

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

    layers.MaxPooling2D((2, 2)),

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

    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

    layers.Dense(512, activation='relu'),

    layers.Dense(1, activation='sigmoid')  # Single output unit with sigmoid activation

])

model.compile(optimizer='adam',

              loss='binary\_crossentropy',

              metrics=['accuracy'])

model.fit(train\_ds, validation\_data=val\_ds, epochs=10)

test\_loss, test\_acc = model.evaluate(val\_ds)

print(f'Test Accuracy: {test\_acc:.4f}')



Q7. Write a python program to classify different handwritten character images using the SVM classifier.

Code: import numpy as np

from sklearn import svm

from sklearn.datasets import fetch\_openml

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import classification\_report, accuracy\_score

mnist = fetch\_openml('mnist\_784', version=1, data\_home='MNIST Dataset')

X = mnist.data

y = mnist.target.astype(int)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

model = svm.SVC(kernel='linear')

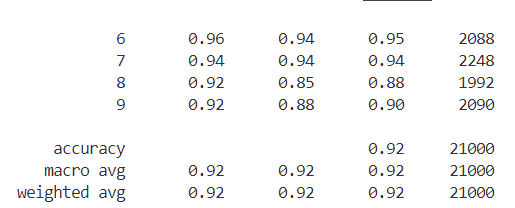
model.fit(X\_train, y\_train)

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

accuracy = accuracy\_score(y\_test, y\_pred)

print(accuracy)

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



Q8. Write a python program to classify different face images using CNN.

Code: import tensorflow as tf

from tensorflow.keras import layers, models

dataset\_path = r'C:\Rick\Study\MCA\3rd Semester\Python Lab\Assignment 7\Faces\_Q8'

train\_ds = tf.keras.preprocessing.image\_dataset\_from\_directory(

    dataset\_path,

    validation\_split=0.2,

    subset="training",

    seed=123,

    image\_size=(150, 150),

    batch\_size=32

)

val\_ds = tf.keras.preprocessing.image\_dataset\_from\_directory(

    dataset\_path,

    validation\_split=0.2,

    subset="validation",

    seed=123,

    image\_size=(150, 150),

    batch\_size=32

)

model = models.Sequential([

    layers.Input(shape=(150, 150, 3)),

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

    layers.MaxPooling2D((2, 2)),

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

    layers.MaxPooling2D((2, 2)),

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

    layers.MaxPooling2D((2, 2)),

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

    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

    layers.Dense(512, activation='relu'),

    layers.Dense(len(train\_ds.class\_names), activation='softmax')

])

model.compile(optimizer='adam',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

model.fit(train\_ds, validation\_data=val\_ds, epochs=10)

test\_loss, test\_acc = model.evaluate(val\_ds)

print(test\_acc)



Q9. Write a python program to identify a person from the walking style (gait recognition) using convolutional recurrent neural network.

Code: import os

import numpy as np

import cv2

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import TimeDistributed, Conv2D, MaxPooling2D, Flatten, LSTM, Dense, Dropout

from tensorflow.keras.utils import to\_categorical

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

from tensorflow.keras.optimizers import Adam

np.random.seed(42)

# Parameters

DATA\_DIR = r'C:\Rick\Study\MCA\3rd Semester\Python Lab\Assignment 7\GaitDataset\_Classi'

CLASSES = ['0001', '0002']

IMG\_HEIGHT = 64

IMG\_WIDTH = 64

CHANNELS = 3

SEQUENCE\_LENGTH = 30

BATCH\_SIZE = 32

EPOCHS = 20

LEARNING\_RATE = 1e-4

def load\_data(data\_dir, classes, sequence\_length, img\_size, channels):

    X = []

    y = []

    label\_map = {cls: idx for idx, cls in enumerate(classes)}

    for cls in classes:

        cls\_dir = os.path.join(data\_dir, cls)

        if not os.path.isdir(cls\_dir):

            print(f"Directory {cls\_dir} does not exist. Skipping.")

            continue

        img\_files = sorted([

            os.path.join(cls\_dir, img) for img in os.listdir(cls\_dir)

            if img.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff'))

        ])

        print(f"Loading {len(img\_files)} images for class '{cls}'")

        # Create sequences

        for i in range(0, len(img\_files) - sequence\_length + 1, sequence\_length):

            seq = []

            for j in range(i, i + sequence\_length):

                img\_path = img\_files[j]

                img = cv2.imread(img\_path)

                if img is None:

                    print(f"Warning: Unable to read image {img\_path}. Skipping.")

                    break

                if channels == 1:

                    img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

                else:

                    img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

                img = cv2.resize(img, img\_size)

                if channels == 1:

                    img = img.reshape(img\_size[0], img\_size[1], 1)

                seq.append(img)

            if len(seq) == sequence\_length:

                X.append(seq)

                y.append(label\_map[cls])

    X = np.array(X)

    y = np.array(y)

    print(f"Total sequences: {X.shape[0]}")

    print(f"Shape of X: {X.shape}")

    print(f"Shape of y: {y.shape}")

    return X, y

def preprocess\_data(X, y, num\_classes):

    X = X.astype('float32') / 255.0

    y = to\_categorical(y, num\_classes)

    return X, y

def build\_crnn\_model(sequence\_length, img\_height, img\_width, channels, num\_classes):

    model = Sequential()

    model.add(TimeDistributed(Conv2D(32, (3, 3), activation='relu'),

                              input\_shape=(sequence\_length, img\_height, img\_width, channels)))

    model.add(TimeDistributed(MaxPooling2D((2, 2))))

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

    model.add(TimeDistributed(MaxPooling2D((2, 2))))

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

    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Flatten()))

    # LSTM layer

    model.add(LSTM(128, return\_sequences=False))

    # Fully connected layers

    model.add(Dense(64, activation='relu'))

    model.add(Dropout(0.5))

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

    model.summary()

    return model

def main():

    # Load data

    X, y = load\_data(DATA\_DIR, CLASSES, SEQUENCE\_LENGTH, (IMG\_WIDTH, IMG\_HEIGHT), CHANNELS)

    num\_classes = len(CLASSES)

    # Preprocess data

    X, y = preprocess\_data(X, y, num\_classes)

    # Split into train and test

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(

        X, y, test\_size=0.2, random\_state=42, stratify=y

    )

    print(f"Training sequences: {X\_train.shape[0]}")

    print(f"Testing sequences: {X\_test.shape[0]}")

    # Build model

    model = build\_crnn\_model(SEQUENCE\_LENGTH, IMG\_HEIGHT, IMG\_WIDTH, CHANNELS, num\_classes)

    # Compile model

    optimizer = Adam(lr=LEARNING\_RATE)

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

    # Train model

    history = model.fit(

        X\_train, y\_train,

        validation\_data=(X\_test, y\_test),

        epochs=EPOCHS,

        batch\_size=BATCH\_SIZE

    )

    # Evaluate model

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

    print(f"Test Loss: {loss}")

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

    # Optionally, save the model

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

    print("Model saved as 'gait\_recognition\_crnn\_model.h5'")

if \_\_name\_\_ == '\_\_main\_\_':

    main()



Q10. Write a python program to classify breast cancer from histopathological images using VGG-16 and DenseNet-201 CNN architectures

Code: import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import VGG16, DenseNet201

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

from sklearn.metrics import classification\_report, confusion\_matrix

train\_dir = r'C:\Rick\Study\MCA\3rd Semester\Python Lab\Assignment 7\Fish\_DS\_Q10\train'

val\_dir = r'C:\Rick\Study\MCA\3rd Semester\Python Lab\Assignment 7\Fish\_DS\_Q10\valid'

test\_dir = r'C:\Rick\Study\MCA\3rd Semester\Python Lab\Assignment 7\Fish\_DS\_Q10\test'

# Image preprocessing and data augmentation

image\_size = (224, 224)

batch\_size = 32

train\_datagen = ImageDataGenerator(rescale=1./255,

                                   rotation\_range=20,

                                   width\_shift\_range=0.2,

                                   height\_shift\_range=0.2,

                                   shear\_range=0.2,

                                   zoom\_range=0.2,

                                   horizontal\_flip=True)

val\_datagen = ImageDataGenerator(rescale=1./255)

test\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory(train\_dir, target\_size=image\_size, batch\_size=batch\_size, class\_mode='binary')

val\_generator = val\_datagen.flow\_from\_directory(val\_dir, target\_size=image\_size, batch\_size=batch\_size, class\_mode='binary')

test\_generator = test\_datagen.flow\_from\_directory(test\_dir, target\_size=image\_size, batch\_size=batch\_size, class\_mode='binary')

# Function to create VGG16 model

def build\_vgg16\_model(input\_shape=(224, 224, 3)):

    base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    x = base\_model.output

    x = GlobalAveragePooling2D()(x)

    x = Dropout(0.5)(x)

    x = Dense(1024, activation='relu')(x)

    predictions = Dense(1, activation='sigmoid')(x)

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

    # Freeze the layers except the last few layers

    for layer in base\_model.layers:

        layer.trainable = False

    return model

# Function to create DenseNet201 model

def build\_densenet\_model(input\_shape=(224, 224, 3)):

    base\_model = DenseNet201(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    x = base\_model.output

    x = GlobalAveragePooling2D()(x)

    x = Dropout(0.5)(x)

    x = Dense(1024, activation='relu')(x)

    predictions = Dense(1, activation='sigmoid')(x)

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

    for layer in base\_model.layers:

        layer.trainable = False

    return model

# Compile and train the models

def compile\_and\_train(model, train\_generator, val\_generator, epochs=10):

    model.compile(optimizer=Adam(learning\_rate=1e-4), loss='binary\_crossentropy', metrics=['accuracy'])

    history = model.fit(train\_generator, validation\_data=val\_generator, epochs=epochs, verbose=1)

    return history

# Load models

vgg16\_model = build\_vgg16\_model()

densenet\_model = build\_densenet\_model()

# Train the models

print("Training VGG16 Model...")

vgg16\_history = compile\_and\_train(vgg16\_model, train\_generator, val\_generator, epochs=10)

print("Training DenseNet201 Model...")

densenet\_history = compile\_and\_train(densenet\_model, train\_generator, val\_generator, epochs=10)

# Evaluate the models on the test set

def evaluate\_model(model, test\_generator):

    test\_loss, test\_acc = model.evaluate(test\_generator, verbose=1)

    print(f'Test accuracy: {test\_acc:.4f}')

    predictions = model.predict(test\_generator, verbose=1)

    y\_pred = (predictions > 0.5).astype(int).ravel()

    y\_true = test\_generator.classes

    print("Confusion Matrix:")

    print(confusion\_matrix(y\_true, y\_pred))

    print("Classification Report:")

    print(classification\_report(y\_true, y\_pred, target\_names=['Class 0', 'Class 1']))

# Evaluate VGG16

print("Evaluating VGG16 Model...")

evaluate\_model(vgg16\_model, test\_generator)

# Evaluate DenseNet201

print("Evaluating DenseNet201 Model...")

evaluate\_model(densenet\_model, test\_generator)

