**Lab 7**

**Statistics, Machine Learning, Deep Learning**

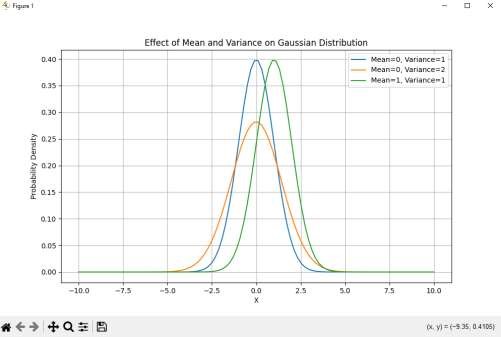
**1. Write a Python program that computes the value of the Gaussian distribution at a given vector X. Hence, plot the effect of varying mean and variance to the normal distribution.**

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, 100) means = [0, 0, 1] variances = [1, 2, 1] plt.figure(figsize=(10, 6)) for mean, variance in zip(means, variances):

plt.plot(x\_values, gaussian\_distribution(x\_values, mean, variance), label=f'Mean={mean}, Variance={variance}') plt.title('Effect of Mean and Variance on Gaussian Distribution') plt.xlabel('X') plt.ylabel('Probability Density') plt.legend() plt.grid(True)

plt.show()

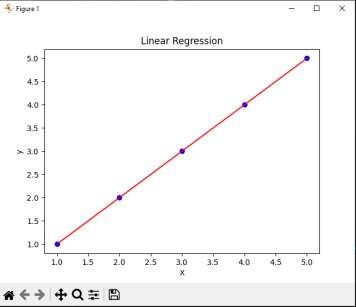


1. **Write a python program to implement linear regression.** import numpy as np from sklearn.linear\_model import LinearRegression import matplotlib.pyplot as plt X = np.array([[1], [2], [3], [4], [5]]) y = np.array([1, 2, 3, 4, 5])

model = LinearRegression() model.fit(X, y) y\_pred = model.predict(X)

plt.scatter(X, y, color='blue') plt.plot(X, y\_pred, color='red') plt.title('Linear Regression') plt.xlabel('X') plt.ylabel('y')

plt.show()



1. **Write a python program to implement gradient descent.**

import numpy as np import matplotlib.pyplot as plt def f(x):

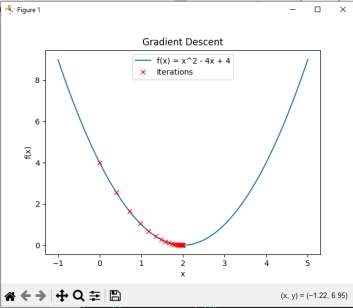
return x\*\*2 - 4\*x + 4 def df(x):

return 2\*x - 4

def gradient\_descent(initial\_x, learning\_rate, num\_iterations):

x = initial\_x x\_history = [x] for i in range(num\_iterations): gradient = df(x) x = x - learning\_rate \* gradient x\_history.append(x) return x, x\_history initial\_x = 0 learning\_rate = 0.1 num\_iterations = 50 x, x\_history = gradient\_descent(initial\_x, learning\_rate, num\_iterations) print("Local minimum: {:.2f}".format(x)) x\_vals = np.linspace(-1, 5, 100) plt.plot(x\_vals, f(x\_vals), label='f(x) = x^2 - 4x + 4') plt.plot(x\_history, f(np.array(x\_history)), 'rx', label='Iterations') plt.xlabel('x') plt.ylabel('f(x)') plt.title('Gradient Descent') plt.legend()

plt.show()



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

import os import numpy as np import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import ImageDataGenerator from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Flatten, Dense, Dropout from tensorflow.keras.layers import Flatten, Dense, Dropout, Input

dataset\_path = r"C:\Users\Viswajeet\Desktop\Python\pythonCollage\Collage\_python\Lab 7\flowers"

data\_gen = ImageDataGenerator(rescale=1.0/255, validation\_split=0.2) train\_data = data\_gen.flow\_from\_directory(

directory=dataset\_path, target\_size=(64, 64),

batch\_size=32, class\_mode='categorical',

subset='training'

)

val\_data = data\_gen.flow\_from\_directory(

directory=dataset\_path, target\_size=(64, 64), batch\_size=32, class\_mode='categorical',

subset='validation'

)

model = Sequential([

Input(shape=(64, 64, 3)),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.5),

Dense(64, activation='relu'),

Dropout(0.5),

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

])

model.compile(optimizer='adam',

loss='categorical\_crossentropy', metrics=['accuracy']) steps\_per\_epoch = len(train\_data)

validation\_steps = len(val\_data)

history = model.fit( train\_data,

validation\_data=val\_data, epochs=20, verbose=1,

steps\_per\_epoch=steps\_per\_epoch,

validation\_steps=validation\_steps

)

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Training Accuracy') plt.plot(history.history['val\_accuracy'], label='Validation Accuracy') plt.title('Training and Validation Accuracy') plt.xlabel('Epochs') plt.ylabel('Accuracy') plt.legend()

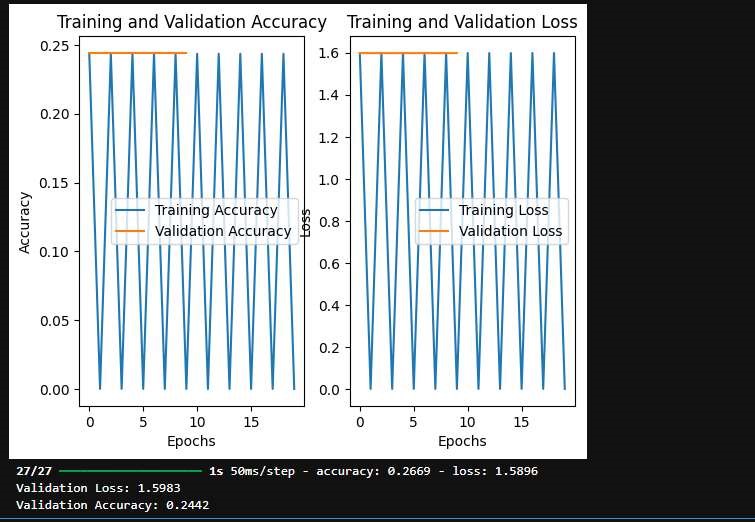
# Loss plot plt.subplot(1, 2, 2) plt.plot(history.history['loss'], label='Training Loss') plt.plot(history.history['val\_loss'], label='Validation Loss') plt.title('Training and Validation Loss') plt.xlabel('Epochs') plt.ylabel('Loss')

plt.legend()

plt.show()

# Final Evaluation

loss, accuracy = model.evaluate(val\_data) print(f"Validation Loss: {loss:.4f}") print(f"Validation Accuracy: {accuracy:.4f}")



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

import os import numpy as np import cv2 from skimage.feature import hog from skimage import exposure from sklearn import svm from sklearn.model\_selection import train\_test\_split from sklearn.metrics import classification\_report, accuracy\_score from sklearn.preprocessing import LabelEncoder from tensorflow.keras.preprocessing.image import ImageDataGenerator

dataset\_path = r"C:\Users\Viswajeet\Desktop\Python\pythonCollage\Collage\_python\Lab 7\flowers"

image\_size = (64, 64) batch\_size = 32 data\_gen = ImageDataGenerator(rescale=1./255, validation\_split=0.2) train\_data\_gen = data\_gen.flow\_from\_directory( directory=dataset\_path, target\_size=image\_size, batch\_size=batch\_size, class\_mode='categorical', subset='training'

)

val\_data\_gen = data\_gen.flow\_from\_directory( directory=dataset\_path, target\_size=image\_size, batch\_size=batch\_size, class\_mode='categorical', subset='validation'

)

class\_names = list(train\_data\_gen.class\_indices.keys()) def extract\_features\_and\_labels(data\_gen):

features = [] labels = []

for batch\_images, batch\_labels in data\_gen: for image, label in zip(batch\_images, batch\_labels):

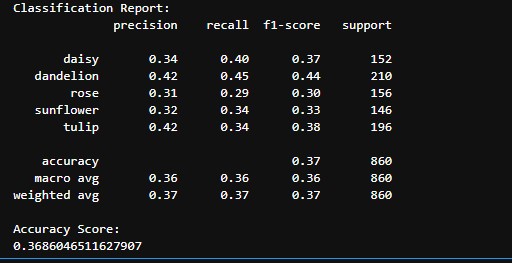
gray\_image = cv2.cvtColor((image \* 255).astype(np.uint8), cv2.COLOR\_RGB2GRAY) hog\_features = hog(gray\_image, pixels\_per\_cell=(8, 8), cells\_per\_block=(2, 2), visualize=False) features.append(hog\_features) labels.append(np.argmax(label)) if len(features) >= data\_gen.samples:

break

return np.array(features), np.array(labels)

X\_train, y\_train = extract\_features\_and\_labels(train\_data\_gen) X\_val, y\_val = extract\_features\_and\_labels(val\_data\_gen) le = LabelEncoder() y\_train = le.fit\_transform(y\_train) y\_val = le.transform(y\_val) clf = svm.SVC(kernel='linear') clf.fit(X\_train, y\_train) y\_pred = clf.predict(X\_val) print("Classification Report:") print(classification\_report(y\_val, y\_pred, target\_names=class\_names)) print("Accuracy Score:")

print(accuracy\_score(y\_val, y\_pred))



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

import tensorflow as tf from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense from tensorflow.keras.preprocessing.image import ImageDataGenerator import os dataset\_path = r'C:\Users\Viswajeet\Desktop\Python\pythonCollage\Collage\_python\Lab 7\flowers' IMG\_SIZE = (128, 128) 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, fill\_mode='nearest', validation\_split=0.2

)

train\_generator = train\_datagen.flow\_from\_directory( dataset\_path, target\_size=IMG\_SIZE, batch\_size=BATCH\_SIZE, class\_mode='categorical',

subset='training'

)

validation\_generator = train\_datagen.flow\_from\_directory( dataset\_path, target\_size=IMG\_SIZE, batch\_size=BATCH\_SIZE, class\_mode='categorical',

subset='validation'

)

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(128, 128, 3)),

MaxPooling2D(pool\_size=(2, 2)),

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

MaxPooling2D(pool\_size=(2, 2)),

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

MaxPooling2D(pool\_size=(2, 2)),

Flatten(),

Dense(128, activation='relu'),

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

])

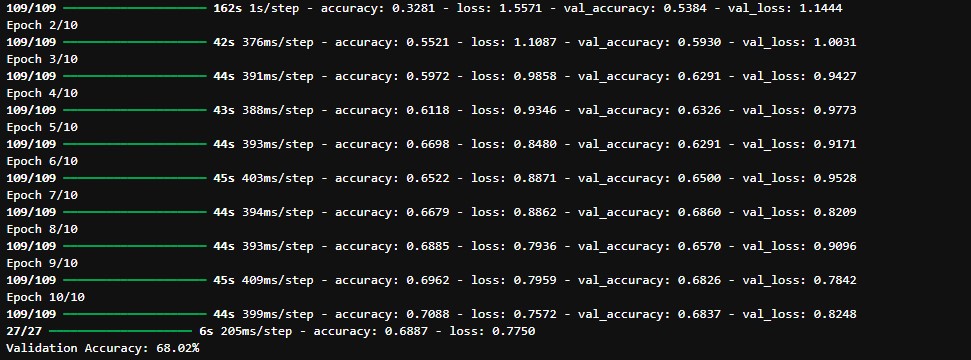
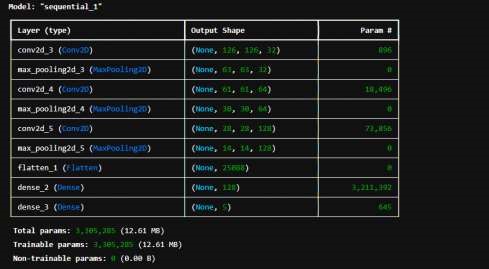
model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) model.summary()

history = model.fit( train\_generator, epochs=10, validation\_data=validation\_generator

)

val\_loss, val\_acc = model.evaluate(validation\_generator)

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



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

import os import cv2 import numpy as np from sklearn import svm from sklearn.metrics import accuracy\_score, classification\_report from sklearn.model\_selection import train\_test\_split from tensorflow.keras.preprocessing.image import ImageDataGenerator dataset\_path = r'C:\Users\Viswajeet\Desktop\Python\pythonCollage\Collage\_python\Lab 7\flowers' IMG\_SIZE = (128, 128)

def load\_images\_from\_folder(folder):

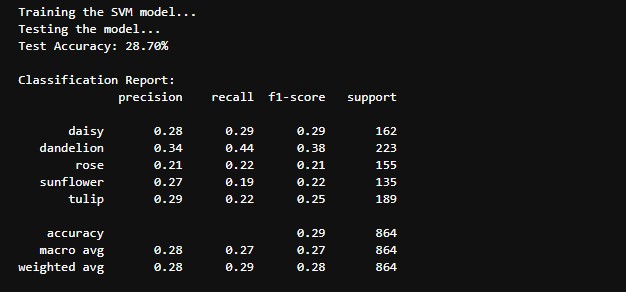
images = [] labels = [] label\_names = os.listdir(folder) for label\_index, label\_name in enumerate(label\_names):

label\_folder = os.path.join(folder, label\_name) if os.path.isdir(label\_folder): for filename in os.listdir(label\_folder):

img\_path = os.path.join(label\_folder, filename) img = cv2.imread(img\_path)

if img is not None:

img = cv2.resize(img, IMG\_SIZE) img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) images.append(img.flatten()) labels.append(label\_index) return np.array(images), np.array(labels), label\_names print("Loading dataset...") x, y, label\_names = load\_images\_from\_folder(dataset\_path) x = x / 255.0 x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42) clf = svm.SVC(kernel='linear') print("Training the SVM model...") clf.fit(x\_train, y\_train) print("Testing the model...") y\_pred = clf.predict(x\_test) accuracy = accuracy\_score(y\_test, y\_pred) print(f"Test Accuracy: {accuracy \* 100:.2f}%") print("\nClassification Report:") print(classification\_report(y\_test, y\_pred, target\_names=label\_names))



**10. Write a python program to classify breast cancer from histopathological images using VGG-16 and DenseNet201 CNN architectures** import pandas as pd import numpy as np from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler, LabelEncoder from sklearn.metrics import accuracy\_score, confusion\_matrix import matplotlib.pyplot as plt from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense, Dropout from tensorflow.keras.optimizers import Adam from tensorflow.keras.callbacks import EarlyStopping from tensorflow.keras.layers import Input

csv\_file = r'C:\Users\Viswajeet\Desktop\Python\pythonCollage\Collage\_python\Lab 7\breast-cancer.csv' df = pd.read\_csv(csv\_file) print(df.info())

categorical\_cols = df.select\_dtypes(include=['object']).columns for col in categorical\_cols: if df[col].nunique() == 2: le = LabelEncoder() df[col] = le.fit\_transform(df[col])

else:

df = pd.get\_dummies(df, columns=[col])

X = df.iloc[:, :-1].values y = df.iloc[:, -1].values

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

X\_train = scaler.fit\_transform(X\_train) X\_test = scaler.transform(X\_test) def build\_model(input\_shape): model = Sequential([

Input(shape=(input\_shape,)),

Dense(128, activation='relu'),

Dropout(0.3),

Dense(64, activation='relu'),

Dropout(0.3),

Dense(32, activation='relu'),

Dense(1, activation='sigmoid')

])

return model

input\_shape = X\_train.shape[1] model = build\_model(input\_shape) model.compile(optimizer=Adam(learning\_rate=0.001), loss='binary\_crossentropy', metrics=['accuracy']) early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

history = model.fit( X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=50, batch\_size=32, callbacks=[early\_stopping]

)

y\_pred = model.predict(X\_test) y\_pred = (y\_pred > 0.5).astype(int) y\_test = y\_test.astype(int) accuracy = accuracy\_score(y\_test, y\_pred) print(f'Test Accuracy: {accuracy \* 100:.2f}%') conf\_matrix = confusion\_matrix(y\_test, y\_pred) print("Confusion Matrix:") print(conf\_matrix) plt.plot(history.history['accuracy']) plt.plot(history.history['val\_accuracy']) plt.title('Model accuracy') plt.ylabel('Accuracy') plt.xlabel('Epoch') plt.legend(['Train', 'Test'], loc='upper left') plt.show() plt.plot(history.history['loss']) plt.plot(history.history['val\_loss']) plt.title('Model loss') plt.ylabel('Loss') plt.xlabel('Epoch') plt.legend(['Train', 'Test'], loc='upper left')

plt.show()



