**Question.no : 4**

**BRAIN TUMOUR DETECTION**

Problem Statement:

From the given dataset, Detect the Brain Tumor using Python (without using sklearn, openCV).Implement the Python Code in AWS Lambda function. Create an AWS API for the lambda

function and display the results by executing the API using python.

Source code:

from google.colab import drive

drive.mount('/content/drive')

1. Data preprocessing steps and importing required libraries

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.image as img

import seaborn as sb

import numpy as np

from imutils import paths

from skimage.io import imshow

bt\_data = pd.read\_csv('drive/MyDrive/Datasets/Question 4/Qn3\_Braintumor/Brain\_Tumor.csv')

bt\_data

(i) Data Cleaning:

bt\_data = bt\_data.dropna()

(ii) Eploratory Data Analysis:

bt\_data.info()

bt\_data.describe()

bt\_data.head()

print(bt\_data.corr())

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

sb.heatmap(bt\_data.corr())

(iii) Exploring Target Class(Tumor has or not)

plt.figure(figsize=(8,6))

p = sb.barplot(x=np.unique(bt\_data['Class']), y=bt\_data['Class'].value\_counts())

p.set\_xlabel("Class", fontsize = 10)

p.set\_ylabel("Count", fontsize = 10)

2.Machine Learning Approach using Logistic Regression

X = bt\_data.drop(['Class','Image'], axis = 1)

X

Y = bt\_data['Class']

Y

X = np.asarray(X).astype('float32')

Y = np.asarray(Y).astype('float32')

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

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.1)

class LogitRegression() :

def \_\_init\_\_( self, learning\_rate, iterations ) :

self.learning\_rate = learning\_rate

self.iterations = iterations

# Function for model training

def fit( self, X, Y ) :

# no\_of\_training\_examples, no\_of\_features

self.m, self.n = X.shape

# weight initialization

self.W = np.zeros( self.n )

self.b = 0

self.X = X

self.Y = Y

# gradient descent learning

for i in range( self.iterations ) :

self.update\_weights()

return self

# Helper function to update weights in gradient descent

def update\_weights( self ) :

A = 1 / ( 1 + np.exp( - ( self.X.dot( self.W ) + self.b ) ) )

# calculate gradients

tmp = ( A - self.Y.T )

tmp = np.reshape( tmp, self.m )

dW = np.dot( self.X.T, tmp ) / self.m

db = np.sum( tmp ) / self.m

# update weights

self.W = self.W - self.learning\_rate \* dW

self.b = self.b - self.learning\_rate \* db

return self

# Hypothetical function h( x )

def predict( self, X ) :

Z = 1 / ( 1 + np.exp( - ( X.dot( self.W ) + self.b ) ) )

Y = np.where( Z > 0.5, 1, 0 )

return Y

model = LogitRegression( learning\_rate = 0.05, iterations = 5000 )

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

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

Y\_pred

correctly\_classified = 0

# counter

for count in range( np.size( Y\_pred ) ) :

if Y\_test[count] == Y\_pred[count] :

correctly\_classified = correctly\_classified + 1

print( "Accuracy on test set : %.2f"%((correctly\_classified / count ) \* 100) )

Y\_test

3. Neural Network Approach using CNN

def import\_images(folder):

images = []

imagePaths = list(paths.list\_images(folder))

for (i,imagePath) in enumerate(imagePaths):

img\_1 = img.imread(imagePath)

if img is not None:

images.append(img\_1)

return images

bt\_imgs = import\_images('drive/MyDrive/Datasets/Question 4/Qn3\_Braintumor/Brain Tumor')

imshow(bt\_imgs[1])

np.array(bt\_imgs).shape

from skimage.color import rgb2gray

gray\_imgs = []

for i in bt\_imgs:

gray\_imgs.append(np.asarray(rgb2gray(i)))

imshow(gray\_imgs[0])

gray\_imgs[0].shape

new\_Y = np.append(Y, 0)

new\_Y

x\_train, x\_test, y\_train, y\_test = train\_test\_split(np.asarray(gray\_imgs), new\_Y, test\_size = 0.1)

from keras import Sequential

from keras.layers import Dense, Flatten

from tensorflow.keras import layers as ls

cnn\_model = Sequential()

cnn\_model.add(ls.Conv2D(32, (3,3), activation="relu",input\_shape = (240,240,1)))

cnn\_model.add(ls.MaxPooling2D((2,2)))

cnn\_model.add(ls.Conv2D(64, (3,3), activation="relu",input\_shape = (240,240,1)))

cnn\_model.add(ls.MaxPooling2D((2,2)))

cnn\_model.add(Flatten())

cnn\_model.add(Dense(64, activation = 'relu'))

cnn\_model.add(Dense(128, activation = 'tanh'))#relu

cnn\_model.add(Dense(1,activation='sigmoid'))

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

history\_cnn = cnn\_model.fit(x\_train, y\_train, epochs = 10, validation\_split = 0.2)

#prediction

i = 55

predict = cnn\_model.predict(X\_train[i].reshape(-1,240,240))

print("Predicted:",predict, "Actual:",Y\_train[i])

imshow(X\_train[i])

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

plt.plot(history\_cnn.history['accuracy'], 'green')

plt.plot(history\_cnn.history['val\_accuracy'], 'red')

plt.title('Accuracy for Model')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'validate'], loc='upper left')

Output:













