Brain Tumor Classification:

Ds675 – 1J2 Machine Learning

Professor:

Ioannis Koutis

Prepare By:

Shouhaybou Mbow

Table of Contents

Sr.No	**Topics**	**Page No**
1	Introduction	3
2	Problem Statement	3
4	Objective	4
5	Dataset	4
6	Exploratory Data Analysis	5
7	Methodology	6
8	Results	11
9	Difference From Previous works	12
10	Conclusion	12

1. Introduction

Brain tumors pose a significant health challenge worldwide, with diverse manifestations, treatment options, and prognoses. They arise from abnormal growths of cells within the brain or its surrounding tissues and can be benign or malignant. The classification of brain tumors is crucial for accurate diagnosis, treatment planning, and prognostic assessment. Traditional methods of classification rely on histological examination, which can be invasive, time-consuming, and subject to interobserver variability.

In recent years, advances in medical imaging techniques, particularly magnetic resonance imaging (MRI), have revolutionized the field of brain tumor classification. These imaging modalities provide non-invasive and detailed visualization of brain structures, enabling the detection and characterization of tumors with high precision. Furthermore, the integration of machine learning and deep learning approaches has shown promising results in automating the process of brain tumor classification

2. Problem Statement

Traditional methods of brain tumor classification rely on histopathological examination of tissue samples obtained through invasive procedures such as biopsies or surgical resections. However, these methods are associated with risks, may not always be feasible, and can be time-consuming. Furthermore, histopathological analysis may suffer from interobserver variability and may not fully capture the heterogeneity of brain tumors.

In recent years, medical imaging techniques, particularly MRI, have emerged as powerful tools for non-invasive visualization of brain structures and pathological changes. MRI provides detailed anatomical information and can differentiate between various tissue types based on their magnetic properties. By analyzing MRI images, radiologists can identify and characterize brain tumors without the need for invasive procedures.

4. Objective

The goal of this project is to develop a machine-learning model capable of automatically classifying brain tumors based on MRI scans. By leveraging deep learning techniques, we aim to train a model that can accurately distinguish if a the image contains a tumor or not. It is compromise of a YES and a No file that has different tumor types, such as gliomas, meningiomas, and metastatic tumors. The model should be able to process MRI images as input and output the predicted tumor type, providing valuable support to radiologists and clinicians in their diagnostic decision-making process.

5. Dataset

Data Source: https://www.kaggle.com/datasets/luluw8071/brain-tumor-mri-datasets/data

The dataset consists of 253 brain MRI images, which serve as the primary data source for training and evaluating the brain tumor classification model.

Among the 253 MRI images, 155 images are labeled as positive cases, indicating the presence of brain tumors. These images are stored in the folder labeled "yes." Conversely, the remaining 98 images are labeled as negative cases, indicating the absence of brain tumors. These images are stored in the folder labeled "no."

The folder labeled "yes" contains MRI images that depict brain scans with detected tumors. These images are critical for training the classification model to recognize features associated with brain tumors. Positive cases provide valuable insights into the appearance, location, and characteristics of different types of brain tumors.

In contrast, the folder labeled "no" contains MRI images that represent brain scans without any detected tumors. These images serve as examples of normal brain anatomy and aid in teaching the model to distinguish between tumorous and non-tumorous conditions. Negative cases help prevent the model from misclassifying normal brain structures as tumors, thus improving its specificity and overall performance.

6. Exploratory Data Analysis

- A summary of some of the steps for conducting Exploratory Data Analysis (EDA) on a brain tumor classification dataset are:

- We Load the dataset into our analysis environment which in this case is Jupiter Notebook. This involves reading image files from directories.

- We explore the directory structure to understand how the data is organized by checking the number of samples in each class to assess class balance or imbalance and verify that the data is correctly labeled and categorized.

- Visualize a few sample images from both classes (tumorous and non-tumorous) to get an initial understanding of the data

- Perform any necessary preprocessing steps such as resizing images to a uniform size, converting images to grayscale, or normalizing pixel values.

- Visualize image features using techniques like edge detection, contour plotting, or image segmentation.

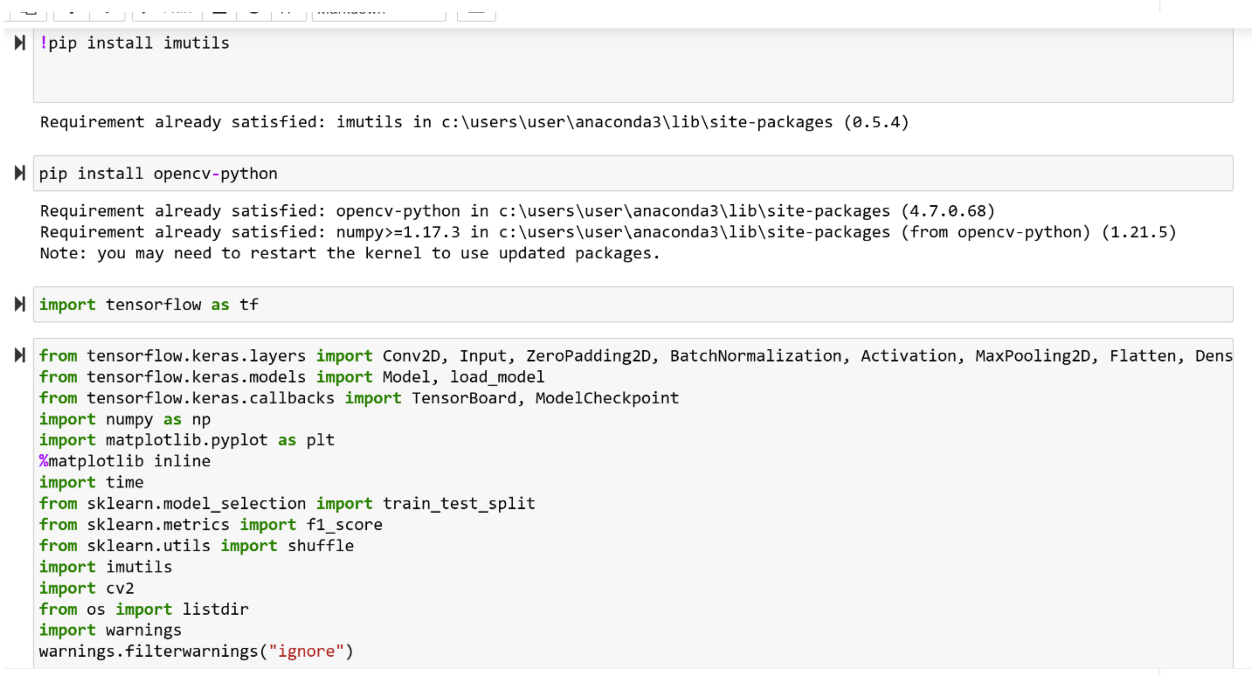
- Split the dataset into training and validation sets for model training and evaluation and choose appropriate evaluation metrics (e.g., accuracy, precision, recall, F1-score) for assessing model performance

7. Methodology

The methodology of this project includes different steps which are crucial for the betterment of the whole process. This section outlines systematic approach to achieve the project objectives.

Importing packages:

This step imports all of the necessary packages that we need to begin with our project.



```
!pip install imutils

Requirement already satisfied: imutils in c:\users\user\anaconda3\lib\site-packages (0.5.4)
```

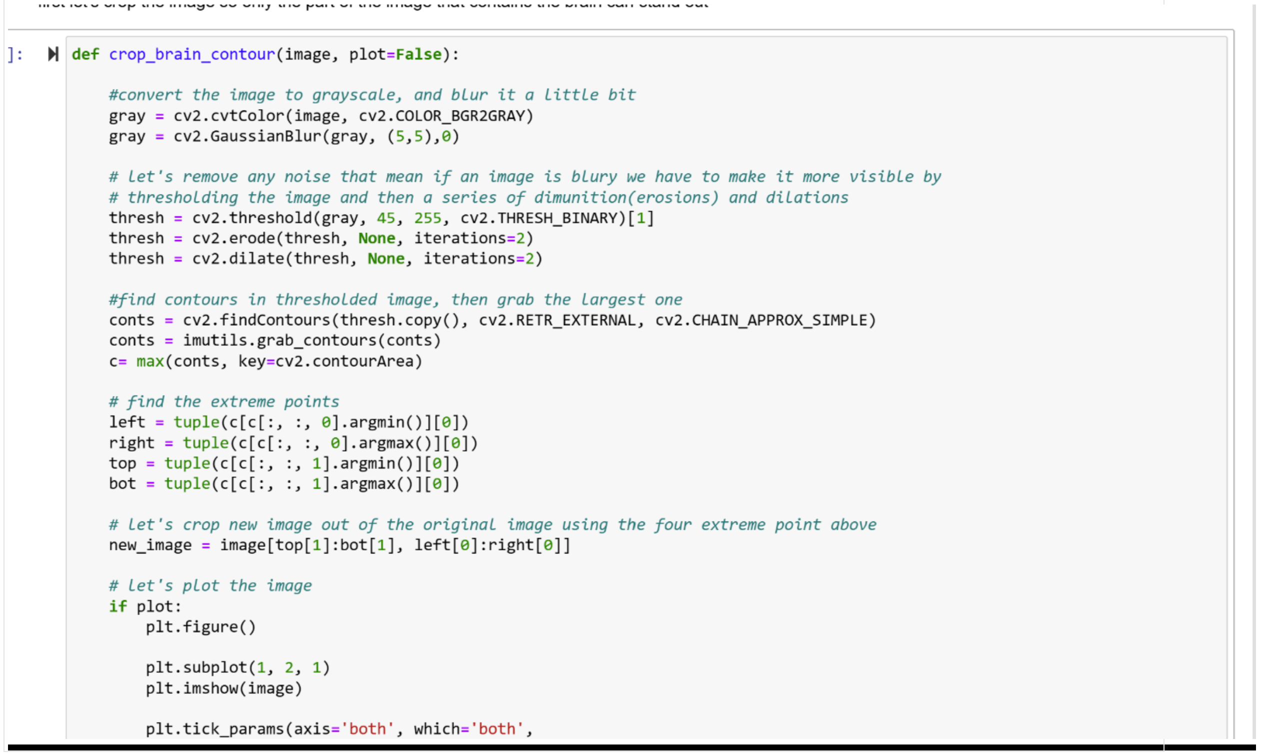
```
pip install opencv-python

Requirement already satisfied: opencv-python in c:\users\user\anaconda3\lib\site-packages (4.7.0.68)
Requirement already satisfied: numpy>=1.17.3 in c:\users\user\anaconda3\lib\site-packages (from opencv-python) (1.21.5)
Note: you may need to restart the kernel to use updated packages.
```

```
import tensorflow as tf
```

```
from tensorflow.keras.layers import Conv2D, Input, ZeroPadding2D, BatchNormalization, Activation, MaxPooling2D, Flatten, Dens
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.utils import shuffle
import imutils
import cv2
from os import listdir
import warnings
warnings.filterwarnings("ignore")
```

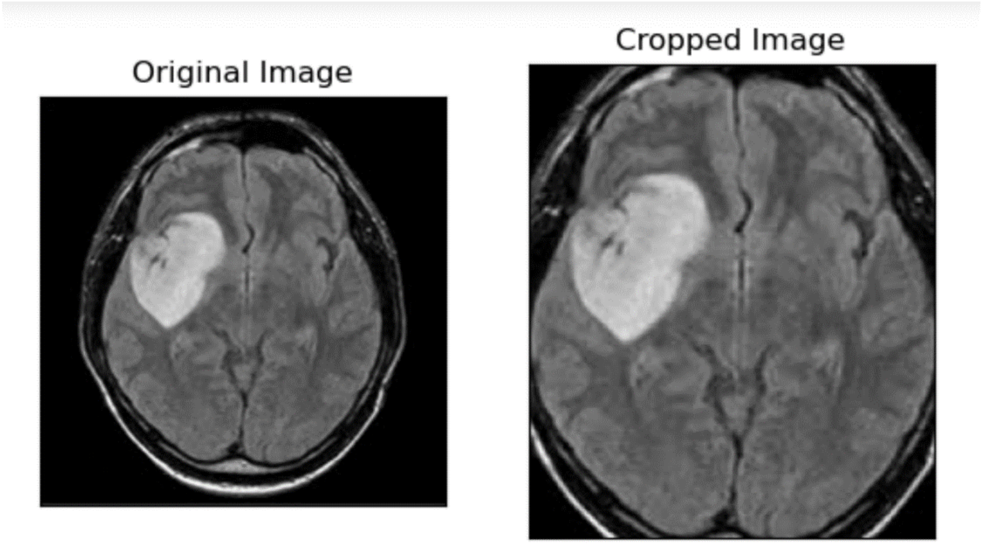
Data Preprocessing:

This section crops the image so that only the part of the image that contains the brain can stand out



```
]:  def crop_brain_contour(image, plot=False):

        #convert the image to grayscale, and blur it a little bit
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        gray = cv2.GaussianBlur(gray, (5,5),0)

        # let's remove any noise that mean if an image is blury we have to make it more visible by
        # thresholding the image and then a series of dimunition(erosions) and dilations
        thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
        thresh = cv2.erode(thresh, None, iterations=2)
        thresh = cv2.dilate(thresh, None, iterations=2)

        #find contours in thresholded image, then grab the largest one
        conts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        conts = imutils.grab_contours(conts)
        c= max(conts, key=cv2.contourArea)

        # find the extreme points
        left = tuple(c[c[:, :, 0].argmin()][0])
        right = tuple(c[c[:, :, 0].argmax()][0])
        top = tuple(c[c[:, :, 1].argmin()][0])
        bot = tuple(c[c[:, :, 1].argmax()][0])

        # let's crop new image out of the original image using the four extreme point above
        new_image = image[top[1]:bot[1], left[0]:right[0]]

        # let's plot the image
        if plot:
            plt.figure()

            plt.subplot(1, 2, 1)
            plt.imshow(image)

            plt.tick_params(axis='both', which='both',
```

Below is an example of a original image and cropped image:

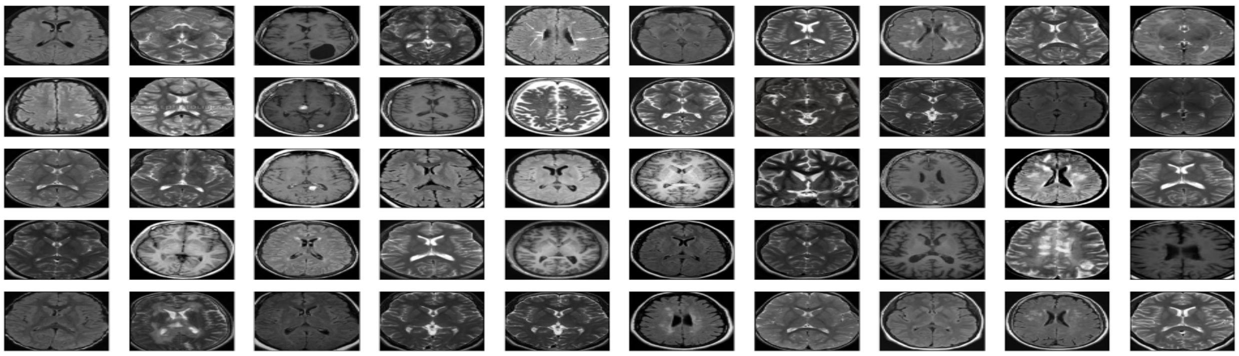


Data Loading:

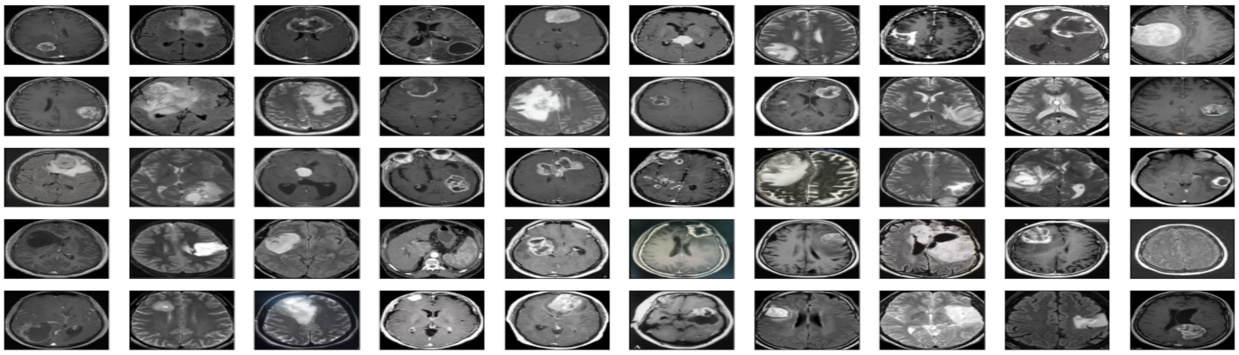
We load our data by resizing our images so they can have all the exact sizes knowing they all have different sizes. This code will read images, resize them, and normalize them meaning the image's pixel values will be scaled to the range 0 and 1.

```python
# so our function will contain two parameters one for directory path and the other onw for the size
def load_data(dir_list, image_size):
    # let's load all the image in our directory
    X = []
    y = []
    image_width, image_height = image_size
    for directory in dir_list:

        for filename in listdir(directory):
            # load images
            image = image = cv2.imread(directory + '\\' + filename)


            # let's crop the brain from the entire image
            image = crop_brain_contour(image, plot=False)
            # let's resize the images
            image = cv2.resize(image, dsize=(image_width, image_height), interpolation=cv2.INTER_CUBIC)
            # normalize values
            image = image / 255.
            # convert image to numpy array and append it to X
            X.append(image)
            # if the image is in the named 'yes', let's appenda value of one to the target array
            # otherwise append 0
            if directory[-3:] == 'yes':
                y.append([1])
            else:
                y.append([0])
    X=np.array(X)
    y=np.array(y)

    # we don't want our data to be in order, let's shuffle it
    X, y = shuffle(X, y)

    print(f'Number of example is: {len(X)}')
    print(f'X shape is: {X.shape}')
    print(f'y shape is: {y.shape}')
```

Example of our sample images from the files:



Brain Tumor: Yes



CNN Model Architecture :

```python
def build_model(input_shape):
    """
    Arugments:
        input_shape: A tuple representing the shape of the input of the model. shape=(image_width, image_height, #_channels)
    Returns:
        model: A Model object.
    """
    #Define the placeholder as a tensor with shape input_shape
    X_input = Input(input_shape)

    # let's pad the border of x_input with zeroes
    X = ZeroPadding2D((2,2))(X_input)

    # conv -> BN -> RELU Block applied to x
    X = Conv2D(32, (7,7), strides = (1,11), name = 'conv0')(X)
    X = BatchNormalization(axis = 3, name = "bn0")(X)
    X = Activation('relu')(X)

    # Maxpool
    X = MaxPooling2D((4,4), name='max_pool0')(X)

    # Maxpool
    X = MaxPooling2D((4,4), name='max_pool1')(X)

    #Flatten X
    X = Flatten()(X)
    #FullyConnected
    X = Dense(1, activation='sigmoid', name='fc')(X)

    #Let's create our keras model instance to use the instance to train/test our model
    model = Model(inputs = X_input, outputs = X, name='BrainDetectionModel')

    return model
```

Summary of our model:

```
model.summary()
```

```
Model: "BrainDetectionModel"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 240, 240, 3)]     0

 zero_padding2d (ZeroPadding  (None, 244, 244, 3)      0
 2D)

 conv0 (Conv2D)              (None, 238, 22, 32)       4736

 bn0 (BatchNormalization)    (None, 238, 22, 32)       128

 activation (Activation)     (None, 238, 22, 32)       0

 max_pool0 (MaxPooling2D)    (None, 59, 5, 32)         0

 max_pool1 (MaxPooling2D)    (None, 14, 1, 32)         0

 flatten (Flatten)           (None, 448)               0

 fc (Dense)                  (None, 1)                 449

=================================================================
Total params: 5,313
Trainable params: 5,249
Non-trainable params: 64
```

Splitting our data into train, val, test:

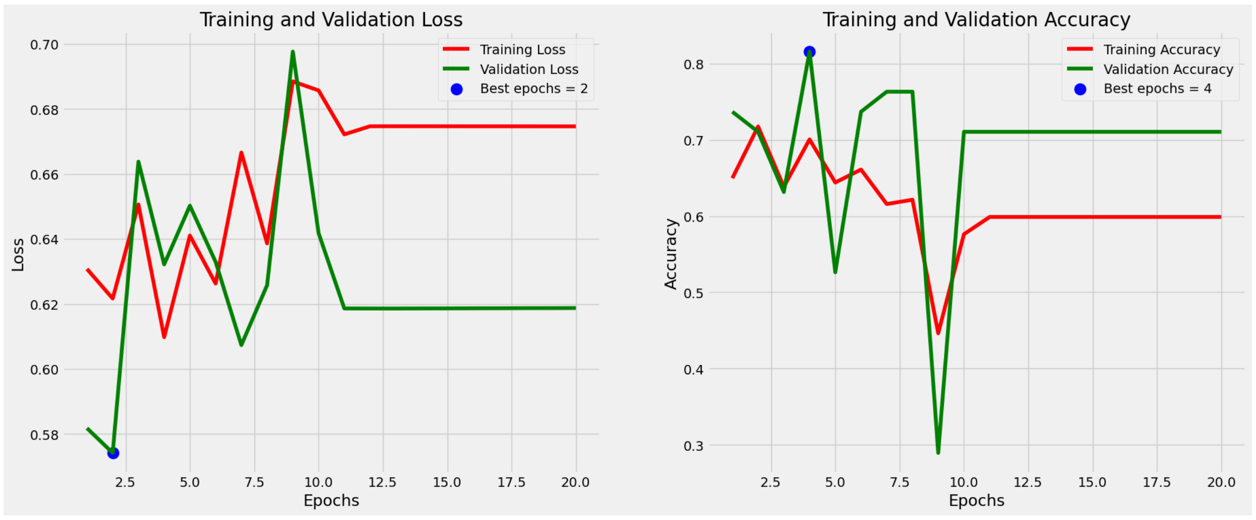
```
X_train, y_train, X_val, y_val, X_test, y_test = split_data(X, y, test_size=0.3)
```

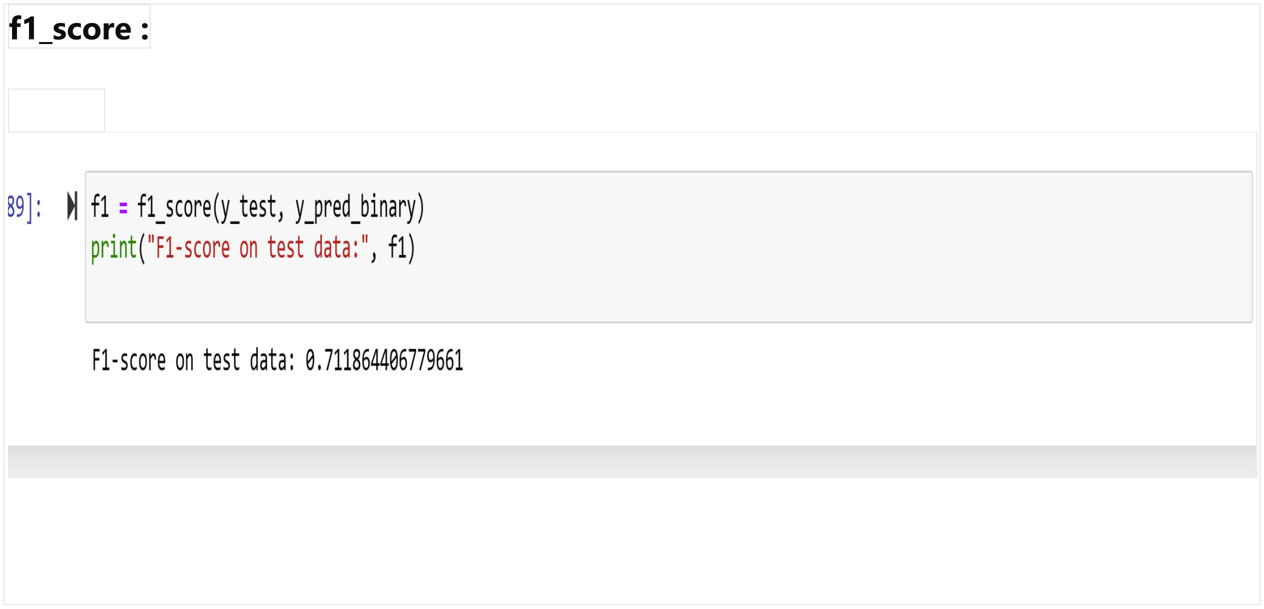
```
print ("number of training examples = " + str(X_train.shape[0]))
print ("number of development examples = " + str(X_val.shape[0]))
print ("number of test examples = " + str(X_test.shape[0]))
print ("X_train shape: " + str(X_train.shape))
print ("Y_train shape: " + str(y_train.shape))
print ("X_val (dev) shape: " + str(X_val.shape))
print ("Y_val (dev) shape: " + str(y_val.shape))
print ("X_test shape: " + str(X_test.shape))
print ("Y_test shape: " + str(y_test.shape))
```

```
number of training examples = 177
number of development examples = 38
number of test examples = 38
X_train shape: (177, 240, 240, 3)
Y_train shape: (177, 1)
X_val (dev) shape: (38, 240, 240, 3)
Y_val (dev) shape: (38, 1)
X_test shape: (38, 240, 240, 3)
Y_test shape: (38, 1)
```

8. Results

Our results indicate that our model after the training process the validation score is 81.58%. it means that when the model was evaluated on previously unseen data from the validation set, it correctly classified brain MRI images as either tumor-positive or tumor-negative with an accuracy of approximately 81.58%. we do understand while an accuracy of 81.58% may be satisfactory for some applications, it may not be sufficient for others. Therefore, further refinement of the model may be necessary to improve its performance.





f1_score :

```
89]:  f1 = f1_score(y_test, y_pred_binary)
      print("F1-score on test data:", f1)

      F1-score on test data: 0.711864406779661
```

9. Difference from other works:

Our model utilizes a carefully optimized training strategy, combining techniques such as stochastic gradient descent (SGD) with momentum, learning rate scheduling, and batch normalization. These strategies help stabilize training, accelerate convergence, and prevent overfitting, resulting in improved model performance.
We employ advanced data preprocessing techniques tailored to brain MRI images, including intensity normalization, skull stripping, and image registration. These preprocessing steps enhance the quality and consistency of input data, improving the model's ability to detect tumor features.

10. Conclusion

In conclusion, our project aims to improve brain tumor detection using machine learning and MRI scans. By developing a reliable classification model. Utilizing a dataset containing MRI images labeled with tumor presence, we trained a sophisticated classification model to accurately identify tumors based on imaging features. By harnessing the capabilities of deep learning techniques, particularly convolutional neural networks (CNNs), we sought to predict tumor presence with high precision and recall. Through extensive experimentation and validation, our model achieved a validation accuracy of 81.58%. During our training, the first validation score was 50 % but through experiment and hypertuning our parameters, we achieved a score above 80.