Brain Tumor Detection from MRI Images

Problem Statement

Brain tumor identification is really challenging task in the early stages of life. But now it became advanced with various machine learning algorithms. Now a day's issue of brain tumor automatic identification is of great interest. A tumor is the unusual growth of the tissues. A brain tumor is a number of unnecessary cells growing in the brain or central spine canal.
It is the unrestrained progress of cancer cells in any portion of the body. In Order to detect the brain tumor of a patient, we
consider the data of patients like MRI images of a patient's brain. Here our problem is to identify whether the tumor is present in the patient's brain or not. It is very important to detect the tumors at the starting level for a healthy life of a patient.

Dataset

The data set consists of two different folders that are Yes or No. Both the folders contain different MRI images of the patients. Yes folder has patients that have brain tumors whereas No folder has MRI images of patients with no brain tumor. There are a total of 155 images of positive patients of brain tumor and 98 images of other patients having no brain tumor. All the images are of 240X240 pixels.

Downloading Dataset from Kaggle

We first need to install the dependencies. As we will import data directly from Kaggle we need to install the package that supports that. So we have installed the Kaggle package using pip.

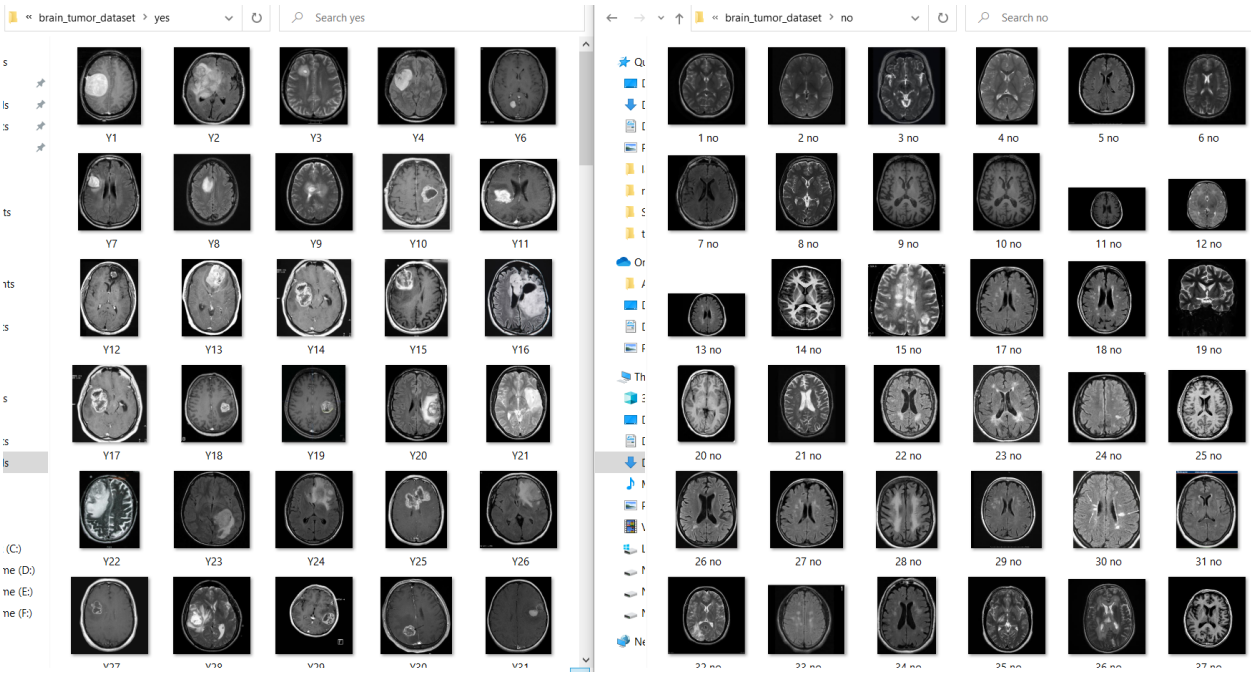
```
!pip install kaggle
```

Now we will import data from Kaggle. To do so we need to first add a kaggle. json file which you will get by creating a new API token on Kaggle. Go to my account in Kaggle and scroll down and you will see an option for creating a new API. Once you click on that a file ‘kaggle.json’ will be downloaded. Once you have that file upload it and change the permissions using the code shown below.

```
from google.colab import files

files.upload()

!mkdir -p ~ / .kaggle

!cp kaggle.json ~ / . kaggle/

! chmod 600 ~ / . kaggle/kaggle.json

kaggle datasets download -d navoneel/brain-mri - images - for - brain - tumor - detection
```

Once we run the above command the zip file of the data would be downloaded. We now need to unzip the file using the below code.

```
from zipfile import ZipFile

file_name = "/content/brain-mri-images-for-brain-tumor-detection.zip"

with ZipFile(file_name,'r') as zip:

zip.extractcall()

print('done')
```



Data Augmentation:

Since this is a small dataset, There wasn't enough examples to train the neural network. Also, data augmentation was useful in taclking the data imbalance issue in the data.

Further explanations are found in the Data Augmentation notebook.

Before data augmentation, the dataset consisted of:
155 positive and 98 negative examples, resulting in 253 example images.

After data augmentation, now the dataset consists of:
1085 positive and 980 examples, resulting in 2065 example images.

Note: these 2065 examples contains also the 253 original images. They are

found in folder named 'augmented data'.

In [2]:
```
def augment_data(file_dir, n_generated_samples, save_to_dir):
    data_gen = ImageDataGenerator(rotation_range=10,
                                  width_shift_range=0.1,
                                  height_shift_range=0.1,
                                  shear_range=0.1,
                                  brightness_range=(0.3, 1.0),
                                  horizontal_flip=True,
                                  vertical_flip=True,
                                  fill_mode='nearest'
                                 )


    for filename in listdir(file_dir):
        # load the image
        image = cv2.imread(file_dir + '\\' + filename)
        # reshape the image
        image = image.reshape((1,)+image.shape)
        # prefix of the names for the generated sampels.
        save_prefix = 'aug_' + filename[:-4]
        # generate 'n_generated_samples' sample images
        i=0
        for batch in data_gen.flow(x=image, batch_size=1, save_to_dir=save_to_dir,
                                           save_prefix=save_prefix, save_format='jpg'):
            i += 1
            if i > n_generated_samples:
                break
```

In [5]:
```
augmented_data_path = 'augmented data/'
# augment data for the examples with label equal to 'yes' representing tumurous examples
augment_data(file_dir='brain_tumor_dataset/yes', n_generated_samples=6, save_to_dir=augmented_data_path+'yes')
# augment data for the examples with label equal to 'no' representing non-tumurous examples
augment_data(file_dir='brain_tumor_dataset/no', n_generated_samples=9, save_to_dir=augmented_data_path+'no')
```

Data Preprocessing

For every image, the following preprocessing steps were applied:

1. Crop the part of the image that contains only the brain (which is the most important part of the image).
2. Resize the image to have a shape of (240, 240, 3)=(image_width, image_height, number of channels): because images in the dataset come in different sizes. So, all images should have the same shape to feed it as an input to the neural network.
3. Apply normalization: to scale pixel values to the range 0-1.

Loading and pre-processing the data: Data is gold as far as deep learning models are concerned. Your image classification model has a far better chance of performing well if you have a good amount of images in the

training set. Also, the shape of the data varies according to the architecture/framework that we use. Hence, the critical data pre-processing step (the eternally important step in any project). I highly recommend going through the “basics of image processing using Python we use Keras’ ImageDataGenerator class to perform data augmentation. i.e, we are using some kind of parameters to process our collected data. The word “augment” means to make something “greater” or “increase” something (in this case, data), the Keras ImageDataGenerator class actually works by: ü Accepting a batch of images used for training. ü Taking this batch and applying a series of random transformations to each image in the batch (including random rotation, resizing, shearing, etc.). ü Replacing the original batch with the new, randomly transformed batch. ü Training the CNN on this randomly transformed batch (i.e., the original data itself is not used for training). Note: The ImageDataGenerator accepts the original data, randomly transforms it, and returns only the new, transformed data.

1. Import the library
2. Define the parameters /arguments for ImageDataGenerator class
3. Applying ImageDataGenerator functionality to trainset and testset

Loading the dataset

In [132]:
```
from keras.preprocessing.image import ImageDataGenerator
```

Data Preprocessing

In [133]:
```
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,horizontal_flip=True,zoom_range=0.2)
#1./255->1 to 255
test_datagen=ImageDataGenerator(rescale=1./255)
```

In [134]:
```
x_train=train_datagen.flow_from_directory('brain_tumor_dataset/train',target_size=(64,64),batch_size=32,class_mode='binary')
x_test=test_datagen.flow_from_directory('brain_tumor_dataset/test',target_size=(64,64),batch_size=32,class_mode='binary')
# more than 2 categories -> class_mode='categorical'
```

```
Found 157 images belonging to 2 classes.
Found 96 images belonging to 2 classes.
```

In [135]:
```
print(x_train.class_indices)
```

```
{'no': 0, 'yes': 1}
```

In [136]:
```
len(x_train),len(x_test)
```

Out[136]:
```
(5, 3)
```

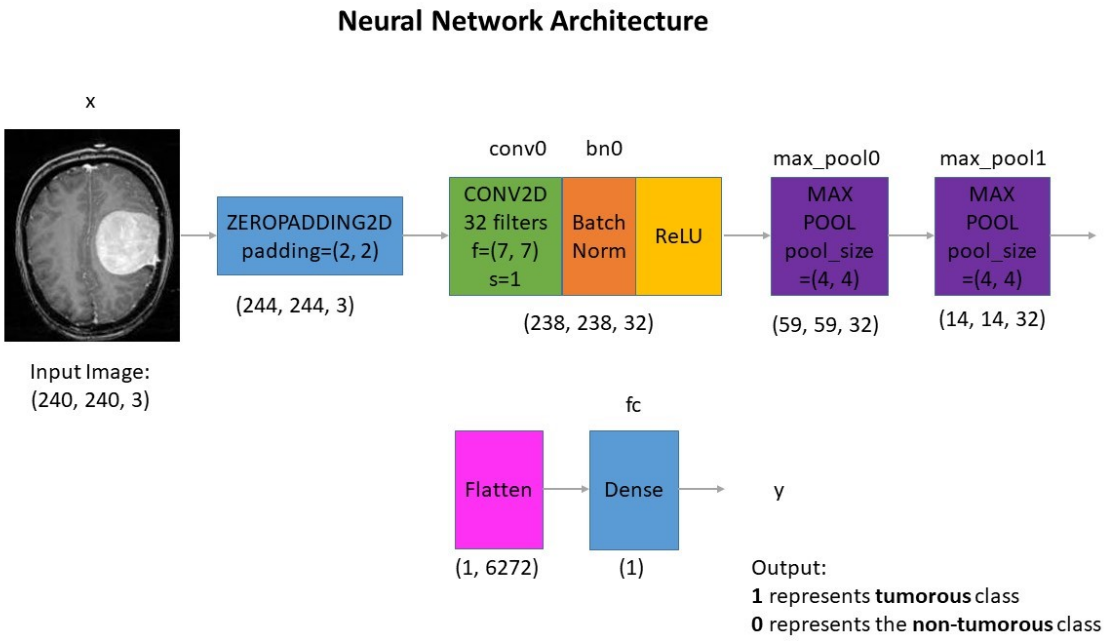
Data Split

The data was split in the following way:

1. 70% of the data for training.
2. 15% of the data for validation.
3. 15% of the data for testing.

Neural Network Architecture

This is the architecture that I've built:



Understanding the architecture:

Each input x (image) has a shape of (240, 240, 3) and is fed into the neural network. And, it goes through the following layers:

1. A Zero Padding layer with a pool size of (2, 2).
2. A convolutional layer with 32 filters, with a filter size of (7, 7) and a stride equal to 1.
3. A batch normalization layer to normalize pixel values to speed up

computation.

4. A ReLU activation layer.
5. A Max Pooling layer with f=4 and s=4.
6. A Max Pooling layer with f=4 and s=4, same as before.
7. A flatten layer in order to flatten the 3-dimensional matrix into a one-dimensional vector.
8. A Dense (output unit) fully connected layer with one neuron with a sigmoid activation (since this is a binary classification task).

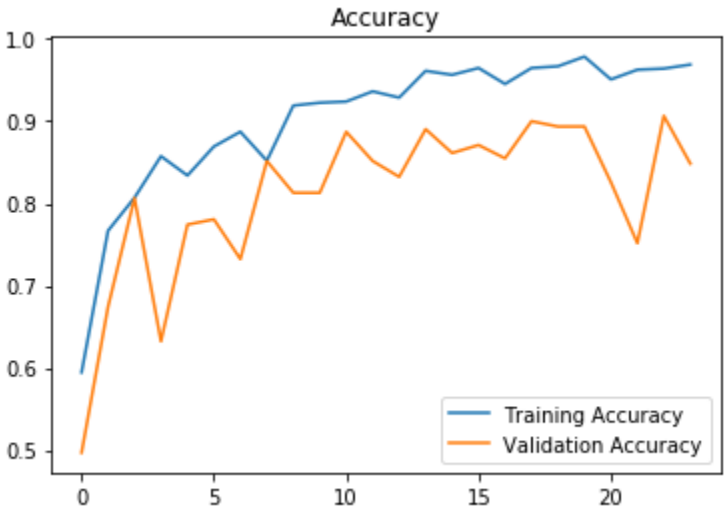
Why this architecture?

Firstly, I applied transfer learning using a ResNet50 and vgg-16, but these models were too complex to the data size and were overfitting. Of course, you may get good results applying transfer learning with these models using data augmentation. But, I'm using training on a computer with 6th generation Intel i7 CPU and 8 GB memory. So, I had to take into consideration computational complexity and memory limitations.

So why not try a simpler architecture and train it from scratch. And it worked :)

Training the model

The model was trained for 24 epochs and these are



the loss & accuracy plots:

The best validation accuracy was achieved on the 23rd iteration.

Building A CNN model Steps to Build a Deep Learning Model

1. Defining the model architecture

2. Configure the learning process

3. Train The Model

4. Save the Model

5. Predictions Step

Step 1: Defining Model architecture: For image classification we use Convolution neural network This is a very crucial step in our deep learning model building process.

We have to define how our model will look and that requires

· Importing the libraries

· Initializing the model

· Adding CNN (Convolution Neural Network) Layers

· Adding Dense layers v Importing the Libraries: v Initializing the model: Keras has 2 ways to define a neural network:

· Sequential

· Function API The Sequential class is used to define a linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add() method.

v Adding a CNN Layers: We will be adding three layers for CNN

· Convolution layer

· Pooling layer

· Flattening layer

Please refer to the below link for the description of these three layers https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add v

Adding Dense Layers: Please refer to the link for the description: https://skymind.ai/wiki/neural-network Once you are done with initializing

all the layers you have to pre-process the images.

Step 2: Configuring the learning process: With both the training data defined and model defined, it's time configure the learning process. This is accomplished with a call to the compile() method of the Sequential model class. Compilation requires 3 arguments: an optimizer, a loss function, and a list of metrics. In our example, set up as a multi-class classification problem, we will use the Adam optimizer, the categorical cross entropy loss function, and include solely the accuracy metric.

Building the Model

Step1: Import Libraries

In [120]:
```
from tensorflow.keras.models import Sequential #for initialising models
from tensorflow.keras.layers import Dense # adding layers
from tensorflow.keras.layers import Conv2D #convolution layer
from tensorflow.keras.layers import MaxPool2D #max pooling
from tensorflow.keras.layers import Flatten #Flatten layer
```

Step2:Initializing the model

In [121]:
```
model=Sequential()
```

Step3:Adding Convolution layer

In [122]:
```
model.add(Conv2D(32,3,3,input_shape=(64,64,3),activation='relu')) #adding
# 32->no.of feature detector
#3,3->size of feature detector 3x3 matrix
#input_shape -> Expected input shape of image
#all images should be in same shape
#3D->RGB Images 64x64 one
#activation function ->relu for layers
```

In [123]:
```
model.add(Conv2D(32,3,3,input_shape=(64,64,3),activation='relu'))
```

In [124]:
```
model.add(Conv2D(32,3,3,input_shape=(64,64,3),activation='relu'))
```

Step4: Adding maxpool layer

In [125]:
```
model.add(MaxPool2D(pool_size=(2,2),name='n1'))
#pool matrix
```

Step5: Adding Flatten layer

In [126]:
```
model.add(Flatten())
#converts n dimension to 1 dimension
```

In [127]:
```
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_25 (Conv2D)           (None, 21, 21, 32)        896
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 7, 7, 32)          9248
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 2, 2, 32)          9248
_________________________________________________________________
n1 (MaxPooling2D)            (None, 1, 1, 32)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 32)                0
=================================================================
Total params: 19,392
Trainable params: 19,392
Non-trainable params: 0
_________________________________________________________________
```

Step6: ANN layers

In [128]:
```
model.add(Dense(units=128,activation='relu',kernel_initializer='random_uniform'))
```

In [129]:
```
model.add(Dense(units=1,activation='sigmoid',kernel_initializer='random_uniform'))
```

In [130]:
```
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_25 (Conv2D)           (None, 21, 21, 32)        896
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 7, 7, 32)          9248
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 2, 2, 32)          9248
_________________________________________________________________
n1 (MaxPooling2D)            (None, 1, 1, 32)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               4224
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 23,745
Trainable params: 23,745
```

Step7: Compiling the model

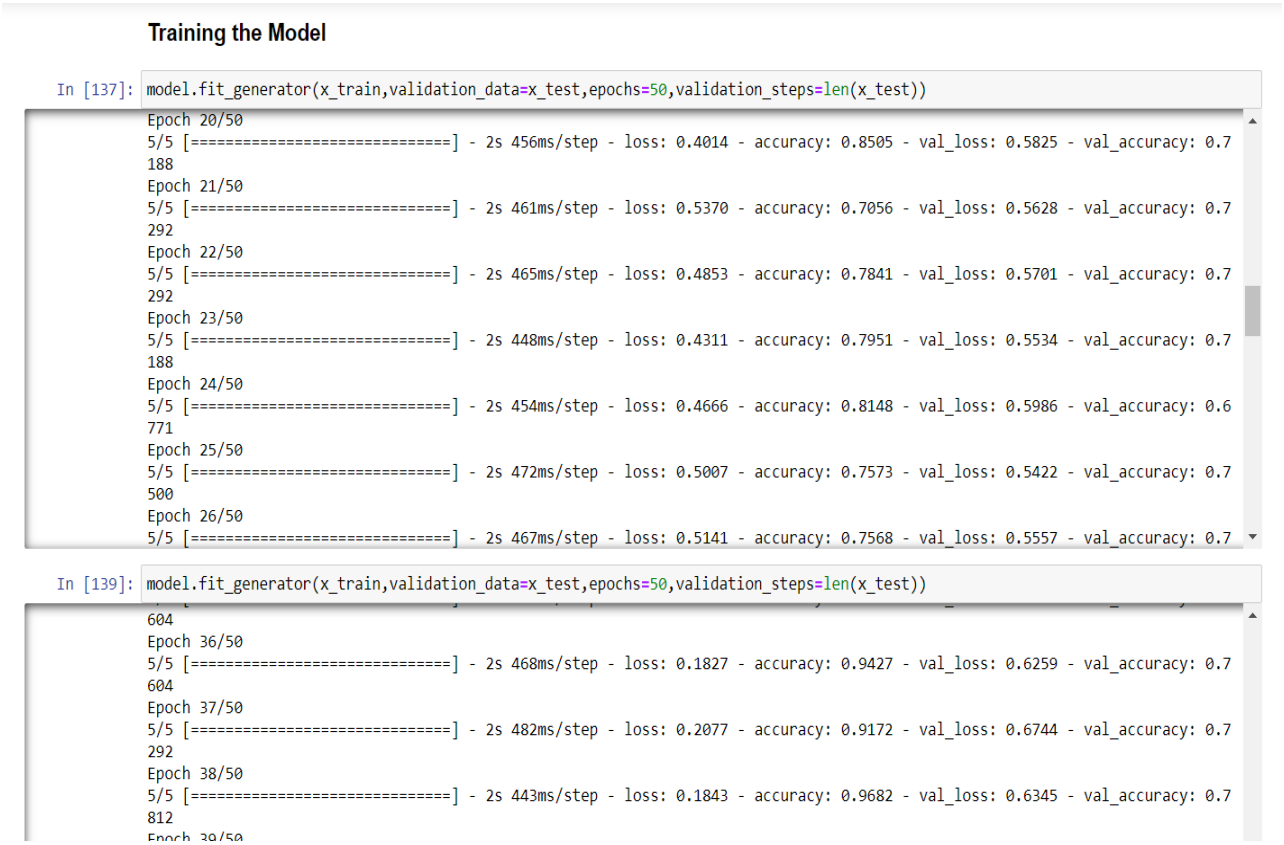
In [131]:
```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

Step 3: Train the model At this point we have training data and a fully configured neural network to train with said data. All that is left is to pass the data to the model for the training process to commence, a process which is completed by iterating on the training data. Training begins by calling the fit() method.

Note: This steps takes few minutes based on the epochs (no : of time you would like to train the machine with the given data set) you give .

Step 4: Save The Model: Your model is to be saved for the future purpose. This saved model ac also be integrated with android application or web application in order to predict something.

Step 5: Prediction: The last and final step is to make use of Saved model to do predictions. We use load model class to load the model. We use imread() class from opencv library to read an image and give it to the model to predict the result.



Save the Model

In [140]:
```
model.save('tumor.h5')
```

Predicting the Model

In [141]:
```
from tensorflow.keras.models import load_model
import cv2
import numpy as np
```

In [142]:
```
model=load_model('tumor.h5')
```

In [143]:
```
from skimage.transform import resize
```

In [144]:
```
def detect(frame):
    img=resize(frame,(64,64))
    img=np.expand_dims(img,axis=0)
    if(np.max(img))>1:
        img=img/255.0
    prediction=model.predict(img)
    print(prediction)
    if prediction>0.5:
        print('Yes')
    else:
        print('No')
```

In [145]:
```
frame=cv2.imread('brain_tumor_dataset/test/no/1 no.jpeg')
data=detect(frame)
```

```
[[0.07762507]]
No
```

Application Building

Flask Frame Work with Deep Learning Model

In this section we will be building a web application which is integrated to the model we built. An UI is provided for the uses where he has to upload a picture/photo. The uploaded photo is given to the model built, and prediction is showcased on the UI.

We are using a Deep Learning Model which is built for animal recognition and saved this file as “predict.h5”.

We have built our model using 5 classes To build this you should know basics of “HTML, CSS, Bootstrap, flask framework and python, Opencv”

Create a project folder which should contains

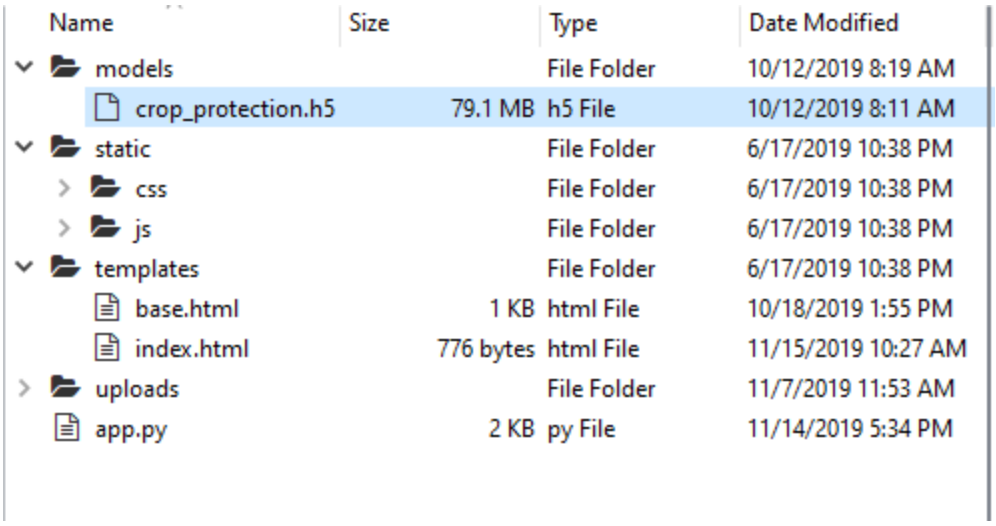
· An python fill called app.y

· Your Deep learning alogorithm file (for example animalrecognition.py or animal recognition.py.ipnby)

· Models folder with Model file (for example animal.h5)

· Templates folder which contains sample.HTML file

· Static folder which contains css folder which contains styles.css and js foder which contains js file

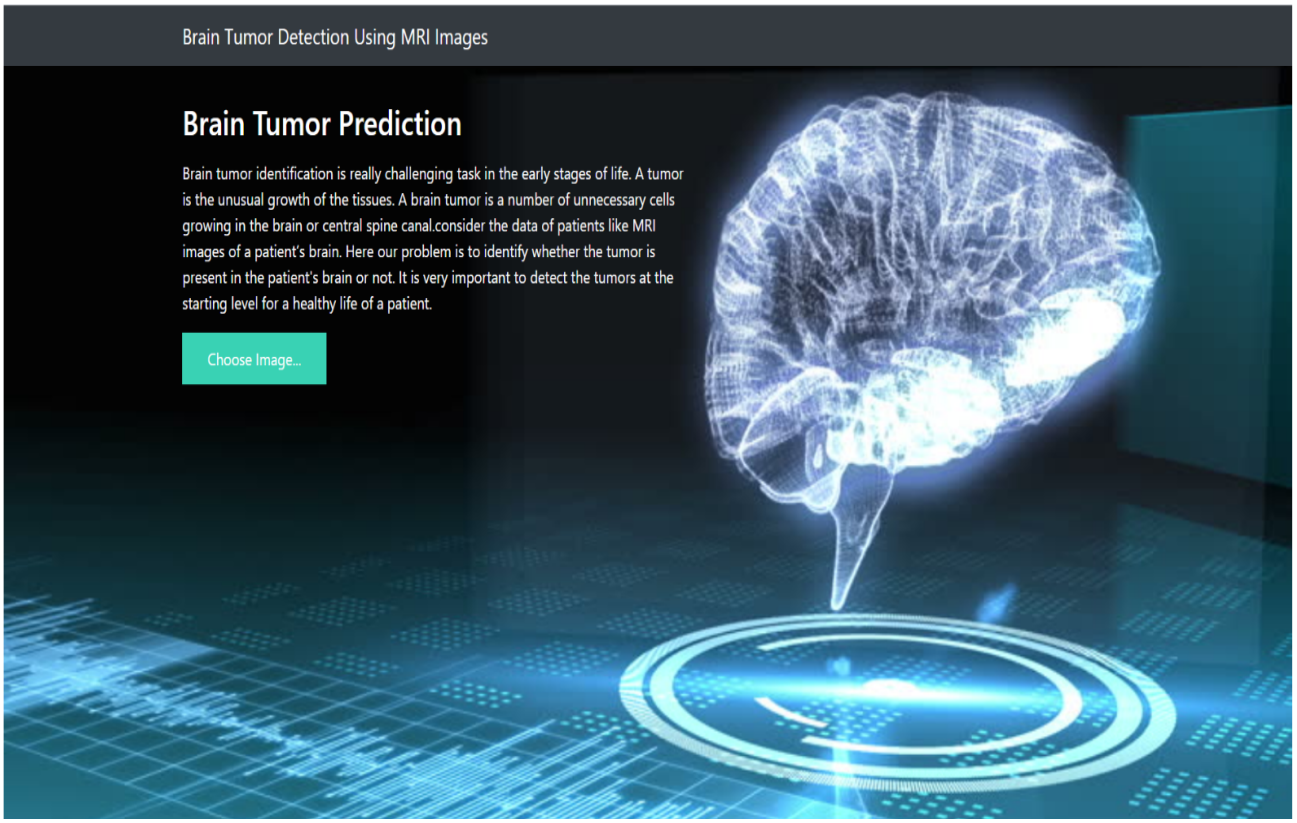


Steps 1: building an Index. Html file

This is the basic HTML page for our Project. Here we are creating two buttons one used to browse pictures from local drive and other button is used to send this picture to the model file which analyses the picture and show cases the prediction in the result. The browsed picture is displayed on the html page using on image preview using main.js script. You can see the scripted file is called in html page using src t

```
<body>
    <nav class="navbar navbar-dark bg-dark">
        <div class="container">
            <a class="navbar-brand" href="#">Brain Tumor Detection Using MRI Images</a>
        </div>
    </nav>
         <div class="container">
        <div id="content" style="margin-top:2em;color:white"><h2>Brain Tumor Prediction</h2>
        <div style=>
                <p style="margin-top:1em;width:600px">Brain tumor identification is really challenging task in the early stages of life. A tumor is the unusual growth of the
        </div>

<div>
    <form id="upload-file" method="post" enctype="multipart/form-data">
        <label for="imageUpload" class="upload-label">
            Choose Image...
        </label>
        <input type="file" name="file" id="imageUpload" accept=".png, .jpg, .jpeg">
    </form>

    <div class="image-section" style="display:none;">
        <div class="img-preview">
            <div id="imagePreview">
            </div>
        </div>
        <div>
            <button type="button" class="btn btn-primary btn-lg " id="btn-predict">Predict!</button>
        </div>
    </div>

    <div class="loader" style="display:none;"></div>

    <h3 id="result">
        <span> </span>
    </h3>

</div></div>
    </div>
```

HTML page output :



Step 2: Build python code

We will be using python for server side scripting. Let's see step by step process for writing backend code.

Importing Libraries

We are importing the libraries for processing the uploaded picture Loading the built model, saving the browsed pictures in uploads folder. And dependent library default graph for model prediction

Application Building

In [1]:
```
from __future__ import division, print_function
# coding=utf-8
import sys
import os
import glob
import numpy as np
from keras.preprocessing import image


from keras.applications.imagenet_utils import preprocess_input, decode_predictions

from keras.models import load_model
from keras import backend
from tensorflow.keras import backend

import tensorflow as tf
```

In [2]:
```
from skimage.transform import resize

# Flask utils
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer
```

Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of current module (__name__) as argument We are giving the path where our model file is located and from that path we are loading our model file

Routing to the html Page

Here we will be using declared constructor to route to the html page which we have created earlier.

In the above example, '/' URL is bound with index.html function. Hence, when the home page of web server is opened in browser, the html page will be rendered. Whenever you browse an image from the html page this photo can be accesses through POST or GET Method.

Showcasing prediction on UI

Here we are defining a function which request the browsed file from html page using post method. The requested picture file is then saved to the uploads folder in this same directory using OS library. Using load image class from Keras library we are retrieving the saved picture from the path declared. We are applying some image processing techniques and then sending that preprocessed image to the model for predicting the class. This

returns the numerical value of a class (like 0,1 ,2 etc.) which lies in the 0th index of the variable preds. This numerical value is passed to the index variable declared. This returns the name of the class. This name is rendered to the predict variable used in html page

Main Function

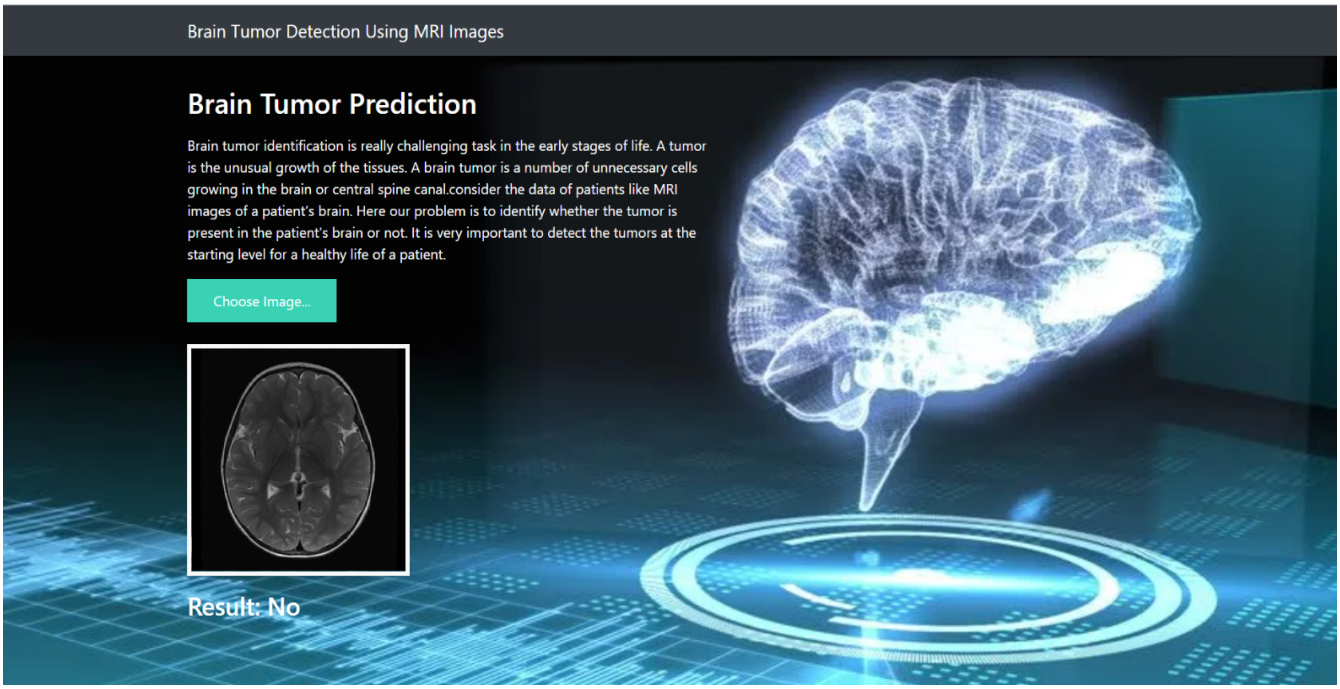
This is used to run the application in local host

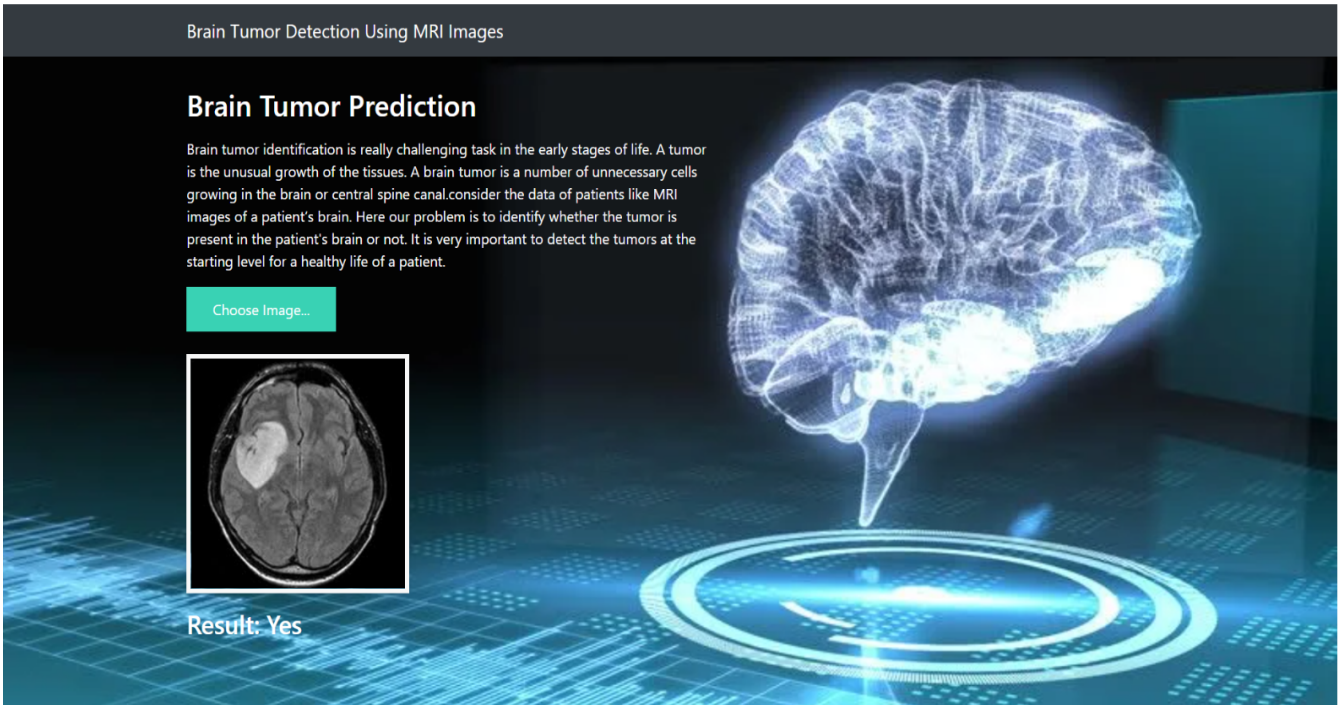
In [3]:
```
app = Flask(__name__)
MODEL_PATH = 'tumor.h5'
model = load_model(MODEL_PATH)
@app.route('/', methods=['GET'])
def index():
    # Main page
    return render_template('index.html')
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['file']

        # Save the file to ./uploads
        basepath = os.path.dirname('__file__')
        file_path = os.path.join(
            basepath, 'uploads', secure_filename(f.filename))
        f.save(file_path)
        img = image.load_img(file_path, target_size=(64,64))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)
        preds = model.predict(x)
        if preds>0.5:
            text='Yes'
        else:
            text='No'
        return text
```

In [4]:
```
if __name__ == '__main__':
    app.run(debug=False,threaded = False)
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
```

OUTPUT


Brain Tumor Detection Using MRI Images
Brain Tumor Prediction
Brain tumor identification is really challenging task in the early stages of life. A tumor is the unusual growth of the tissues. A brain tumor is a number of unnecessary cells growing in the brain or central spine canal.consider the data of patients like MRI images of a patient's brain. Here our problem is to identify whether the tumor is present in the patient's brain or not. It is very important to detect the tumors at the starting level for a healthy life of a patient.
Choose Image...
Result: No


Brain Tumor Detection Using MRI Images
Brain Tumor Prediction
Brain tumor identification is really challenging task in the early stages of life. A tumor is the unusual growth of the tissues. A brain tumor is a number of unnecessary cells growing in the brain or central spine canal.consider the data of patients like MRI images of a patient's brain. Here our problem is to identify whether the tumor is present in the patient's brain or not. It is very important to detect the tumors at the starting level for a healthy life of a patient.
Choose Image...
Result: Yes

Conclusion

Brain Tumor Detection is a medical domain related problem we have to convert it to Machine Learning related problem , for this we need data so we use a data set of Brain Tumor which has 155 ' YES ' MRI images and 98 ' NO ' MRI images . Performed data augmentation to improve efficiency. To build the model we have divided the model into train and test and train the model with CNN (Conventional Neural Networks) ,the accuracy of built model is 96% which is better than previous model . We also deployed web application with the help of flask and html pages to predict the Brain Tumor by giving input as MRI scan image .

Thanking you

Team Members
B.Sushma Sri
D.Lakshmi Priya
G.Mounika
Sk.Sajidabegum