PROJECT REPORT

(20211 Semester August-November 2021)

BRAIN TUMOR DETECTION

Submitted by:

Piyusha Sayal (18105074) Rhythm Mahajan (18105104)

Under the Guidance of:

Dr. Jasbir Kaur

Assistant Professor

ECE Department

Punjab Engineering College

(Deemed to be University)

Department of Electronics and Communication Engineering

Punjab Engineering College, Chandigarh

DECLARATION

We, hereby, declare that the project work entitled "**Brain Tumor Detection**" is an authentic record of our work carried out as a requirement of Neural Project for the award of the degree of B.E. Electronics and Communication Engineering, PEC University of Technology, Chandigarh, under the guidance of **Dr Jasbir Kaur** from September to November 2021.

Submitted by:
Piyusha Sayal (18105074)
Rhythm Mahajan (18105104)

Date: 18th November 2021

Certified that the above statement made by the students is correct to the best of my knowledge and belief.

Dr. Jasbir Kaur
Assistant Professor
ECE Department
Punjab Engineering College
(Deemed to be University)

ACKNOWLEDGEMENT

We avail this prospect to express our thoughtful sense of sincerity, earnestness and deep gratitude to all those who have performed a vital role in the achievement of the project "**Brain Tumor Detection**" by providing their guidance and support.

We wish to express our sincere obligation to our mentor **Dr Jasbir Kaur** for their incessant encouragement and inspiration throughout the project. Without their enduring spur, generous guidance and help, we would not have been able to cope up with the project work.

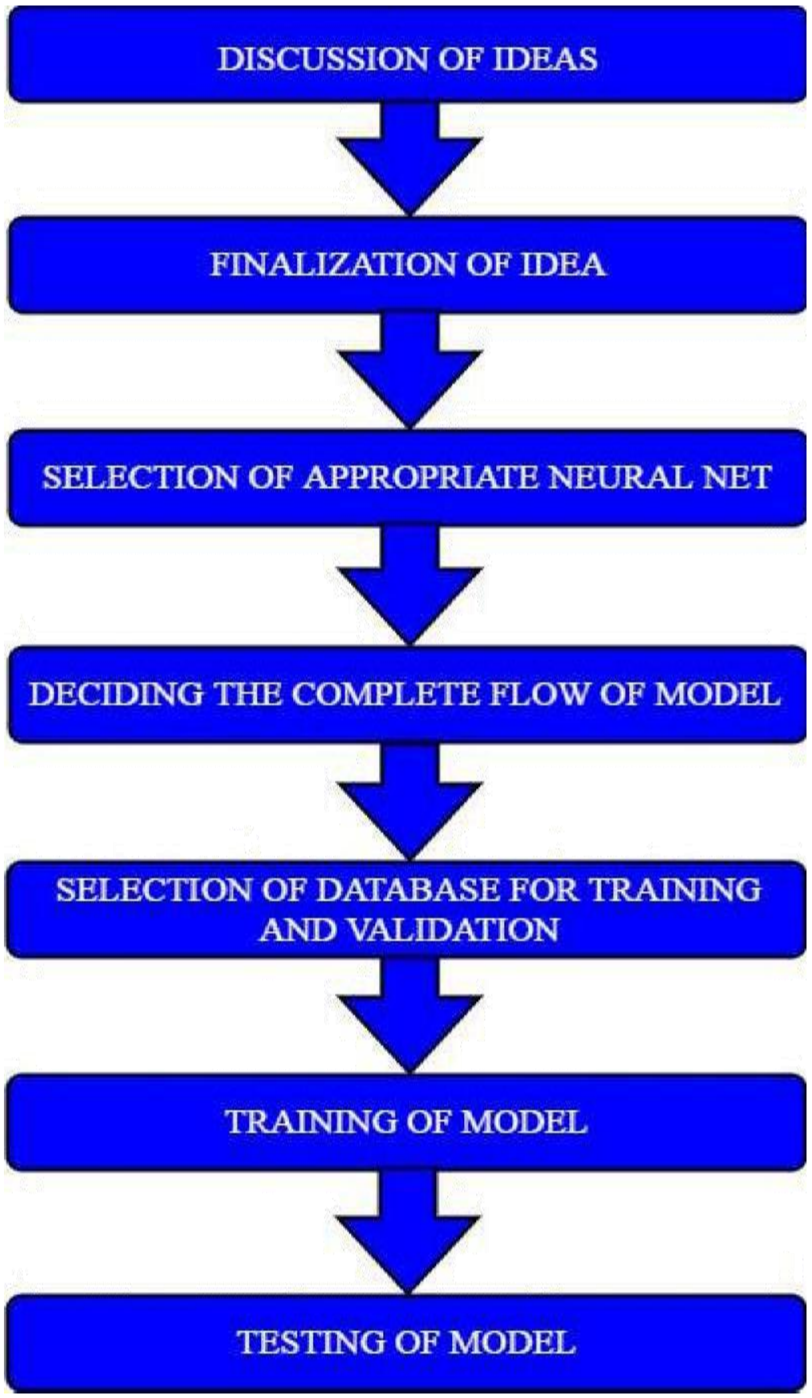
We are also grateful to the college and staff of the ECE Department, who taught the fundamentals and allowed us to take up this project which has been a great learning experience for us.

Submitted by:
Piyusha Sayal(18105074)
Rhythm Mahajan (18105104)

TABLE OF CONTENTS

1. ROADMAP

The roadmap followed for the project is as follows:


DISCUSSION OF IDEAS
FINALIZATION OF IDEA
SELECTION OF APPROPRIATE NEURAL NET
DECIDING THE COMPLETE FLOW OF MODEL
SELECTION OF DATABASE FOR TRAINING AND VALIDATION
TRAINING OF MODEL
TESTING OF MODEL

2. INTRODUCTION

Tumor is an abnormal growth of cells in the human brain. Tumor can be categorized as benign (non-cancerous) and malignant (cancerous). Earlier stages of tumor used to be detected manually through image observation by doctors which time consuming and might also give inaccurate results. Treatment of brain tumor depends on many factors such as proper diagnosis and different factors like the type of tumor, location, size, and state of development. There are many types of brain tumor and only expert doctors can able to give accurate results.

Image processing is among the most demanding and promising fields nowadays. Today many computer added tools are used in the medical field. These tools have the property of quick and accurate results.

We intend to use a Convolution Neural Network (CNN) based classifier for tumor detection based on image processing of MRI scans. CNN based classifier will be used to compare the trained and test data, in order to get time-efficient and accurate results.

2.1 TECHNOLOGIES USED

This section has an in-depth analysis of Python and its libraries used by us in the implementation of the project. Also, we have discussed the Integrated Development Environment (IDE) used to implement the project.

2.1.1 Python

Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming.

Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

Python 2.0 was released in 2000 and introduced new features, such as list comprehensions and a cycle-detecting garbage collection system.

Python 3.0, released in 2008, was a major revision of the language that is not completely backwards-compatible, and much Python 2 code does not run unmodified on Python 3.

2.1.2 TensorFlow

Created by the Google Brain team, TensorFlow is an open-source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient front-end API for building applications with the framework while executing those applications in high-performance C++. TensorFlow can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations. Best of all, TensorFlow supports production prediction at scale, with the same models used for training.

2.1.3 Keras

Keras is an open-source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model.

2.1.4 Google Colab

Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook.

3. PROBLEM FORMULATION

3.1 Model

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

3.1.1 Convolution Layer

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, colour, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network that has a wholesome understanding of images in the dataset, similar to how we would.

There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying Valid Padding in the case of the former, or the Same Padding in the case of the latter.

3.1.2 Pooling Layer

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model. There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling.

The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power. After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes.

3.1.3 Fully Connected Layer

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation is applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the Softmax Classification technique.

3.2 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, the folder has patients that have brain tumors whereas the 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 240X240 pixels.

Reference Link:
https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection

3.3 Algorithm

Convolution Neural Network (CNN) in Python is used to train our model using MNIST dataset.

CNN is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various objects in the image and be able to differentiate one from the other. The required pre-processing in CNN is much less than other classification algorithms.

List of python libraries used:

1. **Opencv**: OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

2. **Tensorflow:** TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

3. **Keras**: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.

4. **Numpy**: NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python

CNN

```
m1=Sequential()
m1.add(BatchNormalization(input_shape = (28,28,3)))
m1.add(Convolution2D(32, (3,3), activation ='relu', input_shape = (28, 28, 3)))
m1.add(MaxPooling2D(pool_size=2))
m1.add(Convolution2D(filters=64, kernel_size=4, padding='same', activation='relu'))
m1.add(MaxPooling2D(pool_size=2))
m1.add(Convolution2D(filters=128, kernel_size=3, padding='same', activation='relu'))
m1.add(MaxPooling2D(pool_size=2))
m1.add(Convolution2D(filters=128, kernel_size=2, padding='same', activation='relu'))
m1.add(MaxPooling2D(pool_size=2))
m1.add(Dropout(0.25))
m1.add(Flatten())
m1.add(Dense(units=128,activation = 'relu'))
m1.add(Dense(units = 64, activation = 'relu'))
m1.add(Dense(units = 32, activation = 'relu'))
m1.add(Dense(units = 2, activation = 'softmax'))
m1.compile(optimizer='adam', loss = 'categorical_crossentropy',metrics = ['accuracy'])
history = m1.fit(X_train,y_train,
                 epochs=50,
                 validation_data=(X_test,y_test),
                 verbose = 1,
                 initial_epoch=0)
```

4. METHODOLOGY

We will build a classification model that would take MRI images of the patient and compute if there is a tumor in the brain or not. We will be using Brain MRI Images for Brain Tumor Detection that is publicly available on Kaggle. We will first build the model using simple custom layers convolutional neural networks and then evaluate it. At last, we will compute some prediction by the model and compare the results. Automated brain tumor detection is very necessary as high accuracy is needed when human life is involved. Automated detection of tumor in MR images involves feature extraction and classification using a machine learning algorithm.

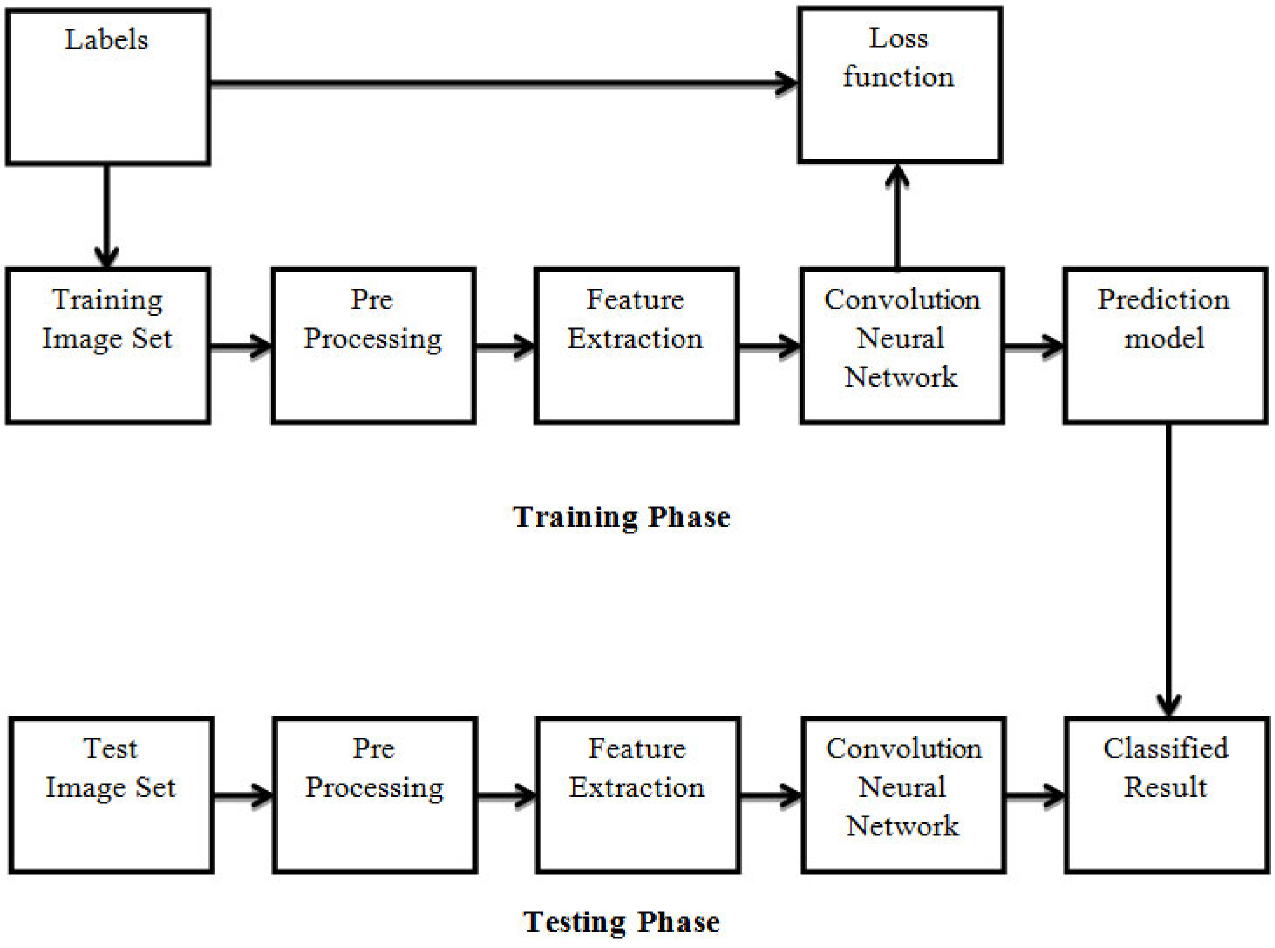


Figure 1: Block diagram of proposed brain tumor classification using CNN

The process took place in two-step i.e. training and testing phase. The training phase always takes place before the testing phase. The feature extraction and classification is done by a convolution neural network (CNN). Training image sets are used to train the model and testing datasets are used to validate the model. The loss function is used to improve the accuracy of the model. Less the value of loss function more accurately the prediction is done.

A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers.

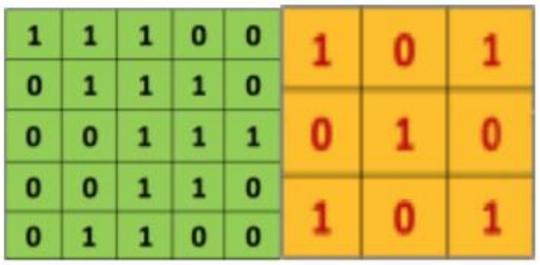
There are four main operations in the ConvNet:

1. Convolution
2. Non-Linearity (ReLU)
3. Pooling or Sub Sampling
4. Classification (Fully Connected Layer)

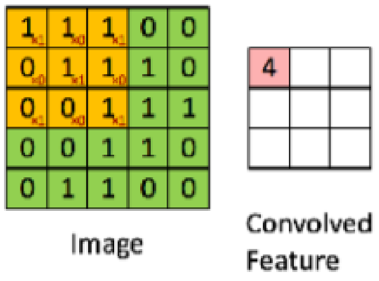
An Image is a matrix of pixel values. Essentially, every image can be represented as a matrix of pixel value Channel is a conventional term used to refer to a certain component of an image. An image from a standard digital camera will have three channels – red, green and blue – you can imagine those as three 2d-matrices stacked over each other (one for each colour), each having pixel values in the range 0 to 255.

4.1 Convolution

The primary purpose of Convolution in case of a ConvNet is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. Every image can be considered as a matrix of pixel values. Consider a 5 x 5 image whose pixel values are only 0 and 1 (note that for a grayscale image, pixel values range from 0 to 255, the green matrix below is a special case where pixel values are only 0 and 1.



Also, consider another 3 x 3 matrix as shown. Then, the Convolution of the 5 x 5 image and the 3 x 3 matrix can be computed as shown in the animation in Fig below:

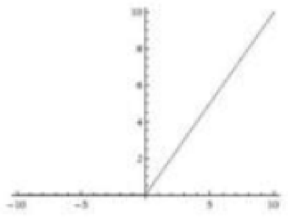


The output matrix is called Convolved Feature or Feature Map. In CNN terminology, the 3×3 matrix is called a 'filter' or 'kernel' or 'feature detector' and the matrix formed by sliding the filter over the image and computing the dot product is called the 'Convolved Feature' or 'Activation Map' or the 'Feature Map'. It is important to note that filters act as feature detectors from the original input image.

4.2 Non-Linearity (ReLU)

An additional operation called ReLU has been used after every Convolution operation. ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by:

Output = Max(zero, Input)



ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear.

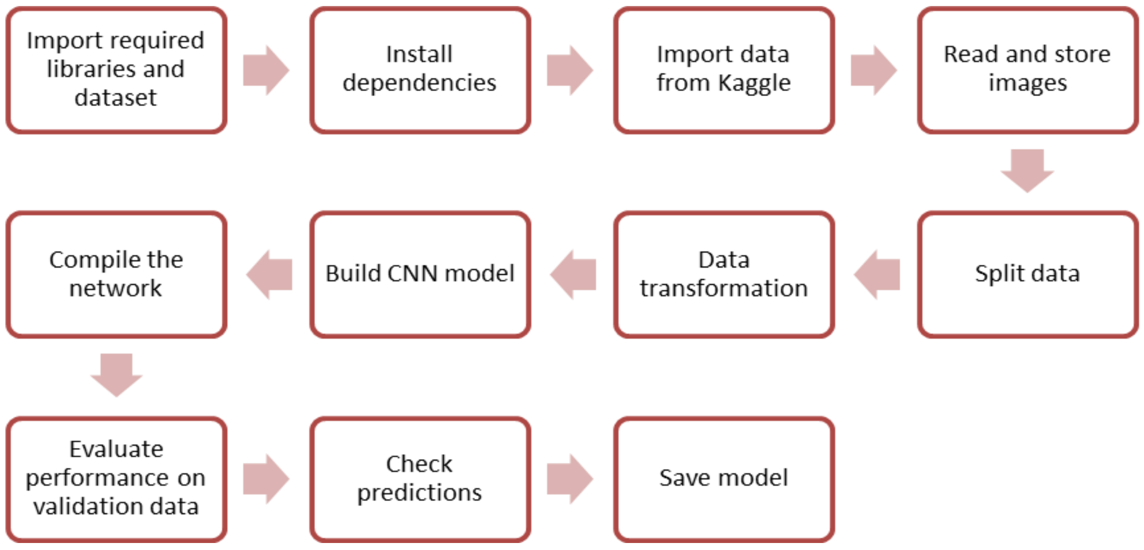
4.3 Pooling or Sub Sampling

Spatial Pooling (also called subsampling or down sampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc. In case of Max Pooling, we define a spatial neighborhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window.
In practice, Max Pooling has been shown to work better.

4.4 Classification (Fully Connected Layer)

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. By this we combine the leaning of cnn and then converting that leaning into probability using softmax function and then maximum probability digit is extracted.

FLOW DIAGRAM



5. RESULTS

We made a classification model with the help of custom CNN layers to classify whether the patient has a brain tumor or not through MRI images. With a few training samples, the model gave 89.29% accuracy.

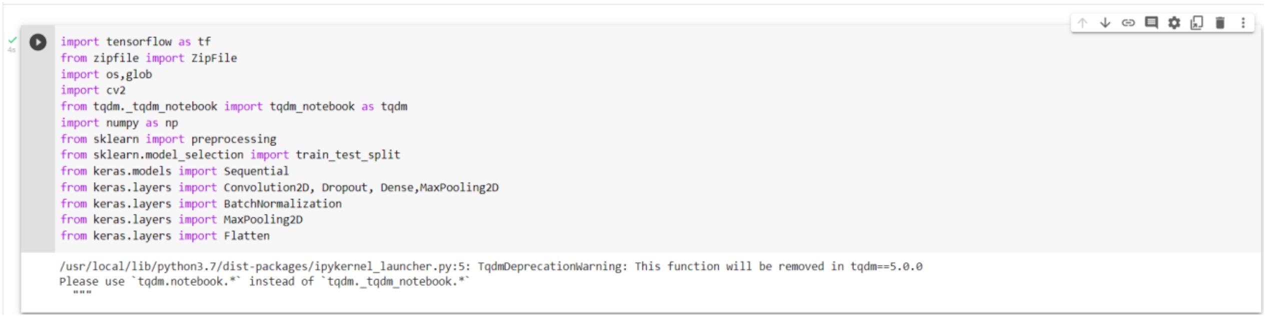
Our Dataset contains tumor and non-tumor MRI images and collected from kaggle. In this work, efficient automatic brain tumor detection is performed by using convolution neural network. Simulation is performed by using python language. The accuracy is calculated. The training accuracy, validation accuracy and validation loss are calculated to find the efficiency of proposed brain tumor classification scheme.

6. CONCLUSION AND FUTURE SCOPE

In summary, we propose a CNN-based method for the segmentation of brain tumors in MRI images. There are several existing techniques are available for brain tumor segmentation and classification to detect brain tumor. There are many techniques available present a study of existing techniques for brain tumor detection and their advantages and limitations. To overcome these limitations, propose a Convolution Neural Network (CNN) based classifier. CNN based classifier used to compare the trained and test data, from this get the best result.

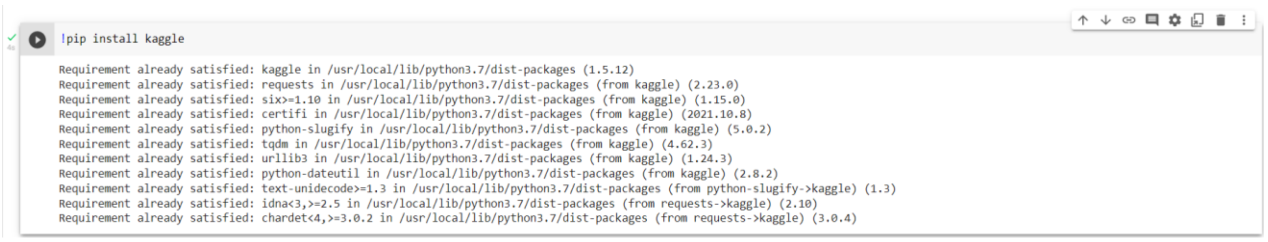
If we increase the training data maybe by more MRI images of patients or perform data augmentation techniques we can achieve higher classification accuracy. Also, we can make use of pre-trained architectures like Vgg16 or Resnet 34 for improving the model performance.

7. CODE



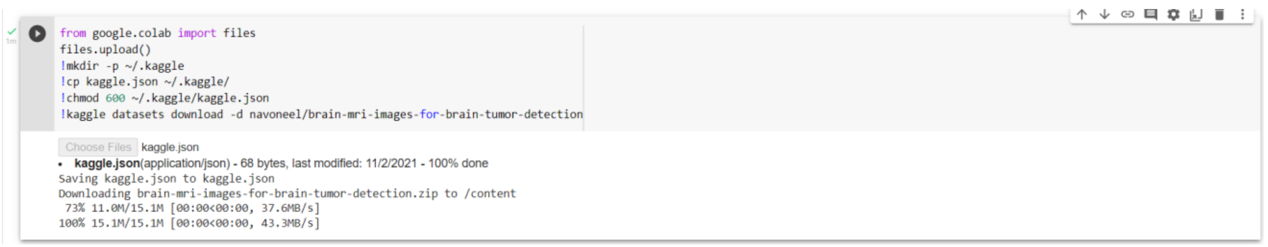
```
import tensorflow as tf
from zipfile import ZipFile
import os,glob
import cv2
from tqdm._tqdm_notebook import tqdm_notebook as tqdm
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Convolution2D, Dropout, Dense,MaxPooling2D
from keras.layers import BatchNormalization
from keras.layers import MaxPooling2D
from keras.layers import Flatten
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.*` instead of `tqdm._tqdm_notebook.*`
  """
```



```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (1.5.12)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from kaggle) (2.23.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.15.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from kaggle) (2021.10.8)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages (from kaggle) (5.0.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from kaggle) (4.62.3)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.24.3)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle) (3.0.4)
```

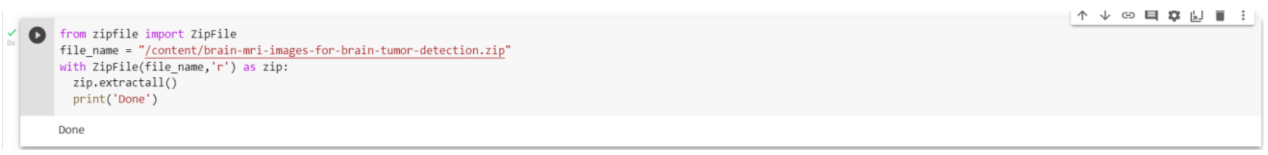


```
from google.colab import files
files.upload()
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d navoneel/brain-mri-images-for-brain-tumor-detection
```

Choose Files kaggle.json

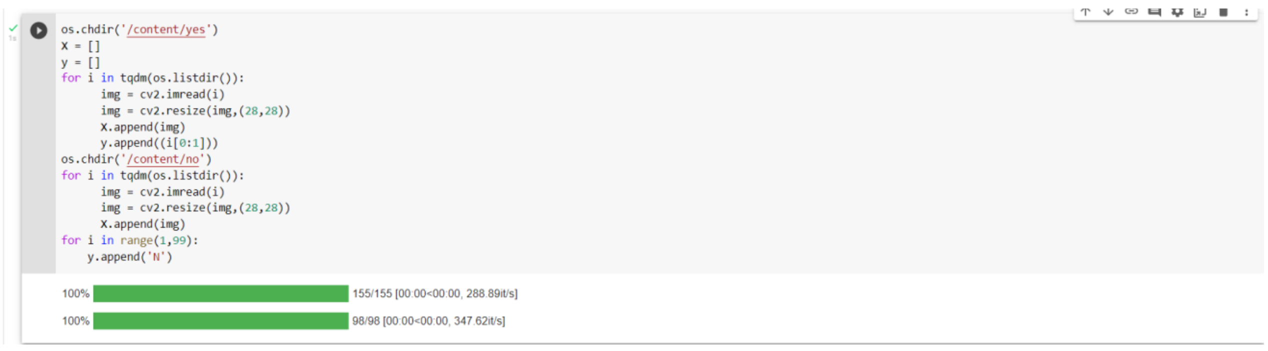
- **kaggle.json**(application/json) - 68 bytes, last modified: 11/2/2021 - 100% done

```
Saving kaggle.json to kaggle.json
Downloading brain-mri-images-for-brain-tumor-detection.zip to /content
 73% 11.0M/15.1M [00:00<00:00, 37.6MB/s]
100% 15.1M/15.1M [00:00<00:00, 43.3MB/s]
```



```
from zipfile import ZipFile
file_name = "/content/brain-mri-images-for-brain-tumor-detection.zip"
with ZipFile(file_name,'r') as zip:
  zip.extractall()
  print('Done')
```

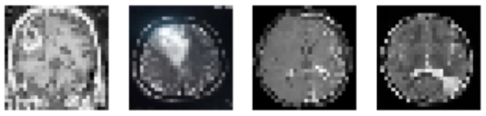
```
Done
```



```
os.chdir('/content/yes')
X = []
y = []
for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(28,28))
      X.append(img)
      y.append((i[0:1]))
os.chdir('/content/no')
for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(28,28))
      X.append(img)
for i in range(1,99):
    y.append('N')
```

```
100% 155/155 [00:00<00:00, 288.89it/s]
100% 98/98 [00:00<00:00, 347.62it/s]
```

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 10))
for i in range(4):
    plt.subplot(1, 4, i+1)
    plt.imshow(X[i], cmap="gray")
    plt.axis('off')
plt.show()
```



```
[7] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
    print ("Shape of an image in X_train: ", X_train[0].shape)
    print ("Shape of an image in X_test: ", X_test[0].shape)

    Shape of an image in X_train:  (28, 28, 3)
    Shape of an image in X_test:  (28, 28, 3)
```

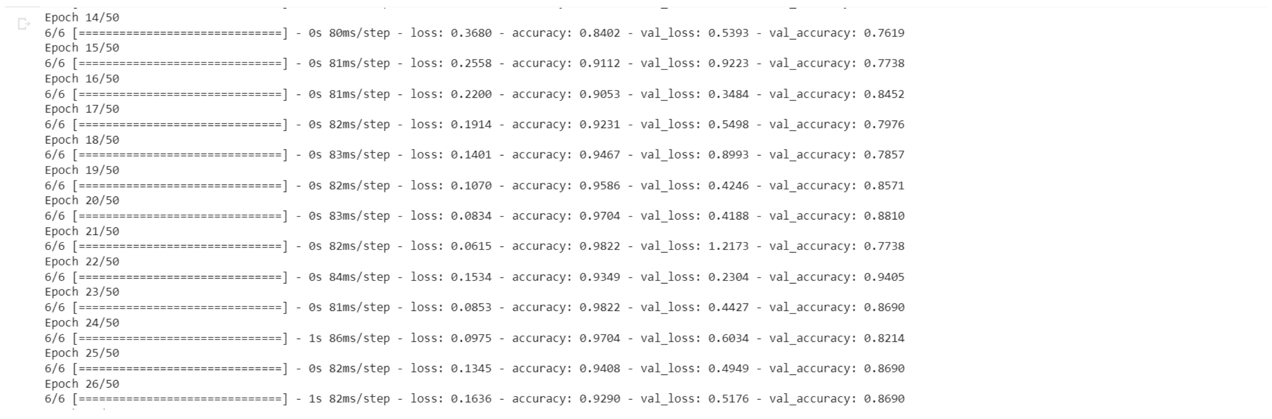
```
le = preprocessing.LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.fit_transform(y_test)
y_train = tf.keras.utils.to_categorical(y_train, num_classes=2)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=2)
y_train = np.array(y_train)
X_train = np.array(X_train)
y_test = np.array(y_test)
X_test = np.array(X_test)
```

```
[9] print("X_train Shape: ", X_train.shape)
    print("X_test Shape: ", X_test.shape)
    print("y_train Shape: ", y_train.shape)
    print("y_test Shape: ", y_test.shape)

    X_train Shape:  (169, 28, 28, 3)
    X_test Shape:  (84, 28, 28, 3)
    y_train Shape:  (169, 2)
    y_test Shape:  (84, 2)
```

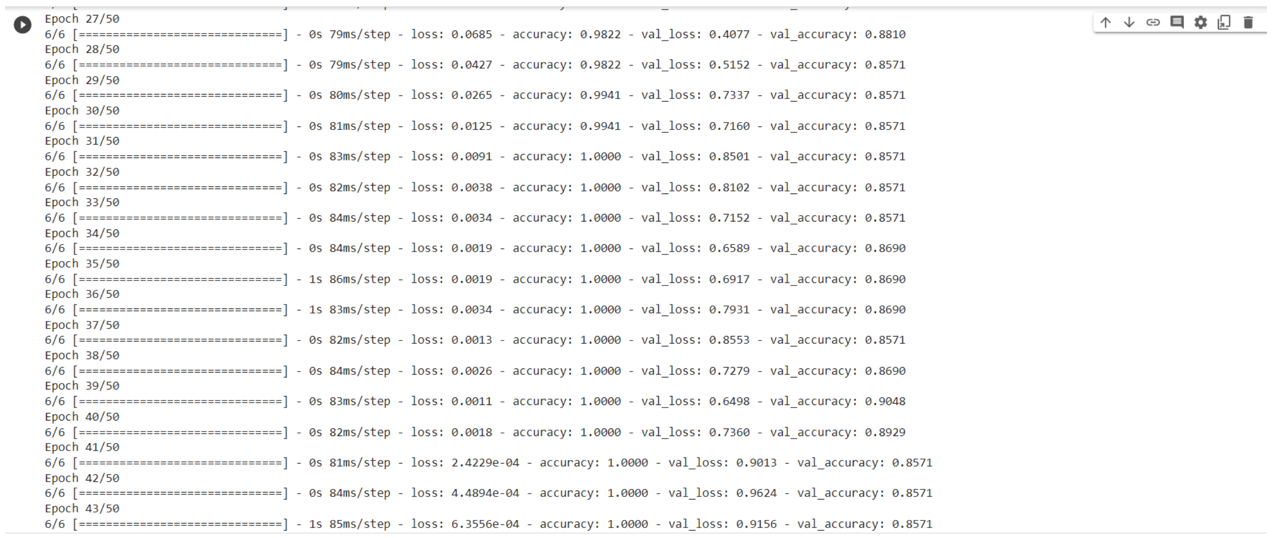
```
m1=Sequential()
m1.add(BatchNormalization(input_shape = (28,28,3)))
m1.add(Convolution2D(32, (3,3), activation ='relu', input_shape = (28, 28, 3)))
m1.add(MaxPooling2D(pool_size=2))
m1.add(Convolution2D(filters=64, kernel_size=4, padding='same', activation='relu'))
m1.add(MaxPooling2D(pool_size=2))
m1.add(Convolution2D(filters=128, kernel_size=3, padding='same', activation='relu'))
m1.add(MaxPooling2D(pool_size=2))
m1.add(Convolution2D(filters=128, kernel_size=2, padding='same', activation='relu'))
m1.add(MaxPooling2D(pool_size=2))
m1.add(Dropout(0.25))
m1.add(Flatten())
m1.add(Dense(units=128,activation = 'relu'))
m1.add(Dense(units = 64, activation = 'relu'))
m1.add(Dense(units = 32, activation = 'relu'))
m1.add(Dense(units = 2, activation = 'softmax'))
```

```
[11] m1.compile(optimizer='adam', loss = 'categorical_crossentropy',metrics = ['accuracy'])
```

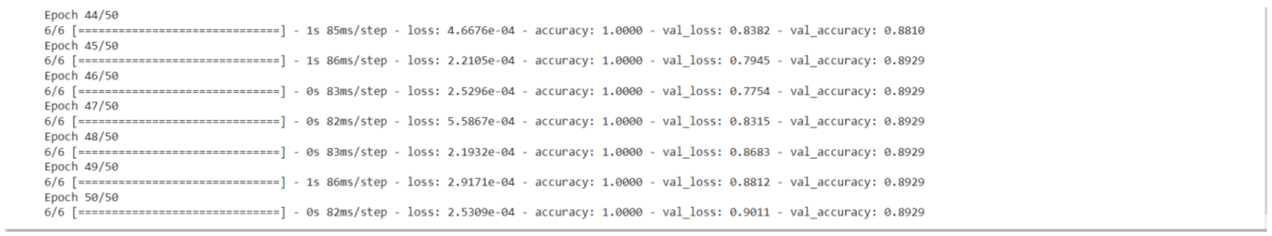
```
history = m1.fit(X_train,y_train,
                    epochs=50,
                    validation_data=(X_test,y_test),
                    verbose = 1,
                    initial_epoch=0)

Epoch 1/50
6/6 [==============================] - 2s 140ms/step - loss: 0.6932 - accuracy: 0.5976 - val_loss: 0.6692 - val_accuracy: 0.6071
Epoch 2/50
6/6 [==============================] - 1s 85ms/step - loss: 0.6482 - accuracy: 0.6272 - val_loss: 1.5430 - val_accuracy: 0.6071
Epoch 3/50
6/6 [==============================] - 1s 93ms/step - loss: 0.6459 - accuracy: 0.6154 - val_loss: 0.8996 - val_accuracy: 0.6071
Epoch 4/50
6/6 [==============================] - 0s 80ms/step - loss: 0.6516 - accuracy: 0.6805 - val_loss: 0.6117 - val_accuracy: 0.6071
Epoch 5/50
6/6 [==============================] - 0s 83ms/step - loss: 0.6335 - accuracy: 0.6805 - val_loss: 1.0931 - val_accuracy: 0.6071
Epoch 6/50
6/6 [==============================] - 0s 80ms/step - loss: 0.6118 - accuracy: 0.6568 - val_loss: 1.0934 - val_accuracy: 0.6071
Epoch 7/50
6/6 [==============================] - 1s 85ms/step - loss: 0.5793 - accuracy: 0.7041 - val_loss: 0.9116 - val_accuracy: 0.6190
Epoch 8/50
6/6 [==============================] - 0s 80ms/step - loss: 0.5155 - accuracy: 0.7456 - val_loss: 1.8508 - val_accuracy: 0.6071
Epoch 9/50
6/6 [==============================] - 0s 83ms/step - loss: 0.5133 - accuracy: 0.7633 - val_loss: 0.5518 - val_accuracy: 0.7857
Epoch 10/50
6/6 [==============================] - 1s 85ms/step - loss: 0.4867 - accuracy: 0.7515 - val_loss: 1.2685 - val_accuracy: 0.6071
Epoch 11/50
6/6 [==============================] - 0s 81ms/step - loss: 0.4322 - accuracy: 0.8166 - val_loss: 0.6359 - val_accuracy: 0.7857
Epoch 12/50
6/6 [==============================] - 0s 78ms/step - loss: 0.4015 - accuracy: 0.8225 - val_loss: 1.1113 - val_accuracy: 0.6190
Epoch 13/50
6/6 [==============================] - 0s 80ms/step - loss: 0.3593 - accuracy: 0.8343 - val_loss: 0.5120 - val_accuracy: 0.7857
```



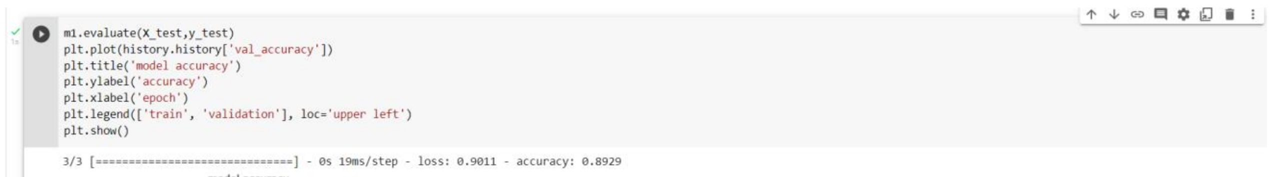
```
Epoch 14/50
6/6 [==============================] - 0s 80ms/step - loss: 0.3680 - accuracy: 0.8402 - val_loss: 0.5393 - val_accuracy: 0.7619
Epoch 15/50
6/6 [==============================] - 0s 81ms/step - loss: 0.2558 - accuracy: 0.9112 - val_loss: 0.9223 - val_accuracy: 0.7738
Epoch 16/50
6/6 [==============================] - 0s 81ms/step - loss: 0.2200 - accuracy: 0.9053 - val_loss: 0.3484 - val_accuracy: 0.8452
Epoch 17/50
6/6 [==============================] - 0s 82ms/step - loss: 0.1914 - accuracy: 0.9231 - val_loss: 0.5498 - val_accuracy: 0.7976
Epoch 18/50
6/6 [==============================] - 0s 83ms/step - loss: 0.1401 - accuracy: 0.9467 - val_loss: 0.8993 - val_accuracy: 0.7857
Epoch 19/50
6/6 [==============================] - 0s 82ms/step - loss: 0.1070 - accuracy: 0.9586 - val_loss: 0.4246 - val_accuracy: 0.8571
Epoch 20/50
6/6 [==============================] - 0s 83ms/step - loss: 0.0834 - accuracy: 0.9704 - val_loss: 0.4188 - val_accuracy: 0.8810
Epoch 21/50
6/6 [==============================] - 0s 82ms/step - loss: 0.0615 - accuracy: 0.9822 - val_loss: 1.2173 - val_accuracy: 0.7738
Epoch 22/50
6/6 [==============================] - 0s 84ms/step - loss: 0.1534 - accuracy: 0.9349 - val_loss: 0.2304 - val_accuracy: 0.9405
Epoch 23/50
6/6 [==============================] - 0s 81ms/step - loss: 0.0853 - accuracy: 0.9822 - val_loss: 0.4427 - val_accuracy: 0.8690
Epoch 24/50
6/6 [==============================] - 1s 86ms/step - loss: 0.0975 - accuracy: 0.9704 - val_loss: 0.6034 - val_accuracy: 0.8214
Epoch 25/50
6/6 [==============================] - 0s 82ms/step - loss: 0.1345 - accuracy: 0.9408 - val_loss: 0.4949 - val_accuracy: 0.8690
Epoch 26/50
6/6 [==============================] - 1s 82ms/step - loss: 0.1636 - accuracy: 0.9290 - val_loss: 0.5176 - val_accuracy: 0.8690
```



```
Epoch 27/50
6/6 [==============================] - 0s 79ms/step - loss: 0.0685 - accuracy: 0.9822 - val_loss: 0.4077 - val_accuracy: 0.8810
Epoch 28/50
6/6 [==============================] - 0s 79ms/step - loss: 0.0427 - accuracy: 0.9822 - val_loss: 0.5152 - val_accuracy: 0.8571
Epoch 29/50
6/6 [==============================] - 0s 80ms/step - loss: 0.0265 - accuracy: 0.9941 - val_loss: 0.7337 - val_accuracy: 0.8571
Epoch 30/50
6/6 [==============================] - 0s 81ms/step - loss: 0.0125 - accuracy: 0.9941 - val_loss: 0.7160 - val_accuracy: 0.8571
Epoch 31/50
6/6 [==============================] - 0s 83ms/step - loss: 0.0091 - accuracy: 1.0000 - val_loss: 0.8501 - val_accuracy: 0.8571
Epoch 32/50
6/6 [==============================] - 0s 82ms/step - loss: 0.0038 - accuracy: 1.0000 - val_loss: 0.8102 - val_accuracy: 0.8571
Epoch 33/50
6/6 [==============================] - 0s 84ms/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 0.7152 - val_accuracy: 0.8571
Epoch 34/50
6/6 [==============================] - 0s 84ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.6589 - val_accuracy: 0.8690
Epoch 35/50
6/6 [==============================] - 1s 86ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.6917 - val_accuracy: 0.8690
Epoch 36/50
6/6 [==============================] - 1s 83ms/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 0.7931 - val_accuracy: 0.8690
Epoch 37/50
6/6 [==============================] - 0s 82ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.8553 - val_accuracy: 0.8571
Epoch 38/50
6/6 [==============================] - 0s 84ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.7279 - val_accuracy: 0.8690
Epoch 39/50
6/6 [==============================] - 0s 83ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.6498 - val_accuracy: 0.9048
Epoch 40/50
6/6 [==============================] - 0s 82ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.7360 - val_accuracy: 0.8929
Epoch 41/50
6/6 [==============================] - 0s 81ms/step - loss: 2.4229e-04 - accuracy: 1.0000 - val_loss: 0.9013 - val_accuracy: 0.8571
Epoch 42/50
6/6 [==============================] - 0s 84ms/step - loss: 4.4894e-04 - accuracy: 1.0000 - val_loss: 0.9624 - val_accuracy: 0.8571
Epoch 43/50
6/6 [==============================] - 1s 85ms/step - loss: 6.3556e-04 - accuracy: 1.0000 - val_loss: 0.9156 - val_accuracy: 0.8571
```

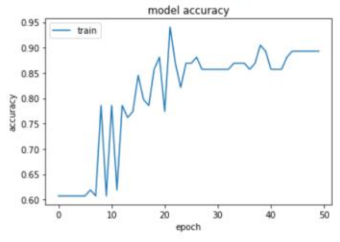


```
Epoch 44/50
6/6 [==============================] - 1s 85ms/step - loss: 4.6676e-04 - accuracy: 1.0000 - val_loss: 0.8382 - val_accuracy: 0.8810
Epoch 45/50
6/6 [==============================] - 1s 86ms/step - loss: 2.2105e-04 - accuracy: 1.0000 - val_loss: 0.7945 - val_accuracy: 0.8929
Epoch 46/50
6/6 [==============================] - 0s 83ms/step - loss: 2.5296e-04 - accuracy: 1.0000 - val_loss: 0.7754 - val_accuracy: 0.8929
Epoch 47/50
6/6 [==============================] - 0s 82ms/step - loss: 5.5867e-04 - accuracy: 1.0000 - val_loss: 0.8315 - val_accuracy: 0.8929
Epoch 48/50
6/6 [==============================] - 0s 83ms/step - loss: 2.1932e-04 - accuracy: 1.0000 - val_loss: 0.8683 - val_accuracy: 0.8929
Epoch 49/50
6/6 [==============================] - 1s 86ms/step - loss: 2.9171e-04 - accuracy: 1.0000 - val_loss: 0.8812 - val_accuracy: 0.8929
Epoch 50/50
6/6 [==============================] - 0s 82ms/step - loss: 2.5309e-04 - accuracy: 1.0000 - val_loss: 0.9011 - val_accuracy: 0.8929
```



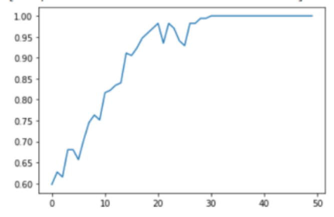
```python
m1.evaluate(X_test,y_test)
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
3/3 [==============================] - 0s 19ms/step - loss: 0.9011 - accuracy: 0.8929
```

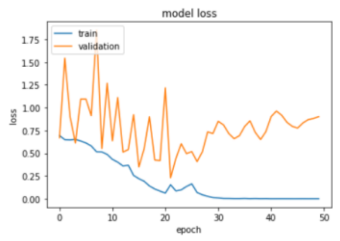


```
plt.plot(history.history['accuracy'])
```

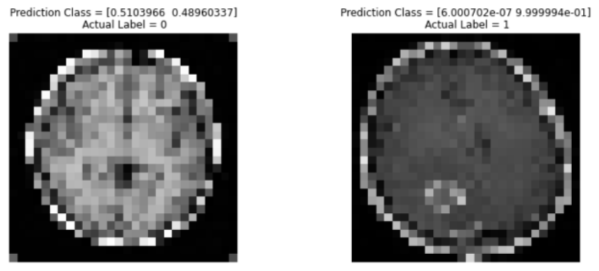
```
[<matplotlib.lines.Line2D at 0x7fa61945b350>]
```



```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

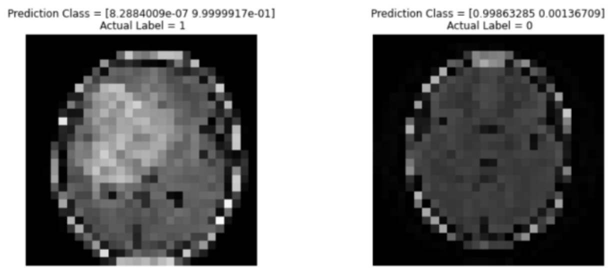


```
y_predicted=m1.predict(X_test)
y_actual=np.argmax(y_test,axis=1)
L = 2
W = 2
fig, axes = plt.subplots(L, W, figsize = (12,12))
axes = axes.ravel()
for i in np.arange(0, L * W):
    axes[i].imshow(X_test[i])
    axes[i].set_title(f"Prediction Class = {y_predicted[i]}\n Actual Label = {y_actual[i]}")
    axes[i].axis('off')
plt.subplots_adjust(wspace=0.5)
```



Prediction Class = [0.5103966 0.48960337]
Actual Label = 0

Prediction Class = [6.000702e-07 9.999994e-01]
Actual Label = 1



Prediction Class = [8.2884009e-07 9.9999917e-01]
Actual Label = 1

Prediction Class = [0.99863285 0.00136709]
Actual Label = 0

```
import tensorflow as tf
tf.keras.models.save_model(m1,'/content')
```

```
INFO:tensorflow:Assets written to: /content/assets
```

8. REFERENCES

1. Brain Tumor Detection using Convolutional Neural Network, Sachin R Jadhav, Shubham S Salve, Harshal S Mohagaonkar , Akhilesh D Rakibe, Nishant G Langade

   Reference Link: https://www.irjet.net/archives/V7/i1/IRJET-V7I1211.pdf
2. Brain Tumor Classification Using Convolutional Neural Networks, J. Seetha and S. Selvakumar Raja

   Reference Link: https://biomedpharmajournal.org/vol11no3/brain-tumor-classification-using-convolutional-neural-networks/
3. https://www.python.org/
4. https://colab.research.google.com/