DSCI-6003-01 MACHINE LEARNING

TERM PROJECT REPORT

ROSHINI BANDI – 00820265

MEDICAL IMAGE ANALYSIS USING CNN ALGORITHM

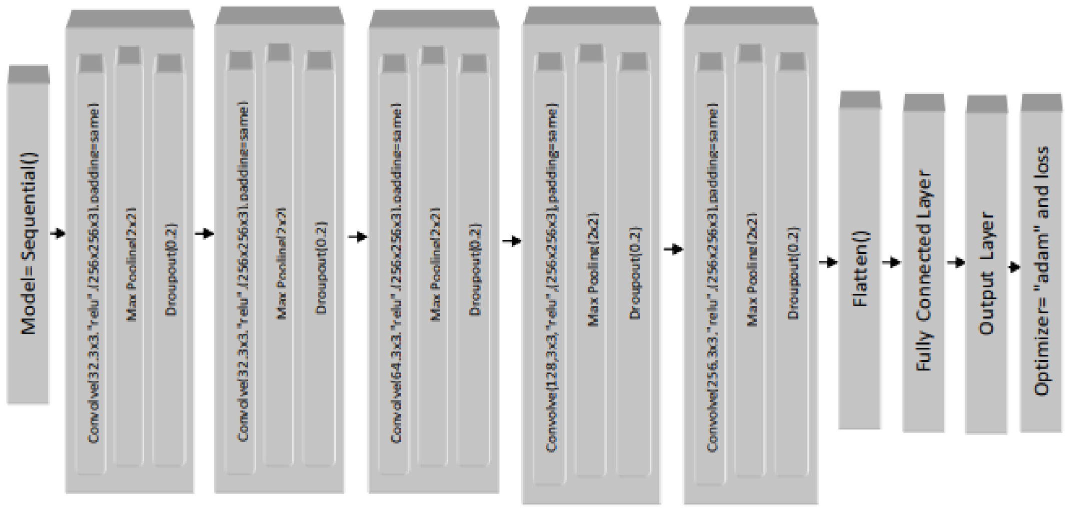
Objective:

Machine learning-based technologies, as well as other medical science domains, play a crucial part in diagnosing the condition in a much clearer manner than ever before, thereby giving a viable alternative to surgical biopsy for brain tumors. Automated flaw detection in medical imaging using machine learning has become a hot topic in a variety of medical diagnostics. Its application in MRI brain tumor detection is critical because it collects data about abnormal tissues, which is necessary for treatment planning.

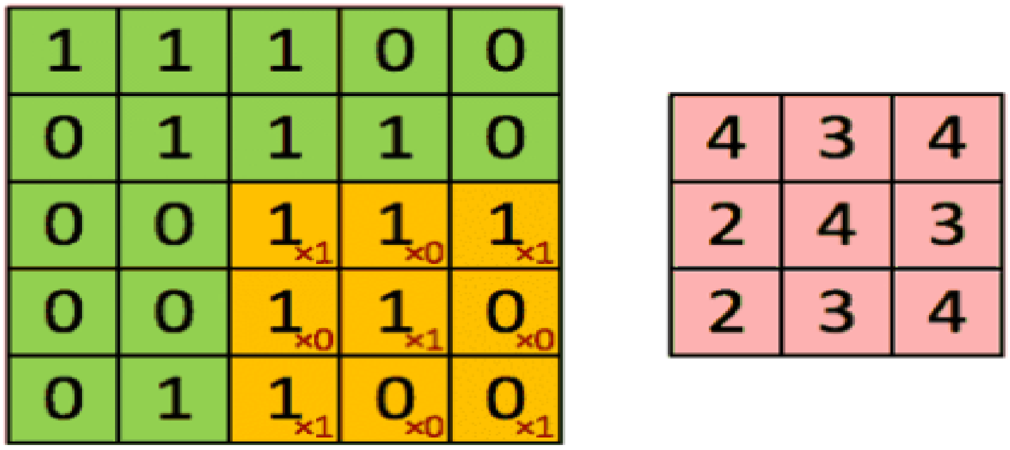
Introduction to CNN Algorithm:

CNN (Convolutional Neural Network) is the most extensively used machine learning algorithm in image recognition and visual learning tasks. In CNN, both 2-D based and 3-D based structures of an organism are examined in order to determine whether they are normal or abnormal. For 2-D based imaging, such as X-rays, and 3-D based imaging, such as CT and MRI. CNN have been utilised in a variety of ways, including image recognition, classification, detection, segmentation, localization, image processing, and face detection etc. Lines, gradients, circles, and even eyes and faces may be detected swiftly by convolutional neural networks in the input image. Because of this trait, convolutional neural networks are particularly powerful in computer vision. Convolutional neural networks, unlike previous vision - based approaches, can function on a raw image without any pre-processing.

The CNN sequential model is brought to life using various layers . A feed-forward neural network with up to 20 or 30 layers is known as a convolutional neural network. The Convolutional layer, Pooling layer, Fully connected layer, and RELU layer are the layers. With each layer, the CNN grows more sophisticated, identifying bigger portions of the image. The first layers focus on the most fundamental aspects, such as colours and borders. The visual data begins to grow larger as it travels through the CNN layers.

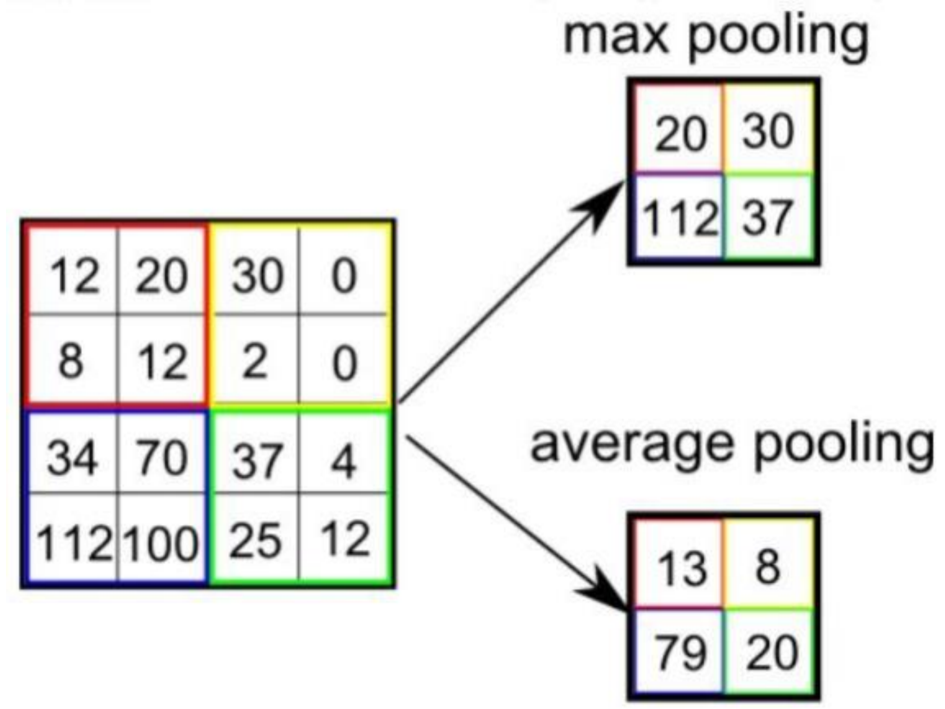

Model= Sequential()
Convolve(32,3x3,"relu",(256x256x3),padding=same)
Max Pooling(2x2)
Droupout(0.2)
Convolve(32,3x3,"relu",(256x256x3),padding=same)
Max Pooling(2x2)
Droupout(0.2)
Convolve(64,3x3,"relu",(256x256x3),padding=same)
Max Pooling(2x2)
Droupout(0.2)
Convolve(128,3x3,"relu",(256x256x3),padding=same)
Max Pooling(2x2)
Droupout(0.2)
Convolve(256,3x3,"relu",(256x256x3),padding=same)
Max Pooling(2x2)
Droupout(0.2)
Flatten()
Fully Connected Layer
Output Layer
Optimizer= "adam" and loss

Convolutional layer: The convolutional layer is classified into two main functions. The input values at a picture point are the first function, and the filter is the second. Taking the dot product of two functions yields a result. The image is resized to 256x256 pixels from its original size. The activation function for the convolutional layer is the rectified linear unit (RELU), with padding equal to the input picture and a total number of filters of 32,32, 64,128,256 for various convolutional layers. The convolutional layer is a sort of layer that delivers the power of a convolutional neural network. By learning visual attributes using small squares of input data, convolution keeps the relationship between pixels. It's a mathematical method for making a map out of two inputs like an image matrix and a filter.



RELU layer: The RELU layer is a function that converts negative input numbers into zero. For all positive values, RELU is linear (identity), while for all negative values, RELU is zero. It also aids in avoiding the vanishing gradient issue. If a RELU neuron is trapped in the negative side and always produces 0, it is said to be "dead." Because the slope of RELU in the negative range is also 0, it's improbable that a neuron will recover once it's gone negative. Such neurons are essentially worthless because they don't play any part in discerning the input. Over time, you may find that a big portion of your network is idle. A Rectifier can be applied to the dataset to approximate the non-linearity. RELU is an example of a unit that uses a rectifier.

Pooling layer: To reduce the number of parameters and image size, the pooling layer is put in between the convolution and RELU layers. The most frequent method is max-pooling. The Pooling layer is in charge of shrinking the Convolved Feature's spatial size. This is done to reduce the amount of computer power needed to process the data by decreasing the dimensionality of the data.



Max Pooling returns the most fees from the part of the image included with the aid of using the Kernel. On the alternative hand, Average Pooling returns the common of all of the values from the image's Kernel section. Max Pooling additionally acts as a Noise Suppressant. It gets rid of all noisy activations at the same time as additionally acting de-noising and dimensionality reduction.

Fully Connected layer: The entirely linked layer is created using the dense technique, with 256 units using RELU as the activation function. Each of the two classes has one unit in the output layer, as well as a sigmoid as an activation function. In the Fully Connected Layer, every neuron in the preceding layer is connected to every neuron in the Fully Connected Layer, such as convolution, RELU, and pooling. The output of the previous layer is fed into this layer, which splits it into classes. Using a Fully Connected layer to learn non-linear combinations of high-level information represented by the convolutional layer's output is a (usually) low-cost method. Fully Connected layer is now acquiring a non-linear function within that area.

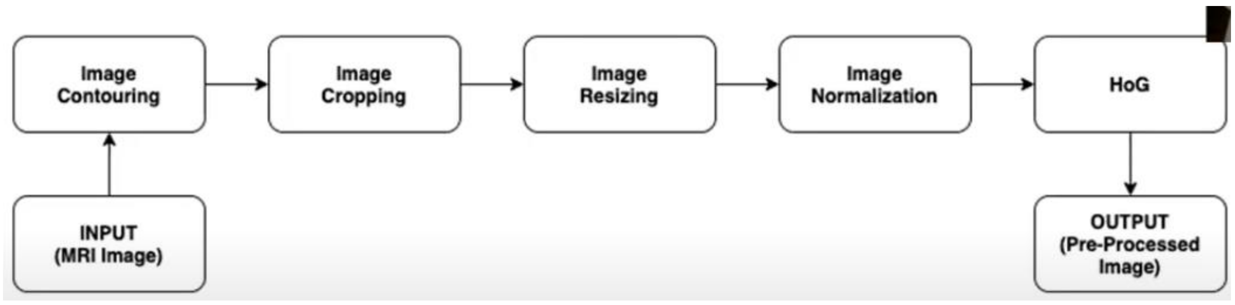
Data Collection:

The original dataset was taken from the following link:
https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri

The data set was gathered from the Kaggle source. Images can be in the form of.csv (comma separated values),.dat (data) files in grayscale, RGB, or HSV, or simply in the form of a.zip file or JPEG, as in the case of our online Kaggle dataset.

Image pre-processing:

The aim of pre-processing is to improve the images quality so that we can better analyse it. We can reduce unwanted distortions and boost some qualities that are important for the application we're working on by pre-processing. Those functionalities may differ depending on the application. Rescaling, noise removal, Binary Thresholding, and morphological operations like erosion and dilation, contour shaping are all part of our pre-processing (edge-based methodology). The image's memory space is decreased in the first step of pre-processing by scaling the gray-level of the pixels in the range 0-255.
Because the shape of the brain is not segmented as tumor here, we employed the Gaussian blur filter for noise removal because it is known to produce better results than the Median filter. The quantity of preprocessing performed by a CNN is much smaller than that used by other classification algorithms. While filters are hand-engineered in basic approaches, CNN can learn these filters/characteristics with adequate training.



Preprocessing is necessary because it improves picture data and reinforces a number of image properties that are crucial for subsequent processing. The MR image is subjected to the following pre-processing procedures. The median filter is used to remove noise from brain MR images once the RGB MR picture is converted to grayscale. Because great accuracy is required, the noise must be removed before proceeding with the operation. The edges of a filtered image are then discovered using canny edge detection, as shown. The edge detected image is required for image segmentation. The purpose of segmentation is to turn an image representation into something that can be examined more easily.

Training and Testing the datasets:

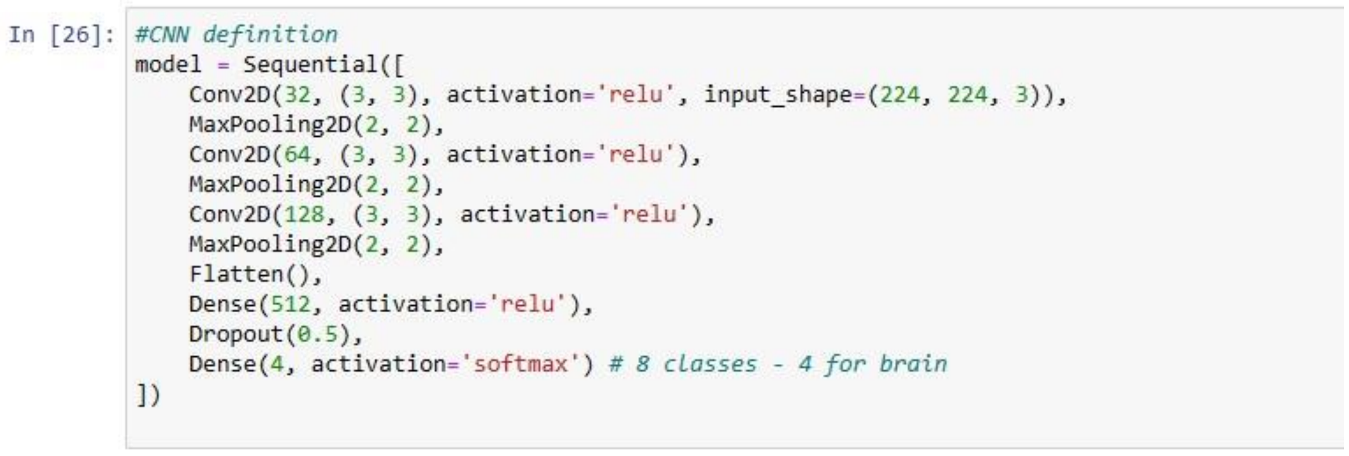
Training datasets: The sample of the data is used to fit the model. The data collection used to train the model includes weights and distortions in the case of a neural network. The model is first fitted using a training data set, which is a collection of instances of the model. Parameters (for example the weights of the connections between neurons in artificial neural networks).

A supercomputer is used to train the model on the training data set. A supervised learning process is used to train the model on the training data set. Each observation in supervised learning issues contains an output observed variable and one or more input observed variables.

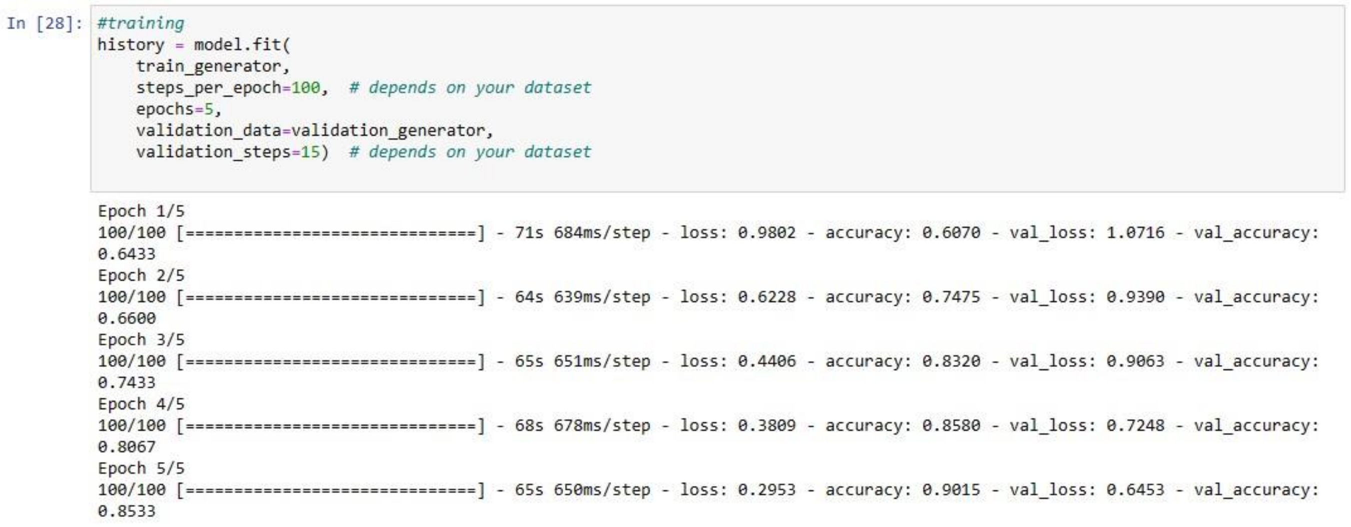
Testing datasets: An unbiased evaluation of a final model fit on the training dataset using a sample of data. The gold standard for evaluating the model is the Test dataset. It's only used once a model has been fully trained (using the train and validation sets). The test set is typically used to compare and contrast competing models. A test set is a collection of instances used solely to evaluate a fully stated classifier's performance (i.e. generalisation).

The final model is used to forecast classifications of instances in the test set to accomplish this. Against measure the model's accuracy, these predictions are compared to the true classifications of the cases. The test data set is typically used to determine whether your model is operating appropriately. At the end of each assignment, a test data set is applied to your model to validate its performance. All you need is your test data to fit on the training data set and ensure that it is working well before moving on to the next step.

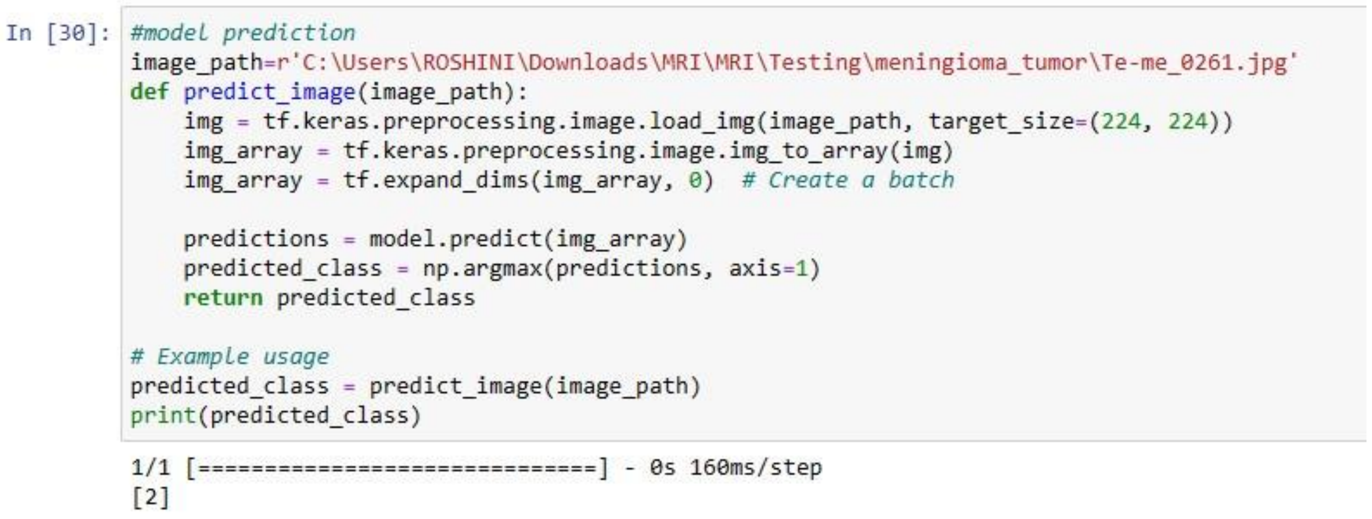
Implementation Outputs:



```
In [26]: #CNN definition
         model = Sequential([
             Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
             MaxPooling2D(2, 2),
             Conv2D(64, (3, 3), activation='relu'),
             MaxPooling2D(2, 2),
             Conv2D(128, (3, 3), activation='relu'),
             MaxPooling2D(2, 2),
             Flatten(),
             Dense(512, activation='relu'),
             Dropout(0.5),
             Dense(4, activation='softmax') # 8 classes - 4 for brain
         ])
```



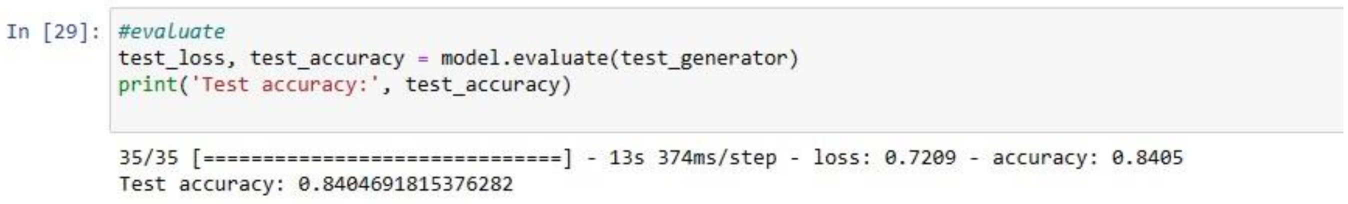
```
In [28]: #training
         history = model.fit(
             train_generator,
             steps_per_epoch=100,  # depends on your dataset
             epochs=5,
             validation_data=validation_generator,
             validation_steps=15)  # depends on your dataset

         Epoch 1/5
         100/100 [==============================] - 71s 684ms/step - loss: 0.9802 - accuracy: 0.6070 - val_loss: 1.0716 - val_accuracy:
         0.6433
         Epoch 2/5
         100/100 [==============================] - 64s 639ms/step - loss: 0.6228 - accuracy: 0.7475 - val_loss: 0.9390 - val_accuracy:
         0.6600
         Epoch 3/5
         100/100 [==============================] - 65s 651ms/step - loss: 0.4406 - accuracy: 0.8320 - val_loss: 0.9063 - val_accuracy:
         0.7433
         Epoch 4/5
         100/100 [==============================] - 68s 678ms/step - loss: 0.3809 - accuracy: 0.8580 - val_loss: 0.7248 - val_accuracy:
         0.8067
         Epoch 5/5
         100/100 [==============================] - 65s 650ms/step - loss: 0.2953 - accuracy: 0.9015 - val_loss: 0.6453 - val_accuracy:
         0.8533
```



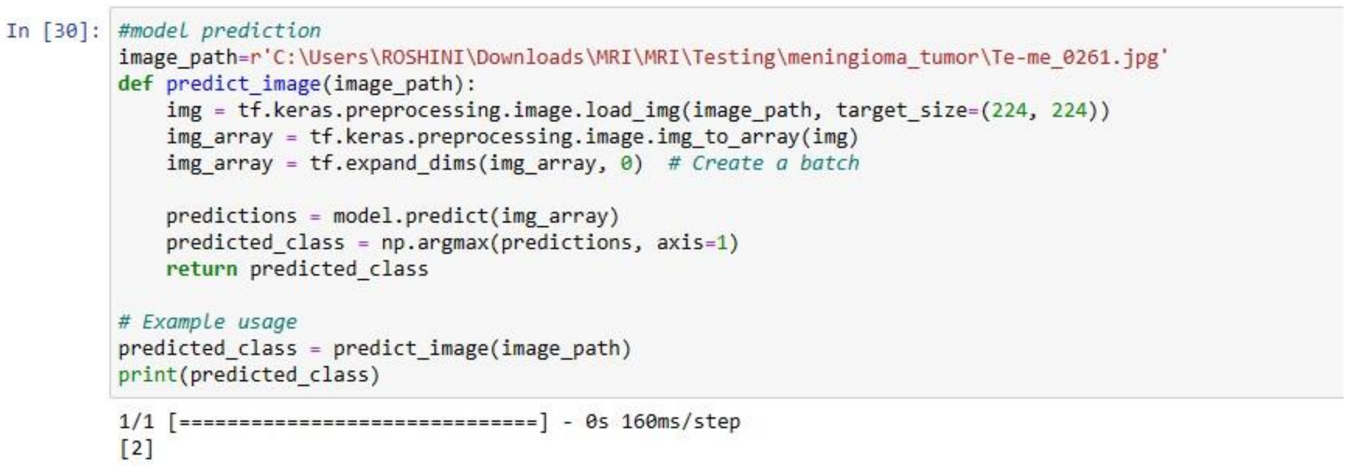
```
In [30]: #model prediction
         image_path=r'C:\Users\ROSHINI\Downloads\MRI\MRI\Testing\meningioma_tumor\Te-me_0261.jpg'
         def predict_image(image_path):
             img = tf.keras.preprocessing.image.load_img(image_path, target_size=(224, 224))
             img_array = tf.keras.preprocessing.image.img_to_array(img)
             img_array = tf.expand_dims(img_array, 0)  # Create a batch

             predictions = model.predict(img_array)
             predicted_class = np.argmax(predictions, axis=1)
             return predicted_class

         # Example usage
         predicted_class = predict_image(image_path)
         print(predicted_class)

         1/1 [==============================] - 0s 160ms/step
         [2]
```

Evaluation & Results:

On our training and testing sets, we ran the two methods, and the results are displayed above. We now concentrate on determining how significant the model considers each symptom (input) to be.

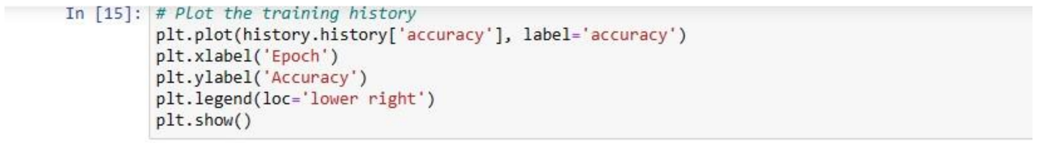


```
In [29]: #evaluate
         test_loss, test_accuracy = model.evaluate(test_generator)
         print('Test accuracy:', test_accuracy)

         35/35 [==============================] - 13s 374ms/step - loss: 0.7209 - accuracy: 0.8405
         Test accuracy: 0.8404691815376282
```

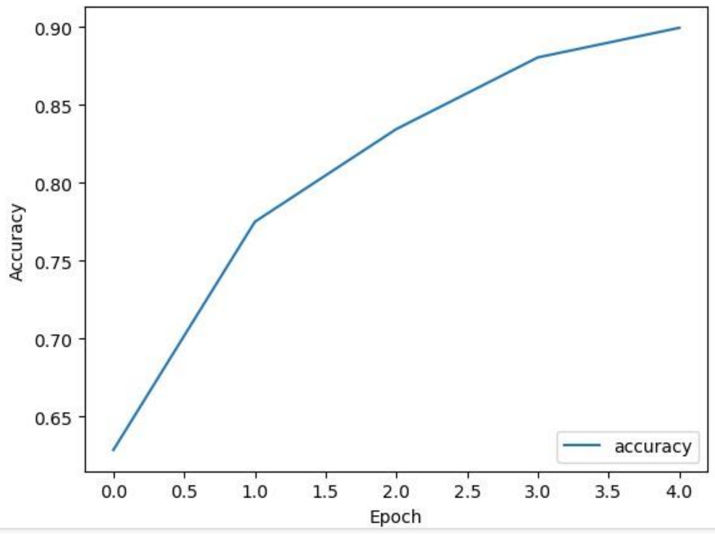


```
In [30]: #model prediction
         image_path=r'C:\Users\ROSHINI\Downloads\MRI\MRI\Testing\meningioma_tumor\Te-me_0261.jpg'
         def predict_image(image_path):
             img = tf.keras.preprocessing.image.load_img(image_path, target_size=(224, 224))
             img_array = tf.keras.preprocessing.image.img_to_array(img)
             img_array = tf.expand_dims(img_array, 0)  # Create a batch

             predictions = model.predict(img_array)
             predicted_class = np.argmax(predictions, axis=1)
             return predicted_class

         # Example usage
         predicted_class = predict_image(image_path)
         print(predicted_class)

         1/1 [==============================] - 0s 160ms/step
         [2]
```

Visualisation Output:



```
In [15]: # Plot the training history
         plt.plot(history.history['accuracy'], label='accuracy')
         plt.xlabel('Epoch')
         plt.ylabel('Accuracy')
         plt.legend(loc='lower right')
         plt.show()
```



Conclusion:

The main goal of this research work is to design efficient automatic brain tumor classification with high accuracy, performance and low complexity. In the conventional brain tumor classification is performed by using CNN, texture and shape feature extraction and Sequential model, Binary Thresholding. And the classification results are given as benign or malignant. Finally, the loss function (binary cross entropy) is applied to achieve high accuracy. The training accuracy, and validation loss are calculated. Therefore, accuracy is high and validation loss is very low.

Future Scope:

Build an app-based computer program in hospitals which allows doctors to simply determine the impact of tumor and suggest treatment accordingly. Because the performance and complexity of CNN are dependent on computer file representation, we will attempt to forecast tumor localization as well as stage from volume-based 3D pictures. Training, planning, and computer guidance during surgery are all improved by building three-dimensional (3D) anatomical models from specific patients.

Improve testing accuracy and computation time by using classifier boosting techniques like using more number images with more data augmentation, fine-tuning hyper parameters, training for a extended time i.e. using more epochs, adding more appropriate layers etc. Classifier boosting is accomplished by developing a model from the training data and then a second model that seeks to repair the faults in the original model for faster prediction. Such strategies will be used to improve the accuracy even more, potentially allowing this mechanism to be a valuable asset to any clinical facility dealing with brain tumors. For more complex datasets, we will use U-Net architecture instead of CNN where the max pooling layers are just replaced by up sampling ones. Finally, we may want to apply very wide and deep convolutional nets on video sequences where the temporal structure provides extremely useful information that is lacking or less visible in static images. Unsupervised transfer learning may attract more and more attention within the future.