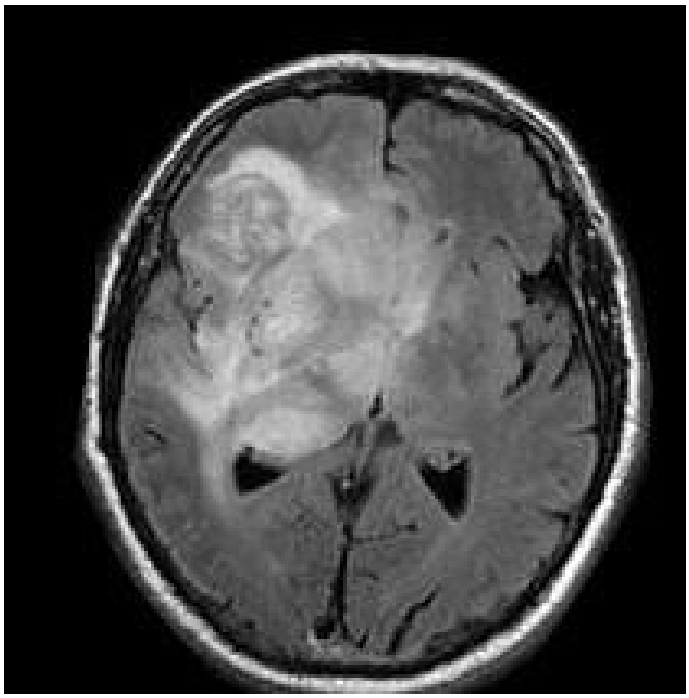
# Brain Tumor Detection with Machine Learning

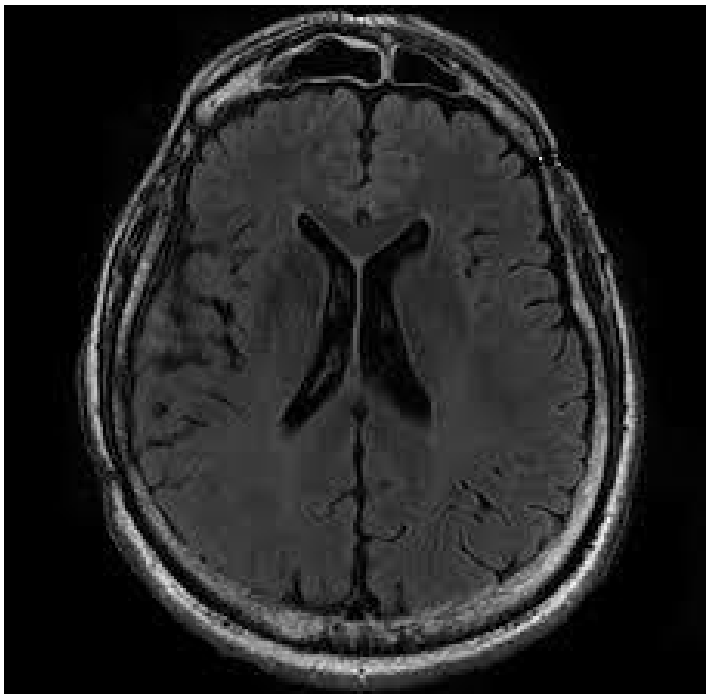
The dataset I used for this experiment is obtained from the Kaggle link:
https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection

So, first, let me describe the problem that we will be solving over here. In this, we want to classify an **MRI Scan** of a patient's brain obtained in the axial plane as to whether there is a presence of a tumour or not. I am sharing a sample image of what an MRI scan looks like with tumour and without one.

**MRI image with a tumour:**



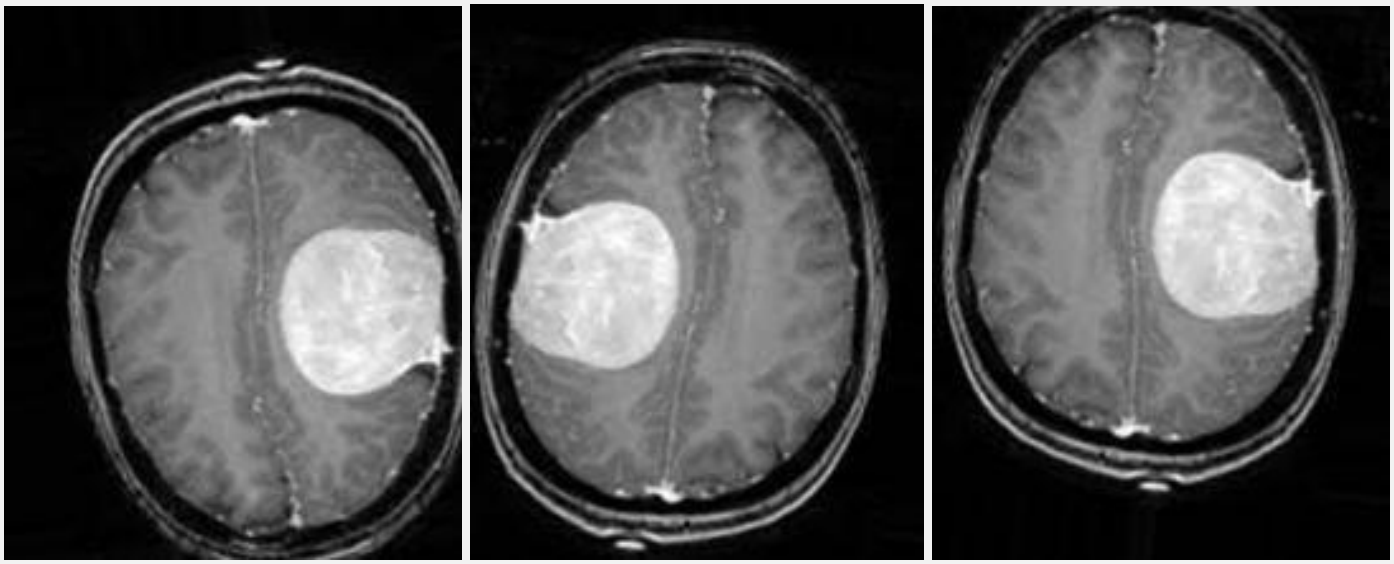
**MRI image without tumour:**



Now how will we use AI or Deep Learning in particular, to classify the images as a tumour or not? The answer is Convolutional Neural Networks (CNN). CNN or ConvNet is a class of Deep Learning, mostly applied to analyze visual images. There are many frameworks in python to apply CNN such as TensorFlow, PyTorch to train the model. I will be using the Keras library with TensorFlow backend to train this model.

1) **Data Visualization:**
We have a total of 253 MRI images. Out of them, 155 are labelled "yes", which indicates that there is a tumour and the remaining 98 are labelled "no", which indicates that there is no tumour. We encounter a new problem known as data imbalance. Data imbalance is where the number of observations per class is not equally distributed. Our Neural Network would not be given enough training in the "no" class.

2) **Data Augmentation:**
In Data Augmentation, we take a particular MRI image and perform various sorts of image enhancements such as rotate, mirror and flip to get more number of images. We will apply more augmentation to the class with less number of images to get an approximately equal number of images to both classes.



**Augmented Data**

**3) Preprocessing of Images:**

Here we are going to preprocess of image. We will convert the image to grayscale, and blur it slightly. After that, we will find contours in the thresholded image, then grab the largest one. Also, we will crop the new image out of the original image using the four extreme points (left, right, top, bottom).

**4) Splitting the data:**

After getting an array of the preprocessed image first we will shuffle the dataset. Shuffling the data improves the model. In the next step, we split our data into the training set, test set and validation set. 70% (1445 images) of the images will go to the training set, which will be used by our neural network to get trained. The remaining 30% (619 images) will go to the test set and validation set equally(mean 15%-15% of the whole set goes to test set and validation set), with which we will apply our trained neural network and classify them to check the accuracy of our Neural Network.

**5) Building the CNN Model:**

I have made a model using Keras library with TensorFlow at the backend. I have tried to keep the model architecture as simple as possible, which is given below:

```
model = Sequential()
model.add(Conv2D(32, (7, 7), activation='relu', input_shape = INPUT_SHAPE,use_bias=False))
model.add(BatchNormalization(axis = 3, name = 'bn0'))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(2))
model.add(Activation('softmax'))
```

I have used the softmax activation function at the end of the model.

**6) Compiling the model:**

I have compiled the model by Adam optimizer with 0.001 learning rate using Accuracy Metrics.

model.compile(loss = "categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics=['accuracy'])

**7) Training the model:**

I have trained the model with Validation Dataset with a batch size of 32 and only 22 epochs (iterations).

model.fit(x=trainData, y=trainLabel, batch_size=32, epochs=22, validation_data=(valData, valLabel))

**8) Calculating the accuracy of the model:**

With the help of test dataset, I have calculated the accuracy of our model, which is 97.1%. Pretty good!

# Predict for the test set
Y_newpred=model.predict(testData)
print(Y_newpred)

scores = model.evaluate(testData, testLabel, verbose=0)
print('Accuracy: {}% \n Error: {}%'.format(scores[1]*100, 100 - scores[1]*100))

```
[ ]: # Predict for the test set
     Y_newpred=model.predict(testData)
     print(Y_newpred)
```

```
[34]: scores = model.evaluate(testData, testLabel, verbose=0)
      print('Accuracy: {}% \n Error: {}%'.format(scores[1]*100, 100 - scores[1]*100))
```

```
Accuracy: 97.09677696228027%
 Error: 2.9032230377197266%
```

```
[36]: model.save('ShivModel.h5')
```

**9) Saving the model:**

At last, we will save the model using this syntax,

model.save('model_name.h5')

**Reference:**

https://www.kaggle.com/ethernext/brain-tumour-detection-with-cnn-96-accuracy

By Shiv Modi,
UG Mechanical Engineer, IIT Bombay.