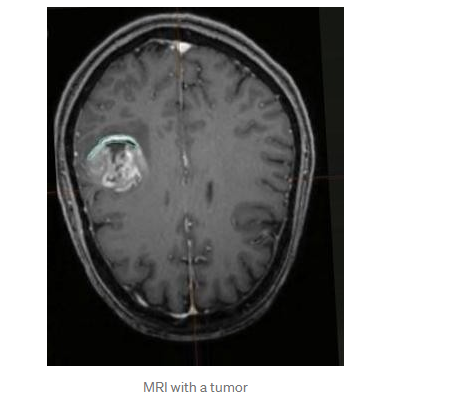
Brain Tumor Classification using Deep Learning with Keras in Python

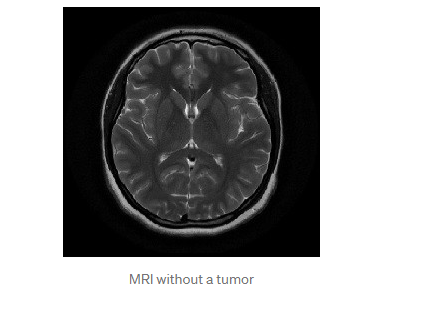
# INTRODUCTION

# Computer vision techniques have shown tremendous results in some areas in the medical domain like surgery and therapy of different diseases. Even researchers are trying to experiment with the detection of different diseases like cancer in the lungs and kidneys. Different medical imaging datasets are publicly available today for researchers like [Cancer Imaging Archive](https://www.cancerimagingarchive.net/) where we can get data access of large databases free of cost. A brain tumor is one of the problems wherein the brain of a patient’s different abnormal cells develops. They are called tumors that can again be divided into different types.

# Approach

# 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 whether there is a presence of tumor or not. I am sharing a sample image of what an MRI scan looks like with tumor and without one.





We see that in the first image, to the left side of the brain, there is a tumor formation, whereas in the second image, there is no such formation. So, we can see that there is a clear distinction between the two images. Now how will we use AI or Deep Learning in particular, to classify the images as a tumor or not?

The answer is **Convolution 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 *Tensor Flow* to train the model. I will be using the *Keas* library with *Tensor Flow* backend to train this model. Okay! Enough of technical terms, let’s get back to solving the problem.

My approach for solving this problem is divided into 5 major steps:

* Dataset Description
* Data Import and preprocessing
* Model Building
* Model Analysis and comparison
* Conclusion

# ****1. Dataset Description****

The dataset used for this problem is named [Brain MRI Images for Brain Tumor Detection](https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection) . It consists of MRI scans of two classes:

* NO - Tumor does not present i.e., normal, encoded as 0
* YES - Tumor present, encoded as 1

There are 9498 images belonging to 2 classes (Yes-tumor and No-tumor) for Training, 1000 images belonging to 2 classes for Validation.

# 2. Data Import and preprocessing

In this step, we first arrange the images which are currently present in two folders Yes and No into three separate folders Train, Test, and Val.

We import the data and arrange it into separate folders for further model building process. There 4999 no tumor images to train\_no\_dir. In the next step, next 500 no tumor images to validation\_no\_dir, 750 no tumor images to test\_no\_dir, 4999 yes tumor images to train\_yes\_dir, 500 yes tumor images to validation\_yes\_dir and 500 yes tumor images to test\_yes\_dir.

Rescales all images by 1/255

**Why we have to rescale by 1. / 255?**

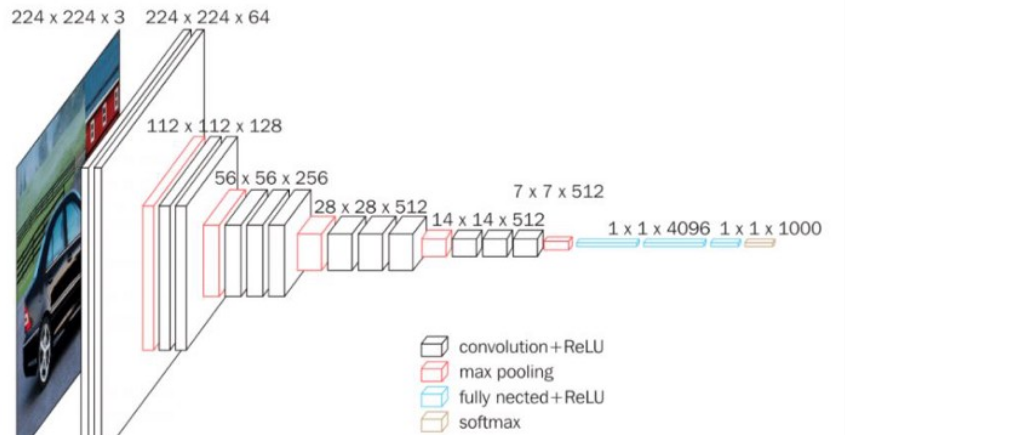
Rescale is a value by which we will multiply the data before any other processing. Our original images consist in RGB coefficients in the 0-255, but such values would be too high for our model to process (given a typical learning rate), so we target values between 0 and 1 instead by scaling with a 1/255. Factor.

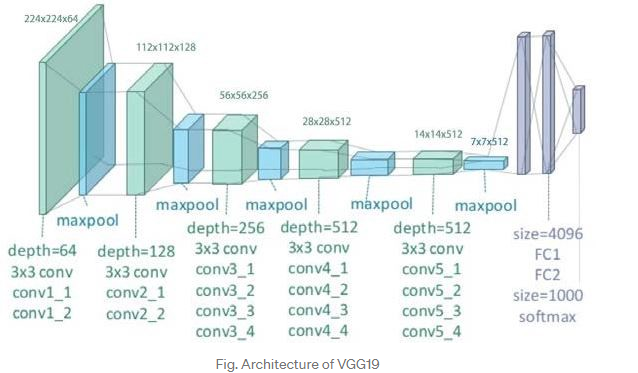
As I have used VGG16 network pre-trained model which require input images of shape (224,224) due to this reason some of the wide images look abnormal so we need to resizes all images to 224 × 224, Batch Size = 20 and using class\_mode = binary (Because we use binary\_crossentropy loss, we need binary labels.)

# 3. Model building

**Import VGG16**

In this step, we have built different customized deep learning models using the transfer learning approach. We have built VGG16, sequential model using Keras framework. The detailed architecture of [VGG16](https://arxiv.org/pdf/1409.1556.pdf), is shown below.





# 4. Model Analysis and comparison

Fine Tuning of VGG16 Use weights of 'imagenet’, exclude Top Layers

from tensorflow.keras.applications import VGG16

conv\_base = VGG16(weights='imagenet',

include\_top=False,

input\_shape=(224, 224, 3))

conv\_base.summary()

Layer (type) Output Shape Param #

=================================================================

input\_1 (InputLayer) [(None, 224, 224, 3)] 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_conv1 (Conv2D) (None, 224, 224, 64) 1792

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_conv2 (Conv2D) (None, 224, 224, 64) 36928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_pool (MaxPooling2D) (None, 112, 112, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_conv1 (Conv2D) (None, 112, 112, 128) 73856

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_conv2 (Conv2D) (None, 112, 112, 128) 147584

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_pool (MaxPooling2D) (None, 56, 56, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv1 (Conv2D) (None, 56, 56, 256) 295168

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv2 (Conv2D) (None, 56, 56, 256) 590080

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv3 (Conv2D) (None, 56, 56, 256) 590080

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_pool (MaxPooling2D) (None, 28, 28, 256) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv1 (Conv2D) (None, 28, 28, 512) 1180160

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv2 (Conv2D) (None, 28, 28, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv3 (Conv2D) (None, 28, 28, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_pool (MaxPooling2D) (None, 14, 14, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv1 (Conv2D) (None, 14, 14, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv2 (Conv2D) (None, 14, 14, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv3 (Conv2D) (None, 14, 14, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_pool (MaxPooling2D) (None, 7, 7, 512) 0

=================================================================

Total params: 14,714,688

Trainable params: 14,714,688

Non-trainable params: 0

# Hybrid Method

# By Adding Flatten, Dropout and Dense layers In VGG16 and Compile Model with 'binary\_crossentropy'

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras import layers

from tensorflow.keras.optimizers import RMSprop

NUM\_CLASSES = 1

model = Sequential()

model.add(conv\_base)

model.add(layers.Flatten())

model.add(layers.Dropout(0.5))

model.add(layers.Dense(NUM\_CLASSES, activation='sigmoid'))

model.layers[0].trainable = False

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

vgg16 (Model) (None, 7, 7, 512) 14714688

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 25088) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout (Dropout) (None, 25088) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 1) 25089

=================================================================

Total params: 14,739,777

Trainable params: 25,089

Non-trainable params: 14,714,688

**Early Stopping.**

**Stop training when a monitored metric has stopped improving.**

Assuming the goal of a training is to minimize the loss. With this, the metric to be monitored would be 'loss', and mode would be 'min'. A model.fit() training loop will check at end of every epoch whether the loss is no longer decreasing, considering the min\_delta and patience if applicable. Once it's found no longer decreasing, model.stop\_training is marked True and the training terminates.

The quantity to be monitored needs to be available in logs dict. To make it so, pass the loss or metrics at model.compile().

**Arguments**

* **monitor**: Quantity to be monitored.
* **min\_delta**: Minimum change in the monitored quantity to qualify as an improvement, i.e. an absolute change of less than min\_delta, will count as no improvement.
* **patience**: Number of epochs with no improvement after which training will be stopped.
* **verbose**: verbosity mode.
* **mode**: One of {"auto", "min", "max"}. In min mode, training will stop when the quantity monitored has stopped decreasing; in "max" mode it will stop when the quantity monitored has stopped increasing; in "auto" mode, the direction is automatically inferred from the name of the monitored quantity.

**Model Checkpoint.**

Callback to save the Keras model or model weights at some frequency.

ModelCheckpoint callback is used in conjunction with training using model.fit() to save a model or weights (in a checkpoint file) at some interval, so the model or weights can be loaded later to continue the training from the state saved.

A few options this callback provides include:

* Whether to only keep the model that has achieved the "best performance" so far, or whether to save the model at the end of every epoch regardless of performance.
* Definition of 'best'; which quantity to monitor and whether it should be maximized or minimized.
* The frequency it should save at. Currently, the callback supports saving at the end of every epoch, or after a fixed number of training batches.
* Whether only weights are saved, or the whole model is saved.

Note: If you get WARNING:tensorflow:Can save best model only with <name> available, skipping see the description of the monitor argument for details on how to get this right.

# 5. Conclusion

We can conclude our model by analyze the model accuracy on validation and testing data.

**Accuracy**

10/10 [==============================] - 902s 90s/step - loss: 0.2923 - acc: 0.8500 - val\_loss: 0.1277 - val\_acc: 0.9800

Epoch 00047: early stopping

**98.0 % validation Accuracy**

# Diagnostic plot

# 

# 

# Test the Model on Unseen Data

# 58/58 - 936s - loss: 0.1841 - acc: 0.9452

# Testing Accuracy is 94.5% which can be considered the best for our model