Introduction to Image Processing and Recognition

**Brain Tumor Detection using ConvNets**horizontal line

By: Saurabh Chauhan (18CSU201)

Tushar Jindal (18CSU218)

Vaibhav Saini (18CSU225)

horizontal line

# 

**Contents**

1. **Introduction**
   1. **Background**
   2. **About the dataset**
   3. **Convolutional Neural Networks**
2. **A high-level flow diagram of our methodology**
3. **Methodology**
   1. **Libraries used**
   2. **Pre-processing with data augmentation**
   3. **Building and training the model**
   4. **Testing the model**
   5. **Deployment using Streamlit, and Heroku cloud service**
4. **Results**
5. **Model’s Interpretability using SHAP**
6. **Discussion**

# Introduction

The fields of medical imaging and radiology have for long empowered doctors to make better diagnoses, with MRIs proving to be one of the most powerful diagnostic tools at any medical professional’s disposal since their serendipitous invention. They are non-invasive, and the staple imaging technique for both studying and diagnosing neurological conditions in particular. The key here is that after using these tools to get better information, the diagnosis is often made qualitatively by the radiologist’s judgement and experience.

Now consider the parallel field of artificial intelligence with its whole class of learning algorithms, inspired from our very own neurology, and how they have been advanced to become very performant at image recognition tasks. In the recent decades, Convolutional neural networks have proven themselves to be the best at a variety of computer vision and image processing tasks. Since then, many researchers have tried to feed the MRI scans into various learning algorithms and see how well they predict the likelihood of brain tumours in patients based on their MRI scans. A quick survey of the available literature tells us that CNN and many of its variants have indeed performed exceptionally well in detecting tumours and lesions in the brain.

**About the dataset:**

The dataset we have taken is from Kaggle, of 253 MRI scan images, labelled with ‘yes’ or ‘no’ for the presence or absence of brain tumor. [You may refer to it here.](https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection)

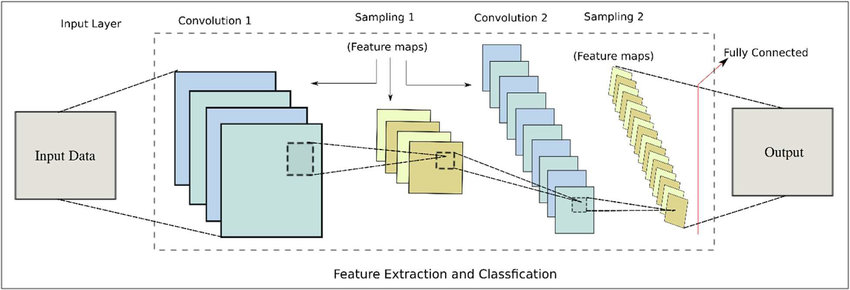
## Convolutional Neural Networks

CNNs are a class of deep neural architectures that have gained popularity in the past decade for computer vision and related classification and recognition tasks. But depending on the task’s implementation details, they may also have applications in natural language processing, and financial time series data.

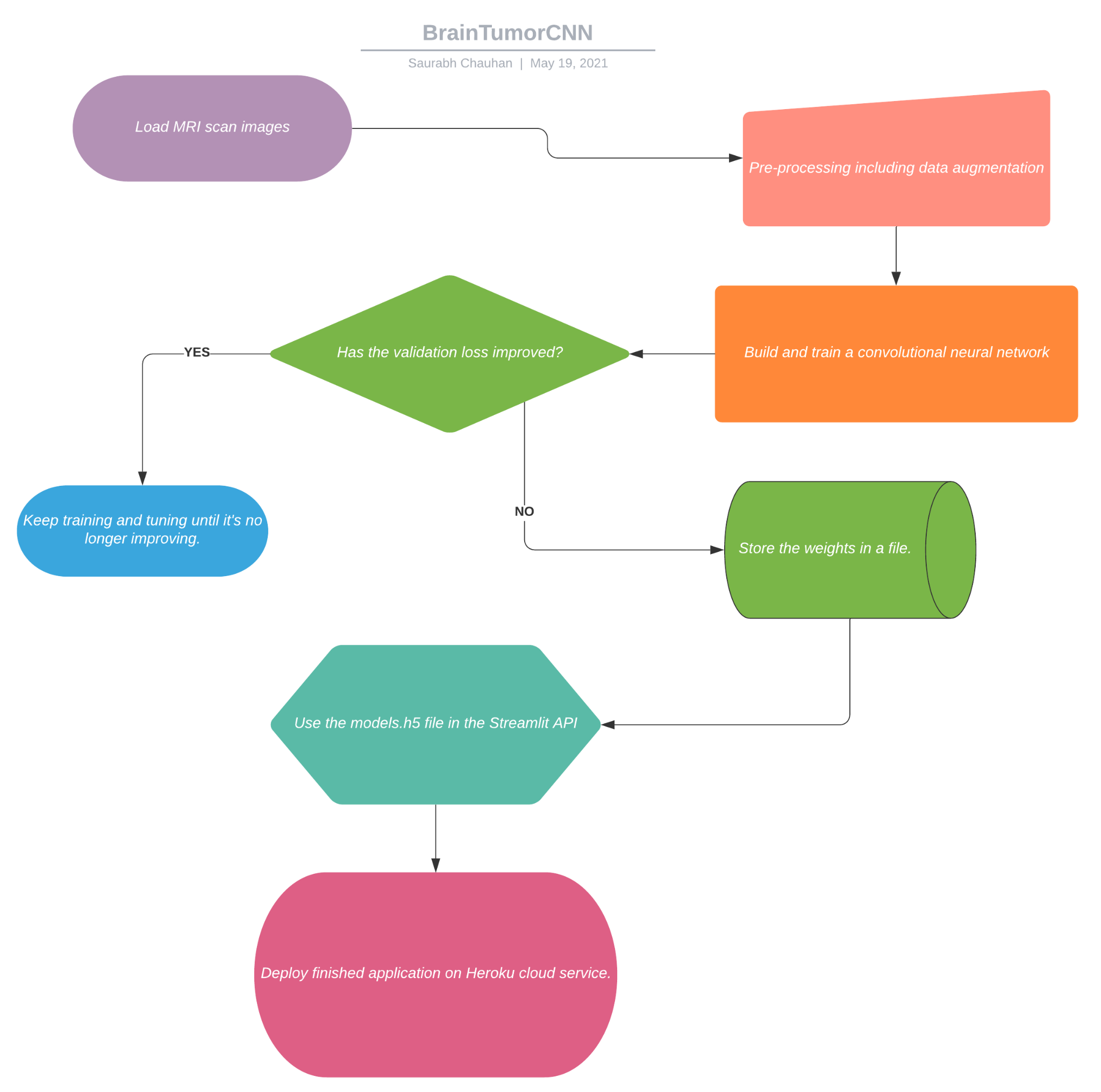
One of many architectures inspired from biological systems, CNNs differ from ordinary MLPs in that they’re not fully connected graphs; hence they’re not as prone to overfitting.

While the overfitting problem is fixed in other architectures through regularization, weight decay and many other techniques, CNNs accomplish this by recognizing hierarchies within data, and building on that complexity by layering simpler filters or kernels.

Out of all the other architectures, feeding data to CNNs requires the least amount of preprocessing. Its architecture makes it capable of optimizing for the right filters, therefore solving the problem of feature extraction.



### A high-level overview of our project



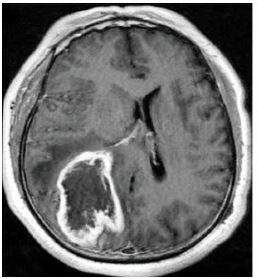
### 

### Methodology

Primarily, we will be using the open-source library for computer vision, OpenCV, to load and process our images.

Tensorflow and Keras will be invoked to train and fit our CNN model. The EarlyStopping function is used to prevent our model from overfitting the data, by setting an arbitrarily large number of training epochs and stopping training as soon as the performance on the validation set starts to stagnate.

Pandas and NumPy will be used for the manipulation of data frames and advanced array operations, and various functions from Seaborn and Matplotlib will be invoked for the inline visualizations. For our reference, let us first see what an MRI scan with a tumor looks like, and an MRI scan without tumors for contrast.

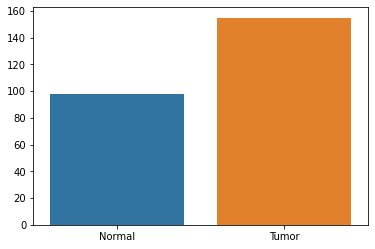


***1. An MRI scan with a clear brain tumor.***

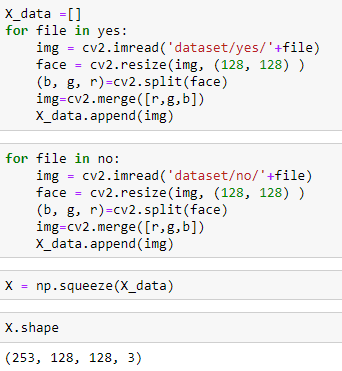


***2. MRI scan of a non-tumorous brain.***

Let’s now see the distribution of our samples among the two classes:



**Pre-processing**

As our scan images may be of different sizes, we need to first resize our images to some standard size. We have chosen 128 by 128 pixel grids for each image, as shown in the snippet of code below:

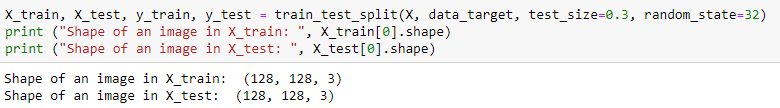
**As you can see, we use 2 for-loops to iterate through the images in each of the class folders, namely ‘yes’ and ‘no’, resize them to 128x128, and append them to the array X\_data.**

**squeeze() from NumPy is used to remove any redundant axes of length 1.**

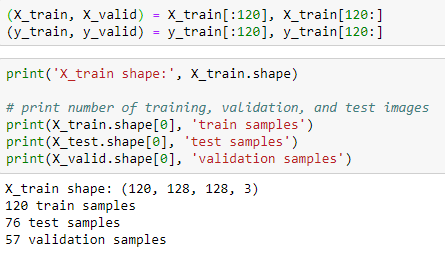
Before feeding these tensors of pixel values into our models, we first need to normalize the intensity values. So, we divide all the values in our ndarray by 255:

We typecast our ndarray to float as needed for the scaling.

Next, we split our dataset into training and testing sets, with a ratio of 70:30. We picked a constant seed value to ensure the same initialization/shuffling of data before splitting, when our model is run on different machines/multiple calls on the same machine.



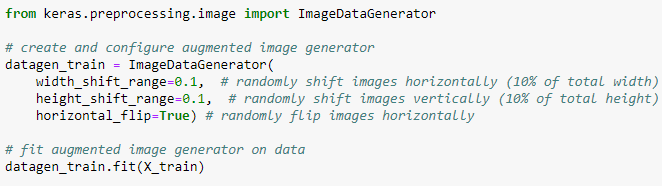
Now, for the validation set:



**Data Augmentation**

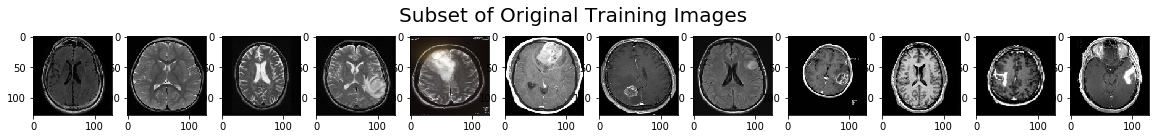
Before we go on to train a CNN model, we also need to consider the volume of data we feed it. Another thing to consider is that the algorithm might process an image differently based on its orientation. Data Augmentation is about generating new images by either cropping, padding, flipping or rotating the existing images. In this way, our model has more data to work with without requiring any new real world sampling.

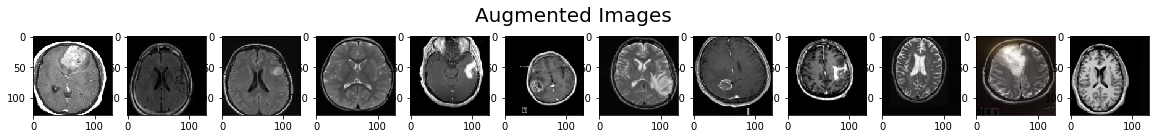
To do that, we will invoke ImageDataGenerator() from tf.keras.preprocessing.image :



We could customize these parameters to generate differently new data that might change the model’s performance, and so a bit of experimentation is necessary at this stage.

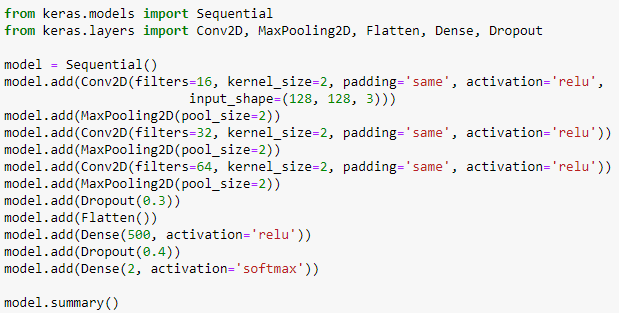
To compare the original images and the augmented images:





**Constructing the model’s architecture**

From Keras.models we import a Sequential model, and from keras.layers we call Conv2D(), MaxPooling2D(), Flatten(), Dense() and Dropout().

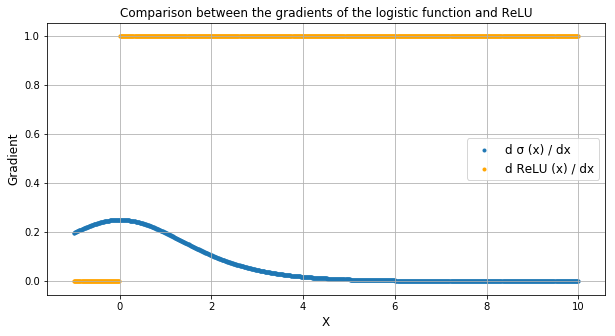


**Why use ReLU?**

The rectified linear unit function, or ReLU, has many benefits over non-linear functions like the sigmoid. For one thing, it is much less computationally intensive. As we keep adding more layers of ReLU, the computing cost only scales linearly, while computing the gradients of sigmoid layers scales exponentially.

But more importantly, ReLU reduces the problem of “vanishing gradients” during the backpropagation step of the training. This problem is about gradients of a neuron approaching zero for higher values of input. ReLU simply has a derivative of either 0 or 1, depending on the sign of the input.

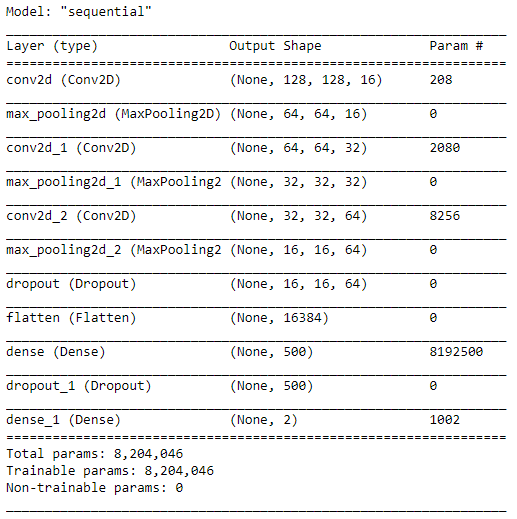
To better understand why this overcomes vanishing gradients, see below:



**Why use Dropout?**

This layer acts as a mask to remove the contributions of some neurons based on some threshold. If this is applied to the input vector, this can filter out some of the unnecessary features. If within the hidden layers, it nullifies some neurons from contributing to the output. In this way, we can prevent overfitting the training data. If we didn’t use Dropout, our model would learn some features from the first few samples it trains on, and keep using those features to bias its learning of features that may only appear in the later samples. Here we have tuned the Dropout parameter to be around 0.3.

In summary, our model’s parameters are initialized as:

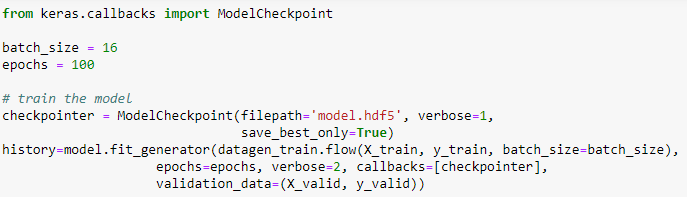


Since our classes are mutually exclusive, we will compile our model using sparse categorical cross entropy as our loss.

As our classes are sparse, we will use Adam optimizer for stochastic gradient descent.



**Now, we are ready to train the model**

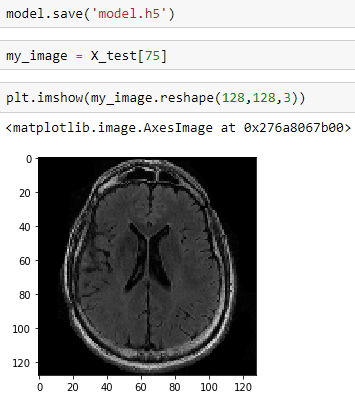


We have used ModelCheckpoint() from keras.callbacks to be able to deploy the best version of our model into a web application hosted on the cloud service Heroku.

This will save the parameters of the network into a file named ‘model.hdf5’.



It stopped training after 99 epochs as the validation loss was not improving.



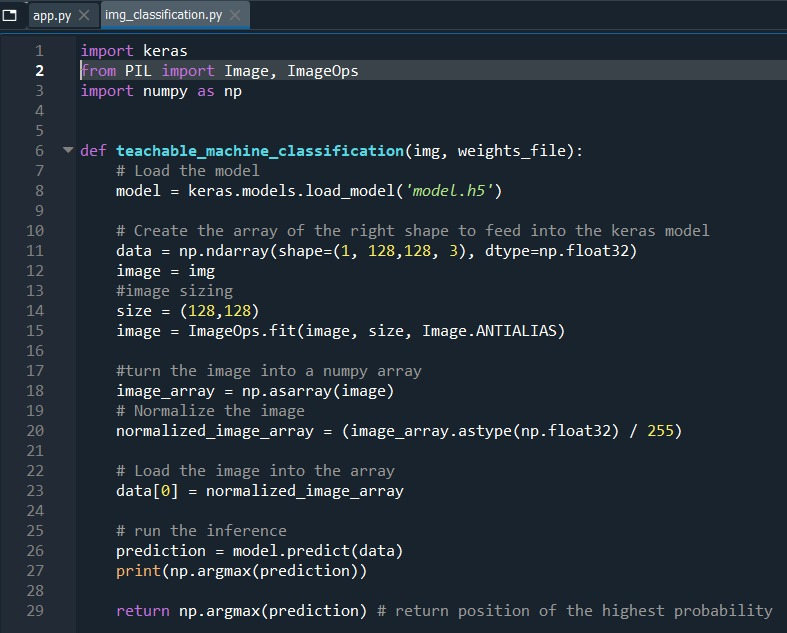
Now to make this network ready for deployment, we just need to save these parameters into a file ‘model.h5’.

**Deployment**

Now we need to import our pre-trained model into an application that will ask the user to input an image of MRI scans, and the application would predict whether or not the scan is tumorous.

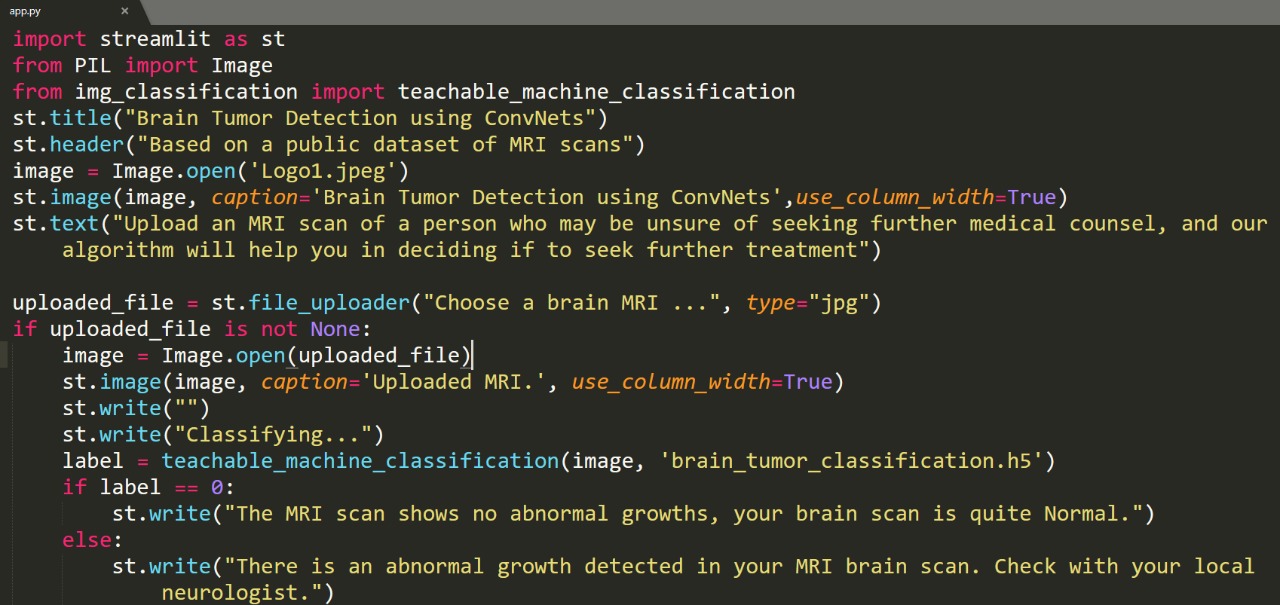
To do that, we have used the Streamlit framework to create an application using pure Python. Streamlit is essentially an open-source framework developed for machine learning and data science projects.

Then, we hosted our application on the cloud service Heroku. Heroku is a platform as a service(PaaS), that enables developers to build, run and deploy applications entirely on the cloud. To take a look at the source code for our application:



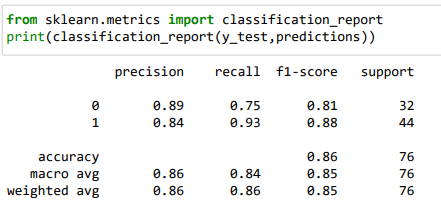
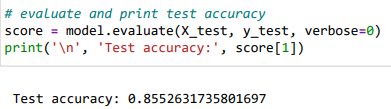
Here we have defined a function called teachable\_machine\_classification() that takes as input an image from the user, and the weights file ‘model.h5’, and outputs the prediction of the class with the highest probability.

Now, the same function will be imported on to our main app.py file that describes the front end interface of our application. It can be seen called below:



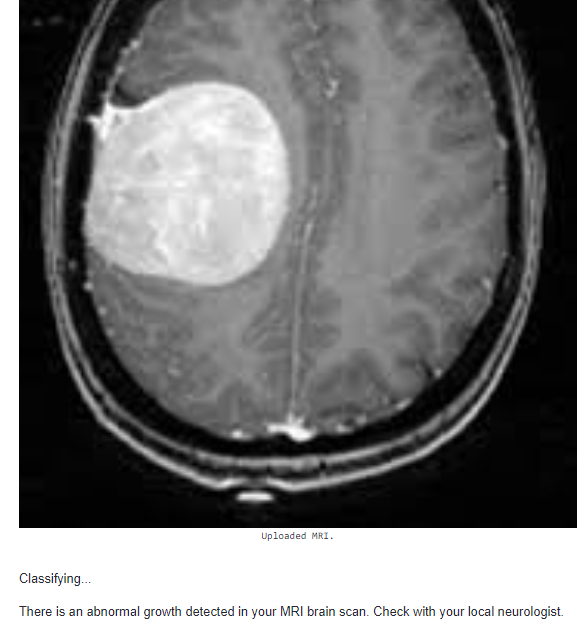
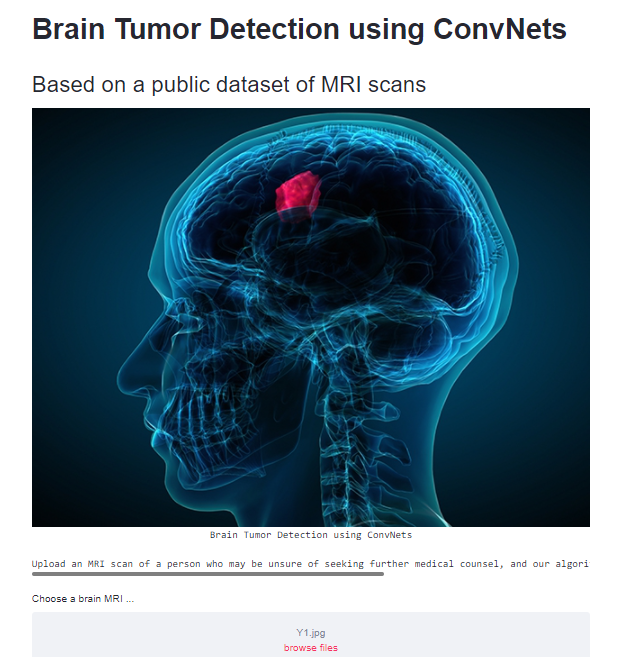
[***You can try out our application by clicking right here***](https://brain-tumorclassification.herokuapp.com/)***!***

**Results**

As we can see, our model made accurate predictions on the testing set 85.52% of the time.

Let’s take an example picture and run it on our web application. I will use this image and feed it into the pre-trained network.

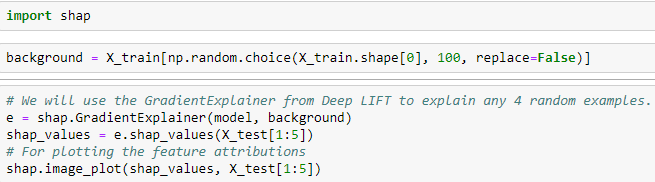
**Output:**

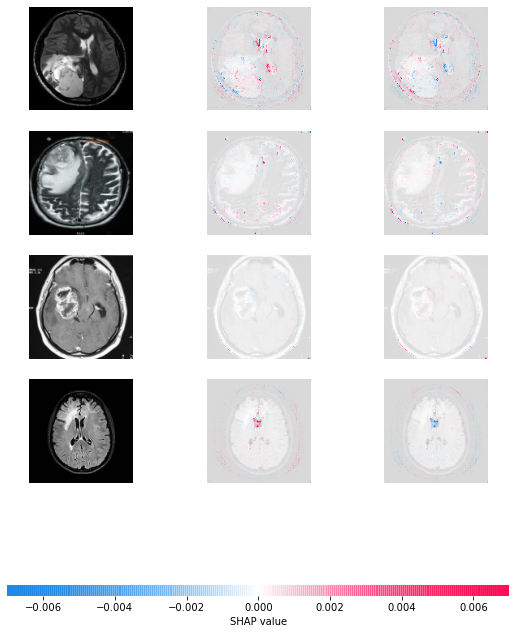


As you can see, the algorithm detects an abnormal growth in the scan and recommends that the user check with a neurologist.

**Model’s Interpretability using SHAP**

SHAP is yet another open-source library that we’ve invoked to make our model more interpretable. It’s a game theoretic approach to explaining the outputs of any machine learning model, and the name SHAP comes from the game theoretic concept of Shapley values, which is a way of attributing the contribution of different features to the net output.

We’ve imported shap and picked some random examples from the set. Then we used the GradientExplainer from Deep LIFT to explain those outputs in terms of feature attributions, and plotted those Shapley values. Red pixels indicate positive attribution to the output, and blue indicate negative attribution.



As we know, machine learning models are practically black boxes to us, and we don’t always know how the machine achieved its task and what configuration of the network led to it. SHAP offers us a way of looking under the hood and plotting the attributions of each feature across different layers of a network.

SHAP can offer us interesting tools to compare how a machine learns and makes decisions, compared to how a human doctor would make the same judgements based on intuited learning from long experience.

**Discussion**

Along the same line of inquiry as the steps we’ve walked you through in this report, some compelling evidence is lining up to the possibility that in the future, algorithms might hope to replace the diagnostician’s judgement as experts can often suffer from tunnel vision and other perfectly human cognitive biases, that machines do not. Artificial neural networks are capable of learning patterns with much greater resolution- subtle texture changes in the brain that the expert might miss, an algorithm might not.

In their infancy, these classes of self-learning algorithms might be nothing more than a diagnostic aid to the medical professionals, but as they’re applied more responsibly and diligently, informed by the neurologist’s insight, might turn into cutting-edge tools of scientific discovery.

There are many neurological conditions that are yet to be understood and treated, and although the scope of this project is classification of tumors alone, these same algorithms can be used to extract novel features using segmentation, and leveraging the machine’s superior ability for pattern recognition and processing inhuman amounts of data.