Classification of tumor and non-tumor from Brain MRI using Convolution Neural Network

**Introduction**

We already heard of image or facial recognition or self-driving cars. These are real-life implementations of Convolutional Neural Networks (CNNs). A specific kind of such a deep neural network is the convolutional network, mostly applied to analyze visual images. Deep learning is a subfield of machine learning that is inspired by artificial neural networks, which in turn are inspired by biological neural networks.

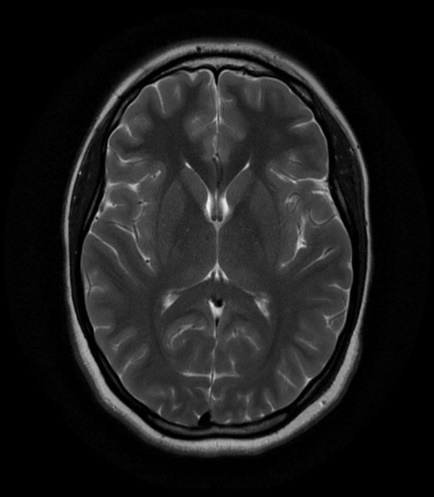
Here we use CNN to classify an **MRI Scan** of a patient’s brain obtained in the axial plane as whether there is a presence of tumour or not.

**Brain MRI Dataset**

The dataset, I’ll be using to create CNN model, is obtained from the Kaggle link

<https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection>

Here are the sample pictures, so we can see what an MRI scan looks like with tumour and without one.



Pic 1

Pic 2

We see that in the pic 1, to the left side of the brain, there is a tumour formation, whereas in the pic 2, there is no such formation. So, we can see that there is a clear distinction between the two images.

**Short Overview on Convolutional Neural Networks**(**CNN**).

CNNs specifically are inspired by the biological visual cortex. The cortex has small regions of cells that are sensitive to the specific areas of the visual field. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.

CNN is a deep, feed-forward artificial neural network.  feed-forward neural networks are also known as multi-layer perceptrons (MLPs). The models are called "feed-forward" because information flows right through the model. There are no feedback connections in which outputs of the model are fed back into itself.

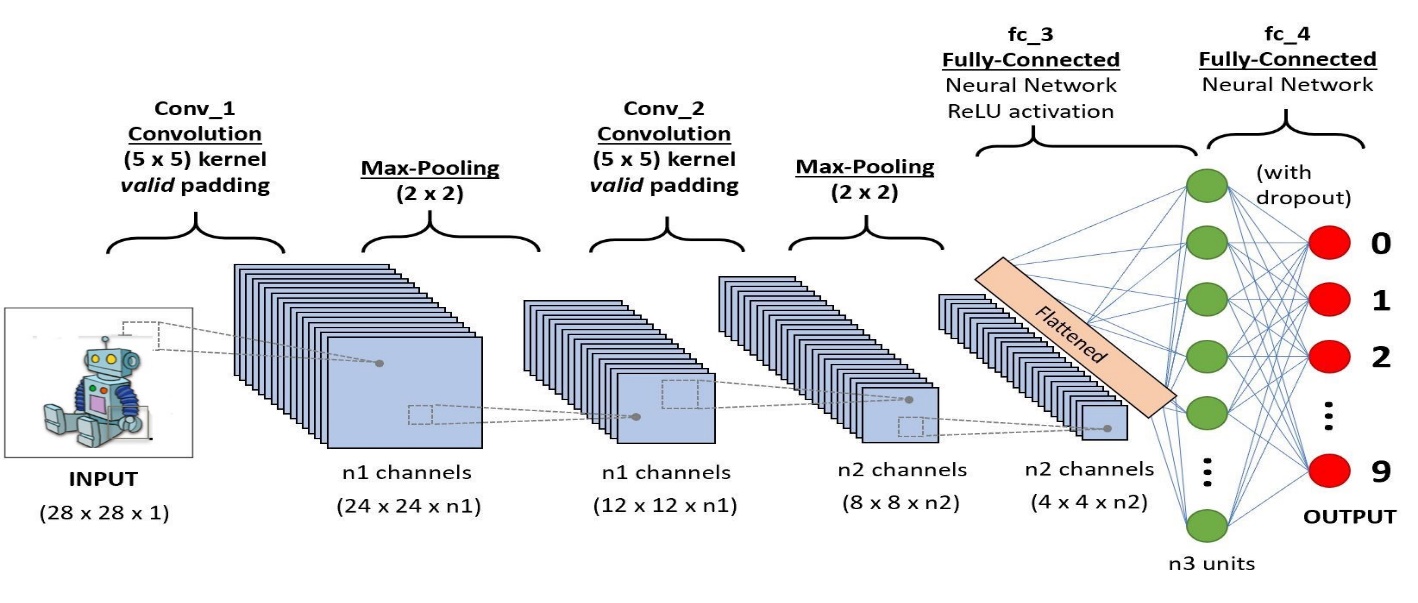


Figure 1 Convolutional Neural Network from

The image shows that feed an image as an input to the network, which goes through multiple convolutions, subsampling, a fully connected layer and finally outputs something.

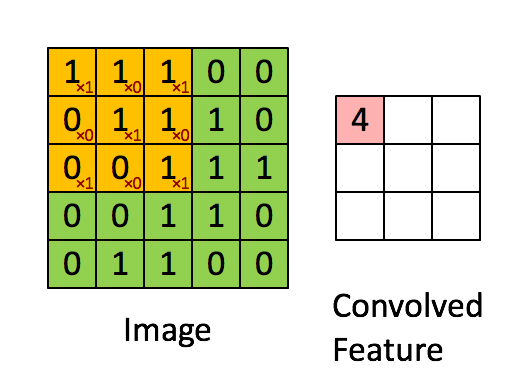
1. **Convolution Layer**

Figure: Convolution

An image is nothing but a matrix of pixel values. In other words, imagine an image represented as a 5x5 matrix of values, and we take a 3x3 matrix and slide that 3x3 window or kernel around the image. At each position of that matrix, we multiply the values of your 3x3 window by the values in the image that are currently being covered by the window. As a result, you'll get a single number that represents all the values in that window of the images. We use this layer to filtering as the window moves over the image, we check for patterns in that section of the image.

1. **Pooling Layer**

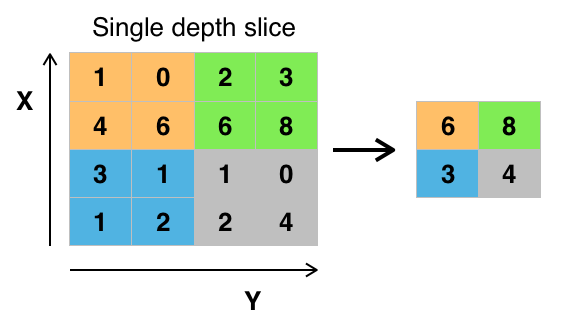
The objective of subsampling is to get an input representation by reducing its dimensions, which helps in reducing overfitting. One of the techniques of subsampling is max pooling. With this technique, we select the highest pixel value from a region depending on its size. In other words, max pooling takes the largest value from the window of the image currently covered by the kernel. For example, we can have a max-pooling layer of size 2 x 2 will select the maximum pixel intensity value from 2 x 2 region. The pooling layer then works a lot like the convolution layer, the only difference is the function that is applied to the kernel and the image window isn't linear.

Figure: Max-Pooling

1. **Fully Connected Layer**

The objective of the fully connected layer is to flatten the high-level features that are learned by convolutional layers and combining all the features. It passes the flattened output to the output layer where we use a softmax classifier or a sigmoid to predict the input class label.

**CNN Model Development**

**Part 1 - Data Pre-processing**

In data pre-processing we have three steeps

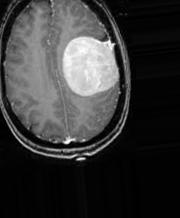
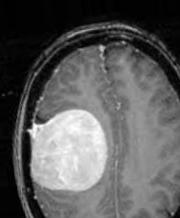
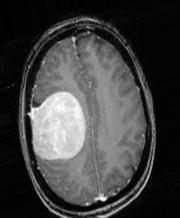
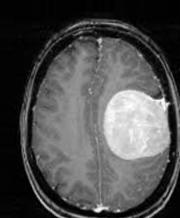
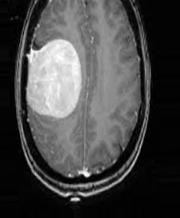
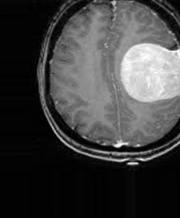
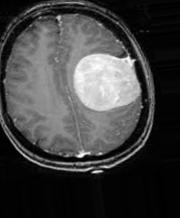
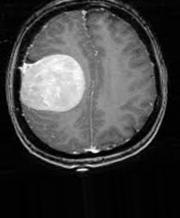
**Data Visualization**

In the first step, we will analyze the MRI data. 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.

In our example, we have 253 images with 155 belonging to the “yes” class and 98 belonging to the “no” class. Which create a new problem known as data imbalance. ***Data imbalance*** is where the number of observations per class is not equally distributed (here, we have 155 belonging to “yes’ class and only 98 belonging to the “no” class). Our Neural Network not be given enough training in the “no” class.

**Data Augmentation**

In order to solve this, we use a technique called ***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 approximately equal number of images to both classes.



Data Augmentation

From the above images, we can see the various augmentation that has been applied to an MRI image in the “yes” class. In this way, we augment all the images of our dataset.

**Splitting the data**

In the next step, we split our data to training set and test set. 80% of the images will go to the training set, which will be used by our neural network to get trained. The remaining 20% will go to the test set, with which we will apply our trained neural network and classify them to check the accuracy of our Neural Network.

**Part 2: Building the CNN Model**

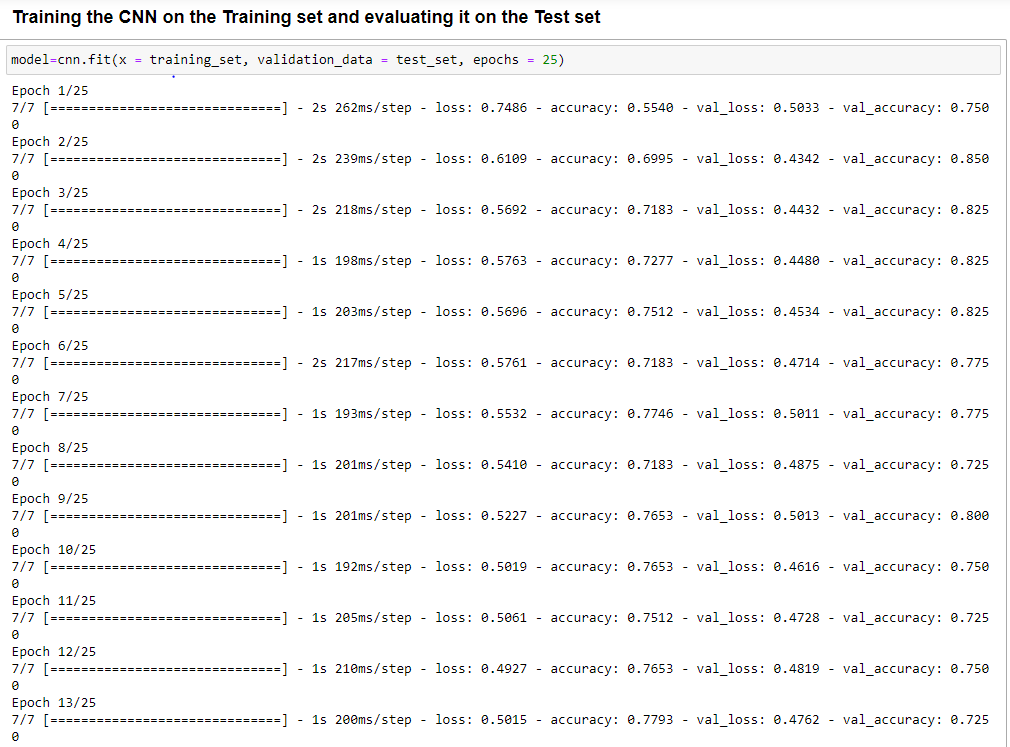
Now we design a neural network using *Keras*library with various convolutional and pooling layers.



**Part 3: Training the CNN model**

Finally, we come to the stage where we will fit the image data to the trained neural network.

We will train the images data for around 30 “***epochs***”. An epoch can be thought of an *iteration* in which we feed the training images again and again for the Neural Network to get trained better with the training images.



From the above image, you can see that the “***acc***” [Accuracy] of the training set keeps improving with each iteration. This means that the Neural Network model is able to improve in classifying the image as Tumor or Not a Tumor. Keep a note of the “***val\_acc***” [Validation Accuracy] which indicates the accuracy of the model on the test\_set.

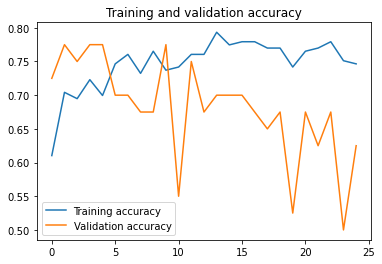
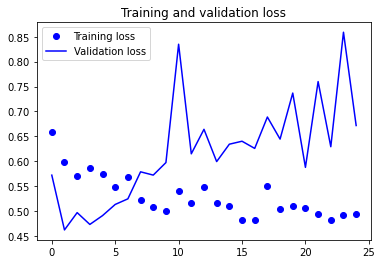
At the end of the 25th epoch, we see that the trained CNN model has a validation accuracy of **80%**. 

It denotes that this Neural Network can correctly classify about 80% of the test set images as Tumor or Not a Tumor.

**Part 4: Analysis of the CNN model**

After the training, we finally plot the “accuracy” and “loss” of both “train\_generator” and “validation\_generator” for all the 25 epochs(iterations).

Accuracy and Loss Graph



From the above two plots, you can see that the validation accuracy almost became stagnant after 4-5 epochs and rarely increased at certain epochs. In the beginning, the validation accuracy was linearly increasing with loss, but then it did not increase much.

The validation loss shows that this is the sign of overfitting, similar to validation accuracy it linearly decreased but after 4-5 epochs, it started to increase. This means that the model tried to memorize the data and succeeded.

With this in mind, it's time to introduce some dropout into our model and see if it helps in reducing overfitting.

**Conclusion**

Here we create a CNN model to classify tumour, non-tumour MRI brain image using keras library with validation accuracy 80%, using this model, we can feed in an individual MRI image and check whether it has a tumour or not.