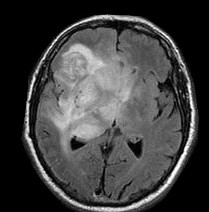
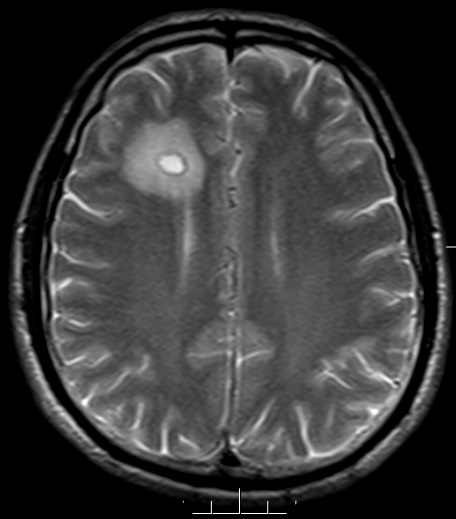
**INTRODUCTION**

The brain tumors, are the most common and aggressive disease, leading to a very short life expectancy in their highest grade. Thus, treatment planning is a key stage to improve the quality of life of patients. Generally, various image techniques such as Computed Tomography (CT), Magnetic Resonance Imaging (MRI)and ultrasound image are used to evaluate the tumor in a brain, lung, liver, breast, prostate, etc. Especially, in this work MRI images are used to diagnose tumor in the brain. However, the huge amount of data generated by MRI scan thwarts manual classification of tumor vs non-tumor in a particular time. But it having some limitation (i.e) accurate quantitative measurements is provided for limited number of images. Hence trusted and automatic classification scheme are essential to prevent the death rate of human. The automatic brain tumor classification is very challenging task in large spatial and structural variability of surrounding region of brain tumor. In this work, automatic brain tumor detection is proposed by using Convolutional Neural Networks (CNN) classification. The deeper architecture design is performed by using small kernels. The weight of the neuron is given as small. Experimental results show that the CNN archives rate of 89.2% accuracy with low complexity and compared with the all other state of arts methods.

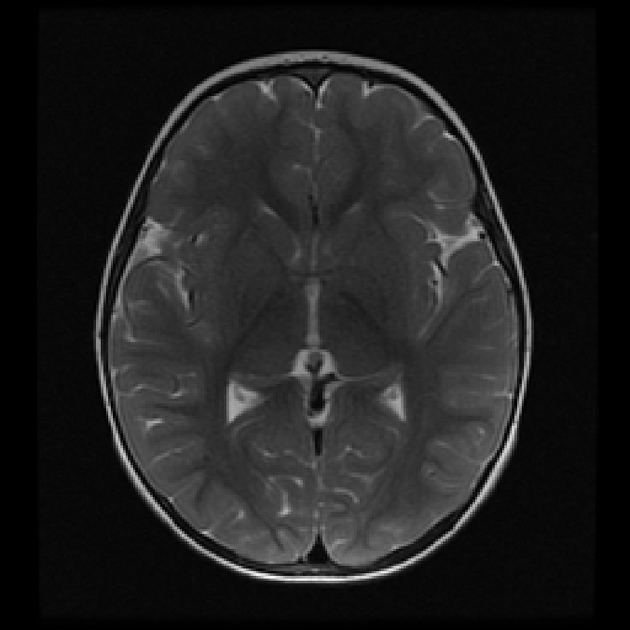
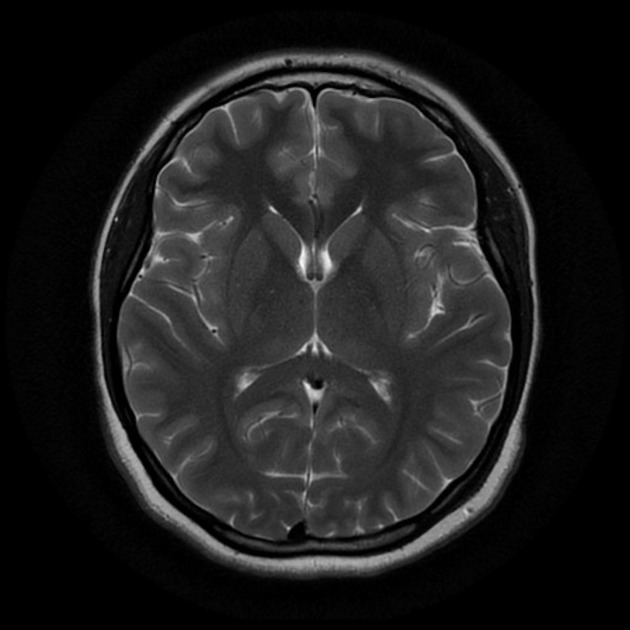
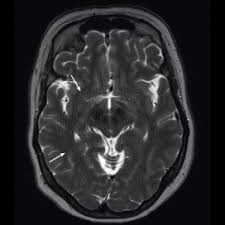
**ANALYSIS**

**DATASET DESCRIPTION**

* The image data that was used for this problem is Brain MRI Images of brain scans and come from various sources.
* The dataset contains the images in two groups, first one containing images that were diagnosed with a Tumor and the second group of images that weren’t diagnosed with tumor.
* The decision is given as:
  + YES, for tumor is present. Encoded as 1. COUNT: 155
  + NO for tumor is not present. Encoded as 0. COUNT: 98
* The count of images in each group and sample of the same are given below in Figure 1 and Figure 2.



**Figure 1:** Tumor – Yes, encoded as 1



**Figure 2:** Tumor – No, encoded as 0

**DATA PRE-PROCESSING**

Since the source of data is from multiple sources, size of the images in the data set are not fixed and varies throughout. Reshaping is required to process the images according to model expectations, during the training phase, and is achieved by using the code given below:

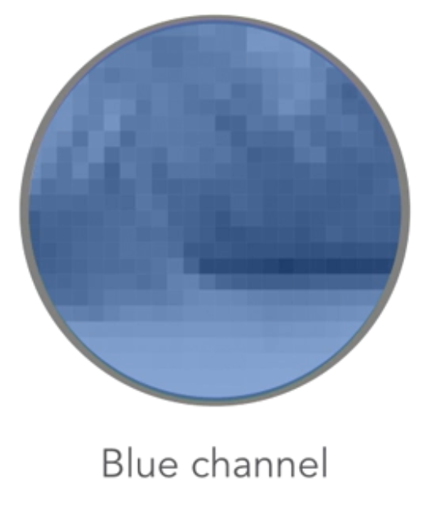
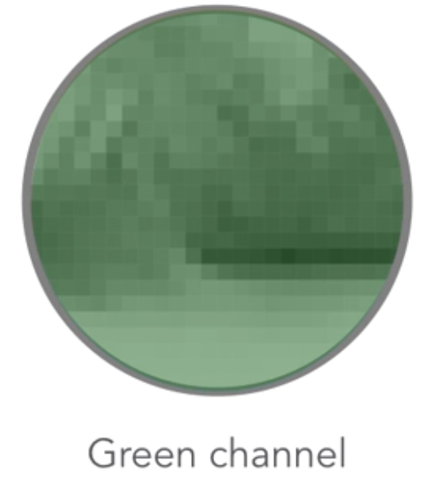
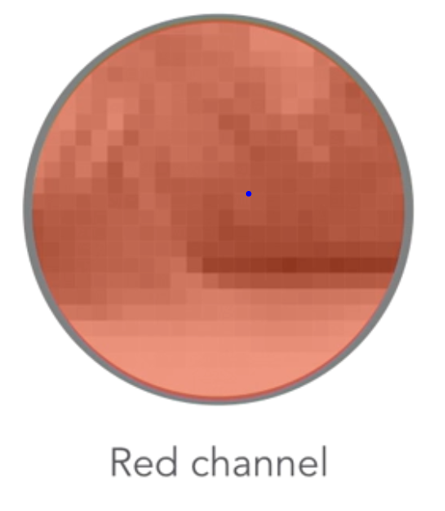
X\_train = X\_train.reshape(60000,28,28,1)

Where, the first number is the number of images, then comes the shape of each image (28x28), and the last number is 1, which signifies that the images are greyscale.

Each image is made of layers of color code, i.e., RGB and the based on the shade of the color of each pixel they are assigned some numerical value, higher value for lighter shades and a lower value for darker shades. The combination of values for each pixel in each layer added together gives the final numerical value, which the algorithm reads to determine the presence of tumor.

**IMAGE RECOGNITION**

The image recognition takes in images in form of a numerical array, where each pixel is allotted a specific value according to the color code. Each image is made up of three layers of Red, Green and Blue, and combination of the final value depends on the value of pixels for each layer. The division of the layer is given below in Figure 3.



**Figure 3:** Division of each pixel in RGB layers

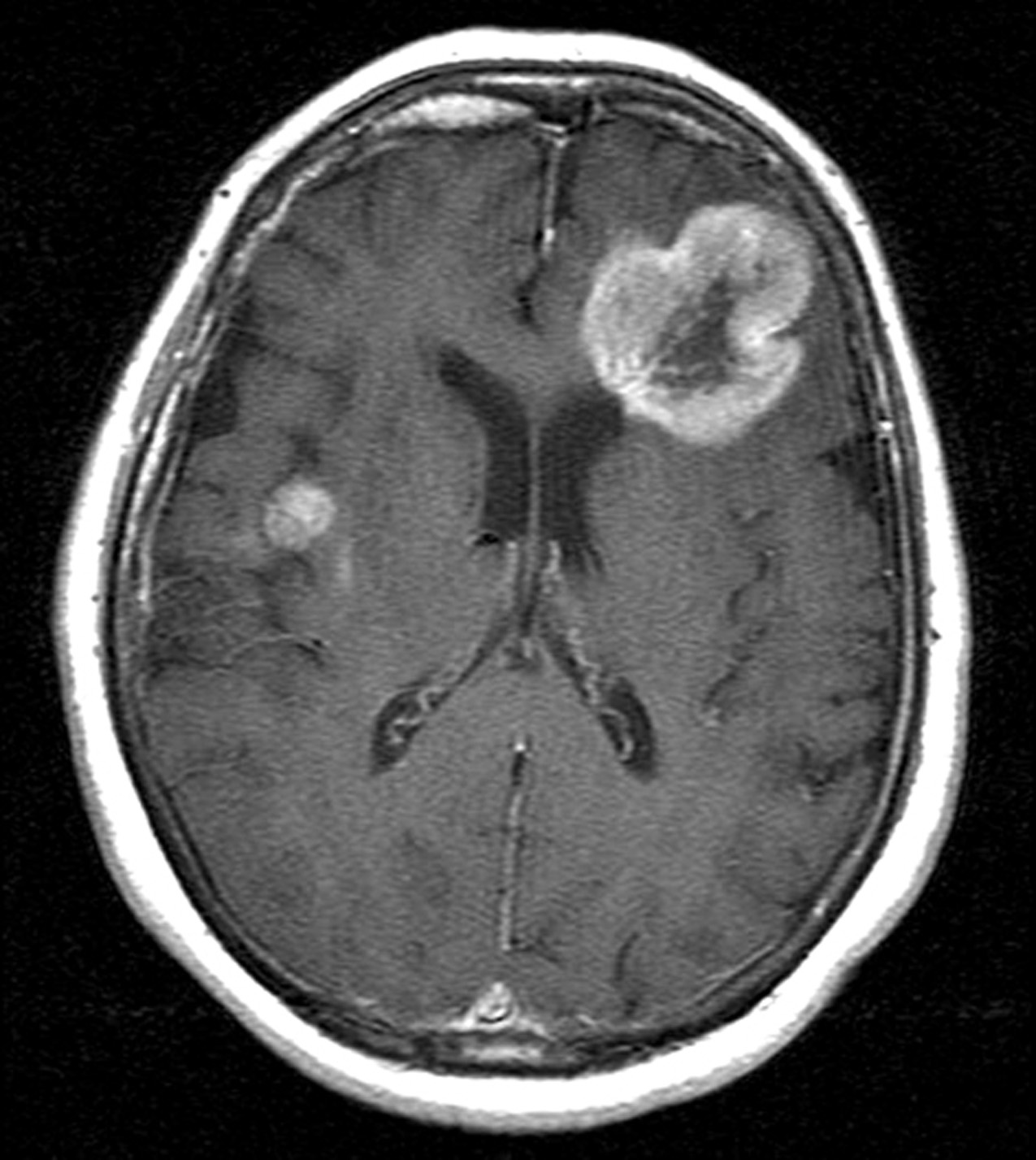
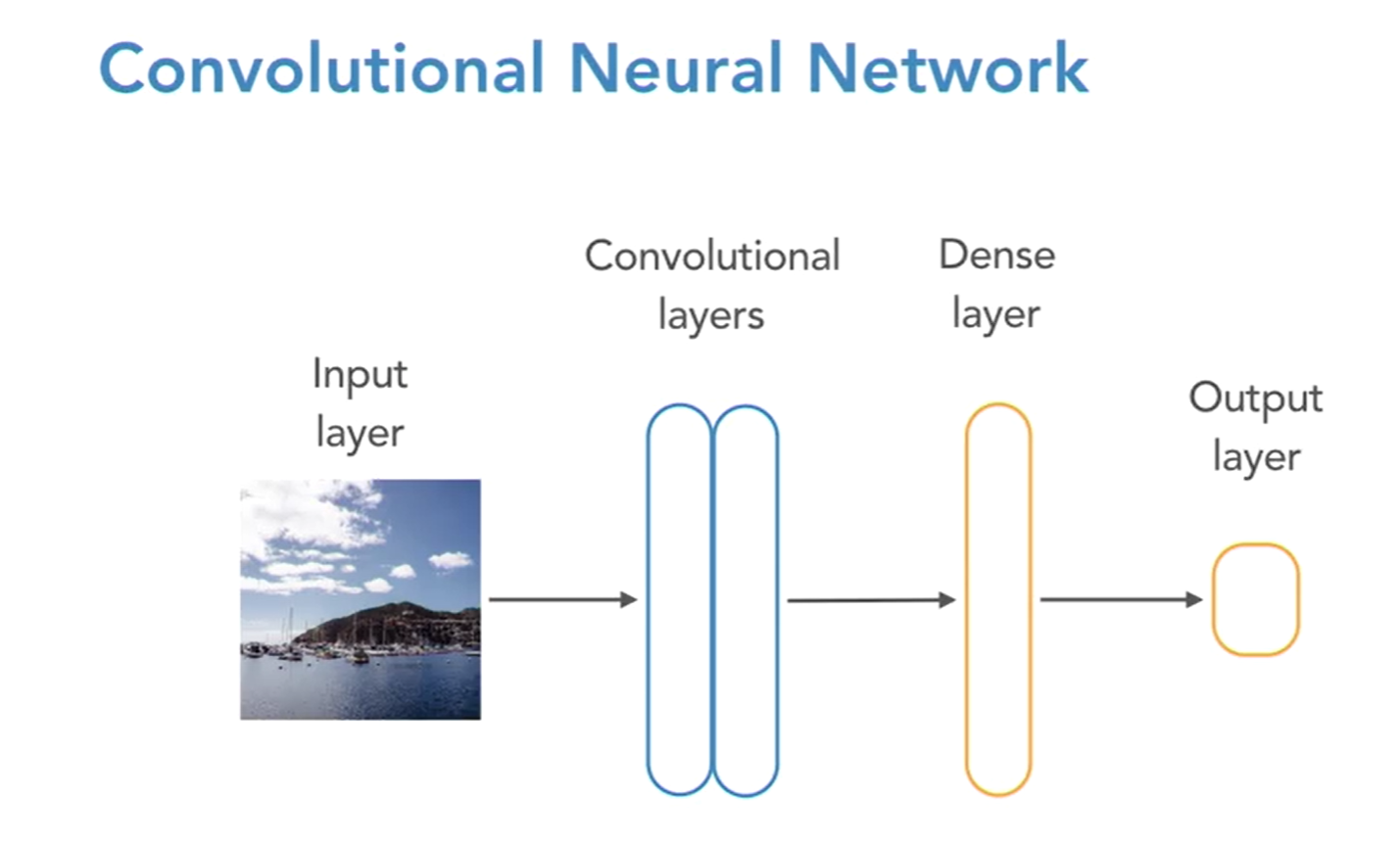
**CNN (****Convolutional Neural Network)**

In deep learning a CNN is a class of deep neural networks, most commonly applied to analyzing images by converting them to numerical values and creating decision trees based on the weights and values leaned from the training dataset. A Convolutional Neural Network has the following layers:

* **Convolutional layer:** The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter.
* **Pooling layer:** Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum.
* **ReLU layer:** ReLU is the abbreviation of rectified linear unit, which applies the non-saturating activation function. It effectively removes negative values from an activation map by setting them to zero. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.
* **Loss layer:** The "loss layer" specifies how training penalizes the deviation between the predicted (output) and true labels and is normally the final layer of a neural network. Various loss functions appropriate for different tasks may be used.

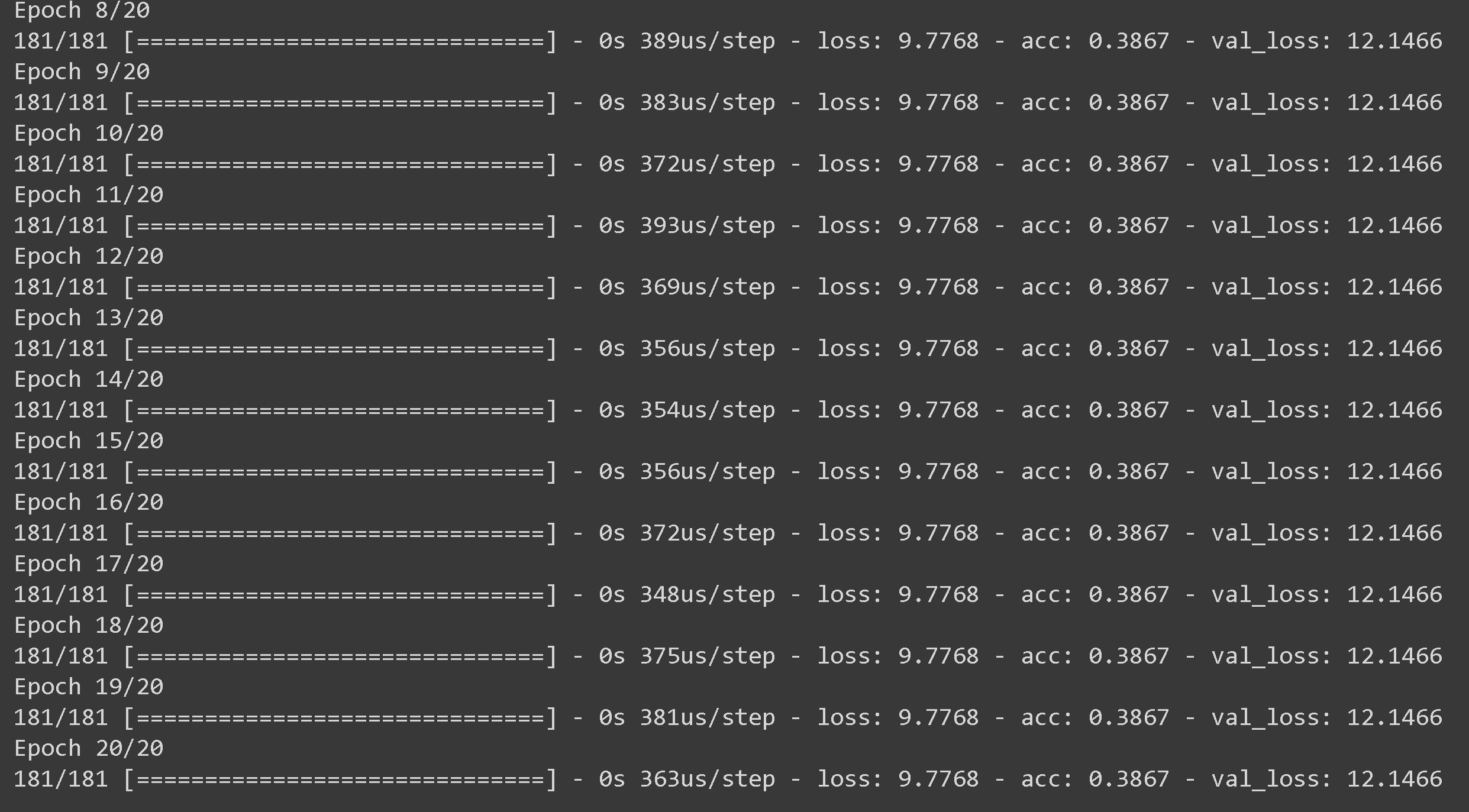
**METHEDOLOGY**

**STEP 1: Creating the dense layer**



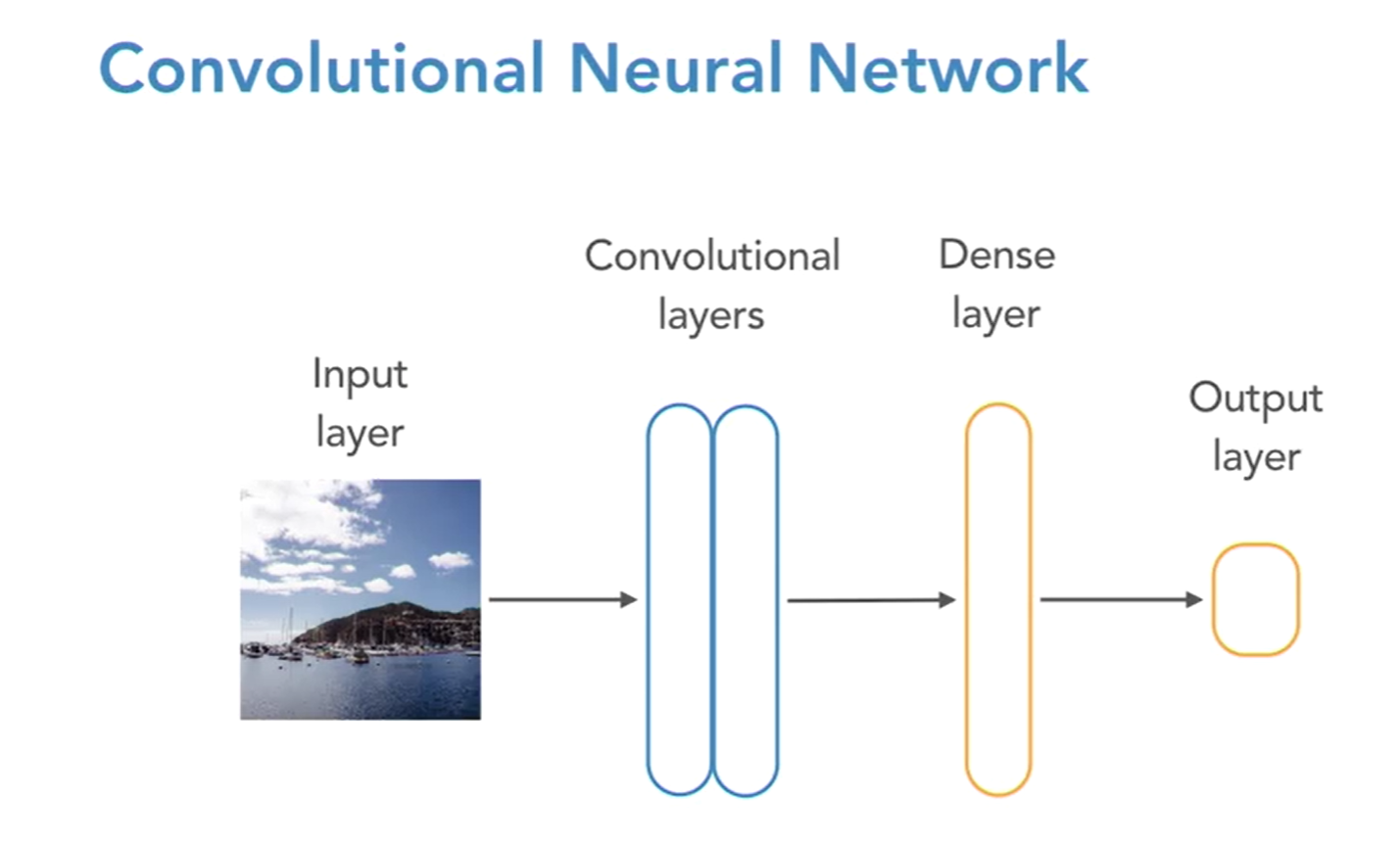
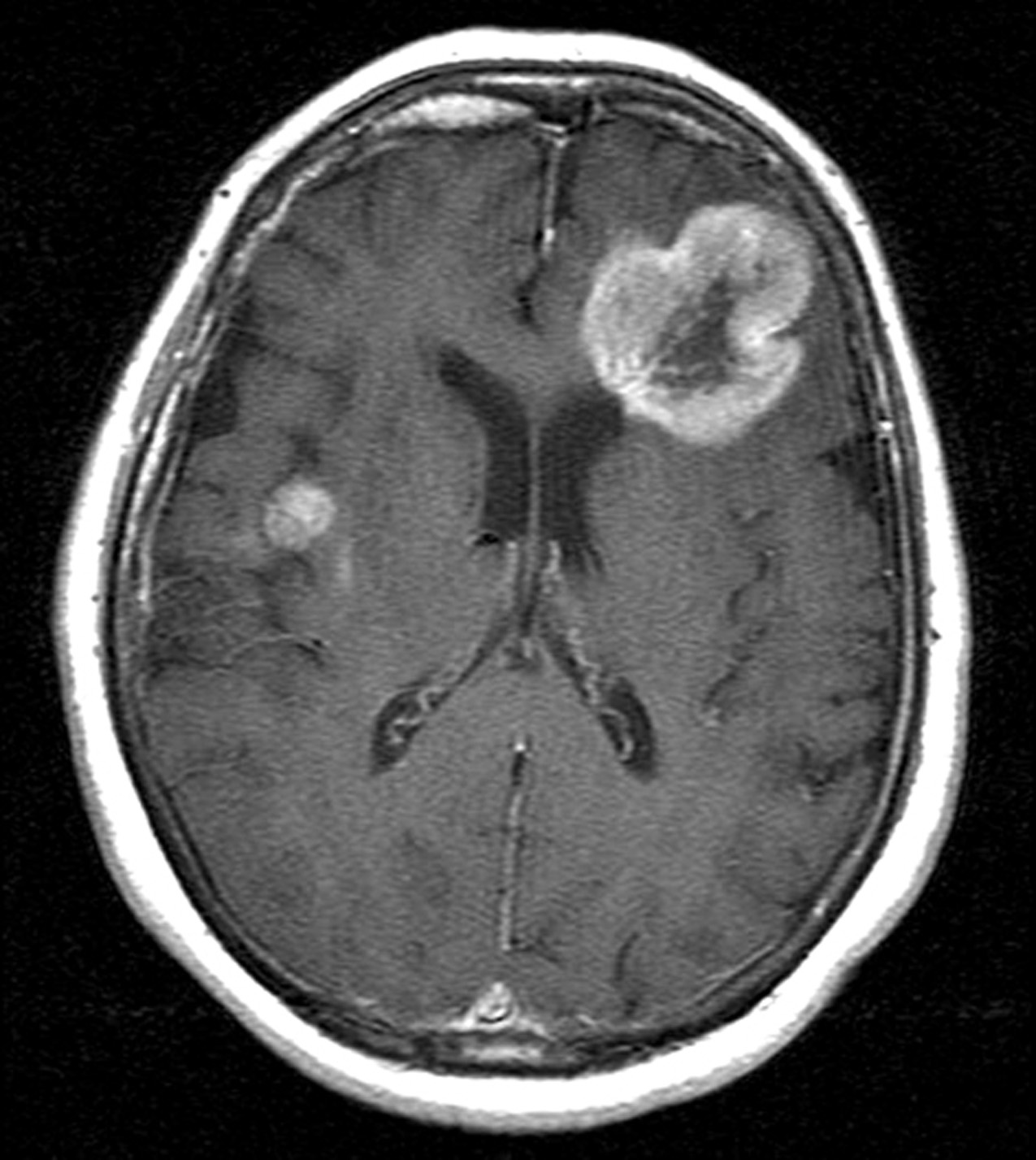
In the first step, we will pass the image into a dense layer. The image is first divided into numbe rof pixals each pixal assigned with unique number. These numbers of the image are passed through the Dense layer where an algorithm is being applied to identify the pixal density in the image.



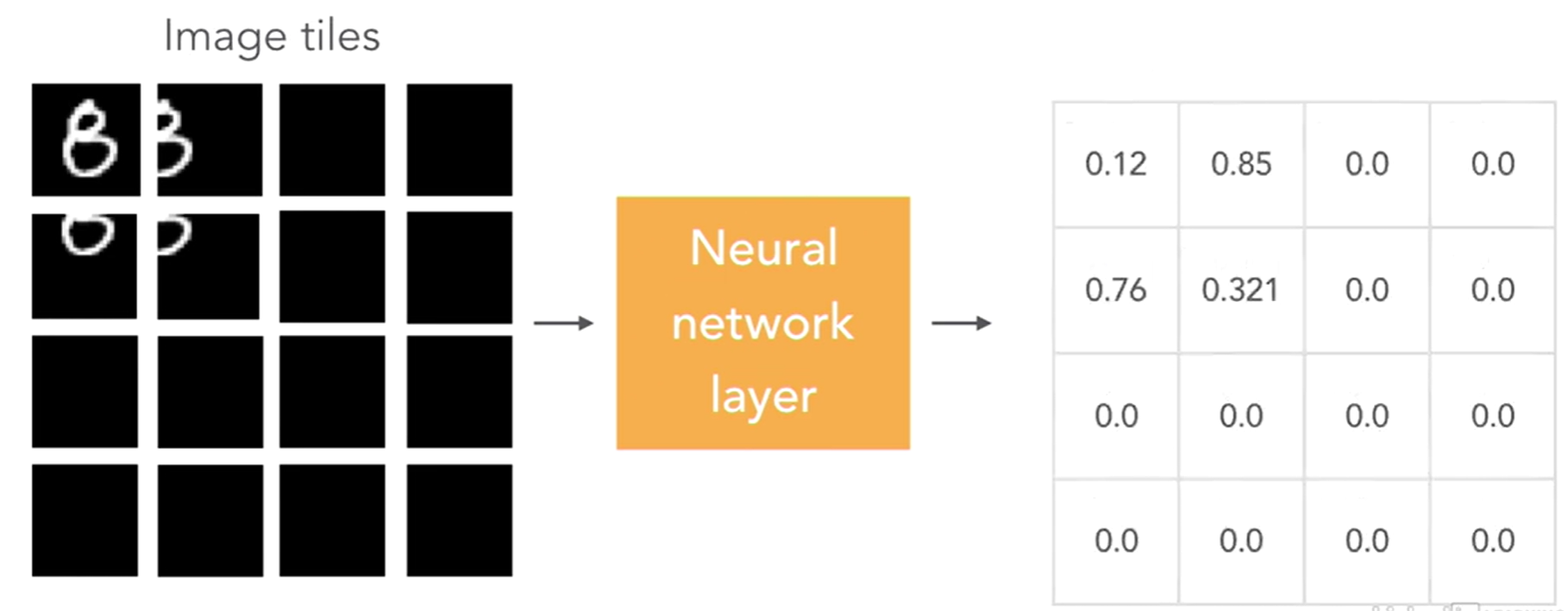
we can see from the above figures, the image has been passed through the 512 nodes inside the dense layer. We see that we found about 8.3 million parameters are being created and used to find out the desired output.

In the second figure, we can see the accuracy of the model is 38% which is very low for the model. So, inorder to get better ouptut and image recognition, we will add more layers and algorithms model

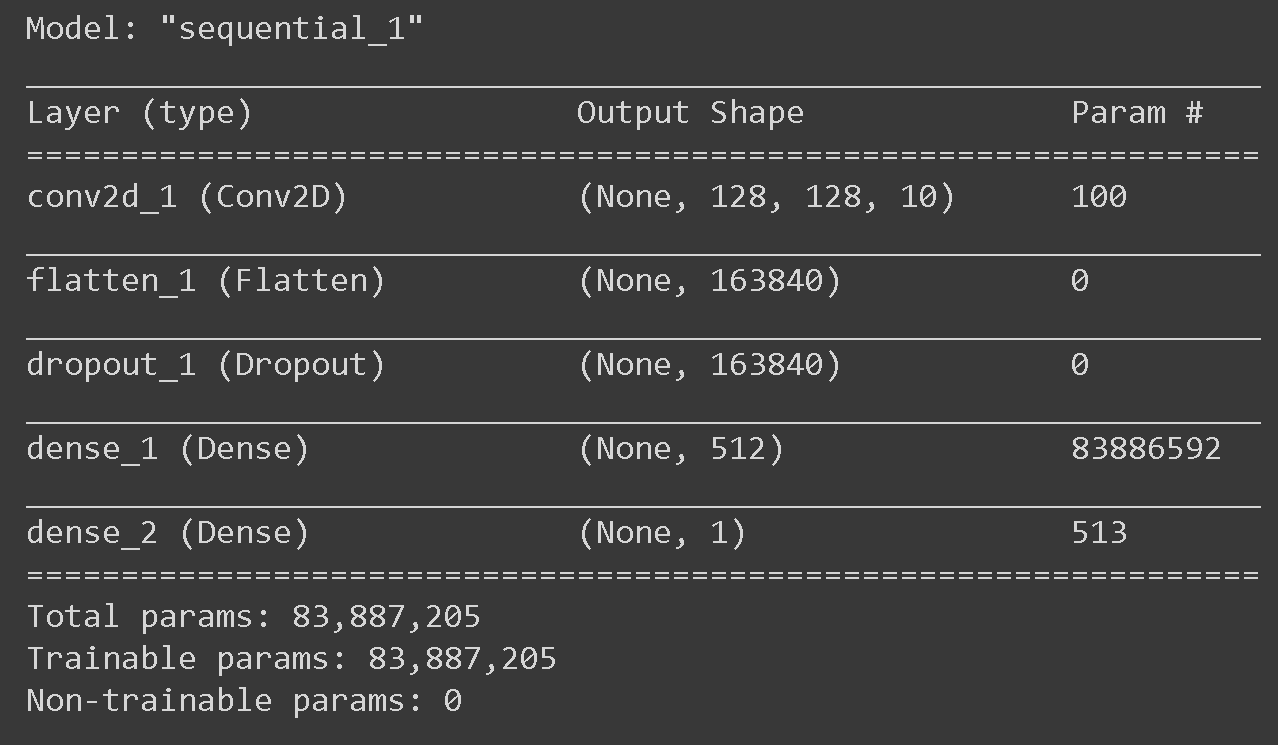
**STEP 2: Convolutional layer**



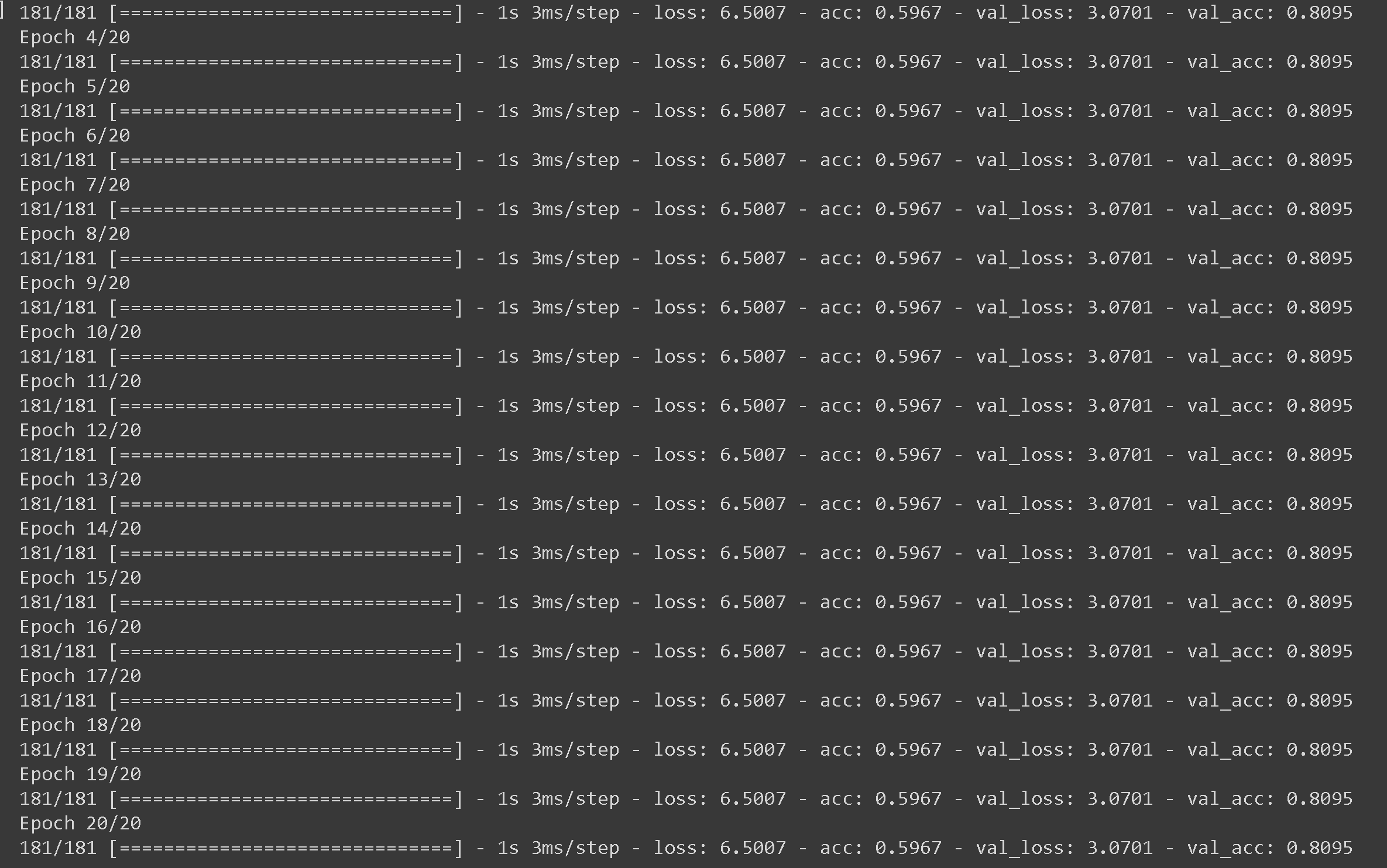
The above figure shows the layout of our model. We pass the image through the convolutional layer. Convolutional layer divides the image into pixels and will assign the numeric value to the pixels. The below image shows how the values are assigned for the image we give as an input.



Here, the pixels where the tumour exists are assigned values and other pixels that dosent have will be assigned lower values or 0. So the pixels that are assigned only will be sent through the other layer for further detection. This way we can improve the results and have better accuracy in detecting the tumour.

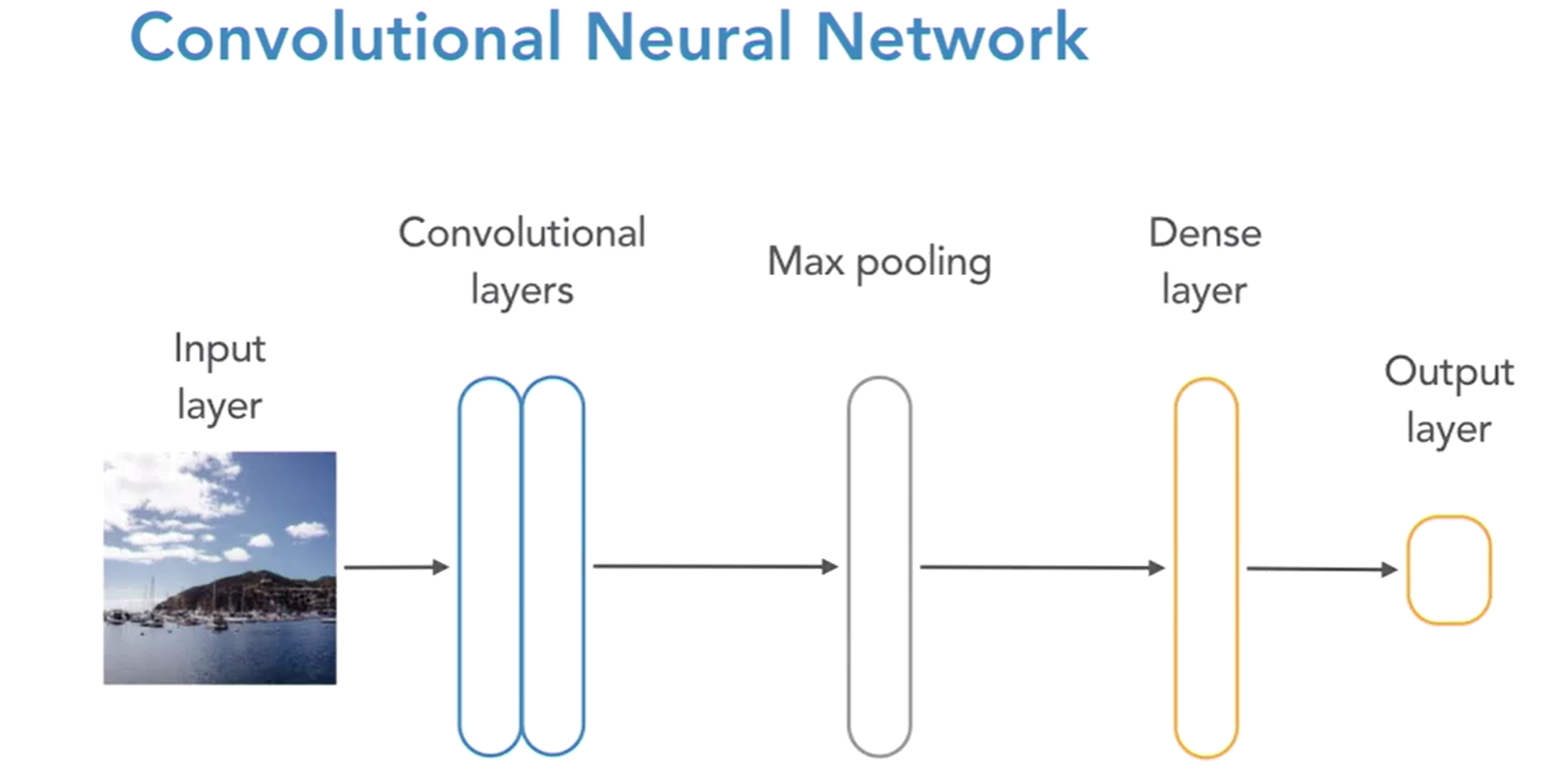
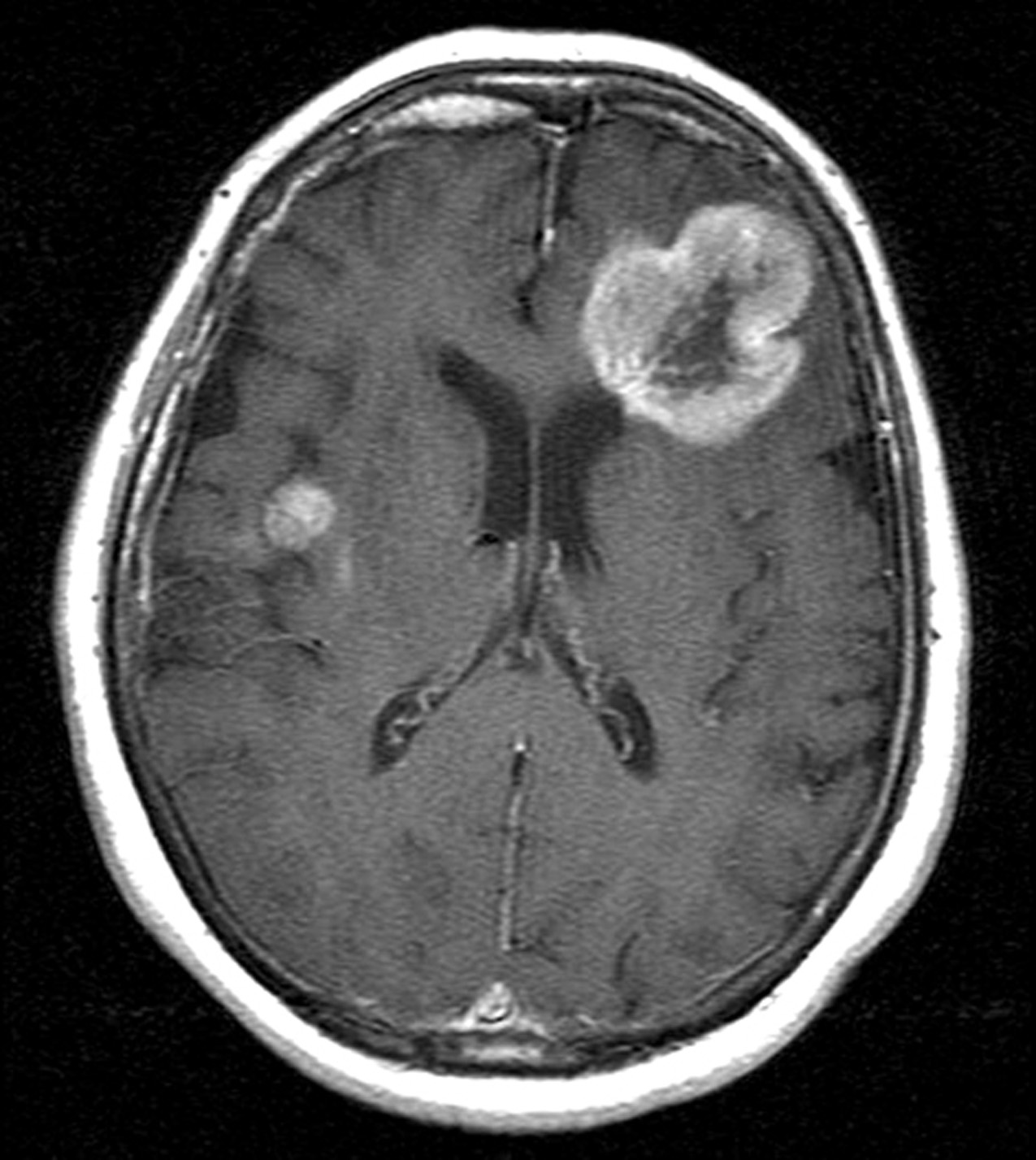


The above figure shows the results of the model. The total number of parameters are about 8.3 million. The batch size is 100 that means each time 100 images are sent through the node lines which are 512. The output shows 1 which means tumour is present.

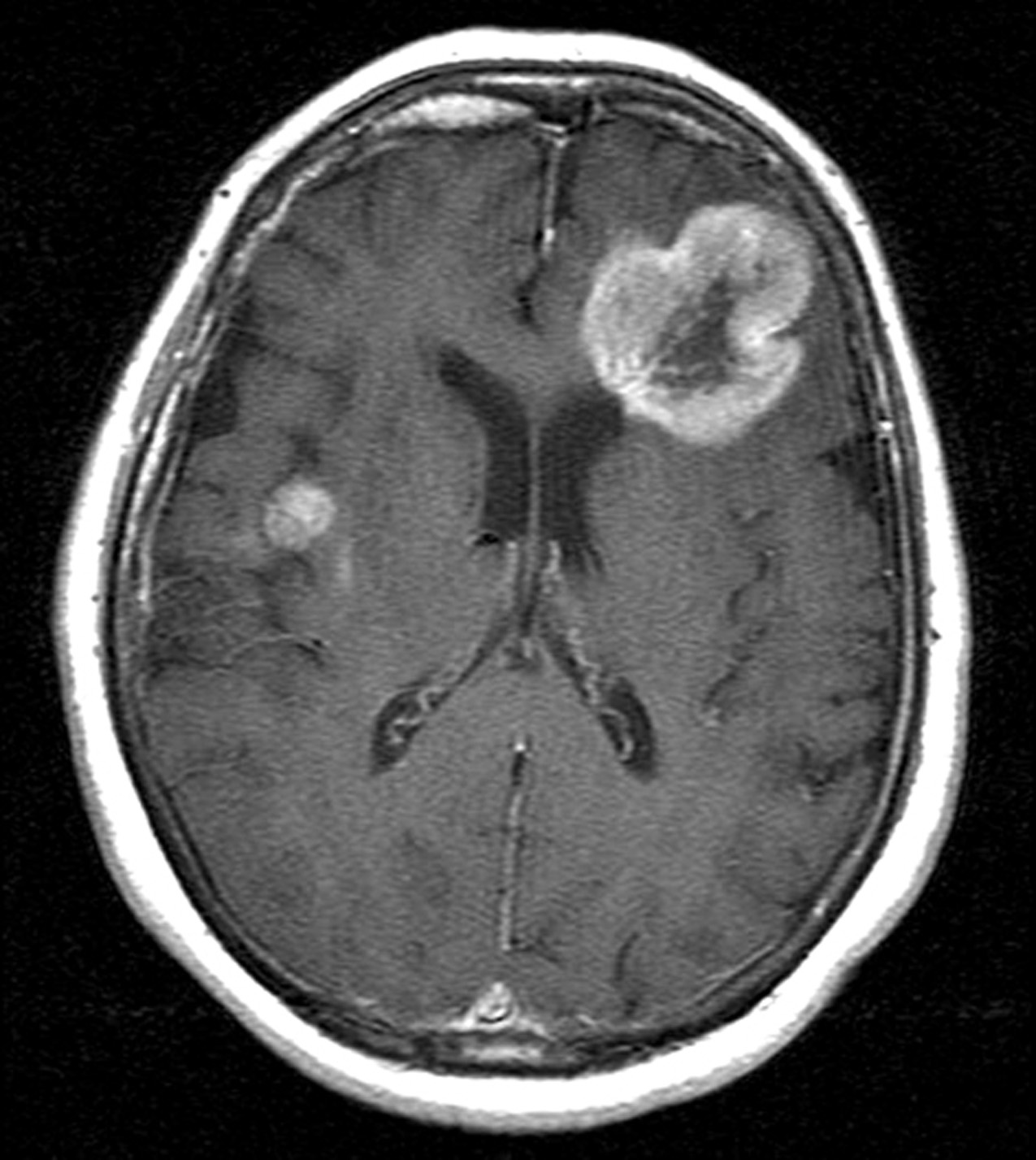


Here, we can see the accuracy of 59%. Although this is better than the previous result, we still need more accuracy and in the next step, we will try to achieve that.

**STEP 3: Max pooling layer**

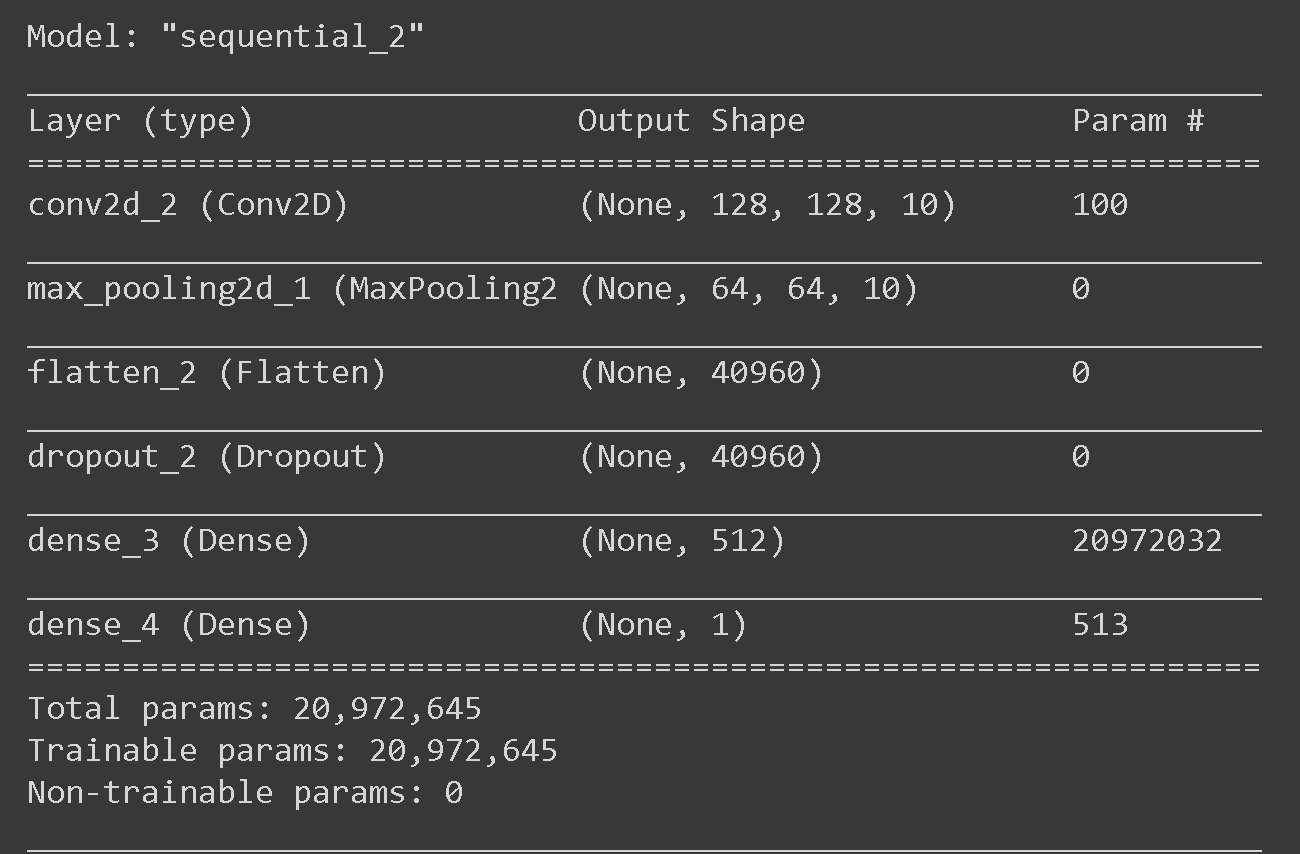


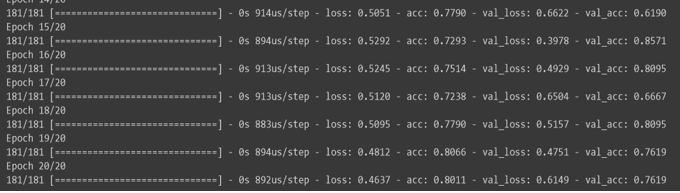
The above figure shows the layout of the model. We can see max pooling layer is being added into the model after convolutional layer and before dense layer.

In max pooling layer, when the image is being sent into the layers, it will be divide into number of smaller blocks and the values are assigned accoring tot the pixel density and change. Then the blocks are divided into 2x2 matrix as shown in the below picture and the maximum value from the divided blocks will be taken into the consideration and will be sent to the next layer for tumour detection. The below image shows how it is being done with maximum numbers are being highlighted and passed to another layer.

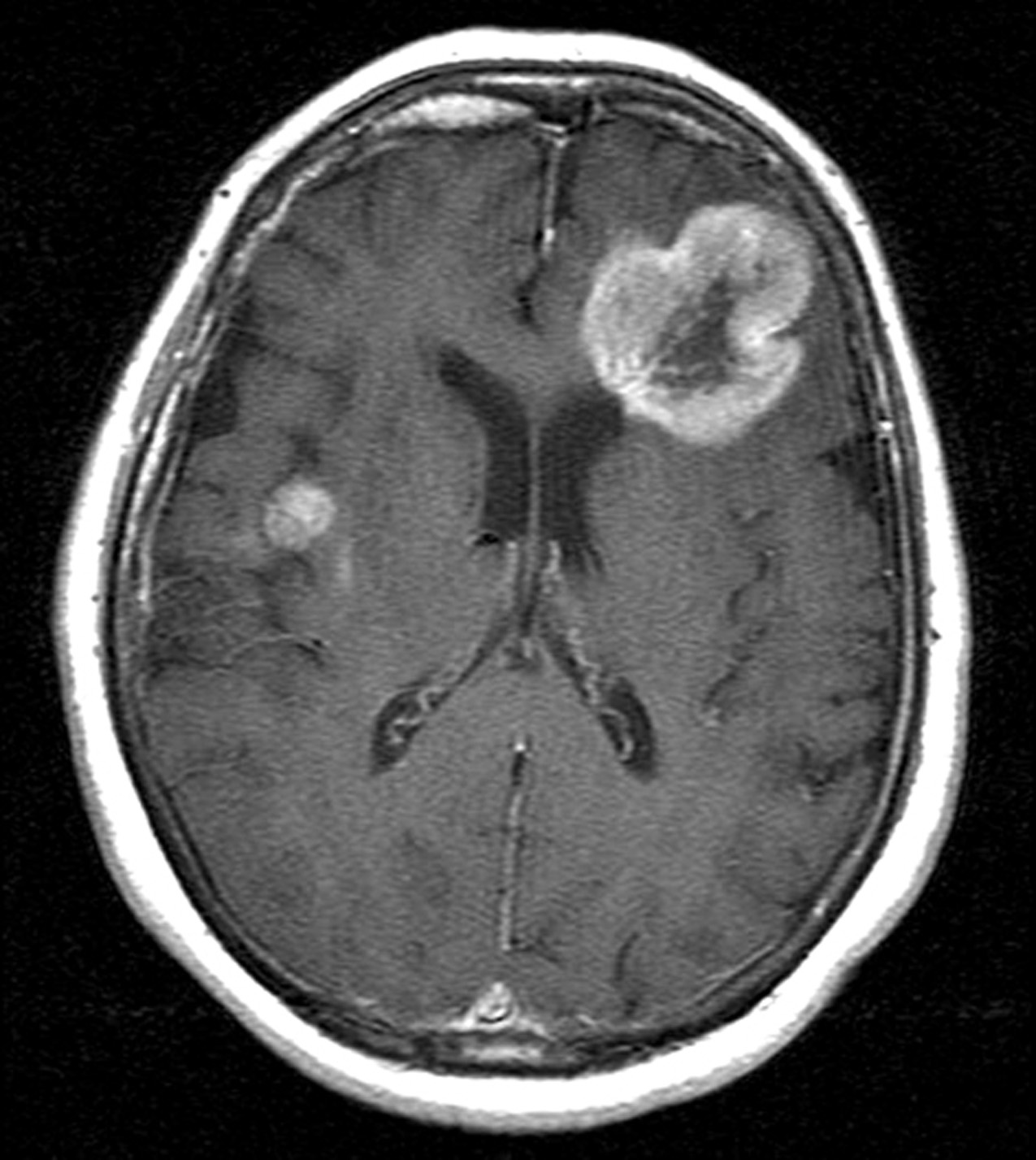
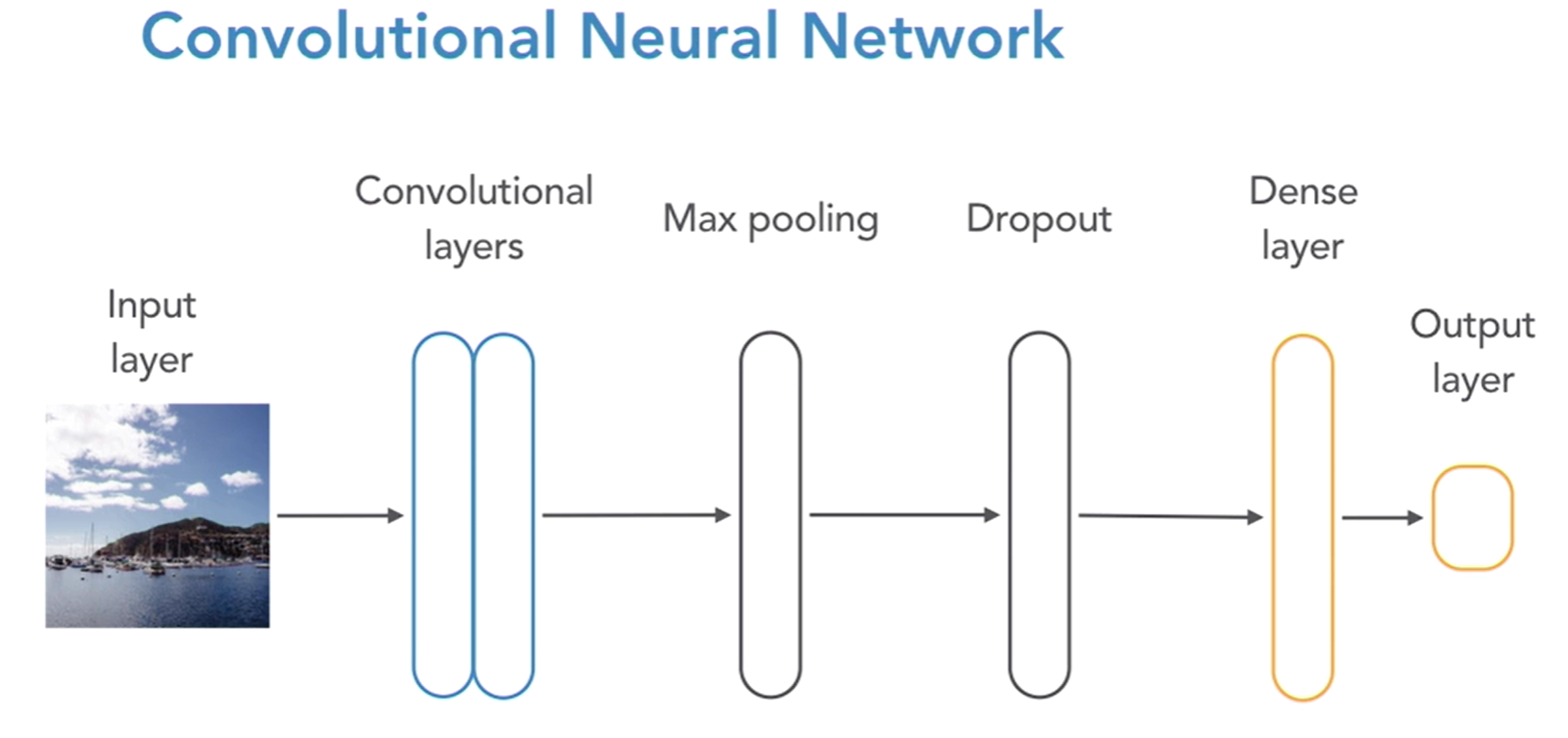


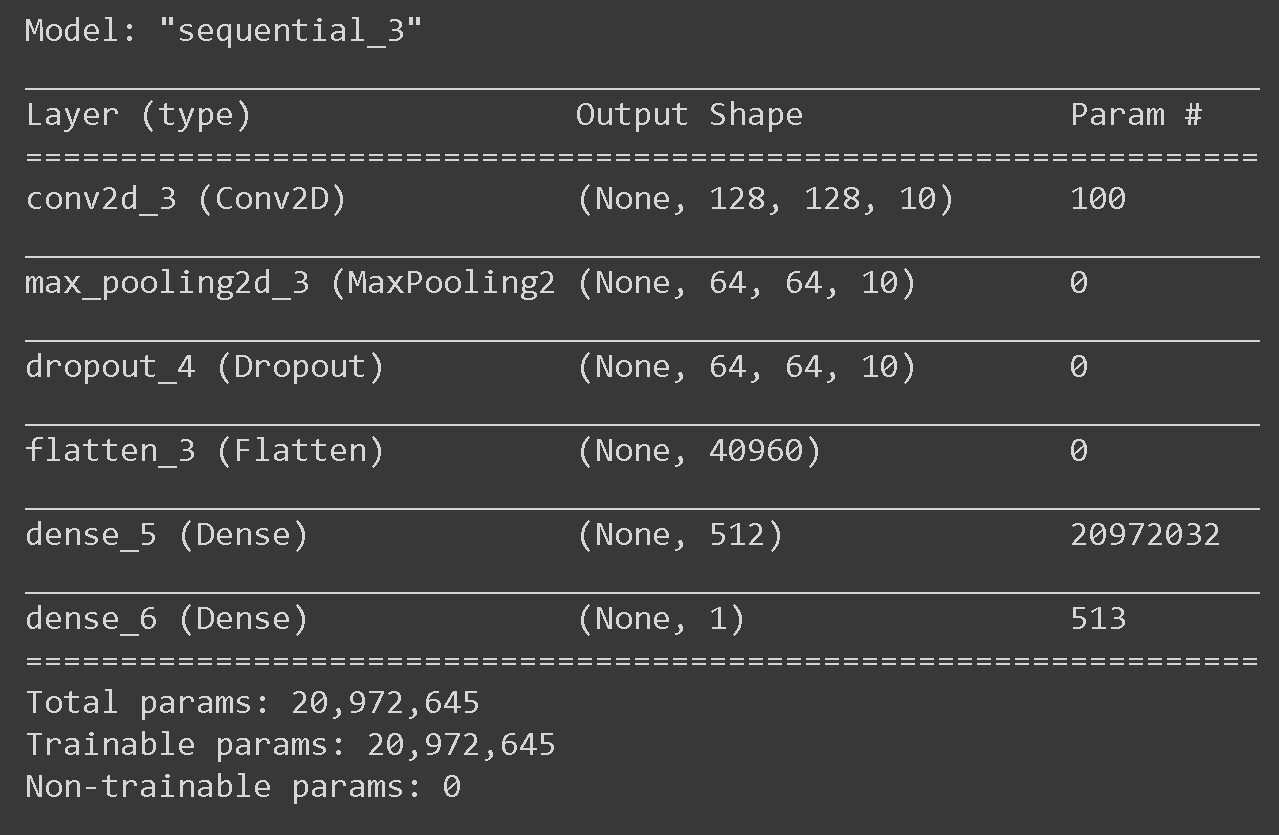




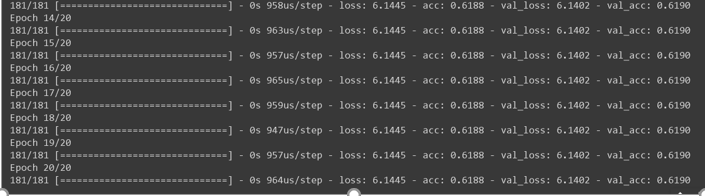
The above image shows the results of the model. We can observe that the accuracy of the model has been increase to 80% which is far better than the previous model. In the next model, we will add the Drop out layer and see how it will impact our results .

**STEP 4: Dropout layer**

**** ****

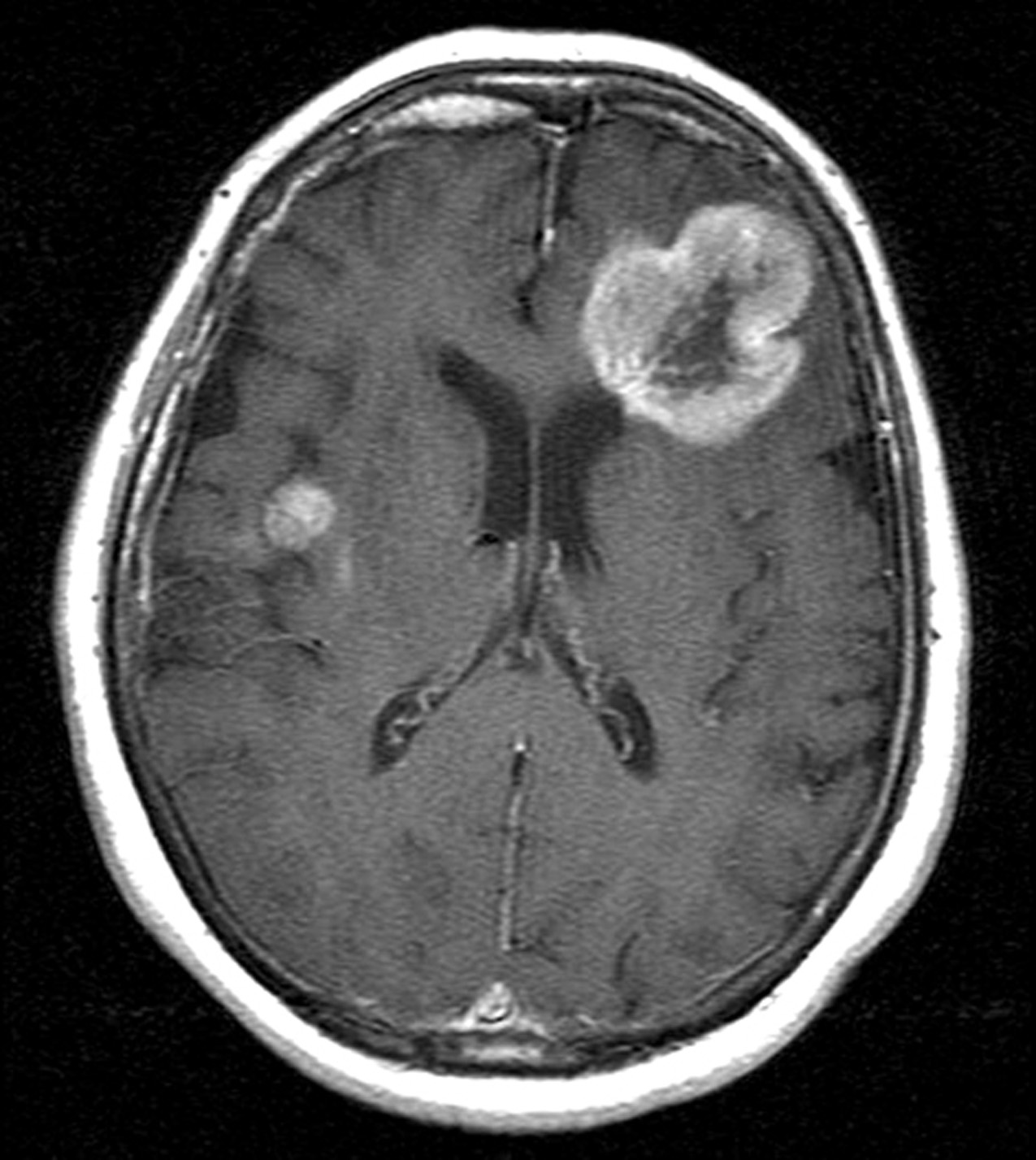
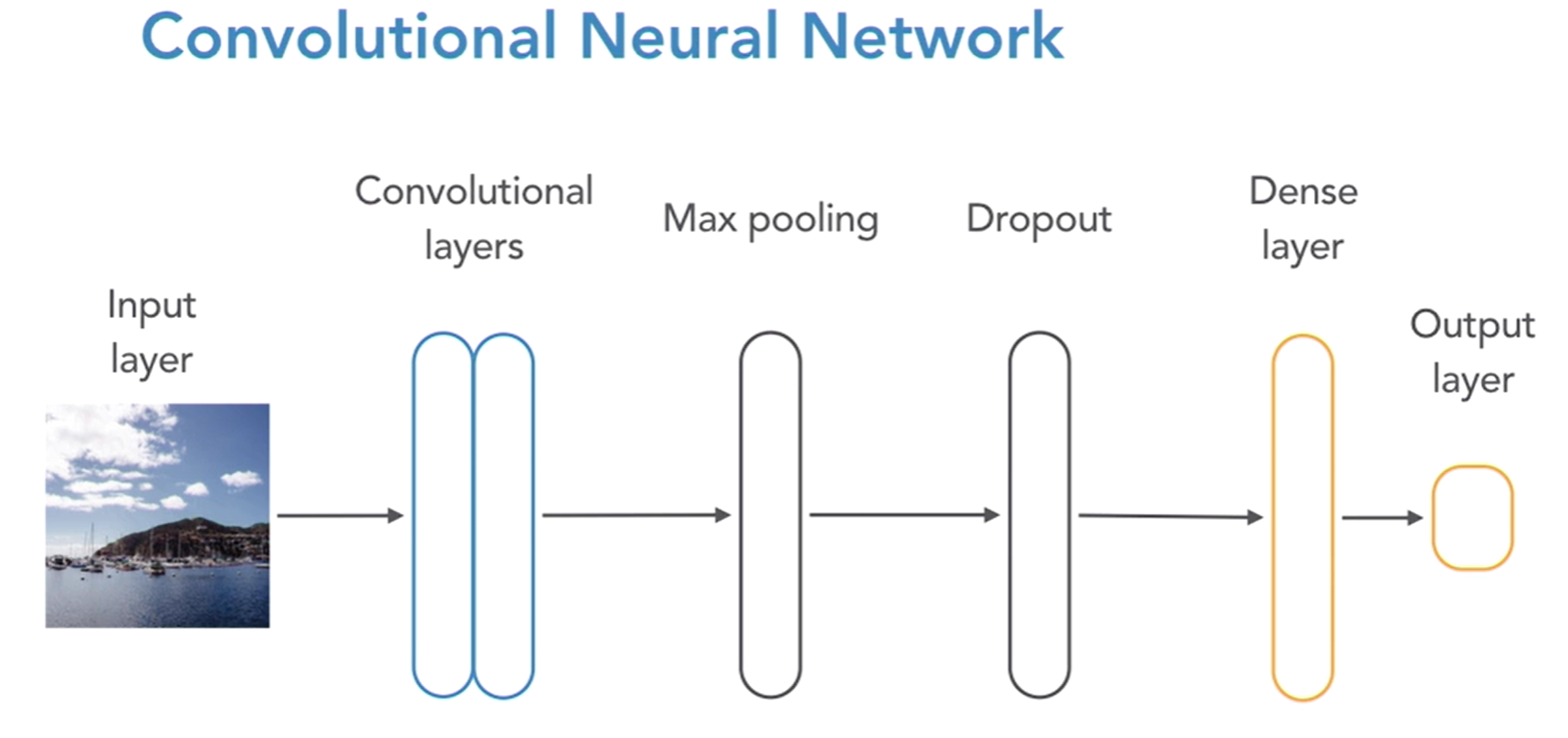
****

We add the Dropout layer to our model for better training. By adding the layer, the model will randomly select some of the nodes and drops the nodes going into the next layer. This way we can train the model better and efficient.

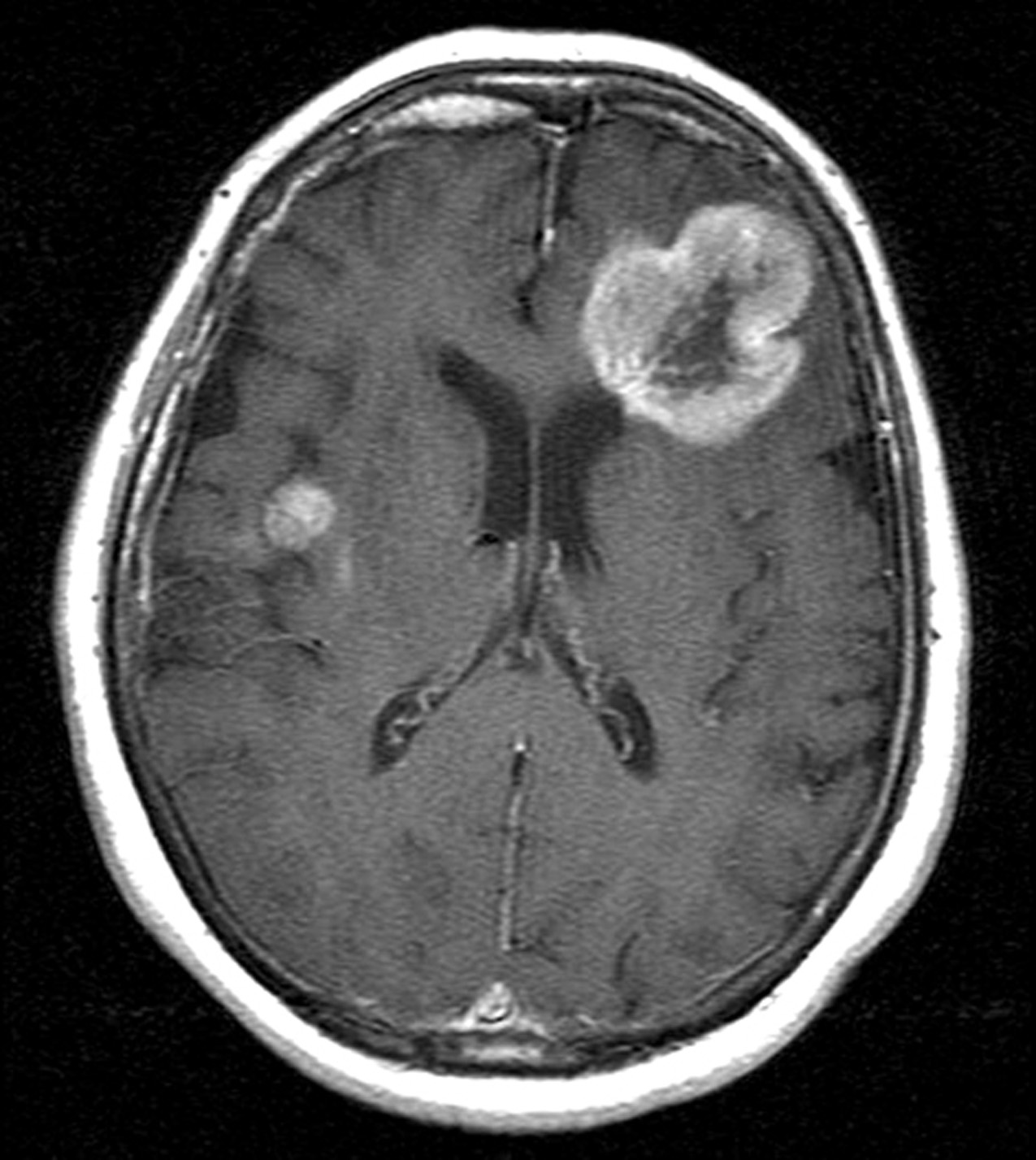
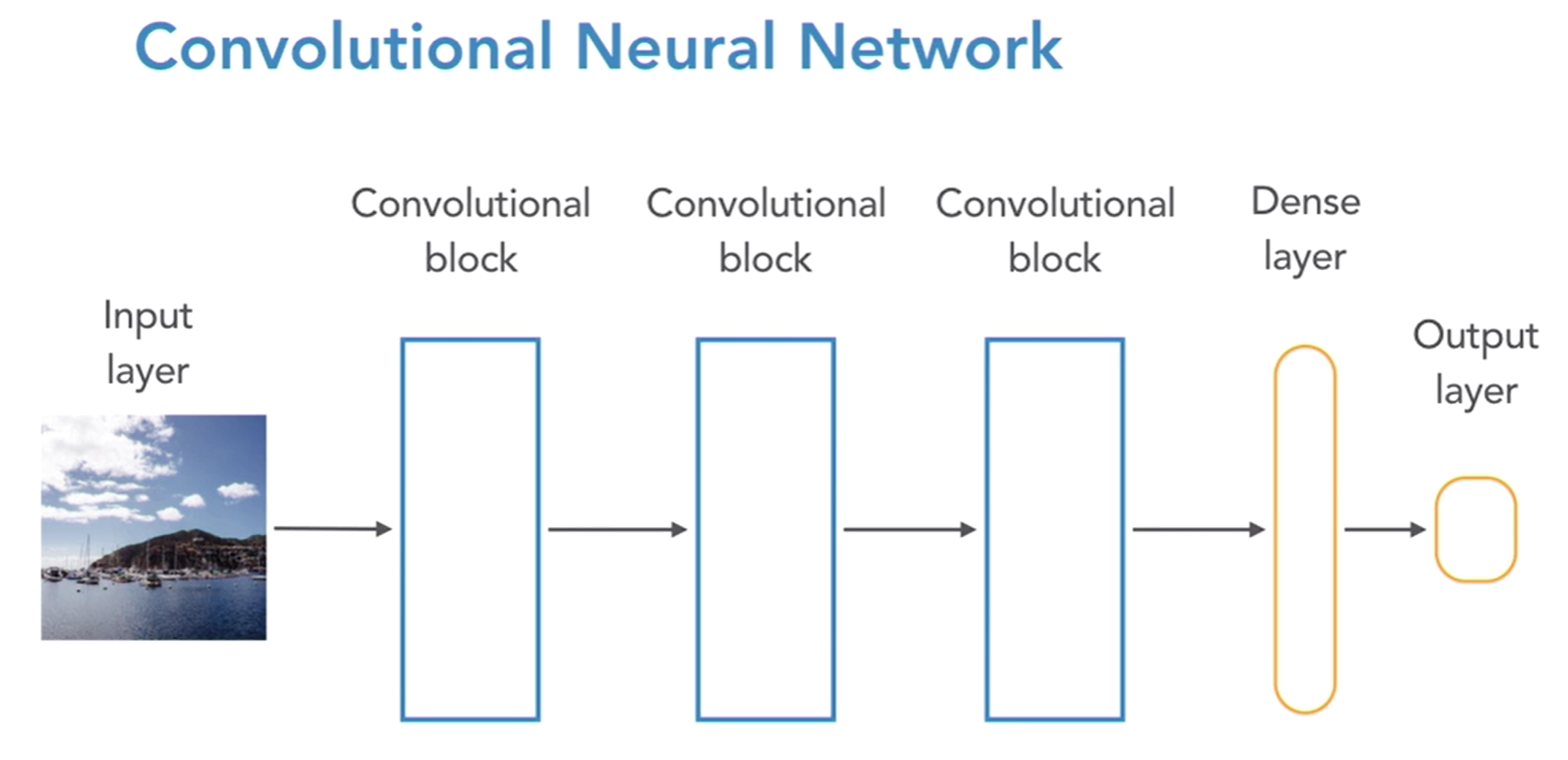


We got the accuracy of 61% which is very low compared to other models. IN the next model we will try to rectify this.

**STEP 5: Complete Neural Network**

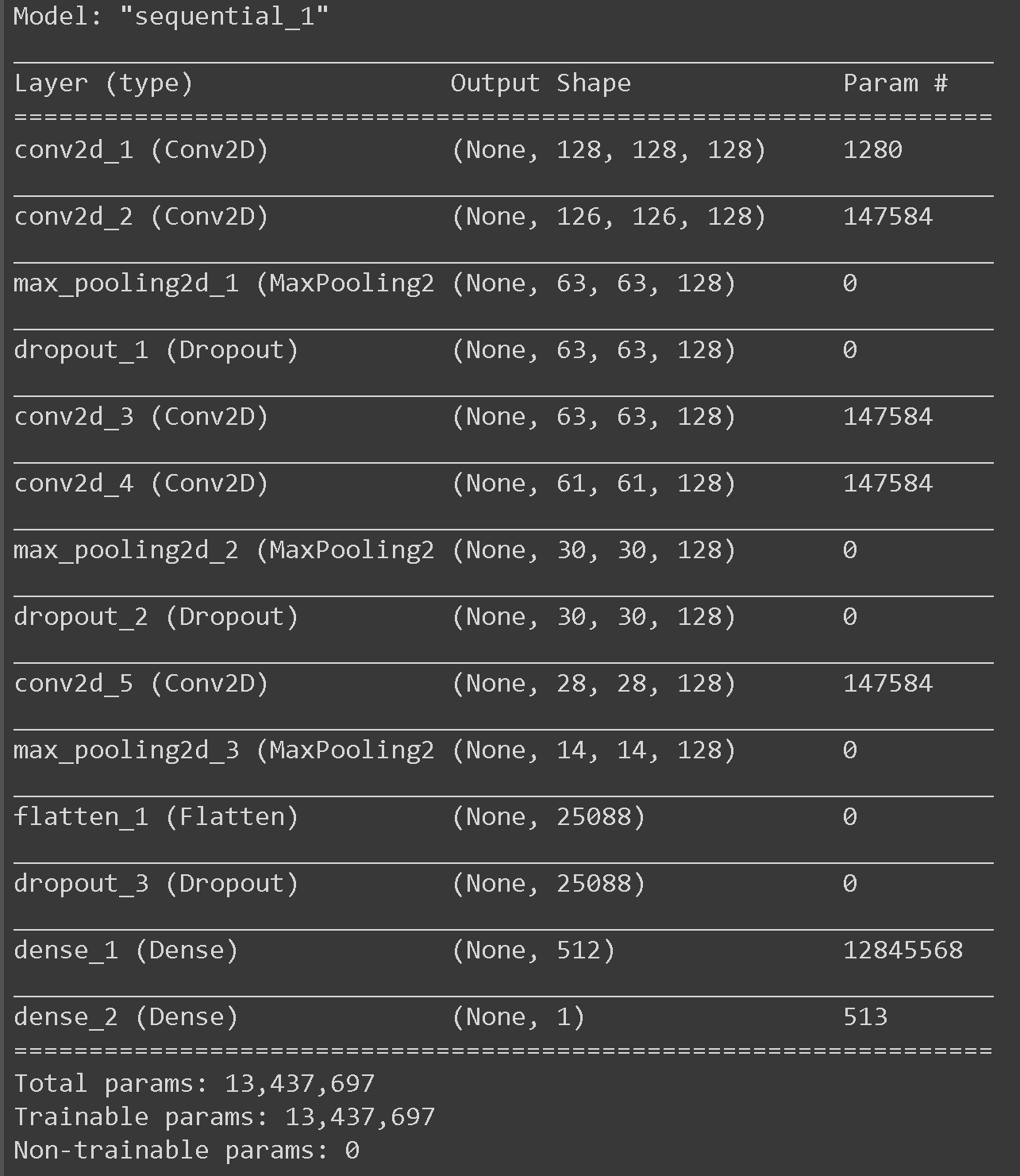
**** ****

The above figure shows the full flow of the model and how many alyers are present in the deep learning model.

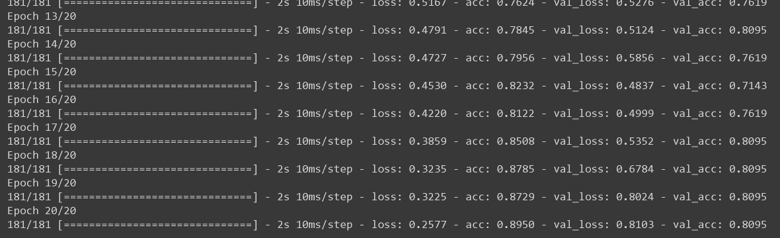
**** ****

This above figure represents the way convolutional blocks are placed one after the other and the ouput id given to the dense layer and finally output. Based on the question we need to answer, the number of blocks will be increased which reults in more accuracy.

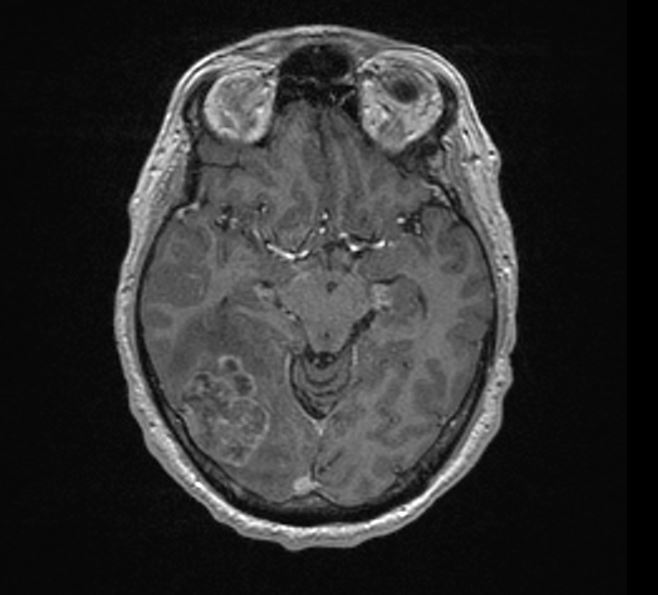
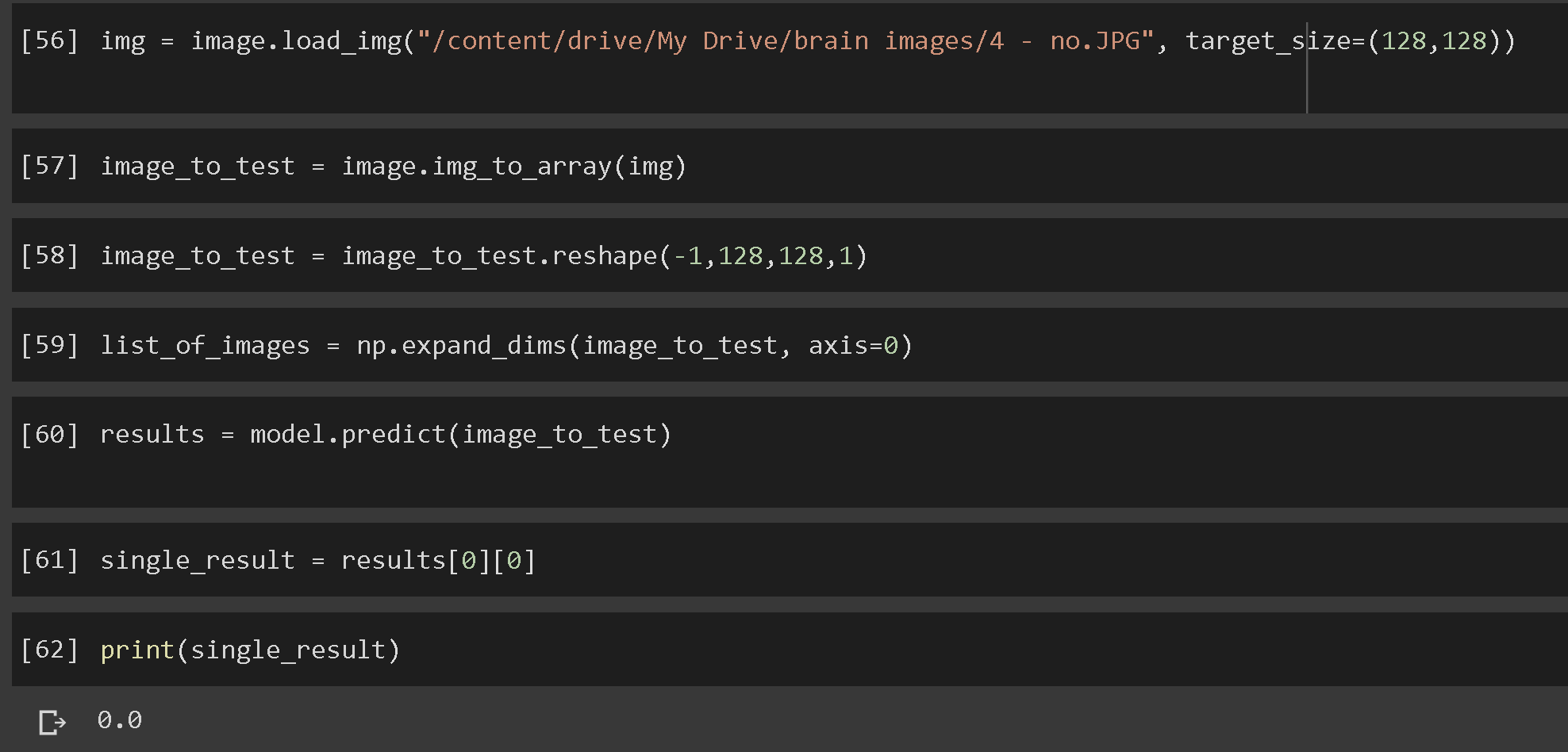
Each convolutional block has three layers convolutional , max pooling and dropout layer. We might have got less accuracy in the above model when we used dropout layer but when we use the dropout layer inside a convolutional block and having many blocks, we might train the model better and have better accuracy compared to other models.

****

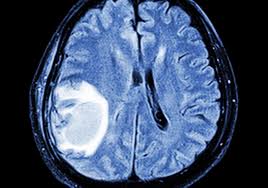
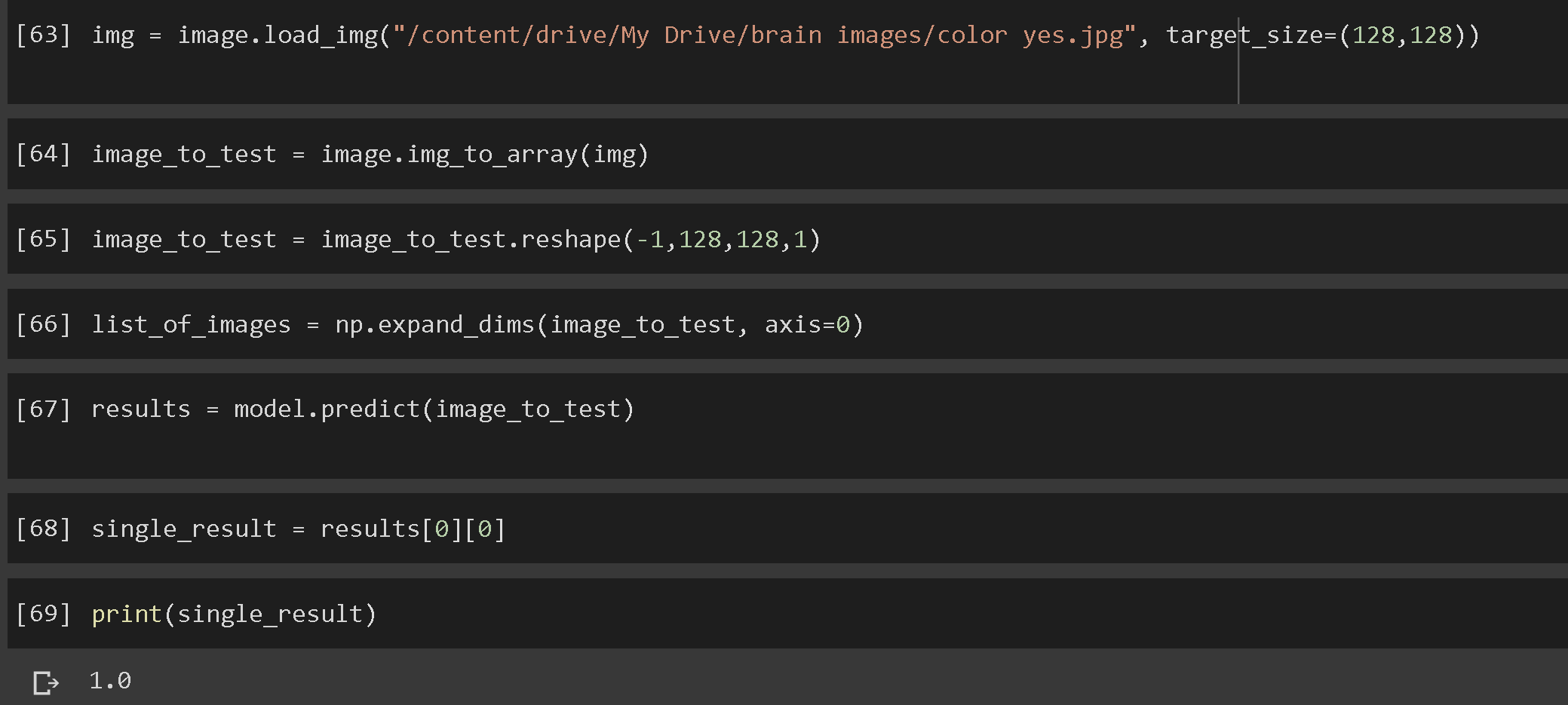
The below picture shows the output of our model. We got the accuracy of 89% which is very good for the model. We can see from previous models, as each layer is being applied the accuracy has been increased.



**Prediction**

Here, we downloaded a random MRI scanned brain image that dosent have tumour and applied that to our Deep learning model and the results shows 0 which interpretates into there is no brain tumour which is true.

In the same way as above, we here downlaoded image which has tumour and sent it through our deep learning model. We got the output as 1 wghich means that the tumour is present. Our model showed accurate results in both the cases.

**Conclusion**

This model of detecting Brain Tumour was developed to assist the pathology lab and not as ultimate judgement on whether the patient has tumour or not. By using Keras in Python we have developed a deep learning model consisting of multiple layers which are improving our accuracy of the model.

In this model we have achieved an overall accuracy of 89%, which is marginally large as it is being used in the medical industry to detect tumour. However, our focus was to reduce the False Negative error rate instead of False Positive.

In False Positive, a healthy patient might get true results for brain tumour present, which can again be rechecked. However, in false Negative, if a person having brain tumour is tested negative it can be very dangerous. Hence our model is more likely to detect a brain tumour present at a higher accuracy.

Code:

from google.colab import drive

drive.mount('/content/drive',force\_remount=True)

import numpy as np

import pandas as pd

from PIL import Image #to open image

import os

from os import listdir

from pathlib import Path

import cv2

import random

from sklearn.model\_selection import train\_test\_split

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D

pd.options.mode.chained\_assignment = None

demo\_neg =Image.open('/content/drive/My Drive/BrainTumor/no/38 no.jpg').resize((128,128))

demo\_neg

demo\_pos =Image.open('/content/drive/My Drive/BrainTumor/yes/Y10.jpg').resize((128,128))

demo\_pos

directory = "/content/drive/My Drive/BrainTumor"

type = ['yes', 'no']

img\_data=[]

data\_label =[]

data\_set =[]

for image\_type in type:

path = os.path.join(directory, image\_type)

img\_type\_index = type.index(image\_type)

for image in os.listdir(path):

img\_array = cv2.imread(os.path.join(path, image), cv2.IMREAD\_GRAYSCALE)

img\_array = img\_array.astype(np.float32)

img\_array = cv2.resize(img\_array, (128, 128))

img\_data.append([img\_array, img\_type\_index])

random.shuffle(img\_data)

for array\_img, data\_type in img\_data:

data\_set.append(array\_img)

data\_label.append(data\_type)

len(data\_label)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(data\_set, data\_label, test\_size = 0.2,

random\_state = 45)

x\_train = np.array(x\_train).reshape(-1,128,128, 1)

x\_train = x\_train/255.0

x\_test = np.array(x\_test).reshape(-1,128,128, 1)

y\_train = np.array(y\_train)

model = Sequential()

model.add(Conv2D( 128,(3,3), padding='same', input\_shape = x\_train.shape[1:], activation = "relu"))

model.add(Conv2D(128, (3,3), activation = "relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), padding='same', activation="relu"))

model.add(Conv2D(128, (3, 3), activation="relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), padding='same', activation="relu"))

model.add(Dense(512))

model.add(Dense(1,activation='sigmoid'))

model.add(Conv2D(128, (3,3), activation="relu"))

model.add(MaxPooling2D(pool\_size = (2,2)))

model = Sequential()

model.add(Conv2D( 128,(3,3), padding='same', input\_shape = x\_train.shape[1:], activation = "relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dropout(0.25))

model.add(Dense(512))

model.add(Dense(1,activation='sigmoid'))

model.summary()

# Compile the model

model.compile(

loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy']

)

#Train The model

x = model.fit(

x\_train,

y\_train,

batch\_size=50,

epochs=20,

validation\_split = 0.1

)

# Save neural network structure

model\_structure = model.to\_json()

f = Path('/content/drive/My Drive/model/model \_structure.json')

f.write\_text(model\_structure)

# Save neural network's trained weights

model.save\_weights("/content/drive/My Drive/model/model\_weights.h5")

from IPython.display import SVG

from IPython.core.pylabtools import figsize

figsize = (1,1)

import pydot

from keras.utils.vis\_utils import model\_to\_dot

SVG(model\_to\_dot(model,show\_shapes = True,dpi=60).create(prog='dot', format='svg'))

#### Tumour prediction

from google.colab import drive

drive.mount('/content/drive',force\_remount=True)

from keras.models import model\_from\_json

from pathlib import Path

from keras.preprocessing import image

import numpy as np

import cv2

f = Path("/content/drive/My Drive/model/model \_structure.json")

model\_structure = f.read\_text()

model = model\_from\_json(model\_structure)

model.load\_weights("/content/drive/My Drive/model/model\_weights.h5")

img = image.load\_img("/content/drive/My Drive/brain images/download.jpg", target\_size=(128,128))

image\_to\_test = image.img\_to\_array(img)

image\_to\_test = image\_to\_test.reshape(-1,128,128,1)

list\_of\_images = np.expand\_dims(image\_to\_test, axis=0)

results = model.predict(image\_to\_test)

single\_result = results[0][0]

print(single\_result)