# B.M.S College of Engineering



**P.O. Box No.: 1908 Bull Temple Road, Bangalore-560 019**

### DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

**Course - Deep Learning**

**Code - 20IS6SPEDLG**

**AY 2020-21**

**REPORT ON**

**Brain Tumor Classification using CNNs**

Submitted by-

**Prateek Jain - 1BM18IS149**

**Akash Mishra - 1BM18IS006**

**Abhijit Kumar – 1BM18IS002**

Submitted To - **Rashmi R**

# B.M.S College of Engineering

**P.O. Box No.: 1908 Bull Temple Road, Bangalore-560 019**

### DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **CONTENTS** | **Page No.** |
| **Introduction** | 3 |
| **Data Collection/Analysis** | 4 |
| **Data Pre-processing** | 5 |
| **API’s used with flow diagram of model** | 8 7 |
| **Implementation with code** | 10 |
| **Performance Measures with code** | 14 |
| **Results** | 15 |
| **Conclusion** | 17 |
| **References** | 18 |

INTRODUCTION

Deep learning using the convolutional neural network architecture is a powerful tool for classification of images and videos. In this project we have tried to utilize this powerful architecture in the field of medical imaging.

Often in medical imaging there is a manual task of image captioning, where the subject matter expert will observe the image and given his expertise in the form a caption, this captioning is often useful for classification, identification or diagnostic purposes.

This data when collected over a large period time can be used as material for training deep learning models for specific diagnostic purposes. With the advent of large scale production grade datasets like ImageNet and also with the emergence of transfer learning, they can be used to train powerful models in the field of computer vision and solve problems like classification.

We have tried to develop a model using medical imaging data and deliver powerful insights and inferences based on the data. In our project we have demonstrated a specific use case of the powerful CNN architecture for diagnostic purposes on the brain-tumor dataset obtained from Kaggle.

More about the dataset and the characteristics along with its features will be discussed in subsequent sections of this report.

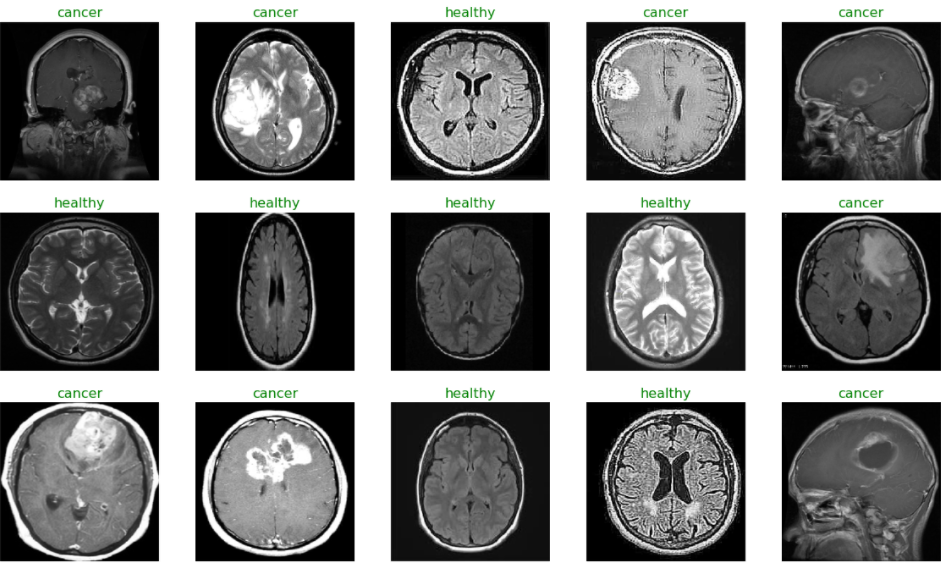
### 2. Data Collection and Analysis

The brain tumour dataset was obtained from Kaggle. Kaggle is a widely renowned as a data science forum with several popular datasets. The brain tumour dataset contains X-ray images of brain along with the caption/diagnosis.

The dataset contains about 139 images, since the number of images is small we have used image augmentation to increase the number of images in the dataset.

The dataset contains a .csv file with the names of all the image files(jpg/jpeg) in the dataset containing actual x-ray images separated into two folders depending on the diagnosis. Using the .csv file we can obtain the absolute path of all images in the dataset by concatenating the base path with the filenames in the .csv file, the .csv file also contains the caption/diagnosis for all the images in the dataset.

The dataset contains 139 image files containing the x-ray images. The .csv file 12 columns which are the filename, diagnosis/class, colour-mode ( RGB or L ), file format(jpg/jpeg/tiff), shape of the image being the prominent features of the dataset which is important data for pre-processing of the dataset, so that we can reduce all the images into the same shape, colour mode and same number of channels before they are forwarded to the CNN model.



3. Data Pre-Processing

## Since our dataset has only 139 files we can proceed with transfer learning, but we can increase the number of images we have in the dataset by doing data augmentation. A part from data augmentation we have to pre-process the dataset as we donot have the required format of data. In the dataset we have to convert the images files into 2D vectors and since the files are in different folders we have to create a dataframe and shuffle them so that the model does not suffer from any unnatural pattern in the data created by us.

## Data Augmentation:

## When working with deep learning models, It is a peculiar situation when there is not much data to train the model. We used the Image Generator class in Keras which can be used to augment images in real time when the model is training. The image augmentation technique is a great way to expand the size of your dataset. You can come up with new transformed images from your original dataset. Keras Image Generator class can apply any random transformations on each training image as it is passed to the model. This will not only make the model more robust but will also save up on the overhead memory. These image augmentation techniques not only expand the size of your dataset but also incorporate a level of variation in the dataset which allows your model to generalize better on unseen data. Also, the model becomes more robust when it is trained on new, slightly altered images. Keras ImageDataGenerator class provides a quick and easy way to augment your images. It provides a host of different augmentation techniques like standardization, rotation, shifts, flips, brightness change, and many more.

## 

## Apart from data-augmentation we had to import the dataset into a dataframe and shuffle the dataset as all the instances of a classes are together and also obtain their respective labels by creating pandas series to hold these labels and shuffle them in the same way the dataframe was shuffled.

## 3.1 Data Pre-processing code

tumor\_dir=r'../input/brian-tumor-dataset/Brain Tumor Data Set/Brain Tumor Data Set/Brain Tumor'

healthy\_dir=r'../input/brian-tumor-dataset/Brain Tumor Data Set/Brain Tumor Data Set/Healthy'

filepaths = []

labels= []

dict\_list = [tumor\_dir, healthy\_dir]

for i, j **in** enumerate(dict\_list):

flist=os.listdir(j)

for f **in** flist:

fpath=os.path.join(j,f)

filepaths.append(fpath)

if i==0:

labels.append('cancer')

else:

labels.append('healthy')

Fseries = pd.Series(filepaths, name="filepaths")

Lseries = pd.Series(labels, name="labels")

tumor\_data = pd.concat([Fseries,Lseries], axis=1)

tumor\_df = pd.DataFrame(tumor\_data)

print(tumor\_df.head())

print(tumor\_df["labels"].value\_counts())

*#splitting data*

train\_images, test\_images = train\_test\_split(tumor\_df, test\_size=0.3, random\_state=42)

train\_set, val\_set = train\_test\_split(tumor\_df, test\_size=0.2, random\_state=42)

*#Generate batches of tensor image data with real-time data augmentation.*

image\_gen = ImageDataGenerator(preprocessing\_function= tf.keras.applications.mobilenet\_v2.preprocess\_input)

train = image\_gen.flow\_from\_dataframe(dataframe= train\_set,x\_col="filepaths",y\_col="labels",

target\_size=(244,244),

color\_mode='rgb',

class\_mode="categorical", *#used for Sequential Model*

batch\_size=32,

shuffle=False *#do not shuffle data*

)

test = image\_gen.flow\_from\_dataframe(dataframe= test\_images,x\_col="filepaths", y\_col="labels",

target\_size=(244,244),

color\_mode='rgb',

class\_mode="categorical",

batch\_size=32,

shuffle= False

)

val = image\_gen.flow\_from\_dataframe(dataframe= val\_set,x\_col="filepaths", y\_col="labels",

target\_size=(244,244),

color\_mode= 'rgb',

class\_mode="categorical",

batch\_size=32,

shuffle=False

)

## ALGORITHM

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms.

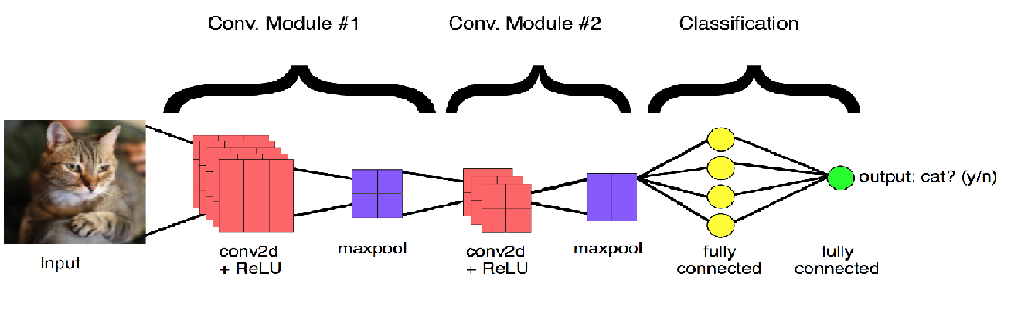
While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.



For this problem we use the CNN architecture as it is best suited for computer vision problems and since our task is classification based on X-ray images we can either train a CNN model from scratch or implement transfer learning. In our implementation we have built a model from scratch. The Flow diagram of the model created is shown below.

**Flow Diagram:**



We have used the sequential API offered by Keras to build the CNN architecture. The image is first fed into a convolution layer with 32 filters and the activation function used is relu and the output from this layer is passed to a max pooling layer as a result of this the spatial dimension of the image is reduced by half, after this the batch normalization is applied, we repeat this step again and finally we flatten the inputs by using a flatten layer and proceed by adding dense layers, since the model can be prone to over fitting we add a dropout layer to reduce over fitting. Since we are doing binary classification the output of our last layer is passed through a sigmoid activation function.

The model summary is shown below:

Model: "sequential"

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

Layer (type) Output Shape Param #

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

conv2d (Conv2D) (None, 128, 128, 32) 416

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

conv2d\_1 (Conv2D) (None, 128, 128, 32) 4128

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

batch\_normalization (BatchNo (None, 128, 128, 32) 128

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

max\_pooling2d (MaxPooling2D) (None, 64, 64, 32) 0

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

dropout (Dropout) (None, 64, 64, 32) 0

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

conv2d\_2 (Conv2D) (None, 64, 64, 64) 8256

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

conv2d\_3 (Conv2D) (None, 64, 64, 64) 16448

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

batch\_normalization\_1 (Batch (None, 64, 64, 64) 256

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

max\_pooling2d\_1 (MaxPooling2 (None, 32, 32, 64) 0

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

dropout\_1 (Dropout) (None, 32, 32, 64) 0

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

flatten (Flatten) (None, 65536) 0

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

dense (Dense) (None, 512) 33554944

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

dropout\_2 (Dropout) (None, 512) 0

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

dense\_1 (Dense) (None, 2) 1026

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

Total params: 33,585,602

Trainable params: 33,585,410

Non-trainable params: 192

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

None

Implementation with code

# Import Statements

In [25]:

import os

import keras

from keras.models import Sequential

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

from PIL import Image

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

plt.style.use('dark\_background')

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

from sklearn.preprocessing import OneHotEncoder

# One Hot Encoding the Target Classes

In [26]:

encoder = OneHotEncoder()

encoder.fit([[0], [1]])

*# 0 - Tumor*

*# 1 - Normal*

Out[26]:

OneHotEncoder()

# Creating 3 Important Lists --

1. data list for storing image data in numpy array form
2. paths list for storing paths of all images
3. result list for storing one hot encoded form of target class whether normal or tumor

In [27]:

*# This cell updates result list for images with tumor*

data = []

paths = []

result = []

for r, d, f **in** os.walk(r'../input/brain-mri-images-for-brain-tumor-detection/yes'):

for file **in** f:

if '.jpg' **in** file:

paths.append(os.path.join(r, file))

for path **in** paths:

img = Image.open(path)

img = img.resize((128,128))

img = np.array(img)

if(img.shape == (128,128,3)):

data.append(np.array(img))

result.append(encoder.transform([[0]]).toarray())

In [28]:

*# This cell updates result list for images without tumor*

paths = []

for r, d, f **in** os.walk(r"../input/brain-mri-images-for-brain-tumor-detection/no"):

for file **in** f:

if '.jpg' **in** file:

paths.append(os.path.join(r, file))

for path **in** paths:

img = Image.open(path)

img = img.resize((128,128))

img = np.array(img)

if(img.shape == (128,128,3)):

data.append(np.array(img))

result.append(encoder.transform([[1]]).toarray())

In [29]:

data = np.array(data)

data.shape

Out[29]:

(139, 128, 128, 3)

In [30]:

result = np.array(result)

result = result.reshape(139,2)

# Splitting the Data into Training & Testing

In [31]:

x\_train,x\_test,y\_train,y\_test = train\_test\_split(data, result, test\_size=0.2, shuffle=True, random\_state=0)

# Model Building

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

In [32]:

model = Sequential()

model.add(Conv2D(32, kernel\_size=(2, 2), input\_shape=(128, 128, 3), padding = 'Same'))

model.add(Conv2D(32, kernel\_size=(2, 2), activation ='relu', padding = 'Same'))

model.add(BatchNormalization())

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

model.add(Dropout(0.25))

model.add(Conv2D(64, kernel\_size = (2,2), activation ='relu', padding = 'Same'))

model.add(Conv2D(64, kernel\_size = (2,2), activation ='relu', padding = 'Same'))

model.add(BatchNormalization())

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

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))

model.compile(loss = "categorical\_crossentropy", optimizer='Adamax')

print(model.summary())

Model: "sequential\_2"

Total params: 33,585,602

Trainable params: 33,585,410

Non-trainable params\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [33]:

y\_train.shape

Out[33]:

(111, 2)

In [34]:

history = model.fit(x\_train, y\_train, epochs = 30, batch\_size = 40, verbose = 1,validation\_data = (x\_test, y\_test))

Epoch 1/30

3/3 [==============================] - 1s 244ms/step - loss: 70.0754 -:

# Plotting Losses

In [35]:

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

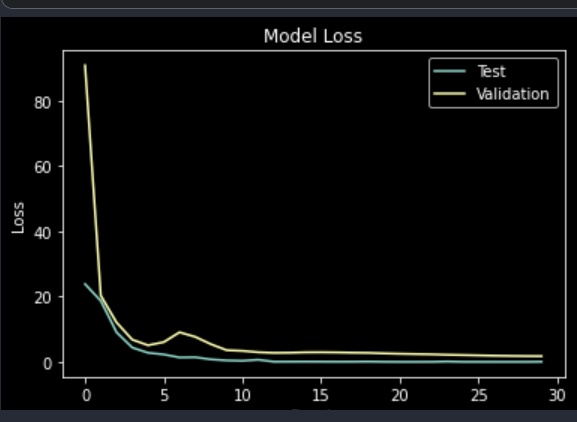
plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Test', 'Validation'], loc='upper right')

plt.show()



# Just Checking the Model

In [36]:

def names(number):

if number==0:

return 'Its a Tumor'

else:

return 'No, Its not a tumor'

In [37]:

from matplotlib.pyplot import imshow

img = Image.open(r"../input/brain-mri-images-for-brain-tumor-detection/no/N17.jpg")

x = np.array(img.resize((128,128)))

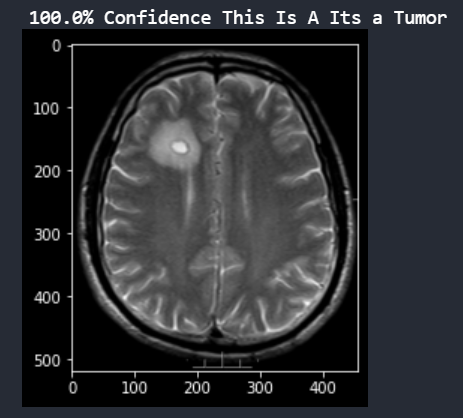
x = x.reshape(1,128,128,3)

res = model.predict\_on\_batch(x)

classification = np.where(res == np.amax(res))[1][0]

imshow(img)

print(str(res[0][classification]\*100) + '% Confidence This Is ' + names(classification))



99.99988079071045% Confidence This Is No, Its not a tumor

In [38]:

from matplotlib.pyplot import imshow

img = Image.open(r"../input/brain-mri-images-for-brain-tumor-detection/yes/Y3.jpg")

x = np.array(img.resize((128,128)))

x = x.reshape(1,128,128,3)

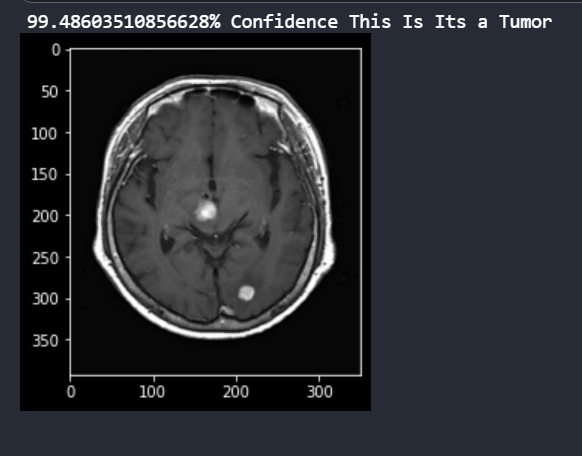
res = model.predict\_on\_batch(x)

classification = np.where(res == np.amax(res))[1][0]

imshow(img)

print(str(res[0][classification]\*100) + '% Confidence This Is A ' + names(classification))

90.67105054855347% Confidence This Is A Its a Tumor



RESULTS

Since our problem is a binary classification problem we have used metrics like confusion matrix and accuracy score. Along with the above mentioned metrics we have plotted graphs for the training and validation losses against the number epochs, similarly we have plotted graphs for training accuracy and validation accuracy against the number of epochs, using these metrics we can get a clear idea of how our model is performing and whether it is over fitting/under fitting.

The above plot shows the training and validation accuracy with the number of epochs. The code for the graph is shown below:

acc = History.history["accuracy"] *# report of model*

val\_acc = History.history["val\_accuracy"] *# history of validation data*

loss = History.history["loss"] *# Training loss*

val\_loss = History.history["val\_loss"] *# validation loss*

plt.figure(figsize=(8,8))

plt.subplot(2,1,1) *# 2 rows and 1 columns*

*#plotting respective accuracy*

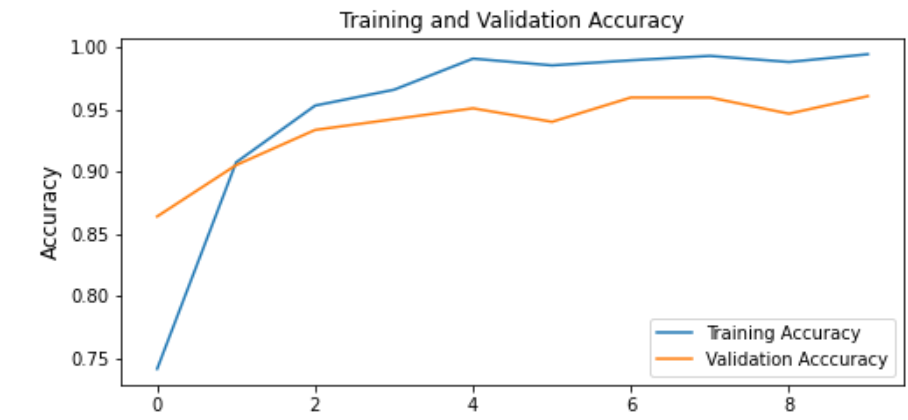
plt.plot(acc,label="Training Accuracy")

plt.plot(val\_acc, label="Validation Acccuracy")

plt.legend()

plt.ylabel("Accuracy", fontsize=12)

plt.title("Training and Validation Accuracy", fontsize=12)

****

plt.figure(figsize=(8,8))

plt.subplot(2,1,1)

plt.plot(loss, label="Training Loss") *#Training loss*

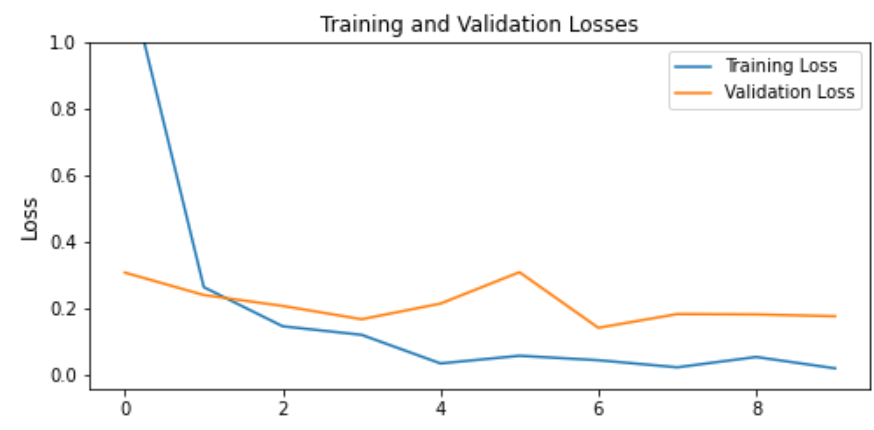
plt.plot(val\_loss, label="Validation Loss") *# Validation Loss*

plt.legend()

plt.ylim([min(plt.ylim()),1])

plt.ylabel("Loss", fontsize=12)

plt.title("Training and Validation Losses", fontsize=12)

****

*# Confusion Matrix*

**from** **sklearn.metrics** **import** confusion\_matrix, accuracy\_score

plt.figure(figsize = (10,5))

cm = confusion\_matrix(y\_test, pred2)

sns.heatmap(cm, annot=**True**, fmt = 'g'

## PERFORMANCE MEASURES WITH CODE

## 

## Based on the confusion matrix we calculate the precision, recall, f1-score and othere metrics which helps us to get a better understanding of the results.

precision recall f1-score support

cancer 0.96 0.99 0.98 779

healthy 0.99 0.95 0.97 601

accuracy 0.97 1380

macro avg 0.98 0.97 0.97 1380

weighted avg 0.97 0.97 0.97 1380

## CONCLUSIONS

## In this project we presented a very specific application of CNNs in the healthcare domain. We can leverage these architectures for automating several manual tasks in the healthcare domain and provide highly accurate diagnosis using these neural network architectures.

## Medical imaging plays an important role in diagnosis which requires a subject matter expert to manually caption each picture, this process can be automated using these CNN architectures to obtain high degree of confidence and accuracy in the task being performed.

## By using the ever-evolving field of ML/AI for these specific healthcare domains we can increase the accuracy of the diagnosis and mainly reduce the need for manual captioning of medical images.

## Our project can be generalized for any dataset for providing an accurate diagnosis and serving the community at large. In conclusion the field of computer vision can be used in many healthcare domains for a range of purposes like the one we presented in this document

## REFERENCES

* kaggle.com
* colab.google.com
* Medium.com
* Shahzadi, Iram, et al. "CNN-LSTM: Cascaded framework for brain Tumour classification." *2018 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*. IEEE, 2018.
* Martinez, Elizabeth, et al. "MRI Brain Tumour Segmentation using a CNN Over a Multi-parametric Feature Extraction." *2020 IEEE Colombian Conference on Applications of Computational Intelligence (IEEE ColCACI 2020)*. IEEE, 2020.