B. Tech CSE & ECE

19ECE357 – PATTERN RECOGNITION

TERM WORK

GROUP NO.: 12 (ECE - A)

GROUP MEMBERS:

1. CHADA HARSHINI REDDY – CB.EN.U4ECE19061
2. GAYATHREE ELANJELIAN – CB.EN.U4ECE19062

BRAIN TUMOR DETECTION USING DEEP LEARNING MODELS

ABSTRACT

A brain tumour is a disease caused due to the growth of abnormal cells in the brain. There are two main categories of brain tumour, they are non-cancerous (benign) brain tumour and cancerous(malignant) brain tumour. Survival rate of a tumour prone patient is difficult to predict because brain tumour is uncommon and are different types. As per the cancer research by United Kingdom, around 15 out of every 100 people with brain cancer will be able to survive for ten or more years after being diagnosed. Treatment for brain tumour depends on various factors like: the type of tumour, how abnormal the cells are and where it is in the brain etc. With the growth of Artificial Intelligence, Deep learning models are used to diagnose the brain tumour by taking the images of magnetic resonance imaging. Magnetic Resonances Imaging (MRI) is a type of scanning method that uses strong magnetic fields and radio waves to produce detailed images of the inner body. The research work carried out uses Deep learning models like convolutional neural network (CNN) model and VGG-16 architecture (built from scratch) to detect the tumour region in the scanned brain images. We have considered Brain MRI images of 253 patients, out of which 155 MRI images are tumorous and 98 of them are non-tumorous. The paper presents a comparative study of the outcomes of CNN model and VGG-16 architecture used.

FEATURE(S) TAKEN

✓ *DATASET DESCRIPTION*

The dataset chosen for this work is collected from Kaggle which consists of MRI scanned images of 253 patients out of which 155 of them are malignant and 98 of them are benign. The presented model detects the tumour in the scanned Brain MRI image of a patient.

✓ *FEATURES TAKEN*

- Edges and blob like structures
- Kernels for extracting the spatial features of a part of the image.

MODEL

The proposed work uses Deep learning models like **Convolutional Neural Network** (*CNN*) model and **VGG-16 architecture** (*both built from scratch*) to detect the tumour region in the scanned Brain MRI images of a patient.

PROGRAM

Import statements

```
import os
import time
import keras
import cv2 as cv
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from keras.optimizers import adam_v2
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Activation, Flatten, Zerc
```

Start time

```
Start = time.time()
```

Constants Declaration

```
HeightCNN = 240
WidthCNN = 240

HeightVGG = 224
WidthVGG = 224

TumorImage = 1085
NonTumorImage = 980

TestingTumor = 175
TestingNonTumor = 135

ValidationTumor = 152
ValidationNonTumor = 158

Epoch = 25
```

Preprocessing

Geting filenames and their classes

```
Df = pd.DataFrame()
for i in os.listdir("/content/drive/MyDrive/Brai MRI datasets/1/no"):
  Df = Df.append({"filename": "/content/drive/MyDrive/Brai MRI datasets/1/no/" + i, "class
for i in os.listdir("/content/drive/MyDrive/Brai MRI datasets/1/yes"):
  Df = Df.append({"filename": "/content/drive/MyDrive/Brai MRI datasets/1/yes/" + i, "clas
```

Augmentation

Creating an image generator for Augmentation

```
train_datagen = ImageDataGenerator(rotation_range=5,
                                   width_shift_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   brightness_range=[0.2,0.8])
Generator = train_datagen.flow_from_dataframe(Df,
                                              class_mode="binary",
                                              target_size=(HeightVGG,HeightVGG),
                                              color_mode = "rgb",
                                              batch_size=1,
                                              rescale=1.0/255)
```

```
Found 253 validated image filenames belonging to 2 classes.
```

Getting images data from Image generator based on our needs

[ ] ↳ *1 cell hidden*

Label encoding classes to integer

[ ] ↳ *3 cells hidden*

Eroding and dilating the images (making the image blur)

Eroding and dilating (function)

[ ] ↳ *1 cell hidden*

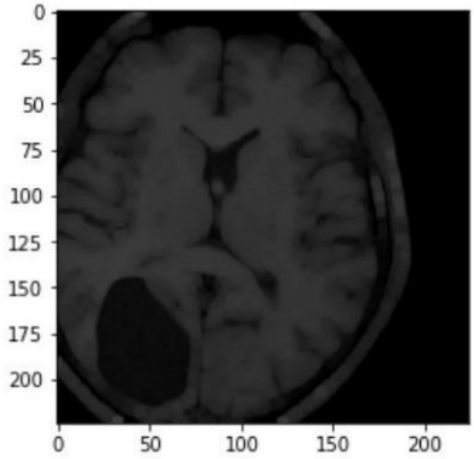
Eroding and dilating the images in the dataframe

```
AugmentedDf.Image224X224 = AugmentedDf.Image224X224.apply(Erosion_Dilation)
```

Visualization of the processed data

```
plt.imshow(AugmentedDf.Image224X224[0])
```

```
<matplotlib.image.AxesImage at 0x7fba0c6acd10>
```




```
AugmentedDf.Image224X224[0].shape
```

```
(224, 224, 3)
```

Contouring and resizing the image

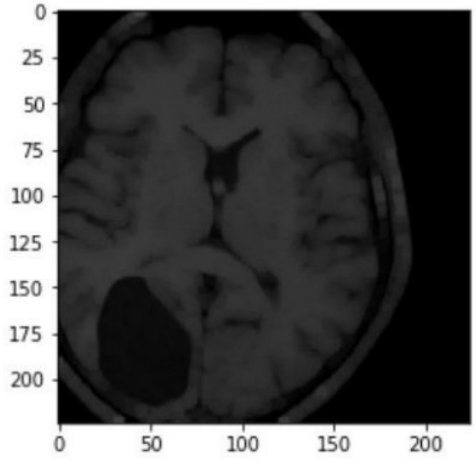
Countouring and resizing(function)

```
def Contour(src,Dim):
  UpperBoundary = 255
  LowerBoundary = 20

  src = cv.resize(src,Dim)
  Lower = np.array(LowerBoundary, dtype="uint8")
  Upper = np.array(UpperBoundary, dtype="uint8")
  mask = cv.inRange(src, Lower, Upper)
  output = cv.bitwise_and(src, src, mask=mask)

  ret,thresh = cv.threshold(mask, 40, 255, 0)
  contours, hierarchy = cv.findContours(thresh, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
  if len(contours) != 0:
    c = max(contours, key = cv.contourArea)
    x,y,w,h = cv.boundingRect(c)
    cv.rectangle(output,(x,y),(x+w,y+h),(255,255,255),1)
    return cv.resize(src[y:y+h,x:x+w],Dim)
  return src
```

Visualizing the images after and before contour, resizing

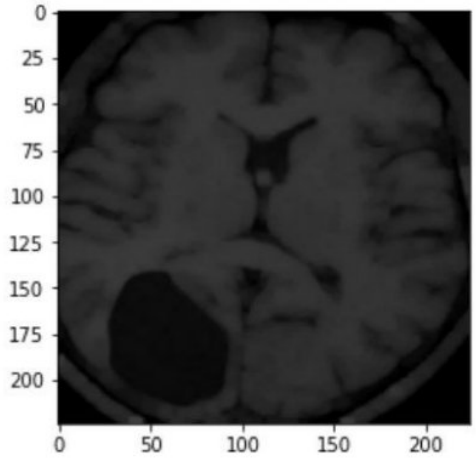
```
plt.imshow(AugmentedDf.Image224X224[0])
```

```
<matplotlib.image.AxesImage at 0x7fba0c1875d0>
```




```
plt.imshow(Contour(AugmentedDf.Image224X224[0],(HeightVGG,WidthVGG)))
```

```
<matplotlib.image.AxesImage at 0x7fba0c65da10>
```



Creating contour images of (224x224) for VGG and (224x224) for CNN

```
AugmentedDf.Image224X224 = AugmentedDf.Image224X224.apply(Contour,args=((HeightVGG,WidthVG
AugmentedDf["Image240X240"] = AugmentedDf.Image224X224.apply(Contour,args=((HeightCNN,Widt
```

Reordering the columns in the dataset

```
AugmentedDf = AugmentedDf[["Image240X240","Image224X224","class"]]
```

Visualizing the columns changes

```
AugmentedDf.head()
```

	Image240X240	Image224X224	class
0	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	0
1	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	1
2	[[[1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], ...	[[[1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], ...	1
3	[[[6, 6, 6], [6, 6, 6], [6, 6, 6], [6, 6, 6], ...	[[[6, 6, 6], [6, 6, 6], [6, 6, 6], [6, 6, 6], ...	0
4	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	0

Split

Creating the Train,Test,Validate split based on the given count

```
TrainingDf = pd.DataFrame()
ValidationDf = pd.DataFrame()
TestingDf = pd.DataFrame()

for i in AugmentedDf.index:
  if TestingTumor != 0 and AugmentedDf['class'][i] == 1:
    TestingTumor -= 1
    TestingDf  = TestingDf.append(AugmentedDf.iloc[i],verify_integrity=True, ignore_index
  elif TestingNonTumor != 0 and AugmentedDf['class'][i] == 0:
    TestingNonTumor -= 1
    TestingDf  = TestingDf.append(AugmentedDf.iloc[i],verify_integrity=True, ignore_index
  elif ValidationTumor != 0 and AugmentedDf['class'][i] == 1:
    ValidationTumor -= 1
    ValidationDf  = ValidationDf.append(AugmentedDf.iloc[i],verify_integrity=True, ignore_
  elif ValidationNonTumor != 0 and AugmentedDf['class'][i] == 0:
    ValidationNonTumor -= 1
    ValidationDf  = ValidationDf.append(AugmentedDf.iloc[i],verify_integrity=True, ignore_
  else:
    TrainingDf = TrainingDf.append(AugmentedDf.iloc[i],verify_integrity=True, ignore_inde›
```

Visualizing the shape of train,test and validate dataframe

```
TrainingDf.shape,TestingDf.shape,ValidationDf.shape
```

```
((1445, 3), (310, 3), (310, 3))
```

Shuffling the rows in the train,test and validate dataframe

```
TrainingDf = TrainingDf.sample(frac = 1)
TestingDf = TestingDf.sample(frac = 1)
ValidationDf = ValidationDf.sample(frac = 1)
```

Numpy to tensor converter

Numpy to tensor converting(function)

```
def CNN_Formatter(X,Y):
  return tf.convert_to_tensor(list(X)),tf.convert_to_tensor(list(Y))
```

Converting and creating tensor iterator for model

```
X_CNN , Y_CNN = CNN_Formatter(TrainingDf.Image240X240,TrainingDf["class"])
X_CNN_Val,Y_CNN_Val = CNN_Formatter(ValidationDf.Image240X240,ValidationDf["class"])
X_CNN_Test,Y_CNN_Test = CNN_Formatter(TestingDf.Image240X240,TestingDf["class"])
```

Model

CNN Model

CNN Sequential model

```
ModelCNN = Sequential()
ModelCNN.add(InputLayer(input_shape=(HeightCNN,WidthCNN,3)))
ModelCNN.add(ZeroPadding2D(padding=(2,2)))
ModelCNN.add(Conv2D(filters=32,kernel_size=(7,7)))
ModelCNN.add(BatchNormalization())
ModelCNN.add(Activation(activation='relu'))
ModelCNN.add(MaxPooling2D(pool_size=(4,4),strides=4))
ModelCNN.add(MaxPooling2D(pool_size=(4,4),strides=4))
ModelCNN.add(Flatten())
ModelCNN.add(Dense(units=1))
```

CNN Model summary

```
ModelCNN.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 zero_padding2d (ZeroPadding  (None, 244, 244, 3)      0
 2D)

 conv2d (Conv2D)             (None, 238, 238, 32)      4736

 batch_normalization (BatchN  (None, 238, 238, 32)     128
 ormalization)

 activation (Activation)     (None, 238, 238, 32)      0

 max_pooling2d (MaxPooling2D  (None, 59, 59, 32)       0
 )

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 1)                 6273

=================================================================
Total params: 11,137
Trainable params: 11,073
Non-trainable params: 64
_________________________________________________________________
```

CNN model Compile

```
ModelCNN.compile(optimizer = 'adam', loss = tf.keras.losses.BinaryCrossentropy(from_logits
```

CNN model fit

```
ModelCNN.fit(x = X_CNN,y = Y_CNN,epochs=Epoch, validation_data=(X_CNN_Val,Y_CNN_Val),batch
```

```
Epoch 1/25
145/145 [==============================] - 36s 35ms/step - loss: 0.7413 - accuracy:
Epoch 2/25
145/145 [==============================] - 4s 31ms/step - loss: 0.5630 - accuracy: 0
Epoch 3/25
145/145 [==============================] - 5s 31ms/step - loss: 0.5189 - accuracy: 0
Epoch 4/25
145/145 [==============================] - 5s 33ms/step - loss: 0.4919 - accuracy: 0
Epoch 5/25
145/145 [==============================] - 5s 31ms/step - loss: 0.4232 - accuracy: 0
Epoch 6/25
145/145 [==============================] - 5s 33ms/step - loss: 0.3933 - accuracy: 0
Epoch 7/25
145/145 [==============================] - 5s 33ms/step - loss: 0.3834 - accuracy: 0
Epoch 8/25
145/145 [==============================] - 5s 33ms/step - loss: 0.4609 - accuracy: 0
Epoch 9/25
```

```
145/145 [==============================] - 5s 31ms/step - loss: 0.3207 - accuracy: 0
Epoch 10/25
145/145 [==============================] - 5s 33ms/step - loss: 0.2972 - accuracy: 0
Epoch 11/25
145/145 [==============================] - 5s 31ms/step - loss: 0.3076 - accuracy: 0
Epoch 12/25
145/145 [==============================] - 5s 31ms/step - loss: 0.2856 - accuracy: 0
Epoch 13/25
145/145 [==============================] - 5s 33ms/step - loss: 0.2684 - accuracy: 0
Epoch 14/25
145/145 [==============================] - 5s 31ms/step - loss: 0.2374 - accuracy: 0
Epoch 15/25
145/145 [==============================] - 5s 33ms/step - loss: 0.2386 - accuracy: 0
Epoch 16/25
145/145 [==============================] - 5s 33ms/step - loss: 0.2720 - accuracy: 0
Epoch 17/25
145/145 [==============================] - 5s 33ms/step - loss: 0.2227 - accuracy: 0
Epoch 18/25
145/145 [==============================] - 5s 31ms/step - loss: 0.2110 - accuracy: 0
Epoch 19/25
145/145 [==============================] - 5s 31ms/step - loss: 0.1834 - accuracy: 0
Epoch 20/25
145/145 [==============================] - 5s 33ms/step - loss: 0.1857 - accuracy: 0
Epoch 21/25
145/145 [==============================] - 5s 33ms/step - loss: 0.1786 - accuracy: 0
Epoch 22/25
145/145 [==============================] - 5s 31ms/step - loss: 0.1720 - accuracy: 0
Epoch 23/25
145/145 [==============================] - 4s 31ms/step - loss: 0.1697 - accuracy: 0
Epoch 24/25
145/145 [==============================] - 5s 33ms/step - loss: 0.1793 - accuracy: 0
Epoch 25/25
145/145 [==============================] - 5s 33ms/step - loss: 0.1496 - accuracy: 0
<keras.callbacks.History at 0x7fba00157210>
```

CNN model evaluate

```
ModelCNN.evaluate(X_CNN_Test,Y_CNN_Test,batch_size=10)
```

```
31/31 [==============================] - 0s 13ms/step - loss: 0.5080 - accuracy: 0.8
[0.5080010294914246, 0.8548387289047241]
```

```
End = time.time()
Diff = End - Start
print(f"Runtime of the program is {End - Start}")
```

```
Runtime of the program is 590.0425834655762
```

VGG Model

VGG Sequential model

```
ModelVGG = Sequential()
ModelVGG.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same", a
ModelVGG.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
ModelVGG.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
ModelVGG.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
ModelVGG.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
ModelVGG.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
ModelVGG.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
ModelVGG.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
ModelVGG.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
ModelVGG.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
ModelVGG.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
ModelVGG.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
ModelVGG.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
ModelVGG.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
ModelVGG.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
ModelVGG.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
ModelVGG.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
ModelVGG.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
ModelVGG.add(Flatten())
ModelVGG.add(Dense(units=4096,activation="relu"))
ModelVGG.add(Dense(units=4096,activation="relu"))
ModelVGG.add(Dense(units=2, activation="sigmoid"))
```

VGG Model summary

```
ModelVGG.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)            (None, 224, 224, 64)      1792

 conv2d_2 (Conv2D)            (None, 224, 224, 64)      36928

 max_pooling2d_2 (MaxPooling  (None, 112, 112, 64)     0
 2D)

 conv2d_3 (Conv2D)            (None, 112, 112, 128)     73856

 conv2d_4 (Conv2D)            (None, 112, 112, 128)     147584

 max_pooling2d_3 (MaxPooling  (None, 56, 56, 128)      0
 2D)

 conv2d_5 (Conv2D)            (None, 56, 56, 256)       295168

 conv2d_6 (Conv2D)            (None, 56, 56, 256)       590080

 conv2d_7 (Conv2D)            (None, 56, 56, 256)       590080

 max_pooling2d_4 (MaxPooling  (None, 28, 28, 256)      0
```

```
2D)

conv2d_8 (Conv2D)           (None, 28, 28, 512)       1180160

conv2d_9 (Conv2D)           (None, 28, 28, 512)       2359808

conv2d_10 (Conv2D)          (None, 28, 28, 512)       2359808

max_pooling2d_5 (MaxPooling  (None, 14, 14, 512)      0
2D)

conv2d_11 (Conv2D)          (None, 14, 14, 512)       2359808

conv2d_12 (Conv2D)          (None, 14, 14, 512)       2359808

conv2d_13 (Conv2D)          (None, 14, 14, 512)       2359808

max_pooling2d_6 (MaxPooling  (None, 7, 7, 512)        0
2D)

flatten_1 (Flatten)         (None, 25088)             0

dense_1 (Dense)             (None, 4096)              102764544

dense_2 (Dense)             (None, 4096)              16781312

dense_3 (Dense)             (None, 2)                 8194

=================================================================
Total params: 134,268,738
Trainable params: 134,268,738
Non-trainable params: 0
_________________________________________________________________
```

VGG model compile

```
opt = tf.keras.optimizers.Adam(lr=0.001)
ModelVGG.compile(optimizer=opt, loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning:
  super(Adam, self).__init__(name, **kwargs)
```

Double-click (or enter) to edit

VGG model fit

```
def VGG_Formatter(X,Y):
  Encoder = OneHotEncoder(sparse=False)
  Y = np.array(Y)

  return tf.convert_to_tensor(list(X)),tf.convert_to_tensor(list(Encoder.fit_transform(Y.r
```

```
X_VGG , Y_VGG = VGG_Formatter(TrainingDf.Image224X224,TrainingDf["class"])
X_VGG_Val,Y_VGG_Val = VGG_Formatter(ValidationDf.Image224X224,ValidationDf["class"])
X_VGG_Test,Y_VGG_Test = VGG_Formatter(TestingDf.Image224X224,TestingDf["class"])
```

```
Y_VGG.shape
```

```
TensorShape([1445, 2])
```

```
ModelVGG.fit(x = X_VGG,y = Y_VGG,epochs=Epoch, validation_data=(X_VGG_Val,Y_VGG_Val),batch
```

```
Epoch 1/25
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1096: User
  return dispatch_target(*args, **kwargs)
37/37 [==============================] - 75s 2s/step - loss: 57.8442 - accuracy: 0
Epoch 2/25
37/37 [==============================] - 42s 1s/step - loss: 0.6927 - accuracy: 0.
Epoch 3/25
37/37 [==============================] - 42s 1s/step - loss: 0.7113 - accuracy: 0.
Epoch 4/25
37/37 [==============================] - 42s 1s/step - loss: 0.7150 - accuracy: 0.
Epoch 5/25
37/37 [==============================] - 42s 1s/step - loss: 0.8526 - accuracy: 0.
Epoch 6/25
37/37 [==============================] - 41s 1s/step - loss: 0.6925 - accuracy: 0.
Epoch 7/25
37/37 [==============================] - 41s 1s/step - loss: 0.6923 - accuracy: 0.
Epoch 8/25
37/37 [==============================] - 42s 1s/step - loss: 0.6922 - accuracy: 0.
Epoch 9/25
37/37 [==============================] - 42s 1s/step - loss: 0.9764 - accuracy: 0.
Epoch 10/25
37/37 [==============================] - 42s 1s/step - loss: 0.6936 - accuracy: 0.
Epoch 11/25
37/37 [==============================] - 42s 1s/step - loss: 0.6929 - accuracy: 0.
Epoch 12/25
37/37 [==============================] - 42s 1s/step - loss: 0.6979 - accuracy: 0.
Epoch 13/25
37/37 [==============================] - 42s 1s/step - loss: 0.8158 - accuracy: 0
Epoch 14/25
37/37 [==============================] - 41s 1s/step - loss: 0.7092 - accuracy: 0
Epoch 15/25
37/37 [==============================] - 41s 1s/step - loss: 0.6933 - accuracy: 0.
Epoch 16/25
37/37 [==============================] - 42s 1s/step - loss: 0.6919 - accuracy: 0.
Epoch 17/25
37/37 [==============================] - 41s 1s/step - loss: 0.6924 - accuracy: 0
Epoch 18/25
37/37 [==============================] - 41s 1s/step - loss: 0.6922 - accuracy: 0.
Epoch 19/25
37/37 [==============================] - 41s 1s/step - loss: 0.6922 - accuracy: 0
Epoch 20/25
37/37 [==============================] - 41s 1s/step - loss: 0.6921 - accuracy: 0.
Epoch 21/25
37/37 [==============================] - 41s 1s/step - loss: 0.6921 - accuracy: 0.
Epoch 22/25
37/37 [==============================] - 41s 1s/step - loss: 0.6921 - accuracy: 0.
Epoch 23/25
37/37 [==============================] - 41s 1s/step - loss: 0.6921 - accuracy: 0
```

```
Epoch 24/25
37/37 [==============================] - 41s 1s/step - loss: 0.6922 - accuracy: 0.
Epoch 25/25
37/37 [==============================] - 41s 1s/step - loss: 0.6922 - accuracy: 0.
<keras.callbacks.History at 0x7fb993788410>
```

VGG model evaluate

```
ModelVGG.evaluate(X_VGG_Test,Y_VGG_Test,batch_size=10)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1096: User
  return dispatch_target(*args, **kwargs)
31/31 [==============================] - 8s 126ms/step - loss: 0.6884 - accuracy: 0.
[0.6883537173271179, 0.5645161271095276]
```

```
End = time.time()
```

```
End = time.time()
print(f"Runtime of the program is {End - Start - Diff}")
```

```
Runtime of the program is 1270.9331192970276
```

PERFORMANCE METRICS/ OUTPUT

Accuracy and computation time are used as the metric for evaluation.

- **Convolutional Neural Network**

  Accuracy = 0.933

  Validation Accuracy = 0.864

  Test set accuracy = 0.916

  Computation time = 590 seconds

- **VGG-16 architecture**

  Accuracy = 0.853

  Validation Accuracy = 0.844

  Test set accuracy = 0.815

  Computation time = 1270 seconds

REFERENCES

S. Grampurohit, V. Shalavadi, V. R. Dhotargavi, M. Kudari and S. Jolad, "Brain Tumor Detection Using Deep Learning Models," *2020 IEEE India Council International Subsections Conference (INDISCON)*, 2020, pp. 129-134, doi: 10.1109/INDISCON50162.2020.00037.

BRAIN TUMOR DETECTION USING DEEP LEARNING MODELS

Sneha Grampurohit
Electronics and Communication Engineering
K.L.E Institute of Technology
Hubli, India
snehagrampurohit5@gmail.com

Venkamma Shalavadi
Electronics and Communication Engineering
K.L.E Institute of Technology
Hubli, India
priyankashalavadi@gmail.com

Vaishnavi R. Dhotargavi
Electronics and Communication Engineering
K.L.E Institute of Technology
Hubli, India
vaishnavidhotargavi98@gmail.com

Megha Kudari
Electronics and Communication Engineering
K.L.E Institute of Technology
Hubli, India
meghakudari0@gmail.com

Mrs Soumya Jolad
Asst. Prof. Electronics and Communication Engineering
K.L.E Institute of Technology
Hubli, India
soumyajolad4@gmail.com

Abstract: A brain tumor is a disease caused due to the growth of abnormal cells in the brain. There are two main categories of brain tumor, they are non-cancerous (benign) brain tumor and cancerous(malignant) brain tumor. Survival rate of a tumor prone patient is difficult to predict because brain tumor is uncommon and are different types. As per the cancer research by United Kingdom, around 15 out of every 100 people with brain cancer will be able to survive for ten or more years after being diagnosed. Treatment for brain tumor depends on various factors like: the type of tumor, how abnormal the cells are and where it is in the brain etc. With the growth of Artificial Intelligence, Deep learning models are used to diagnose the brain tumor by taking the images of magnetic resonance imaging. Magnetic Resonances Imaging (MRI) is a type of scanning method that uses strong magnetic fields and radio waves to produce detailed images of the inner body. The research work carried out uses Deep learning models like convolutional neural network (CNN) model and VGG-16 architecture (built from scratch) to detect the tumor region in the scanned brain images. We have considered Brain MRI images of 253 patients, out of which 155 MRI images are tumorous and 98 of them are non-tumorous. The paper presents a comparative study of the outcomes of CNN model and VGG-16 architecture used.

Keywords: CNN, VGG-16, Tumor cells, Data pre-processing, Convolution.

I. INTRODUCTION

A brain tumor is a disease caused due to the abnormal growth of mass in the brain. Normally in our body, new cells are produced which replace the old and damaged cells in a controlled manner. But in case of brain tumor, tumor cells go on multiplying uncontrollably. As per the National Brain Tumor Society nearly 70,000 people in United States are suffering from primary brain tumor. Brain tumor is ranked as 10th most common tumor in India. The presence of tumor is noticed by the Magnetic Resonance Imaging [MRI] scanning. The MRI scanning should be diagnosed by the physician and later based on the results; the treatments shall be started. This procedure can be a little time consuming. Hence to overcome this, the proposed work presents an automated system that will classify if the subjected patient is suffering from brain tumor. This system can assist the physician to make early decisions so that the treatments are carried out at an earlier stage. The proposed approach uses CNN and VGG-16 architecture and weights to train the model for this binary problem. Accuracy is used as the metric for evaluation.

In the presented approach, we have augmented the dataset (MRI images of brain), performed certain data pre-processing steps to convert the raw data, further investigated two deep learning models namely CNN and VGG-16 and have presented the comparative analysis in the results section. Depending upon the algorithm complexity, computation time and other results one can choose any of the above-mentioned algorithm in their work. This automatic detection system can assist the physician to make early decisions and hence start the treatments at an early stage.

II. OVERVIEW

The dataset considered consists of MRI scanned images of 253 patients out of which 155 of them are tumorous and 98 of them are non-tumorous. The presented work aims to develop a detection model that detects the tumour in the MRI scanned image of a patient. The detection model in general can be given as:

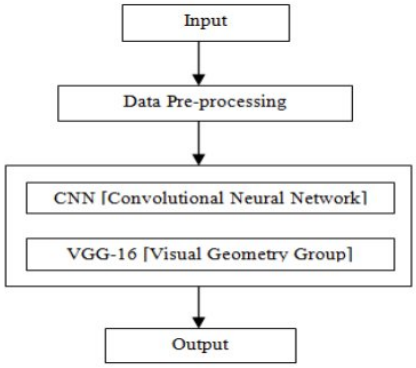


Fig 1. Flowchart of General Brain tumour detection model

978-1-7281-8734-1/20/$31.00 ©2020 IEEE
DOI 10.1109/INDISCON50162.2020.00037

A. Input

It is assumed that the patient is all fit and capable to undergo an MRI scan as per the doctor's assistance. The present work considers the Brain MRI images of a patient as the input.

B. Data Pre-processing

For the easy interpretation of the data, the data should be transformed from its raw state. The data pre-processing steps we considered are:

1. Importing libraries
2. Data augmentation
3. Import the augmented data
4. Convert the images to grayscale
5. Removal of noise using dilations and erosions And smoothening of images.
6. Grab the largest contour.
7. Find the extreme points of the contoured image
8. Resize the image
9. Crop the images using the extreme points
10. Splitting of dataset.

C. Algorithms used:

The algorithms used in the proposed work are:

1. Convolutional Neural Network [CNN].
2. VGG-16 Architecture.

D. Output

The system is trained to detect the tumour in the MRI of the patient and thus predict whether the patient is suffering from tumour or no.

III. METHODOLOGY

A. Data pre-processing

Data pre-processing is a method in which the raw data is converted to a useful data by applying some pre-processing techniques. The pre-processing techniques we have used are:

***Step1**: Importing libraries*

The libraries such as TensorFlow, NumPy, pandas, matplotlib, os and scikit-learn etc are imported.

***Step 2**: Data augmentation*

The images in the dataset are increased by creating modified versions of the image using techniques like rotation etc, this process is known as Image data-augmentation. *Image Data Generator* class is used to generate the images. Before augmentation, 155 tumorous and 98 non-tumorous images were present and after augmentation we have 1085 tumorous and 980 non-tumorous images

***Step 3**: Import the augmented data*

.

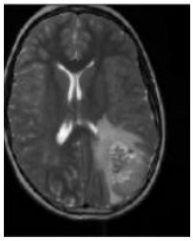


Fig2: The original image

***Step 4**: Convert the images to grayscale*

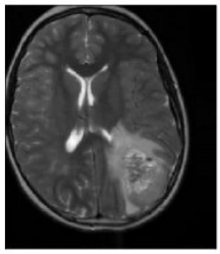


Fig 3 : Original image converted to gray scale

***Step 5**: Removal of noise and Smoothening the image*

Noise is removed using series of erosions and dilations. Erosion involves removal of pixel at the edges of the image. Dilation is the reverse process of erosion where it adds the pixels at the edges of the image. Smoothening of the image is done using Gaussian blur technique. In Gaussian blur operation the image is convolved with the Gaussian filter. Gaussian filter is a low pass filter that removes high frequency components in the image. Thus, smoothens the image.

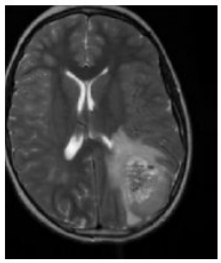


Fig 4: Image after applying Gaussian blur and removal of noise

***Step 6:** Grab the largest contour.*

Contour refers to the boundary or the outline of a shape.

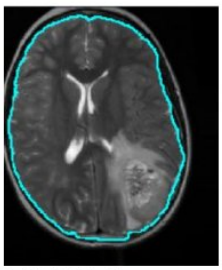


Fig 5: Find the largest contour

***Step 7**: Find the extreme points of the contoured image.*

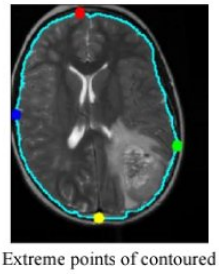


Fig 6: Extreme points of contoured image

Step 8: *Resize the image*
All the images before pre-processing were of different size, hence all the augmented images were resized to 240x240 for CNN and 224x224 for VGG-16.

Step 9: *Crop the images using the extreme points*
The images were cropped along the extreme points

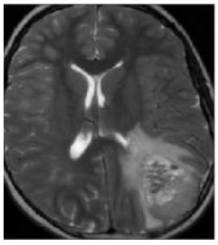


Fig 7: Cropped image

Step 10*: Splitting of dataset:*
The dataset is divided into 3 sets of data. They are
Training data:
No. Of images: 1445
No. Of tumorous images: 758
No. Of non-tumorous images: 687
Testing data:
No. Of images: 310
No. Of tumorous images: 175
No. Of non-tumorous images: 135
Validation data:
No. Of images: 310
No. Of tumorous images: 152
No. Of non-tumorous images: 158

B. Convolutional Neural Network (CNN)

Each layer in a CNN applies a different set of filters, typically hundreds or thousands of them and combines the results, feeding the output into the next layer in the network. During training, a CNN automatically learns the values for these filters.
CNN gives us two key benefits

1. *Local variance*: The concept of local invariance allows us to classify an image as containing object regardless of where in the image the object appears. We obtain this local invariance through the usage of "pooling layers" which identifies regions of our input volume with a high response to a filter.
2. *Compositionality*: Each filter composes a local path of lower level features into a higher-level representation, this composition allows our network to learn more rich features deeper in network.

Kernel: A kernel can be visualized as a small matrix that slides across, from left to right and top to bottom of a large image. At each pixel in the input image, the neighborhood of the image is convolved with the kernel and the output is stored.
Example: Convolving (denoted mathematically as * operator) a 3x3 region of an image with a 3x3 kernel.

$$O_{ij} = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} * \begin{bmatrix}93 & 139 & 101\\ 26 & 252 & 196\\ 135 & 230 & 18\end{bmatrix}$$

$$= \begin{bmatrix}10.3 & 15.4 & 11.2\\ 2.6 & 28.0 & 21.7\\ 15.0 & 25.5 & 2.0\end{bmatrix}$$

$$= 132$$

After applying this convolution, the pixel located at the co-ordinate (i, j) of the output image O to is set to Oij=132. By applying convolutions filters, nonlinear activation functions, pooling and back propagations, CNN's are able to learn filters that can detect edges and blob like structures in lower level layers of the network and then use the edges and structures as "building blocks", eventually detecting high level objects in the deeper layer of the network.
The flow graph of Convolutional Neural Network

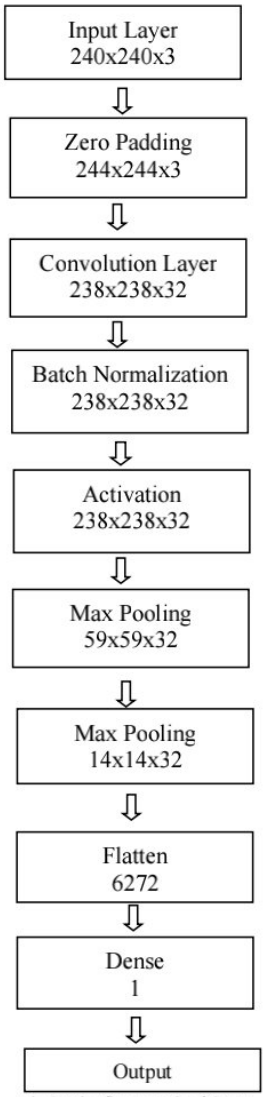


Fig 8: The flow graph of CNN

1. Input layer:

We have resized the images to 240x240 pixels height=240pixels and width=240 pixels with depth=3 [number of channels of the image].

2. *Zero padding:*

We need to "pad" the borders of an image to retain the original image size when applying a convolution. The same is true for filters inside a CNN. Using zero, we can "pad" our input along the borders such that output volume size matches input volume size.

We have performed zero padding on our images using pool size (2,2).

Example: Consider an image represented in the form of matrix:

95	242	186	152	39
39	19	220	153	180
5	247	212	59	46
46	77	133	110	79
156	35	79	93	116

Fig 9: A 5x5 image in its matrix form

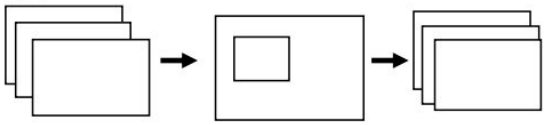
Padding the image with zeroes i.e. of pool size (2,2)

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	95	242	186	152	39	0	0
0	0	39	19	220	153	180	0	0
0	0	5	247	212	59	46	0	0
0	0	46	77	133	110	79	0	0
0	0	156	35	79	93	116	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Fig 10: Zero padding applies to image

3. *Convolution:*

The convolution layer is considered as the core building block of a "Convolutional Neural Network". The convolution layer parameter consists of a set of K learnable filter (i.e. kernels). Consider the forward pass of the CNN



Step1: K-Kernels each Waiting to be applied to The image.

Step2: Kernel applied to the input volume

Step3: The output Convolved image Produces Activation Maps.

Fig 8: Convolution of k-kernels with an image

In the convolution layer of our presented work, we have used 32 filters of size (7,7). The size (7,7) refers to receptive field [local region of input volume to which each neuron is connected]. We have used (7x7) receptive field, i.e. each neuron will connect to a 7x7 local region by the image for a total 7x7x3 =147 weights. After applying all 32 filters to the input volume, we now have 32, 2-dimenstional activation maps. Every entry in the output volume is thus an output of a neuron that "looks" at only a small region of an input. In this manner the network "learns" filters that activate when they see a specific type of feature. After the convolution operation, the size of image is now (238, 238,32) depth has increased due to the filter used.

Stride: we have used the stride values s=1

Example: Consider an Image being convolved with Laplacian Kernel with stride=1.

95	242	186	152	39
39	19	220	153	180
5	247	212	59	46
46	77	133	110	79
156	35	79	93	116

Fig 9: Image

0	1	0
1	-1	1
0	1	1

Fig 10: Laplacian kernel

Using s=1, our kernel slides from left to right and top to bottom, one pixel at a time producing the below output,

692	-315	-6
-680	-194	305
153	-59	-86

Fig 11: The resultant matrix

Batch normalization: Batch normalization layers (or BN in short) as the name suggests, are used to normalize the activation of a given input volume before passing it into next layers in network.

If we consider x to be our mini batch of activation, then we can compute the normalized $\hat{x}$ via the following equation.

$$\widehat{x_i} = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \varepsilon}} \qquad (1)$$

During training, we compute the μ_β and σ_β over each mini batch β, where

$$\mu_\beta = \frac{1}{M}\sum_{i=1}^{m} x_i \qquad (2)$$

$$\sigma_\beta^2 = \frac{1}{M}\sum_{i=1}^{m}(x_i - \mu_\beta)^2 \qquad (3)$$

We set ε equal to a small positive value to avoid taking the square root of zero. Applying this equation implies that the activation having a batch normalization layer will have approximately zero mean and unit variance. Batch normalization also has the added benefits of helping "stabilize" training, allowing for a larger variety of learning rates and regularization strengths.

4. *Activation:*

After convolution layer, we have applied a nonlinear activation function. An activation layer accepts an input volume of size W_{input} x H_{input} x D_{input} and then applies the activation function (below fig). Since the activation function is applied in an element wise manner, the output of an activation layer is always same as input dimension, i.e.,

$W_{input}=W_{output}$, $H_{input}=H_{output}$, $D_{input}=D_{output}$.

Example: Consider an image to which RELU activation with max (0, x) function applied

-249	-91	-37
250	-134	101
27	61	-153

Fig 12: An example image

0	0	0
250	0	101
27	61	0

Fig 13: Resultant matrix

5. Pooling layers

There are two methods to reduce the size of an input volume.

- Convolutional layers with stride >1
- POOL layers

The primary function of the POOL layers is to progressively reduce the spatial size (i.e. width and height) of the input volume. Doing this allows us to reduce the amount of parameters and computation in the network. Pooling also helps us control over fitting.

We have used Max function in the pooling layer i, e Max-pooling with the pool size of FxS i.e. [receptive field size into stride]. Applying the pooling operation yields an output volume of size W_{output} x H_{output} x D_{output},

Where

$$W_{output} = ((W_{input}-F)/S) +1$$
$$H_{output} = ((H_{input}-F)/S) +1$$
$$D_{output} = D_{input}$$

Example: Consider an image to which a mac pooling of pool size (2x2) is applied with stride=1

181	237	170	223
229	181	89	108
109	93	48	66
108	21	71	19

Fig14: Input image matrix

237	237	223
229	181	108
109	93	71

Fig 15: Resultant Matrix

In the presented work in CNN, we have used 2 max pooling layers with pool size 4x4 and stride=1.

6. Flatten

This layer is used in order to flatten 3-dimension matrix to 1-dimensionmatrix. In the presented work, after flatten layer the array size obtained is 6272.

7. Dense

Dense is an output unit. It is fully connected with one neuron with sigmoid activation as we have binary problem of detecting brain tumor detection. If dense output is 1 then it indicates tumorous MRI. If dense output is 0 then it indicates non tumorous MRI.

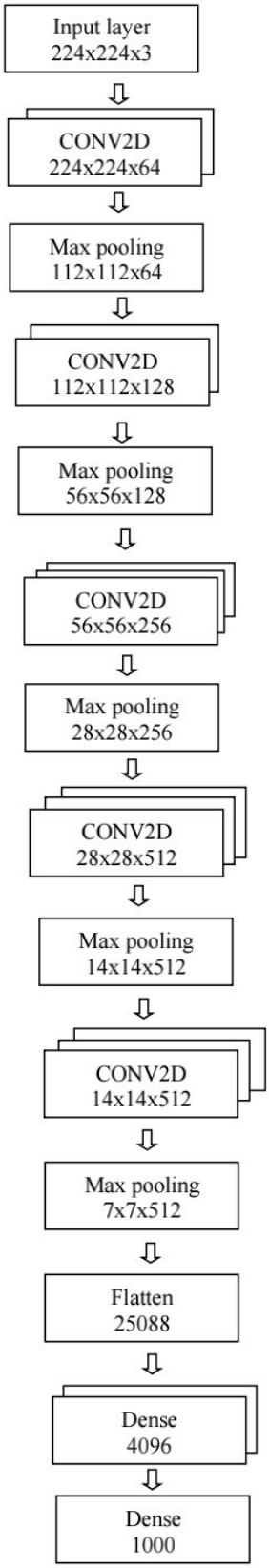
C. VGG-16 ARCHITECTURE

The VGG network was introduced by Simonyi and Zisserman in this 2014 paper, "Very deep convolutional network for large scale image recognition".

The VGG family of Convolutional Neural Networks can be characterized by two key points:

1. All convolution layers in the network use only 3X3 filters.
2. Stacking multiple convolution + RELU layer sets (the number of consecutive convolution layers+ RELU layers normally increases the deeper we go) before applying a pool operation.

In the presented work, we haven't used the transfer learning technique, instead we have built the VGG-16 architecture and made the necessary changes in the architecture to have the better accuracy. The architecture of VGG-16 architecture used id given by:



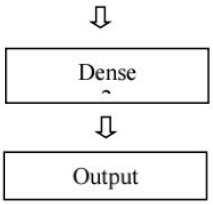


Fig 16: Flow graph of VGG-16 Architecture

IV. RESULTS

The comparative results after using the same dataset on CNN model and VGG-16 architecture are as shown below.

CNN Results:

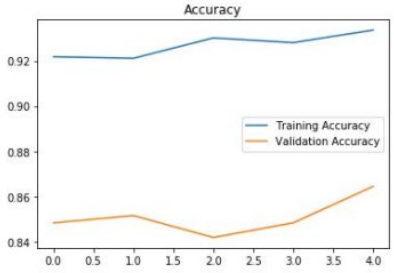


FIG 17: Accuracy chart

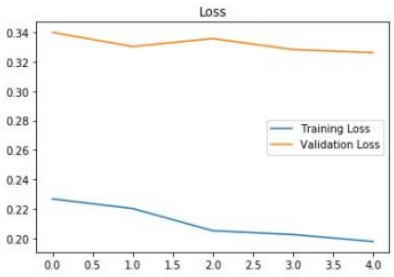


Fig 18: Loss chart

From the above plots of results, it can be inferred that:

- The accuracy obtained is 93 percent with validation accuracy equal to 86 percent. This accuracy is obtained at the 23rd epoch of the training.

VGG-16 Results:

From the VGG-16 result plots below one can infer that:

- All though the accuracy rate is increasing up to 100 percent (Fig 20), we have estimated the best model at 17th epoch which yields 97.16 percent training accuracy and 97.42 percent validation accuracy as after the 17th epoch, although the accuracy was increasing, validation accuracy was gradually decreasing.

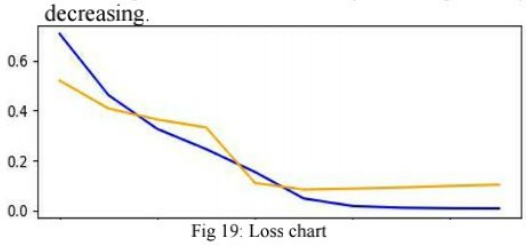


Fig 19: Loss chart

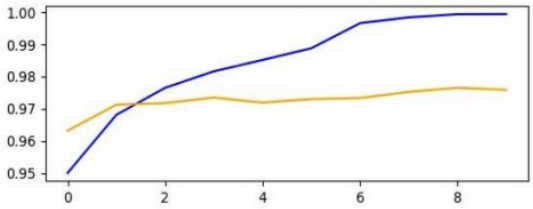


Fig 20: Accuracy chart

In fig 19 and 20 blue color refers to Training loss and Training accuracy and orange lines represent Validation loss and Validation accuracy.

Table 1: Parameter wise comparison

Parameter	CNN	VGG-16
No. of images used	2065 Training-1445 Test-310 Validation- 310	
Time consumed [From pre-processing till obtaining results]	0:5:03 [5mins: 3secs] GPU (GOOGLE COLAB)	0:15:25 [15mins:25secs] GPU (GOOGLE COLAB)
Epochs carried out	25	
Accuracy	0.9336	0.9716
Validation Accuracy	0.8645	0.9742
Test set accuracy	0.916	0.919

V. CONCLUSION

In proposed work, Deep neural networks such as CNN and VGG-16 are investigated on MRI images of Brain. Both the models have given an effective result, However VGG-16 takes a greater computational time and memory but has given satisfactory results compared to CNN. Due to the availability of huge data being produced and stored by the medical sector, Deep learning will play an important role in data analysis in the upcoming days.

VI. REFERENCE

1. Yuehao Pan, Weimin Huang et.al "Brain Tumor Grading based on Neural networks and Convolutional Neural Networks",*IEEE.*
2. Darko Zikic, Yani Ioannou et.al "Segmentation of Brain tumor tisssues with convolutional neural networks", *MiCCAI,2014,Brats challenge.*
3. Karen, Zisserman "Very deep convolutional networks for large scale image recognition",*ICLR.*[Published in 2014].
4. Taranjit Kaur, Tapan Kuamar Gandhi, "Automated brain image classification based on VGG-16 and tranfer learning",*ICIT* [Published in 2019].