Configuration Manual

MSc Research Project
MSc Data Analytics

Vibhash Anil Kumar Shrivastava
Student ID: x19236263

School of Computing
National College of Ireland

Supervisor: Dr. Barry Haycock

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Vibhash Anil Kumar Shrivastava		
Student ID:	X19236263		
Programme:	MSc Data Analytics	**Year:**	Jan 2021-2022
Module:	MSc Research Project		
Lecturer:	Dr. Barry Haycock		
Submission Due Date:	31st January 2022		
Project Title:	Implementation of Cascaded CNN architecture for fully automated Multiple modalities-based Brain tumour segmentation using MRI scans using selective overlapping patches.		

Word Count: 1047 **Page Count: 12**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Vibhash Shrivastava

Date: 31/01/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Vibhash Anil Kumar Shrivastava
X19236263

1. Introduction

This Document will discuss the Hardware, System configuration, Software and various technology stack require for the execution of the Research project. Below are the detailed stages that needs to be done in order to execute the Deep learning project.

2. System Configuration

- Processor – AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx @ 2.10 GHz
- GPU – 2 GB of AMD Vega RADEON
- RAM – 8GB DDR4
- Operating System – Windows 10 64-bit
- Storage – 500GB SSD

2.1Hardware

Device specifications

Device name	LAPTOP-9HK0R30G
Processor	AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz
Installed RAM	8.00 GB (5.95 GB usable)
Device ID	2B96DEF7-78BB-49DD-9104-EA7F63CCC783
Product ID	00327-35896-22274-AAOEM
System type	64-bit operating system, x64-based processor

Fig 1.Hardware Configuration for Project

2.2 Software

Software Used – Google Colab Pro

	Colab Free	Colab Pro	Colab Pro +
Guarantee of resources	Low	High	Even Higher
GPU	K80	K80, T4 and P100	K80, T4 and P100
RAM	16 GB	32 GB	52 GB
Runtime	12 hours	24 hours	24 hours
Background execution	No	No	Yes
Costs	Free	9.99$ per month	49.99$ per month
Target group	Casual user	Regular user	Heavy user

Table1.Google Colab versions comparasion

2.3 Technology Stack

Technologies Used –

- Pythion 3.9

- NumPy 1.21.4
- Pandas
- Keras 2.4.3
- TensorFlow 1.15 – backend
- SimpleTK
- Matplotlib

3. Implementation

The below diagram gives the overview of the End-to-End flow of working of our fully automated segmentation model, which includes stages from pre-processing to model training and post processing and finally prediction.

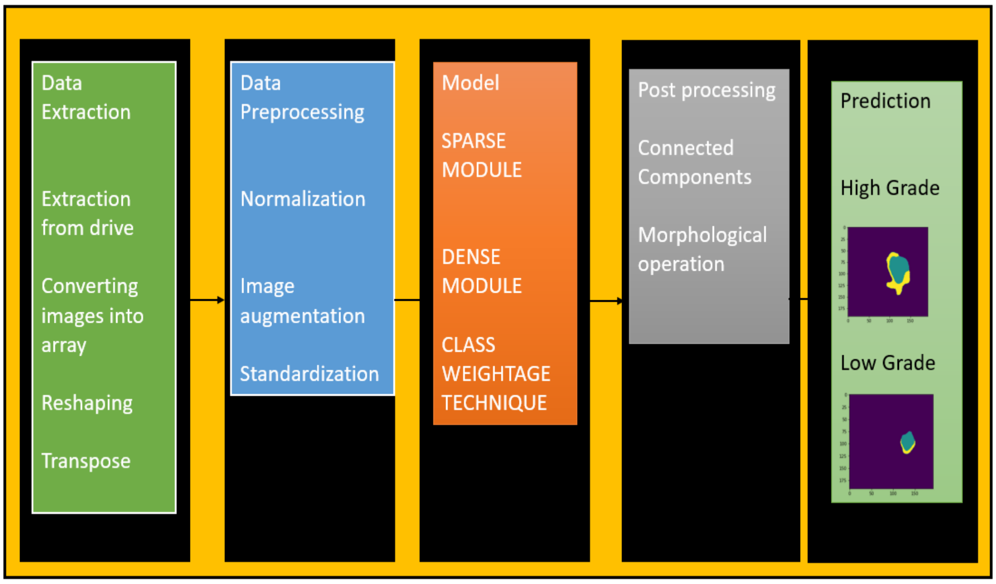


Fig 2..End to End Pipeline of brain tumor segenation project

3. Data Visualization

Here we have used thea images from the datset and visualize the image with high grade and low grade tumor and without tumor.

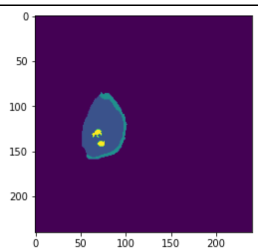
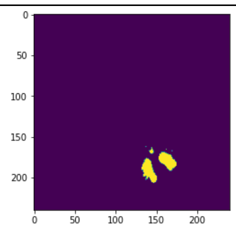
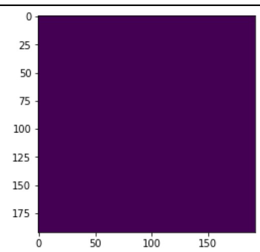
High grade tumor	Low grade Tumor	Without tumor
		

Fig 3.Data Visualization

4. Data Extracting & Preparation

Here first the data was downloaded in the drive and later extracted using the Google Colab pro software (Since size of data is 3GB) while extraction requires more space. Here we see we done image conversion and transpose along with reshaping the images in order to feed to the training and validation datasets.

Dataset source Link - https://www.kaggle.com/sanglequang/brats2018=

```
[ ] path = '/content/drive/My Drive/Brain_Tumor/Data/HGG/'
```

```
path1 = '/content/drive/My Drive/Brain_Tumor/Data/Data/HGG/'
path2 = '/content/drive/My Drive/Brain_Tumor/Data/Data/LGG/'
import SimpleITK as sitk
from tqdm import tqdm
import numpy as np
import os
```

```
[ ] def load_data(path):
      my_dir = sorted(os.listdir(path))
      data = []
      gt = []
      for p in tqdm(my_dir):
        data_list = sorted(os.listdir(path+p))
        img_itk = sitk.ReadImage(path + p + '/'+ data_list[0])
        flair = sitk.GetArrayFromImage(img_itk)
        img_itk = sitk.ReadImage(path + p + '/'+ data_list[1])
        seg =  sitk.GetArrayFromImage(img_itk)
        img_itk = sitk.ReadImage(path + p + '/'+ data_list[2])
        t1 =  sitk.GetArrayFromImage(img_itk)
        img_itk = sitk.ReadImage(path + p + '/'+ data_list[3])
        t1ce =  sitk.GetArrayFromImage(img_itk)
        img_itk = sitk.ReadImage(path + p + '/'+ data_list[4])
        t2 =  sitk.GetArrayFromImage(img_itk)
        data.append([flair,t1,t1ce,t2])
        gt.append(seg)
      data = np.asarray(data,dtype=np.float32)
      gt = np.asarray(gt,dtype=np.uint8)
      return data,gt
```

```
[ ] # for HGG
    #data1,gt1 = load_data(path1)   #dividing HGG in three parts i.e. 210 into three sets of 70
    # for LGG
    data2,gt2 = load_data(path2)  #LGG having 75 patients

    100%|██████████| 75/75 [03:31<00:00,  2.81s/it]
```

```
[ ] np.save('/content/drive/My Drive/Brain_Tumor/Data/data2.npy',data2)
    np.save('/content/drive/My Drive/Brain_Tumor/Data/gt2.npy',gt2)
```

```
[ ] import numpy as np
    from tqdm import tqdm
```

```
[ ] # data = np.load('/content/drive/My Drive/Brain_Tumor/Data/data2.npy')
    # gt = np.load('/content/drive/My Drive/Brain_Tumor/Data/gt2.npy')

    data = data2
    gt = gt2
```

```
[ ] data = np.transpose(data,(0,2,3,4,1))
    data.shape,gt.shape,data.dtype,gt.dtype

    ((75, 155, 240, 240, 4), (75, 155, 240, 240), dtype('float32'), dtype('uint8'))
```

Fig 4.Data extraction and reshaping

5. Training & Validation datasets preparation

In this step we divide the datasets into training and validation datasets where we use 80/20 for splitting the training datasets and 75/25 for splitting the validation datasets and after printed the shape of the images. Here we used TensorFlow as backend.

```
[ ] data.shape,gt.shape

    ((75, 155, 240, 240, 4), (75, 155, 240, 240))

[ ] data = data[:,30:120,30:222,30:222,:].reshape([-1,192,192,4])
    gt = gt[:,30:120,30:222,30:222].reshape([-1,192,192,1])

[ ] data.shape,gt.shape

    ((6750, 192, 192, 4), (6750, 192, 192, 1))

[ ] gt[np.where(gt==4)]=3

[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, Y_train, Y_test = train_test_split(data, gt, test_size=0.20, random_state=42)
    X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.25,random_state=42)

[ ] %tensorflow_version 1.x

    TensorFlow 1.x selected.

[ ] from keras.utils import to_categorical,normalize
    Y_train = to_categorical(Y_train)
    Y_val = to_categorical(Y_val)
    X_train = (X_train-np.mean(X_train))/np.max(X_train)
    X_test = (X_test-np.mean(X_test))/np.max(X_test)
    X_val = (X_val-np.mean(X_val))/np.max(X_val)

    Using TensorFlow backend.

[ ] X_train.shape,Y_train.shape, X_val.shape, Y_val.shape, X_test.shape, Y_test.shape

    ((4050, 192, 192, 4),
     (4050, 192, 192, 4),
     (1350, 192, 192, 4),
     (1350, 192, 192, 4),
     (1350, 192, 192, 4),
     (1350, 192, 192, 1))

[ ] X_train.shape,Y_train.shape,X_val.shape,Y_val.shape

    ((4050, 192, 192, 4),
     (4050, 192, 192, 4),
     (1350, 192, 192, 4),
     (1350, 192, 192, 4))
```

Fig 5.Training & Validatoion datasets preparation

6. Important Packages for Implementation

In this stage we had imported all the necessary library packages required to execute the model which include different layers of CNN library required for training and complying purpose from keras application.

```
[ ] %tensorflow_version 1.x
    from keras.models import Model
    from keras.layers import MaxPooling2D,Conv2D,Dense,BatchNormalization,concatenate,Input,Dropout,Maximum,Activation,Dense,Flatten,UpSampling2D,Conv2DTranspose
    from keras.optimizers import SGD,Adam,RMSprop
    import keras.callbacks as callbacks
    import keras.initializers as initializers
    from keras.callbacks import Callback
    from keras import regularizers
```

Fig 6. Keras application packages required for exection.

7. Alternate Methodology Used – UNET

Below is the architecture of the Unet which we used as an alternate methodology which has one encoder at input and two decoders at output for segmentation of the brain images, but the drawback was it took around 9 hours for training the model using GPU.

```python
############################################ ALternate Methodology- U-net ############################################

############ Encoder #############
input_ = Input(shape=(192,192,4),name='input')

block1_conv1 = Conv2D(64,3,padding='same',activation='relu',name='block1_conv1')(input_)
block1_conv2 = Conv2D(64,3,padding='same',activation='relu',name='block1_conv2')(block1_conv1)
block1_norm = BatchNormalization(name='block1_batch_norm')(block1_conv2)
block1_pool = MaxPooling2D(name='block1_pool')(block1_norm)

block2_conv1 = Conv2D(128,3,padding='same',activation='relu',name='block2_conv1')(block1_pool)
block2_conv2 = Conv2D(128,3,padding='same',activation='relu',name='block2_conv2')(block2_conv1)
block2_norm = BatchNormalization(name='block2_batch_norm')(block2_conv2)
block2_pool = MaxPooling2D(name='block2_pool')(block2_norm)

encoder_dropout_1 = Dropout(0.2,name='encoder_dropout_1')(block2_pool)

block3_conv1 = Conv2D(256,3,padding='same',activation='relu',name='block3_conv1')(encoder_dropout_1)
block3_conv2 = Conv2D(256,3,padding='same',activation='relu',name='block3_conv2')(block3_conv1)
block3_norm = BatchNormalization(name='block3_batch_norm')(block3_conv2)
block3_pool = MaxPooling2D(name='block3_pool')(block3_norm)

block4_conv1 = Conv2D(512,3,padding='same',activation='relu',name='block4_conv1')(block3_pool)
block4_conv2 = Conv2D(512,3,padding='same',activation='relu',name='block4_conv2')(block4_conv1)
block4_norm = BatchNormalization(name='block4_batch_norm')(block4_conv2)
block4_pool = MaxPooling2D(name='block4_pool')(block4_norm)
################### Encoder end ######################

block5_conv1 = Conv2D(1024,3,padding='same',activation='relu',name='block5_conv1')(block4_pool)
# encoder_dropout_2 = Dropout(0.2,name='encoder_dropout_2')(block5_conv1)

########### Decoder ################

up_pool1 = Conv2DTranspose(1024,3,strides = (2, 2),padding='same',activation='relu',name='up_pool1')(block5_conv1)
merged_block1 = concatenate([block4_norm,up_pool1],name='merged_block1')
decod_block1_conv1 = Conv2D(512,3, padding = 'same', activation='relu',name='decod_block1_conv1')(merged_block1)

up_pool2 = Conv2DTranspose(512,3,strides = (2, 2),padding='same',activation='relu',name='up_pool2')(decod_block1_conv1)
merged_block2 = concatenate([block3_norm,up_pool2],name='merged_block2')
decod_block2_conv1 = Conv2D(256,3,padding = 'same',activation='relu',name='decod_block2_conv1')(merged_block2)

decoder_dropout_1 = Dropout(0.2,name='decoder_dropout_1')(decod_block2_conv1)

up_pool3 = Conv2DTranspose(256,3,strides = (2, 2),padding='same',activation='relu',name='up_pool3')(decoder_dropout_1)
merged_block3 = concatenate([block2_norm,up_pool3],name='merged_block3')
decod_block3_conv1 = Conv2D(128,3,padding = 'same',activation='relu',name='decod_block3_conv1')(merged_block3)

up_pool4 = Conv2DTranspose(128,3,strides = (2, 2),padding='same',activation='relu',name='up_pool4')(decod_block3_conv1)
merged_block4 = concatenate([block1_norm,up_pool4],name='merged_block4')
decod_block4_conv1 = Conv2D(64,3,padding = 'same',activation='relu',name='decod_block4_conv1')(merged_block4)
############ Decoder End ######################################

# decoder_dropout_2 = Dropout(0.2,name='decoder_dropout_2')(decod_block4_conv1)
```

Fig 7.Alternate methology used for segementing using Unet Model.

8. Model Implemented

Here we have implemented our core CNN model (3 variant architecture) using the spars and dense architecture

Below diagram shows the model architecture used for the implementation which was discussed in brief in the report. Here the spare module consist of 6 different layers of operation and Dense module consist of different Convolutional layers of different size with two different activation function and concatenation at the output.

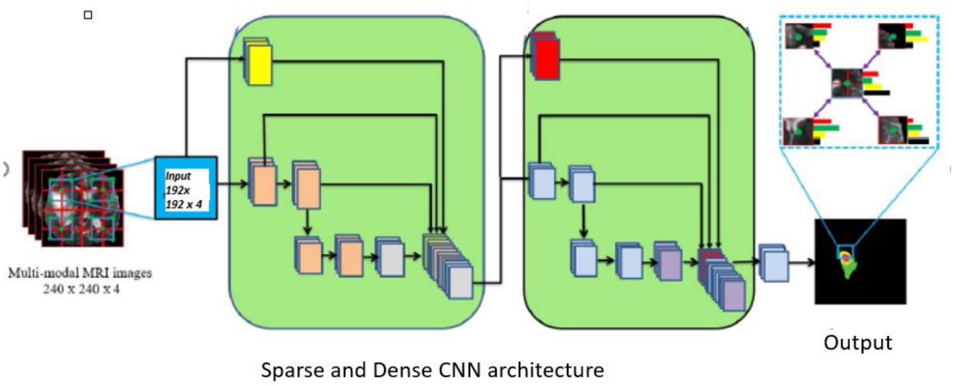


Fig 1.Sparse and Dense module CNN architecture

```
[ ] ########################################## Core Methodology - 3 CNN m
def build_model_SparseMultiOCM(input_shape=(192,192,4), kernel_size=(3,

        """
        Parameters
        ----------
        input_shape: the input of the model
            Shape: tuple of 3 numbers
            Default size: (64, 64, 4)
        kernel_size: kernel of the model
            Shape: tuple of 2 numbers
            Default size: (3, 3)
        load_model_weights: the stored parameters
             Default: None
        nb_classes: The number of classes
            Default: 4
        Returns
        -------
        model: an instance of the created keras model.
        """

        inputs= Input(shape=input_shape)

        x = Conv2D(
            filters=42,
            kernel_size=kernel_size,
            strides=1,
            padding='same',
            activation='relu')(inputs)

        conv1 = Conv2D(
            filters= 42,
            kernel_size=kernel_size,
            strides=1,
            padding='same',
            activation='relu')(x)

        x = MaxPooling2D(
            pool_size=(2,2))(conv1)

        x = Conv2D(
            filters=32,
            kernel_size=kernel_size,
            strides=1,
            padding='same',
            activation='relu')(x)

        x = Conv2D(
            filters=32,
            kernel_size=kernel_size,
            strides=1,
            padding='same',
            activation='relu')(x)

        x = UpSampling2D(
```

```
def build_model_InputSparseMultiOCM(input_shape=(192,192,4), kernel_s

        """
        Parameters
        ----------
        input_shape: the input of the model
            Shape: tuple of 3 numbers
            Default size: (64, 64, 4)
        kernel_size: kernel of the model
            Shape: tuple of 2 numbers
            Default size: (3, 3)
        load_model_weights: the stored parameters
             Default: None
        nb_classes: The number of classes
            Default:4
        Returns
        -------
        model: an instance of the created keras model.
        """

        inputs= Input(shape=input_shape)

        x = Conv2D(
            filters= 42,
            kernel_size=kernel_size,
            strides=1,
            padding='same',
            activation='relu')(inputs)

        conv1 = Conv2D(
            filters= 42,
            kernel_size=kernel_size,
            strides=1,
            padding='same',
            activation='relu')(x)

        x = MaxPooling2D(
            pool_size=(2,2))(conv1)

        x = Conv2D(
            filters=32,
            kernel_size=kernel_size,
            strides=1,
            padding='same',
            activation='relu')(x)

        x = Conv2D(
            filters=32,
            kernel_size=kernel_size,
            strides=1,
            padding='same',
            activation='relu')(x)

        x = UpSampling2D(
            size=(2,2))(x)
```

```
def build_model_DenseMultiOCM(input_shape=(192,192,4), kernel_size=(3, 3), load_model_weigh

        """
        Parameters
        ----------
        input_shape: the input of the model
            Shape: tuple of 3 numbers
            Default size: (64, 64, 4)
        kernel_size: kernel of the model
            Shape: tuple of 2 numbers
            Default size: (3, 3)
        load_model_weights: the stored parameters
             Default: None
        nb_classes: The number of classes
            Default: 4
        Returns
        -------
        model: an instance of the created keras model
        """

        inputs= Input(shape=input_shape)

        conv1 = Conv2D(
            filters= 32,
            kernel_size=(1, 1),
            strides=1,
            padding='same',
            activation='relu')(inputs)

        conv2 = Conv2D(
            filters= 32,
            kernel_size=kernel_size,
            strides=1,
            padding='same',
            activation='relu')(inputs)

        conv2 = Conv2D(
            filters= 32,
            kernel_size=kernel_size,
            strides=1,
            padding='same',
            activation='relu')(conv2)

        conv3 = Conv2D(
            filters= 32,
            kernel_size=kernel_size,
            strides=1,
            padding='same',
            activation='relu')(conv2)

        conv3 = Conv2D(
            filters= 32,
            kernel_size=kernel_size,
```

Fig 8.Three CNN model made using sparse and dense module CNN

10.Image Pre-processing

Here we have use three different steps for image pre-procesing which include image normalization, standardaization and agumentaion process along with isolation of healty region from the image with tumorus region.

```python
def image_preprocessing(self, slices, size=[4, 155, 240, 240], thresh=-9):

     """
    Parameters
    ----------
    slices: the MRI image's slices
    size : is the size of slices(nb_nb_channels, nb_slices, height, width)
       Default:[4, 155, 240, 240]
    thresh: a threshold to isolate the background pixels from the mean of intensities
       Default:-9

    Returns
    -------
    norm_slices: a normalized MRI sequences
    """

     norm_slices = np.zeros((size[0], size[1], size[2], size[3]))

     for channel in range(size[0]):

        for slice in range(size[1]):

            norm_slices[channel][slice] = self.image_normalization(slices[channel][slice], thresh)

     return norm_slices
def image_normalization(self, slices, thresh, noise_cap=1.0):

     """
    Parameters
    ----------
    slices: the MRI image's slices
    thresh: a threshold to isolate the background pixels from the mean of intensities
    noise_cap: a percent of noise that will be removed
       Default: 1.0

    Returns
    -------
    norm_slices: a normalized slice
    """

     slices = np.clip(slices, np.percentile(slices, noise_cap), np.percentile(slices, noise_cap))

     img = slices[np.nonzero(slices)]

     img= (slices - np.mean(img))/ np.std(img)
     del slices

     img[img==img.min()]= thresh

     return img
```

Fig 9.Image preporcessing stage

11. Post processing.

Here we performed two post processing operations first by removing the small connected region form the image and second is the morphological operations on the image.

```python
def connected_component(self, pred_image, nb_slices, thresh=110):

     """
    Parameters
    ----------
    pred_image: a predicted image by a trained model
    nb_slices:  the number of slices in a image (the depth)
    thresh: a threshold to determine of connected components
            that we want to remove
      Default: 100

    Returns
    -------
    pred_image: a predicted image without some regions
    """

     for k in range (nb_slices):

        img = pred_image[k]
        img = img.astype(np.uint8)

        nb_components, output, stats, centroids = cv2.connectedComponentsWithStats(img, connectivity=8)

        sizes = stats[1:, -1]
        del stats, centroids
        nb_components = nb_components - 1
        min_size = thresh

        tmp_img2 = np.zeros((output.shape))
        for i in range(0, nb_components):
                if sizes[i] >= min_size:
                    tmp_img2[output == i + 1] = 255

        del nb_components, min_size, sizes
        tmp_img3 = np.zeros((output.shape))
        del output
        for i in range((img.shape[0])):
                for j in range((img.shape[0])):
                    if(tmp_img2[i][j] > 0):
                        tmp_img3[i][j] = img[i][j]

        pred_image[k] = tmp_img3
        del tmp_img2, tmp_img3


     return pred_image
```

Fig 10.Image post processing stage

12. Model Trained

Here we have selected compiler and loss funtion for our training model and check pointer along with class weighted approach for imbalanced problem.

```python
from keras import backend as K
def dice_coef(y_true, y_pred, epsilon=1e-6):
    intersection = K.sum(K.abs(y_true * y_pred), axis=-1)
    return (2. * intersection) / (K.sum(K.square(y_true),axis=-1) + K.sum(K.square(y_pred),axis=-1) + epsilon)

def dice_coef_loss(y_true, y_pred):
    return 1-dice_coef(y_true, y_pred)
```

```python
model.compile(optimizer=Adam(lr=1e-5),loss=dice_coef_loss,metrics=[dice_coef])
#model.load_weights('./Model Checkpoints/weights.hdf5')
checkpointer = callbacks.ModelCheckpoint(filepath = '/content/drive/My Drive/Brain_Tumor/Model_Checkpoints/weights.hdf5',save_best_only=True)
training_log = callbacks.TensorBoard(log_dir='/content/drive/My Drive/Brain_Tumor/Model_Checkpoints')
```

```python
class_weighting= [0.28, 0.08, 0.43, 0.21]
Init_train = Training(history, 8, 1, class_weighting)
history = model.fit(X_train,Y_train,validation_data=(X_val,Y_val),batch_size=16,epochs=16,callbacks=[training_log,checkpointer],shuffle=True)
```

```
Train on 4050 samples, validate on 1350 samples
Epoch 1/16
4050/4050 [==============================] - 1830s 452ms/step - loss: 0.0326 - dice_coef: 0.9674 - val_loss: 0.0348 - val_dice_coef: 0.9652
Epoch 2/16
4050/4050 [==============================] - 1773s 438ms/step - loss: 0.0300 - dice_coef: 0.9700 - val_loss: 0.0294 - val_dice_coef: 0.9706
Epoch 3/16
4050/4050 [==============================] - 1863s 460ms/step - loss: 0.0245 - dice_coef: 0.9755 - val_loss: 0.0284 - val_dice_coef: 0.9716
Epoch 4/16
4050/4050 [==============================] - 1806s 446ms/step - loss: 0.0205 - dice_coef: 0.9795 - val_loss: 0.0199 - val_dice_coef: 0.9801
Epoch 5/16
4050/4050 [==============================] - 1809s 447ms/step - loss: 0.0179 - dice_coef: 0.9821 - val_loss: 0.0180 - val_dice_coef: 0.9820
Epoch 6/16
4050/4050 [==============================] - 1764s 436ms/step - loss: 0.0163 - dice_coef: 0.9837 - val_loss: 0.0169 - val_dice_coef: 0.9831
Epoch 7/16
4050/4050 [==============================] - 1745s 431ms/step - loss: 0.0151 - dice_coef: 0.9849 - val_loss: 0.0152 - val_dice_coef: 0.9848
Epoch 8/16
4050/4050 [==============================] - 1811s 447ms/step - loss: 0.0141 - dice_coef: 0.9859 - val_loss: 0.0142 - val_dice_coef: 0.9858
Epoch 9/16
4050/4050 [==============================] - 1780s 439ms/step - loss: 0.0131 - dice_coef: 0.9869 - val_loss: 0.0132 - val_dice_coef: 0.9868
Epoch 10/16
4050/4050 [==============================] - 1799s 444ms/step - loss: 0.0124 - dice_coef: 0.9876 - val_loss: 0.0120 - val_dice_coef: 0.9880
Epoch 11/16
4050/4050 [==============================] - 1769s 437ms/step - loss: 0.0117 - dice_coef: 0.9883 - val_loss: 0.0111 - val_dice_coef: 0.9889
Epoch 12/16
4050/4050 [==============================] - 1834s 453ms/step - loss: 0.0111 - dice_coef: 0.9889 - val_loss: 0.0112 - val_dice_coef: 0.9888
Epoch 13/16
4050/4050 [==============================] - 1923s 475ms/step - loss: 0.0104 - dice_coef: 0.9896 - val_loss: 0.0102 - val_dice_coef: 0.9898
Epoch 14/16
4050/4050 [==============================] - 1882s 465ms/step - loss: 0.0101 - dice_coef: 0.9899 - val_loss: 0.0101 - val_dice_coef: 0.9899
Epoch 15/16
4050/4050 [==============================] - 1875s 463ms/step - loss: 0.0096 - dice_coef: 0.9904 - val_loss: 0.0093 - val_dice_coef: 0.9907
Epoch 16/16
4050/4050 [==============================] - 1795s 443ms/step - loss: 0.0092 - dice_coef: 0.9908 - val_loss: 0.0092 - val_dice_coef: 0.9908
```

Fig 11.Training stage of model

13. Evaluation

The model were evaluted based on the dice and loss score for that we have used our saved trained model for estimating the validation loss and dice coefficent as stated on below code which includes the highest value at the 16th epcoch of the training step.

```
[ ] #model.save('/content/drive/My Drive/Brain_Tumor/Model_Checkpoints/model.h5')

[ ] # summarize history for loss
    import matplotlib.pyplot as plt
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('dice coef loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
    plt.plot(history.history['dice_coef'])
    plt.plot(history.history['val_dice_coef'])
    plt.title('model score')
    plt.ylabel('dice coef')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

Fig 12.Code for plotting loss and dice scores for model

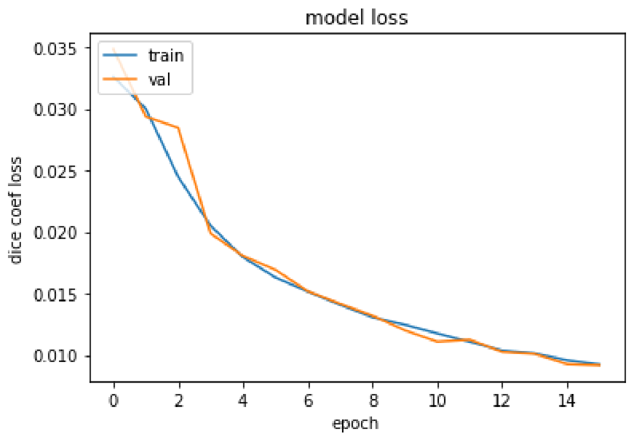


Fig 13.Graph of Model loss vs Epoch

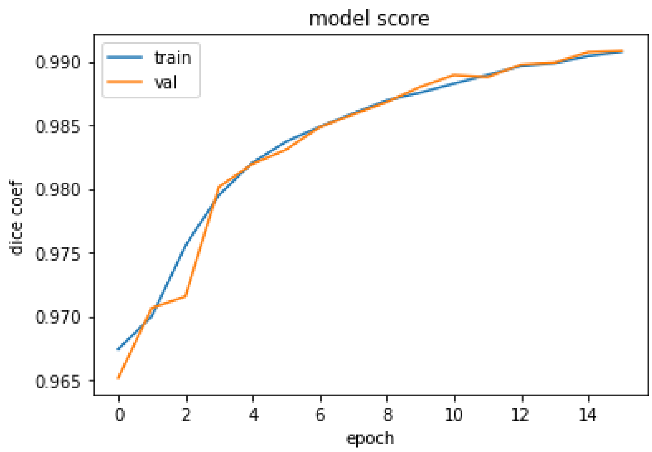


Fig 14. Graph of Model score vs Epoch

14. Prediction

Here we have done the preidction by testing it on brian tumor MRI images with low grade, high grade and images with no brain tumor.

```
[ ] # from keras.models import load_model
    # model = load_model('/content/drive/My Drive/Brain_Tumor/Model_Checkpoints/model.h5')

[ ] Y_pre = np.argmax(model.predict(X_test),axis=-1)

[ ] np.unique(Y_pre)

    array([0, 1, 2])

[ ] Y_pre.shape

    (1350, 192, 192)

[ ] Y_pre=Y_pre.reshape(-1,192,192,1)

[ ] Y_test.shape,X_test.shape

    ((1350, 192, 192, 1), (1350, 192, 192, 4))

[ ] for i in range(450,500):
      print('X_test '+ str(i))
      plt.imshow(X_test[i,:,:,2])
      plt.show()
      print("Actual")
      plt.imshow(Y_pre[i,:,:,0])
      plt.show()
      print("Predicted")
      plt.imshow(Y_test[i,:,:,0])
      plt.show()
      print("=================================================================")
```

Fig 15.Code for printing the prediction results

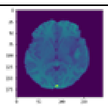
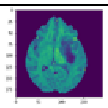
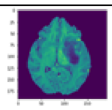
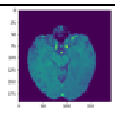
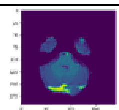
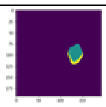
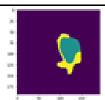
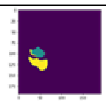
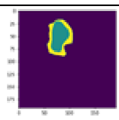
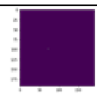
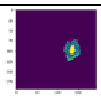
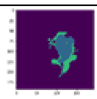
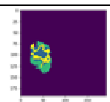
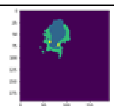
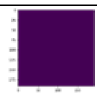
Parameter	Low Grade	High Grade	Low Grade	High Grade	No tumor
Test image					
Actual					
Predicted					

Table 2. Prediction resulst on low, high grade and brain images with no tumor.

15. References.

1. Axel, D, 2014. *Brain Tumor Segmentation with Deep Neural Networks*. In: Proceedings MICCAI-BRATS. Issue 2014, pp. 01-05.

2. https://arxiv.org/pdf/1505.03540.pdf

3. https://mymoodle.ncirl.ie/pluginfile.php/205095/mod_resource/content/0/Masters%20Research%20Project%20Handbook_2021-2022_Final.pdf