National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Diksha Arvind Chaudhary
Student ID: X18184898

School of Computing
National College of Ireland

Supervisor: Dr. Rashmi Gupta

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Diksha Arvind Chaudhary
Student ID:	X18184898
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Rashmi Gupta
Submission Due Date:	28/09/2020
Project Title:	Configuration Manual
Word Count:	2747
Page Count:	14

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:	Diksha Arvind Chaudhary
Date:	27th September 2020

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

Diksha Arvind Chaudhary
X18184898

1 Introduction

The purpose of this document is to provide details of the process followed during the project coding phase. Hardware and software configurations are specified to reproduce the research in the future. This contains the programming and deployment phases for smooth code execution and the steps to be taken to execute the code.

2 System Configuration

2.1 Hardware Configuration

The hardware specification details are given below in the Figure 1 on which the code is executed:

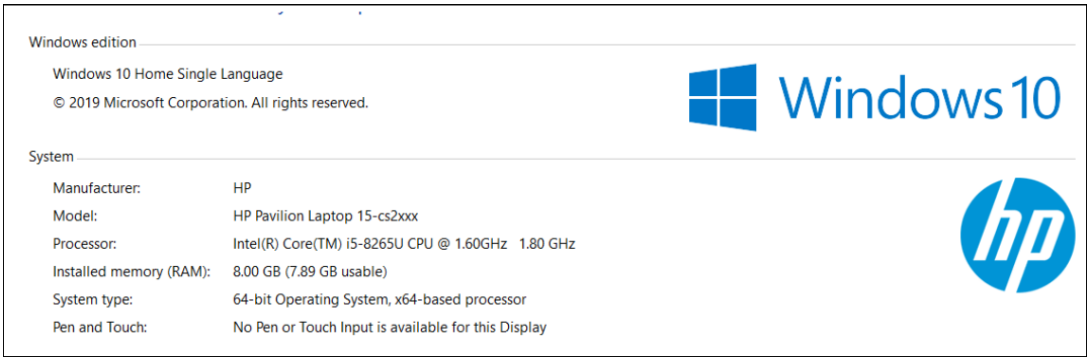


Figure 1: Hardware configuration of the system

2.2 Software Configuration

This section provides the details of the software and its specifications.

2.2.1 Google Colab:

The research is carried out using Google's cloud infrastructure, also known as Google Colab. All the libraries are imported and coding of the model is done in google colab. The dataset is stored on google drive and in google colab using below code the drive is mounted. The link to authorization is provided after execution of following command, if

we click on the link the authorization code is generated. The drive is mounted successfully after entering the authorisation code as shown in Figure 2.

```
from google.colab import drive
drive.mount('/content/drive')
```

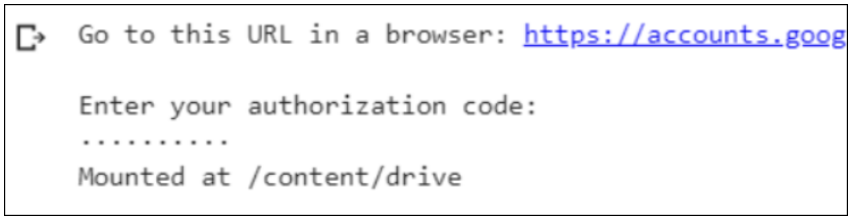


Figure 2: Mount drive on google colab

Colab notebook runtime is set to GPU for faster code execution in cloud environment. It's a setting given by Google Colab to allow us to run the machine learning code close to the Jupyter environment. There is another TPU option that can also be used but we use GPU setting for this project which is shown in Figure 3.

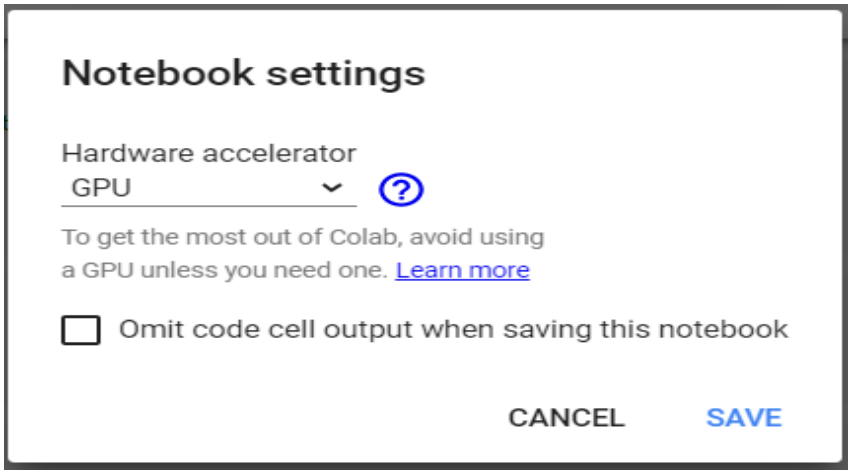


Figure 3: Colab Notebook setting

2.2.2 Anaconda - Jupyter Notebook:

Anaconda is an open source, easy to use platform for coding Python and R [1]. Jupyter is a user-friendly Integrated Development Environment (IDE) that Anaconda provides for code development and results assessment. Anaconda can be downloaded from official website of anaconda [2]. The download options for different Operating systems is provided in Figure 4.

[1]https://www.anaconda.com/
[2]https://www.anaconda.com/products/individual

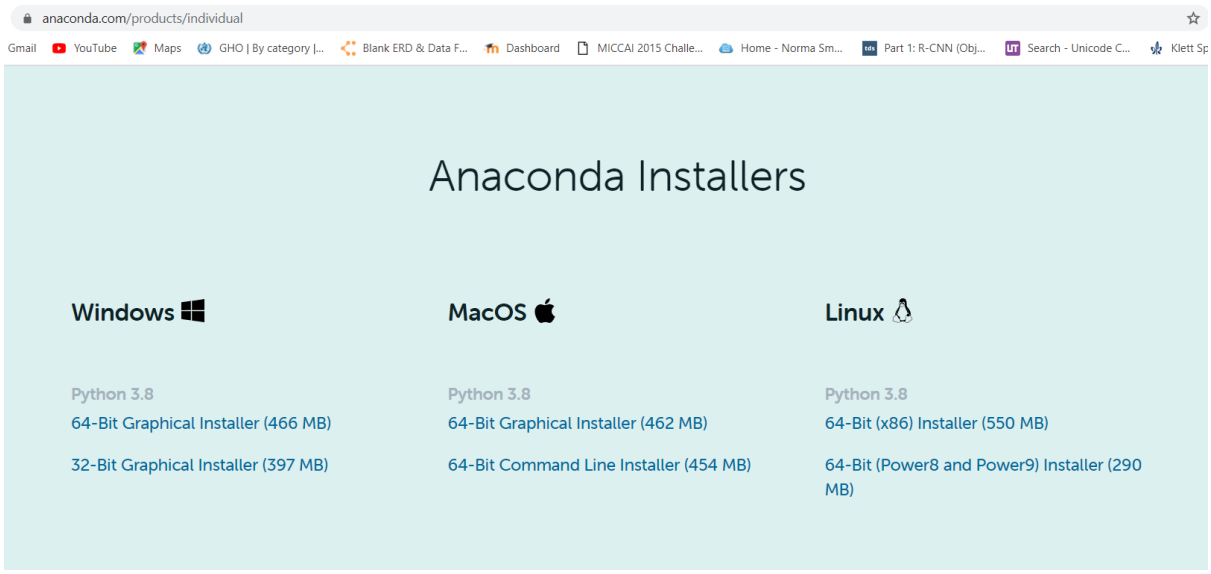


Figure 4: Anaconda Installer Download Page

Once the Anaconda is install, the prompt will be shown displaying different IDE Figure 5. Jupyter IDE is launched for code development of different models using Python version 3.

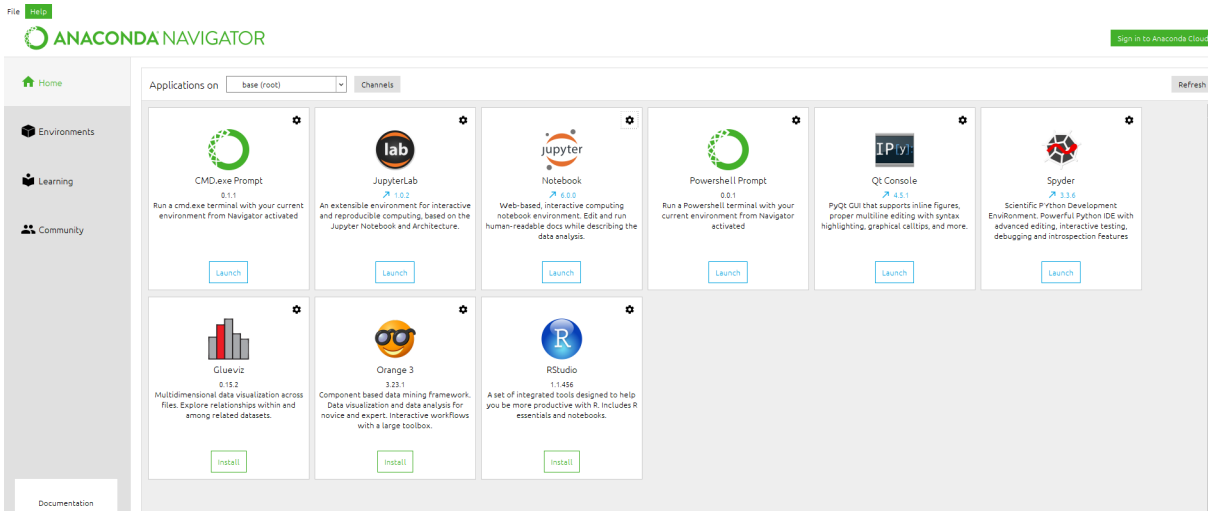


Figure 5: Anaconda Prompt Page

2.2.3 Other Softwares:

Google chrome is a good web browser that supports Jupyter and helps in code execution. Overleaf is used for research project documentation, Figure 6 shows the use of overleaf for the research project documentation.

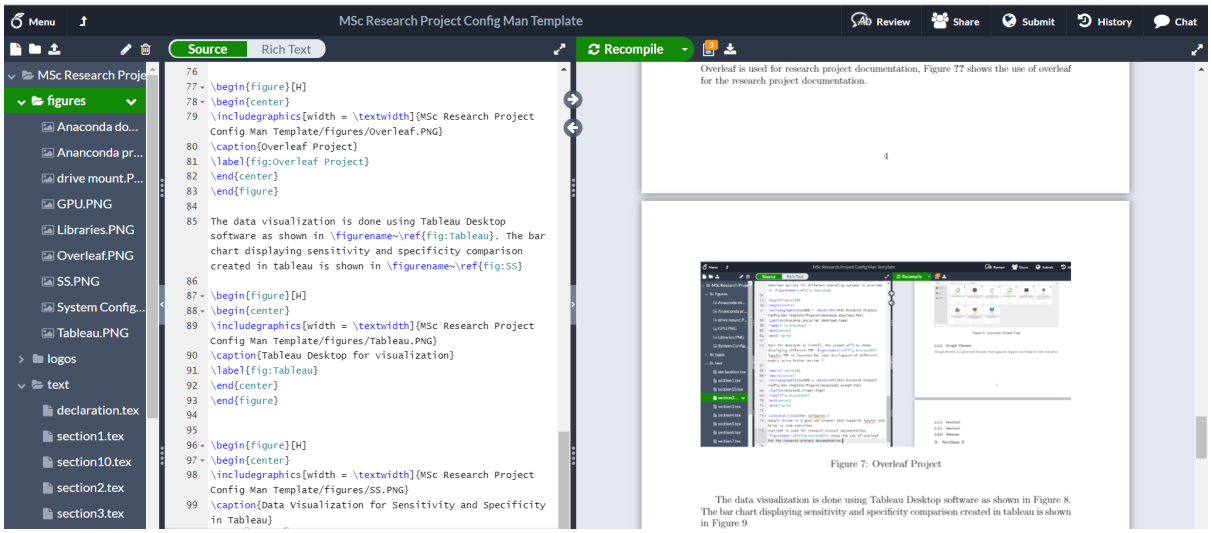


Figure 6: Overleaf Project

The data visualization is done using Tableau Desktop software as shown in Figure 7. The bar chart displaying sensitivity and specificity comparison created in tableau is shown in Figure 8

Figure 7: Tableau Desktop for visualization

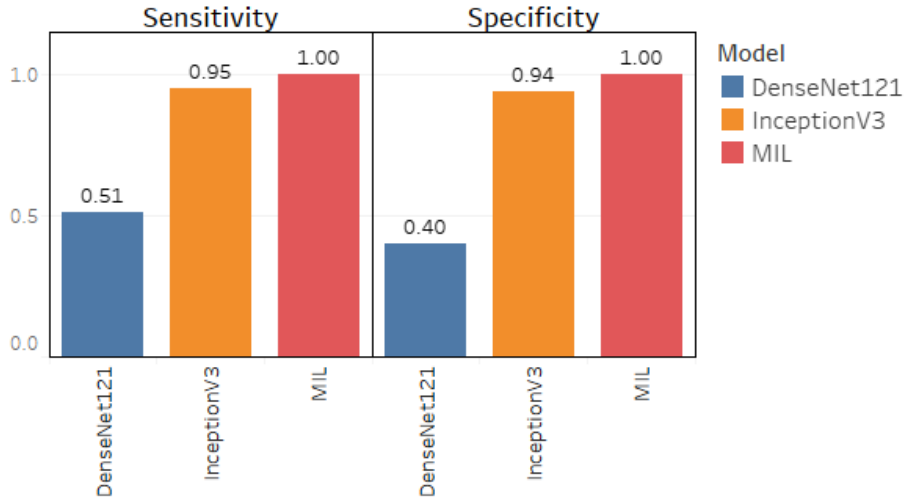


Figure 8: Data Visualization for Sensitivity and Specificity in Tableau

3 Data Preparation

The research dataset is taken from the challenge of Brain Tumor segmentation [3] shown in Figure 9. The .nii.gz files from BRATS are converted into jpg format using mathematical operation and provided on Kaggle which are further used in the project [4].

Figure 9: BRATS Dataset for research Project

The dataset contained one folder, Brain tumor and a csv file is provided containing image name and category (0-healthy and 1-tumor). The code for loading data into data-frame and checking the imbalance and code for separating the brain images to different category folder is shown below.

```
Loading Csv data into Pandas DataFrame
data_df = pd.read_csv('D:/Brain_New/Brain Tumor.csv')
Bias Check in Data
data_df_grp.plot(x = 'Class', y= 'Image', kind = 'bar', legend=True)
```

[3] http://braintumorsegmentation.org/
[4] https://www.kaggle.com/jakeshbohaju/brain-tumor

Code for Comparing the Class Category and Copying to Relevant Folder

```
src = 'D:/Brain_New/Brain Tumor/Brain Tumor/'
dest = 'D:/Brain_New/Brain_tumor_class/'

img_list = data_df["Class"]
for names in os.listdir(src):
        i = 0
                for i in range(len(img_list)):

                if names[:-4] == data_df.at[i,"Image"]:
                        cat = data_df.at[i,"Class"]
                        cate = str(cat)
                        shutil.copy2( src + names, dest + cate)
```

4 Data Transformation

After the data is pre-processed and folders are created, the folders are split into train, test and validation sets. The code for data transformation is shown below.

```
for cls in classes_dir:
        os.makedirs(root_dir +'train' + cls)
        os.makedirs(root_dir +'val' + cls)
        os.makedirs(root_dir +'test' + cls)

        src = root_dir + cls

        allFileNames = os.listdir(src)
        np.random.shuffle(allFileNames)
        train_FileNames,val_FileNames,test_FileNames= np.split(np.array(allFileNames),
                                                        [int(len(allFileNames)* (1 - val_ratio + test_ratio)),
                                                        int(len(allFileNames)* (1 - test_ratio))])

        train_FileNames = [src+'\\'+ name for name in train_FileNames.tolist()]
        val_FileNames = [src+'\\' + name for name in val_FileNames.tolist()]
        test_FileNames = [src+'\\' + name for name in test_FileNames.tolist()]

        for name in train_FileNames:
                shutil.copy(name, root_dir +'\\train' + cls)

        for name in val_FileNames:
                shutil.copy(name, root_dir +'\\val' + cls)

        for name in test_FileNames:
                shutil.copy(name, root_dir +'\\test' + cls)
```

The dataset was then uploaded to Google Drive by using the upload folder option as shown in Figure 10.

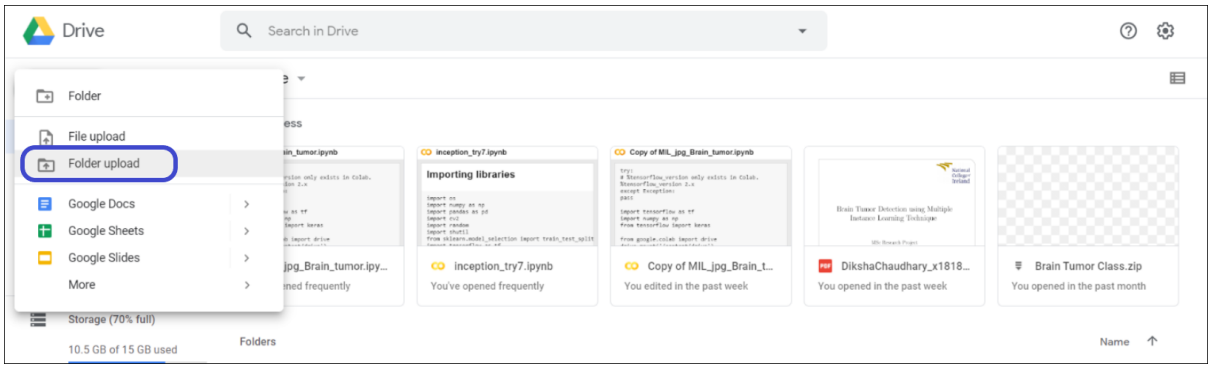


Figure 10: BRATS Dataset upload on Google Drive

5 Implementation of Baseline Models

The data can be used for model implementation after data preparation, using Transfer Learning based on pre-trained models. These datasets are used to implement DenseNet121

and InceptionV3.

5.1 DenseNet121

5.1.1 Model Building

The data is fed to ImageDataGenerator class of Keras, which consists of several functions for loading and performing real time augmentation [5]. The code in below shows the real time data augmentation performed during the model building phase of DenseNet121.

```
Model Building
#image size reduced to 64*64 and upsampled train data
train_gen=ImageDataGenerator(rotation_range=90,
                                    width_shift_range=0.1,
                                    height_shift_range=0.1,
                                    rescale = 1./255,
                                    shear_range = 0.2,
                                    zoom_range = 0.5,
                                    horizontal_flip = True,
                                    fill_mode="nearest")

train = train_gen.flow_from_directory("D:\\Brain_New\\Brain Tumor\\Brain Tumor Class\\train\\",
                                    class_mode="categorical",
                                    target_size=(64, 64),
                                    color_mode="rgb",
                                    shuffle=True,
                                    batch_size=32)
```

```
valid_gen=ImageDataGenerator(rotation_range=90,
                                    width_shift_range=0.1,
                                    height_shift_range=0.1,
                                    rescale = 1./255,
                                    shear_range = 0.2,
                                    zoom_range = 0.5,
                                    horizontal_flip = True,
                                    fill_mode="nearest")

valid = valid_gen.flow_from_directory("D:\\Brain_New\\Brain Tumor\\Brain Tumor Class\\val\\",
                                    class_mode="categorical",
                                    target_size=(64, 64),
                                    color_mode="rgb",
                                    shuffle=True,
                                    batch_size=32)
```

The model is trained using pretrained DenseNet121 model[6]. The weights are trained on ImageNet, the code is shown in below.

```
base_model = densenet.DenseNet121(input_shape=(64, 64, 3),
                                    weights='imagenet',
                                    include_top=False,
                                    pooling='avg')

for layer in base_model.layers:
        layer.trainable = True

x = base_model.output

predictions = Dense(2, activation='softmax')(x)
```

The hyper parameters used for the model are given here.

```
optimizer = Adam(lr=0.0001,beta_1=0.9,beta_2=0.999, epsilon=1e-08)
model.compile(loss='binary_crossentropy',optimizer='adam', metrics=['acc','mae'])
```

The model is executed using model.fit_generator for 100 epochs and the accuracy and mean absolute error can be seen for every epoch as shown in Figure 11.

[5]https://keras.io/api/preprocessing/image/
[6]https://keras.io/api/applications/densenet/

```
model_history = model.fit_generator(
        train,
        epochs=100,
        steps_per_epoch=95,
        validation_data=valid,
        validation_steps=18)
```

```
Epoch 1/100
95/95 [==============================] - 279s 3s/step - loss: 0.5384 - acc: 0.7932 - mean_absolute_error: 0.2492 - val_loss:
0.6023 - val_acc: 0.8741 - val_mean_absolute_error: 0.1391
Epoch 2/100
95/95 [==============================] - 286s 3s/step - loss: 0.2606 - acc: 0.8942 - mean_absolute_error: 0.1489 - val_loss:
1.4982 - val_acc: 0.7872 - val_mean_absolute_error: 0.2221
Epoch 3/100
95/95 [==============================] - 284s 3s/step - loss: 0.2260 - acc: 0.9067 - mean_absolute_error: 0.1321 - val_loss:
0.8179 - val_acc: 0.8741 - val_mean_absolute_error: 0.1307
Epoch 4/100
```

Figure 11: Code Execution

5.1.2 Model Evaluation

Similar to train and valid, ImageDataGenerator is used for loading the test data and perform the same real time augmentation on the test data. The code is show in below. Further model.evaluate_generator is used for evaluation of test data and accuracy and loss is obtained and code can be seen below.

```
test_gen=ImageDataGenerator(rotation_range=90,
                            width_shift_range=0.1,
                            height_shift_range=0.1,
                            rescale = 1./255,
                            shear_range = 0.2,
                            zoom_range = 0.5,
                            horizontal_flip = True,
                            fill_mode="nearest")

test = test_gen.flow_from_directory("D:\Brain_New\Brain Tumor\Brain Tumor Class\Test\",
                            class_mode="categorical",
                            target_size=(64, 64),
                            color_mode="rgb",
                            shuffle=True,
                            batch_size=32)

test_acc = model.evaluate_generator(
                            test,
                            steps=len(test),
                            verbose=1
)
```

The confusion matrix is generated for the model and further the most important metrics for medical imaging i.e., Sensitivity and Specificity is calculated. The calculation for these metrics is shown in the code in Figure 12

```
print('sensitivity', cm[0, 0] \(cm[0, 1] + cm[0, 0])) print('specificity', cm[1, 1] \(cm[1, 1] + cm[1, 0]))
```

```
sensitivity 0.5192307692307693
specificity 0.4
```

Figure 12: Evaluation Metrics - Sensitivity and Specificity

Further, confusion matrix and classification report is generated using sklearn.metrics [7]. The classification report shows the evaluation metrics like Precision, Recall and F1-score. The code is given below and Figure 12 shows the output of classification model.

[7]https://scikit-learn.org/stable/modules/model_evaluation.html

```
Y_pred = model.predict_generator(test, steps = 6)
y_pred = np.argmax(Y_pred, axis=1)
print('Confusion Matrix')
print(confusion_matrix(test.classes, y_pred))
print('Classification Report')
target_names = ['0', '1']
print(classification_report(test.classes, y_pred, target_names=target_names))
```

```
Confusion Matrix
[[56 48]
 [51 34]]
Classification Report
              precision    recall  f1-score   support

           0       0.52      0.54      0.53       104
           1       0.41      0.40      0.41        85

    accuracy                           0.48       189
   macro avg       0.47      0.47      0.47       189
weighted avg       0.47      0.48      0.48       189
```

Figure 13: Classification Report - Precision, Recall and F1-score

5.2 InceptionV3

5.2.1 Model Building

InceptionV3 follows the same steps that are provided for DenseNet121 of loading the data using ImageDataGenerator with real time augmentation. The model is pre-trained on InceptionV3 [8] and the code is shown below.

```
pre_trained_model = InceptionV3(input_shape=(200, 200, 3),
                                include_top=False,
                                weights='imagenet')
```

The code for compiling the model is provided in here. The model uses learning rate 0.0001 and adam optimizer. The loss function used is binary crossentropy and the metrics for evaluation are provided in model.compile.

```
adam = Adam(lr=0.0001)
model.compile(loss='binary_crossentropy',optimizer=adam, metrics=['accuracy','mae'])
```

The model is execute using model.fit_generator class of Keras function and the code is provided below. Every epoch runs for 94 steps and accuracy, loss and mean average error can be seen for every epoch in Figure 14.

```
train_step = trainGenerator.n\\trainGenerator.batch_size
val_step = validGenerator.n\\validGenerator.batch_size

history = model.fit_generator(trainGenerator,
                              steps_per_epoch=train_step,
                              validation_data=validGenerator,
                              validation_steps=val_step,
                              epochs=100,callbacks=[log, mc])
```

[8] https://keras.io/api/applications/inceptionv3/

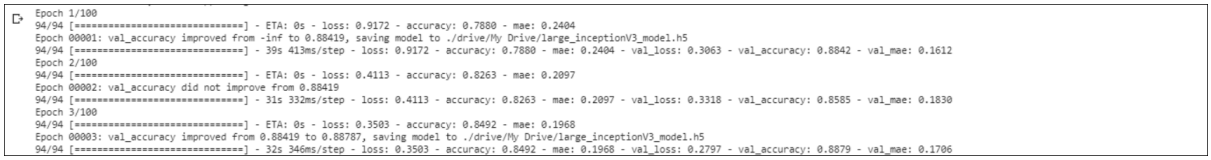


Figure 14: Model Execution

5.2.2 Model Evaluation

Once the model is executed successfully for 100 epochs then test accuracy for InceptionV3 is evaluated using model.evaluate_generator function. The code for evaluation of test accuracy is shown below and the output can be seen in Figure 15.

```
 test_pred = model.evaluate_generator(test_gen,verbose=1)
print("Testing Categorical Accuracy :"+str(test_pred[1])+ "Testing loss :  "+str(test_pred[0]))
```

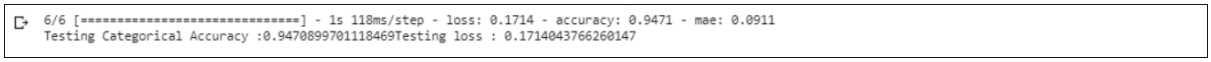
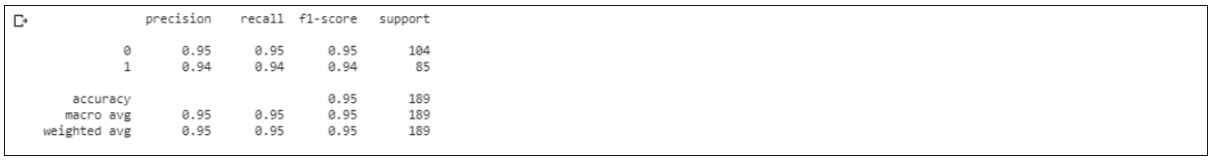


Figure 15: Evaluating the Test Accuracy

The classification report is generated in the similar way as explained for DenseNet121. The code for classification report is provided and output can be seen in Figure 16

```
from sklearn.metrics import classification_report

target_names = ['0', '1']
print(classification_report(test_gen.classes, pred, target_names=target_names))
```



	precision	recall	f1-score	support
0	0.95	0.95	0.95	104
1	0.94	0.94	0.94	85
accuracy			0.95	189
macro avg	0.95	0.95	0.95	189
weighted avg	0.95	0.95	0.95	189

Figure 16: Classification Report - Precision, Recall and F1-score

The two main metrics Sensitivity and Specificity are calculated in following manner and result is shown in Figure 17.

```
print('sensitivity', cm[0, 0] \(cm[0, 1] + cm[0, 0])) print('specificity', cm[1, 1] \(cm[1, 1] + cm[1, 0]))
```

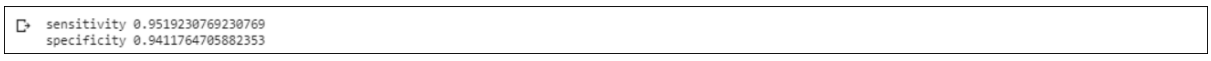


Figure 17: Evaluation Metrics - Sensitivity and Specificity

6 Implementation of Newly Proposed Model - Multiple Instance Learning

Multiple instance learning (MIL) is novelty of the research project. The MIL is never used for the detection of Brain tumors. The approach is capable of handling data weakly supervised by using the multiple instances concept. Different functions are created in the implementation of MIL. The code of MIL is based on Attention based mechanism and the code is referred from Ilse et al. (2018); Wang et al. (2018) [9].

The data is loaded as positive and negative paths and then the data is split using k-fold split as shown below.

```
import numpy as np
import glob
import KFold

def load_dataset(dataset_path, n_folds, rand_state):
        pos_path = glob.glob(dataset_path+'\0\Ima*')
        neg_path = glob.glob(dataset_path+'\1\Ima*')

        pos_num = len(pos_path)
        neg_num = len(neg_path)

        all_path = pos_path + neg_path

        kf = KFold(n_splits=n_folds, shuffle=True, random_state=rand_state)
        datasets = []
        for train_idx, test_idx in kf.split(all_path):
                dataset =
                dataset['train'] = [all_path[ibag] for ibag in train_idx]
                dataset['test'] = [all_path[ibag] for ibag in test_idx]
                datasets.append(dataset)
        return datasets
```

The evaluation metrics used are bag accuracy and bag loss. The calculation for bag accuracy and bag loss written in function bag_accuracy and bag_loss as shown below.

```
def bag_accuracy(y_true, y_pred):
        y_true = K.mean(y_true, axis=0, keepdims=False)
        y_pred = K.mean(y_pred, axis=0, keepdims=False)
        acc = K.mean(K.equal(y_true, K.round(y_pred)))

        return acc
def bag_loss(y_true, y_pred):
        y_true = K.mean(y_true, axis=0, keepdims=False)
        y_pred = K.mean(y_pred, axis=0, keepdims=False)
        loss = K.mean(K.binary_crossentropy(y_true, y_pred), axis=-1)
        return loss

def cmat(y_true, y_pred):
        y_true = K.mean(y_true, axis=0, keepdims=False)
        y_pred = K.mean(y_pred, axis=0, keepdims=False)
        cm = keras.metrics.confusion_matrix(y_test, y_pred)
        return cm
```

The augmentation techniques used are provided in this section. The section shows the function used for random flip operation on images and random rotate using cv2 library functions as shown below.

[9] https://github.com/utayao/Atten_Deep_MIL

```
def random_flip_img(img, horizontal_chance=0, vertical_chance=0):
        flip_horizontal = False
        if random.random() < horizontal_chance:
                flip_horizontal = True

        flip_vertical = False
        if random.random() < vertical_chance:
                flip_vertical = True

        if not flip_horizontal and not flip_vertical:
                return img

        flip_val = 1
        if flip_vertical:
                flip_val = -1 if flip_horizontal else 0

        if not isinstance(img, list):
                res = cv2.flip(img, flip_val)  0 = X axis, 1 = Y axis, -1 = both
        else:
                res = []
                for img_item in img:
                        img_flip = cv2.flip(img_item, flip_val)
                        res.append(img_flip)
        return res

def random_rotate_img(images):
        rand_roat = np.random.randint(4, size=1)
        angle = 90*rand_roat
        center = (images.shape[0] / 2, images.shape[1] / 2)
        rot_matrix = cv2.getRotationMatrix2D(center, angle[0], scale=1.0)

        img_inst = cv2.warpAffine(images, rot_matrix, dsize=images.shape[:2], borderMode=cv2.BORDER_CONSTANT)

        return img_inst
```

The custom layers have been implemented by combination of convolutional layers and fully connected layers as shown in below code. The layers contains hyper parameters passed to the model and metrics that are needed to be calculated during every epochs.

```
def cell_net(input_dim, useMulGpu=False):


        lr = 1e-2
        weight_decay = 0.005
        momentum = 0.9

        data_input = Input(shape=input_dim, dtype='float32', name='input')
        conv1 = Conv2D(36, kernel_size=(4,4), kernel_regularizer=l2(weight_decay), activation='relu')(data_input)
        conv1 = MaxPooling2D((2,2))(conv1)

        conv2 = Conv2D(48, kernel_size=(3,3), kernel_regularizer=l2(weight_decay), activation='relu')(conv1)
        conv2 = MaxPooling2D((2,2))(conv2)
        x = Flatten()(conv2)

        fc1 = Dense(512, activation='relu',kernel_regularizer=l2(weight_decay), name='fc1')(x)
        fc1 = Dropout(0.5)(fc1)
        fc2 = Dense(512, activation='relu', kernel_regularizer=l2(weight_decay), name='fc2')(fc1)
        fc2 = Dropout(0.5)(fc2)

        alpha = Mil_Attention(L_dim=128, output_dim=1, kernel_regularizer=l2(weight_decay), name='alpha',
                use_gated=False)(fc2)
        x_mul = multiply([alpha, fc2])

        out = Last_Sigmoid(output_dim=1, name='FC1_sigmoid')(x_mul)

        model = Model(inputs=[data_input], outputs=[out])

        if useMulGpu == True:
                parallel_model = multi_gpu_model(model, gpus=2)
                parallel_model.compile(optimizer=Adam(lr=lr, beta_1=0.9, beta_2=0.999), loss=bag_loss,
                metrics=[bag_accuracy,
                tf.keras.metrics.TruePositives(),tf.keras.metrics.FalsePositives(), tf.keras.metrics.FalseNegatives()])
        else:
                model.compile(optimizer=Adam(lr=lr, beta_1=0.9, beta_2=0.999), loss=bag_loss, metrics=[bag_accuracy,
                bag_loss], tf.keras.metrics.TruePositives(), tf.keras.metrics.FalsePositives(),
                tf.keras.metrics.FalseNegatives()])
                parallel_model = model

        return parallel_model
```

The generate batch function is used to take the input path from load dataset and create batches for train and test bags using respective data path. The images in the bag are appended with the label.

The following function is a function for evaluation of train data. Keras fit_generators functions are used for training the model on train data. Further, Train loss, train acc,

val loss and val accuracy is calculated as shown below.

The test eval function takes the test set and model as input and calculates the test loss and test accuracy of the model as shown in the code below.

```
def test_eval(model, test_set, Train_set):
        num_test_batch = len(test_set)
        test_loss = np.zeros((num_test_batch, 1), dtype=float)
        test_acc = np.zeros((num_test_batch, 1), dtype=float)
        for ibatch, batch in enumerate(test_set):
                result = model.test_on_batch(x=batch[0], y=batch[1])
                test_loss[ibatch] = result[0]
                test_acc[ibatch] = result[1]
                return np.mean(test_loss), np.mean(test_acc)
```

The training of the model is carried out by the following function shown below. All the functions mentioned above are called in this function. The final test accuracy is returned by the model training function.

```
def model_training(input_dim, dataset, irun, ifold):

        train_bags = dataset['train']
        test_bags = dataset['test']

        train_set = generate_batch(train_bags)
        test_set = generate_batch(test_bags)

        model = cell_net(input_dim, useMulGpu=False)
        print(model.history)

        t1 = time.time()
        num_batch = len(train_set)
        model_name = train_eval(model, train_set, irun, ifold, test_set)


        print("load saved model weights")
        model.load_weights(model_name)

        test_loss, test_acc = test_eval(model, test_set,train_set)

        t2 = time.time()

        print ('run time:', (t2 - t1) / 60.0, 'min')
        print ('test_acc=:.3f'.format(test_acc))

        return test_acc
```

The main function is shown in the below. The model is executed and the evaluation plots are generated after successful execution of the main function.

```
if __name__ == "__ main__":

        print ('Called with args:')
        input_dim = (120,120,3)

        run = 1
        n_folds = 2
        acc = np.zeros((run, n_folds), dtype=float)
        data_path = '\content\Brain Tumor Class'

        for irun in range(run):
                dataset = load_dataset(dataset_path=data_path, n_folds=n_folds, rand_state=irun)
                for ifold in range(n_folds):
                print ('run=', irun, ' fold=', ifold)
                acc[irun][ifold] = model_training(input_dim, dataset[ifold], irun, ifold)
        print ('mi-net mean accuracy = ', np.mean(acc))
        print ('std = ', np.std(acc))
```

The scripts and functions mentioned above are all provided in the ICT solution along with this project.

References

Ilse, M., Tomczak, J. M. and Welling, M. (2018). Attention-based deep multiple instance learning, *Conference Proceeding* .

Wang, X., Yan, Y., Tang, P., Bai, X. and Liu, W. (2018). Revisiting multiple instance neural networks, *Pattern Recognition* **74**: 15–24.