Configuration Manual

MSc Research Project
MSc in Data Analytics

Proshunjeet Sengupta
Student ID: 22131183

School of Computing
National College of Ireland

Supervisor: Prof. Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Proshunjeet Sengupta		
Student ID:	22131183		
Programme:	Msc in Data Analytics	**Year:**	2023
Module:	Msc Research Project		
Supervisor:	Vladimir Milosavljevic		
Submission Due Date:	14-12-2023		
Project Title:	Alzheimer disease early-stage diagnosis using Deep Learning methodologies		
Word Count:	1200 words	**Page Count**:	23

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: Proshunjeet Sengupta

Date: 14-12-2023

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

Proshunjeet Sengupta
Student ID: 22131183

1 Introduction

This Configuration Manual lists together all prerequisites needed to duplicate the studies and its effects on a specific setting. A glimpse of the source for Data analysis and after that images are augmented and prediction algorithms are built for detection of class.
The report is organized as follows, with details relating environment configuration provided in Section 2. Information about data gathering is detailed in Section 3. Exploratory Data Analysis is done in Section 4. Image Augmentation and Class Balancing is included in Section 5. In section 6, the Image Augmentation of balanced images is described. Details about models that were created and tested are provided in Section 7. How the results are calculated and shown is described in Section 8.

2 System Requirements

The specific needs for hardware as well as software to put the research into use are detailed in this section.

2.1 Hardware Requirements

The necessary hardware specs are shown in Figure 1 below.

Device name	LAPTOP-321A56J6
Processor	Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz
Installed	RAM 8.00 GB (7.78 GB usable)
System type	64-bit operating system, x64-based processor

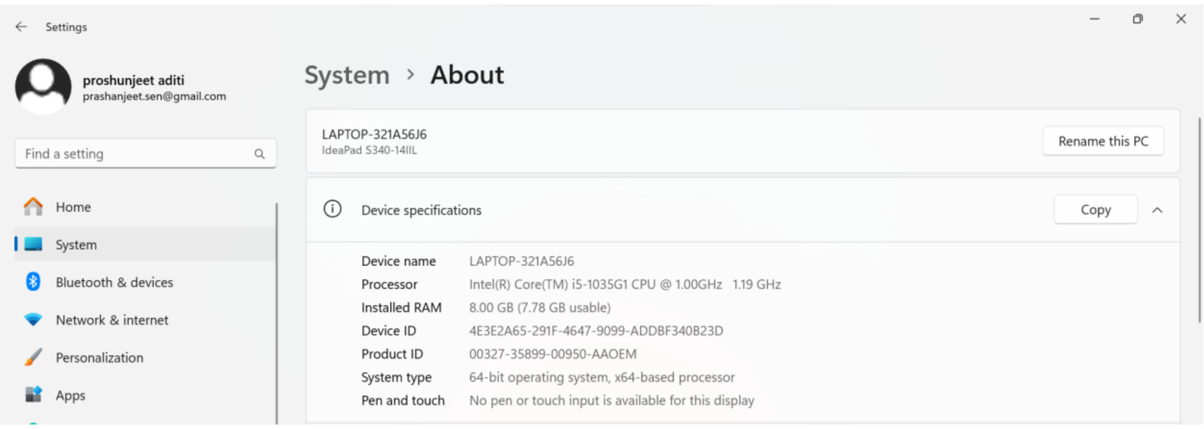

Figure 1: Hardware Requirements

2.2 Software Requirements

- Anaconda 3 (Version 4.8.0)
- Jupyter Notebook (Version 6.0.3)
- Python (Version 3.7.6)

2.3 Code Execution

The code can be run in jupyter notebook. The jupyter notebook comes with Anaconda 3, run the jupyter notebook from startup. This will open jupyter notebook in web browser. The web browser will show the folder structure of the system, move to the folder where the code file is located. Open the code file from the folder and to run the code, go to Kernel menu and Run all cells.

3 Data Collection

The dataset is taken from Kaggle public repository from the link [Alzheimer MRI Preprocessed Dataset (kaggle.com)](). The Dataset is consisting of Preprocessed MRI (Magnetic Resonance Imaging) Images resized into 128 x 128 pixels. The images belong four classes of images.

Class - 1: Mild Demented (896 images)
Class - 2: Moderate Demented (64 images)
Class - 3: Non-Demented (3200 images)
Class - 4: Very Mild Demented (2240 images)

4 Exploratory Data Analysis

Figure 2 includes a list of every Python library necessary to complete the project.

```
import glob, random, re
import os, sys
import pandas as pd
import shutil
import cv2
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
#Sharpening of images
from skimage.io import imshow, imread
import warnings
warnings.filterwarnings("ignore")
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv1D, Conv2D, MaxPooling2D, Dense, Flatten, Layer, Input, Dropout, Attention
from keras.applications.vgg16 import VGG16
from keras.applications.vgg19 import VGG19
from keras.applications.inception_v3 import InceptionV3
from keras.applications.efficientnet_v2 import EfficientNetV2B0
from keras.applications.densenet import DenseNet121
```

Figure 2: Necessary Python libraries

The Figure 3 represents the block of code to set the path for data folder and read the count of images in each class.

```
path_mildDemented = 'archive/Dataset/Mild_Demented/'
path_moderateDemented = 'archive/Dataset/Moderate_Demented/'
path_nonDemented = 'archive/Dataset/Non_Demented/'
path_veryMildDemented = 'archive/Dataset/Very_Mild_Demented/'
```

```
categories = ['Mild_Demented', 'Moderate_Demented', 'Non_Demented', 'Very_Mild_Demented']
print(categories)
```

```
['Mild_Demented', 'Moderate_Demented', 'Non_Demented', 'Very_Mild_Demented']
```

```
#initialization and importing for data analysi
count_mildDemented=len(os.listdir(path_mildDemented))
count_moderateDemented=len(os.listdir(path_moderateDemented))
count_nonDemented=len(os.listdir(path_nonDemented))
count_veryMildDemented=len(os.listdir(path_veryMildDemented))
count_mildDemented, count_moderateDemented, count_nonDemented, count_veryMildDemented
```

```
(896, 64, 3200, 2240)
```

Figure 3: Data Path and Count

As seen in Figure 4, illustrate the code to generate bar plot of the value counts.

```
#plotting graph for leaf count
fig = plt.figure(figsize = (10, 5))
values=[count_mildDemented, count_moderateDemented, count_nonDemented, count_veryMildDemented]
#creating the bar plot
plt.bar(categories, values, color ='blue', width = 0.4)

plt.xlabel("Classes")
plt.ylabel("Number of Images")
plt.show()
```

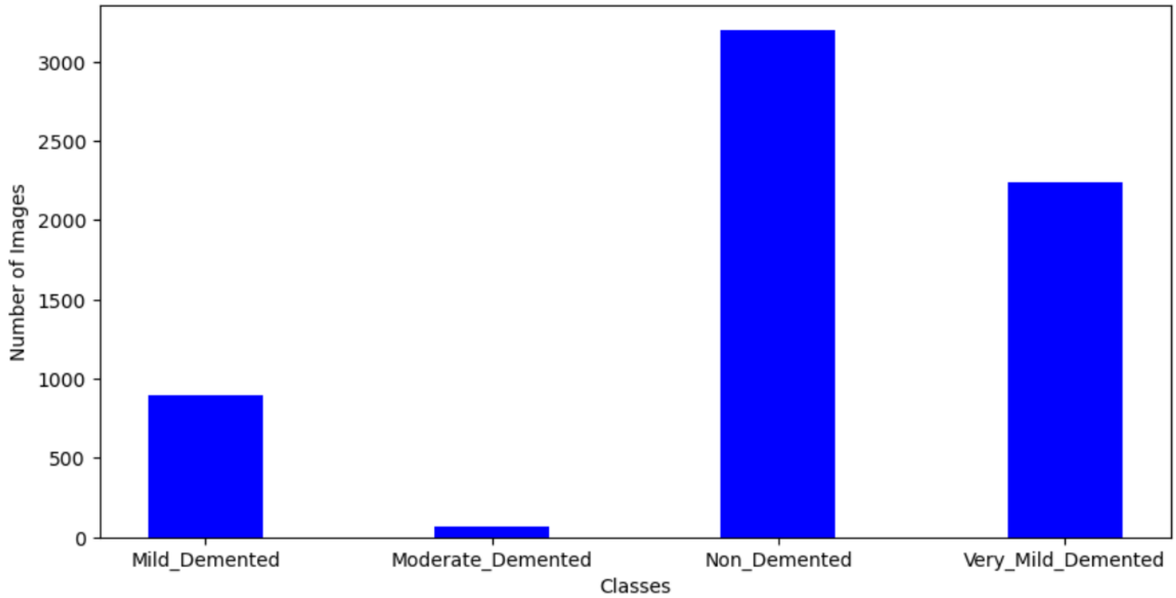

Figure 4: Data Classes and count

In figure 5, the code to generate list of images for each class.

```
mildDemented = glob.glob(path_mildDemented+"*.jpg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d  images.\nFirst 5 filenames:" % len(mildDemented))
print ('\n'.join(mildDemented[:5]))
```

```
Total of 896  images.
First 5 filenames:
archive/Dataset/Mild_Demented\mild.jpg
archive/Dataset/Mild_Demented\mild_10.jpg
archive/Dataset/Mild_Demented\mild_100.jpg
archive/Dataset/Mild_Demented\mild_101.jpg
archive/Dataset/Mild_Demented\mild_102.jpg
```

```
moderateDemented = glob.glob(path_moderateDemented + "*.jpg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d  images.\nFirst 5 filenames:" % len(moderateDemented))
print ('\n'.join(moderateDemented[:5]))
```

```
Total of 64  images.
First 5 filenames:
archive/Dataset/Moderate_Demented\moderate.jpg
archive/Dataset/Moderate_Demented\moderate_10.jpg
archive/Dataset/Moderate_Demented\moderate_11.jpg
archive/Dataset/Moderate_Demented\moderate_12.jpg
archive/Dataset/Moderate_Demented\moderate_13.jpg
```

```
nonDemented = glob.glob(path_nonDemented + "*.jpg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d  images.\nFirst 5 filenames:" % len(nonDemented))
print ('\n'.join(nonDemented[:5]))
```

```
Total of 3200  images.
First 5 filenames:
archive/Dataset/Non_Demented\non.jpg
archive/Dataset/Non_Demented\non_10.jpg
archive/Dataset/Non_Demented\non_100.jpg
archive/Dataset/Non_Demented\non_1000.jpg
archive/Dataset/Non_Demented\non_1001.jpg
```

```
veryMildDemented= glob.glob(path_veryMildDemented + "*.jpg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d  images.\nFirst 5 filenames:" % len(veryMildDemented))
print ('\n'.join(veryMildDemented[:5]))
```

```
Total of 2240  images.
First 5 filenames:
archive/Dataset/Very_Mild_Demented\verymild.jpg
archive/Dataset/Very_Mild_Demented\verymild_10.jpg
archive/Dataset/Very_Mild_Demented\verymild_100.jpg
archive/Dataset/Very_Mild_Demented\verymild_1000.jpg
archive/Dataset/Very_Mild_Demented\verymild_1001.jpg
```

Figure 5: Images in each class

5 Image Augmentation and Class Balancing

The Figure 6, illustrates the read the image and show image shape and plot image.

```python
def plot_image(img, cmap='gray'):
    fig = plt.figure(figsize=(8,8))
    axes = fig.add_subplot(111)
    axes.imshow(img, cmap=cmap)
```

```python
img1 = cv2.imread(veryMildDemented[1])
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
plot_image(img1)
width, height, dimension = img1.shape
print(f'Width RGB = {width}')
print(f'Height RGB = {height}')
print(f'Dimension RGB = {dimension}')
```

```
Width RGB = 128
Height RGB = 128
Dimension RGB = 3
```

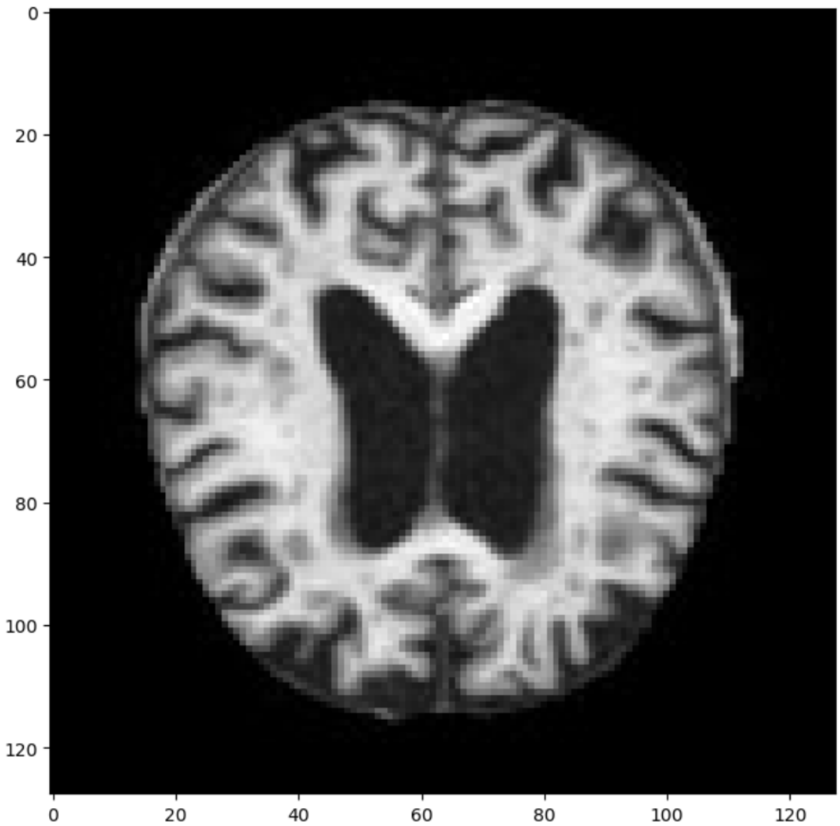

Figure 6: Read Image

The Figure 7, illustrate the read the image and generate histogram of the image.

```
vals = img1.mean(axis=2).flatten()
# plot histogram with 255 bins
b, bins, patches = plt.hist(vals,100)
plt.xlim([0,255])
plt.show()
```

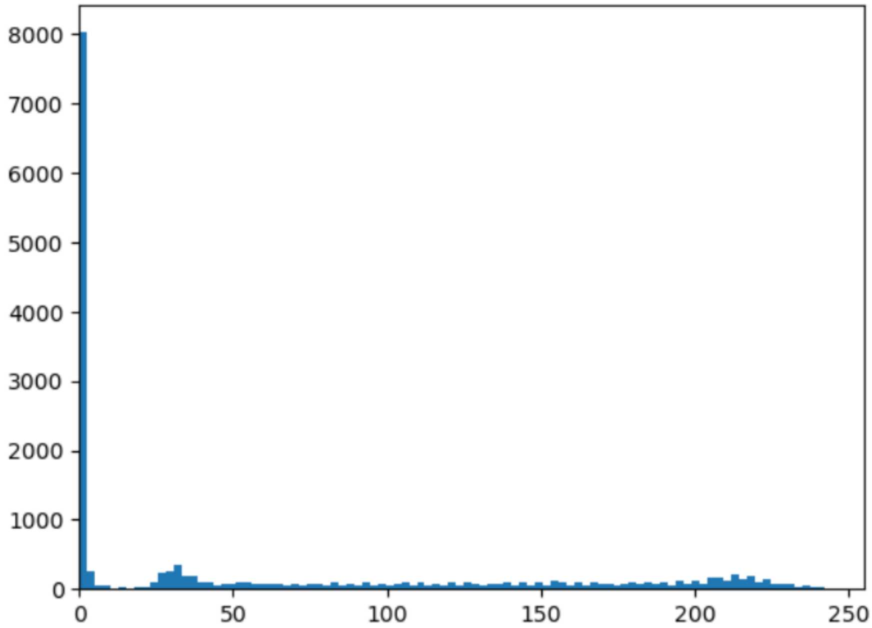

Figure 7: Histogram of image

The Figure 8 and Figure 9 illustrates the read the image in grayscale and generate histogram of the image.

```
hist = cv2.calcHist([img1], [0], None, [256], [0, 256])
plt.figure()
plot_image(img1)
plt.figure()
plt.title("Grayscale Histogram")
plt.xlabel("Bins")
plt.ylabel("# of Pixels")
plt.plot(hist)
plt.xlim([0, 256])
```

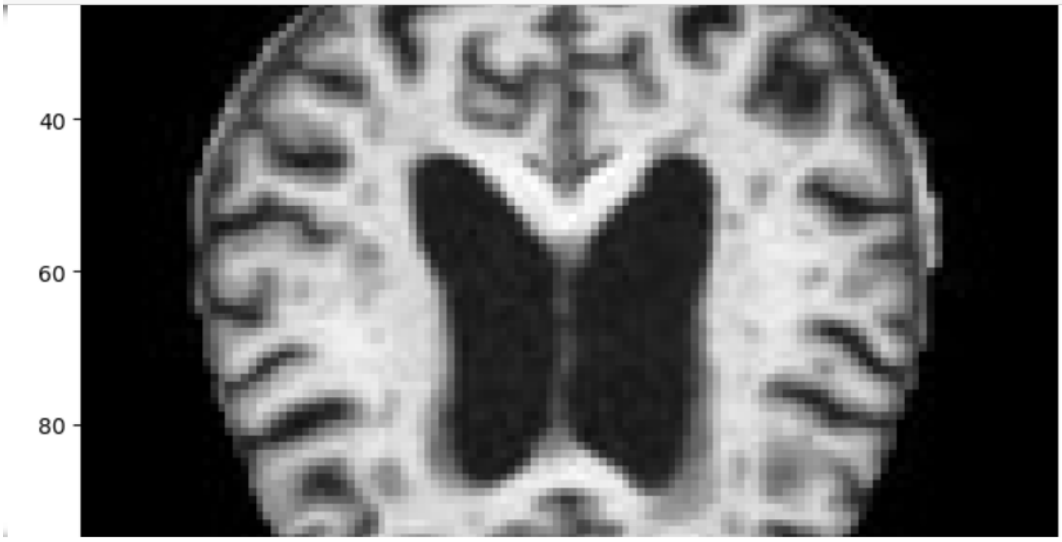

Figure 8: Read Image in gray scale and generate histogram

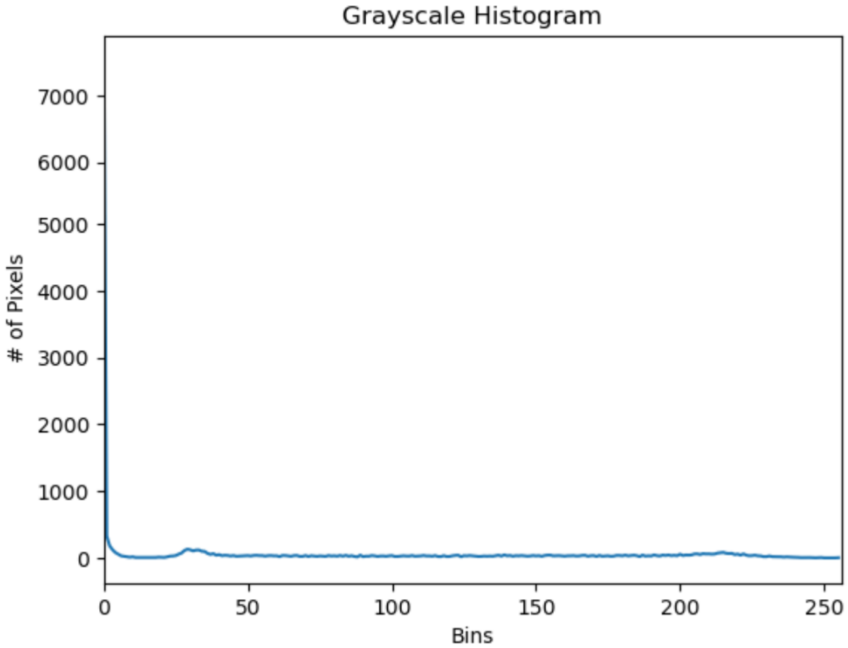

Figure 9: Histogram of Image

Figure 10, illustrates the read the image in RGB and generate histogram of the image.

```python
# tuple to select colors of each channel line
colors = ("red", "green", "blue")

# create the histogram plot, with three lines, one for
# each color
plt.figure()
plt.xlim([0, 256])
for channel_id, color in enumerate(colors):
    histogram, bin_edges = np.histogram(img1, bins=100, range=(0, 256))
    plt.plot(bin_edges[0:-1], histogram, color=color)

plt.title("Color Histogram")
plt.xlabel("Color value")
plt.ylabel("Pixel count")
```

```
Text(0, 0.5, 'Pixel count')
```

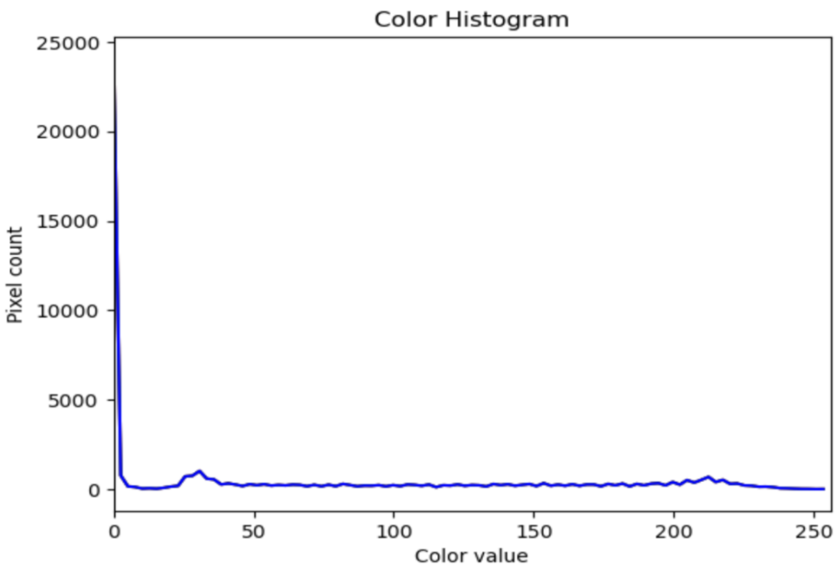

Figure 10: Histogram of RGB Image

```python
img1_gray = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
plot_image(img1_gray)
width, height = img1_gray.shape
print(f'Width Grayscale = {width}')
print(f'Height Grayscale = {height}')
print(f'Image Shape Grayscale {img1_gray.shape}')
```

```
Width Grayscale = 128
Height Grayscale = 128
Image Shape Grayscale (128, 128)
```

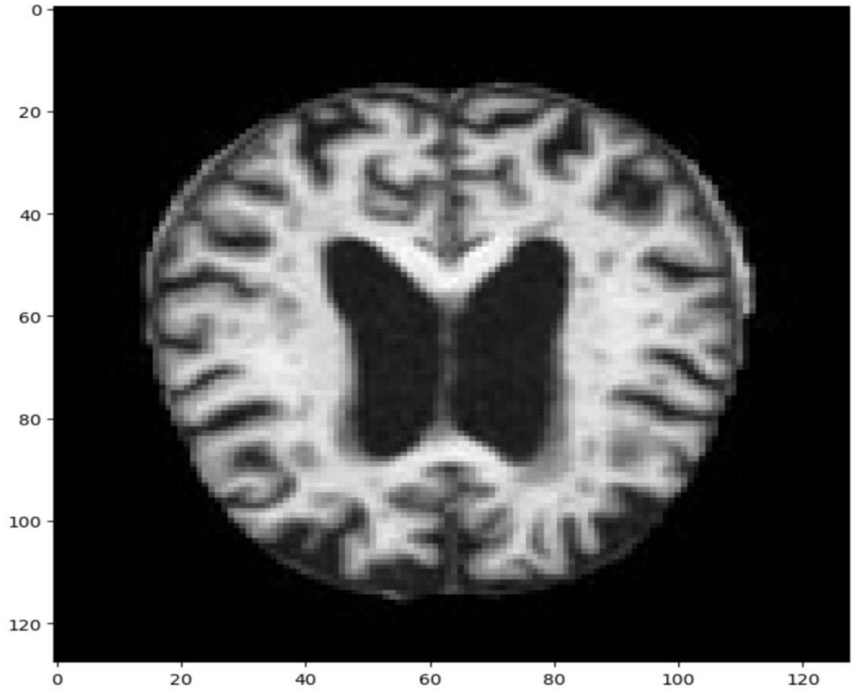

Figure 11: Read GrayScale Image

The Figure 12 and 13, illustrate the code to generate adaptive thresholds of the images and display the contours.

```python
img1_gray = cv2.adaptiveThreshold(img1_gray,5,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV,11,3)
plot_image(img1_gray)
```

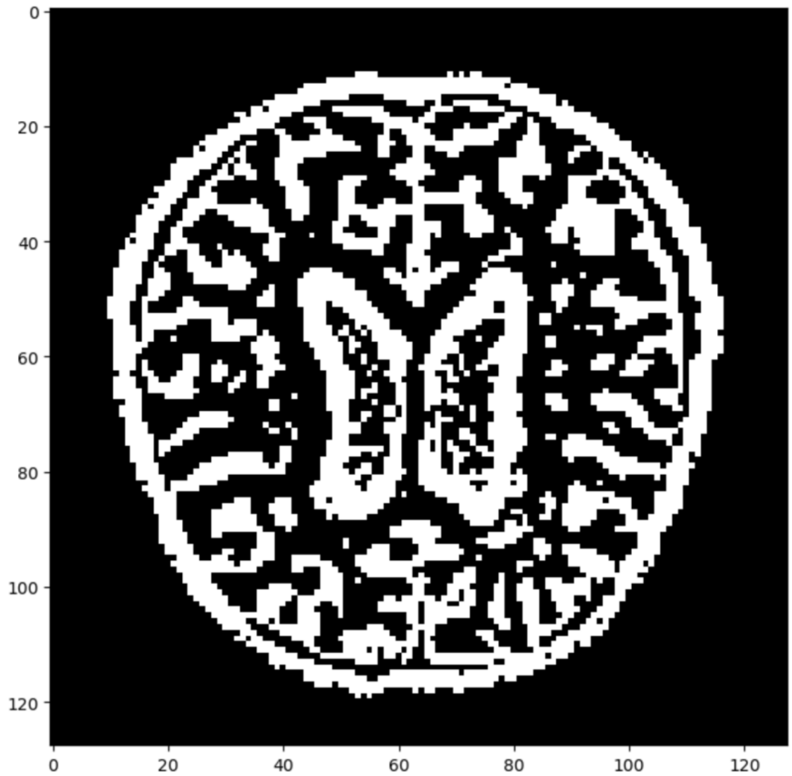

Figure 12: Adaptive threshold

```
contours = cv2.findContours(img1_gray, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours = contours[0] if len(contours) == 2 else contours[1]
contours = sorted(contours, key=cv2.contourArea, reverse=True)
for c in contours:
    x,y,w,h = cv2.boundingRect(c)
    img1_ROI = img1[y:y+h, x:x+w]
    break
plot_image(img1_ROI)
width, height, dimension = img1_ROI.shape
print(f'Width = {width}')
print(f'Height = {height}')
print(f'Dimension = {dimension}')
```

```
Width = 109
Height = 107
Dimension = 3
```

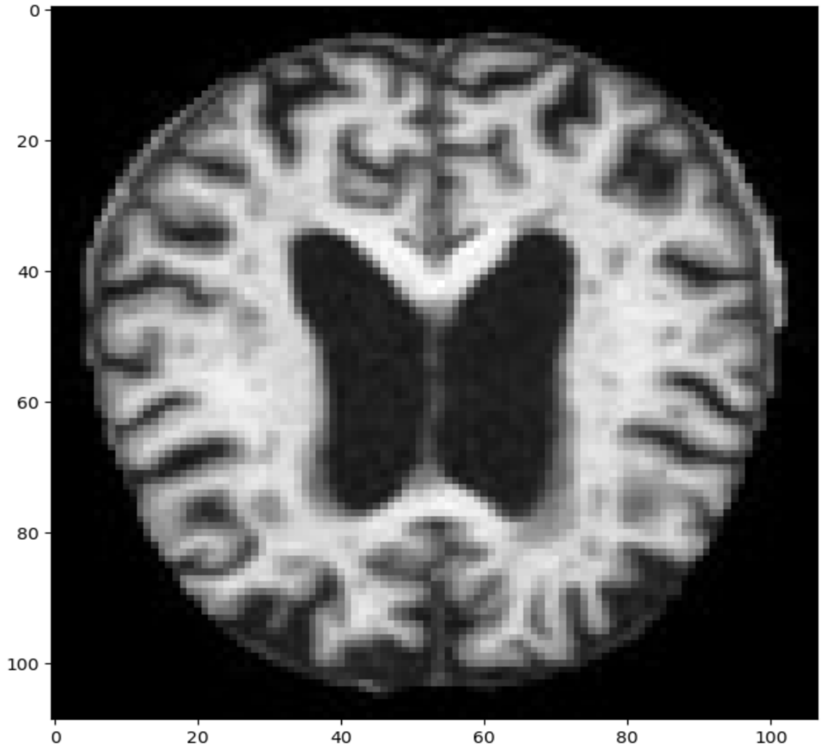

Figure 13: Adaptive threshold and contouring

The Figure 14, illustrates the code to get the maximum number of class count and set path for new oversampled folder.

```
n = np.max([count_mildDemented, count_moderateDemented, count_nonDemented, count_veryMildDemented])
n
```

```
3200
```

```
path_mildDemented = 'OverSampled/Dataset/Mild_Demented/'
path_moderateDemented = 'OverSampled/Dataset/Moderate_Demented/'
path_nonDemented = 'OverSampled/Dataset/Non_Demented/'
path_veryMildDemented = 'OverSampled/Dataset/Very_Mild_Demented/'
count_mildDemented=len(os.listdir(path_mildDemented))
count_moderateDemented=len(os.listdir(path_moderateDemented))
count_nonDemented=len(os.listdir(path_nonDemented))
count_veryMildDemented=len(os.listdir(path_veryMildDemented))
count_mildDemented, count_moderateDemented, count_nonDemented, count_veryMildDemented
```

```
(3200, 3198, 3200, 3203)
```

Figure 14: Count for maximum to oversample

The Figure 15, illustrate the code to generate ImageDataGenerator to create images using a same class image till the count reaches the maximum number.

```
datagen = ImageDataGenerator(fill_mode='nearest')
```

```
img = load_img(mildDemented[5])
x = img_to_array(img)
x = x.reshape((1,) + x.shape)
print(x.shape)
```

```
(1, 128, 128, 3)
```

```
i = count_mildDemented
for batch in datagen.flow(x, batch_size=1, save_to_dir=path_mildDemented, save_prefix='mildDemented', save_format='jpg'):
    i += 1
    if i > n:
        break
```

Figure 15: Image Data generator for mild Demented

The Figure 16 and 17 illustrates the code to get the maximum number of class count and set path for new oversampled folder for all the classes.

```
img = load_img(moderateDemented[4])
x = img_to_array(img)
x = x.reshape((1,) + x.shape)
print(x.shape)
```

```
(1, 128, 128, 3)
```

```
i = count_moderateDemented
for batch in datagen.flow(x, batch_size=1, save_to_dir=path_moderateDemented, save_prefix='moderateDemented', save_format='jpg')
    i += 1
    if i > n:
        break
```

Figure 16: Image Data generator for moderate Demented

```
img = load_img(veryMildDemented[3])
x = img_to_array(img)
x = x.reshape((1,) + x.shape)
print(x.shape)
```

```
(1, 128, 128, 3)
```

```
i = count_veryMildDemented
for batch in datagen.flow(x, batch_size=1, save_to_dir=path_veryMildDemented, save_prefix='veryMildDemented', save_format='jpg'):
    i += 1
    if i > n:
        break
```

Figure 17: Image Data generator for very mild Demented

The Figure 18 illustrates the code to get the class count and generate graph for it.

```
#initialization and importing for data analysi
count_mildDemented=len(os.listdir(path_mildDemented))
count_moderateDemented=len(os.listdir(path_moderateDemented))
count_nonDemented=len(os.listdir(path_nonDemented))
count_veryMildDemented=len(os.listdir(path_veryMildDemented))
count_mildDemented, count_moderateDemented, count_nonDemented, count_veryMildDemented
```

```
(3200, 3201, 3200, 3204)
```

```
#plotting graph for leaf count
fig = plt.figure(figsize = (10, 5))
values=[count_mildDemented, count_moderateDemented, count_nonDemented, count_veryMildDemented]
#creating the bar plot
plt.bar(categories, values, color ='blue', width = 0.4)

plt.xlabel("Classes")
plt.ylabel("Number of Images")
plt.show()
```

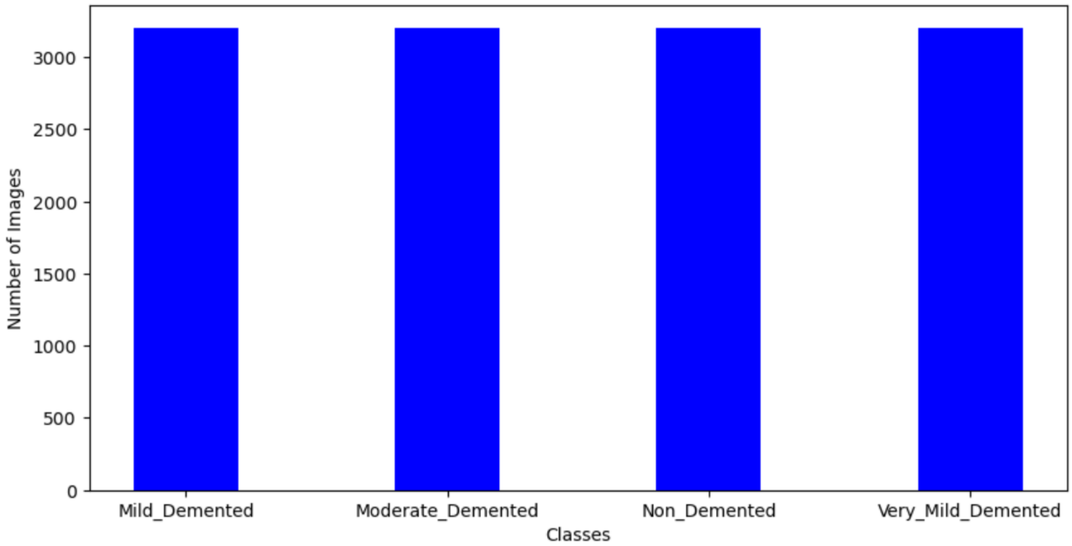

Figure 18: Oversampled images

The Figure 19-20 illustrates the code to list the train and test folder and generate list of images for each class.

```
trainDir = '/content/drive/MyDrive/Alzeihmer/Data/train'
testDir = '/content/drive/MyDrive/Alzeihmer/Data/test'
```

```
try:
    os.makedirs(trainDir)
    os.makedirs(testDir)
    print("Folders created")
except:
    print("Folders already created")
```

```
Folders already created
```

```
mildDemented = glob.glob(path_mildDemented+"*.jpg")
# Print out the first 5 file names to verify we're in the right folder
print ("Total of %d  images.\nFirst 5 filenames:" % len(mildDemented))
print ('\n'.join(mildDemented[:5]))
```

```
Total of 3200  images.
First 5 filenames:
OverSampled/Dataset/Mild_Demented\mild.jpg
OverSampled/Dataset/Mild_Demented\mildDemented_0_1.jpg
OverSampled/Dataset/Mild_Demented\mildDemented_0_10.jpg
OverSampled/Dataset/Mild_Demented\mildDemented_0_100.jpg
OverSampled/Dataset/Mild_Demented\mildDemented_0_1006.jpg
```

Figure 19: Images in each class

```
moderateDemented = glob.glob(path_moderateDemented + "*.jpg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d  images.\nFirst 5 filenames:" % len(moderateDemented))
print ('\n'.join(moderateDemented[:5]))
```

```
Total of 3201  images.
First 5 filenames:
OverSampled/Dataset/Moderate_Demented\moderate.jpg
OverSampled/Dataset/Moderate_Demented\moderateDemented_0_0.jpg
OverSampled/Dataset/Moderate_Demented\moderateDemented_0_1003.jpg
OverSampled/Dataset/Moderate_Demented\moderateDemented_0_1005.jpg
OverSampled/Dataset/Moderate_Demented\moderateDemented_0_1006.jpg
```

```
nonDemented = glob.glob(path_nonDemented + "*.jpg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d  images.\nFirst 5 filenames:" % len(nonDemented))
print ('\n'.join(nonDemented[:5]))
```

```
Total of 3200  images.
First 5 filenames:
OverSampled/Dataset/Non_Demented\non.jpg
OverSampled/Dataset/Non_Demented\non_10.jpg
OverSampled/Dataset/Non_Demented\non_100.jpg
OverSampled/Dataset/Non_Demented\non_1000.jpg
OverSampled/Dataset/Non_Demented\non_1001.jpg
```

```
veryMildDemented= glob.glob(path_veryMildDemented + "*.jpg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d  images.\nFirst 5 filenames:" % len(veryMildDemented))
print ('\n'.join(veryMildDemented[:5]))
```

```
Total of 3204  images.
First 5 filenames:
OverSampled/Dataset/Very_Mild_Demented\verymild.jpg
OverSampled/Dataset/Very_Mild_Demented\veryMildDemented_0_1008.jpg
OverSampled/Dataset/Very_Mild_Demented\veryMildDemented_0_102.jpg
OverSampled/Dataset/Very_Mild_Demented\veryMildDemented_0_1023.jpg
OverSampled/Dataset/Very_Mild_Demented\veryMildDemented_0_1032.jpg
```

Figure 20: Images in each class

The Figure 21 illustrates the code for function definition to classify the images into the defined folder category and generating train and test data.

```
try:
    for category in categories:
        path = os.path.join(trainDir, category)
        os.makedirs(path)
        path = os.path.join(testDir, category)
        os.makedirs(path)
    print("Folders created")
except:
    print("Folders already created")
```

```
Folders already created
```

```
def generateData(lst,fnm):
    for i in range(len(lst)):
        if(i<=(len(lst)-len(lst)*.2)):
            destination=trainDir+ '/'+fnm
        else:
            destination=testDir+'/'+fnm
        shutil.copy(lst[i], destination)
```

Figure 21: Images in each class

The Figure 22 illustrates the code to execute the function for each class.

```
try:
    generateData(mildDemented,"Mild_Demented")
    print("Images set in training and testing folders")
except:
    print("Images already set in training and testing folders")
```

```
Images set in training and testing folders
```

```
try:
    generateData(moderateDemented,"Moderate_Demented")
    print("Images set in training and testing folders")
except:
    print("Images already set in training and testing folders")
```

```
Images set in training and testing folders
```

```
try:
    generateData(nonDemented,"Non_Demented")
    print("Images set in training and testing folders")
except:
    print("Images already set in training and testing folders")
```

```
Images set in training and testing folders
```

```
try:
    generateData(veryMildDemented,"Very_Mild_Demented")
    print("Images set in training and testing folders")
except:
    print("Images already set in training and testing folders")
```

```
Images set in training and testing folders
```

Figure 22: Images in each class

6 Image Augmentation

The Figure 23, illustrate the code to use ImageDataGenerator to generate augmented images for the deep learning models.

```
IMG_SIZE = 100
```

```
# This function will plot images in the form of a grid with 1 row and 5 columns where images are placed in each column.
def plotImages(images_arr):
    fig, axes = plt.subplots(4, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.title('Before Augmentation')
    plt.show()
datagen = ImageDataGenerator()
pic = datagen.flow_from_directory(directory=trainDir, target_size=(IMG_SIZE,IMG_SIZE))
augmented_images = [pic[0][0][0] for i in range(20)]
plotImages(augmented_images)
```

```
Found 10951 images belonging to 4 classes.
```

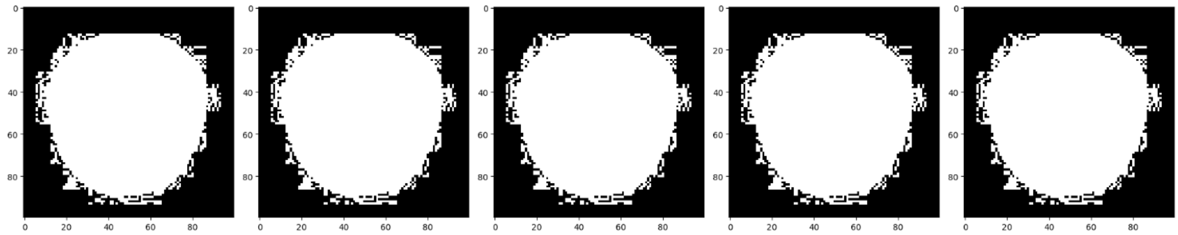

Figure 23: Image Data Generator

Figures 24 shows the code to create training data with rescaling the images.

```
gen = ImageDataGenerator(rescale=1/255)
gen
```

```
<keras.src.preprocessing.image.ImageDataGenerator at 0x18b6e8aa810>
```

```
train = gen.flow_from_directory(directory=trainDir, target_size=(IMG_SIZE,IMG_SIZE))
```

```
Found 10951 images belonging to 4 classes.
```

```
# This function will plot images in the form of a grid with 1 row and 5 columns where images are placed in each column.
def plotImages(images_arr):
    fig, axes = plt.subplots(4, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.title('After Augmentation')
    plt.show()
augmented_images = [train[0][0][0] for i in range(20)]
plotImages(augmented_images)
```

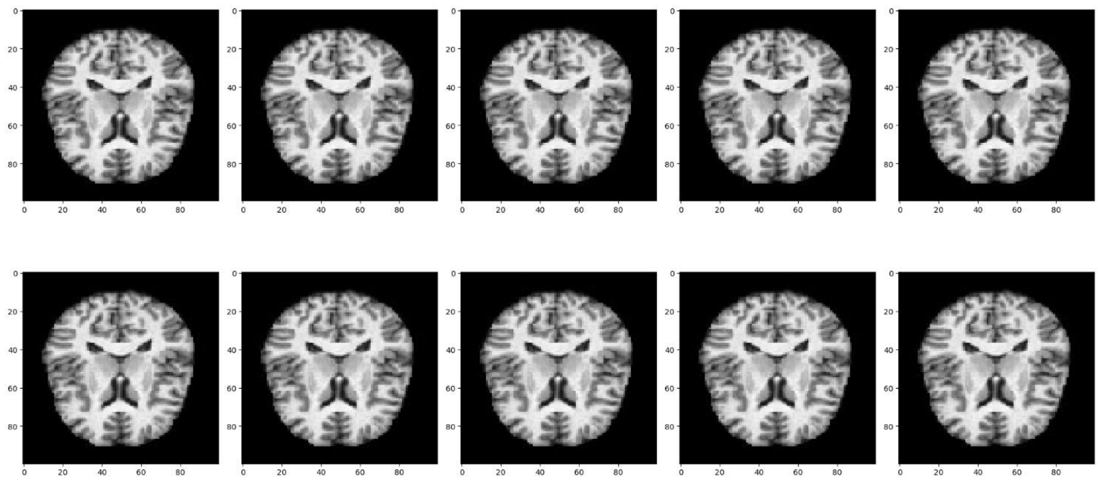

Figure 24: Image Data Generator

Figures 25 shows the code to create testing data with rescaling the images.

```
test = gen.flow_from_directory(directory=testDir, target_size=(IMG_SIZE,IMG_SIZE))
```

```
Found 2563 images belonging to 4 classes.
```

```
augmented_images = [test[0][0][0] for i in range(20)]
plotImages(augmented_images)
```

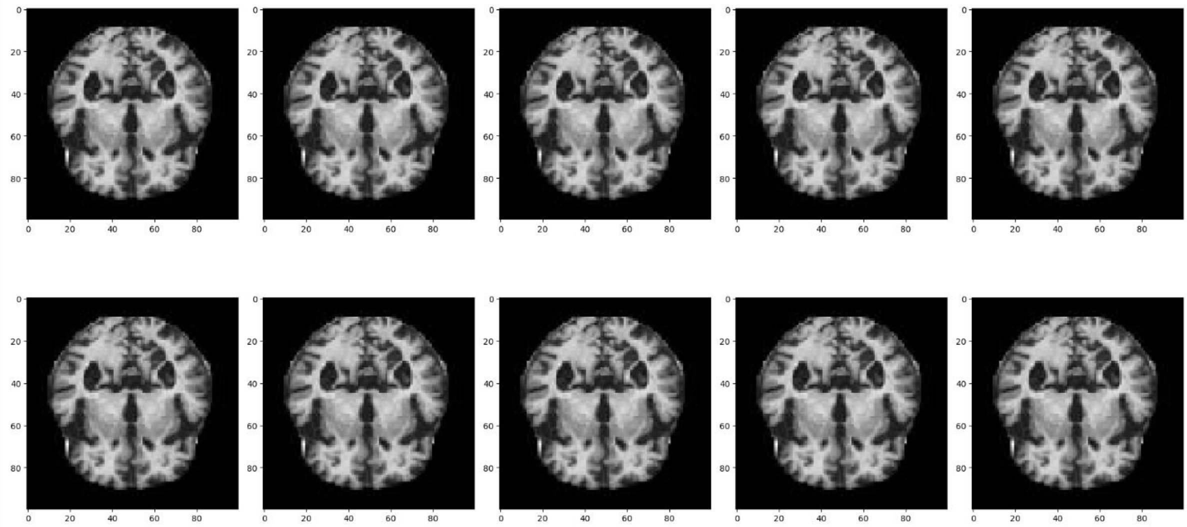

Figure 25: Image Data Generator

7 Deep Learning Models

7.1 InceptionNet

```
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

```
# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)
```

```
WARNING:tensorflow:From C:\Users\ashis\anaconda3\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_out
side_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\ashis\anaconda3\Lib\site-packages\keras\src\layers\normalization\batch_normalization.py:979: T
he name tf.nn.fused_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.
```

```
# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(4, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
for layer in base_model.layers:
    layer.trainable = False
```

```
model.compile(optimizer = 'adam', loss= 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, None, None, 3)]      0         []

 conv2d (Conv2D)             (None, None, None, 32)       864       ['input_1[0][0]']

 batch_normalization (Batch  (None, None, None, 32)       96        ['conv2d[0][0]']
 Normalization)

 activation (Activation)     (None, None, None, 32)       0         ['batch_normalization[0][0]']

 conv2d_1 (Conv2D)           (None, None, None, 32)       9216      ['activation[0][0]']

 batch_normalization_1 (Bat  (None, None, None, 32)       96        ['conv2d_1[0][0]']
 chNormalization)

 activation_1 (Activation)   (None, None, None, 32)       0         ['batch_normalization_1[0][0]'
```

```
callback = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, mode='min', verbose=True)
```

```
history = model.fit(train, validation_data= test, epochs=50, batch_size=10000, callbacks=[callback])
```

```
Epoch 1/50
WARNING:tensorflow:From C:\Users\ashis\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTe
nsorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\ashis\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.execut
ing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

343/343 [==============================] - 86s 238ms/step - loss: 0.6139 - accuracy: 0.7804 - val_loss: 1.2169 - val_accuracy:
0.4729
Epoch 2/50
343/343 [==============================] - 64s 187ms/step - loss: 0.3947 - accuracy: 0.8224 - val_loss: 1.0728 - val_accuracy:
0.4803
```

Figure 26: Implementation of InceptionNet

7.2 CNN

```
model=Sequential()
model.add(Conv2D(kernel_size=64,strides=5, filters=5, padding='same',input_shape=input_shape, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(64, activation= 'relu'))
model.add(Dense(4, activation= 'sigmoid'))
model.compile(optimizer = 'adam', loss= 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_94 (Conv2D)          (None, 20, 20, 5)         61445

 max_pooling2d_4 (MaxPoolin  (None, 10, 10, 5)         0
 g2D)

 dropout_1 (Dropout)         (None, 10, 10, 5)         0

 flatten_1 (Flatten)         (None, 500)               0

 dense_4 (Dense)             (None, 64)                32064

 dense_5 (Dense)             (None, 4)                 260

=================================================================
Total params: 93769 (366.29 KB)
Trainable params: 93769 (366.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history = model.fit(train, validation_data= test, epochs=50, batch_size=10000, callbacks=[callback])
```

```
Epoch 1/50
343/343 [==============================] - 211s 612ms/step - loss: 0.5741 - accuracy: 0.7424 - val_loss: 1.2076 - val_accuracy:
0.4959
Epoch 2/50
343/343 [==============================] - 212s 619ms/step - loss: 0.4569 - accuracy: 0.7849 - val_loss: 1.0793 - val_accuracy:
0.4776
Epoch 3/50
343/343 [==============================] - 209s 611ms/step - loss: 0.4284 - accuracy: 0.8003 - val_loss: 0.9714 - val_accuracy:
0.4791
Epoch 4/50
343/343 [==============================] - 211s 615ms/step - loss: 0.4181 - accuracy: 0.8021 - val_loss: 1.0097 - val_accuracy:
0.4659
```

Figure 27: Implementation of CNN

7.3 Attention CNN

```
x=Input(shape=input_shape)
flatten=Flatten()(x)
conv_layer = Conv2D(filters =3, kernel_size = 3, padding = "same", activation = "relu")(x)
conv_flatten=Flatten()(conv_layer)
attention_layer = Attention(score_mode="dot")([flatten, conv_flatten])
flatten = Flatten()(attention_layer)
outputs=Dense(1, activation="sigmoid")(flatten)
model=Model(x,outputs)
```

```
model.compile(optimizer = 'rmsprop', loss= 'binary_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_3 (InputLayer)        [(None, 100, 100, 3)]        0         []

 conv2d_95 (Conv2D)          (None, 100, 100, 3)          84        ['input_3[0][0]']

 flatten_2 (Flatten)         (None, 30000)                0         ['input_3[0][0]']

 flatten_3 (Flatten)         (None, 30000)                0         ['conv2d_95[0][0]']

 attention (Attention)       (None, 30000)                0         ['flatten_2[0][0]',
                                                                     'flatten_3[0][0]']

 flatten_4 (Flatten)         (None, 30000)                0         ['attention[0][0]']

 dense_6 (Dense)             (None, 1)                    30001     ['flatten_4[0][0]']

==================================================================================================
Total params: 30085 (117.52 KB)
Trainable params: 30085 (117.52 KB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

```
history = model.fit(train, validation_data= test, epochs=10, batch_size=10000, callbacks=[callback])
```

```
Epoch 1/10
343/343 [==============================] - 11s 29ms/step - loss: 0.5803 - accuracy: 0.7461 - val_loss: 0.5632 - val_accuracy: 0.7500
Epoch 2/10
343/343 [==============================] - 10s 29ms/step - loss: 0.5630 - accuracy: 0.7500 - val_loss: 0.5630 - val_accuracy: 0.7500
Epoch 3/10
343/343 [==============================] - 10s 29ms/step - loss: 0.5628 - accuracy: 0.7500 - val_loss: 0.5627 - val_accuracy: 0.7500
Epoch 4/10
343/343 [==============================] - 10s 29ms/step - loss: 0.5627 - accuracy: 0.7500 - val_loss: 0.5629 - val_accuracy: 0.7500
Epoch 5/10
343/343 [==============================] - 10s 29ms/step - loss: 0.5626 - accuracy: 0.7500 - val_loss: 0.5636 - val_accuracy: 0.7500
```

Figure 28: Implementation of Attention CNN

7.4 VGG19

```
model = VGG19(include_top=False,  weights='imagenet',  input_shape=input_shape)
model.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_4 (InputLayer)        [(None, 100, 100, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 100, 100, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 100, 100, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 50, 50, 64)        0         
                                                                 
 block2_conv1 (Conv2D)       (None, 50, 50, 128)       73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 50, 50, 128)       147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 25, 25, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 25, 25, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 25, 25, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 25, 25, 256)       590080    
                                                                 
 block3_conv4 (Conv2D)       (None, 25, 25, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 12, 12, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 12, 12, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 12, 12, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 12, 12, 512)       2359808   
                                                                 
 block4_conv4 (Conv2D)       (None, 12, 12, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 6, 6, 512)         0         
                                                                 
 block5_conv1 (Conv2D)       (None, 6, 6, 512)         2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 6, 6, 512)         2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 6, 6, 512)         2359808   
                                                                 
 block5_conv4 (Conv2D)       (None, 6, 6, 512)         2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 3, 3, 512)         0         
                                                                 
=================================================================
Total params: 20024384 (76.39 MB)
Trainable params: 20024384 (76.39 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 29: Implementation of VGG19

```
vgg19 = model.output
vgg19 = Flatten()(vgg19)
vgg19 = Dense(256, activation='relu')(vgg19)
vgg19 = Dropout(0.02)(vgg19)
output_layer = Dense(1, activation='tanh')(vgg19)

model = Model(inputs=model.input, outputs=output_layer)
```

```
model.compile(optimizer = 'sgd', loss= 'binary_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 100, 100, 3)]     0

 block1_conv1 (Conv2D)       (None, 100, 100, 64)      1792

 block1_conv2 (Conv2D)       (None, 100, 100, 64)      36928

 block1_pool (MaxPooling2D)  (None, 50, 50, 64)        0

 block2_conv1 (Conv2D)       (None, 50, 50, 128)       73856

 block2_conv2 (Conv2D)       (None, 50, 50, 128)       147584

 block2_pool (MaxPooling2D)  (None, 25, 25, 128)       0

 block3_conv1 (Conv2D)       (None, 25, 25, 256)       295168

 block3_conv2 (Conv2D)       (None, 25, 25, 256)       590080

 block3_conv3 (Conv2D)       (None, 25, 25, 256)       590080

 block3_conv4 (Conv2D)       (None, 25, 25, 256)       590080

 block3_pool (MaxPooling2D)  (None, 12, 12, 256)       0

 block4_conv1 (Conv2D)       (None, 12, 12, 512)       1180160
```

```
history = model.fit(train, validation_data= test, epochs=10, batch_size=10000, callbacks=[callback])
```

```
Epoch 1/10
343/343 [==============================] - 1099s 3s/step - loss: 3.8471 - accuracy: 0.7492 - val_loss: 3.8562 - val_accuracy:
0.7500
Epoch 2/10
343/343 [==============================] - 1148s 3s/step - loss: 3.8562 - accuracy: 0.7500 - val_loss: 3.8562 - val_accuracy:
0.7500
Epoch 3/10
343/343 [==============================] - 1107s 3s/step - loss: 3.8562 - accuracy: 0.7500 - val_loss: 3.8562 - val_accuracy:
0.7500
Epoch 4/10
343/343 [==============================] - 1086s 3s/step - loss: 3.8562 - accuracy: 0.7500 - val_loss: 3.8562 - val_accuracy:
0.7500
```

Figure 30: Implementation of VGG19

7.5 DenseNet 121

```
model = DenseNet121(include_top=False, weights="imagenet", classifier_activation="sigmoid", input_shape=input_shape)
model.summary()
```

```
==================================================================================================
 input_2 (InputLayer)         [(None, 100, 100, 3)]       0          []

 zero_padding2d (ZeroPaddin   (None, 106, 106, 3)         0          ['input_2[0][0]']
 g2D)

 conv1/conv (Conv2D)          (None, 50, 50, 64)          9408       ['zero_padding2d[0][0]']

 conv1/bn (BatchNormalizati   (None, 50, 50, 64)          256        ['conv1/conv[0][0]']
 on)

 conv1/relu (Activation)      (None, 50, 50, 64)          0          ['conv1/bn[0][0]']

 zero_padding2d_1 (ZeroPadd   (None, 52, 52, 64)          0          ['conv1/relu[0][0]']
 ing2D)

 pool1 (MaxPooling2D)         (None, 25, 25, 64)          0          ['zero_padding2d_1[0][0]']

 conv2_block1_0_bn (BatchNo   (None, 25, 25, 64)          256        ['pool1[0][0]']
```

```
denseNet = model.output
denseNet = Flatten()(denseNet)
denseNet = Dense(256, activation='tanh')(denseNet)
denseNet = Dropout(0.02)(denseNet)
output_layer = Dense(4, activation='sigmoid')(denseNet)

model = Model(inputs=model.input, outputs=output_layer)
```

```
model.compile(optimizer = 'sgd', loss= 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                 Output Shape                Param #    Connected to
==================================================================================================
 input_2 (InputLayer)         [(None, 100, 100, 3)]       0          []

 zero_padding2d (ZeroPaddin   (None, 106, 106, 3)         0          ['input_2[0][0]']
 g2D)

 conv1/conv (Conv2D)          (None, 50, 50, 64)          9408       ['zero_padding2d[0][0]']

 conv1/bn (BatchNormalizati   (None, 50, 50, 64)          256        ['conv1/conv[0][0]']
 on)

 conv1/relu (Activation)      (None, 50, 50, 64)          0          ['conv1/bn[0][0]']

 zero_padding2d_1 (ZeroPadd   (None, 52, 52, 64)          0          ['conv1/relu[0][0]']
 ing2D)
```

```
history = model.fit(train, validation_data= test, epochs=50, batch_size=10000, callbacks=[callback])
```

```
Epoch 1/50
343/343 [==============================] - 401s 1s/step - loss: 0.4026 - accuracy: 0.8280 - val_loss: 1.5562 - val_accuracy: 0.
4663
Epoch 2/50
343/343 [==============================] - 392s 1s/step - loss: 0.1882 - accuracy: 0.9255 - val_loss: 0.9871 - val_accuracy: 0.
6964
Epoch 3/50
343/343 [==============================] - 380s 1s/step - loss: 0.0791 - accuracy: 0.9716 - val_loss: 1.0395 - val_accuracy: 0.
7222
```

Figure 22: Implementation of DenseNet

8 Model result

This section explains the performance of the models.

8.1 Model Scores

```
modelScores= pd.DataFrame({'Models':['InceptionNet', 'DenseNet121', 'CNN', 'Attention Based CNN', 'VGG19'],
                           'Accuracy':[accuracyInc, accuracyDns, accuracyCnn, accuracyAtn, accuracyV19]})
```

```
modelScores
```

	Models	Accuracy
0	InceptionNet	54.896605
1	DenseNet121	81.076860
2	CNN	54.857588
3	Attention Based CNN	75.000000
4	VGG19	75.000000

Figure 23: Model Performance

```
plt.figure(figsize=(8,5))
plt.bar(modelScores['Models'], modelScores['Accuracy'])
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3, 4],
 [Text(0, 0, 'InceptionNet'),
  Text(1, 0, 'DenseNet121'),
  Text(2, 0, 'CNN'),
  Text(3, 0, 'Attention Based CNN'),
  Text(4, 0, 'VGG19')])
```

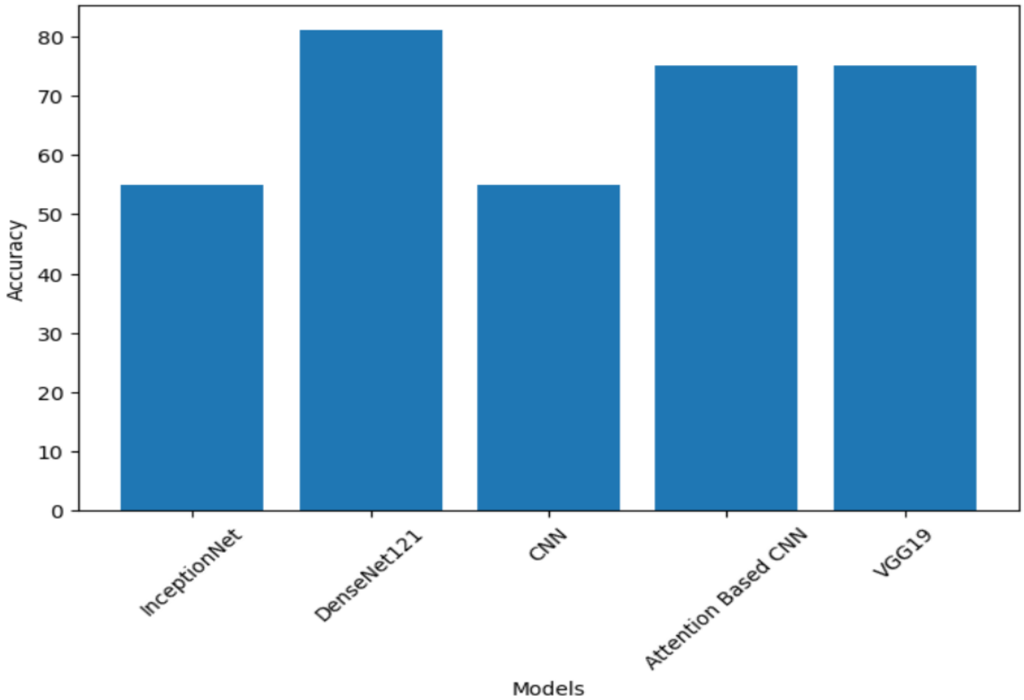

Figure 24: Model Performance

References

Alzheimer MRI Preprocessed Dataset (kaggle.com)

OpenCV: OpenCV modules

Keras Applications