MRI Images for Prediction of Stages of Alzheimer's Disease Using Deep Learning Models

Cassie Shen, Ziyang Zhang, Xiangcheng Chen, Chelsea Hua, Chunhe Wang

Table of Contents:

I. Abstract

Aware of the negative effects that Alzheimer's Disease could bring to a single person and to families, we want to find the early stage of AD in MRI brain images–by applying three different deep learning models, we successfully classified different stages of Alzheimer's based on their MRI brain image. During data exploratory analysis, we noticed the imbalance between four AD categories and we did data augmentation by flipping images and adding noises to images while the key features were not touched. Aligned with the general biological ideas of AD, our images showed a trend of loss of gray matter (cerebral cortex) and shrinkage of hippocampus. The three applied deep learning models, which are NasNetMobile, VGG16, and CNN, have accuracies 0.69, 0.99 and 0.97 respectively.

II. Background and Introduction

2.1 Background

Alzheimer's disease (AD) is the most common form of dementia, which is also a retreat of brain neurons among the orders worldwide. AD can cause many complications such as

Impaired Memory, Restlessness, and Language Deterioration. There are now 5.8 million patients in the United States. The number of people affected is reportedly expected to double in the next 30 years, affecting 1 in 85 people by 2050. Unfortunately, we have not found any effective therapeutic. Therefore, it is important to accurately diagnose AD, especially in its early stages, also known as amnestic mild cognitive impairment (MCI). AD is known to be associated with structural atrophy, pathological amyloid deposition and metabolic alterations in the brain. Currently, several biomarkers have been shown to be sensitive to AD and MCI, including brain atrophy measured in magnetic resonance (MR) imaging, hypometabolism measured by functional imaging , and quantification of specific proteins as measured by CSF.

With the rise of machine learning, the identification of different stages of AD by cluster analysis of MRI has been a mainstream approach in recent years. For example, many high-dimensional classification methods use structural MRI brain images to classify between AD (or MCI) and healthy controls. In contrast, our current project focuses on the impact of different deep learning models on the analysis of MRI.

2.2 Introduction

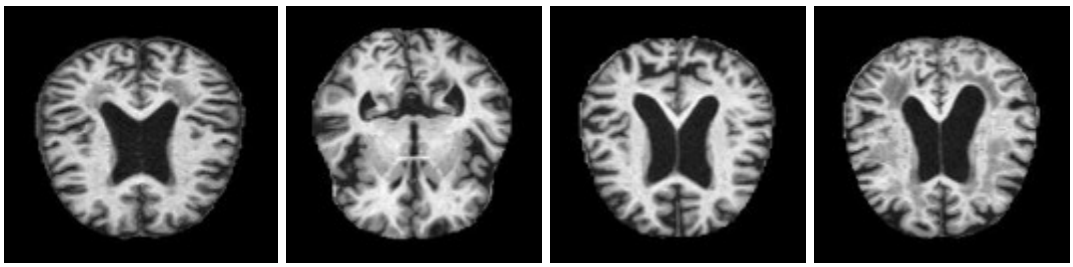
We are aware of the negative effects that Alzheimer's Disease could bring to a single person and to his or her family. To do some help for this common disease, we focus on the early diagnosis. Since the development of Alzheimer's Disease is a progressive process, we believe that machine learning methods could step in—by classifications of different stages of Alzheimer's Disease, we could help define an MRI image of an early stage of Alzheimer's or even an early stage of cognitive impairment. (MRI is magnetic resonance imaging.) Moreover, we could help locate the key features of the brain for Alzheimer's. By distinguishing the MRI

image of the early stage of Alzhermer's, we hope early interventions and prevention methods could be introduced to patients with early MCI (Mild Cognitive Impairment) or early AD (Alzheimer's Disease.)

In our project, we have around 6500 MRI images of 4 different stages of Alzheimer's—NonDemented, Very Mild Demented, Mild Denmentd, and Moderate. First, we will explore the images to find the key figures and degeneration trend in the brain when Alzheimer's becomes severe and will prepare the dataset for a deep learning model.

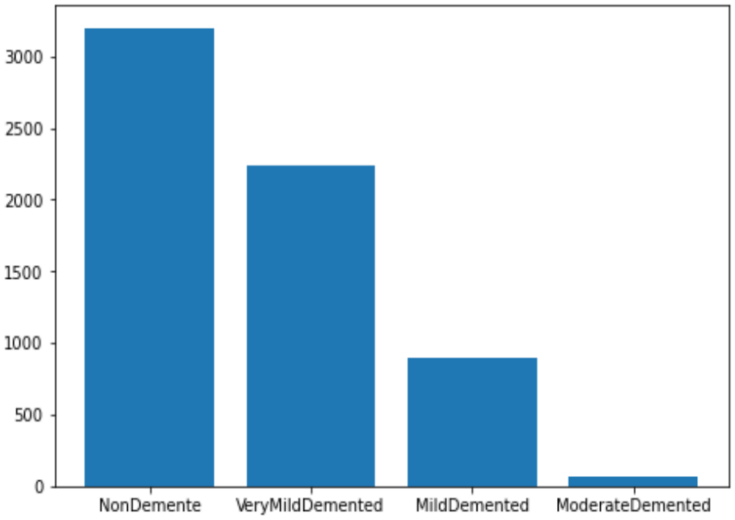
III. Exploratory Data Analysis

Our MRI image dataset belongs to the axiel plane MRI image. Different from other MRI datasets, our image is black and white and the size is 128, 128.



From left to right are Non Demente, VeryMildDemented, MildDemented, ModerateDemented. We can clearly find that the MRI goes from left to right, and the brain atrophy is getting more and more serious. Alzheimer's disease causes the brain to shrink from the outside to the inside, so we can observe that the degree of brain atrophy in the third and fourth pictures is significantly higher than that in the first and second pictures.

3.1 Data Count



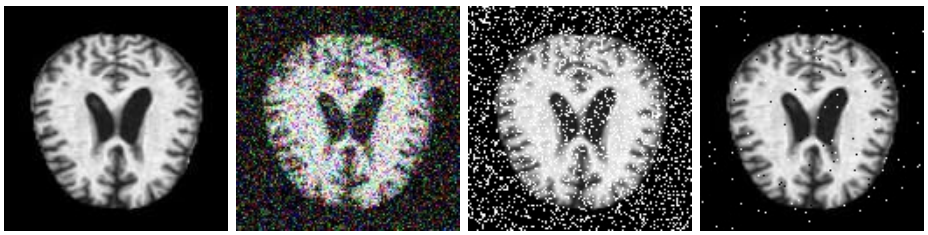
We can observe from the bar plot above, the distribution of data across the four categories is extremely uneven. The entire dataset contains 3200 NonDemente categories, 2240 VeryMildDemented categories, 896 Mild Demented categories, and 64 Moderat Demented categories. Extremely uneven distribution of data may overfit our model, or make our model less capable of identifying classes with few data. So in the next step we will increase the moderate-demented data in a targeted manner.

3.2 Data Augmentation

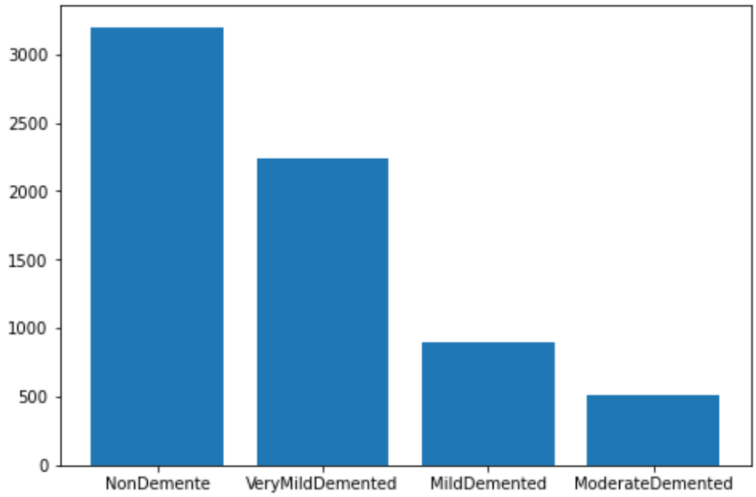
Image noise is a random variation of brightness or color information in the images captured. It is degradation in image signal caused by external sources. Images containing multiplicative noise have the characteristic that the brighter the area the noisier it is. But mostly it is additive.We can model a noisy image as $A(x,y) = H(x,y) + B(x,y)$ .Where, $A(x,y)$= function of noisy image, $H(x,y)$= function of image noise , $B(x,y)$= function of original image. Based on the above ideas, we derived four methods for extending Moderate Demented data, inverting the data, adding Gaussian noise,Salt and Pepper noise, and Random noise.

Based on this theory we found that our data has symmetry, if we reverse the ModerateDemented data, then the ModerateDemented data will increase and will not bring errors to the data. Subsequently, Gaussian Noise is a statistical noise having a probability density function equal to normal distribution, Random Gaussian function is added to Image function to generate this noise. Salt and Pepper noise is added to an image by addition of both random bright (with 255 pixel value) and random dark (with 0 pixel value) all over the image. Random noise is more like a variant of Gaussian noise, the difference is that Gaussian noise is colored and random noise is white.

After running the data through our four methods, we get new data.



From left to right are the data obtained after inversion, Gaussian noise data, pepper and salt noise data, and random noise data. The data of MonderateDemented has also increased from 64 to 555. Although we can continue to increase the data, if we blindly increase the data, it may add unnecessary errors to the subsequent models we create. The image below is our new data bar plot.



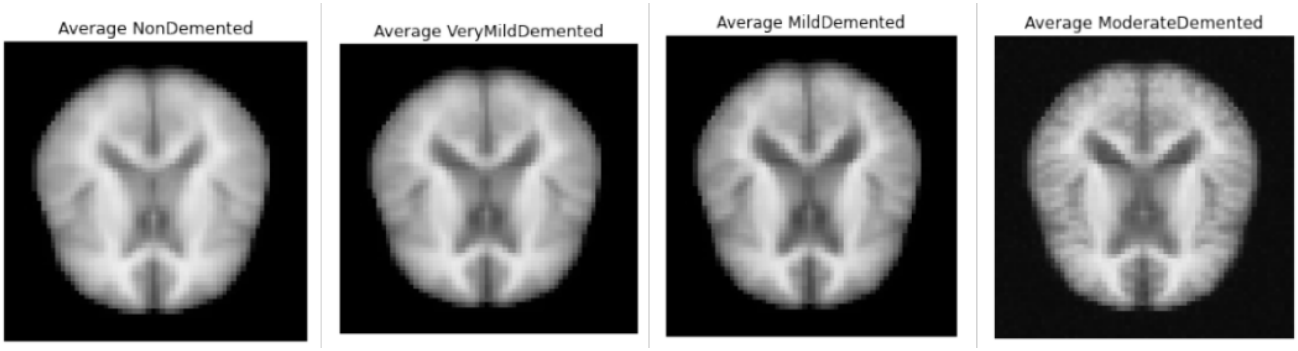
3.3 Principal component analysis (PCA)

In our case, Principal component analysis (PCA) provides a method to explore functional brain connectivity, which gives us some characteristic images.

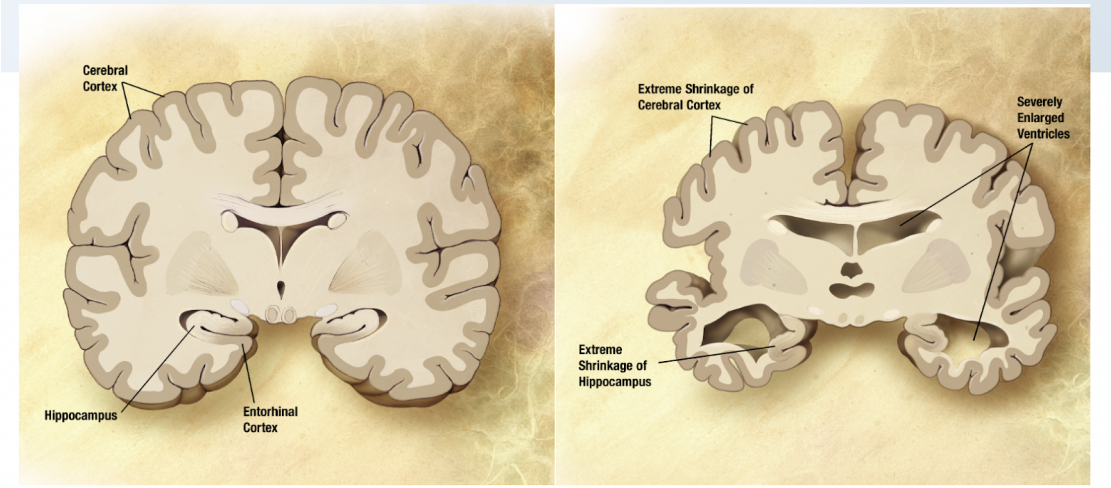
In general, PCA is important because it helps to find latent characteristic images among all the data, reduce the dimensionality, and visualize data easier and faster training because it uses less hardware to run.

What we did was to use eigenimages function to find out the pca components for each category and then plot eigenimages in a grid. We used the existing package to get a few dozen small images of each category. Since our data are two-dimensional images and the specifications are very consistent in colour and form, our resulting images of PCA present a few dozen small images of each category without yielding very strong and effective information. After arriving at the pca results, we had an initial knowledge of the exploration of the data at this point due to the specificity of our dataset, and therefore we did not continue to do PCA and put images here. If interested, the results have been shown in the code.

3.4 Mean Image



We derived the average images of the four categories by means of the averaging function. As we can see, from left to right, they show the degree of severity of being diagnosed as increasingly demented axial planes. It's actually a trend of degeneration of the brain when Alzheimer's becomes severe. As we research, the changes stand for the loss of neurons.



(From pcrm)

Compared to the above picture, it clearly shows that there is severe shrinkage in the cerebral cortex, ventricles, and hippocampus, which are associated with human memory. Especially for the hippocampus, it is a complex brain structure embedded deep into the temporal lobe that has a major role in learning and memory. Therefore, Our resulting image variations are

very similar to those of our study, which confirms the initial accuracy of our dataset and gives us the motivation to continue our research.

IV. Data Load and Data Preparation

4.1 Data Split and Set Parameters

After the noise process, we increases the instance of moderate demented to 384. Since the original dataset does not contain the train and test set, we split the data manually. We randomized images and divided them into 80%(train) and 20%(test) first. Then again, among the 80% of train, we extract 20% of it as validation. Thus we have 64% train, 20% test and 16% validation. After that, we split each set into 4 stages of AD as it should be. Then it comes to the parameters setup. We have 3 main parameters, the first one is the image size. The original image size is (128, 128). But in model 1, we convert the image size to (224, 224). For model 2 and 3, we use the original image size as (128, 128). The batch size is set as 64 and we use 80-100 epochs which depends on how complex the model is. At the end, we converted 4 different stages to a class array and extracted X, y.

V. Model Selection

After we finish the exploratory data analysis section, we go into the section of model prediction. We will train the model on the train X, y that we have split in the data preparation and evaluate the model on the test X, y.

5.1 NASNetMobile

Model Logic and Algorithm:

The first model we choose is called NasNetMobile, which is a model in the package of Tensorflow.Keras in python. It is a convolutional neural network that is trained on more than a million images from the ImageNet database. When training, the model searches for the best convolutional layer, and by stacking together more copies of this layer, NasNet is created. First, we load the base NasNetMobile model from Tensorflow.Keras , and open only the last 4 layers in the base model to be trainable. The default image input for NasNetMobile is (224, 224, 3), which means we need to resize the input image from (128, 128) to (224, 224)--this will cause the problem of increasement of noises. When we resize the image, the logic behind it is the computer estimates the pixels around the image and adds these pixels to the image in order to make it fulfill the new size (224, 224).

Base Model Built and Model Summary:

```
base_model = tf.keras.applications.NASNetMobile(input_shape=(224,224,3),
                                                 include_top=False, weights="imagenet")
```

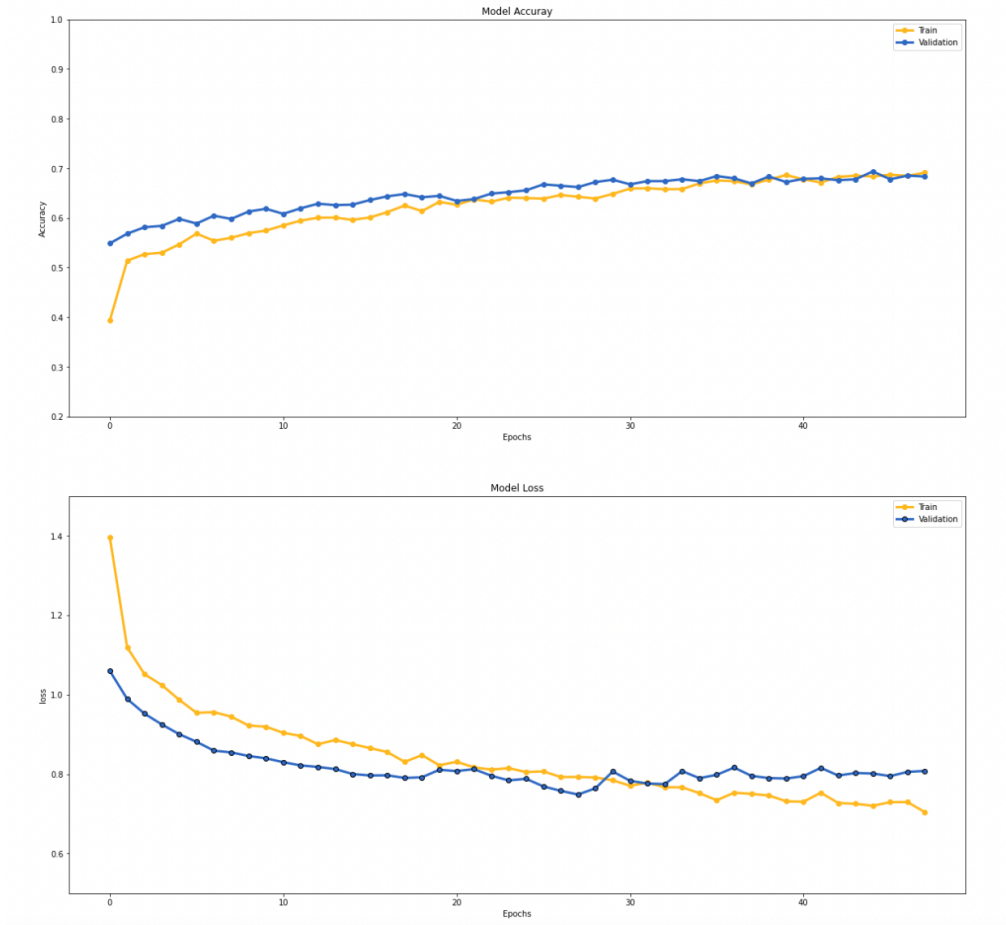
We designed the base model as the last 4 layers are trainable, and added some layers of batch normalization, Activation('relu'), 0.5 dropout rate, dense (kernel_initializer='he_uniform'). We set epoch equals 80, printing metrics (accuracy, loss, and so on), and early stopping methods.

Model Result:

For Train Dataset:

The model stopped training at 48/80 epoches with loss of 0.7041 and accuracy rate of 0.6912.

From the below Model Accuracy and Model Loss line plot (Figure 5.1.1), we can see the accuracy of NasNetModel increased in the beginning of the training and the growth was slowing down afterwards. And the final accuracy is around 0.69, not as good as we expected. For Model Loss, we can see in the graph, during 30 epoch to 50 epoch, the loss of validation dataset did not decrease.

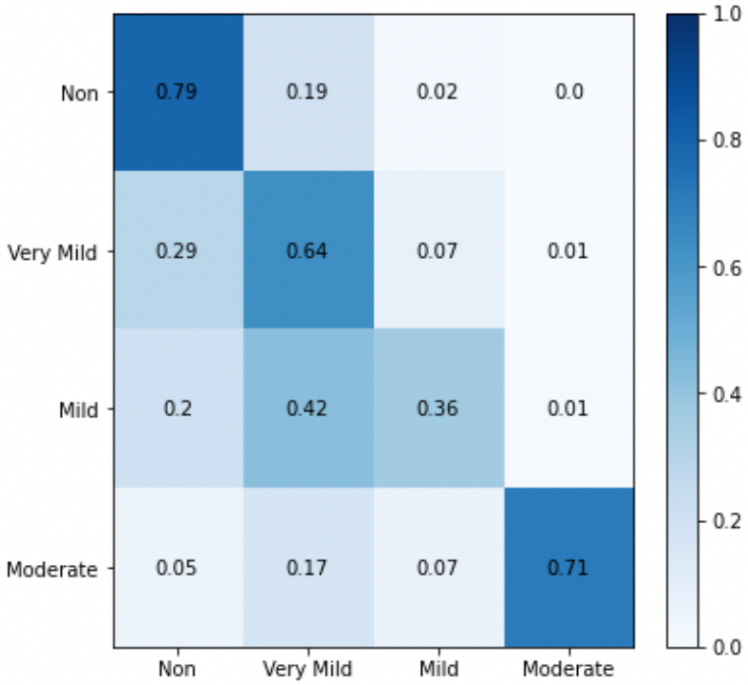


(Figure 5.1.1. Model Accuracy and Model Loss for NasnetMobile with 4 layers Trainable)

For Test Dataset:

When the model was evaluated on the Test Dataset, it achieved 0.6783 accuracy and 0.8559 loss, more precisely it has 0.7302 precision, 0.6046 recall, 0.8931 auc, and 0.6603 f1 score.

From the confusion matrix below (Figure 5.1.2), Non Demented and Moderate are relatively well predicted, while very mild and mild are not well distinguished.



(Figure 5.1.2. Confusion Matrix for NasnetMobile with 4 layers Trainable)

Since the accuracy is around 0.69 and very mild and mild categories are not well predicted, we decide to open 2 more layers, so the last 6 layers in the base model are trainable.

Updated NasNetMobile Model Summary:

```
base_model = tf.keras.applications.NASNetMobile(input_shape=(224,224,3),
                                                 include_top=False, weights="imagenet")
```

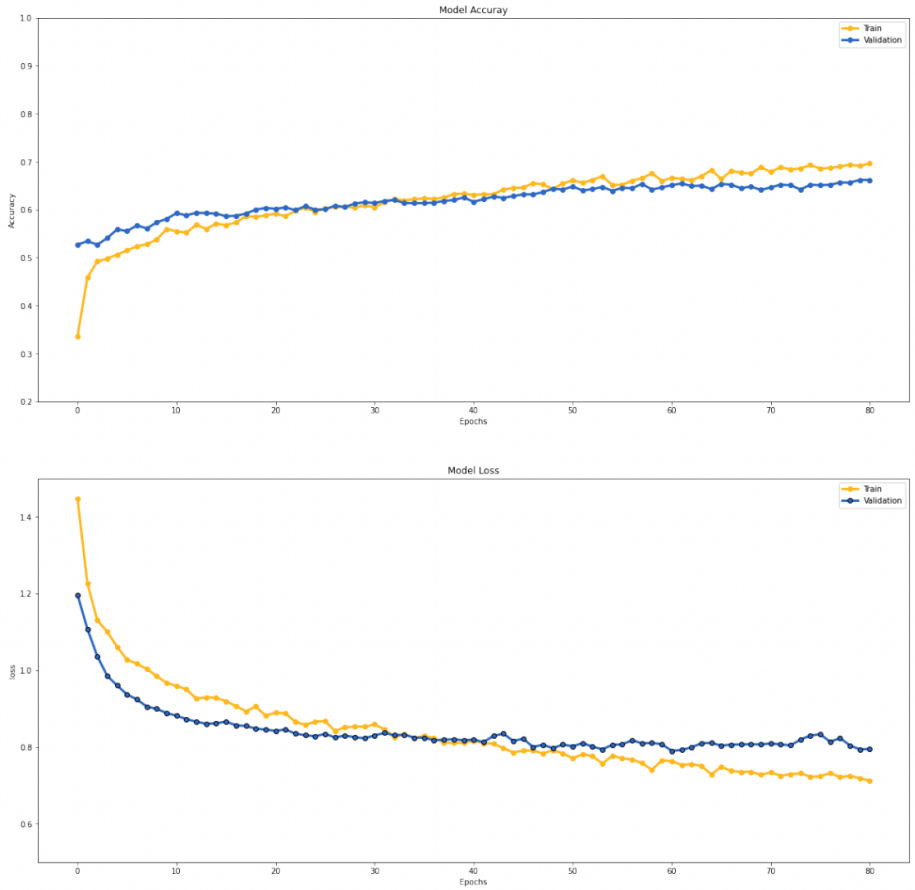
The updated NasNetMobile Model is similar to the previous one, but the last layers in the base model are trainable right now.

Model result:

For Train Dataset:

The model stopped training at 81/100 epoches with loss of 0.7125 and accuracy rate of 0.6963.

From the below Model Accuracy and Model Loss line plot (Figure 5.1.3), we can see even though we have opened two more layers in the model to be trainable, the accuracy is still under 0.7. For Model Loss, the result is even worse in the beginning epochs and also is not improved in the end.



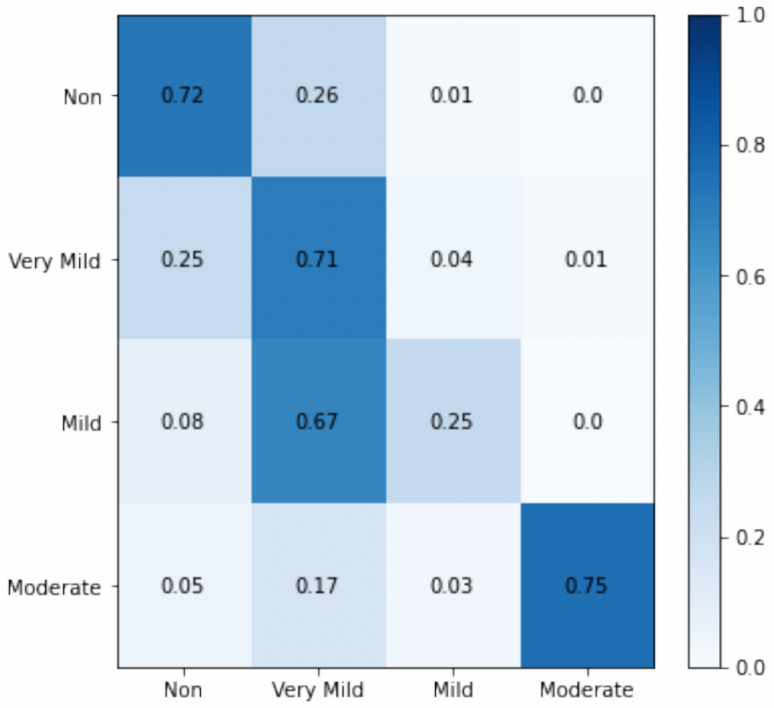
(Figure 5.1.3. Model Accuracy and Model Loss for NasnetMobile with 6 layers Trainable)

For Test Dataset:

When the model was evaluated on the Test Dataset, it achieved 0.6589 accuracy and 0.8592 loss, more precisely it has 0.6741 precision, 0.6180 recall, 0.8840 auc, and 0.6449 f1 score.

From the confusion matrix below (Figure 5.1.4), Non Demented and Moderate are still relatively well predicted, while very mild and mild are not well distinguished.

And a very serious problem should be indicated here is that NasNetMobile Model wrongly predicted Mild Demented as Very Mild Demented, which could be very dangerous and unacceptable.



(Figure 5.1.4. Confusion Matrix for NasnetMobile with 6 layers Trainable)

Model Conclusion:

The accuracy is around 0.69 for both NasNetMobile models and opening new layers does not improve the classification a lot. The model is rather accurate in differentiating Non and Moderate, but is not as accurate in differentiating Very Mild and Mild.

If we open more layers, the accuracy may increase, but the model will consume much more time to be trained.

Since the default image input size is (224, 224, 3), when we resize the image to satisfy the default input size, we add more noise to the image.

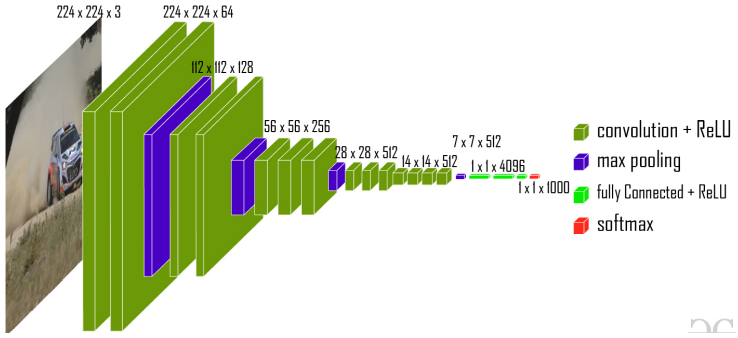
NasNetMobile is trained on ImageNet dataset, so it might not be the correct model for our dataset. We could try NasNetLarge instead of NasNetMobile in further studies.

5.2 VGG16

VGG-16 is a convolutional neural network developed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab(VGG) at Oxford University in 2014 in “Very Deep Convolutional Networks for Large-Scale Image Recognition.” The model is trained on the ImageNet dataset.

Model Logic and Algorithm:

VGG-16 has a simple and clean architecture, with 13 convolutional layers, 5 max-pooling layers, and 3 fully connected layers. It all uses 3*3 filters.

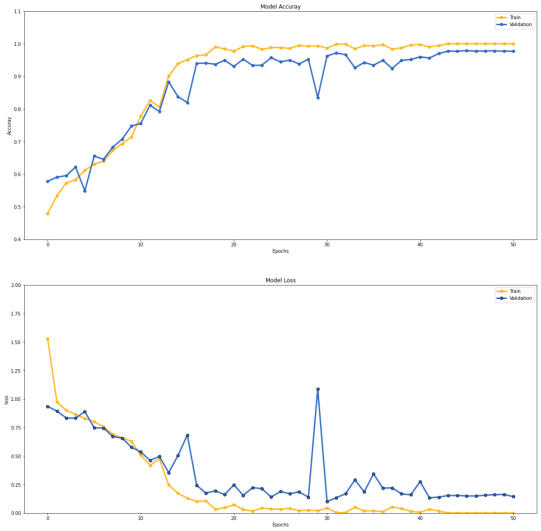


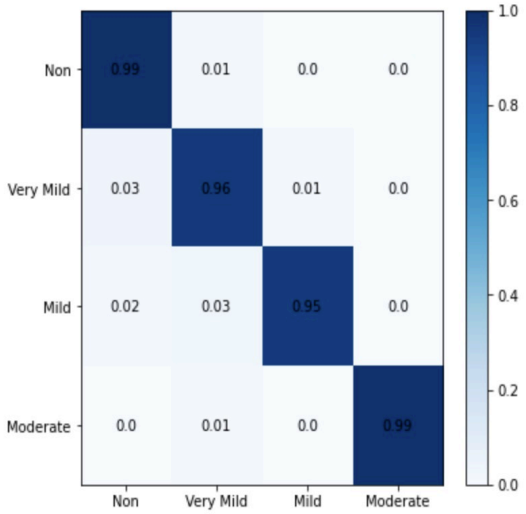
(Figure 5.2.1 VGG16 Architecture)

We use the VGG-16 package from Keras. In this model, we unfreeze the last 6 layers to be trainable since VGG is a time-consuming model. In addition, we add a fully connected layer using ReLU, an output prediction layer using the sigmoid with a 0.2 dropout rate. The default input size is (224, 224, 3). But we can change it to (128, 128, 3) unless we unfreeze all the VGG16 layers. Here, (128, 128, 3) fits better for our image size. Next, we compile using the “adam” optimizer and apply early stopping to avoid overfitting.

Model Result:

The model early stopped at epochs 61/100. For the training dataset, VGG has reached an accuracy of 0.9772 and a loss of 0.1432. The test dataset has a relatively high accuracy of 0.9744 and a loss of 0.1089. The below confusion matrix plot shows that it performed very well in classifying 4 different labels.





(Figure 5.2.2. Model Accuracy and Model Loss for VGG16 with 6 layers Trainable)

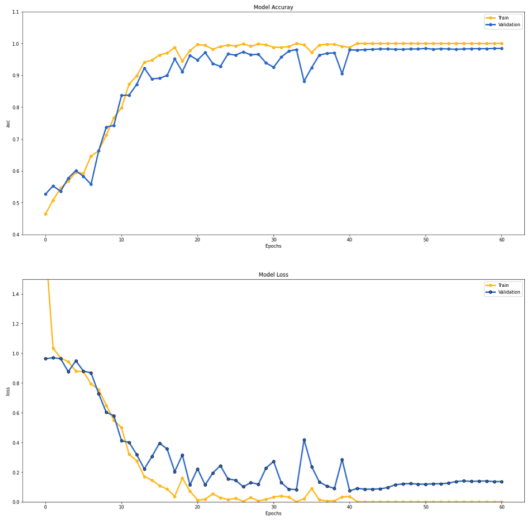
(Figure 5.2.3. Confusion Matrix for VGG16 with 6 layers Trainable)

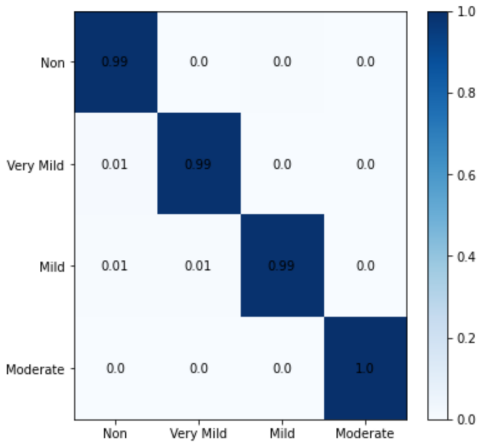
Model Improvement:

From the model loss plot, we found that there is a validation loss point that is relatively higher than others. It might be a noise or a mistake caused by incorrectly using sigmoid function. Softmax is appropriate for multi classification while sigmoid is for binary classification. Therefore modify the sigmoid to softmax. Next we unfreeze one more layer to be trainable to see if there is further improvement.

Model Result:

Compared with the initial VGG, this model's accuracy has been improved from 0.9772 to 0.9845 on the training dataset and from 0.9744 to 0.9912 on the test. The Model Loss performance of the validation set has also improved. Loss has reduced from 0.1432 to 0.1368 on the training set, and from 0.1089 to 0.0401on the test dataset. Overall, the VGG16 model performs better.





(Figure 5.2.4. Model Accuracy and Model Loss for VGG16 with 7 layers Trainable)

(Figure 5.2.5. Confusion Matrix for VGG16 with 7 layers Trainable)

5.3 CNN

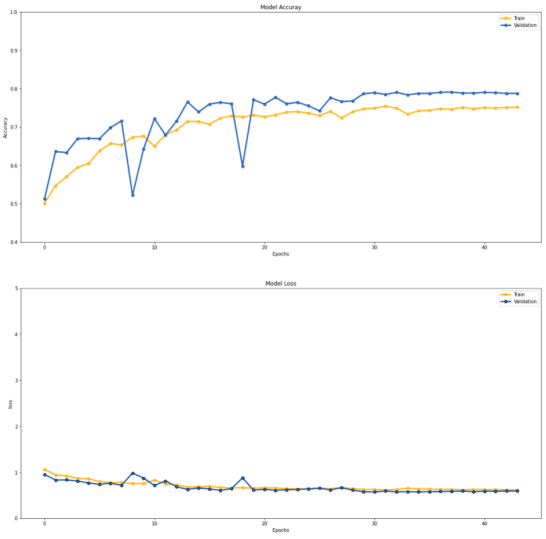
Model Algorithm and Set Up:

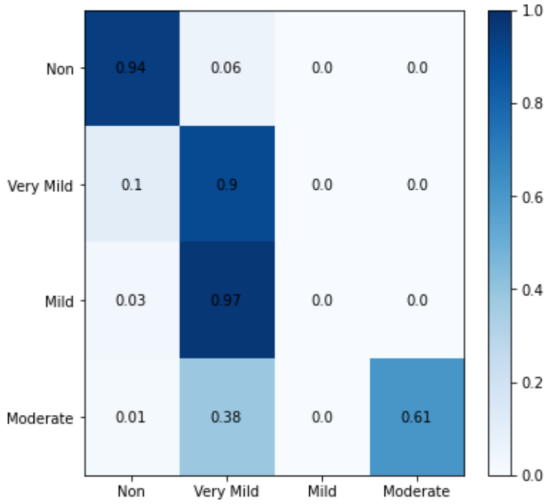
We build this model using a typical CNN architecture. Using Keras, we built 2 ReLU layer convolutional layer with 3*3 filters, add batch normalization and MaxPooling2D layers, and fully connected layers with ReLU. To reduce overfitting, each fully connected layer is

followed by a dropout layer with 0.2 dropout rate. Lastly, We compile our first simple CNN with SGD optimizer and set learning rate as 0.001. Lastly, we apply early stopping to avoid overfitting.

Model Result:

The 2-layer CNN model only get an accuracy of 0.7874 and a loss of 0.6178 on the training dataset. Similarly on the test dataset, we get an accuracy of 0.7825 and a loss of 0.5981. From the confusion matrix below (Figure 5.3.2), Non Demented and very mild is relatively well predicted, however, moderate and mild are not well predicted, in particular our model has never predicted Mild Demented correctly, which has the same problem in our NasNetMobile model. Thus, we consider increasing the trainable layer and adjust the input size for improvement.





(Figure 5.3.1. Model Accuracy and Model Loss for CNN with 2 layers)

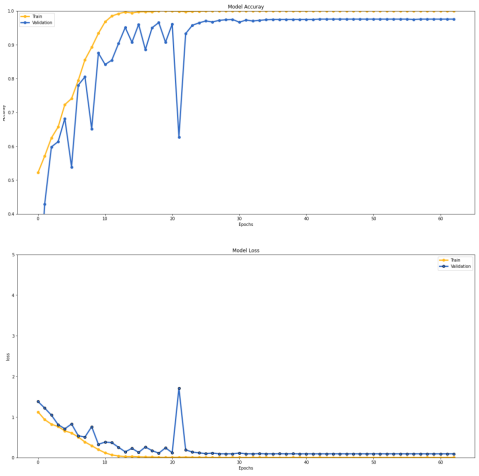
(Figure 5.3.2. Confusion Matrix for CNN with 2 layers)

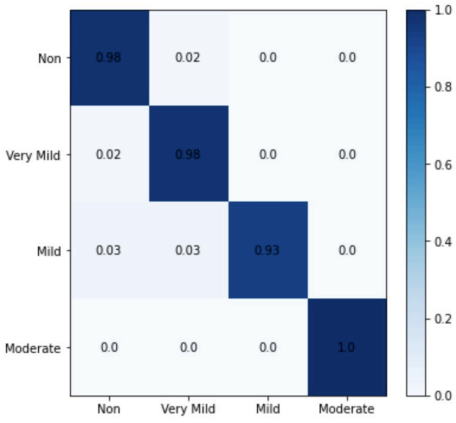
Model Improvement:

We add 4 more ReLU convolutional layers with a 7*7 filter and reduce the dropout rate by 0.05 to contain more information. Lastly we adjust the input size to (128, 128, 3) to fit our original image size.

Model Result:

Our new model stopped early at the epoch 63/10. It reaches a high accuracy of 0.9754 and a loss of 0.0934 on the train dataset. The model prediction on the test dataset also gives us a good result, the category accuracy is relatively high to 0.9744. The loss reduces to 0.0827. The confusion matrix of our 6-layer CNN model is similar to the VGG-16 model; it performs very well in classifying 4 different labels.





(Figure 5.3.3. Model Accuracy and Model Loss for CNN with 2 layers)

(Figure 5.3.4. Confusion Matrix for CNN with 2 layers)

5.4 Model Conclusion

NasNetMobile models have accuracies around 0.69 and opening 2 more layers does not improve the classification a lot. Resizing the image increases lots of noise. The model is rather accurate in differentiating Non and Moderate, but is not as accurate in differentiating Very Mild and Mild.

The VGG16 model performed the best among these three models, we got an accurate prediction score of 0.97 on the test data. The model can accurately predict all types of labels. It's a well-performed model. But training VGG16 is time-consuming and has a high memory usage.

The CNN model has 0.97 on the test data. Surprisingly, accuracy has reached 0.97 on the test dataset by simply building 6 convolutional layers. It’s also time-consuming. For further interpretation, we may add possible layers to the model to visualize which part of our image region is significant.

VI. Conclusion

During exploratory analysis through Alzheimer’s MRI brain images, we found out that degeneration of the cerebral cortex is a prominent feature of Alzheimer's disease, and we also noticed that there is a trend of degeneration of the brain when Alzheimer’s becomes severe, e.g. loss of neurons, affected hippocampus. When we observed the imbalance between four AD categories–NonDemented, Very Mild Demented, Mild Demented, and a relatively small number of Moderate, we did data augmentation by flipping images and adding noises to images while the key features were not touched.

Applying three different deep learning models, our classification of Alzheimer's stages by analyzing the MRI brain images seems to succeed. The three neural network models, which are NasNetMobile, VGG16, and CNN, have accuracies 0.69, 0.99 and 0.97 respectively.

VII. Shortcomings and Suggestions for Further Study

Dataset Original Problems:

Firstly, the dataset is not up to date. The data was published back in 2014. Given that Alzheimer's is a long-term disease requiring more stable and robust technical support, the dataset may currently lag behind in time on today's detection technology or different approved standards exist. Secondly, The colours of image data are rather monochromatic, reducing the possibilities for data mining, and data visualization. Thirdly, there exists uneven distribution of image data. Among the four categories, there exists large differences in the amount of data。 Especially, the number of data in category ‘Moderate Demented’ is much smaller than other categoris, which would make the model less accurate. At this point, we have reduced the effects of the problems by adding noise to some original images.

Team’s Challenges:

Due to lack of medical expertise, the criteria used to differentiate our selected images into four categories are somewhat biased. Thus, finding valid information from image recognition can be more difficult than with other readily available data. Since we only have the image data, we need to find our own features to classify the criteria. Besides, Alzheimer's disease background information takes time to learn and time may not be enough. Lastly, Image recognition is

relatively unfamiliar for us and we do not know a lot about the corresponding method and all of these take time to learn and practice.

Future Improvements:

Our aim is to improve the model's ability to classify the four disease categories. What we want to achieve is to do a better segmentation of the images into the different categories. At this point, we made it successfully to improve the evaluation of the test image and enhance the confusion matrix.

This project was planned very close in time and the background knowledge required was deep, so if we had more time I believe we could have done better, especially in trying to build the model by ourselves instead of importing the package very often; also we wish to find a more balanced dataset for training, especially to give more data images for very mild and moderate categories with more themes.

VIII. References

[1] Oscar Darias Plasencia. (2021). Alzheimer Diagnosis with Deep Learning:Model Implementation.https://towardsdatascience.com/alzheimer-diagnosis-with-deep-learning-model-implementation-5a0fd31f148f
[2] YASIR HUSSEIN SHAKIR., Alzheimer's Classification - NASNetMobile. https://www.kaggle.com/code/yasserhessein/alzheimer-s-classification-nasnetmobile
[3] Davide Bombassei de Bona, Federica Boscolo, Maria Dalle Vacche, Antonio Marittimi, Marco Pinamonti. (2022). Alzheimer MRI Model - group 7. https://www.kaggle.com/code/davidebombassei/alzheimer-mri-model-group-7#Second-Try:-Use-a-Dataset-where-train,-test-and-validation-sets-are-divided-on-subject-basis
[4] Murugan, S., Venkatesan, C., Sumithra, M. G., Gao, X.-Z., Elakkiya, B., Akila, M., & Manoharan, S. (2021). DEMNET: A deep learning model for early diagnosis of alzheimer diseases and dementia from Mr Images. IEEE Access, 9, 90319–90329. https://doi.org/10.1109/access.2021.3090474

[5] Mofrad, S. A., Lundervold, A. J., Vik, A., & Lundervold, A. S. (2021). Cognitive and MRI trajectories for prediction of alzheimer's disease. Scientific Reports, 11(1). https://doi.org/10.1038/s41598-020-78095-7
[6] Chandra, A., Dervenoulas, G., & Politis, M. (2018). Magnetic Resonance Imaging in alzheimer's disease and mild cognitive impairment. Journal of Neurology, 266(6), 1293–1302. https://doi.org/10.1007/s00415-018-9016-3
[7] Li, W., Zhao, Y., Chen, X., Xiao, Y., & Qin, Y. (2019). Detecting alzheimer's disease on small dataset: A knowledge transfer perspective. IEEE Journal of Biomedical and Health Informatics, 23(3), 1234–1242. https://doi.org/10.1109/jbhi.2018.2839771
[8] https://www.kaggle.com/yasserhessein/dataset-alzheimer and https://www.kaggle.com/datasets/sachinkumar413/alzheimer-mri-dataset
[9]VGG-16 Architecture. Geeksforgeeks. https://www.geeksforgeeks.org/vgg-16-cnn-model/
[10] GeÌ ron, Aureì lien. 2019. Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd ed. CA 95472: O'Reilly.
[11] Simonyan, Karen, and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition" (2014).

Appendix

Python code files

NEW model

June 8, 2022

1 Environment Set up

Set up combination

[1]:
```
import os
from os.path import join
from glob import glob
import numpy as np
import pandas as pd
import tensorflow as tf

import matplotlib.pyplot as plt
import PIL
import random

from tensorflow.keras.callbacks import ReduceLROnPlateau

import cv2 as cv

from sklearn.metrics import accuracy_score, balanced_accuracy_score,␣
↪roc_auc_score, \
    confusion_matrix, precision_score, recall_score, f1_score

from tensorflow.keras.models import Sequential
from keras.models import Sequential,load_model
from tensorflow.keras.layers import InputLayer, BatchNormalization, Dropout,␣
↪Flatten, Dense, Activation, MaxPool2D, Conv2D
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.callbacks import Callback,ModelCheckpoint,ReduceLROnPlateau
from keras import backend as K

import copy
import warnings
warnings.filterwarnings('ignore')

from keras.preprocessing.image import load_img, img_to_array
import matplotlib
```

```
import seaborn as sns
from sklearn.utils import shuffle

from sklearn.decomposition import PCA
from math import ceil
from tensorflow.keras.preprocessing import image
%matplotlib inline

import tensorflow.keras
import random
from tensorflow.keras.layers import *
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import *
from tensorflow.keras.losses import *
from tensorflow.keras.optimizers import *

from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,␣
↪Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,␣
↪ModelCheckpoint

from matplotlib import cm
from matplotlib import axes
```

Set up 1 - Data load - Cassie

[2]:
```
import os
from os.path import join
from glob import glob
import numpy as np
import pandas as pd
import tensorflow as tf

import matplotlib.pyplot as plt
import PIL
import random

from tensorflow.keras.callbacks import ReduceLROnPlateau

import cv2 as cv

from sklearn.metrics import accuracy_score, balanced_accuracy_score,␣
↪roc_auc_score, \
    confusion_matrix, precision_score, recall_score, f1_score
```

```
import seaborn as sns
```

Set up 2 - Model 1 - Cassie

[3]:
```
from tensorflow.keras.models import Sequential
from keras.models import Sequential,load_model
from tensorflow.keras.layers import InputLayer, BatchNormalization, Dropout,␣
 ↪Flatten, Dense, Activation, MaxPool2D, Conv2D
from keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.callbacks import Callback,ModelCheckpoint,ReduceLROnPlateau
from keras import backend as K
```

Set up 3

[4]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
 ↪all files under the input directory

import os


import copy
import warnings
warnings.filterwarnings('ignore')

import cv2

from keras.preprocessing.image import load_img, img_to_array
import matplotlib
import matplotlib.pylab as plt
import numpy as np
import seaborn as sns
from sklearn.utils import shuffle

from sklearn.decomposition import PCA
from math import ceil
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
%matplotlib inline
```

Set up 4 - accuracy/loss curves

[5]:
```
from matplotlib import pyplot as plt
```

Set up 5 VGG

[6]:
```
import tensorflow.keras
import tensorflow as tf
import os
import numpy as np
import random
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.layers import *
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import *
from tensorflow.keras.losses import *
from tensorflow.keras.optimizers import *
```

Set up 6 CNN

[7]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,␣
↪Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,␣
↪ModelCheckpoint

import os
import cv2
import matplotlib.pyplot as plt
import seaborn as sns
```

Set up 7 Model Heatmap – Cassie

[8]:
```
from matplotlib import cm
from matplotlib import axes
```

2 Load Datasets

[9]:
```
# define two functions to load 4 types of data
def import_data(directory, foldername, y, typename):
    """
    this funtion ...
    """
    data_df = pd.DataFrame({
```

```
        "X": sorted(glob(join(directory, foldername, "*"))),
        "y": y,
        "type": typename
    })
    return data_df
```

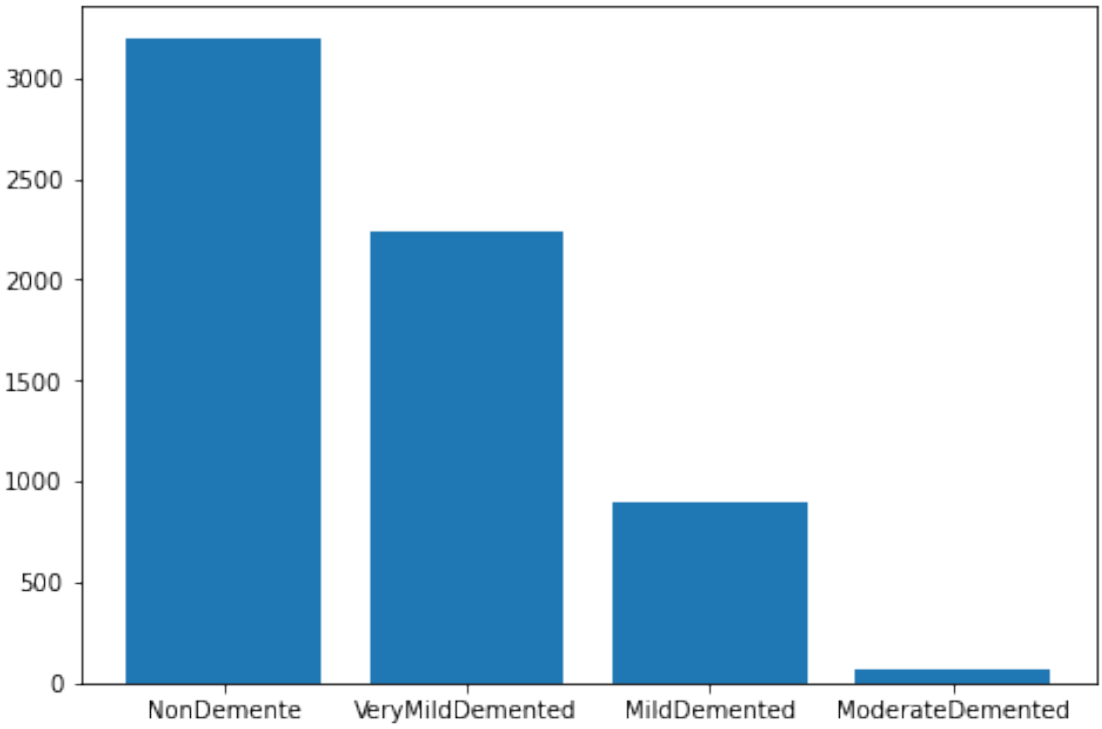
[111]:
```
# set the directory of data import

directory = "Dataset"

# build four dataframe
non_demented_df = import_data(directory, "./NonDemented", 0, "NonDemented")
very_mild_df = import_data(directory, "./VeryMildDemented", 1,␣
↪"VeryMildDemented")
mild_df = import_data(directory, "./MildDemented", 2, "MildDemented")
moderate_df = import_data(directory, "./ModerateDemented", 3,␣
↪"ModerateDemented")
```

3 EDA section

1. count 4 types + bar plot
2. flip and add noise to increase the number "moderate" images
3. count 4 types + bar plot + show: flip, noise example images
4. average 4 type images + PCA

[112]:
```
# show class imbalance 1
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
size = [len(non_demented_df),len(very_mild_df),len(mild_df),len(moderate_df)]
ax.bar(('NonDemente','VeryMildDemented','MildDemented','ModerateDemented'),size)
plt.show
```

[112]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[121]:
```
# create new folders named "NEW"
from os import listdir
from matplotlib import image
# load all images in a directory
# create flipped versions of an image
from PIL import Image
from matplotlib import pyplot

loaded_images = list()
for filename in listdir('Dataset/ModerateDemented'):
    image = Image.open('Dataset/ModerateDemented/' + filename)
    hoz_flip = image.transpose(Image.FLIP_LEFT_RIGHT)
    hoz_flip.save('Dataset/ModerateDemented/F_MRI' + filename)#*****
```

[124]:
```
# after combine newdata and original data step 1
# random noise
def random_noise(image,noise_num):
    img_noise = image
    rows,cols,chn = img_noise.shape

    for i in range(noise_num):
        x = np.random.randint(0 , rows)
        y = np.random.randint(0,cols)
```

```
        img_noise[x,y,:] = 255
    return img_noise
# salt and pepper
def sp_noise(noise_img, proportion):
    height,width = noise_img.shape[0],noise_img.shape[1]
    num = int(height*width*proportion)
    for i in range(num):
        w = random.randint(0,width - 1)
        h = random.randint(0, height - 1)
        if random.randint(0,1) == 0:
            noise_img[h,w] = 0
        else:
            noise_img[h,w] = 255
    return noise_img

# gaussian_noise
def gaussian_noise(img,mean,sigma):
    img = img / 255
    noise = np.random.normal(mean, sigma,img.shape)
    gaussian_out = img + noise
    gaussian_out = np.clip(gaussian_out,0,1)
    gaussian_out = np.uint8(gaussian_out*255)
    return gaussian_out
# clear newdata file then running below code 3 times
def convert(input_dir,output_dir):
    for filename in os.listdir(input_dir):
        path = input_dir + "/" + filename
        print("doing... ", path)
        noise_img = cv.imread(path)
        #img_noise = gaussian_noise(noise_img,0,0.3)
        #img_noise = sp_noise(noise_img, 0.01)
        img_noise = random_noise(noise_img,3000)
        Nfilename = filename + 'RA.jpg'
        cv.imwrite(output_dir + '/' +  Nfilename ,img_noise)

data_path = 'Dataset/ModerateDemented'
convert('Dataset/ModerateDemented','NEW')
# repate step 1
```

```
doing…  Dataset/ModerateDemented/F_MRImoderate_14.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_28.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_3.jpg
doing…  Dataset/ModerateDemented/moderate_55.jpg
doing…  Dataset/ModerateDemented/moderate_41.jpg
doing…  Dataset/ModerateDemented/moderate_40.jpg
doing…  Dataset/ModerateDemented/moderate_54.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_2.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_29.jpg
```

```
doing… Dataset/ModerateDemented/F_MRImoderate_15.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_17.jpg
doing… Dataset/ModerateDemented/moderate_42.jpg
doing… Dataset/ModerateDemented/moderate_56.jpg
doing… Dataset/ModerateDemented/moderate_57.jpg
doing… Dataset/ModerateDemented/moderate_43.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_16.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_5.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_12.jpg
doing… Dataset/ModerateDemented/moderate_47.jpg
doing… Dataset/ModerateDemented/moderate_53.jpg
doing… Dataset/ModerateDemented/moderate_52.jpg
doing… Dataset/ModerateDemented/moderate_46.jpg
doing… Dataset/ModerateDemented/moderate.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_13.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_4.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_39.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_6.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_11.jpg
doing… Dataset/ModerateDemented/moderate_50.jpg
doing… Dataset/ModerateDemented/moderate_44.jpg
doing… Dataset/ModerateDemented/moderate_45.jpg
doing… Dataset/ModerateDemented/moderate_51.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_10.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_38.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_7.jpg
doing… Dataset/ModerateDemented/F_MRImoderate.jpg
doing… Dataset/ModerateDemented/moderate_7.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_63.jpg
doing… Dataset/ModerateDemented/moderate_36.jpg
doing… Dataset/ModerateDemented/moderate_22.jpg
doing… Dataset/ModerateDemented/moderate_23.jpg
doing… Dataset/ModerateDemented/moderate_37.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_62.jpg
doing… Dataset/ModerateDemented/moderate_6.jpg
doing… Dataset/ModerateDemented/moderate_4.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_60.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_48.jpg
doing… Dataset/ModerateDemented/moderate_21.jpg
doing… Dataset/ModerateDemented/moderate_35.jpg
doing… Dataset/ModerateDemented/moderate_34.jpg
doing… Dataset/ModerateDemented/moderate_20.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_49.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_61.jpg
doing… Dataset/ModerateDemented/moderate_5.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_59.jpg
doing… Dataset/ModerateDemented/moderate_18.jpg
doing… Dataset/ModerateDemented/moderate_24.jpg
```

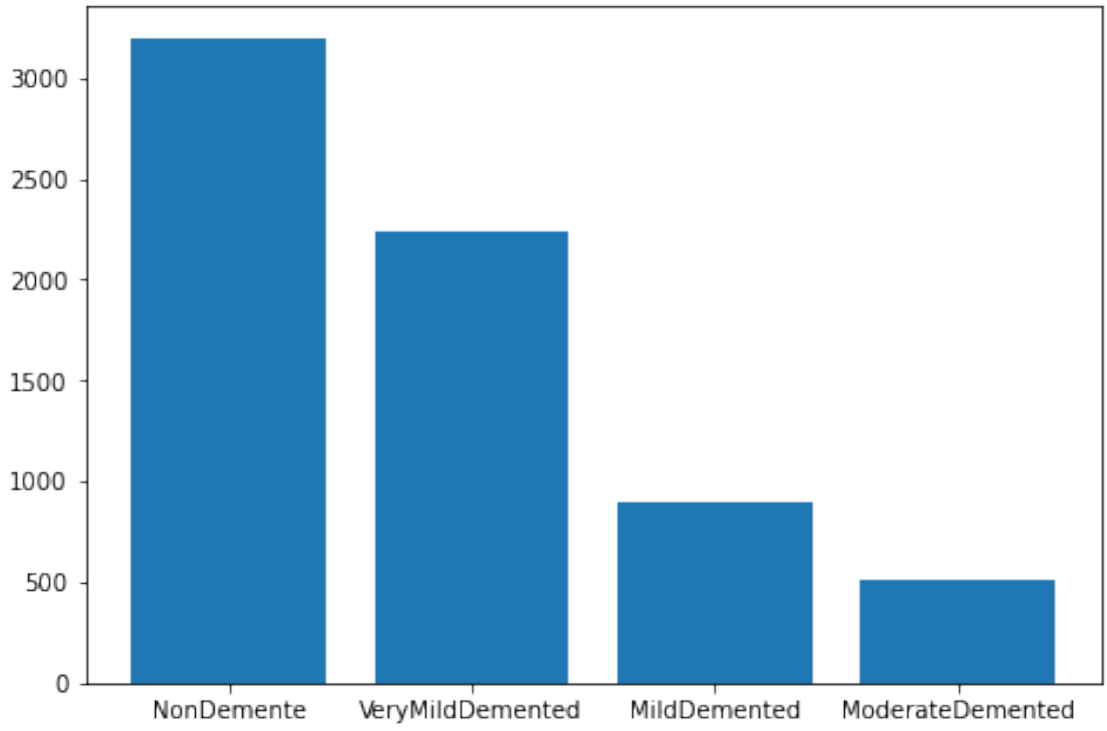
```
doing… Dataset/ModerateDemented/moderate_30.jpg
doing… Dataset/ModerateDemented/moderate_31.jpg
doing… Dataset/ModerateDemented/moderate_25.jpg
doing… Dataset/ModerateDemented/moderate_19.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_64.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_58.jpg
doing… Dataset/ModerateDemented/moderate_2.jpg
doing… Dataset/ModerateDemented/moderate_33.jpg
doing… Dataset/ModerateDemented/moderate_27.jpg
doing… Dataset/ModerateDemented/moderate_26.jpg
doing… Dataset/ModerateDemented/moderate_32.jpg
doing… Dataset/ModerateDemented/moderate_3.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_56.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_42.jpg
doing… Dataset/ModerateDemented/moderate_17.jpg
doing… Dataset/ModerateDemented/moderate_16.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_43.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_57.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_41.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_55.jpg
doing… Dataset/ModerateDemented/moderate_14.jpg
doing… Dataset/ModerateDemented/moderate_28.jpg
doing… Dataset/ModerateDemented/moderate_29.jpg
doing… Dataset/ModerateDemented/moderate_15.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_54.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_40.jpg
doing… Dataset/ModerateDemented/moderate_8.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_44.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_50.jpg
doing… Dataset/ModerateDemented/moderate_39.jpg
doing… Dataset/ModerateDemented/moderate_11.jpg
doing… Dataset/ModerateDemented/moderate_10.jpg
doing… Dataset/ModerateDemented/moderate_38.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_51.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_45.jpg
doing… Dataset/ModerateDemented/moderate_9.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_53.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_47.jpg
doing… Dataset/ModerateDemented/moderate_12.jpg
doing… Dataset/ModerateDemented/moderate_13.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_46.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_52.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_35.jpg
doing… Dataset/ModerateDemented/F_MRImoderate_21.jpg
doing… Dataset/ModerateDemented/moderate_60.jpg
doing… Dataset/ModerateDemented/moderate_48.jpg
doing… Dataset/ModerateDemented/moderate_49.jpg
doing… Dataset/ModerateDemented/moderate_61.jpg
```

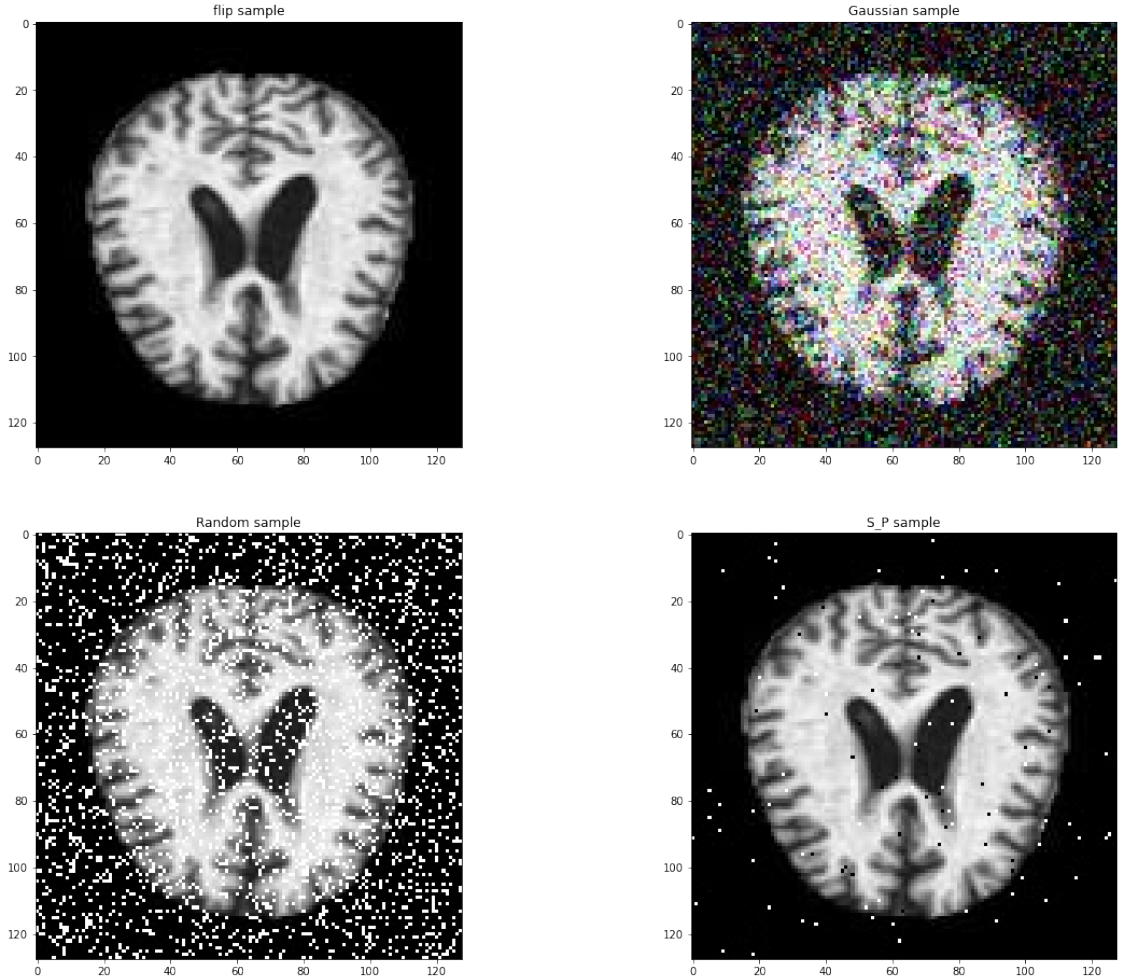
```
doing…  Dataset/ModerateDemented/F_MRImoderate_20.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_34.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_22.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_9.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_36.jpg
doing…  Dataset/ModerateDemented/moderate_63.jpg
doing…  Dataset/ModerateDemented/moderate_62.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_8.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_37.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_23.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_27.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_33.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_32.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_26.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_18.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_30.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_24.jpg
doing…  Dataset/ModerateDemented/moderate_59.jpg
doing…  Dataset/ModerateDemented/moderate_64.jpg
doing…  Dataset/ModerateDemented/moderate_58.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_25.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_31.jpg
doing…  Dataset/ModerateDemented/F_MRImoderate_19.jpg
```

After we did the #2, we could reimport the “moderate”, and show #3, then do #4.

[143]:
```
#3.1
directory = "Dataset"

# build four dataframe
non_demented_df = import_data(directory, "./NonDemented", 0, "NonDemented")
very_mild_df = import_data(directory, "./VeryMildDemented", 1,␣
 ↪"VeryMildDemented")
mild_df = import_data(directory, "./MildDemented", 2, "MildDemented")
moderate_df = import_data(directory, "./ModerateDemented", 3,␣
 ↪"ModerateDemented")

#3.2
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
size = [len(non_demented_df),len(very_mild_df),len(mild_df),len(moderate_df)]
ax.bar(('NonDemente','VeryMildDemented','MildDemented','ModerateDemented'),size)
plt.show


#3.3
plt.figure(figsize=(20, 16))
# choose 4 types new datasets
```

```
images_path = ['/ModerateDemented/F_MRImoderate_2.jpg', '/ModerateDemented/
 ↪F_MRImoderate_2.jpgGA.jpg',
               '/ModerateDemented/F_MRImoderate_2.jpgRA.jpg', '/
 ↪ModerateDemented/F_MRImoderate_2.jpgSP.jpg']
# show MRI
categories = ['flip sample', 'Gaussian sample', 'Random sample', 'S_P sample']
for i in range(4):
    ax = plt.subplot(2, 2, i + 1)
    img = cv.imread(directory + images_path[i])
    img = cv.resize(img, (128, 128))
    plt.imshow(img)
    plt.title(categories[i])
```





[134]:
```
W = 128 # The default size for ResNet is 224 but resize to .5 to save memory␣
 ↪size
H = 128 # The default size for ResNet is 224 but resize to .5 to save memory␣
 ↪size

label_to_class = {
    'MildDemented': 0,
    'ModerateDemented': 1,
    'NonDemented': 2,
    'VeryMildDemented':3
}
class_to_label = {v: k for k, v in label_to_class.items()}
n_classes = len(label_to_class)

def get_images(dir_name='Dataset', label_to_class=label_to_class):
    """read images / labels from directory"""
```

```
    Images = []
    Classes = []


    for label_name in os.listdir(dir_name)[1:5]:
            cls = label_to_class[label_name]

            for img_name in os.listdir('/'.join([dir_name, label_name])):
                img = load_img('/'.join([dir_name, label_name, img_name]),␣
 ↪target_size=(W, H))
                img = img_to_array(img)
                Images.append(img)
                Classes.append(cls)

    Images = np.array(Images, dtype=np.float32)
    Classes = np.array(Classes, dtype=np.float32)
    Images, Classes = shuffle(Images, Classes, random_state=0)

    return Images, Classes
```

[135]:
```
Images, Classes = get_images()
Images.shape, Classes.shape
```

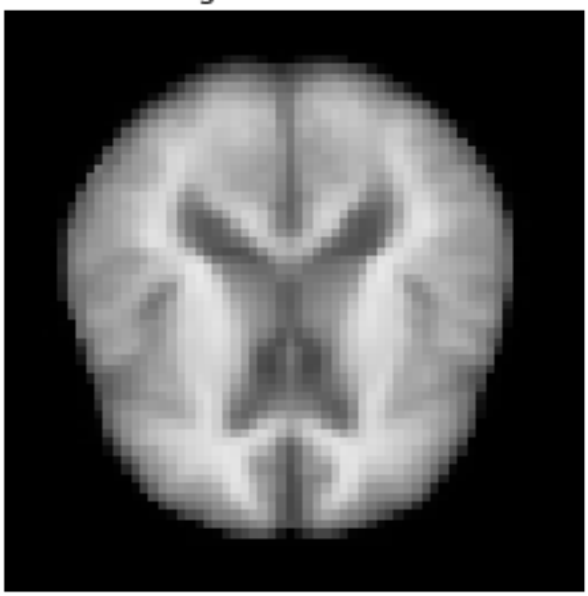
[135]:
```
((6851, 128, 128, 3), (6851,))
```

[136]:
```
# making n X m matrix
def img2np(dir_name, label,size = (64, 64)):
    # iterating through each file
    for label_name in os.listdir(dir_name):
            if(label_name==label):
                for img_name in os.listdir('/'.join([dir_name, label_name])):
                    img = load_img('/'.join([dir_name, label_name, img_name]),␣
 ↪target_size = size, color_mode = 'grayscale')
                    # covert image to a matrix
                    img_ts = img_to_array(img)
                    # turn that into a vector / 1D array
                    img_ts = [img_ts.ravel()]
                    try:
                        # concatenate different images
                        full_mat = np.concatenate((full_mat, img_ts))
                    except UnboundLocalError:
                        # if not assigned yet, assign one
                        full_mat = img_ts
    return full_mat

# run it on our folders
```

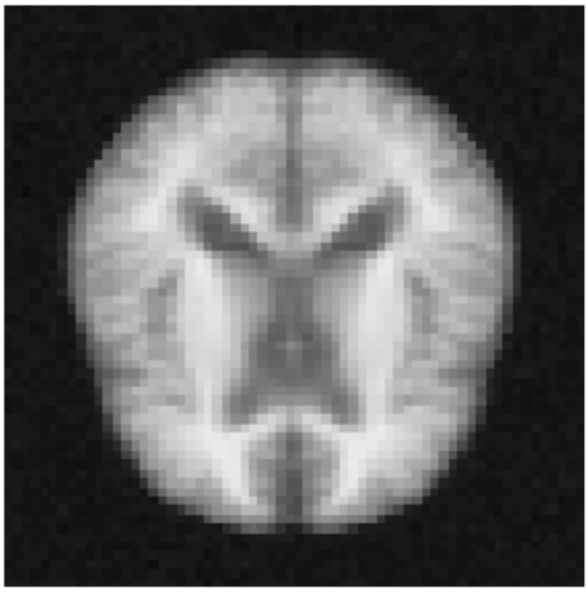
```
MildDemented_images = img2np('Dataset','MildDemented')
ModerateDemented_images = img2np('Dataset','ModerateDemented')
NonDemented_images = img2np('Dataset','NonDemented')
VeryMildDemented_images = img2np('Dataset','VeryMildDemented')
```

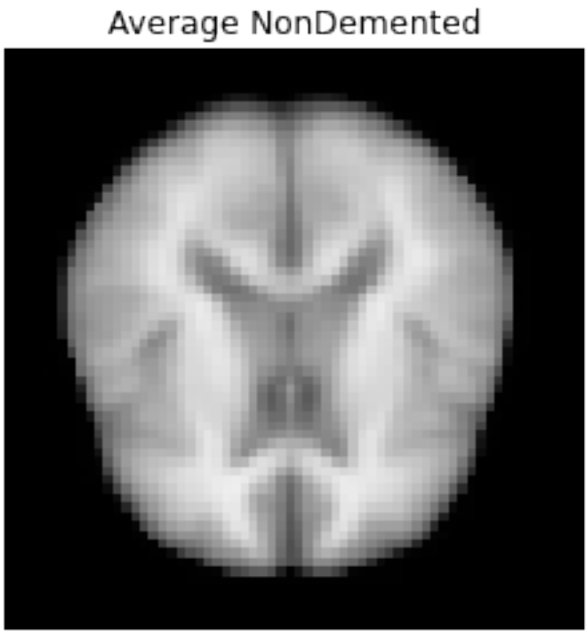
[137]:
```
def find_mean_img(full_mat, title, size = (64, 64)):
    # calculate the average
    mean_img = np.mean(full_mat, axis = 0)
    # reshape it back to a matrix
    mean_img = mean_img.reshape(size)
    plt.imshow(mean_img, vmin=0, vmax=255, cmap='Greys_r')
    plt.title(f'Average {title}')
    plt.axis('off')
    plt.show()
    return mean_img

MildDemented_mean = find_mean_img(MildDemented_images, 'MildDemented')
ModerateDemented_mean = find_mean_img(ModerateDemented_images,␣
↪'ModerateDemented')
NonDemented_mean = find_mean_img(NonDemented_images, 'NonDemented')
VeryMildDemented_mean = find_mean_img(VeryMildDemented_images,␣
↪'VeryMildDemented')

def find_std_img(full_mat, title, size = (64, 64)):
    # calculate the average
    mean_img = np.std(full_mat, axis = 0)
    # reshape it back to a matrix
    mean_img = mean_img.reshape(size)
    plt.imshow(mean_img, vmin=0, vmax=255, cmap='Greys_r')
    plt.title(f'standard deviation {title}')
    plt.axis('off')
    plt.show()
    return mean_img


MildDemented_mean = find_std_img(MildDemented_images, 'MildDemented')
ModerateDemented_mean = find_std_img(ModerateDemented_images,␣
↪'ModerateDemented')
NonDemented_mean = find_std_img(NonDemented_images, 'NonDemented')
VeryMildDemented_mean = find_std_img(VeryMildDemented_images,␣
↪'VeryMildDemented')
```

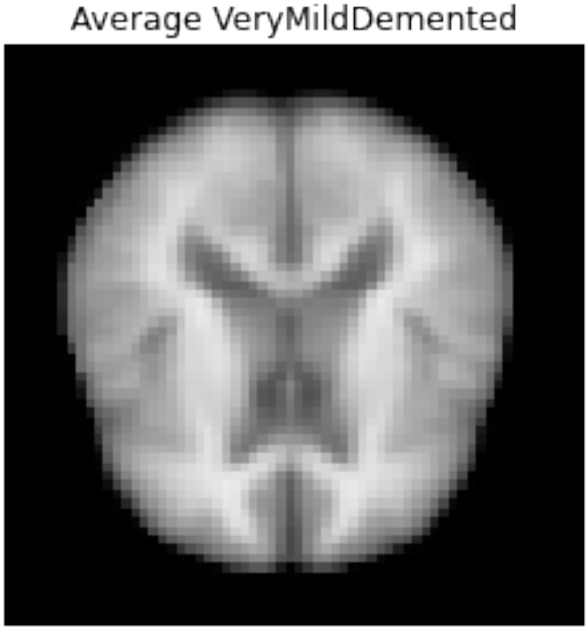
Average MildDemented



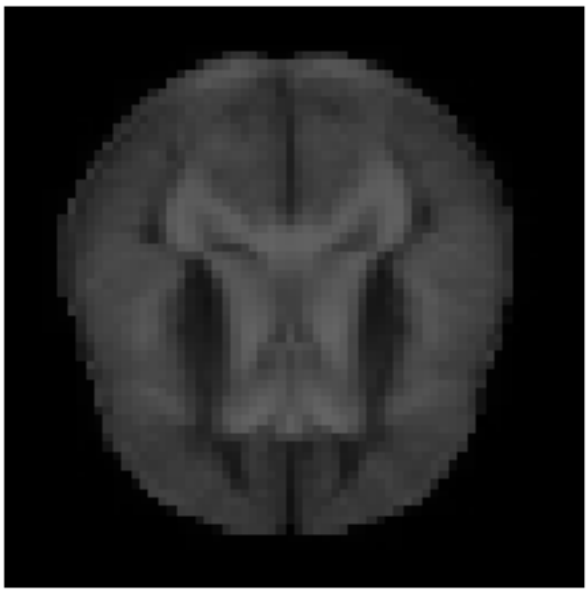
Average ModerateDemented



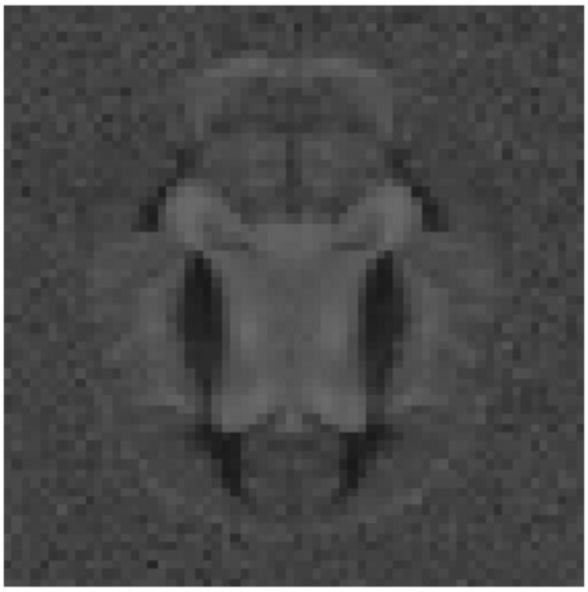

Average NonDemented

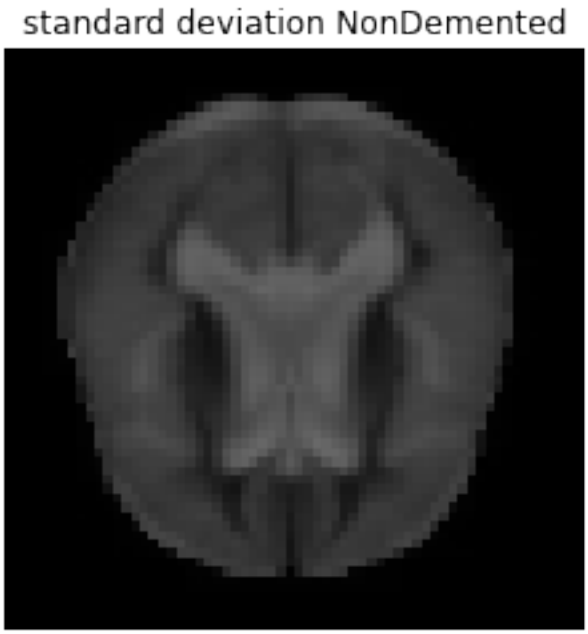

Average VeryMildDemented

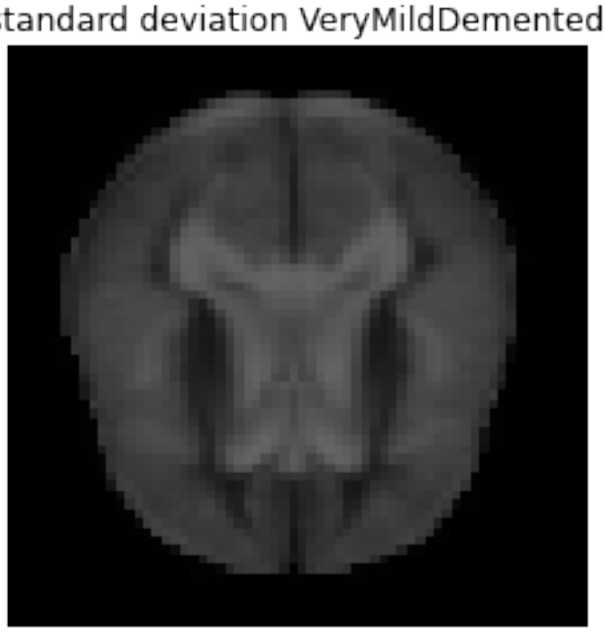
standard deviation MildDemented



standard deviation ModerateDemented



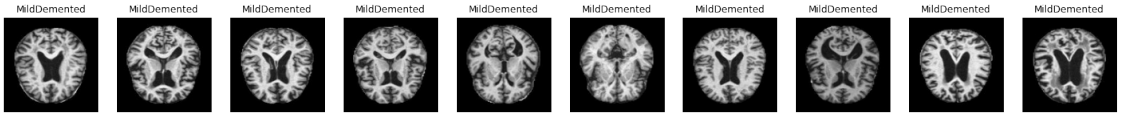




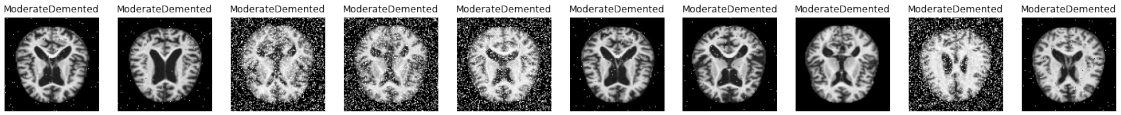
[140]:
```
n_total_images = Images.shape[0]
for target_cls in [0,1,2,3]:
```

```
    indices = np.where(Classes == target_cls)[0] # get target class indices on␣
 ↪Images / Classes
    n_target_cls = indices.shape[0]
    label = class_to_label[target_cls]
    print(label, n_target_cls, n_target_cls*100/n_total_images)

    n_cols = 10 # # of sample plot
    fig, axs = plt.subplots(ncols=n_cols, figsize=(25, 3))

    for i in range(n_cols):

        axs[i].imshow(np.uint8(Images[indices[i]]))
        axs[i].axis('off')
        axs[i].set_title(label)

    plt.show()


def eigenimages(full_mat, title, n_comp = 0.8, size = (64, 64)):
    # fit PCA to describe n_comp * variability in the class
    pca = PCA(n_components = n_comp, whiten = True)
    pca.fit(full_mat)
    print('Number of PC: ', pca.n_components_)
    return pca

def plot_pca(pca, size = (64, 64)):
    # plot eigenimages in a grid
    n = pca.n_components_
    fig = plt.figure(figsize=(8, 8))
    r = int(n**.5)
    c = ceil(n/ r)
    for i in range(n):
        ax = fig.add_subplot(r, c, i + 1, xticks = [], yticks = [])
        ax.imshow(pca.components_[i].reshape(size),
                  cmap='Greys_r')
    plt.axis('off')
    plt.show()

MildDemented_mean = plot_pca(eigenimages(MildDemented_images, 'MildDemented'))
ModerateDemented_mean = plot_pca(eigenimages(ModerateDemented_images,␣
 ↪'ModerateDemented'))
NonDemented_mean = plot_pca(eigenimages(NonDemented_images, 'NonDemented'))
VeryMildDemented_mean = plot_pca(eigenimages(VeryMildDemented_images,␣
 ↪'VeryMildDemented'))
```

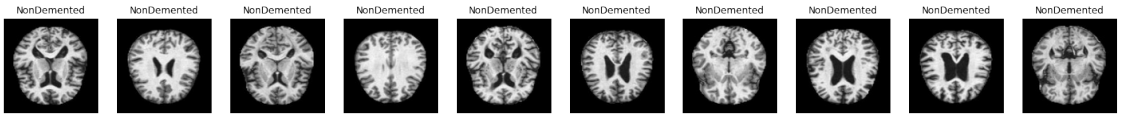
```
MildDemented 896 13.078382717851408
```



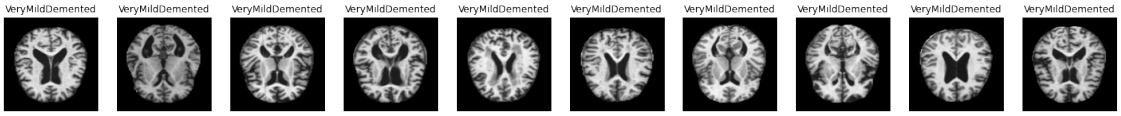
ModerateDemented 515 7.517150780907897



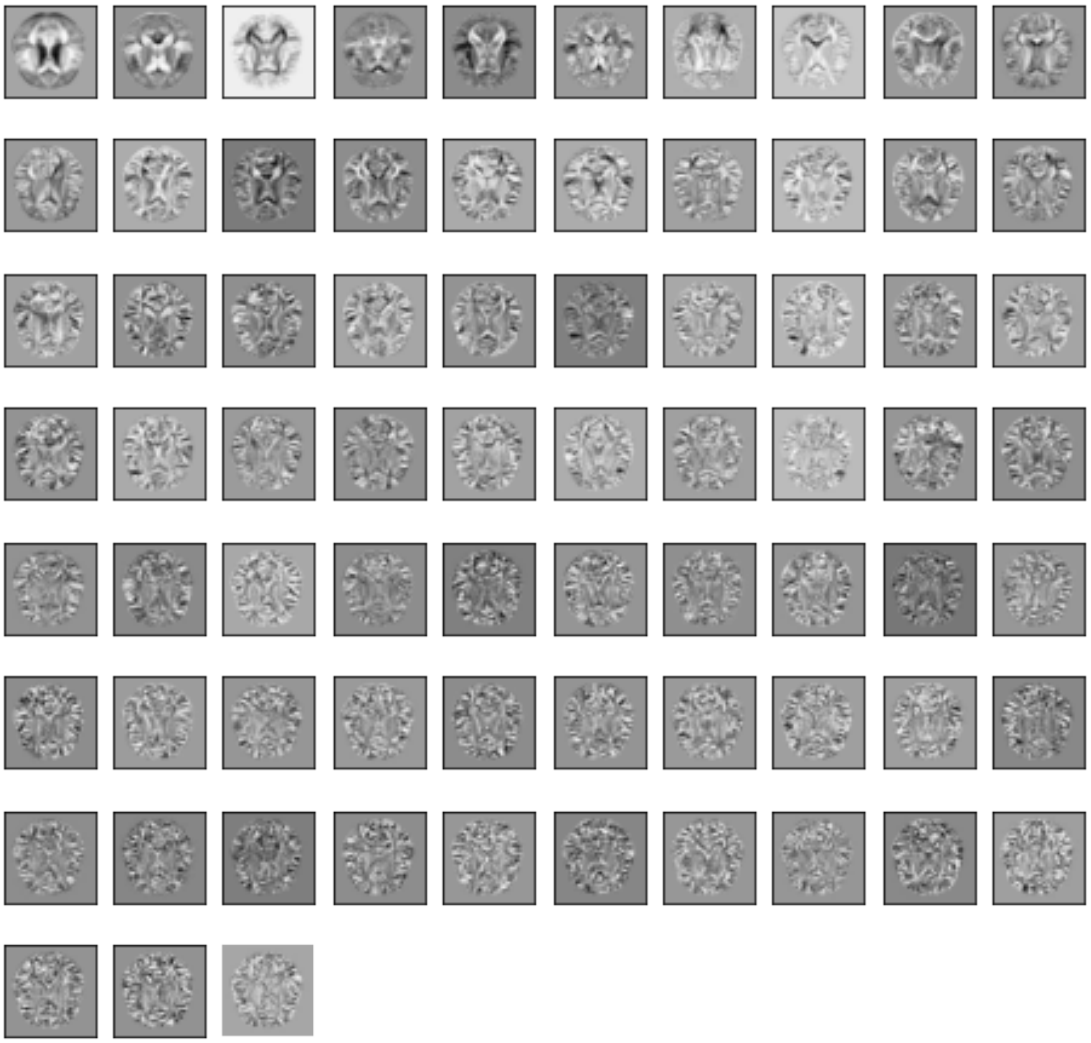
NonDemented 3200 46.708509706612176



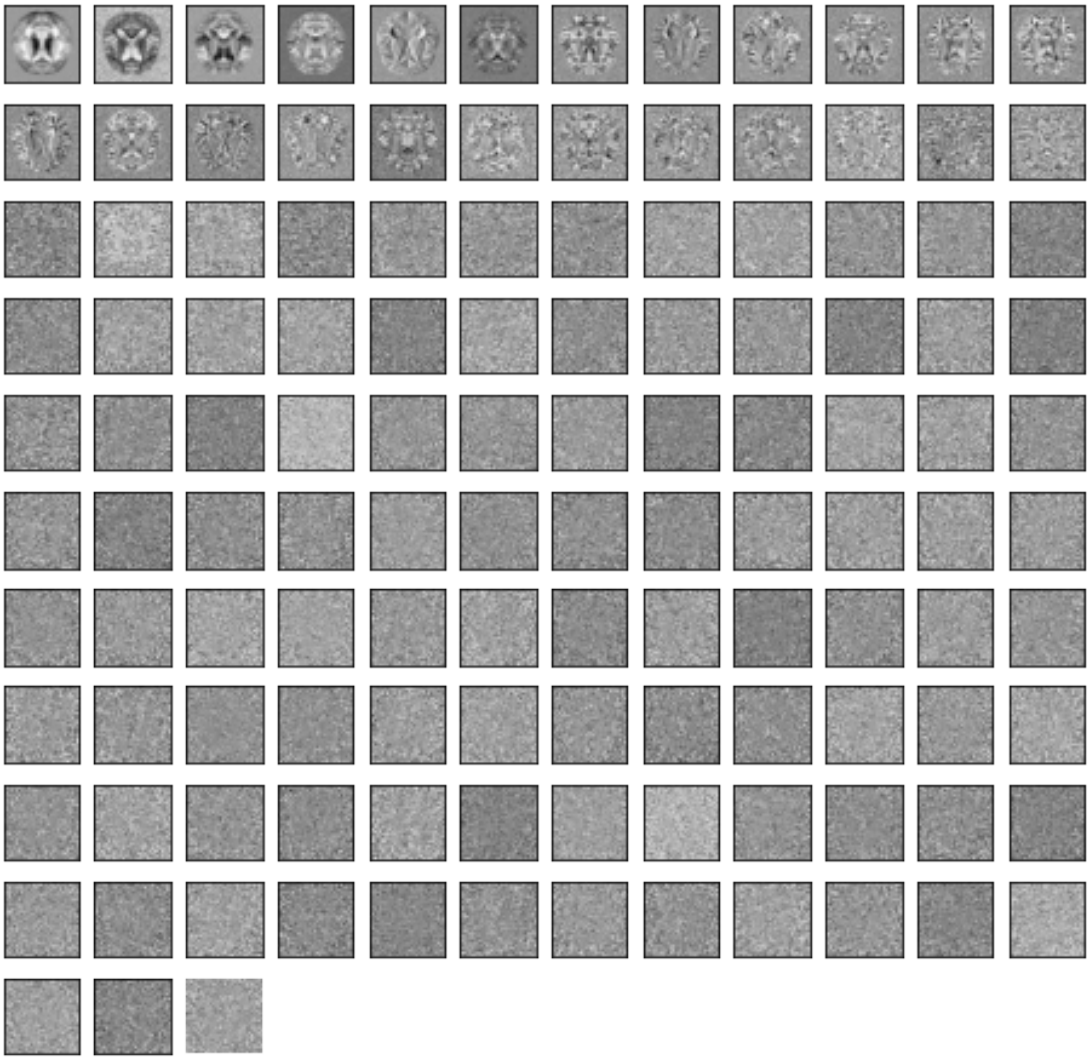
VeryMildDemented 2240 32.69595679462852



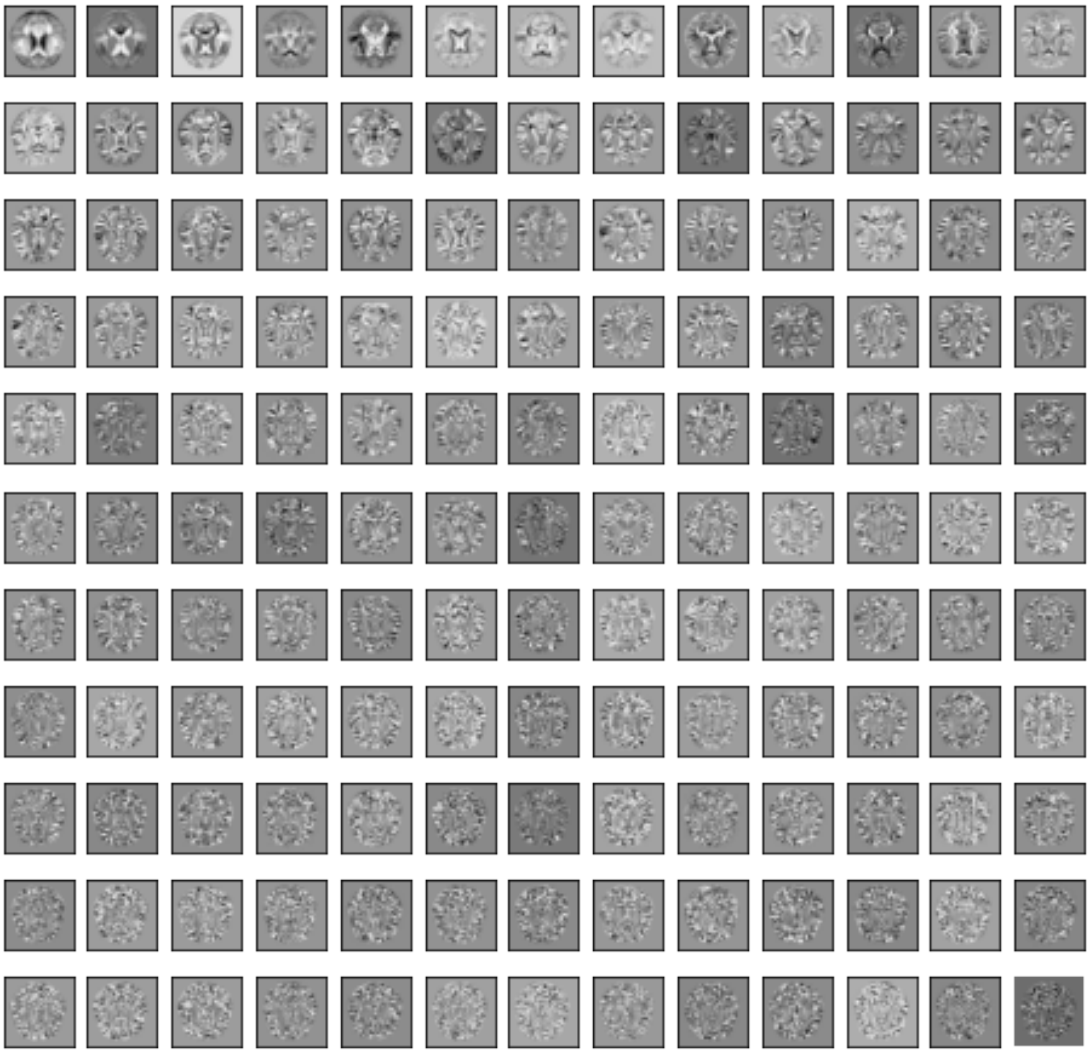
Number of PC: 73



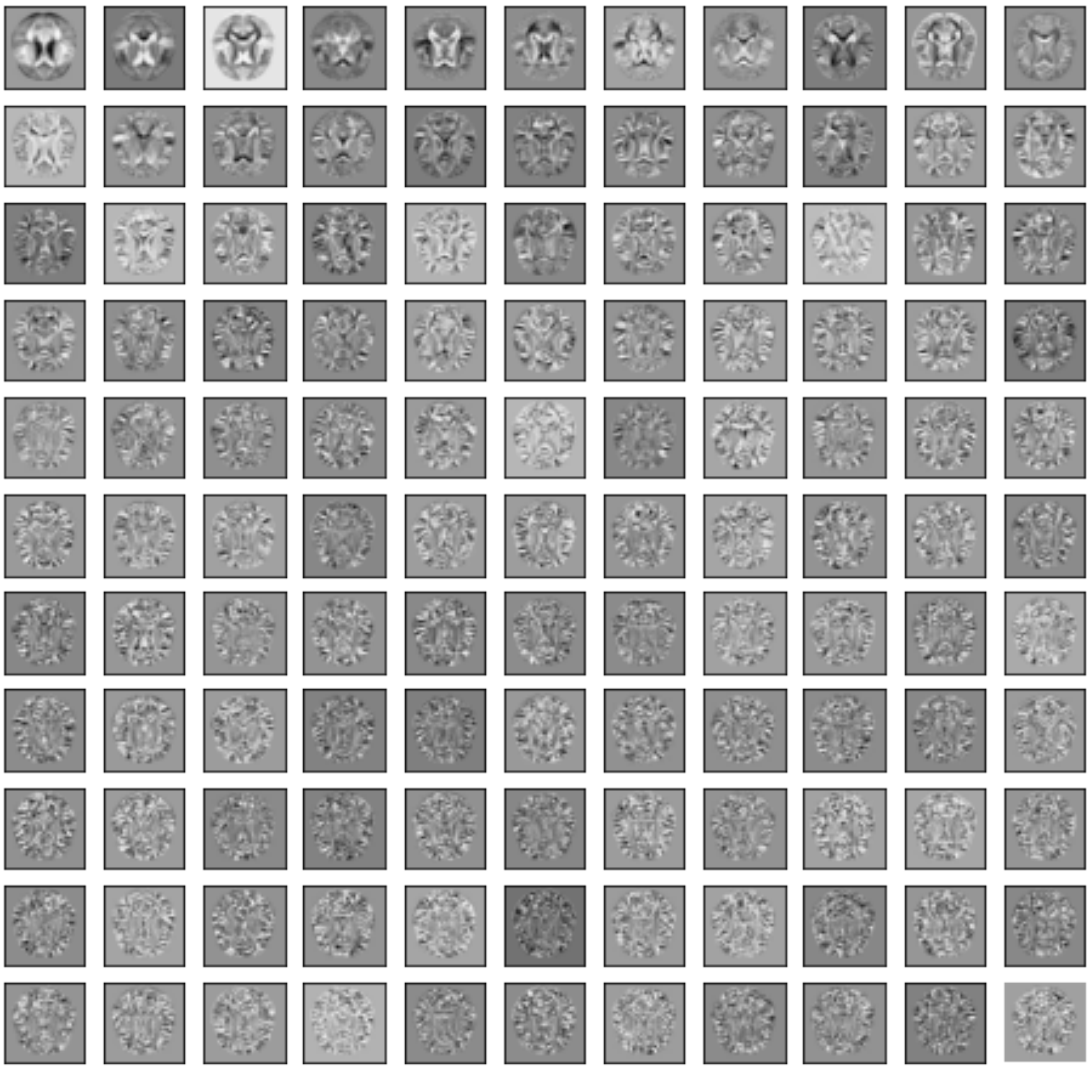
```
Number of PC:  123
```



```
Number of PC:  143
```



```
Number of PC:  121
```



4 Data Preparation

4 directory dataframe 1. split 4 types of data into train, test, validation. 2. Set parameters 3. concat the 4 dataframe

Split 4 types of data

[11]:
```
def random_split_data(data_df):
    """
    this function...
    """
    random_data = data_df.sample(frac=1)
    length = len(random_data)
    test = random_data[:int(0.2*length)]
```

```
    remaining = random_data[int(0.2*length):]
    length_2 = len(remaining)
    train = remaining[:int(0.8*length_2)]
    validation = remaining[int(0.8*length_2):]
    return test, train, validation
```

[12]:
```
# we only use train here and split train dataset to 0.8/0.2 ratio of train/
 ↪test,
# and in new train, we split it to 0.8/0.2 ration of train/validation


# split the four dataframe seperately
non_test, non_train, non_val = random_split_data(non_demented_df)
very_mild_test, very_mild_train, very_mild_val = random_split_data(very_mild_df)
mild_test, mild_train, mild_val = random_split_data(mild_df)
moderate_test, moderate_train, moderate_val = random_split_data(moderate_df)
```

Set parameters

[13]:
```
Autotune = tf.data.experimental.AUTOTUNE
```

[47]:
```
Image_size = [224, 224]
Batch_size = 64
Epochs = 100
```

Concat the 4 dataframe and randomize it

[48]:
```
# concat the 4 data frame
train_total = pd.concat([non_train, very_mild_train, mild_train,␣
 ↪moderate_train])
validation_total = pd.concat([non_val, very_mild_val, mild_val, moderate_val])
test_total = pd.concat([non_test, very_mild_test, mild_test, moderate_test])

# randomize them
train_total = train_total.sample(frac=1)
validation_total = validation_total.sample(frac=1)
test_total = test_total.sample(frac=1)
```

Build three functions to load image, create class array and extract X, y for each train, validation and test.

[49]:
```
# load image using the directory df we create before

def load_image(file):
    """
    this function is...
    """
    image = cv.imread(file)
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
```

```
    # resize it to a same size
    image = cv.resize(image, (Image_size[0], Image_size[1]))
    return image

def encoding_array(y_number):
    """
    this function is...
    """
    class_array = [0,0,0,0]
    class_array[y_number] = 1
    return class_array

def data_preparation(data_set):
    """
    this function is...
    """
    data_set["X"] = data_set["X"].apply(load_image)
    data_set["y"] = data_set["y"].apply(encoding_array)
    X = np.stack(data_set["X"])
    y = np.stack(data_set["y"])
    return X, y
```

[50]:
```
# create X, y for train, validation, test
X_train, y_train = data_preparation(train_total)
X_validation, y_validation = data_preparation(validation_total)
X_test, y_test = data_preparation(test_total)
```

5 Model 1 – NasNetMobile section

1. Model built
2. train
3. fit
4. accuracy line plot + loss line plot
5. predit
6. predit confusion matrix plot

[51]:
```
# import base model
base_model = tf.keras.applications.NASNetMobile(input_shape=(224, 224,3),
                          include_top=False, weights="imagenet")
```

[52]:
```
# Freezing Layers and Train the last 6 layers
for layer in base_model.layers[:-6]:
    layer.trainable=False
```

[53]:
```
# Building Model

model=Sequential()
```

```
model.add(base_model)
model.add(Dropout(0.5))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(4,activation='softmax'))
```

[54]:
```
# Model Summary

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 NASNet (Functional)         (None, 7, 7, 1056)        4269716

 dropout_8 (Dropout)         (None, 7, 7, 1056)        0

 flatten_2 (Flatten)         (None, 51744)             0

 batch_normalization_10 (Bat  (None, 51744)            206976
 chNormalization)

 dense_10 (Dense)            (None, 32)                1655840

 batch_normalization_11 (Bat  (None, 32)               128
 chNormalization)

 activation_760 (Activation)  (None, 32)               0

 dropout_9 (Dropout)         (None, 32)                0

 dense_11 (Dense)            (None, 32)                1056
```

```
batch_normalization_12 (Bat  (None, 32)              128
chNormalization)

activation_761 (Activation)  (None, 32)              0

dropout_10 (Dropout)         (None, 32)              0

dense_12 (Dense)             (None, 32)              1056

batch_normalization_13 (Bat  (None, 32)              128
chNormalization)

activation_762 (Activation)  (None, 32)              0

dropout_11 (Dropout)         (None, 32)              0

dense_13 (Dense)             (None, 32)              1056

batch_normalization_14 (Bat  (None, 32)              128
chNormalization)

activation_763 (Activation)  (None, 32)              0

dense_14 (Dense)             (None, 4)               132

=================================================================
Total params: 6,136,344
Trainable params: 1,762,884
Non-trainable params: 4,373,460
_________________________________________________________________
```

[55]:
```python
def f1_score(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

[56]:
```python
METRICS = [
      tf.keras.metrics.CategoricalAccuracy(name='accuracy'),
      tf.keras.metrics.Precision(name='precision'),
      tf.keras.metrics.Recall(name='recall'),
      tf.keras.metrics.AUC(name='auc'),
      f1_score,
```

```
]
```

[57]:
```
lrd = ReduceLROnPlateau(monitor = 'val_loss',patience = 20,verbose = 1,factor =␣
↪0.50, min_lr = 1e-10)

mcp = ModelCheckpoint('model.h5')

es = EarlyStopping(verbose=1, patience=20)
```

[58]:
```
model.compile(optimizer='Adam', loss='categorical_crossentropy',metrics=METRICS)
```

[59]:
```
history = model.fit(X_train, y_train, validation_data=(X_validation,␣
↪y_validation), batch_size=Batch_size, epochs = Epochs, verbose = 1,␣
↪callbacks=[lrd,mcp,es])
```

```
Epoch 1/100
69/69 [==============================] - 117s 2s/step - loss: 1.4464 - accuracy:
0.3364 - precision: 0.3601 - recall: 0.0758 - auc: 0.5875 - f1_score: 0.1245 -
val_loss: 1.1966 - val_accuracy: 0.5269 - val_precision: 0.8000 - val_recall:
0.1057 - val_auc: 0.7475 - val_f1_score: 0.1860 - lr: 0.0010
Epoch 2/100
69/69 [==============================] - 101s 1s/step - loss: 1.2255 - accuracy:
0.4585 - precision: 0.5240 - recall: 0.1691 - auc: 0.7241 - f1_score: 0.2533 -
val_loss: 1.1076 - val_accuracy: 0.5342 - val_precision: 0.8156 - val_recall:
0.1048 - val_auc: 0.7980 - val_f1_score: 0.1756 - lr: 0.0010
Epoch 3/100
69/69 [==============================] - 105s 2s/step - loss: 1.1300 - accuracy:
0.4920 - precision: 0.5592 - recall: 0.2177 - auc: 0.7663 - f1_score: 0.3108 -
val_loss: 1.0361 - val_accuracy: 0.5269 - val_precision: 0.7343 - val_recall:
0.2771 - val_auc: 0.8178 - val_f1_score: 0.3893 - lr: 0.0010
Epoch 4/100
69/69 [==============================] - 107s 2s/step - loss: 1.1003 - accuracy:
0.4977 - precision: 0.6016 - recall: 0.2533 - auc: 0.7771 - f1_score: 0.3528 -
val_loss: 0.9855 - val_accuracy: 0.5406 - val_precision: 0.7312 - val_recall:
0.3200 - val_auc: 0.8292 - val_f1_score: 0.4384 - lr: 0.0010
Epoch 5/100
69/69 [==============================] - 104s 2s/step - loss: 1.0610 - accuracy:
0.5064 - precision: 0.6227 - recall: 0.2791 - auc: 0.7925 - f1_score: 0.3837 -
val_loss: 0.9601 - val_accuracy: 0.5588 - val_precision: 0.7433 - val_recall:
0.3300 - val_auc: 0.8339 - val_f1_score: 0.4496 - lr: 0.0010
Epoch 6/100
69/69 [==============================] - 105s 2s/step - loss: 1.0273 - accuracy:
0.5148 - precision: 0.6614 - recall: 0.2665 - auc: 0.8043 - f1_score: 0.3799 -
val_loss: 0.9369 - val_accuracy: 0.5552 - val_precision: 0.7515 - val_recall:
0.3336 - val_auc: 0.8389 - val_f1_score: 0.4539 - lr: 0.0010
Epoch 7/100
69/69 [==============================] - 104s 2s/step - loss: 1.0173 - accuracy:
0.5235 - precision: 0.6582 - recall: 0.3028 - auc: 0.8090 - f1_score: 0.4142 -
```

```
val_loss: 0.9238 - val_accuracy: 0.5670 - val_precision: 0.7490 - val_recall:
0.3291 - val_auc: 0.8424 - val_f1_score: 0.4493 - lr: 0.0010
Epoch 8/100
69/69 [==============================] - 102s 1s/step - loss: 1.0031 - accuracy:
0.5278 - precision: 0.6991 - recall: 0.2953 - auc: 0.8146 - f1_score: 0.4156 -
val_loss: 0.9051 - val_accuracy: 0.5606 - val_precision: 0.7643 - val_recall:
0.3282 - val_auc: 0.8471 - val_f1_score: 0.4434 - lr: 0.0010
Epoch 9/100
69/69 [==============================] - 101s 1s/step - loss: 0.9847 - accuracy:
0.5377 - precision: 0.7143 - recall: 0.2967 - auc: 0.8206 - f1_score: 0.4182 -
val_loss: 0.8993 - val_accuracy: 0.5734 - val_precision: 0.7556 - val_recall:
0.3382 - val_auc: 0.8477 - val_f1_score: 0.4585 - lr: 0.0010
Epoch 10/100
69/69 [==============================] - 101s 1s/step - loss: 0.9667 - accuracy:
0.5593 - precision: 0.7230 - recall: 0.3133 - auc: 0.8274 - f1_score: 0.4354 -
val_loss: 0.8883 - val_accuracy: 0.5807 - val_precision: 0.7785 - val_recall:
0.3364 - val_auc: 0.8516 - val_f1_score: 0.4612 - lr: 0.0010
Epoch 11/100
69/69 [==============================] - 104s 2s/step - loss: 0.9594 - accuracy:
0.5548 - precision: 0.7161 - recall: 0.3149 - auc: 0.8303 - f1_score: 0.4376 -
val_loss: 0.8814 - val_accuracy: 0.5925 - val_precision: 0.7826 - val_recall:
0.3282 - val_auc: 0.8539 - val_f1_score: 0.4462 - lr: 0.0010
Epoch 12/100
69/69 [==============================] - 102s 1s/step - loss: 0.9501 - accuracy:
0.5520 - precision: 0.7186 - recall: 0.3229 - auc: 0.8331 - f1_score: 0.4443 -
val_loss: 0.8725 - val_accuracy: 0.5880 - val_precision: 0.7801 - val_recall:
0.3364 - val_auc: 0.8563 - val_f1_score: 0.4611 - lr: 0.0010
Epoch 13/100
69/69 [==============================] - 101s 1s/step - loss: 0.9256 - accuracy:
0.5682 - precision: 0.7425 - recall: 0.3298 - auc: 0.8413 - f1_score: 0.4565 -
val_loss: 0.8658 - val_accuracy: 0.5934 - val_precision: 0.8009 - val_recall:
0.3373 - val_auc: 0.8599 - val_f1_score: 0.4668 - lr: 0.0010
Epoch 14/100
69/69 [==============================] - 101s 1s/step - loss: 0.9295 - accuracy:
0.5598 - precision: 0.7354 - recall: 0.3330 - auc: 0.8380 - f1_score: 0.4582 -
val_loss: 0.8605 - val_accuracy: 0.5925 - val_precision: 0.7944 - val_recall:
0.3382 - val_auc: 0.8601 - val_f1_score: 0.4732 - lr: 0.0010
Epoch 15/100
69/69 [==============================] - 102s 1s/step - loss: 0.9283 - accuracy:
0.5710 - precision: 0.7489 - recall: 0.3355 - auc: 0.8405 - f1_score: 0.4632 -
val_loss: 0.8618 - val_accuracy: 0.5916 - val_precision: 0.7764 - val_recall:
0.3482 - val_auc: 0.8602 - val_f1_score: 0.4797 - lr: 0.0010
Epoch 16/100
69/69 [==============================] - 101s 1s/step - loss: 0.9188 - accuracy:
0.5675 - precision: 0.7468 - recall: 0.3373 - auc: 0.8422 - f1_score: 0.4648 -
val_loss: 0.8661 - val_accuracy: 0.5861 - val_precision: 0.7903 - val_recall:
0.3400 - val_auc: 0.8596 - val_f1_score: 0.4684 - lr: 0.0010
Epoch 17/100
```

```
69/69 [==============================] - 102s 1s/step - loss: 0.9068 - accuracy:
0.5733 - precision: 0.7527 - recall: 0.3494 - auc: 0.8464 - f1_score: 0.4766 -
val_loss: 0.8566 - val_accuracy: 0.5871 - val_precision: 0.7883 - val_recall:
0.3564 - val_auc: 0.8608 - val_f1_score: 0.4890 - lr: 0.0010
Epoch 18/100
69/69 [==============================] - 101s 1s/step - loss: 0.8930 - accuracy:
0.5863 - precision: 0.7410 - recall: 0.3533 - auc: 0.8515 - f1_score: 0.4776 -
val_loss: 0.8548 - val_accuracy: 0.5916 - val_precision: 0.7866 - val_recall:
0.3428 - val_auc: 0.8620 - val_f1_score: 0.4698 - lr: 0.0010
Epoch 19/100
69/69 [==============================] - 101s 1s/step - loss: 0.9053 - accuracy:
0.5847 - precision: 0.7437 - recall: 0.3357 - auc: 0.8474 - f1_score: 0.4628 -
val_loss: 0.8478 - val_accuracy: 0.5998 - val_precision: 0.7732 - val_recall:
0.3573 - val_auc: 0.8655 - val_f1_score: 0.4872 - lr: 0.0010
Epoch 20/100
69/69 [==============================] - 102s 1s/step - loss: 0.8816 - accuracy:
0.5881 - precision: 0.7507 - recall: 0.3567 - auc: 0.8545 - f1_score: 0.4827 -
val_loss: 0.8451 - val_accuracy: 0.6035 - val_precision: 0.7863 - val_recall:
0.3555 - val_auc: 0.8658 - val_f1_score: 0.4883 - lr: 0.0010
Epoch 21/100
69/69 [==============================] - 102s 1s/step - loss: 0.8896 - accuracy:
0.5913 - precision: 0.7316 - recall: 0.3658 - auc: 0.8527 - f1_score: 0.4877 -
val_loss: 0.8420 - val_accuracy: 0.6016 - val_precision: 0.7657 - val_recall:
0.3665 - val_auc: 0.8658 - val_f1_score: 0.4943 - lr: 0.0010
Epoch 22/100
69/69 [==============================] - 102s 1s/step - loss: 0.8876 - accuracy:
0.5865 - precision: 0.7452 - recall: 0.3651 - auc: 0.8542 - f1_score: 0.4896 -
val_loss: 0.8458 - val_accuracy: 0.6044 - val_precision: 0.7899 - val_recall:
0.3564 - val_auc: 0.8663 - val_f1_score: 0.4896 - lr: 0.0010
Epoch 23/100
69/69 [==============================] - 103s 1s/step - loss: 0.8651 - accuracy:
0.5972 - precision: 0.7524 - recall: 0.3717 - auc: 0.8608 - f1_score: 0.4965 -
val_loss: 0.8346 - val_accuracy: 0.5989 - val_precision: 0.7861 - val_recall:
0.3619 - val_auc: 0.8679 - val_f1_score: 0.4874 - lr: 0.0010
Epoch 24/100
69/69 [==============================] - 103s 1s/step - loss: 0.8578 - accuracy:
0.6041 - precision: 0.7449 - recall: 0.3825 - auc: 0.8630 - f1_score: 0.5060 -
val_loss: 0.8307 - val_accuracy: 0.6071 - val_precision: 0.8075 - val_recall:
0.3555 - val_auc: 0.8697 - val_f1_score: 0.4853 - lr: 0.0010
Epoch 25/100
69/69 [==============================] - 103s 1s/step - loss: 0.8662 - accuracy:
0.5947 - precision: 0.7448 - recall: 0.3670 - auc: 0.8598 - f1_score: 0.4894 -
val_loss: 0.8279 - val_accuracy: 0.5998 - val_precision: 0.8117 - val_recall:
0.3537 - val_auc: 0.8699 - val_f1_score: 0.4839 - lr: 0.0010
Epoch 26/100
69/69 [==============================] - 102s 1s/step - loss: 0.8672 - accuracy:
0.6025 - precision: 0.7495 - recall: 0.3797 - auc: 0.8611 - f1_score: 0.5047 -
val_loss: 0.8343 - val_accuracy: 0.6007 - val_precision: 0.8024 - val_recall:
```

```
0.3592 - val_auc: 0.8694 - val_f1_score: 0.4874 - lr: 0.0010
Epoch 27/100
69/69 [==============================] - 103s 2s/step - loss: 0.8418 - accuracy:
0.6052 - precision: 0.7537 - recall: 0.3813 - auc: 0.8675 - f1_score: 0.5047 -
val_loss: 0.8249 - val_accuracy: 0.6080 - val_precision: 0.7834 - val_recall:
0.3692 - val_auc: 0.8718 - val_f1_score: 0.4931 - lr: 0.0010
Epoch 28/100
69/69 [==============================] - 103s 1s/step - loss: 0.8514 - accuracy:
0.6057 - precision: 0.7342 - recall: 0.4085 - auc: 0.8655 - f1_score: 0.5230 -
val_loss: 0.8298 - val_accuracy: 0.6053 - val_precision: 0.7790 - val_recall:
0.3728 - val_auc: 0.8709 - val_f1_score: 0.4943 - lr: 0.0010
Epoch 29/100
69/69 [==============================] - 102s 1s/step - loss: 0.8528 - accuracy:
0.6041 - precision: 0.7424 - recall: 0.3848 - auc: 0.8653 - f1_score: 0.5048 -
val_loss: 0.8255 - val_accuracy: 0.6126 - val_precision: 0.7837 - val_recall:
0.3765 - val_auc: 0.8729 - val_f1_score: 0.4980 - lr: 0.0010
Epoch 30/100
69/69 [==============================] - 101s 1s/step - loss: 0.8529 - accuracy:
0.6091 - precision: 0.7351 - recall: 0.4078 - auc: 0.8664 - f1_score: 0.5236 -
val_loss: 0.8226 - val_accuracy: 0.6153 - val_precision: 0.7911 - val_recall:
0.3728 - val_auc: 0.8739 - val_f1_score: 0.4966 - lr: 0.0010
Epoch 31/100
69/69 [==============================] - 102s 1s/step - loss: 0.8588 - accuracy:
0.6050 - precision: 0.7280 - recall: 0.3994 - auc: 0.8635 - f1_score: 0.5148 -
val_loss: 0.8302 - val_accuracy: 0.6135 - val_precision: 0.7778 - val_recall:
0.3765 - val_auc: 0.8730 - val_f1_score: 0.5037 - lr: 0.0010
Epoch 32/100
69/69 [==============================] - 101s 1s/step - loss: 0.8459 - accuracy:
0.6150 - precision: 0.7479 - recall: 0.3907 - auc: 0.8680 - f1_score: 0.5121 -
val_loss: 0.8366 - val_accuracy: 0.6180 - val_precision: 0.7883 - val_recall:
0.3701 - val_auc: 0.8725 - val_f1_score: 0.4999 - lr: 0.0010
Epoch 33/100
69/69 [==============================] - 101s 1s/step - loss: 0.8252 - accuracy:
0.6219 - precision: 0.7284 - recall: 0.4320 - auc: 0.8741 - f1_score: 0.5405 -
val_loss: 0.8305 - val_accuracy: 0.6199 - val_precision: 0.7397 - val_recall:
0.4275 - val_auc: 0.8741 - val_f1_score: 0.5304 - lr: 0.0010
Epoch 34/100
69/69 [==============================] - 101s 1s/step - loss: 0.8339 - accuracy:
0.6194 - precision: 0.7324 - recall: 0.4247 - auc: 0.8712 - f1_score: 0.5369 -
val_loss: 0.8322 - val_accuracy: 0.6135 - val_precision: 0.7217 - val_recall:
0.4585 - val_auc: 0.8753 - val_f1_score: 0.5530 - lr: 0.0010
Epoch 35/100
69/69 [==============================] - 102s 1s/step - loss: 0.8233 - accuracy:
0.6214 - precision: 0.7511 - recall: 0.4256 - auc: 0.8757 - f1_score: 0.5412 -
val_loss: 0.8242 - val_accuracy: 0.6135 - val_precision: 0.6675 - val_recall:
0.5014 - val_auc: 0.8744 - val_f1_score: 0.5614 - lr: 0.0010
Epoch 36/100
69/69 [==============================] - 100s 1s/step - loss: 0.8290 - accuracy:
```

```
0.6230 - precision: 0.7207 - recall: 0.4569 - auc: 0.8734 - f1_score: 0.5575 -
val_loss: 0.8242 - val_accuracy: 0.6135 - val_precision: 0.6627 - val_recall:
0.5087 - val_auc: 0.8739 - val_f1_score: 0.5656 - lr: 0.0010
Epoch 37/100
69/69 [==============================] - 100s 1s/step - loss: 0.8231 - accuracy:
0.6225 - precision: 0.7116 - recall: 0.4505 - auc: 0.8748 - f1_score: 0.5499 -
val_loss: 0.8173 - val_accuracy: 0.6144 - val_precision: 0.6731 - val_recall:
0.5087 - val_auc: 0.8754 - val_f1_score: 0.5704 - lr: 0.0010
Epoch 38/100
69/69 [==============================] - 100s 1s/step - loss: 0.8113 - accuracy:
0.6246 - precision: 0.7310 - recall: 0.4489 - auc: 0.8781 - f1_score: 0.5550 -
val_loss: 0.8191 - val_accuracy: 0.6180 - val_precision: 0.6750 - val_recall:
0.5169 - val_auc: 0.8760 - val_f1_score: 0.5695 - lr: 0.0010
Epoch 39/100
69/69 [==============================] - 102s 1s/step - loss: 0.8104 - accuracy:
0.6326 - precision: 0.7265 - recall: 0.4754 - auc: 0.8795 - f1_score: 0.5722 -
val_loss: 0.8200 - val_accuracy: 0.6199 - val_precision: 0.6693 - val_recall:
0.5351 - val_auc: 0.8765 - val_f1_score: 0.5839 - lr: 0.0010
Epoch 40/100
69/69 [==============================] - 103s 1s/step - loss: 0.8109 - accuracy:
0.6328 - precision: 0.7172 - recall: 0.4774 - auc: 0.8789 - f1_score: 0.5710 -
val_loss: 0.8175 - val_accuracy: 0.6253 - val_precision: 0.6712 - val_recall:
0.5415 - val_auc: 0.8773 - val_f1_score: 0.5828 - lr: 0.0010
Epoch 41/100
69/69 [==============================] - 105s 2s/step - loss: 0.8163 - accuracy:
0.6308 - precision: 0.7059 - recall: 0.4859 - auc: 0.8773 - f1_score: 0.5734 -
val_loss: 0.8194 - val_accuracy: 0.6162 - val_precision: 0.6770 - val_recall:
0.5178 - val_auc: 0.8762 - val_f1_score: 0.5724 - lr: 0.0010
Epoch 42/100
69/69 [==============================] - 102s 1s/step - loss: 0.8099 - accuracy:
0.6317 - precision: 0.7298 - recall: 0.4580 - auc: 0.8786 - f1_score: 0.5604 -
val_loss: 0.8128 - val_accuracy: 0.6217 - val_precision: 0.6750 - val_recall:
0.5187 - val_auc: 0.8770 - val_f1_score: 0.5704 - lr: 0.0010
Epoch 43/100
69/69 [==============================] - 101s 1s/step - loss: 0.8088 - accuracy:
0.6321 - precision: 0.7329 - recall: 0.4610 - auc: 0.8795 - f1_score: 0.5633 -
val_loss: 0.8289 - val_accuracy: 0.6272 - val_precision: 0.6745 - val_recall:
0.5214 - val_auc: 0.8750 - val_f1_score: 0.5761 - lr: 0.0010
Epoch 44/100
69/69 [==============================] - 101s 1s/step - loss: 0.7971 - accuracy:
0.6417 - precision: 0.7278 - recall: 0.4815 - auc: 0.8834 - f1_score: 0.5770 -
val_loss: 0.8348 - val_accuracy: 0.6244 - val_precision: 0.6644 - val_recall:
0.5269 - val_auc: 0.8742 - val_f1_score: 0.5765 - lr: 0.0010
Epoch 45/100
69/69 [==============================] - 101s 1s/step - loss: 0.7859 - accuracy:
0.6451 - precision: 0.7214 - recall: 0.4970 - auc: 0.8865 - f1_score: 0.5874 -
val_loss: 0.8157 - val_accuracy: 0.6281 - val_precision: 0.6685 - val_recall:
0.5497 - val_auc: 0.8779 - val_f1_score: 0.5911 - lr: 0.0010
```

```
Epoch 46/100
69/69 [==============================] - 100s 1s/step - loss: 0.7904 - accuracy:
0.6458 - precision: 0.7156 - recall: 0.5169 - auc: 0.8859 - f1_score: 0.5986 -
val_loss: 0.8216 - val_accuracy: 0.6317 - val_precision: 0.6667 - val_recall:
0.5561 - val_auc: 0.8777 - val_f1_score: 0.5994 - lr: 0.0010
Epoch 47/100
69/69 [==============================] - 101s 1s/step - loss: 0.7913 - accuracy:
0.6550 - precision: 0.7117 - recall: 0.5324 - auc: 0.8853 - f1_score: 0.6076 -
val_loss: 0.8000 - val_accuracy: 0.6317 - val_precision: 0.6707 - val_recall:
0.5606 - val_auc: 0.8810 - val_f1_score: 0.5986 - lr: 0.0010
Epoch 48/100
69/69 [==============================] - 103s 1s/step - loss: 0.7828 - accuracy:
0.6524 - precision: 0.7248 - recall: 0.5205 - auc: 0.8883 - f1_score: 0.6054 -
val_loss: 0.8059 - val_accuracy: 0.6363 - val_precision: 0.6749 - val_recall:
0.5734 - val_auc: 0.8816 - val_f1_score: 0.6079 - lr: 0.0010
Epoch 49/100
69/69 [==============================] - 99s 1s/step - loss: 0.7920 - accuracy:
0.6431 - precision: 0.7066 - recall: 0.5308 - auc: 0.8846 - f1_score: 0.6048 -
val_loss: 0.7968 - val_accuracy: 0.6436 - val_precision: 0.6745 - val_recall:
0.5725 - val_auc: 0.8835 - val_f1_score: 0.6076 - lr: 0.0010
Epoch 50/100
69/69 [==============================] - 100s 1s/step - loss: 0.7834 - accuracy:
0.6543 - precision: 0.7192 - recall: 0.5132 - auc: 0.8883 - f1_score: 0.5965 -
val_loss: 0.8064 - val_accuracy: 0.6418 - val_precision: 0.6596 - val_recall:
0.5916 - val_auc: 0.8818 - val_f1_score: 0.6115 - lr: 0.0010
Epoch 51/100
69/69 [==============================] - 101s 1s/step - loss: 0.7706 - accuracy:
0.6609 - precision: 0.7142 - recall: 0.5493 - auc: 0.8913 - f1_score: 0.6206 -
val_loss: 0.8018 - val_accuracy: 0.6481 - val_precision: 0.6698 - val_recall:
0.5825 - val_auc: 0.8826 - val_f1_score: 0.6108 - lr: 0.0010
Epoch 52/100
69/69 [==============================] - 100s 1s/step - loss: 0.7809 - accuracy:
0.6559 - precision: 0.7139 - recall: 0.5511 - auc: 0.8894 - f1_score: 0.6213 -
val_loss: 0.8102 - val_accuracy: 0.6399 - val_precision: 0.6832 - val_recall:
0.5779 - val_auc: 0.8822 - val_f1_score: 0.6185 - lr: 0.0010
Epoch 53/100
69/69 [==============================] - 100s 1s/step - loss: 0.7763 - accuracy:
0.6618 - precision: 0.7097 - recall: 0.5580 - auc: 0.8906 - f1_score: 0.6245 -
val_loss: 0.8010 - val_accuracy: 0.6427 - val_precision: 0.6674 - val_recall:
0.5798 - val_auc: 0.8835 - val_f1_score: 0.6083 - lr: 0.0010
Epoch 54/100
69/69 [==============================] - 100s 1s/step - loss: 0.7567 - accuracy:
0.6693 - precision: 0.7168 - recall: 0.5735 - auc: 0.8959 - f1_score: 0.6351 -
val_loss: 0.7940 - val_accuracy: 0.6472 - val_precision: 0.6694 - val_recall:
0.5998 - val_auc: 0.8861 - val_f1_score: 0.6202 - lr: 0.0010
Epoch 55/100
69/69 [==============================] - 101s 1s/step - loss: 0.7772 - accuracy:
0.6513 - precision: 0.6992 - recall: 0.5739 - auc: 0.8885 - f1_score: 0.6303 -
```

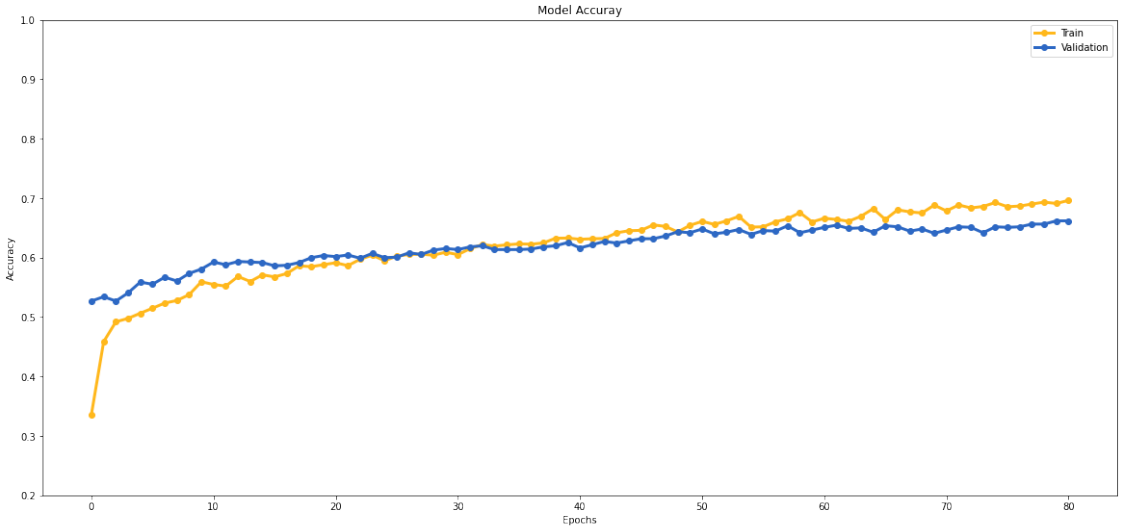
```
val_loss: 0.8056 - val_accuracy: 0.6390 - val_precision: 0.6649 - val_recall:
0.5734 - val_auc: 0.8828 - val_f1_score: 0.6039 - lr: 0.0010
Epoch 56/100
69/69 [==============================] - 100s 1s/step - loss: 0.7706 - accuracy:
0.6518 - precision: 0.6984 - recall: 0.5543 - auc: 0.8899 - f1_score: 0.6174 -
val_loss: 0.8068 - val_accuracy: 0.6454 - val_precision: 0.6721 - val_recall:
0.5980 - val_auc: 0.8842 - val_f1_score: 0.6253 - lr: 0.0010
Epoch 57/100
69/69 [==============================] - 100s 1s/step - loss: 0.7675 - accuracy:
0.6602 - precision: 0.7121 - recall: 0.5639 - auc: 0.8926 - f1_score: 0.6292 -
val_loss: 0.8175 - val_accuracy: 0.6445 - val_precision: 0.6708 - val_recall:
0.5925 - val_auc: 0.8831 - val_f1_score: 0.6168 - lr: 0.0010
Epoch 58/100
69/69 [==============================] - 102s 1s/step - loss: 0.7580 - accuracy:
0.6652 - precision: 0.7133 - recall: 0.5808 - auc: 0.8959 - f1_score: 0.6395 -
val_loss: 0.8096 - val_accuracy: 0.6536 - val_precision: 0.6752 - val_recall:
0.5989 - val_auc: 0.8835 - val_f1_score: 0.6223 - lr: 0.0010
Epoch 59/100
69/69 [==============================] - 100s 1s/step - loss: 0.7404 - accuracy:
0.6757 - precision: 0.7269 - recall: 0.5831 - auc: 0.9009 - f1_score: 0.6457 -
val_loss: 0.8105 - val_accuracy: 0.6418 - val_precision: 0.6800 - val_recall:
0.5871 - val_auc: 0.8841 - val_f1_score: 0.6164 - lr: 0.0010
Epoch 60/100
69/69 [==============================] - 100s 1s/step - loss: 0.7653 - accuracy:
0.6600 - precision: 0.7003 - recall: 0.5637 - auc: 0.8929 - f1_score: 0.6241 -
val_loss: 0.8074 - val_accuracy: 0.6463 - val_precision: 0.6660 - val_recall:
0.5980 - val_auc: 0.8834 - val_f1_score: 0.6175 - lr: 0.0010
Epoch 61/100
69/69 [==============================] - 101s 1s/step - loss: 0.7632 - accuracy:
0.6661 - precision: 0.7121 - recall: 0.5753 - auc: 0.8935 - f1_score: 0.6350 -
val_loss: 0.7899 - val_accuracy: 0.6509 - val_precision: 0.6816 - val_recall:
0.5816 - val_auc: 0.8868 - val_f1_score: 0.6204 - lr: 0.0010
Epoch 62/100
69/69 [==============================] - 100s 1s/step - loss: 0.7530 - accuracy:
0.6641 - precision: 0.7146 - recall: 0.5691 - auc: 0.8961 - f1_score: 0.6321 -
val_loss: 0.7922 - val_accuracy: 0.6545 - val_precision: 0.6831 - val_recall:
0.6071 - val_auc: 0.8865 - val_f1_score: 0.6307 - lr: 0.0010
Epoch 63/100
69/69 [==============================] - 99s 1s/step - loss: 0.7554 - accuracy:
0.6611 - precision: 0.7125 - recall: 0.5762 - auc: 0.8955 - f1_score: 0.6364 -
val_loss: 0.7991 - val_accuracy: 0.6490 - val_precision: 0.6748 - val_recall:
0.6016 - val_auc: 0.8857 - val_f1_score: 0.6233 - lr: 0.0010
Epoch 64/100
69/69 [==============================] - 100s 1s/step - loss: 0.7506 - accuracy:
0.6696 - precision: 0.7116 - recall: 0.5860 - auc: 0.8971 - f1_score: 0.6416 -
val_loss: 0.8098 - val_accuracy: 0.6500 - val_precision: 0.6769 - val_recall:
0.6035 - val_auc: 0.8849 - val_f1_score: 0.6285 - lr: 0.0010
Epoch 65/100
```

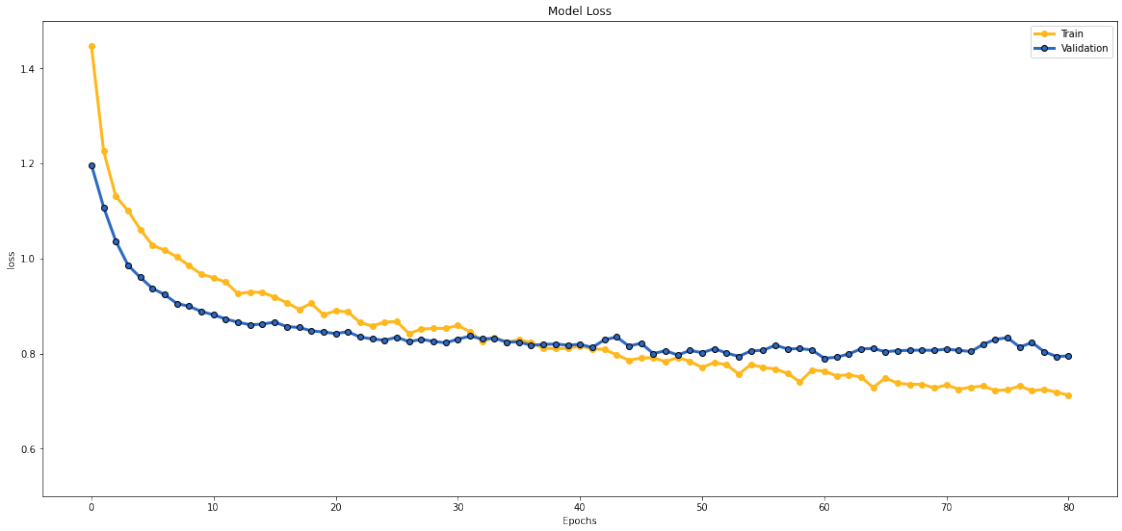
```
69/69 [==============================] - 101s 1s/step - loss: 0.7287 - accuracy:
0.6821 - precision: 0.7270 - recall: 0.6059 - auc: 0.9043 - f1_score: 0.6599 -
val_loss: 0.8106 - val_accuracy: 0.6427 - val_precision: 0.6627 - val_recall:
0.6071 - val_auc: 0.8846 - val_f1_score: 0.6254 - lr: 0.0010
Epoch 66/100
69/69 [==============================] - 100s 1s/step - loss: 0.7486 - accuracy:
0.6641 - precision: 0.7020 - recall: 0.5869 - auc: 0.8968 - f1_score: 0.6374 -
val_loss: 0.8038 - val_accuracy: 0.6536 - val_precision: 0.6742 - val_recall:
0.5962 - val_auc: 0.8844 - val_f1_score: 0.6247 - lr: 0.0010
Epoch 67/100
69/69 [==============================] - 100s 1s/step - loss: 0.7375 - accuracy:
0.6803 - precision: 0.7283 - recall: 0.6038 - auc: 0.9016 - f1_score: 0.6600 -
val_loss: 0.8060 - val_accuracy: 0.6518 - val_precision: 0.6707 - val_recall:
0.6071 - val_auc: 0.8846 - val_f1_score: 0.6293 - lr: 0.0010
Epoch 68/100
69/69 [==============================] - 100s 1s/step - loss: 0.7352 - accuracy:
0.6769 - precision: 0.7151 - recall: 0.6082 - auc: 0.9016 - f1_score: 0.6573 -
val_loss: 0.8067 - val_accuracy: 0.6445 - val_precision: 0.6725 - val_recall:
0.5971 - val_auc: 0.8841 - val_f1_score: 0.6196 - lr: 0.0010
Epoch 69/100
69/69 [==============================] - 100s 1s/step - loss: 0.7353 - accuracy:
0.6750 - precision: 0.7081 - recall: 0.6079 - auc: 0.9014 - f1_score: 0.6530 -
val_loss: 0.8072 - val_accuracy: 0.6481 - val_precision: 0.6751 - val_recall:
0.6062 - val_auc: 0.8855 - val_f1_score: 0.6304 - lr: 0.0010
Epoch 70/100
69/69 [==============================] - 102s 1s/step - loss: 0.7276 - accuracy:
0.6883 - precision: 0.7290 - recall: 0.6212 - auc: 0.9044 - f1_score: 0.6692 -
val_loss: 0.8066 - val_accuracy: 0.6408 - val_precision: 0.6621 - val_recall:
0.6108 - val_auc: 0.8853 - val_f1_score: 0.6273 - lr: 0.0010
Epoch 71/100
69/69 [==============================] - 101s 1s/step - loss: 0.7337 - accuracy:
0.6785 - precision: 0.7156 - recall: 0.6105 - auc: 0.9024 - f1_score: 0.6589 -
val_loss: 0.8093 - val_accuracy: 0.6463 - val_precision: 0.6710 - val_recall:
0.6098 - val_auc: 0.8842 - val_f1_score: 0.6306 - lr: 0.0010
Epoch 72/100
69/69 [==============================] - 100s 1s/step - loss: 0.7251 - accuracy:
0.6885 - precision: 0.7258 - recall: 0.6191 - auc: 0.9051 - f1_score: 0.6665 -
val_loss: 0.8067 - val_accuracy: 0.6518 - val_precision: 0.6654 - val_recall:
0.6162 - val_auc: 0.8854 - val_f1_score: 0.6313 - lr: 0.0010
Epoch 73/100
69/69 [==============================] - 100s 1s/step - loss: 0.7292 - accuracy:
0.6835 - precision: 0.7162 - recall: 0.6180 - auc: 0.9033 - f1_score: 0.6641 -
val_loss: 0.8042 - val_accuracy: 0.6509 - val_precision: 0.6755 - val_recall:
0.6053 - val_auc: 0.8865 - val_f1_score: 0.6345 - lr: 0.0010
Epoch 74/100
69/69 [==============================] - 100s 1s/step - loss: 0.7316 - accuracy:
0.6860 - precision: 0.7263 - recall: 0.6178 - auc: 0.9031 - f1_score: 0.6665 -
val_loss: 0.8192 - val_accuracy: 0.6418 - val_precision: 0.6630 - val_recall:
```

```
0.6026 - val_auc: 0.8842 - val_f1_score: 0.6223 - lr: 0.0010
Epoch 75/100
69/69 [==============================] - 100s 1s/step - loss: 0.7223 - accuracy:
0.6926 - precision: 0.7292 - recall: 0.6403 - auc: 0.9067 - f1_score: 0.6816 -
val_loss: 0.8299 - val_accuracy: 0.6518 - val_precision: 0.6740 - val_recall:
0.6144 - val_auc: 0.8835 - val_f1_score: 0.6333 - lr: 0.0010
Epoch 76/100
69/69 [==============================] - 101s 1s/step - loss: 0.7238 - accuracy:
0.6855 - precision: 0.7226 - recall: 0.6171 - auc: 0.9042 - f1_score: 0.6652 -
val_loss: 0.8332 - val_accuracy: 0.6509 - val_precision: 0.6697 - val_recall:
0.6062 - val_auc: 0.8840 - val_f1_score: 0.6283 - lr: 0.0010
Epoch 77/100
69/69 [==============================] - 101s 1s/step - loss: 0.7317 - accuracy:
0.6867 - precision: 0.7229 - recall: 0.6235 - auc: 0.9036 - f1_score: 0.6701 -
val_loss: 0.8135 - val_accuracy: 0.6518 - val_precision: 0.6733 - val_recall:
0.6199 - val_auc: 0.8866 - val_f1_score: 0.6360 - lr: 0.0010
Epoch 78/100
69/69 [==============================] - 100s 1s/step - loss: 0.7215 - accuracy:
0.6901 - precision: 0.7255 - recall: 0.6344 - auc: 0.9066 - f1_score: 0.6742 -
val_loss: 0.8232 - val_accuracy: 0.6563 - val_precision: 0.6746 - val_recall:
0.6217 - val_auc: 0.8849 - val_f1_score: 0.6374 - lr: 0.0010
Epoch 79/100
69/69 [==============================] - 100s 1s/step - loss: 0.7248 - accuracy:
0.6933 - precision: 0.7289 - recall: 0.6358 - auc: 0.9066 - f1_score: 0.6795 -
val_loss: 0.8040 - val_accuracy: 0.6563 - val_precision: 0.6699 - val_recall:
0.6217 - val_auc: 0.8872 - val_f1_score: 0.6308 - lr: 0.0010
Epoch 80/100
69/69 [==============================] - 101s 1s/step - loss: 0.7185 - accuracy:
0.6910 - precision: 0.7174 - recall: 0.6344 - auc: 0.9068 - f1_score: 0.6740 -
val_loss: 0.7938 - val_accuracy: 0.6618 - val_precision: 0.6773 - val_recall:
0.6217 - val_auc: 0.8896 - val_f1_score: 0.6350 - lr: 0.0010
Epoch 81/100
69/69 [==============================] - ETA: 0s - loss: 0.7125 - accuracy:
0.6963 - precision: 0.7292 - recall: 0.6372 - auc: 0.9081 - f1_score: 0.6786
Epoch 81: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
69/69 [==============================] - 100s 1s/step - loss: 0.7125 - accuracy:
0.6963 - precision: 0.7292 - recall: 0.6372 - auc: 0.9081 - f1_score: 0.6786 -
val_loss: 0.7949 - val_accuracy: 0.6618 - val_precision: 0.6798 - val_recall:
0.6290 - val_auc: 0.8897 - val_f1_score: 0.6380 - lr: 0.0010
Epoch 81: early stopping
```

[108]:
```
from matplotlib import pyplot as plt

def plot_curve(history):


    fig, ax = plt.subplots(2, 1, figsize=(20, 20))
```

```
    ax = ax.ravel()

    ax[0].set_ylim([0.2,1])

    ax[0].plot(history.
↪history["accuracy"],linewidth=3,color="#FFB81C",marker='o')
    ax[0].plot(history.
↪history["val_accuracy"],linewidth=3,marker='o',color="#2D68C4")
    ax[0].set_title('Model {}'.format("Accuray"))
    ax[0].set_xlabel('Epochs')
    ax[0].set_ylabel("Accuracy")
    ax[0].legend(['Train', 'Validation'])

    ax[1].set_ylim([0.5,1.5])

    ax[1].plot(history.history["loss"],linewidth=3,color="#FFB81C",marker='o')
    ax[1].plot(history.history['val_' + "loss"],linewidth=3,
     marker='o', markeredgecolor='black',color="#2D68C4")
    ax[1].set_title('Model {}'.format("Loss"))
    ax[1].set_xlabel('Epochs')
    ax[1].set_ylabel("loss")
    ax[1].legend(['Train', 'Validation'])


plot_curve(history)
```





[61]:
```
model.evaluate(X_test,y_test)
```

```
43/43 [==============================] - 24s 560ms/step - loss: 0.8592 -
accuracy: 0.6589 - precision: 0.6741 - recall: 0.6180 - auc: 0.8840 - f1_score:
0.6449
```

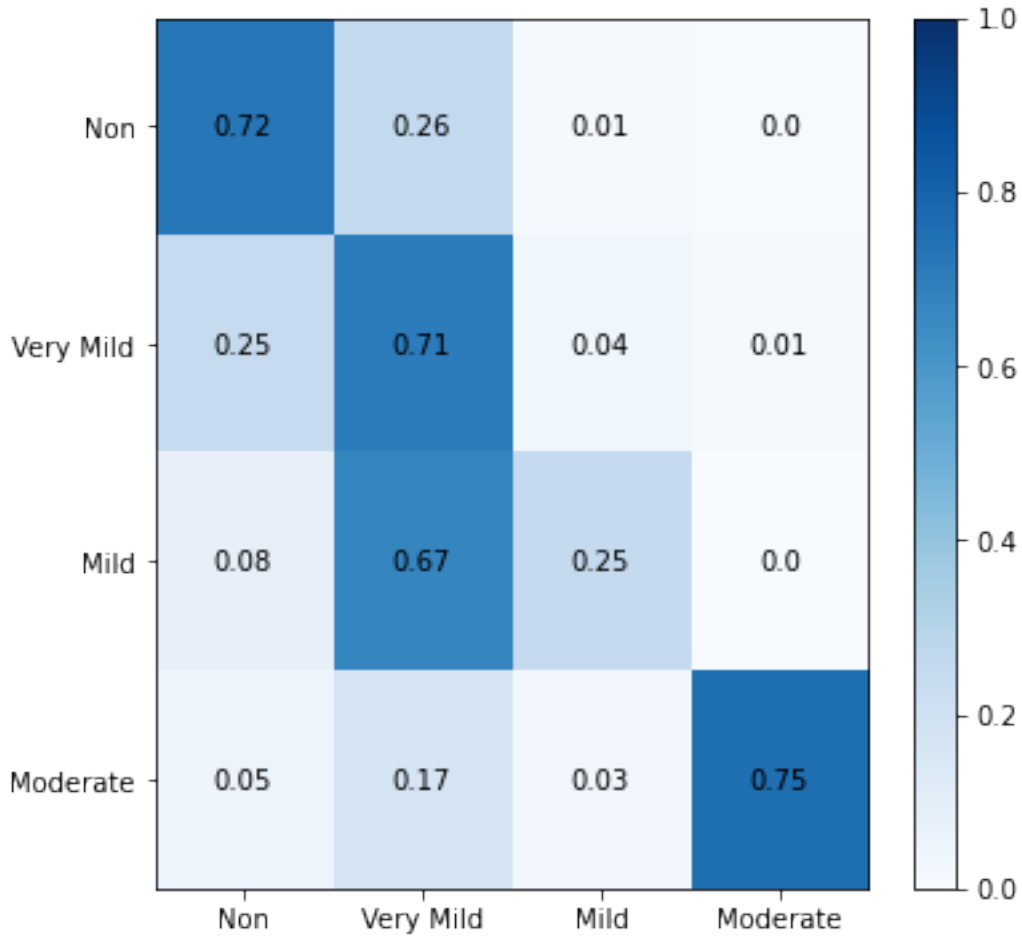
[61]:
```
[0.8592313528060913,
 0.6588751077651978,
 0.6741035580635071,
 0.6179693341255188,
 0.8840410113334656,
 0.644862711429596]
```

[62]:
```
y_pred = model.predict(X_test)
y_pred_1 = np.squeeze(y_pred)
y_pred_2 = np.argmax(y_pred_1,axis=1)
```

[63]:
```
y_new_test = y_test.argmax(axis=1)
matrix = confusion_matrix(y_new_test, y_pred_2, normalize="true")
matrix
```

[63]:
```
array([[0.725     , 0.2609375 , 0.009375  , 0.0046875 ],
       [0.24776786, 0.70535714, 0.03794643, 0.00892857],
       [0.07821229, 0.67039106, 0.25139665, 0.        ],
       [0.04901961, 0.16666667, 0.02941176, 0.75490196]])
```

[64]:
```
def plot_confusion_matrix(matrix):
    cmap = cm.Blues
    fig, ax = plt.subplots(figsize=(6,6))
    ax.set_xticks([0, 1, 2, 3])
    ax.set_xticklabels(["Non", "Very Mild", "Mild", "Moderate"])
    ax.set_yticks([0, 1, 2, 3])
    ax.set_yticklabels(["Non", "Very Mild", "Mild", "Moderate"])
    h = ax.imshow(matrix, cmap = cmap, aspect="auto", vmin = 0, vmax = 1)
    for x in range(4):
        for y in range(4):
            value = float(format('%.2f' % matrix[y,x]))
            plt.text(x, y, value, verticalalignment = 'center',␣
 ↪horizontalalignment = 'center')
    fig.colorbar(h)

plot_confusion_matrix(matrix)
```



6 Model 2 – VGG Section

1. Model built
2. train
3. fit
4. accuracy line plot + loss line plot
5. predit
6. predit confusion matrix plot

[65]:
```
# Image_size = (128, 128, 3) for this model, reload images and no need
# to resize it
```

[66]:
```
# reload directory
train_total = pd.concat([non_train, very_mild_train, mild_train,␣
↪moderate_train])
validation_total = pd.concat([non_val, very_mild_val, mild_val, moderate_val])
test_total = pd.concat([non_test, very_mild_test, mild_test, moderate_test])
```

```
# randomize them
train_total = train_total.sample(frac=1)
validation_total = validation_total.sample(frac=1)
test_total = test_total.sample(frac=1)
```

[67]:
```
def load_image(file):
    """
    this function is...
    """
    image = cv.imread(file)
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    # since the original image size is (128, 128, 3), we don't resize the image
    # here
    return image

def data_preparation(data_set):
    """
    this function is...
    """
    data_set["X"] = data_set["X"].apply(load_image)
    X = np.stack(data_set["X"])
    y = np.stack(data_set["y"])
    return X, y

X_train, y_train = data_preparation(train_total)
X_validation, y_validation = data_preparation(validation_total)
X_test, y_test = data_preparation(test_total)
```

[68]:
```
vgg16 = VGG16(weights='imagenet', input_shape=(128,128,3), include_top=False)
# Set all layers to non-trainable
for layer in vgg16.layers:
    layer.trainable = False
# Set the last 6 vgg layers to trainable
vgg16.layers[-2].trainable = True
vgg16.layers[-3].trainable = True
vgg16.layers[-4].trainable = True
vgg16.layers[-5].trainable = True
vgg16.layers[-6].trainable = True
vgg16.layers[-7].trainable = True
```

[69]:
```
modelvgg = Sequential()
modelvgg.add(Input(shape=(128,128,3)))
modelvgg.add(vgg16)
modelvgg.add(Flatten())
modelvgg.add(Dropout(0.25))
modelvgg.add(Dense(128, activation='relu'))
```

```
modelvgg.add(Dropout(0.2))
modelvgg.add(Dense(4, activation='sigmoid'))
```

[70]:
```
modelvgg.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten_3 (Flatten)         (None, 8192)              0

 dropout_12 (Dropout)        (None, 8192)              0

 dense_15 (Dense)            (None, 128)               1048704

 dropout_13 (Dropout)        (None, 128)               0

 dense_16 (Dense)            (None, 4)                 516

=================================================================
Total params: 15,763,908
Trainable params: 12,848,260
Non-trainable params: 2,915,648
_________________________________________________________________
```

[71]:
```
modelvgg.compile(optimizer=Adam(learning_rate=0.0001),
             loss='sparse_categorical_crossentropy',
             metrics=['sparse_categorical_accuracy'])
```

[72]:
```
lrd = ReduceLROnPlateau(monitor = 'val_loss',patience = 20,verbose = 1,factor =␣
↪0.50, min_lr = 1e-10)

mcp = ModelCheckpoint('model.h5')

es = EarlyStopping(verbose=1, patience=20)

historyvgg = modelvgg.fit(X_train,␣
↪y_train,validation_data=(X_validation,y_validation), batch_size=Batch_size,␣
↪epochs = Epochs, verbose = 1, callbacks=[lrd,mcp,es])
```

```
Epoch 1/100
69/69 [==============================] - 203s 3s/step - loss: 1.5293 -
sparse_categorical_accuracy: 0.4795 - val_loss: 0.9347 -
val_sparse_categorical_accuracy: 0.5779 - lr: 1.0000e-04
Epoch 2/100
69/69 [==============================] - 205s 3s/step - loss: 0.9732 -
```

```
sparse_categorical_accuracy: 0.5338 - val_loss: 0.8943 -
val_sparse_categorical_accuracy: 0.5907 - lr: 1.0000e-04
Epoch 3/100
69/69 [==============================] - 205s 3s/step - loss: 0.9004 -
sparse_categorical_accuracy: 0.5730 - val_loss: 0.8337 -
val_sparse_categorical_accuracy: 0.5953 - lr: 1.0000e-04
Epoch 4/100
69/69 [==============================] - 204s 3s/step - loss: 0.8656 -
sparse_categorical_accuracy: 0.5828 - val_loss: 0.8315 -
val_sparse_categorical_accuracy: 0.6217 - lr: 1.0000e-04
Epoch 5/100
69/69 [==============================] - 202s 3s/step - loss: 0.8291 -
sparse_categorical_accuracy: 0.6116 - val_loss: 0.8895 -
val_sparse_categorical_accuracy: 0.5488 - lr: 1.0000e-04
Epoch 6/100
69/69 [==============================] - 204s 3s/step - loss: 0.8012 -
sparse_categorical_accuracy: 0.6305 - val_loss: 0.7467 -
val_sparse_categorical_accuracy: 0.6554 - lr: 1.0000e-04
Epoch 7/100
69/69 [==============================] - 203s 3s/step - loss: 0.7569 -
sparse_categorical_accuracy: 0.6397 - val_loss: 0.7454 -
val_sparse_categorical_accuracy: 0.6454 - lr: 1.0000e-04
Epoch 8/100
69/69 [==============================] - 204s 3s/step - loss: 0.6899 -
sparse_categorical_accuracy: 0.6728 - val_loss: 0.6706 -
val_sparse_categorical_accuracy: 0.6828 - lr: 1.0000e-04
Epoch 9/100
69/69 [==============================] - 205s 3s/step - loss: 0.6604 -
sparse_categorical_accuracy: 0.6933 - val_loss: 0.6567 -
val_sparse_categorical_accuracy: 0.7074 - lr: 1.0000e-04
Epoch 10/100
69/69 [==============================] - 204s 3s/step - loss: 0.6282 -
sparse_categorical_accuracy: 0.7143 - val_loss: 0.5772 -
val_sparse_categorical_accuracy: 0.7475 - lr: 1.0000e-04
Epoch 11/100
69/69 [==============================] - 199s 3s/step - loss: 0.5084 -
sparse_categorical_accuracy: 0.7777 - val_loss: 0.5355 -
val_sparse_categorical_accuracy: 0.7557 - lr: 1.0000e-04
Epoch 12/100
69/69 [==============================] - 201s 3s/step - loss: 0.4170 -
sparse_categorical_accuracy: 0.8259 - val_loss: 0.4622 -
val_sparse_categorical_accuracy: 0.8113 - lr: 1.0000e-04
Epoch 13/100
69/69 [==============================] - 200s 3s/step - loss: 0.4762 -
sparse_categorical_accuracy: 0.8058 - val_loss: 0.4958 -
val_sparse_categorical_accuracy: 0.7922 - lr: 1.0000e-04
Epoch 14/100
69/69 [==============================] - 199s 3s/step - loss: 0.2452 -
```

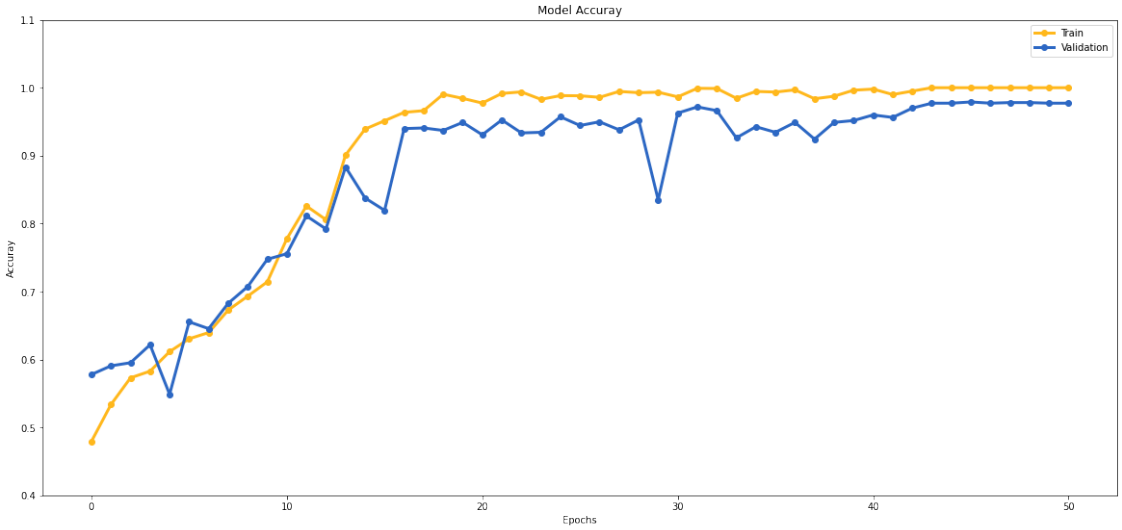
```
sparse_categorical_accuracy: 0.9007 - val_loss: 0.3531 -
val_sparse_categorical_accuracy: 0.8833 - lr: 1.0000e-04
Epoch 15/100
69/69 [==============================] - 201s 3s/step - loss: 0.1733 -
sparse_categorical_accuracy: 0.9391 - val_loss: 0.5041 -
val_sparse_categorical_accuracy: 0.8377 - lr: 1.0000e-04
Epoch 16/100
69/69 [==============================] - 199s 3s/step - loss: 0.1279 -
sparse_categorical_accuracy: 0.9514 - val_loss: 0.6844 -
val_sparse_categorical_accuracy: 0.8195 - lr: 1.0000e-04
Epoch 17/100
69/69 [==============================] - 199s 3s/step - loss: 0.1016 -
sparse_categorical_accuracy: 0.9637 - val_loss: 0.2433 -
val_sparse_categorical_accuracy: 0.9398 - lr: 1.0000e-04
Epoch 18/100
69/69 [==============================] - 200s 3s/step - loss: 0.1042 -
sparse_categorical_accuracy: 0.9662 - val_loss: 0.1739 -
val_sparse_categorical_accuracy: 0.9407 - lr: 1.0000e-04
Epoch 19/100
69/69 [==============================] - 204s 3s/step - loss: 0.0317 -
sparse_categorical_accuracy: 0.9904 - val_loss: 0.1937 -
val_sparse_categorical_accuracy: 0.9371 - lr: 1.0000e-04
Epoch 20/100
69/69 [==============================] - 199s 3s/step - loss: 0.0459 -
sparse_categorical_accuracy: 0.9843 - val_loss: 0.1610 -
val_sparse_categorical_accuracy: 0.9490 - lr: 1.0000e-04
Epoch 21/100
69/69 [==============================] - 199s 3s/step - loss: 0.0715 -
sparse_categorical_accuracy: 0.9774 - val_loss: 0.2460 -
val_sparse_categorical_accuracy: 0.9307 - lr: 1.0000e-04
Epoch 22/100
69/69 [==============================] - 198s 3s/step - loss: 0.0296 -
sparse_categorical_accuracy: 0.9916 - val_loss: 0.1538 -
val_sparse_categorical_accuracy: 0.9526 - lr: 1.0000e-04
Epoch 23/100
69/69 [==============================] - 197s 3s/step - loss: 0.0164 -
sparse_categorical_accuracy: 0.9938 - val_loss: 0.2215 -
val_sparse_categorical_accuracy: 0.9335 - lr: 1.0000e-04
Epoch 24/100
69/69 [==============================] - 198s 3s/step - loss: 0.0429 -
sparse_categorical_accuracy: 0.9829 - val_loss: 0.2138 -
val_sparse_categorical_accuracy: 0.9344 - lr: 1.0000e-04
Epoch 25/100
69/69 [==============================] - 198s 3s/step - loss: 0.0346 -
sparse_categorical_accuracy: 0.9884 - val_loss: 0.1396 -
val_sparse_categorical_accuracy: 0.9572 - lr: 1.0000e-04
Epoch 26/100
69/69 [==============================] - 198s 3s/step - loss: 0.0329 -
```

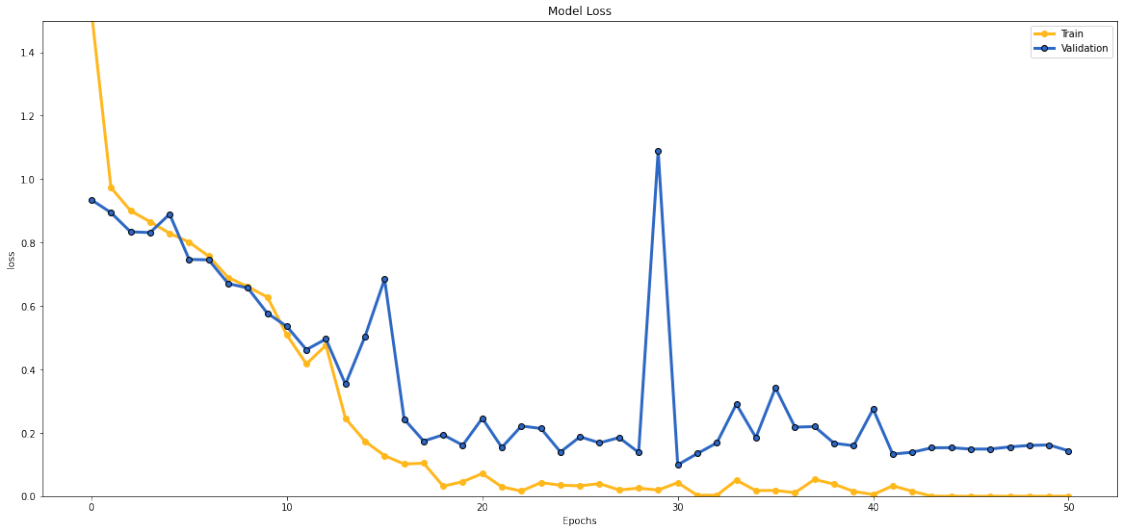
```
sparse_categorical_accuracy: 0.9881 - val_loss: 0.1875 -
val_sparse_categorical_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 27/100
69/69 [==============================] - 199s 3s/step - loss: 0.0396 -
sparse_categorical_accuracy: 0.9859 - val_loss: 0.1682 -
val_sparse_categorical_accuracy: 0.9499 - lr: 1.0000e-04
Epoch 28/100
69/69 [==============================] - 199s 3s/step - loss: 0.0197 -
sparse_categorical_accuracy: 0.9945 - val_loss: 0.1849 -
val_sparse_categorical_accuracy: 0.9380 - lr: 1.0000e-04
Epoch 29/100
69/69 [==============================] - 198s 3s/step - loss: 0.0255 -
sparse_categorical_accuracy: 0.9927 - val_loss: 0.1382 -
val_sparse_categorical_accuracy: 0.9526 - lr: 1.0000e-04
Epoch 30/100
69/69 [==============================] - 200s 3s/step - loss: 0.0194 -
sparse_categorical_accuracy: 0.9934 - val_loss: 1.0885 -
val_sparse_categorical_accuracy: 0.8350 - lr: 1.0000e-04
Epoch 31/100
69/69 [==============================] - 199s 3s/step - loss: 0.0430 -
sparse_categorical_accuracy: 0.9865 - val_loss: 0.0990 -
val_sparse_categorical_accuracy: 0.9626 - lr: 1.0000e-04
Epoch 32/100
69/69 [==============================] - 199s 3s/step - loss: 0.0030 -
sparse_categorical_accuracy: 0.9991 - val_loss: 0.1345 -
val_sparse_categorical_accuracy: 0.9717 - lr: 1.0000e-04
Epoch 33/100
69/69 [==============================] - 203s 3s/step - loss: 0.0032 -
sparse_categorical_accuracy: 0.9989 - val_loss: 0.1682 -
val_sparse_categorical_accuracy: 0.9663 - lr: 1.0000e-04
Epoch 34/100
69/69 [==============================] - 202s 3s/step - loss: 0.0506 -
sparse_categorical_accuracy: 0.9845 - val_loss: 0.2903 -
val_sparse_categorical_accuracy: 0.9262 - lr: 1.0000e-04
Epoch 35/100
69/69 [==============================] - 201s 3s/step - loss: 0.0177 -
sparse_categorical_accuracy: 0.9945 - val_loss: 0.1850 -
val_sparse_categorical_accuracy: 0.9426 - lr: 1.0000e-04
Epoch 36/100
69/69 [==============================] - 202s 3s/step - loss: 0.0184 -
sparse_categorical_accuracy: 0.9936 - val_loss: 0.3418 -
val_sparse_categorical_accuracy: 0.9344 - lr: 1.0000e-04
Epoch 37/100
69/69 [==============================] - 202s 3s/step - loss: 0.0110 -
sparse_categorical_accuracy: 0.9968 - val_loss: 0.2179 -
val_sparse_categorical_accuracy: 0.9490 - lr: 1.0000e-04
Epoch 38/100
69/69 [==============================] - 199s 3s/step - loss: 0.0530 -
```

```
sparse_categorical_accuracy: 0.9838 - val_loss: 0.2197 -
val_sparse_categorical_accuracy: 0.9243 - lr: 1.0000e-04
Epoch 39/100
69/69 [==============================] - 198s 3s/step - loss: 0.0382 -
sparse_categorical_accuracy: 0.9874 - val_loss: 0.1672 -
val_sparse_categorical_accuracy: 0.9490 - lr: 1.0000e-04
Epoch 40/100
69/69 [==============================] - 197s 3s/step - loss: 0.0152 -
sparse_categorical_accuracy: 0.9963 - val_loss: 0.1593 -
val_sparse_categorical_accuracy: 0.9517 - lr: 1.0000e-04
Epoch 41/100
69/69 [==============================] - 198s 3s/step - loss: 0.0054 -
sparse_categorical_accuracy: 0.9979 - val_loss: 0.2747 -
val_sparse_categorical_accuracy: 0.9599 - lr: 1.0000e-04
Epoch 42/100
69/69 [==============================] - 198s 3s/step - loss: 0.0327 -
sparse_categorical_accuracy: 0.9900 - val_loss: 0.1329 -
val_sparse_categorical_accuracy: 0.9562 - lr: 1.0000e-04
Epoch 43/100
69/69 [==============================] - 197s 3s/step - loss: 0.0157 -
sparse_categorical_accuracy: 0.9950 - val_loss: 0.1387 -
val_sparse_categorical_accuracy: 0.9699 - lr: 1.0000e-04
Epoch 44/100
69/69 [==============================] - 198s 3s/step - loss: 1.8712e-04 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1528 -
val_sparse_categorical_accuracy: 0.9772 - lr: 1.0000e-04
Epoch 45/100
69/69 [==============================] - 199s 3s/step - loss: 2.0657e-05 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1531 -
val_sparse_categorical_accuracy: 0.9772 - lr: 1.0000e-04
Epoch 46/100
69/69 [==============================] - 198s 3s/step - loss: 1.9679e-05 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1486 -
val_sparse_categorical_accuracy: 0.9790 - lr: 1.0000e-04
Epoch 47/100
69/69 [==============================] - 199s 3s/step - loss: 2.0105e-05 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1490 -
val_sparse_categorical_accuracy: 0.9772 - lr: 1.0000e-04
Epoch 48/100
69/69 [==============================] - 199s 3s/step - loss: 7.7057e-06 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1556 -
val_sparse_categorical_accuracy: 0.9781 - lr: 1.0000e-04
Epoch 49/100
69/69 [==============================] - 199s 3s/step - loss: 1.8010e-05 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1603 -
val_sparse_categorical_accuracy: 0.9781 - lr: 1.0000e-04
Epoch 50/100
69/69 [==============================] - 202s 3s/step - loss: 9.9811e-06 -
```

```
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1618 -
val_sparse_categorical_accuracy: 0.9772 - lr: 1.0000e-04
Epoch 51/100
69/69 [==============================] - ETA: 0s - loss: 6.7273e-05 -
sparse_categorical_accuracy: 1.0000
Epoch 51: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
69/69 [==============================] - 199s 3s/step - loss: 6.7273e-05 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1432 -
val_sparse_categorical_accuracy: 0.9772 - lr: 1.0000e-04
Epoch 51: early stopping
```

[141]:
```
from matplotlib import pyplot as plt

def plot_curve(history):


    fig, ax = plt.subplots(2, 1, figsize=(20, 20))
    ax = ax.ravel()

    ax[0].set_ylim([0.4,1.1])

    ax[0].plot(history.
 ↪history["sparse_categorical_accuracy"],linewidth=3,color="#FFB81C",marker='o')
    ax[0].plot(history.history['val_' +␣
 ↪"sparse_categorical_accuracy"],linewidth=3,marker='o',color="#2D68C4")
    ax[0].set_title('Model {}'.format("Accuray"))
    ax[0].set_xlabel('Epochs')
    ax[0].set_ylabel("Accuray")
    ax[0].legend(['Train', 'Validation'])

    ax[1].set_ylim([0,1.5])

    ax[1].plot(history.history["loss"],linewidth=3,color="#FFB81C",marker='o')
    ax[1].plot(history.history['val_' + "loss"],linewidth=3,
     marker='o', markeredgecolor='black',color="#2D68C4")
    ax[1].set_title('Model {}'.format("Loss"))
    ax[1].set_xlabel('Epochs')
    ax[1].set_ylabel("loss")
    ax[1].legend(['Train', 'Validation'])


plot_curve(historyvgg)
```





[73]:
```
modelvgg.evaluate(X_test,y_test)
```

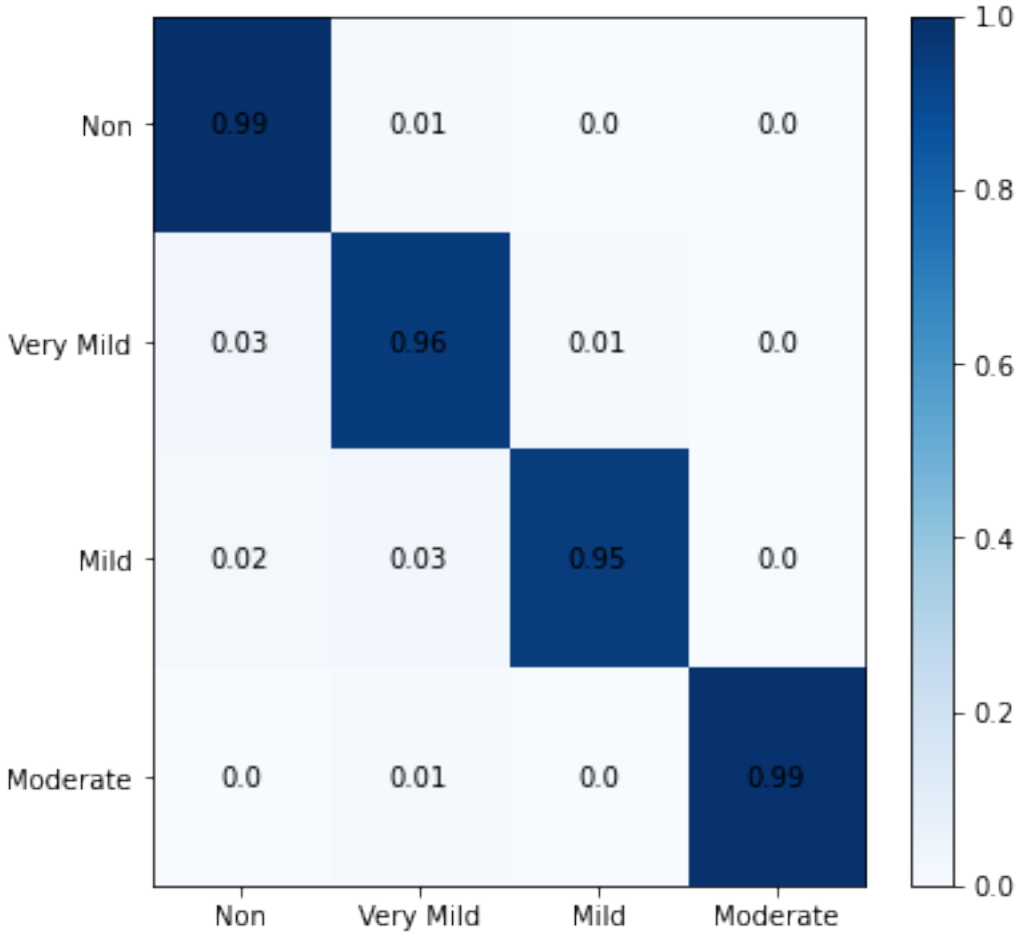
```
43/43 [==============================] - 33s 762ms/step - loss: 0.1089 -
sparse_categorical_accuracy: 0.9744
```

[73]:
```
[0.10893804579973221, 0.9744338989257812]
```

[74]:
```
y_pred = modelvgg.predict(X_test)
y_pred_1 = np.squeeze(y_pred)
y_pred_2 = np.argmax(y_pred_1,axis=1)
matrix = confusion_matrix(y_test, y_pred_2, normalize="true")
matrix
```

[74]:
```
array([[0.9921875 , 0.0078125 , 0.        , 0.        ],
       [0.03125   , 0.95535714, 0.01339286, 0.        ],
       [0.02234637, 0.02793296, 0.94972067, 0.        ],
       [0.        , 0.00980392, 0.        , 0.99019608]])
```

[75]:
```
plot_confusion_matrix(matrix)
```



7 Model 3 section

1. Model built
2. train
3. fit
4. accuracy line plot + loss line plot
5. predit
6. predit confusion matrix plot

[76]:
```
# Image_size = (128, 128, 3) for this model, reload images and no need
# to resize it
```

[77]:
```
# reload directory
train_total = pd.concat([non_train, very_mild_train, mild_train,␣
↪moderate_train])
validation_total = pd.concat([non_val, very_mild_val, mild_val, moderate_val])
test_total = pd.concat([non_test, very_mild_test, mild_test, moderate_test])

# randomize them
train_total = train_total.sample(frac=1)
validation_total = validation_total.sample(frac=1)
test_total = test_total.sample(frac=1)
```

[78]:
```
# load image using the directory df we create before

def load_image(file):
    """
    this function is...
    """
    image = cv.imread(file)
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    return image

def encoding_array(y_number):
    """
    this function is...
    """
    class_array = [0,0,0,0]
    class_array[y_number] = 1
    return class_array

def data_preparation(data_set):
    """
    this function is...
    """
    data_set["X"] = data_set["X"].apply(load_image)
    data_set["y"] = data_set["y"].apply(encoding_array)
    X = np.stack(data_set["X"])
    y = np.stack(data_set["y"])
    return X, y
```

[79]:
```
# create X, y for train, validation, test
X_train, y_train = data_preparation(train_total)
X_validation, y_validation = data_preparation(validation_total)
X_test, y_test = data_preparation(test_total)
```

[80]:
```
model3 = Sequential()

# Convolutional layer 1
model3.add(Conv2D(64,(7,7), input_shape=(128, 128, 3), padding='same',␣
 ↪activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2,2)))

#Convolutional layer 2
model3.add(Conv2D(128,(7,7), padding='same', activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2,2)))

# Convolutional layer 3
model3.add(Conv2D(128,(7,7), padding='same', activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2,2)))

# Convolutional layer 4
model3.add(Conv2D(256,(7,7), padding='same', activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2,2)))

 # Convolutional layer 5
model3.add(Conv2D(256,(7,7), padding='same', activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2,2)))

# Convolutional layer 6
model3.add(Conv2D(512,(7,7), padding='same', activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2,2)))

model3.add(Flatten())

# Full connect layers

model3.add(Dense(units= 1024, activation='relu'))
model3.add(Dropout(0.25))
model3.add(Dense(units=512, activation='relu'))
model3.add(Dropout(0.25))
model3.add(Dense(units=4, activation='softmax'))



model3.compile(optimizer=SGD(learning_rate=0.001),␣
 ↪loss='categorical_crossentropy',
```

```
                    metrics= ['categorical_accuracy'])
```

[139]:
```
model3.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 64)      9472

 batch_normalization_15 (Bat  (None, 128, 128, 64)     256
 chNormalization)

 max_pooling2d (MaxPooling2D  (None, 64, 64, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 64, 64, 128)       401536

 batch_normalization_16 (Bat  (None, 64, 64, 128)      512
 chNormalization)

 max_pooling2d_1 (MaxPooling  (None, 32, 32, 128)      0
 2D)

 conv2d_2 (Conv2D)           (None, 32, 32, 128)       802944

 batch_normalization_17 (Bat  (None, 32, 32, 128)      512
 chNormalization)

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 16, 16, 256)       1605888

 batch_normalization_18 (Bat  (None, 16, 16, 256)      1024
 chNormalization)

 max_pooling2d_3 (MaxPooling  (None, 8, 8, 256)        0
 2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 256)         3211520

 batch_normalization_19 (Bat  (None, 8, 8, 256)        1024
 chNormalization)

 max_pooling2d_4 (MaxPooling  (None, 4, 4, 256)        0
 2D)
```

```
conv2d_5 (Conv2D)            (None, 4, 4, 512)         6423040

batch_normalization_20 (Bat  (None, 4, 4, 512)         2048
chNormalization)

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

flatten_4 (Flatten)          (None, 2048)              0

dense_17 (Dense)             (None, 1024)              2098176

dropout_14 (Dropout)         (None, 1024)              0

dense_18 (Dense)             (None, 512)               524800

dropout_15 (Dropout)         (None, 512)               0

dense_19 (Dense)             (None, 4)                 2052

=================================================================
Total params: 15,084,804
Trainable params: 15,082,116
Non-trainable params: 2,688
_________________________________________________________________
```

[82]:
```
model3_es = EarlyStopping(monitor = 'loss', min_delta = 1e-11, patience = 12,␣
 ↪verbose = 1)
model3_rlr = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.2, patience =␣
 ↪6, verbose = 1)
model3_mcp = ModelCheckpoint(filepath = 'model3_weights.h5', monitor =␣
 ↪'val_categorical_accuracy',
                      save_best_only = True, verbose = 1)

history3 = model3.fit(X_train, y_train, validation_data=(X_validation,␣
 ↪y_validation), batch_size=Batch_size, epochs=Epochs, verbose = 1,
                    callbacks=[model3_es, model3_rlr, model3_mcp])
```

```
Epoch 1/100
69/69 [==============================] - ETA: 0s - loss: 1.1261 -
categorical_accuracy: 0.5219
Epoch 1: val_categorical_accuracy improved from -inf to 0.11395, saving model to
model3_weights.h5
69/69 [==============================] - 329s 5s/step - loss: 1.1261 -
categorical_accuracy: 0.5219 - val_loss: 1.3795 - val_categorical_accuracy:
0.1139 - lr: 0.0010
Epoch 2/100
69/69 [==============================] - ETA: 0s - loss: 0.9348 -
```

```
categorical_accuracy: 0.5710
Epoch 2: val_categorical_accuracy improved from 0.11395 to 0.42935, saving model
to model3_weights.h5
69/69 [==============================] - 331s 5s/step - loss: 0.9348 -
categorical_accuracy: 0.5710 - val_loss: 1.2189 - val_categorical_accuracy:
0.4294 - lr: 0.0010
Epoch 3/100
69/69 [==============================] - ETA: 0s - loss: 0.8221 -
categorical_accuracy: 0.6246
Epoch 3: val_categorical_accuracy improved from 0.42935 to 0.59799, saving model
to model3_weights.h5
69/69 [==============================] - 334s 5s/step - loss: 0.8221 -
categorical_accuracy: 0.6246 - val_loss: 1.0453 - val_categorical_accuracy:
0.5980 - lr: 0.0010
Epoch 4/100
69/69 [==============================] - ETA: 0s - loss: 0.7640 -
categorical_accuracy: 0.6568
Epoch 4: val_categorical_accuracy improved from 0.59799 to 0.61349, saving model
to model3_weights.h5
69/69 [==============================] - 333s 5s/step - loss: 0.7640 -
categorical_accuracy: 0.6568 - val_loss: 0.8110 - val_categorical_accuracy:
0.6135 - lr: 0.0010
Epoch 5/100
69/69 [==============================] - ETA: 0s - loss: 0.6572 -
categorical_accuracy: 0.7230
Epoch 5: val_categorical_accuracy improved from 0.61349 to 0.68186, saving model
to model3_weights.h5
69/69 [==============================] - 332s 5s/step - loss: 0.6572 -
categorical_accuracy: 0.7230 - val_loss: 0.7098 - val_categorical_accuracy:
0.6819 - lr: 0.0010
Epoch 6/100
69/69 [==============================] - ETA: 0s - loss: 0.6062 -
categorical_accuracy: 0.7408
Epoch 6: val_categorical_accuracy did not improve from 0.68186
69/69 [==============================] - 335s 5s/step - loss: 0.6062 -
categorical_accuracy: 0.7408 - val_loss: 0.8279 - val_categorical_accuracy:
0.5387 - lr: 0.0010
Epoch 7/100
69/69 [==============================] - ETA: 0s - loss: 0.5060 -
categorical_accuracy: 0.7944
Epoch 7: val_categorical_accuracy improved from 0.68186 to 0.78031, saving model
to model3_weights.h5
69/69 [==============================] - 333s 5s/step - loss: 0.5060 -
categorical_accuracy: 0.7944 - val_loss: 0.5384 - val_categorical_accuracy:
0.7803 - lr: 0.0010
Epoch 8/100
69/69 [==============================] - ETA: 0s - loss: 0.3837 -
categorical_accuracy: 0.8555
```

```
Epoch 8: val_categorical_accuracy improved from 0.78031 to 0.80492, saving model
to model3_weights.h5
69/69 [==============================] - 333s 5s/step - loss: 0.3837 -
categorical_accuracy: 0.8555 - val_loss: 0.5069 - val_categorical_accuracy:
0.8049 - lr: 0.0010
Epoch 9/100
69/69 [==============================] - ETA: 0s - loss: 0.2927 -
categorical_accuracy: 0.8923
Epoch 9: val_categorical_accuracy did not improve from 0.80492
69/69 [==============================] - 332s 5s/step - loss: 0.2927 -
categorical_accuracy: 0.8923 - val_loss: 0.7617 - val_categorical_accuracy:
0.6509 - lr: 0.0010
Epoch 10/100
69/69 [==============================] - ETA: 0s - loss: 0.1978 -
categorical_accuracy: 0.9334
Epoch 10: val_categorical_accuracy improved from 0.80492 to 0.87511, saving
model to model3_weights.h5
69/69 [==============================] - 333s 5s/step - loss: 0.1978 -
categorical_accuracy: 0.9334 - val_loss: 0.3264 - val_categorical_accuracy:
0.8751 - lr: 0.0010
Epoch 11/100
69/69 [==============================] - ETA: 0s - loss: 0.1172 -
categorical_accuracy: 0.9674
Epoch 11: val_categorical_accuracy did not improve from 0.87511
69/69 [==============================] - 333s 5s/step - loss: 0.1172 -
categorical_accuracy: 0.9674 - val_loss: 0.3825 - val_categorical_accuracy:
0.8414 - lr: 0.0010
Epoch 12/100
69/69 [==============================] - ETA: 0s - loss: 0.0653 -
categorical_accuracy: 0.9845
Epoch 12: val_categorical_accuracy did not improve from 0.87511
69/69 [==============================] - 332s 5s/step - loss: 0.0653 -
categorical_accuracy: 0.9845 - val_loss: 0.3690 - val_categorical_accuracy:
0.8541 - lr: 0.0010
Epoch 13/100
69/69 [==============================] - ETA: 0s - loss: 0.0411 -
categorical_accuracy: 0.9916
Epoch 13: val_categorical_accuracy improved from 0.87511 to 0.90337, saving
model to model3_weights.h5
69/69 [==============================] - 334s 5s/step - loss: 0.0411 -
categorical_accuracy: 0.9916 - val_loss: 0.2540 - val_categorical_accuracy:
0.9034 - lr: 0.0010
Epoch 14/100
69/69 [==============================] - ETA: 0s - loss: 0.0238 -
categorical_accuracy: 0.9963
Epoch 14: val_categorical_accuracy improved from 0.90337 to 0.95077, saving
model to model3_weights.h5
69/69 [==============================] - 335s 5s/step - loss: 0.0238 -
```

```
categorical_accuracy: 0.9963 - val_loss: 0.1423 - val_categorical_accuracy:
0.9508 - lr: 0.0010
Epoch 15/100
69/69 [==============================] - ETA: 0s - loss: 0.0279 -
categorical_accuracy: 0.9936
Epoch 15: val_categorical_accuracy did not improve from 0.95077
69/69 [==============================] - 333s 5s/step - loss: 0.0279 -
categorical_accuracy: 0.9936 - val_loss: 0.2269 - val_categorical_accuracy:
0.9070 - lr: 0.0010
Epoch 16/100
69/69 [==============================] - ETA: 0s - loss: 0.0200 -
categorical_accuracy: 0.9968
Epoch 16: val_categorical_accuracy improved from 0.95077 to 0.95989, saving
model to model3_weights.h5
69/69 [==============================] - 335s 5s/step - loss: 0.0200 -
categorical_accuracy: 0.9968 - val_loss: 0.1280 - val_categorical_accuracy:
0.9599 - lr: 0.0010
Epoch 17/100
69/69 [==============================] - ETA: 0s - loss: 0.0152 -
categorical_accuracy: 0.9973
Epoch 17: val_categorical_accuracy did not improve from 0.95989
69/69 [==============================] - 333s 5s/step - loss: 0.0152 -
categorical_accuracy: 0.9973 - val_loss: 0.2611 - val_categorical_accuracy:
0.8851 - lr: 0.0010
Epoch 18/100
69/69 [==============================] - ETA: 0s - loss: 0.0144 -
categorical_accuracy: 0.9970
Epoch 18: val_categorical_accuracy did not improve from 0.95989
69/69 [==============================] - 332s 5s/step - loss: 0.0144 -
categorical_accuracy: 0.9970 - val_loss: 0.1723 - val_categorical_accuracy:
0.9499 - lr: 0.0010
Epoch 19/100
69/69 [==============================] - ETA: 0s - loss: 0.0080 -
categorical_accuracy: 0.9998
Epoch 19: val_categorical_accuracy improved from 0.95989 to 0.96536, saving
model to model3_weights.h5
69/69 [==============================] - 332s 5s/step - loss: 0.0080 -
categorical_accuracy: 0.9998 - val_loss: 0.1091 - val_categorical_accuracy:
0.9654 - lr: 0.0010
Epoch 20/100
69/69 [==============================] - ETA: 0s - loss: 0.0065 -
categorical_accuracy: 0.9995
Epoch 20: val_categorical_accuracy did not improve from 0.96536
69/69 [==============================] - 332s 5s/step - loss: 0.0065 -
categorical_accuracy: 0.9995 - val_loss: 0.2410 - val_categorical_accuracy:
0.9070 - lr: 0.0010
Epoch 21/100
69/69 [==============================] - ETA: 0s - loss: 0.0082 -
```

```
categorical_accuracy: 0.9991
Epoch 21: val_categorical_accuracy did not improve from 0.96536
69/69 [==============================] - 332s 5s/step - loss: 0.0082 -
categorical_accuracy: 0.9991 - val_loss: 0.1197 - val_categorical_accuracy:
0.9608 - lr: 0.0010
Epoch 22/100
69/69 [==============================] - ETA: 0s - loss: 0.0079 -
categorical_accuracy: 0.9989
Epoch 22: val_categorical_accuracy did not improve from 0.96536
69/69 [==============================] - 333s 5s/step - loss: 0.0079 -
categorical_accuracy: 0.9989 - val_loss: 1.7107 - val_categorical_accuracy:
0.6272 - lr: 0.0010
Epoch 23/100
69/69 [==============================] - ETA: 0s - loss: 0.0130 -
categorical_accuracy: 0.9973
Epoch 23: val_categorical_accuracy did not improve from 0.96536
69/69 [==============================] - 334s 5s/step - loss: 0.0130 -
categorical_accuracy: 0.9973 - val_loss: 0.1924 - val_categorical_accuracy:
0.9325 - lr: 0.0010
Epoch 24/100
69/69 [==============================] - ETA: 0s - loss: 0.0083 -
categorical_accuracy: 0.9984
Epoch 24: val_categorical_accuracy did not improve from 0.96536
69/69 [==============================] - 334s 5s/step - loss: 0.0083 -
categorical_accuracy: 0.9984 - val_loss: 0.1354 - val_categorical_accuracy:
0.9572 - lr: 0.0010
Epoch 25/100
69/69 [==============================] - ETA: 0s - loss: 0.0075 -
categorical_accuracy: 0.9989
Epoch 25: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.

Epoch 25: val_categorical_accuracy did not improve from 0.96536
69/69 [==============================] - 332s 5s/step - loss: 0.0075 -
categorical_accuracy: 0.9989 - val_loss: 0.1175 - val_categorical_accuracy:
0.9644 - lr: 0.0010
Epoch 26/100
69/69 [==============================] - ETA: 0s - loss: 0.0056 -
categorical_accuracy: 0.9991
Epoch 26: val_categorical_accuracy improved from 0.96536 to 0.96992, saving
model to model3_weights.h5
69/69 [==============================] - 331s 5s/step - loss: 0.0056 -
categorical_accuracy: 0.9991 - val_loss: 0.0976 - val_categorical_accuracy:
0.9699 - lr: 2.0000e-04
Epoch 27/100
69/69 [==============================] - ETA: 0s - loss: 0.0040 -
categorical_accuracy: 0.9998
Epoch 27: val_categorical_accuracy did not improve from 0.96992
69/69 [==============================] - 333s 5s/step - loss: 0.0040 -
```

```
categorical_accuracy: 0.9998 - val_loss: 0.1115 - val_categorical_accuracy:
0.9672 - lr: 2.0000e-04
Epoch 28/100
69/69 [==============================] - ETA: 0s - loss: 0.0032 -
categorical_accuracy: 0.9998
Epoch 28: val_categorical_accuracy improved from 0.96992 to 0.97174, saving
model to model3_weights.h5
69/69 [==============================] - 332s 5s/step - loss: 0.0032 -
categorical_accuracy: 0.9998 - val_loss: 0.0937 - val_categorical_accuracy:
0.9717 - lr: 2.0000e-04
Epoch 29/100
69/69 [==============================] - ETA: 0s - loss: 0.0029 -
categorical_accuracy: 0.9998
Epoch 29: val_categorical_accuracy improved from 0.97174 to 0.97356, saving
model to model3_weights.h5
69/69 [==============================] - 334s 5s/step - loss: 0.0029 -
categorical_accuracy: 0.9998 - val_loss: 0.0935 - val_categorical_accuracy:
0.9736 - lr: 2.0000e-04
Epoch 30/100
69/69 [==============================] - ETA: 0s - loss: 0.0036 -
categorical_accuracy: 0.9995
Epoch 30: val_categorical_accuracy improved from 0.97356 to 0.97448, saving
model to model3_weights.h5
69/69 [==============================] - 332s 5s/step - loss: 0.0036 -
categorical_accuracy: 0.9995 - val_loss: 0.0931 - val_categorical_accuracy:
0.9745 - lr: 2.0000e-04
Epoch 31/100
69/69 [==============================] - ETA: 0s - loss: 0.0038 -
categorical_accuracy: 0.9995
Epoch 31: val_categorical_accuracy did not improve from 0.97448
69/69 [==============================] - 335s 5s/step - loss: 0.0038 -
categorical_accuracy: 0.9995 - val_loss: 0.1117 - val_categorical_accuracy:
0.9663 - lr: 2.0000e-04
Epoch 32/100
69/69 [==============================] - ETA: 0s - loss: 0.0027 -
categorical_accuracy: 1.0000
Epoch 32: val_categorical_accuracy did not improve from 0.97448
69/69 [==============================] - 334s 5s/step - loss: 0.0027 -
categorical_accuracy: 1.0000 - val_loss: 0.0938 - val_categorical_accuracy:
0.9727 - lr: 2.0000e-04
Epoch 33/100
69/69 [==============================] - ETA: 0s - loss: 0.0033 -
categorical_accuracy: 0.9998
Epoch 33: val_categorical_accuracy did not improve from 0.97448
69/69 [==============================] - 334s 5s/step - loss: 0.0033 -
categorical_accuracy: 0.9998 - val_loss: 0.0933 - val_categorical_accuracy:
0.9699 - lr: 2.0000e-04
Epoch 34/100
```

```
69/69 [==============================] - ETA: 0s - loss: 0.0036 -
categorical_accuracy: 0.9995
Epoch 34: val_categorical_accuracy did not improve from 0.97448
69/69 [==============================] - 333s 5s/step - loss: 0.0036 -
categorical_accuracy: 0.9995 - val_loss: 0.0977 - val_categorical_accuracy:
0.9717 - lr: 2.0000e-04
Epoch 35/100
69/69 [==============================] - ETA: 0s - loss: 0.0037 -
categorical_accuracy: 0.9998
Epoch 35: val_categorical_accuracy did not improve from 0.97448
69/69 [==============================] - 334s 5s/step - loss: 0.0037 -
categorical_accuracy: 0.9998 - val_loss: 0.0933 - val_categorical_accuracy:
0.9736 - lr: 2.0000e-04
Epoch 36/100
69/69 [==============================] - ETA: 0s - loss: 0.0028 -
categorical_accuracy: 0.9998
Epoch 36: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.

Epoch 36: val_categorical_accuracy did not improve from 0.97448
69/69 [==============================] - 332s 5s/step - loss: 0.0028 -
categorical_accuracy: 0.9998 - val_loss: 0.0939 - val_categorical_accuracy:
0.9745 - lr: 2.0000e-04
Epoch 37/100
69/69 [==============================] - ETA: 0s - loss: 0.0028 -
categorical_accuracy: 0.9998
Epoch 37: val_categorical_accuracy did not improve from 0.97448
69/69 [==============================] - 333s 5s/step - loss: 0.0028 -
categorical_accuracy: 0.9998 - val_loss: 0.0945 - val_categorical_accuracy:
0.9745 - lr: 4.0000e-05
Epoch 38/100
69/69 [==============================] - ETA: 0s - loss: 0.0019 -
categorical_accuracy: 1.0000
Epoch 38: val_categorical_accuracy did not improve from 0.97448
69/69 [==============================] - 332s 5s/step - loss: 0.0019 -
categorical_accuracy: 1.0000 - val_loss: 0.0940 - val_categorical_accuracy:
0.9745 - lr: 4.0000e-05
Epoch 39/100
69/69 [==============================] - ETA: 0s - loss: 0.0043 -
categorical_accuracy: 0.9995
Epoch 39: val_categorical_accuracy did not improve from 0.97448
69/69 [==============================] - 332s 5s/step - loss: 0.0043 -
categorical_accuracy: 0.9995 - val_loss: 0.0941 - val_categorical_accuracy:
0.9745 - lr: 4.0000e-05
Epoch 40/100
69/69 [==============================] - ETA: 0s - loss: 0.0033 -
categorical_accuracy: 0.9993
Epoch 40: val_categorical_accuracy did not improve from 0.97448
69/69 [==============================] - 333s 5s/step - loss: 0.0033 -
```

```
categorical_accuracy: 0.9993 - val_loss: 0.0927 - val_categorical_accuracy:
0.9745 - lr: 4.0000e-05
Epoch 41/100
69/69 [==============================] - ETA: 0s - loss: 0.0024 -
categorical_accuracy: 0.9998
Epoch 41: val_categorical_accuracy did not improve from 0.97448
69/69 [==============================] - 335s 5s/step - loss: 0.0024 -
categorical_accuracy: 0.9998 - val_loss: 0.0932 - val_categorical_accuracy:
0.9745 - lr: 4.0000e-05
Epoch 42/100
69/69 [==============================] - ETA: 0s - loss: 0.0024 -
categorical_accuracy: 0.9998
Epoch 42: val_categorical_accuracy did not improve from 0.97448
69/69 [==============================] - 334s 5s/step - loss: 0.0024 -
categorical_accuracy: 0.9998 - val_loss: 0.0926 - val_categorical_accuracy:
0.9745 - lr: 4.0000e-05
Epoch 43/100
69/69 [==============================] - ETA: 0s - loss: 0.0034 -
categorical_accuracy: 0.9998
Epoch 43: val_categorical_accuracy improved from 0.97448 to 0.97539, saving
model to model3_weights.h5
69/69 [==============================] - 333s 5s/step - loss: 0.0034 -
categorical_accuracy: 0.9998 - val_loss: 0.0932 - val_categorical_accuracy:
0.9754 - lr: 4.0000e-05
Epoch 44/100
69/69 [==============================] - ETA: 0s - loss: 0.0027 -
categorical_accuracy: 1.0000
Epoch 44: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 333s 5s/step - loss: 0.0027 -
categorical_accuracy: 1.0000 - val_loss: 0.0928 - val_categorical_accuracy:
0.9754 - lr: 4.0000e-05
Epoch 45/100
69/69 [==============================] - ETA: 0s - loss: 0.0022 -
categorical_accuracy: 1.0000
Epoch 45: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 332s 5s/step - loss: 0.0022 -
categorical_accuracy: 1.0000 - val_loss: 0.0934 - val_categorical_accuracy:
0.9754 - lr: 4.0000e-05
Epoch 46/100
69/69 [==============================] - ETA: 0s - loss: 0.0021 -
categorical_accuracy: 1.0000
Epoch 46: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.

Epoch 46: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 332s 5s/step - loss: 0.0021 -
categorical_accuracy: 1.0000 - val_loss: 0.0935 - val_categorical_accuracy:
0.9754 - lr: 4.0000e-05
Epoch 47/100
```

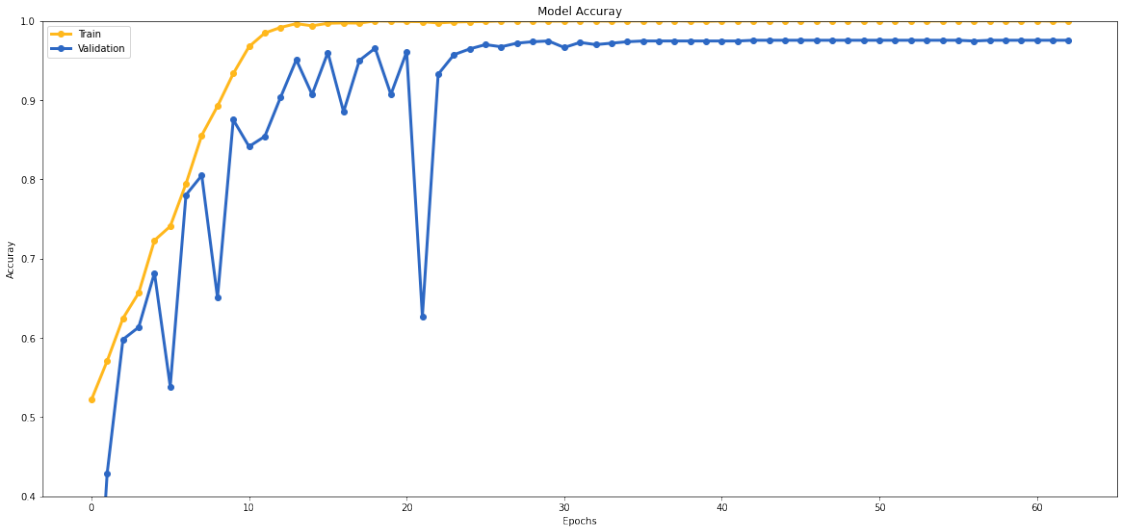
```
69/69 [==============================] - ETA: 0s - loss: 0.0027 -
categorical_accuracy: 1.0000
Epoch 47: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 334s 5s/step - loss: 0.0027 -
categorical_accuracy: 1.0000 - val_loss: 0.0934 - val_categorical_accuracy:
0.9754 - lr: 8.0000e-06
Epoch 48/100
69/69 [==============================] - ETA: 0s - loss: 0.0019 -
categorical_accuracy: 1.0000
Epoch 48: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 333s 5s/step - loss: 0.0019 -
categorical_accuracy: 1.0000 - val_loss: 0.0929 - val_categorical_accuracy:
0.9754 - lr: 8.0000e-06
Epoch 49/100
69/69 [==============================] - ETA: 0s - loss: 0.0025 -
categorical_accuracy: 1.0000
Epoch 49: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 338s 5s/step - loss: 0.0025 -
categorical_accuracy: 1.0000 - val_loss: 0.0933 - val_categorical_accuracy:
0.9754 - lr: 8.0000e-06
Epoch 50/100
69/69 [==============================] - ETA: 0s - loss: 0.0030 -
categorical_accuracy: 0.9998
Epoch 50: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 333s 5s/step - loss: 0.0030 -
categorical_accuracy: 0.9998 - val_loss: 0.0934 - val_categorical_accuracy:
0.9754 - lr: 8.0000e-06
Epoch 51/100
69/69 [==============================] - ETA: 0s - loss: 0.0018 -
categorical_accuracy: 1.0000
Epoch 51: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 337s 5s/step - loss: 0.0018 -
categorical_accuracy: 1.0000 - val_loss: 0.0933 - val_categorical_accuracy:
0.9754 - lr: 8.0000e-06
Epoch 52/100
69/69 [==============================] - ETA: 0s - loss: 0.0021 -
categorical_accuracy: 1.0000
Epoch 52: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.

Epoch 52: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 333s 5s/step - loss: 0.0021 -
categorical_accuracy: 1.0000 - val_loss: 0.0935 - val_categorical_accuracy:
0.9754 - lr: 8.0000e-06
Epoch 53/100
69/69 [==============================] - ETA: 0s - loss: 0.0021 -
categorical_accuracy: 1.0000
Epoch 53: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 334s 5s/step - loss: 0.0021 -
```

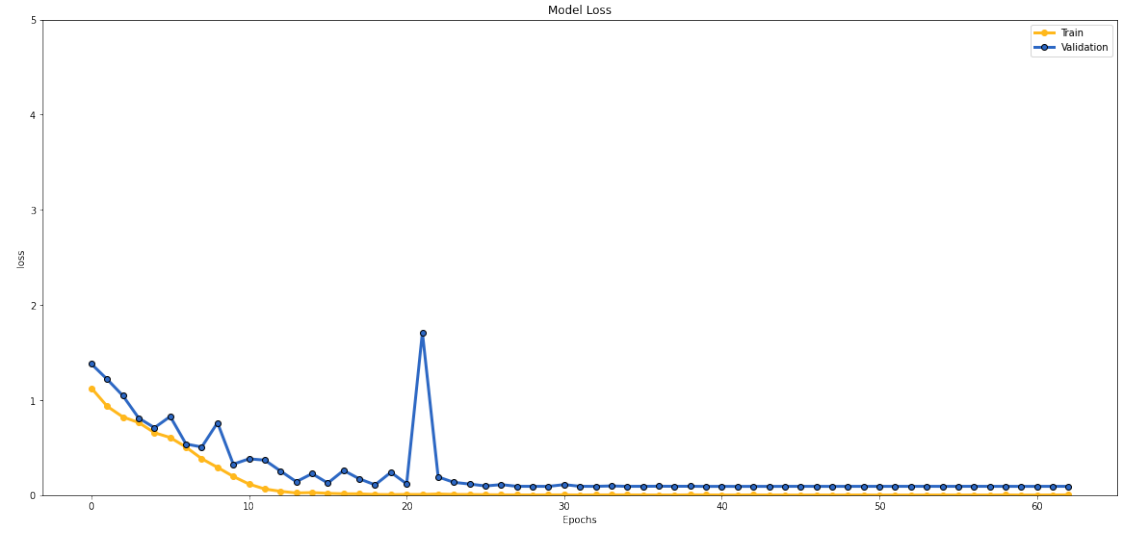
```
categorical_accuracy: 1.0000 - val_loss: 0.0934 - val_categorical_accuracy:
0.9754 - lr: 1.6000e-06
Epoch 54/100
69/69 [==============================] - ETA: 0s - loss: 0.0024 -
categorical_accuracy: 0.9998
Epoch 54: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 335s 5s/step - loss: 0.0024 -
categorical_accuracy: 0.9998 - val_loss: 0.0935 - val_categorical_accuracy:
0.9754 - lr: 1.6000e-06
Epoch 55/100
69/69 [==============================] - ETA: 0s - loss: 0.0026 -
categorical_accuracy: 0.9998
Epoch 55: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 333s 5s/step - loss: 0.0026 -
categorical_accuracy: 0.9998 - val_loss: 0.0933 - val_categorical_accuracy:
0.9754 - lr: 1.6000e-06
Epoch 56/100
69/69 [==============================] - ETA: 0s - loss: 0.0021 -
categorical_accuracy: 1.0000
Epoch 56: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 343s 5s/step - loss: 0.0021 -
categorical_accuracy: 1.0000 - val_loss: 0.0934 - val_categorical_accuracy:
0.9754 - lr: 1.6000e-06
Epoch 57/100
69/69 [==============================] - ETA: 0s - loss: 0.0027 -
categorical_accuracy: 0.9998
Epoch 57: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 343s 5s/step - loss: 0.0027 -
categorical_accuracy: 0.9998 - val_loss: 0.0938 - val_categorical_accuracy:
0.9745 - lr: 1.6000e-06
Epoch 58/100
69/69 [==============================] - ETA: 0s - loss: 0.0024 -
categorical_accuracy: 1.0000
Epoch 58: ReduceLROnPlateau reducing learning rate to 3.200000264769187e-07.

Epoch 58: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 335s 5s/step - loss: 0.0024 -
categorical_accuracy: 1.0000 - val_loss: 0.0933 - val_categorical_accuracy:
0.9754 - lr: 1.6000e-06
Epoch 59/100
69/69 [==============================] - ETA: 0s - loss: 0.0032 -
categorical_accuracy: 0.9998
Epoch 59: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 337s 5s/step - loss: 0.0032 -
categorical_accuracy: 0.9998 - val_loss: 0.0934 - val_categorical_accuracy:
0.9754 - lr: 3.2000e-07
Epoch 60/100
69/69 [==============================] - ETA: 0s - loss: 0.0020 -
```

```
categorical_accuracy: 0.9998
Epoch 60: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 337s 5s/step - loss: 0.0020 -
categorical_accuracy: 0.9998 - val_loss: 0.0931 - val_categorical_accuracy:
0.9754 - lr: 3.2000e-07
Epoch 61/100
69/69 [==============================] - ETA: 0s - loss: 0.0021 -
categorical_accuracy: 1.0000
Epoch 61: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 337s 5s/step - loss: 0.0021 -
categorical_accuracy: 1.0000 - val_loss: 0.0932 - val_categorical_accuracy:
0.9754 - lr: 3.2000e-07
Epoch 62/100
69/69 [==============================] - ETA: 0s - loss: 0.0027 -
categorical_accuracy: 0.9995
Epoch 62: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 337s 5s/step - loss: 0.0027 -
categorical_accuracy: 0.9995 - val_loss: 0.0935 - val_categorical_accuracy:
0.9754 - lr: 3.2000e-07
Epoch 63/100
69/69 [==============================] - ETA: 0s - loss: 0.0039 -
categorical_accuracy: 0.9995
Epoch 63: val_categorical_accuracy did not improve from 0.97539
69/69 [==============================] - 347s 5s/step - loss: 0.0039 -
categorical_accuracy: 0.9995 - val_loss: 0.0934 - val_categorical_accuracy:
0.9754 - lr: 3.2000e-07
Epoch 63: early stopping
```

[142]:
```
def plot_curve(history):


    fig, ax = plt.subplots(2, 1, figsize=(20, 20))
    ax = ax.ravel()

    ax[0].set_ylim([0.4,1])

    ax[0].plot(history.
↪history["categorical_accuracy"],linewidth=3,color="#FFB81C",marker='o')
    ax[0].plot(history.
↪history['val_categorical_accuracy'],linewidth=3,marker='o',color="#2D68C4")
    ax[0].set_title('Model {}'.format("Accuray"))
    ax[0].set_xlabel('Epochs')
    ax[0].set_ylabel("Accuray")
    ax[0].legend(['Train', 'Validation'])

    ax[1].set_ylim([0,5])
```

```
    ax[1].plot(history.history["loss"],linewidth=3,color="#FFB81C",marker='o')
    ax[1].plot(history.history['val_' + "loss"],linewidth=3,
     marker='o', markeredgecolor='black',color="#2D68C4")
    ax[1].set_title('Model {}'.format("Loss"))
    ax[1].set_xlabel('Epochs')
    ax[1].set_ylabel("loss")
    ax[1].legend(['Train', 'Validation'])

plot_curve(history3)
```





[84]:
```
model3.evaluate(X_test,y_test)
```

```
43/43 [==============================] - 25s 590ms/step - loss: 0.0827 -
categorical_accuracy: 0.9744
```

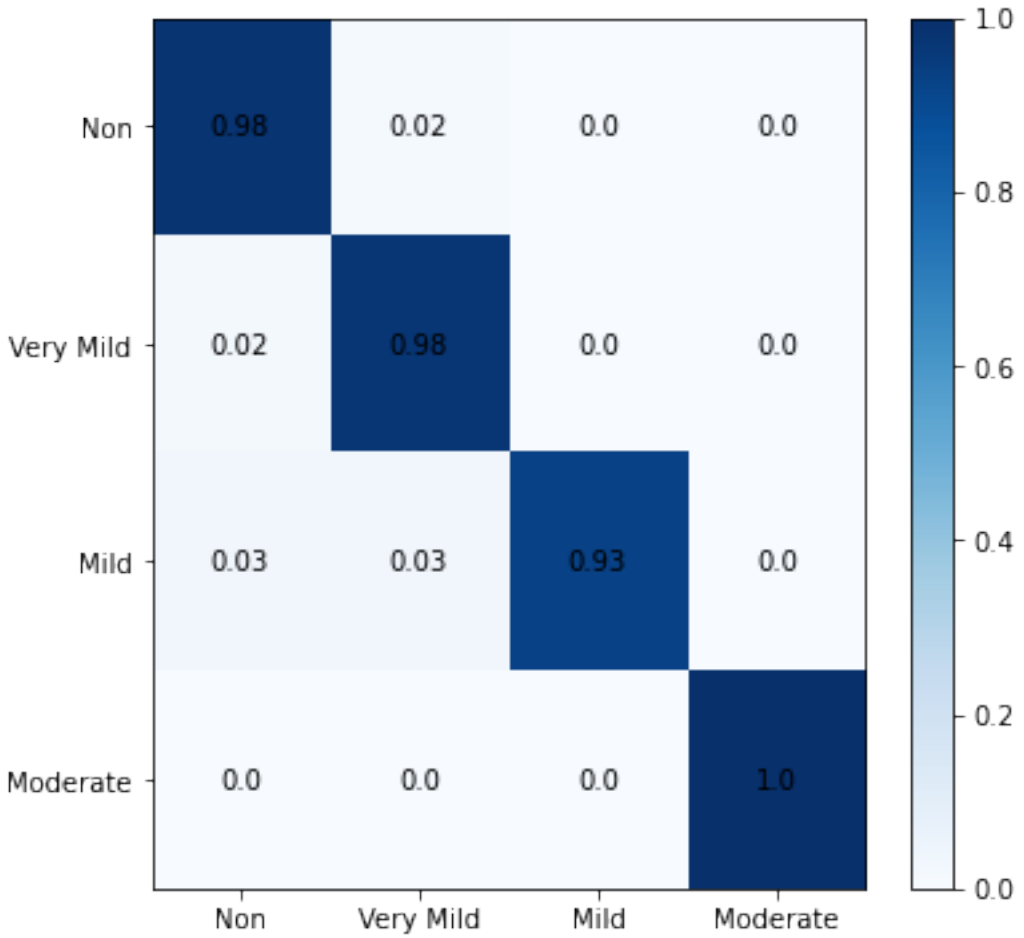
[84]: [0.08266450464725494, 0.9744338989257812]

[85]:
```
y_pred = model3.predict(X_test)
y_pred_1 = np.squeeze(y_pred)
y_pred_2 = np.argmax(y_pred_1,axis=1)
```

[86]:
```
y_new_test = y_test.argmax(axis=1)
matrix = confusion_matrix(y_new_test, y_pred_2, normalize="true")
matrix
```

[86]:
```
array([[0.98125   , 0.015625  , 0.003125  , 0.        ],
       [0.02232143, 0.97544643, 0.00223214, 0.        ],
       [0.03351955, 0.03351955, 0.93296089, 0.        ],
       [0.        , 0.        , 0.        , 1.        ]])
```

[87]:
```
plot_confusion_matrix(matrix)
```



Environment Set up

Set up 1 - Data load - Cassie

In [3]:

```
import os
from os.path import join
from glob import glob
import numpy as np
import pandas as pd
import tensorflow as tf

import matplotlib.pyplot as plt
import PIL
import random

from tensorflow.keras.callbacks import ReduceLROnPlateau

import cv2 as cv

from sklearn.metrics import accuracy_score, balanced_accuracy_score, roc_auc_score, \
    confusion_matrix, precision_score, recall_score, f1_score
import seaborn as sns
```

Set up 2 - Model 1 - Cassie

In [4]:

```
from tensorflow.keras.models import Sequential
from keras.models import Sequential,load_model
from tensorflow.keras.layers import InputLayer, BatchNormalization, Dropout, Flatten, Dense, Activa
from keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.callbacks import Callback,ModelCheckpoint,ReduceLROnPlateau
from keras import backend as K
```

Load Datasets

In [5]:

```
# define two functions to load 4 types of data
def import_data(directory, foldername, y, typename):
    """
    this funtion ...
    """
    data_df = pd.DataFrame({
        "X": sorted(glob(join(directory, foldername, "*"))),
        "y": y,
        "type": typename
    })
    return data_df
```

In [6]:

```
# set the directory of data import

directory = "C:\\Users\\lenovo\\Desktop\\python\\Alzheimer's"

# build four dataframe
non_demented_df = import_data(directory, ".\\NonDemented", 0, "NonDemented")
very_mild_df = import_data(directory, ".\\VeryMildDemented", 1, "VeryMildDemented")
mild_df = import_data(directory, ".\\MildDemented", 2, "MildDemented")
moderate_df = import_data(directory, ".\\ModerateDemented", 3, "ModerateDemented")
```

Data Preparation

4 directory dataframe

1. split 4 types of data into train, test, validation.
2. Set parameters
3. concat the 4 dataframe

Split 4 types of data

In [7]:

```
def random_split_data(data_df):
    """
    this function...
    """
    random_data = data_df.sample(frac=1)
    length = len(random_data)
    test = random_data[:int(0.2*length)]
    remaining = random_data[int(0.2*length):]
    length_2 = len(remaining)
    train = remaining[:int(0.8*length_2)]
    validation = remaining[int(0.8*length_2):]
    return test, train, validation
```

In [8]:

```
# we only use train here and split train dataset to 0.8/0.2 ratio of train/test,
# and in new train, we split it to 0.8/0.2 ration of train/validation


# split the four dataframe seperately
non_test, non_train, non_val = random_split_data(non_demented_df)
very_mild_test, very_mild_train, very_mild_val = random_split_data(very_mild_df)
mild_test, mild_train, mild_val = random_split_data(mild_df)
moderate_test, moderate_train, moderate_val = random_split_data(moderate_df)
```

Set parameters

In [9]:

```
Autotune = tf.data.experimental.AUTOTUNE
```

In [12]:

```
Image_size = [224, 224]
Batch_size = 64
Epochs = 80 # needed to change to 50 afterwards
```

Concat the 4 dataframe and randomize it

In [13]:

```
# concat the 4 data frame
train_total = pd.concat([non_train, very_mild_train, mild_train, moderate_train])
validation_total = pd.concat([non_val, very_mild_val, mild_val, moderate_val])
test_total = pd.concat([non_test, very_mild_test, mild_test, moderate_test])

# randomize them
train_total = train_total.sample(frac=1)
validation_total = validation_total.sample(frac=1)
test_total = test_total.sample(frac=1)
```

Build three functions to load image, create class array and extract X, y for each train, validation and test.

In [14]:

```
# load image using the directory df we create before

def load_image(file):
    """
    this function is...
    """
    image = cv.imread(file)
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    # resize it to a same size
    image = cv.resize(image, (Image_size[0], Image_size[1]))
    return image

def encoding_array(y_number):
    """
    this function is...
    """
    class_array = [0,0,0,0]
    class_array[y_number] = 1
    return class_array

def data_preparation(data_set):
    """
    this function is...
    """
    data_set["X"] = data_set["X"].apply(load_image)
    data_set["y"] = data_set["y"].apply(encoding_array)
    X = np.stack(data_set["X"])
    y = np.stack(data_set["y"])
    return X, y
```

In [15]:

```
# create X, y for train, validation, test
X_train, y_train = data_preparation(train_total)
X_validation, y_validation = data_preparation(validation_total)
X_test, y_test = data_preparation(test_total)
```

Model 1 section

1. Model built
2. train
3. fit
4. accuracy line plot + loss line plot
5. predit
6. predit confusion matrix plot

In [16]:

```
# import base model
base_model = tf.keras.applications.NASNetMobile(input_shape=(224,224,3),
                                                 include_top=False, weights="imagenet")
```

In [17]:

```
#Freezing Layers
for layer in base_model.layers[:-4]:
    layer.trainable=False
```

In [18]:

```
# Building Model

model=Sequential()
model.add(base_model)
model.add(Dropout(0.5))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(4,activation='softmax'))
```

In [19]:

```
# Model Summary

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 NASNet (Functional)         (None, 7, 7, 1056)        4269716

 dropout (Dropout)           (None, 7, 7, 1056)        0

 flatten (Flatten)           (None, 51744)             0

 batch_normalization (BatchN  (None, 51744)            206976
 ormalization)

 dense (Dense)               (None, 32)                1655840

 batch_normalization_1 (Batc  (None, 32)               128
 hNormalization)

 activation_188 (Activation)  (None, 32)               0

 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 32)                1056

 batch_normalization_2 (Batc  (None, 32)               128
 hNormalization)

 activation_189 (Activation)  (None, 32)               0

 dropout_2 (Dropout)         (None, 32)                0

 dense_2 (Dense)             (None, 32)                1056

 batch_normalization_3 (Batc  (None, 32)               128
 hNormalization)

 activation_190 (Activation)  (None, 32)               0

 dense_3 (Dense)             (None, 4)                 132

=================================================================
Total params: 6,135,160
Trainable params: 1,761,764
Non-trainable params: 4,373,396
_________________________________________________________________
```

In [20]:

```
def f1_score(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

In [21]:

```
METRICS = [
      tf.keras.metrics.CategoricalAccuracy(name='accuracy'),
      tf.keras.metrics.Precision(name='precision'),
      tf.keras.metrics.Recall(name='recall'),
      tf.keras.metrics.AUC(name='auc'),
      f1_score,
]
```

In [22]:

```
lrd = ReduceLROnPlateau(monitor = 'val_loss',patience = 20,verbose = 1,factor = 0.50, min_lr = 1e-1

mcp = ModelCheckpoint('model.h5')

es = EarlyStopping(verbose=1, patience=20)
```

In [23]:

```
model.compile(optimizer='Adam', loss='categorical_crossentropy',metrics=METRICS)
```

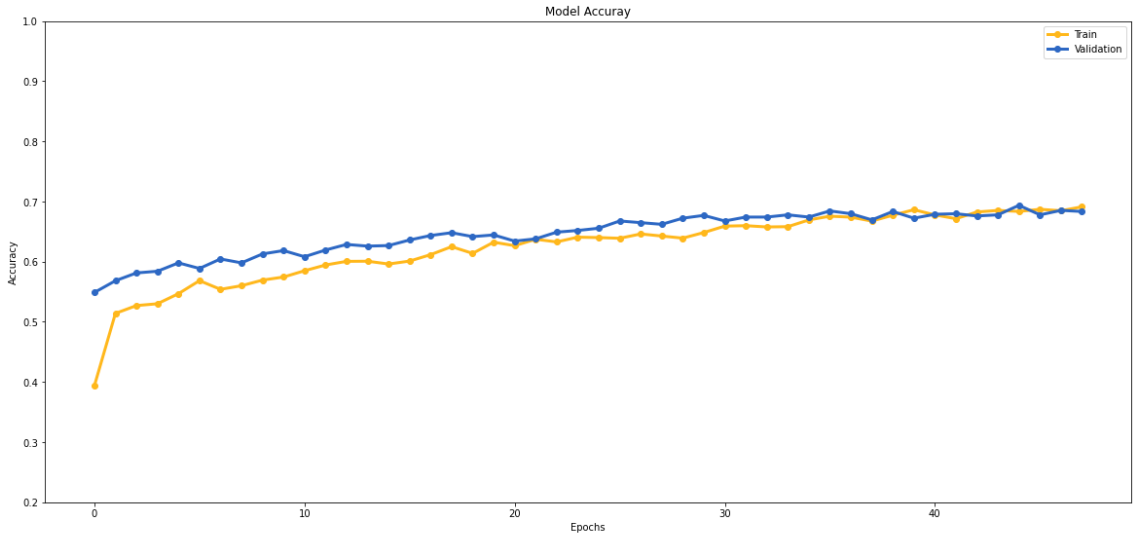
In [24]:

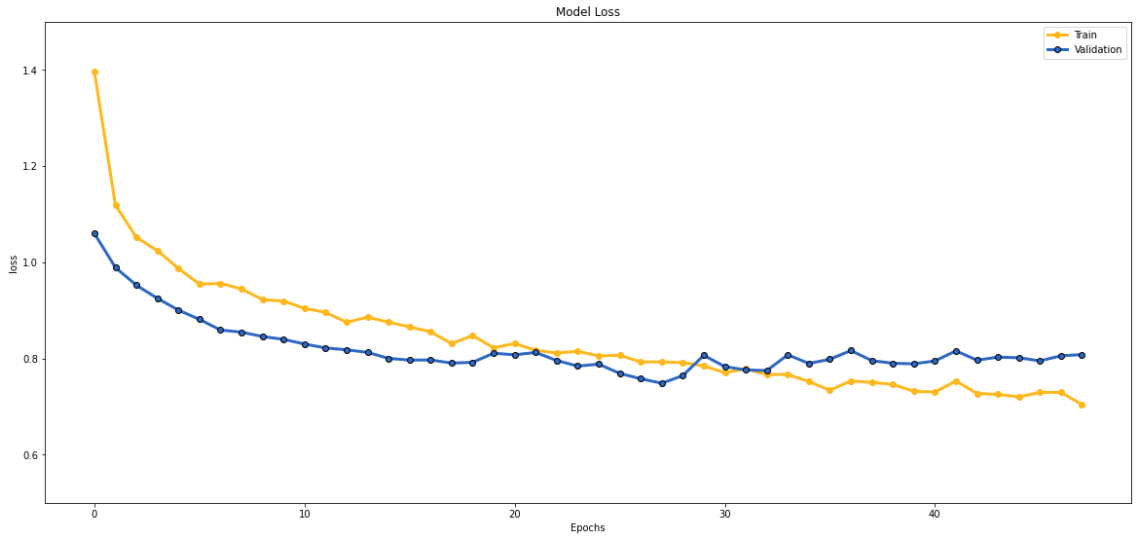
```
history = model.fit(X_train, y_train, validation_data=(X_validation, y_validation), batch_size=Batch
```

```
Epoch 46/80
68/68 [==============================] - 333s 5s/step - loss: 0.7293 - accuracy:
0.6870 - precision: 0.7322 - recall: 0.6105 - auc: 0.9048 - f1_score: 0.6674 - va
l_loss: 0.7949 - val_accuracy: 0.6778 - val_precision: 0.7442 - val_recall: 0.591
5 - val_auc: 0.9034 - val_f1_score: 0.6595 - lr: 0.0010
Epoch 47/80
68/68 [==============================] - 332s 5s/step - loss: 0.7294 - accuracy:
0.6847 - precision: 0.7310 - recall: 0.5960 - auc: 0.9043 - f1_score: 0.6571 - va
l_loss: 0.8054 - val_accuracy: 0.6852 - val_precision: 0.7420 - val_recall: 0.585
0 - val_auc: 0.9034 - val_f1_score: 0.6551 - lr: 0.0010
Epoch 48/80
68/68 [==============================] - ETA: 0s - loss: 0.7041 - accuracy: 0.691
2 - precision: 0.7478 - recall: 0.6063 - auc: 0.9112 - f1_score: 0.6713
Epoch 48: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
68/68 [==============================] - 330s 5s/step - loss: 0.7041 - accuracy:
0.6912 - precision: 0.7478 - recall: 0.6063 - auc: 0.9112 - f1_score: 0.6713 - va
l_loss: 0.8079 - val_accuracy: 0.6834 - val_precision: 0.7580 - val_recall: 0.596
1 - val_auc: 0.9069 - val_f1_score: 0.6679 - lr: 0.0010
Epoch 48: early stopping
```

In [28]:

```
def plot_curve(history):


    fig, ax = plt.subplots(2, 1, figsize=(20, 20))
    ax = ax.ravel()

    ax[0].set_ylim([0.2,1])

    ax[0].plot(history.history["accuracy"],linewidth=3,color="#FFB81C",marker='o')
    ax[0].plot(history.history["val_accuracy"],linewidth=3,marker='o',color="#2D68C4")
    ax[0].set_title('Model {}'.format("Accuray"))
    ax[0].set_xlabel('Epochs')
    ax[0].set_ylabel("Accuracy")
    ax[0].legend(['Train', 'Validation'])

    ax[1].set_ylim([0.5,1.5])

    ax[1].plot(history.history["loss"],linewidth=3,color="#FFB81C",marker='o')
    ax[1].plot(history.history['val_' + "loss"],linewidth=3,
     marker='o', markeredgecolor='black',color="#2D68C4")
    ax[1].set_title('Model {}'.format("Loss"))
    ax[1].set_xlabel('Epochs')
    ax[1].set_ylabel("loss")
    ax[1].legend(['Train', 'Validation'])


plot_curve(history)
```





In [29]:

```
model.evaluate(X_test,y_test)
```

```
42/42 [==============================] - 55s 1s/step - loss: 0.8559 - accuracy: 0.67
83 - precision: 0.7302 - recall: 0.6046 - auc: 0.8931 - f1_score: 0.6603
```

Out[29]:

```
[0.8558794856071472,
 0.6783320903778076,
 0.730215847492218,
 0.6046165227890015,
 0.8931288719177246,
 0.6602844595909119]
```

In [30]:

```
y_pred = model.predict(X_test)
```

```
42/42 [==============================] - 83s 2s/step
```

In [31]:

```
y_pred_1 = np.squeeze(y_pred)
```

In [32]:

```
y_pred_2 = np.argmax(y_pred_1,axis=1)
```

In [33]:

```
# find the maximum value by row
y_new_test = y_test.argmax(axis=1)
```

In [34]:

```
matrix = confusion_matrix(y_new_test, y_pred_2, normalize="true")
```

In [35]:

```
matrix
```

Out[35]:

```
array([[0.7921875 , 0.1890625 , 0.0171875 , 0.0015625 ],
       [0.28571429, 0.63616071, 0.07142857, 0.00669643],
       [0.20111732, 0.42458101, 0.36312849, 0.01117318],
       [0.05263158, 0.17105263, 0.06578947, 0.71052632]])
```

In [36]:

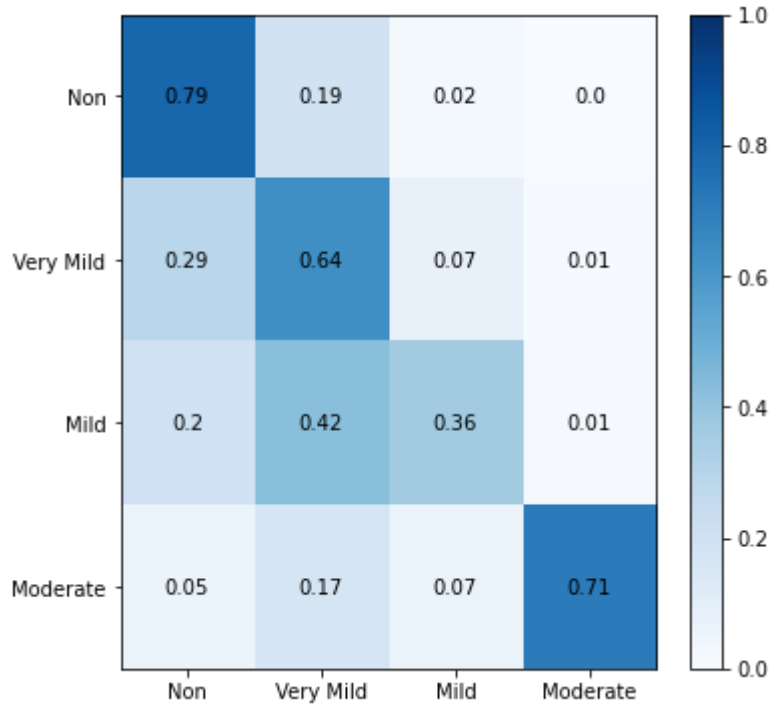
```
from matplotlib import cm
from matplotlib import axes
```

In [37]:

```
cmap = cm.Blues
fig, ax = plt.subplots(figsize=(6,6))
ax.set_xticks([0, 1, 2, 3], labels = ["Non", "Very Mild", "Mild", "Moderate"])
ax.set_yticks([0, 1, 2, 3], labels = ["Non", "Very Mild", "Mild", "Moderate"])
h = ax.imshow(matrix, cmap = cmap, aspect="auto", vmin = 0, vmax = 1)
for x in range(4):
    for y in range(4):
        value = float(format('%.2f' % matrix[y,x]))
        plt.text(x, y, value, verticalalignment = 'center', horizontalalignment = 'center')
fig.colorbar(h)
```

Out[37]:

<matplotlib.colorbar.Colorbar at 0x22a28c60b50>



Conclusion section

In [ ]:

Untitled

June 8, 2022

[8]:
```
import os
from os.path import join
from glob import glob
import numpy as np
import pandas as pd
import tensorflow as tf

import matplotlib.pyplot as plt
import PIL
import random

from tensorflow.keras.callbacks import ReduceLROnPlateau

import cv2 as cv

from sklearn.metrics import accuracy_score, balanced_accuracy_score,␣
↪roc_auc_score, \
    confusion_matrix, precision_score, recall_score, f1_score

from tensorflow.keras.models import Sequential
from keras.models import Sequential,load_model
from tensorflow.keras.layers import InputLayer, BatchNormalization, Dropout,␣
↪Flatten, Dense, Activation, MaxPool2D, Conv2D
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.callbacks import Callback,ModelCheckpoint,ReduceLROnPlateau
from keras import backend as K

import copy
import warnings
warnings.filterwarnings('ignore')

from keras.preprocessing.image import load_img, img_to_array
import matplotlib

import seaborn as sns
from sklearn.utils import shuffle
```

```
from sklearn.decomposition import PCA
from math import ceil
from tensorflow.keras.preprocessing import image
%matplotlib inline

import tensorflow.keras
import random
from tensorflow.keras.layers import *
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import *
from tensorflow.keras.losses import *
from tensorflow.keras.optimizers import *

from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,␣
↪Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,␣
↪ModelCheckpoint

from matplotlib import cm
from matplotlib import axes
```

[9]:
```
# define two functions to load 4 types of data
def import_data(directory, foldername, y, typename):
    """
    this funtion ...
    """
    data_df = pd.DataFrame({
        "X": sorted(glob(join(directory, foldername, "*"))),
        "y": y,
        "type": typename
    })
    return data_df
```

[10]:
```
# set the directory of data import

directory = "Dataset"

# build four dataframe
non_demented_df = import_data(directory, "./NonDemented", 0, "NonDemented")
very_mild_df = import_data(directory, "./VeryMildDemented", 1,␣
↪"VeryMildDemented")
mild_df = import_data(directory, "./MildDemented", 2, "MildDemented")
moderate_df = import_data(directory, "./ModerateDemented", 3,␣
↪"ModerateDemented")
```

[11]:
```
def random_split_data(data_df):
    """
    this function...
    """
    random_data = data_df.sample(frac=1)
    length = len(random_data)
    test = random_data[:int(0.2*length)]
    remaining = random_data[int(0.2*length):]
    length_2 = len(remaining)
    train = remaining[:int(0.8*length_2)]
    validation = remaining[int(0.8*length_2):]
    return test, train, validation
```

[12]:
```
# we only use train here and split train dataset to 0.8/0.2 ratio of train/
 ↪test,
# and in new train, we split it to 0.8/0.2 ration of train/validation


# split the four dataframe seperately
non_test, non_train, non_val = random_split_data(non_demented_df)
very_mild_test, very_mild_train, very_mild_val = random_split_data(very_mild_df)
mild_test, mild_train, mild_val = random_split_data(mild_df)
moderate_test, moderate_train, moderate_val = random_split_data(moderate_df)
```

[13]:
```
Autotune = tf.data.experimental.AUTOTUNE
```

[14]:
```
Image_size = [128, 128]
Batch_size = 64
Epochs = 100
```

[15]:
```
# concat the 4 data frame
train_total = pd.concat([non_train, very_mild_train, mild_train,␣
 ↪moderate_train])
validation_total = pd.concat([non_val, very_mild_val, mild_val, moderate_val])
test_total = pd.concat([non_test, very_mild_test, mild_test, moderate_test])

# randomize them
train_total = train_total.sample(frac=1)
validation_total = validation_total.sample(frac=1)
test_total = test_total.sample(frac=1)
```

[16]:
```
# load image using the directory df we create before

def load_image(file):
    """
    this function is...
    """
```

```
    image = cv.imread(file)
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    # resize it to a same size
    image = cv.resize(image, (Image_size[0], Image_size[1]))
    return image

def encoding_array(y_number):
    """
    this function is...
    """
    class_array = [0,0,0,0]
    class_array[y_number] = 1
    return class_array

def data_preparation(data_set):
    """
    this function is...
    """
    data_set["X"] = data_set["X"].apply(load_image)
    data_set["y"] = data_set["y"].apply(encoding_array)
    X = np.stack(data_set["X"])
    y = np.stack(data_set["y"])
    return X, y
```

[17]:
```
# create X, y for train, validation, test
X_train, y_train = data_preparation(train_total)
X_validation, y_validation = data_preparation(validation_total)
X_test, y_test = data_preparation(test_total)
```

[18]:
```
# reload directory
train_total = pd.concat([non_train, very_mild_train, mild_train,␣
 ↪moderate_train])
validation_total = pd.concat([non_val, very_mild_val, mild_val, moderate_val])
test_total = pd.concat([non_test, very_mild_test, mild_test, moderate_test])

# randomize them
train_total = train_total.sample(frac=1)
validation_total = validation_total.sample(frac=1)
test_total = test_total.sample(frac=1)
```

[19]:
```
def load_image(file):
    """
    this function is...
    """
    image = cv.imread(file)
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    # since the original image size is (128, 128, 3), we don't resize the image
```

```
    # here
    return image

def data_preparation(data_set):
    """
    this function is...
    """
    data_set["X"] = data_set["X"].apply(load_image)
    X = np.stack(data_set["X"])
    y = np.stack(data_set["y"])
    return X, y

X_train, y_train = data_preparation(train_total)
X_validation, y_validation = data_preparation(validation_total)
X_test, y_test = data_preparation(test_total)
```

[20]:
```
vgg16 = VGG16(weights='imagenet', input_shape=(128,128,3), include_top=False)
# Set all layers to non-trainable
for layer in vgg16.layers:
    layer.trainable = False
# Set the last 6 vgg layers to trainable
vgg16.layers[-2].trainable = True
vgg16.layers[-3].trainable = True
vgg16.layers[-4].trainable = True
vgg16.layers[-5].trainable = True
vgg16.layers[-6].trainable = True
vgg16.layers[-7].trainable = True
vgg16.layers[-8].trainable = True
```

[21]:
```
modelvgg = Sequential()
modelvgg.add(Input(shape=(128,128,3)))
modelvgg.add(vgg16)
modelvgg.add(Flatten())
modelvgg.add(Dropout(0.25))
modelvgg.add(Dense(128, activation='relu'))
modelvgg.add(Dropout(0.2))
modelvgg.add(Dense(4, activation='softmax'))
```

[22]:
```
modelvgg.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten (Flatten)           (None, 8192)              0
```

```
 dropout (Dropout)           (None, 8192)              0

 dense (Dense)               (None, 128)               1048704

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 4)                 516

=================================================================
Total params: 15,763,908
Trainable params: 14,028,420
Non-trainable params: 1,735,488
_________________________________________________________________
```

[23]:
```
modelvgg.compile(optimizer=Adam(learning_rate=0.0001),
             loss='sparse_categorical_crossentropy',
             metrics=['sparse_categorical_accuracy'])
```

[24]:
```
lrd = ReduceLROnPlateau(monitor = 'val_loss',patience = 20,verbose = 1,factor =␣
 ↪0.50, min_lr = 1e-10)

mcp = ModelCheckpoint('model.h5')

es = EarlyStopping(verbose=1, patience=20)

historyvgg = modelvgg.fit(X_train,␣
 ↪y_train,validation_data=(X_validation,y_validation), batch_size=Batch_size,␣
 ↪epochs = Epochs, verbose = 1, callbacks=[lrd,mcp,es])
```

```
Epoch 1/100
69/69 [==============================] - 232s 3s/step - loss: 1.7413 -
sparse_categorical_accuracy: 0.4642 - val_loss: 0.9638 -
val_sparse_categorical_accuracy: 0.5269 - lr: 1.0000e-04
Epoch 2/100
69/69 [==============================] - 229s 3s/step - loss: 1.0350 -
sparse_categorical_accuracy: 0.5080 - val_loss: 0.9696 -
val_sparse_categorical_accuracy: 0.5524 - lr: 1.0000e-04
Epoch 3/100
69/69 [==============================] - 227s 3s/step - loss: 0.9683 -
sparse_categorical_accuracy: 0.5456 - val_loss: 0.9641 -
val_sparse_categorical_accuracy: 0.5360 - lr: 1.0000e-04
Epoch 4/100
69/69 [==============================] - 226s 3s/step - loss: 0.9435 -
sparse_categorical_accuracy: 0.5678 - val_loss: 0.8761 -
val_sparse_categorical_accuracy: 0.5770 - lr: 1.0000e-04
Epoch 5/100
69/69 [==============================] - 225s 3s/step - loss: 0.8782 -
sparse_categorical_accuracy: 0.5952 - val_loss: 0.9489 -
```

```
val_sparse_categorical_accuracy: 0.6007 - lr: 1.0000e-04
Epoch 6/100
69/69 [==============================] - 225s 3s/step - loss: 0.8769 -
sparse_categorical_accuracy: 0.5920 - val_loss: 0.8800 -
val_sparse_categorical_accuracy: 0.5825 - lr: 1.0000e-04
Epoch 7/100
69/69 [==============================] - 225s 3s/step - loss: 0.7932 -
sparse_categorical_accuracy: 0.6467 - val_loss: 0.8681 -
val_sparse_categorical_accuracy: 0.5579 - lr: 1.0000e-04
Epoch 8/100
69/69 [==============================] - 223s 3s/step - loss: 0.7557 -
sparse_categorical_accuracy: 0.6627 - val_loss: 0.7281 -
val_sparse_categorical_accuracy: 0.6627 - lr: 1.0000e-04
Epoch 9/100
69/69 [==============================] - 223s 3s/step - loss: 0.6500 -
sparse_categorical_accuracy: 0.7122 - val_loss: 0.6021 -
val_sparse_categorical_accuracy: 0.7375 - lr: 1.0000e-04
Epoch 10/100
69/69 [==============================] - 221s 3s/step - loss: 0.5502 -
sparse_categorical_accuracy: 0.7670 - val_loss: 0.5797 -
val_sparse_categorical_accuracy: 0.7429 - lr: 1.0000e-04
Epoch 11/100
69/69 [==============================] - 223s 3s/step - loss: 0.4992 -
sparse_categorical_accuracy: 0.7983 - val_loss: 0.4113 -
val_sparse_categorical_accuracy: 0.8368 - lr: 1.0000e-04
Epoch 12/100
69/69 [==============================] - 222s 3s/step - loss: 0.3205 -
sparse_categorical_accuracy: 0.8727 - val_loss: 0.4006 -
val_sparse_categorical_accuracy: 0.8377 - lr: 1.0000e-04
Epoch 13/100
69/69 [==============================] - 222s 3s/step - loss: 0.2755 -
sparse_categorical_accuracy: 0.8984 - val_loss: 0.3181 -
val_sparse_categorical_accuracy: 0.8715 - lr: 1.0000e-04
Epoch 14/100
69/69 [==============================] - 224s 3s/step - loss: 0.1701 -
sparse_categorical_accuracy: 0.9414 - val_loss: 0.2213 -
val_sparse_categorical_accuracy: 0.9225 - lr: 1.0000e-04
Epoch 15/100
69/69 [==============================] - 223s 3s/step - loss: 0.1464 -
sparse_categorical_accuracy: 0.9473 - val_loss: 0.3049 -
val_sparse_categorical_accuracy: 0.8888 - lr: 1.0000e-04
Epoch 16/100
69/69 [==============================] - 223s 3s/step - loss: 0.1093 -
sparse_categorical_accuracy: 0.9639 - val_loss: 0.3954 -
val_sparse_categorical_accuracy: 0.8906 - lr: 1.0000e-04
Epoch 17/100
69/69 [==============================] - 223s 3s/step - loss: 0.0862 -
sparse_categorical_accuracy: 0.9701 - val_loss: 0.3561 -
```

```
val_sparse_categorical_accuracy: 0.8997 - lr: 1.0000e-04
Epoch 18/100
69/69 [==============================] - 224s 3s/step - loss: 0.0368 -
sparse_categorical_accuracy: 0.9877 - val_loss: 0.2034 -
val_sparse_categorical_accuracy: 0.9517 - lr: 1.0000e-04
Epoch 19/100
69/69 [==============================] - 222s 3s/step - loss: 0.1596 -
sparse_categorical_accuracy: 0.9450 - val_loss: 0.3161 -
val_sparse_categorical_accuracy: 0.9116 - lr: 1.0000e-04
Epoch 20/100
69/69 [==============================] - 222s 3s/step - loss: 0.0723 -
sparse_categorical_accuracy: 0.9776 - val_loss: 0.1135 -
val_sparse_categorical_accuracy: 0.9626 - lr: 1.0000e-04
Epoch 21/100
69/69 [==============================] - 219s 3s/step - loss: 0.0110 -
sparse_categorical_accuracy: 0.9968 - val_loss: 0.2220 -
val_sparse_categorical_accuracy: 0.9480 - lr: 1.0000e-04
Epoch 22/100
69/69 [==============================] - 221s 3s/step - loss: 0.0170 -
sparse_categorical_accuracy: 0.9943 - val_loss: 0.1135 -
val_sparse_categorical_accuracy: 0.9717 - lr: 1.0000e-04
Epoch 23/100
69/69 [==============================] - 222s 3s/step - loss: 0.0544 -
sparse_categorical_accuracy: 0.9822 - val_loss: 0.1943 -
val_sparse_categorical_accuracy: 0.9371 - lr: 1.0000e-04
Epoch 24/100
69/69 [==============================] - 221s 3s/step - loss: 0.0281 -
sparse_categorical_accuracy: 0.9902 - val_loss: 0.2437 -
val_sparse_categorical_accuracy: 0.9280 - lr: 1.0000e-04
Epoch 25/100
69/69 [==============================] - 222s 3s/step - loss: 0.0151 -
sparse_categorical_accuracy: 0.9952 - val_loss: 0.1546 -
val_sparse_categorical_accuracy: 0.9672 - lr: 1.0000e-04
Epoch 26/100
69/69 [==============================] - 219s 3s/step - loss: 0.0241 -
sparse_categorical_accuracy: 0.9920 - val_loss: 0.1457 -
val_sparse_categorical_accuracy: 0.9635 - lr: 1.0000e-04
Epoch 27/100
69/69 [==============================] - 221s 3s/step - loss: 0.0032 -
sparse_categorical_accuracy: 0.9986 - val_loss: 0.1013 -
val_sparse_categorical_accuracy: 0.9736 - lr: 1.0000e-04
Epoch 28/100
69/69 [==============================] - 221s 3s/step - loss: 0.0289 -
sparse_categorical_accuracy: 0.9913 - val_loss: 0.1297 -
val_sparse_categorical_accuracy: 0.9644 - lr: 1.0000e-04
Epoch 29/100
69/69 [==============================] - 223s 3s/step - loss: 0.0058 -
sparse_categorical_accuracy: 0.9984 - val_loss: 0.1182 -
```

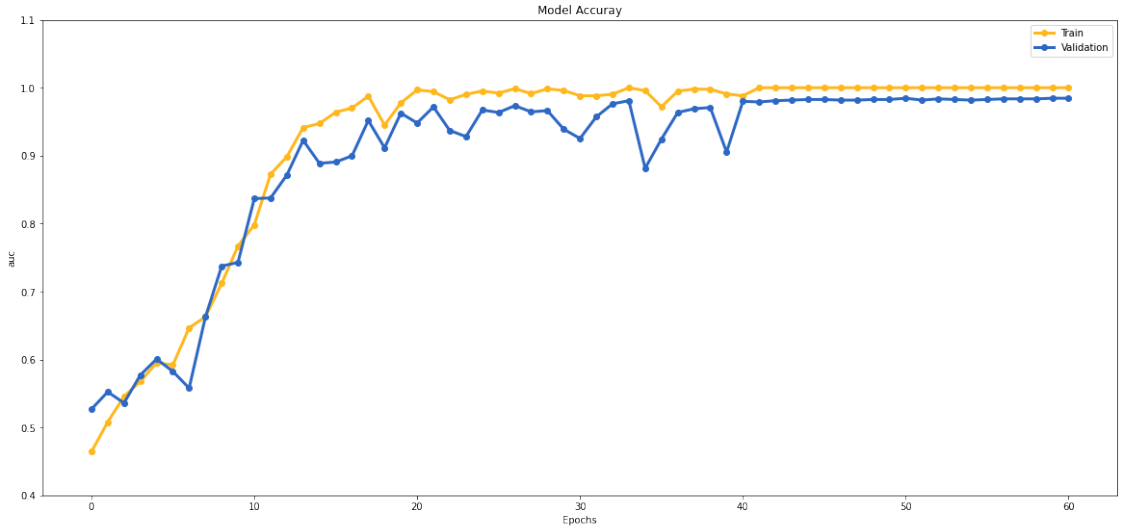
```
val_sparse_categorical_accuracy: 0.9663 - lr: 1.0000e-04
Epoch 30/100
69/69 [==============================] - 223s 3s/step - loss: 0.0166 -
sparse_categorical_accuracy: 0.9959 - val_loss: 0.2276 -
val_sparse_categorical_accuracy: 0.9389 - lr: 1.0000e-04
Epoch 31/100
69/69 [==============================] - 222s 3s/step - loss: 0.0323 -
sparse_categorical_accuracy: 0.9881 - val_loss: 0.2726 -
val_sparse_categorical_accuracy: 0.9253 - lr: 1.0000e-04
Epoch 32/100
69/69 [==============================] - 221s 3s/step - loss: 0.0400 -
sparse_categorical_accuracy: 0.9881 - val_loss: 0.1293 -
val_sparse_categorical_accuracy: 0.9572 - lr: 1.0000e-04
Epoch 33/100
69/69 [==============================] - 222s 3s/step - loss: 0.0314 -
sparse_categorical_accuracy: 0.9904 - val_loss: 0.0858 -
val_sparse_categorical_accuracy: 0.9763 - lr: 1.0000e-04
Epoch 34/100
69/69 [==============================] - 221s 3s/step - loss: 8.5619e-04 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.0827 -
val_sparse_categorical_accuracy: 0.9809 - lr: 1.0000e-04
Epoch 35/100
69/69 [==============================] - 221s 3s/step - loss: 0.0215 -
sparse_categorical_accuracy: 0.9957 - val_loss: 0.4194 -
val_sparse_categorical_accuracy: 0.8815 - lr: 1.0000e-04
Epoch 36/100
69/69 [==============================] - 222s 3s/step - loss: 0.0897 -
sparse_categorical_accuracy: 0.9719 - val_loss: 0.2375 -
val_sparse_categorical_accuracy: 0.9243 - lr: 1.0000e-04
Epoch 37/100
69/69 [==============================] - 221s 3s/step - loss: 0.0133 -
sparse_categorical_accuracy: 0.9950 - val_loss: 0.1344 -
val_sparse_categorical_accuracy: 0.9635 - lr: 1.0000e-04
Epoch 38/100
69/69 [==============================] - 221s 3s/step - loss: 0.0059 -
sparse_categorical_accuracy: 0.9979 - val_loss: 0.1060 -
val_sparse_categorical_accuracy: 0.9690 - lr: 1.0000e-04
Epoch 39/100
69/69 [==============================] - 222s 3s/step - loss: 0.0070 -
sparse_categorical_accuracy: 0.9977 - val_loss: 0.0908 -
val_sparse_categorical_accuracy: 0.9708 - lr: 1.0000e-04
Epoch 40/100
69/69 [==============================] - 221s 3s/step - loss: 0.0331 -
sparse_categorical_accuracy: 0.9909 - val_loss: 0.2851 -
val_sparse_categorical_accuracy: 0.9052 - lr: 1.0000e-04
Epoch 41/100
69/69 [==============================] - 222s 3s/step - loss: 0.0365 -
sparse_categorical_accuracy: 0.9881 - val_loss: 0.0741 -
```

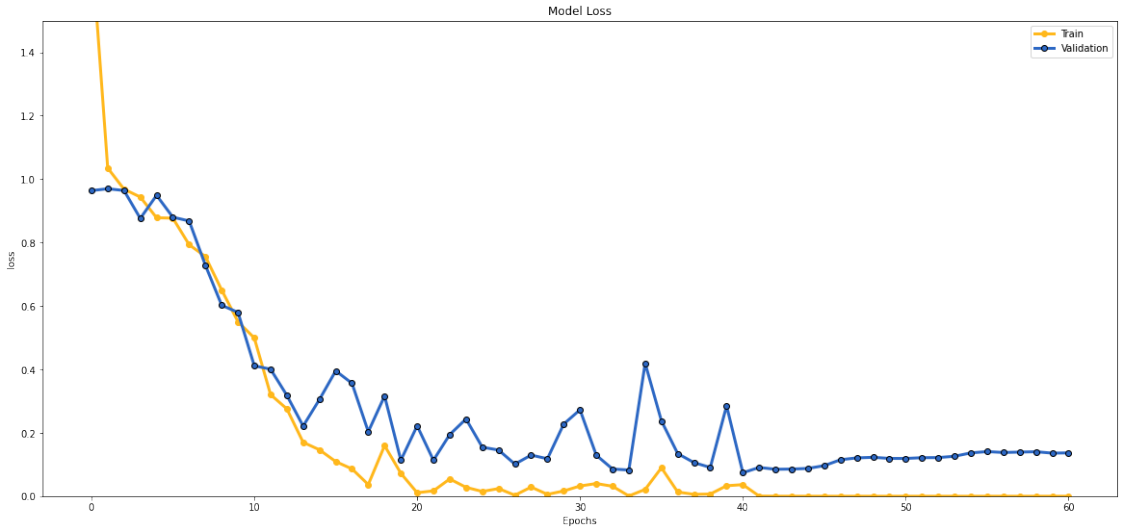
```
val_sparse_categorical_accuracy: 0.9799 - lr: 1.0000e-04
Epoch 42/100
69/69 [==============================] - 220s 3s/step - loss: 5.6574e-04 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.0908 -
val_sparse_categorical_accuracy: 0.9790 - lr: 1.0000e-04
Epoch 43/100
69/69 [==============================] - 220s 3s/step - loss: 1.1320e-04 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.0850 -
val_sparse_categorical_accuracy: 0.9809 - lr: 1.0000e-04
Epoch 44/100
69/69 [==============================] - 222s 3s/step - loss: 8.3552e-05 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.0857 -
val_sparse_categorical_accuracy: 0.9818 - lr: 1.0000e-04
Epoch 45/100
69/69 [==============================] - 221s 3s/step - loss: 4.1718e-05 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.0878 -
val_sparse_categorical_accuracy: 0.9827 - lr: 1.0000e-04
Epoch 46/100
69/69 [==============================] - 222s 3s/step - loss: 2.8910e-05 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.0965 -
val_sparse_categorical_accuracy: 0.9827 - lr: 1.0000e-04
Epoch 47/100
69/69 [==============================] - 224s 3s/step - loss: 9.1559e-06 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1151 -
val_sparse_categorical_accuracy: 0.9818 - lr: 1.0000e-04
Epoch 48/100
69/69 [==============================] - 232s 3s/step - loss: 3.5207e-05 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1209 -
val_sparse_categorical_accuracy: 0.9818 - lr: 1.0000e-04
Epoch 49/100
69/69 [==============================] - 230s 3s/step - loss: 4.9934e-06 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1227 -
val_sparse_categorical_accuracy: 0.9827 - lr: 1.0000e-04
Epoch 50/100
69/69 [==============================] - 225s 3s/step - loss: 3.4975e-06 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1192 -
val_sparse_categorical_accuracy: 0.9827 - lr: 1.0000e-04
Epoch 51/100
69/69 [==============================] - 225s 3s/step - loss: 2.9369e-05 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1191 -
val_sparse_categorical_accuracy: 0.9845 - lr: 1.0000e-04
Epoch 52/100
69/69 [==============================] - 225s 3s/step - loss: 1.0596e-05 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1215 -
val_sparse_categorical_accuracy: 0.9818 - lr: 1.0000e-04
Epoch 53/100
69/69 [==============================] - 228s 3s/step - loss: 4.0213e-06 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1219 -
```

```
val_sparse_categorical_accuracy: 0.9836 - lr: 1.0000e-04
Epoch 54/100
69/69 [==============================] - 226s 3s/step - loss: 3.4438e-06 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1261 -
val_sparse_categorical_accuracy: 0.9827 - lr: 1.0000e-04
Epoch 55/100
69/69 [==============================] - 224s 3s/step - loss: 1.7717e-06 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1367 -
val_sparse_categorical_accuracy: 0.9818 - lr: 1.0000e-04
Epoch 56/100
69/69 [==============================] - 225s 3s/step - loss: 2.6503e-05 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1409 -
val_sparse_categorical_accuracy: 0.9827 - lr: 1.0000e-04
Epoch 57/100
69/69 [==============================] - 225s 3s/step - loss: 4.6452e-06 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1379 -
val_sparse_categorical_accuracy: 0.9836 - lr: 1.0000e-04
Epoch 58/100
69/69 [==============================] - 221s 3s/step - loss: 2.9689e-06 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1392 -
val_sparse_categorical_accuracy: 0.9836 - lr: 1.0000e-04
Epoch 59/100
69/69 [==============================] - 222s 3s/step - loss: 2.2128e-06 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1404 -
val_sparse_categorical_accuracy: 0.9836 - lr: 1.0000e-04
Epoch 60/100
69/69 [==============================] - 222s 3s/step - loss: 4.4068e-06 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1359 -
val_sparse_categorical_accuracy: 0.9845 - lr: 1.0000e-04
Epoch 61/100
69/69 [==============================] - ETA: 0s - loss: 8.3627e-07 -
sparse_categorical_accuracy: 1.0000
Epoch 61: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
69/69 [==============================] - 222s 3s/step - loss: 8.3627e-07 -
sparse_categorical_accuracy: 1.0000 - val_loss: 0.1368 -
val_sparse_categorical_accuracy: 0.9845 - lr: 1.0000e-04
Epoch 61: early stopping
```

[25]:
```
from matplotlib import pyplot as plt

def plot_curve(history):


    fig, ax = plt.subplots(2, 1, figsize=(20, 20))
    ax = ax.ravel()

    ax[0].set_ylim([0.4,1.1])
```

```
    ax[0].plot(history.
↪history["sparse_categorical_accuracy"],linewidth=3,color="#FFB81C",marker='o')
    ax[0].plot(history.history['val_' +␣
↪"sparse_categorical_accuracy"],linewidth=3,marker='o',color="#2D68C4")
    ax[0].set_title('Model {}'.format("Accuray"))
    ax[0].set_xlabel('Epochs')
    ax[0].set_ylabel("auc")
    ax[0].legend(['Train', 'Validation'])

    ax[1].set_ylim([0,1.5])

    ax[1].plot(history.history["loss"],linewidth=3,color="#FFB81C",marker='o')
    ax[1].plot(history.history['val_' + "loss"],linewidth=3,
     marker='o', markeredgecolor='black',color="#2D68C4")
    ax[1].set_title('Model {}'.format("Loss"))
    ax[1].set_xlabel('Epochs')
    ax[1].set_ylabel("loss")
    ax[1].legend(['Train', 'Validation'])


plot_curve(historyvgg)
```





[26]:
```
modelvgg.evaluate(X_test,y_test)
```

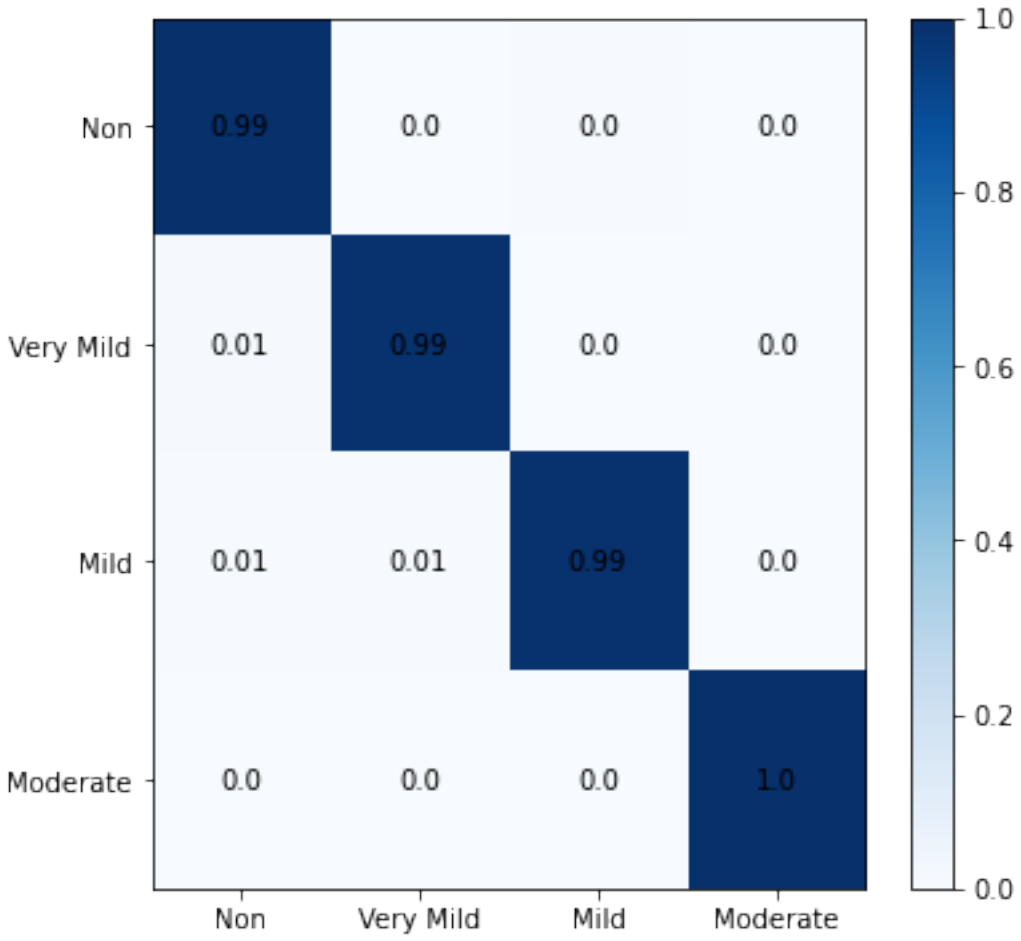
```
43/43 [==============================] - 33s 770ms/step - loss: 0.0402 -
sparse_categorical_accuracy: 0.9912
```

[26]:
```
[0.040185391902923584, 0.9912344813346863]
```

[27]:
```
y_pred = modelvgg.predict(X_test)
y_pred_1 = np.squeeze(y_pred)
y_pred_2 = np.argmax(y_pred_1,axis=1)
matrix = confusion_matrix(y_test, y_pred_2, normalize="true")
matrix
```

[27]:
```
array([[0.9921875 , 0.003125  , 0.0046875 , 0.        ],
       [0.01116071, 0.98883929, 0.        , 0.        ],
       [0.00558659, 0.00558659, 0.98882682, 0.        ],
       [0.        , 0.        , 0.        , 1.        ]])
```

[30]:
```
def plot_confusion_matrix(matrix):
    cmap = cm.Blues
    fig, ax = plt.subplots(figsize=(6,6))
    ax.set_xticks([0, 1, 2, 3])
    ax.set_xticklabels(["Non", "Very Mild", "Mild", "Moderate"])
    ax.set_yticks([0, 1, 2, 3])
    ax.set_yticklabels(["Non", "Very Mild", "Mild", "Moderate"])
    h = ax.imshow(matrix, cmap = cmap, aspect="auto", vmin = 0, vmax = 1)
    for x in range(4):
        for y in range(4):
            value = float(format('%.2f' % matrix[y,x]))
            plt.text(x, y, value, verticalalignment = 'center',␣
 ↪horizontalalignment = 'center')
    fig.colorbar(h)

plot_confusion_matrix(matrix)
```


Non
Very Mild
Mild
Moderate
0.99
0.0
0.0
0.0
0.01
0.99
0.0
0.0
0.01
0.01
0.99
0.0
0.0
0.0
0.0
1.0
Non
Very Mild
Mild
Moderate
1.0
0.8
0.6
0.4
0.2
0.0

[ ]:

Environment Set up

Set up combination

In [2]:

```
import os
from os.path import join
from glob import glob
import numpy as np
import pandas as pd
import tensorflow as tf

import matplotlib.pyplot as plt
import PIL
import random

from tensorflow.keras.callbacks import ReduceLROnPlateau

import cv2 as cv

from sklearn.metrics import accuracy_score, balanced_accuracy_score, roc_auc_score, \
    confusion_matrix, precision_score, recall_score, f1_score

from tensorflow.keras.models import Sequential
from keras.models import Sequential,load_model
from tensorflow.keras.layers import InputLayer, BatchNormalization, Dropout, Flatten, Dense, Activa
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.callbacks import Callback,ModelCheckpoint,ReduceLROnPlateau
from keras import backend as K

import copy
import warnings
warnings.filterwarnings('ignore')

#from keras.preprocessing.image import load_img, img_to_array
import matplotlib

import seaborn as sns
from sklearn.utils import shuffle

from sklearn.decomposition import PCA
from math import ceil
from tensorflow.keras.preprocessing import image
%matplotlib inline

import tensorflow.keras
import random
from tensorflow.keras.layers import *
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import *
from tensorflow.keras.losses import *
from tensorflow.keras.optimizers import *

from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

from matplotlib import cm
from matplotlib import axes
```

Load Datasets

In [3]:

```
# define two functions to load 4 types of data
def import_data(directory, foldername, y, typename):
    """
    this funtion ...
    """
    data_df = pd.DataFrame({
        "X": sorted(glob(join(directory, foldername, "*"))),
        "y": y,
        "type": typename
    })
    return data_df
```

In [4]:

```
# set the directory of data import

directory = "C:\\Users\\lenovo\\Desktop\\python\\Alzheimer's"

# build four dataframe
non_demented_df = import_data(directory, ".\\NonDemented", 0, "NonDemented")
very_mild_df = import_data(directory, ".\\VeryMildDemented", 1, "VeryMildDemented")
mild_df = import_data(directory, ".\\MildDemented", 2, "MildDemented")
moderate_df = import_data(directory, ".\\ModerateDemented", 3, "ModerateDemented")
```

Data Preparation

4 directory dataframe

1. split 4 types of data into train, test, validation.
2. Set parameters
3. concat the 4 dataframe

Split 4 types of data

In [5]:

```
def random_split_data(data_df):
    """
    this function...
    """
    random_data = data_df.sample(frac=1)
    length = len(random_data)
    test = random_data[:int(0.2*length)]
    remaining = random_data[int(0.2*length):]
    length_2 = len(remaining)
    train = remaining[:int(0.8*length_2)]
    validation = remaining[int(0.8*length_2):]
    return test, train, validation
```

In [6]:

```
# we only use train here and split train dataset to 0.8/0.2 ratio of train/test,
# and in new train, we split it to 0.8/0.2 ration of train/validation


# split the four dataframe seperately
non_test, non_train, non_val = random_split_data(non_demented_df)
very_mild_test, very_mild_train, very_mild_val = random_split_data(very_mild_df)
mild_test, mild_train, mild_val = random_split_data(mild_df)
moderate_test, moderate_train, moderate_val = random_split_data(moderate_df)
```

Set parameters

In [7]:

```
Autotune = tf.data.experimental.AUTOTUNE
```

In [10]:

```
Image_size = [224, 224]
Batch_size = 64
Epochs = 5 # needed to change to 50 afterwards
```

Concat the 4 dataframe and randomize it

In [11]:

```
# concat the 4 data frame
train_total = pd.concat([non_train, very_mild_train, mild_train, moderate_train])
validation_total = pd.concat([non_val, very_mild_val, mild_val, moderate_val])
test_total = pd.concat([non_test, very_mild_test, mild_test, moderate_test])

# randomize them
train_total = train_total.sample(frac=1)
validation_total = validation_total.sample(frac=1)
test_total = test_total.sample(frac=1)
```

Build three functions to load image, create class array and extract X, y for each train, validation and test.

In [12]:

```
# load image using the directory df we create before

def load_image(file):
    """
    this function is...
    """
    image = cv.imread(file)
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    # resize it to a same size
    image = cv.resize(image, (Image_size[0], Image_size[1]))
    return image

def encoding_array(y_number):
    """
    this function is...
    """
    class_array = [0,0,0,0]
    class_array[y_number] = 1
    return class_array

def data_preparation(data_set):
    """
    this function is...
    """
    data_set["X"] = data_set["X"].apply(load_image)
    data_set["y"] = data_set["y"].apply(encoding_array)
    X = np.stack(data_set["X"])
    y = np.stack(data_set["y"])
    return X, y
```

In [13]:

```
# create X, y for train, validation, test
X_train, y_train = data_preparation(train_total)
X_validation, y_validation = data_preparation(validation_total)
X_test, y_test = data_preparation(test_total)
```

Model 1

In [15]:

```
from matplotlib import pyplot as plt

def plot_curve(history):


    fig, ax = plt.subplots(2, 1, figsize=(20, 20))
    ax = ax.ravel()

    ax[0].set_ylim([0.6,1])

    ax[0].plot(history.history["auc"],linewidth=3,color="#FFB81C",marker='o')
    ax[0].plot(history.history['val_' + "auc"],linewidth=3,marker='o',color="#2D68C4")
    ax[0].set_title('Model {}'.format("Accuray"))
    ax[0].set_xlabel('Epochs')
    ax[0].set_ylabel("auc")
    ax[0].legend(['Train', 'Validation'])

    ax[1].set_ylim([0.5,1.5])

    ax[1].plot(history.history["loss"],linewidth=3,color="#FFB81C",marker='o')
    ax[1].plot(history.history['val_' + "loss"],linewidth=3,
     marker='o', markeredgecolor='black',color="#2D68C4")
    ax[1].set_title('Model {}'.format("Loss"))
    ax[1].set_xlabel('Epochs')
    ax[1].set_ylabel("loss")
    ax[1].legend(['Train', 'Validation'])
```

In [16]:

```
def plot_confusion_matrix(matrix):
    cmap = cm.Blues
    fig, ax = plt.subplots(figsize=(6,6))
    ax.set_xticks([0, 1, 2, 3])
    ax.set_xticklabels(["Non", "Very Mild", "Mild", "Moderate"])
    ax.set_yticks([0, 1, 2, 3])
    ax.set_yticklabels(["Non", "Very Mild", "Mild", "Moderate"])
    h = ax.imshow(matrix, cmap = cmap, aspect="auto", vmin = 0, vmax = 1)
    for x in range(4):
        for y in range(4):
            value = float(format('%.2f' % matrix[y,x]))
            plt.text(x, y, value, verticalalignment = 'center', horizontalalignment = 'center')
    fig.colorbar(h)
```

Model 2 section

1. Model built
2. train
3. fit
4. accuracy line plot + loss line plot
5. predit
6. predit confusion matrix plot

In [17]:

```
model2 = Sequential()

# Convolutional layer 1
model2.add(Conv2D(32,(3,3), input_shape=(224,224,3), activation='relu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2,2)))

# Convolutional layer 2
model2.add(Conv2D(32,(3,3), activation='relu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2,2)))

model2.add(Flatten())

# Neural network

model2.add(Dense(units= 252, activation='relu'))
model2.add(Dropout(0.2))
model2.add(Dense(units=252, activation='relu'))
model2.add(Dropout(0.2))
model2.add(Dense(units=4, activation='softmax'))

optimizer = tf.keras.optimizers.Adam(learning_rate=0.001, decay=0.0001, clipvalue=0.5)
model2.compile(optimizer=optimizer, loss='categorical_crossentropy',
                   metrics= ['categorical_accuracy'])
```

In [18]:

```
model2.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 batch_normalization (BatchN  (None, 222, 222, 32)     128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 32)      9248

 batch_normalization_1 (Batc  (None, 109, 109, 32)     128
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 32)       0
 2D)

 flatten (Flatten)           (None, 93312)             0

 dense (Dense)               (None, 252)               23514876

 dropout (Dropout)           (None, 252)               0

 dense_1 (Dense)             (None, 252)               63756

 dropout_1 (Dropout)         (None, 252)               0

 dense_2 (Dense)             (None, 4)                 1012

=================================================================
Total params: 23,590,044
Trainable params: 23,589,916
Non-trainable params: 128
_________________________________________________________________
```

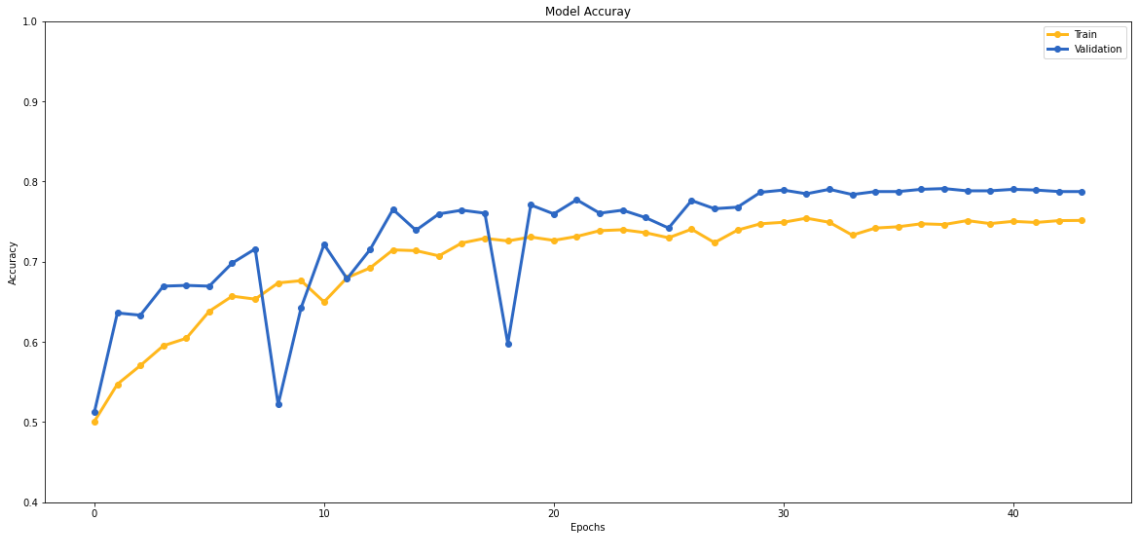
In [27]:

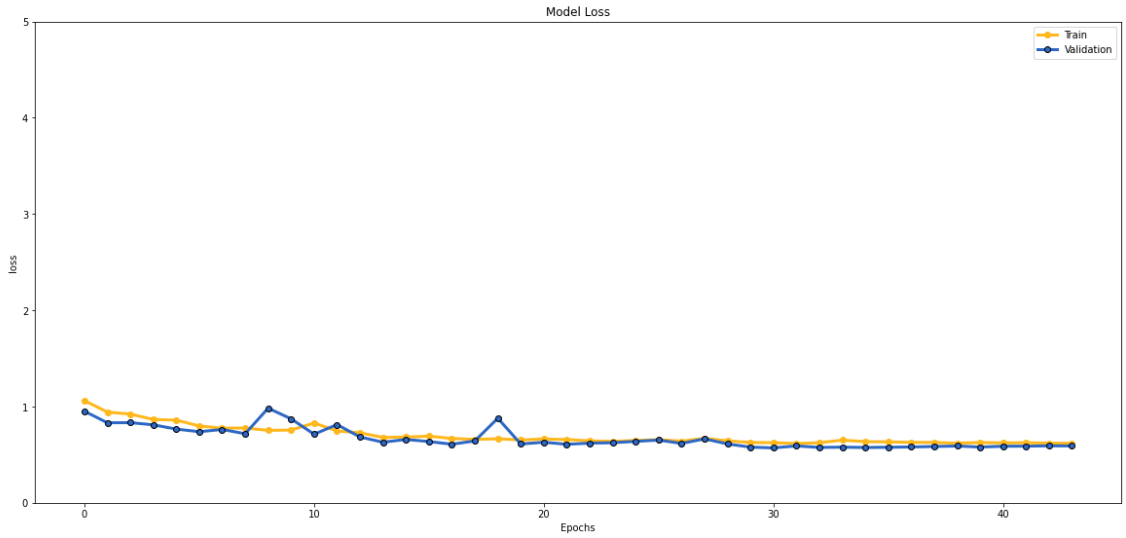
```
## Creating callbacks for the model.
# If the model dosen't continue to improve (loss), the trainning will stop.

# Stop training if loss doesn't keep decreasing.
model2_es = EarlyStopping(monitor = 'loss', min_delta = 1e-11, patience = 12, verbose = 1)
model2_rlr = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.2, patience = 6, verbose = 1)

# Automatically saves the best weights of the model, based on best val_accuracy
model2_mcp = ModelCheckpoint(filepath = 'model1_weights.h5', monitor = 'val_categorical_accuracy',
                      save_best_only = True, verbose = 1)

# Fiting the model.
history = model2.fit(X_train, y_train, validation_data=(X_validation, y_validation), batch_size=Batc
```

```
Epoch 37/50
68/68 [==============================] - ETA: 0s - loss: 0.6279 - categorical_acc
uracy: 0.7472
Epoch 37: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.

Epoch 37: val_categorical_accuracy did not improve from 0.79016
68/68 [==============================] - 280s 4s/step - loss: 0.6279 - categorica
l_accuracy: 0.7472 - val_loss: 0.5801 - val_categorical_accuracy: 0.7902 - lr: 2.
0000e-04
Epoch 38/50
68/68 [==============================] - ETA: 0s - loss: 0.6270 - categorical_acc
uracy: 0.7463
Epoch 38: val_categorical_accuracy improved from 0.79016 to 0.79109, saving model
to model1_weights.h5
68/68 [==============================] - 279s 4s/step - loss: 0.6270 - categorica
l_accuracy: 0.7463 - val_loss: 0.5850 - val_categorical_accuracy: 0.7911 - lr: 4.
0000e-05
Epoch 39/50
68/68 [==============================] - ETA: 0s - loss: 0.6190 - categorical_acc
uracy: 0.7512
```

In [33]:

```
def plot_curve(history):


    fig, ax = plt.subplots(2, 1, figsize=(20, 20))
    ax = ax.ravel()

    ax[0].set_ylim([0.4,1])

    ax[0].plot(history.history["categorical_accuracy"],linewidth=3,color="#FFB81C",marker='o')
    ax[0].plot(history.history['val_categorical_accuracy'],linewidth=3,marker='o',color="#2D68C4")
    ax[0].set_title('Model {}'.format("Accuray"))
    ax[0].set_xlabel('Epochs')
    ax[0].set_ylabel("Accuracy")
    ax[0].legend(['Train', 'Validation'])

    ax[1].set_ylim([0,5])

    ax[1].plot(history.history["loss"],linewidth=3,color="#FFB81C",marker='o')
    ax[1].plot(history.history['val_' + "loss"],linewidth=3,
     marker='o', markeredgecolor='black',color="#2D68C4")
    ax[1].set_title('Model {}'.format("Loss"))
    ax[1].set_xlabel('Epochs')
    ax[1].set_ylabel("loss")
    ax[1].legend(['Train', 'Validation'])

plot_curve(history)
```





In [29]:

```
model2.evaluate(X_test,y_test)
```

```
42/42 [==============================] - 23s 541ms/step - loss: 0.5981 - categorical
_accuracy: 0.7826
```

Out[29]:

```
[0.5981324911117554, 0.782576322555542]
```

In [30]:

```
y_pred = model2.predict(X_test)
y_pred_1 = np.squeeze(y_pred)
y_pred_2 = np.argmax(y_pred_1,axis=1)
```

```
42/42 [==============================] - 21s 507ms/step
```

In [31]:

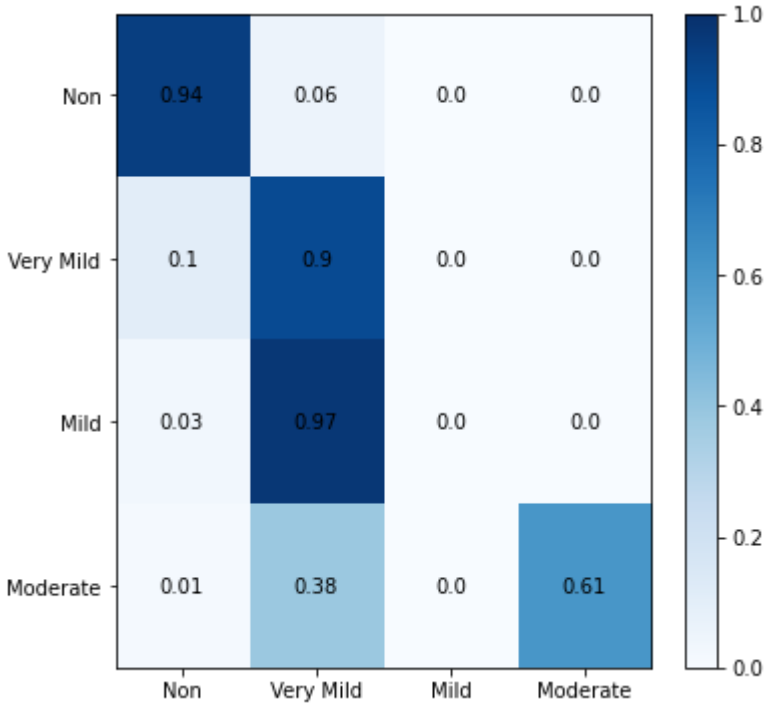
```
y_new_test = y_test.argmax(axis=1)
matrix = confusion_matrix(y_new_test, y_pred_2, normalize="true")
matrix
```

Out[31]:

```
array([[0.94375   , 0.05625   , 0.        , 0.        ],
       [0.10491071, 0.89508929, 0.        , 0.        ],
       [0.02793296, 0.97206704, 0.        , 0.        ],
       [0.01315789, 0.38157895, 0.        , 0.60526316]])
```

In [32]:

```
plot_confusion_matrix(matrix)
```



Conclusion section

In [ ]: