National College of Ireland

Configuration Manual

An Approach to Classify Ocular diseases using Machine Learning and Deep Learning

MSc Research Project

Data Analytics

Ankitha Mallikarjuna Gajaram

Student ID: 19201559

School of Computing

National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ankitha Mallikarjuna Gajaram
Student ID:	x19201559
Programme:	MSc Data Analytics **Year:** 2021-2022
Module:	Research Project
Supervisor:	Vladimir Milosavljevic
Submission Due Date:	31st January 2022
Project Title:	An Approach to Classify Ocular diseases using Machine Learning and Deep Learning
Word Count:	1009 **Page Count** 17

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Ankitha Mallikarjuna Gajaram

Date: 31st January 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

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

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

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

Classifying Ocular Disease using Machine Learning and Deep Learning Techniques

Ankitha Mallikarjuna Gajaram

x19201559

1. Introduction

This configuration manual explains the overall code, system configuration requirement, the dataset taken, and code implementation and the packages used is explained.

2. Environment Configuration

Laptop Model: I have used Dell Inspiron Laptop, i5 processor, Windows 10 and 8GB RAM configuration laptop for my research project.

The Software used for project are:

From Anaconda (version: 1.9.7) I have installed jupyter notebook. For my project, I am using Jupyter Notebook 6.1.2 to write python language.

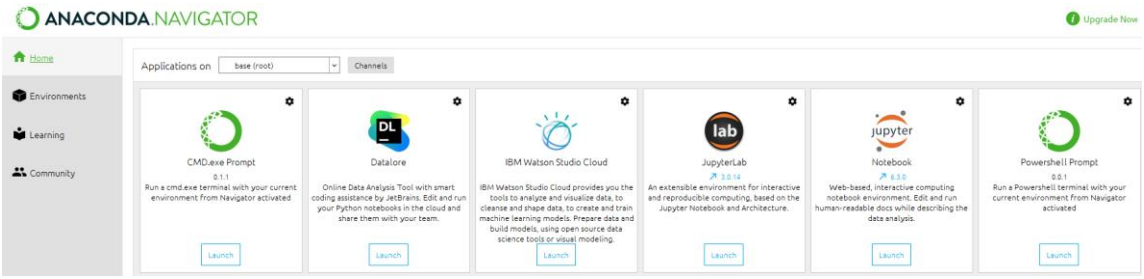

Figure 1

3. Data Storage

Kaggle Repository – The data is taken from Kaggle repository which is open source. The data is downloaded to dataset folder and from jupyter notebook the data is accessed.

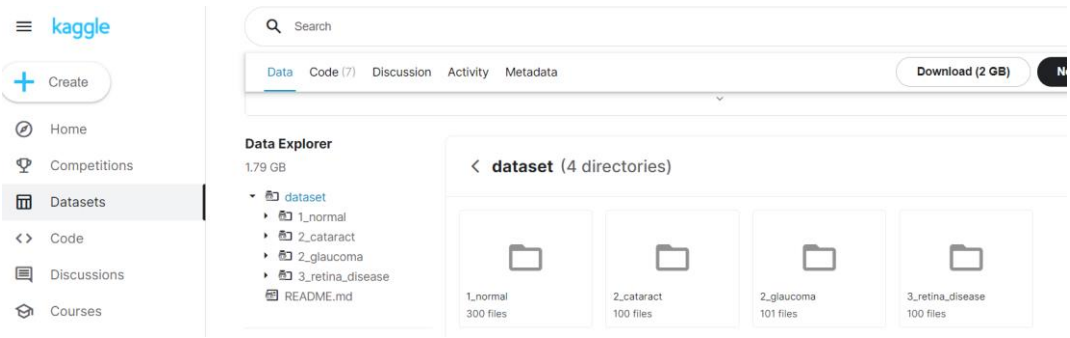

Figure 2

4. Implementation and Results

Libraries imported and installing necessary libraries using the -pip command

Main packages which are used are numpy, pandas, keras, tensorflow, cv2 to load the images.

```
import random
import os
import matplotlib.pyplot as plt
import efficientnet.tfkeras as efn
import tensorflow as tf
import glob
from tqdm import tqdm
from keras import layers
from keras.layers import Input
from keras.layers import Conv2D,MaxPool2D,GlobalAveragePooling2D
from tensorflow.keras import backend as K

from keras.models import Sequential
from keras.layers import Dense,Dropout,Activation,Flatten
from tensorflow.keras.utils import get_custom_objects

from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split

SEED=42
EPOCHS = 100
BATCH_SIZE = 32
IMG_HEIGHT = 192
IMG_WIDTH = 256
```

Figure 3

To work on all the data, we first load it onto system.

```
: import os

  folder = r'C:/Users/Public/dataset/'

  sub_folders = [name for name in os.listdir(folder) if os.path.isdir(os.path.join(folder, name))]

: sub_folders

: ['1_normal', '2_cataract', '2_glaucoma', '3_retina_disease']
```

Figure 4

Glob command is used to load all the internal directory folder and images.

```
import glob, os
files1=[]
for i in folder1:
    files = [f for f in glob.glob(i + "*/*.png", recursive=True)]
    files1.append(files)
```

Figure 5

Making list of Image path

```
def flatten_list(_2d_list):
    flat_list = []
    # Iterate through the outer list
    for element in _2d_list:
        if type(element) is list:
            # If the element is of type list, iterate through the sublist
            for item in element:
                flat_list.append(item)
        else:
            flat_list.append(element)
    return flat_list
```

Figure 6

Removing the Extra "\" using split () function to extract the folder and image.

```
folder =[(str(f.split("\\")[0])+"/"+str(f.split("\\")[1])) for f in folder_name1]
folder
```

```
['C:/Users/Public/dataset/1_normal',
 'C:/Users/Public/dataset/1_normal',
 'C:/Users/Public/dataset/1_normal',
 'C:/Users/Public/dataset/1_normal',
 'C:/Users/Public/dataset/1_normal',
```

Figure 7

Making a DataFrame of extracted folders and files

```
df=pd.DataFrame()
df["folder_name"]= folder
df["files"]=  filename
df["Labels"]= label
df3=df
df
```

	folder_name	files	Labels
0	C:/Users/Public/dataset/1_normal	NL_001.png	1_normal
1	C:/Users/Public/dataset/1_normal	NL_002.png	1_normal
2	C:/Users/Public/dataset/1_normal	NL_003.png	1_normal
3	C:/Users/Public/dataset/1_normal	NL_004.png	1_normal
4	C:/Users/Public/dataset/1_normal	NL_005.png	1_normal
...	...	...	...

Figure 8

Creating Labels for Ocular Disease based on folder name

```
df_new=df
df_new.loc[df['Labels'] == '1_normal', 'Label1'] = 0
df_new.loc[df['Labels'] == '2_cataract', 'Label1'] = 1
df_new.loc[df['Labels'] == '2_glaucoma', 'Label1'] = 2
df_new.loc[df['Labels'] == '3_retina_disease', 'Label1'] = 3
df_new.style.format({
    'Labels1': '{:,.1f}'.format

})

df_new
```

	folder_name	files	Labels	Label1
0	C:/Users/Public/dataset/1_normal	NL_001.png	1_normal	0.0
1	C:/Users/Public/dataset/1_normal	NL_002.png	1_normal	0.0
2	C:/Users/Public/dataset/1_normal	NL_003.png	1_normal	0.0
3	C:/Users/Public/dataset/1_normal	NL_004.png	1_normal	0.0
4	C:/Users/Public/dataset/1_normal	NL_005.png	1_normal	0.0

...

Figure 9

a) Pre-processing data and exploring Features

```
img_visual = cv.imread('C:/Users/Public/dataset/1_normal/NL_011.png')
img_visual = cv.cvtColor(img_visual, cv.COLOR_BGR2RGB)
image_view(img_visual)
```

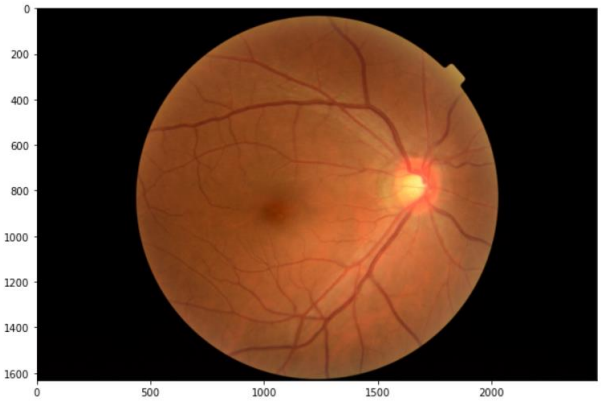

Figure 10

Converting the original image to Grayscale

```
test_gray = cv.cvtColor(img_visual, cv.COLOR_RGB2GRAY)
image_view(test_gray)
```

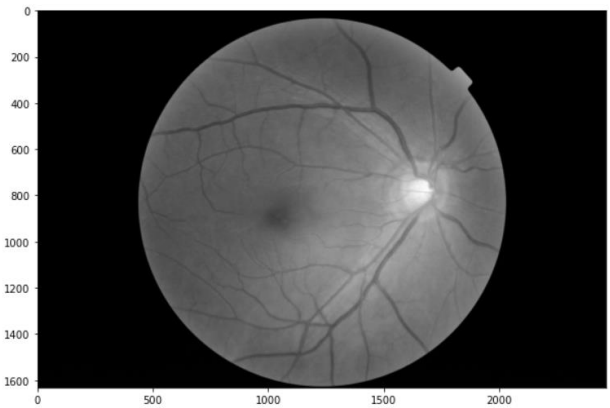

Figure 11

Converting the Greyscale image to Black white image as it has only two values (0 and 1). Adaptive threshold Gaussian is used to convert the image into binary image so that we can exclude the unwanted background from Image and get our Region of Interest..

```
test_thresh = cv.adaptiveThreshold(test_gray,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY_INV,11,3)
th2 = cv.adaptiveThreshold(test_gray,255,cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY,11,2)
image_view(test_thresh)
image_view(th2)
```

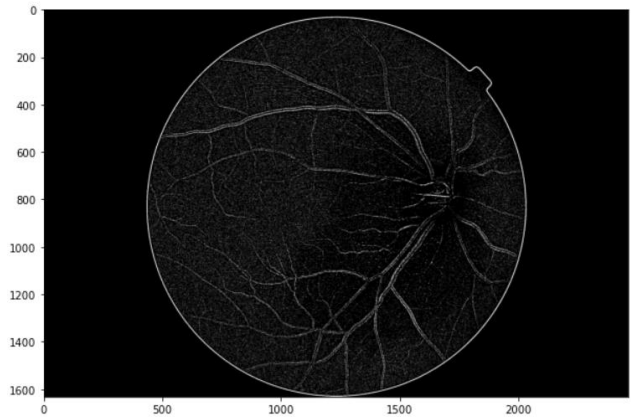

Figure 12

Resizing Image to 400 x 400

```
test_resize = cv.resize(image_Region, (int(width/4), int(height/4)))
image_view(test_resize)
```

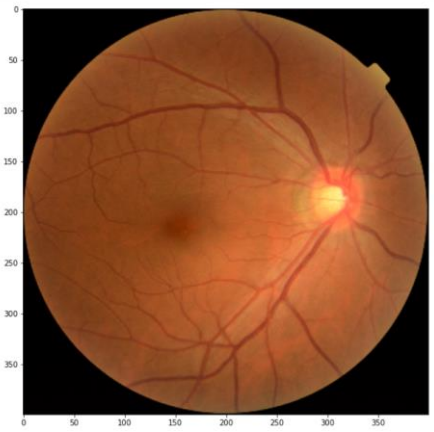

Figure 13

Creating a Function to extract the features of image (Contrast, Energy, Correlation, Homogenity). The distance and theta used are 10 and 90 respectively. The greycomatrix function returns the number of the grey level at some pre-defined distance and pre-defined angle.

```
def feature_glcm_img(matx_occur, Name_f):
    image_features = greycoprops(matx_occur, Name_f)
    r = np.average(image_features)
    return r
distance = 10
teta = 90

contrast = []
homogeneity = []
Energy = []
correlation_test = []


GLCM = greycomatrix(test_resize_gray, [distance], [teta], levels=256, symmetric=True, normed=True)
contrast.append(feature_glcm_img(GLCM, 'contrast'))
homogeneity.append(feature_glcm_img(GLCM, 'homogeneity'))
Energy.append(feature_glcm_img(GLCM, 'energy'))
correlation_test.append(feature_glcm_img(GLCM, 'correlation'))

print(f'Homogenity : {homogeneity[0]}')
print(f'Correlation : {correlation_test[0]}')
print(f'Energy : {Energy[0]}')
print(f'Contrast : {contrast[0]}')
```

```
Homogenity : 0.2978756405081341
Correlation : 0.906593444872137
Energy : 0.09626991455583109
Contrast : 383.9978999135498
```

Figure 14

Looping over all the images (Normal, Cataract, Glaucoma, Retina Disease) to get the features of the image.

```
Normal_eye = 301
Cataract_eye = 101
Glaucoma = 102
Retina = 101
width, height = 400, 400
distance = 10
teta = 90
Eye_info = np.zeros((5, 601))
count = 0
indx = ['contrast', 'homogenity', 'energy', 'correlation', 'Label']

normalpath = 'C:/Users/Public/dataset/1_normal/'
cataractpath = 'C:/Users/Public/dataset/2_cataract/'
glaucomapath = 'C:/Users/Public/dataset/2_glaucoma/'
retinapath = 'C:/Users/Public/dataset/3_retina_disease/'
```

Figure 15

```
for file in range(1, Normal_eye):
    contrast = []
    homogeneity = []
    energy = []
    correlation = []
    label = 0
    image = cv.imread(f'{normalpath}/NL_{str(file).zfill(3)}.png')
    img = Image1(image)

    GLCM = greycomatrix(img, [distance], [teta], levels=256, symmetric=True, normed=True)
    contrast.append(feature_glcm_img(GLCM, 'contrast'))
    homogeneity.append(feature_glcm_img(GLCM, 'homogeneity'))
    energy.append(feature_glcm_img(GLCM, 'energy'))
    correlation.append(feature_glcm_img(GLCM, 'correlation'))

    Eye_info[0, count] = contrast[0]
    Eye_info[1, count] = homogeneity[0]
    Eye_info[2, count] = energy[0]
    Eye_info[3, count] = correlation[0]
    Eye_info[4, count] = label

    count += 1
```

Figure 16

Making a DataFrame of all the features of image and assigning the data into columns.

```
df_1= pd.DataFrame(np.transpose(Eye_info), columns = indx)
df_1
```

	contrast	homogenity	energy	correlation	Label
0	204.125636	0.313721	0.096960	0.894945	0.0
1	335.220717	0.277447	0.093683	0.912148	0.0
2	301.578832	0.318182	0.104449	0.887562	0.0

Figure 17

b) Feature Scaling

Feature Scaling using MinMaxScaler. The MinMaxScaler usually returns 0 or 1. This is done to ensure that we get the values with a range(0 ,1).

```
from sklearn.preprocessing import MinMaxScaler
eye_data = df_1.drop(['Label'], axis='columns')
fscaler = MinMaxScaler()
eye_data = fscaler.fit_transform(eye_data)
eye_data
```

```
array([[0.09319899, 0.32973337, 0.09622744, 0.60695136],
       [0.207082  , 0.21058521, 0.05606246, 0.72606934],
       [0.17785711, 0.34438499, 0.18800413, 0.55583088],
       ...,
       [0.2929104 , 0.40573931, 0.32174015, 0.73146799],
       [0.19388191, 0.14765057, 0.3105528 , 1.        ],
       [0.27672098, 0.12923814, 0.30218012, 0.85711199]])
```

Figure 18

c) Classification using Machine Learning Techniques:

Splitting the data using train_test_split. 25% is taken for testing purpose and 75% is taken for training purpose.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.25)
```

Figure 19

Hyperparameter tuning using GridSearchCV

The best possible parameter for the Model among the various parameters passed is searched by GridSearchCV. It is available in sklearn model_selection package.

- **Random Forest Classifier**

Random Forest Classifier

```
from sklearn.model_selection import GridSearchCV

param = {'n_estimators': [5,10,20,40,90]}
random = RandomForestClassifier()
clf = GridSearchCV(random, param)
clf.fit(x, y)
sorted(clf.cv_results_.keys())
print(f"Random Forest Classifier Best paramaters {clf.best_params_}")
print(f"Random Forest Classifier Model score {clf.best_score_}")
```

```
Random Forest Classifier Best paramaters {'n_estimators': 90}
Random Forest Classifier Model score 0.5341597796143249
```

Figure 20

Predicted value for test data

```
#prediction on the test data
y_predicted1 = model2.predict(x_test)
y_predicted1
```

```
array([0., 2., 0., 0., 2., 3., 0., 0., 0., 1., 0., 0., 0., 0., 2., 0., 2.,
       0., 1., 0., 0., 0., 0., 0., 3., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 3., 0., 0., 3., 0., 3.,
       0., 1., 0., 0., 0., 0., 2., 0., 0., 0., 2., 0., 0., 1., 0., 3., 0.,
       0., 1., 3., 0., 1., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 3.,
       1., 0., 0., 2., 0., 0., 2., 0., 1., 2., 0., 2., 0., 0., 1., 2., 0.,
       1., 2., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 3., 0., 0., 3.,
       3., 1., 0., 0., 0., 0., 1., 3., 0., 2., 1., 0., 0., 0., 0.])
```

Figure 21

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sb
y_predicted1 = model2.predict(x_test)
cm = confusion_matrix(y_test,y_predicted1)
plt.figure(figsize = (10,7))
sb.heatmap(cm, annot=True,cmap="YlOrRd",fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Figure 22

Classification Report

```
: from sklearn.metrics import classification_report
  print(classification_report(y_test,y_predicted1))
```

Figure 23

- **K-Nearest Neighbors Classifier**

GridSearchCV is again applied to find the best parameters for KNN

```
from sklearn.model_selection import GridSearchCV

param = {'n_neighbors': [7,9,13,17]}
KNN = KNeighborsClassifier()
clf = GridSearchCV(KNN, param)
clf.fit(x, y)
sorted(clf.cv_results_.keys())
print(f"KNN Best paramaters {clf.best_params_}")
print(f"K-Nearest Neighbors Classifier Model score {clf.best_score_}")
```

```
KNN Best paramaters {'n_neighbors': 17}
K-Nearest Neighbors Classifier Model score 0.5657851239669421
```

Figure 24

The predicted value obtained from KNN with test data are as follows:

```
#prediction on the test data for KNN
y_predicted2 = model3.predict(x_test)
y_predicted2
```

```
array([0., 1., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 3., 0.,
       0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       1., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
       0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.])
```

Figure 25

Confusion Matrix for KNN

```
from sklearn.metrics import confusion_matrix
import seaborn as sb
y_predicted2 = model3.predict(x_test)
cm = confusion_matrix(y_test,y_predicted2)
plt.figure(figsize = (10,7))
sb.heatmap(cm, annot=True,cmap="YlOrRd",fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Figure 26

Classification report for KNN

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_predicted2))
```

Figure 27

- **Support Vector Machine**

SVM implementation using GridSearchCV to find the best kernel and vectors, it is shown below:

```
from sklearn.model_selection import GridSearchCV

param = {'kernel':['rbf','linear','poly','sigmoid'], 'C':[5,15,30,50]}
svc = svm.SVC()
clf = GridSearchCV(svc, param)
clf.fit(x, y)
sorted(clf.cv_results_.keys())
print(f"SVM Best paramaters {clf.best_params_}")
print(f"SVM Model score {clf.best_score_}")
```

Figure 28

Y predict value for test data for SVM are:

```
#prediction on the test data for SVM
y_predicted3 = model4.predict(x_test)
y_predicted3
```

```
array([0., 1., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
       0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       1., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
       0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Figure 29

Confusion Matrix:

```
from sklearn.metrics import confusion_matrix
import seaborn as sb
y_predicted3 = model4.predict(x_test)
cm = confusion_matrix(y_test,y_predicted3)
plt.figure(figsize = (10,7))
sb.heatmap(cm, annot=True,cmap="YlOrRd",fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Figure 30

Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_predicted3))
```

Figure 31

d) Classification using CNN

Making array of train, test and validation DataFrame.

```
train_df, test_df = train_test_split(df2,
                                     test_size=0.2,
                                     random_state=SEED,
                                     stratify=df2['Label1'])

train_df, val_df = train_test_split(train_df,
                                    test_size=0.15,
                                    random_state=SEED,
                                    stratify=train_df['Label1'])
```

Figure 32

Making array of image dataset using train_test_split.

```
def create_datasets(df2, img_width, img_height):
    imgs = []
    for path in tqdm(df2['path']):
        img = cv.imread(path)
        img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
        img = cv.resize(img, (img_width, img_height))
        imgs.append(img)

    imgs = np.array(imgs, dtype='float32')
    df = pd.get_dummies(df2['Label1'])
    return imgs, df2


train_imgs, train_df = create_datasets(train_df, IMG_WIDTH, IMG_HEIGHT)
val_imgs, val_df = create_datasets(val_df, IMG_WIDTH, IMG_HEIGHT)
test_imgs, test_df = create_datasets(test_df, IMG_WIDTH, IMG_HEIGHT)

train_imgs = train_imgs / 255.0
val_imgs = val_imgs / 255.0
test_imgs = test_imgs / 255.0
```

Figure 33

CNN with 4 convolution layers and 2 Pooling layers.

```
from tensorflow.keras.utils import get_custom_objects

class Mish(tf.keras.layers.Layer):

    def __init__(self, **kwargs):
        super(Mish, self).__init__(**kwargs)
        self.supports_masking = True

    def call(self, inputs):
        return inputs * K.tanh(K.softplus(inputs))

    def get_config(self):
        base_config = super(Mish, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))

    def compute_output_shape(self, input_shape):
        return input_shape
def mish(x):
    return tf.keras.layers.Lambda(lambda x: x*K.tanh(K.softplus(x)))(x)

get_custom_objects().update({'mish': Activation(mish)})
```

Figure 34

I have used a self-activation function ‘mish’ as it smoothens the model and shows maximum accuracy and stability to model than Relu. I have created CNN model using all the convolution, Pooling, and required layers. The Sequential layers is the initial layer which has add function. The layers are added to sequential layer.

```
model = Sequential()
model.add(Conv2D(16, kernel_size=3, padding='same',input_shape=(train_imgs.shape[1:]), activation='mish'))
model.add(Conv2D(16, kernel_size=3, padding='same', activation='mish'))
model.add(MaxPool2D(3))
model.add(Dropout(0.3))
model.add(Conv2D(16, kernel_size=3, padding='same', activation='mish'))
model.add(Conv2D(16, kernel_size=3, padding='same', activation='mish'))
model.add(MaxPool2D(3))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(2, activation='softmax'))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])


model.summary()
```

Figure 35

I have used ‘adam’ optimizer and loss as sparse categorical crossentropy as a parameter to compile the model.

The summary of the model is:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 192, 256, 16)      448

 conv2d_1 (Conv2D)           (None, 192, 256, 16)      2320

 max_pooling2d (MaxPooling2D  (None, 64, 85, 16)       0
 )

 dropout (Dropout)           (None, 64, 85, 16)        0

 conv2d_2 (Conv2D)           (None, 64, 85, 16)        2320

 conv2d_3 (Conv2D)           (None, 64, 85, 16)        2320

 max_pooling2d_1 (MaxPooling  (None, 21, 28, 16)       0
 2D)

 dropout_1 (Dropout)         (None, 21, 28, 16)        0

 flatten (Flatten)           (None, 9408)              0

 dense (Dense)               (None, 2)                 18818

=================================================================
Total params: 26,226
Trainable params: 26,226
Non-trainable params: 0
```

Figure 36

Before fitting the model, Data Augmentation is done to get more data for model to train. I have set the horizontal flip parameter as TRUE. Batch size used is 32. So, 32 training samples were used for 1 iteration.

```
SEED=42
EPOCHS = 100
BATCH_SIZE = 32
```

Figure 37

```
history = model.fit(generator.flow(train_imgs,
                                   labels_train,
                                   batch_size=BATCH_SIZE),
                    epochs=30,
                    steps_per_epoch=len(train_imgs)/BATCH_SIZE,
                    callbacks=[es_callback, reduce_lr],
                    validation_data=(val_imgs, labels_val))


pd.DataFrame(history.history)[['accuracy', 'val_accuracy']].plot()
pd.DataFrame(history.history)[['loss', 'val_loss']].plot()
plt.show()
```

Figure 38

The Model took around 15mins to run 30 EPOCHS.

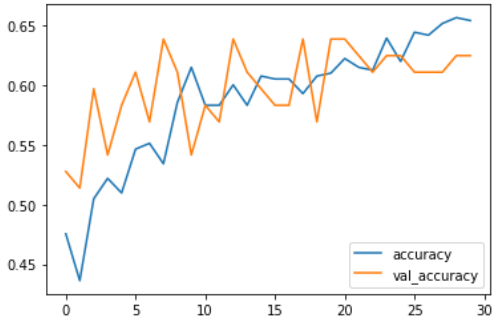

Figure 39

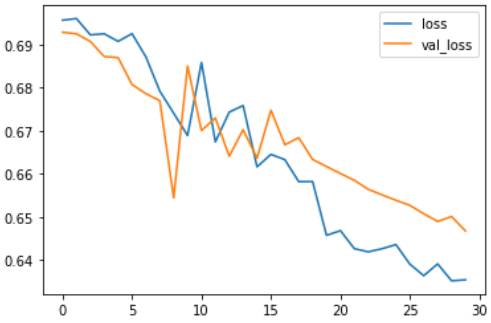

Figure 40

The Accuracy and Validation accuracy. There was a gradual decrease in loss and the model shown approximate 68% accuracy, so EPOCH 30 was used.

CNN Model 2:

I have created another CNN model to achieve better results.

For this I have created train and test dataset based on the label, the test size taken is 0.2

```
train, test = train_test_split(df3, stratify= df3["Label1"], test_size= 0.2)
```

```
train1=train.reset_index()
```

```
test1=test.reset_index()
```

Figure 39

As the index were randomly assigned to train and test, I have later reset the training dataframe and testing dataframe.

To load the x-train and y-train dataset, I have made use of two list x_train1 and y_train1. For loop is used to iterate over folder name and file name and merge them to get the image. After getting the image, I have resized and reshaped the image and get_dummies is used to categories the disease with 0 and 1.

```
x_train1=[]
y_train1=[]
for t in range(len(df3)):
    folder = df3["folder_name"][t]
    file1 = df3["files"][t]
    image = cv.imread(os.path.join(folder, file1))
    image = cv.resize(image, (IMG_SIZE, IMG_SIZE))
    image = image/255.0
    image = image.reshape(-1,IMG_SIZE,IMG_SIZE,3)
    image = image.reshape(image.shape[1], image.shape[2],image.shape[3])
    if t % 50 == 0:
        print(t)

    x_train1.append(image)
    y_train1.append(df3["Label1"][t])
```

Figure 40

CNN Model is applied which has 2 convolution layers, the activation function used is Relu which is mostly default in many neural networks as it overcomes the vanishing gradient problem, Batch Normalization layer as it stabilises the variable gradients and Pooling layer.

```
model_1 = Sequential()

model_1.add(Conv2D(64, 3, 3, padding = 'same', input_shape=(x_train1.shape[1:])))
model_1.add(Activation('relu'))
BatchNormalization(axis=1)
model_1.add(Conv2D(64,3,3 , padding = 'same'))
model_1.add(Activation('relu'))
model_1.add(MaxPooling2D(pool_size=(2,2)))
model_1.add(Flatten())
# Fully connected layer

BatchNormalization()
model_1.add(Dense(512))
model_1.add(Activation('relu'))
BatchNormalization()
model_1.add(Dropout(0.5))
model_1.add(Dense(4))
model_1.add(Activation('softmax'))
```

Figure 41

The Model summary is shown below:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 34, 34, 64)        1792

 activation (Activation)     (None, 34, 34, 64)        0

 conv2d_1 (Conv2D)           (None, 12, 12, 64)        36928

 activation_1 (Activation)   (None, 12, 12, 64)        0

 max_pooling2d (MaxPooling2D  (None, 6, 6, 64)         0
 )

 flatten (Flatten)           (None, 2304)              0

 batch_normalization_1 (Batc  (None, 2304)             9216
 hNormalization)

 dense (Dense)               (None, 512)               1180160

 activation_2 (Activation)   (None, 512)               0

 batch_normalization_2 (Batc  (None, 512)              2048
 hNormalization)

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 4)                 2052

 activation_3 (Activation)   (None, 4)                 0

=================================================================
Total params: 1,232,196
Trainable params: 1,226,564
Non-trainable params: 5,632
```

Figure 42

After that, as of previous model, I compiled the model, the optimizer used is ‘adam’ and loss used is categorical_crossentropy.

Data Augmentation is done after this as to create more images to train the data. Horizontal flip is kept TRUE.

```
generator = ImageDataGenerator(horizontal_flip=True,
                               height_shift_range=0.1,
                               fill_mode='reflect')



es_callback = tf.keras.callbacks.EarlyStopping(patience=20,
                                               verbose=1,
                                               restore_best_weights=True)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(factor=0.1, patience=10, verbose=1)
```

Figure 43

```
history = model_1.fit(generator.flow(x_train1,
                                     y_train1,
                                     batch_size=32),
                      epochs=50,
                      steps_per_epoch=len(x_train1)/32,
                      callbacks=[es_callback, reduce_lr],
                      validation_data=(x_test1, y_test1))


pd.DataFrame(history.history)[['accuracy', 'val_accuracy']].plot()
pd.DataFrame(history.history)[['loss', 'val_loss']].plot()
plt.show()
```

Figure 44

The Augmented data is then fitted, and accuracy and validation accuracy are plotted against EPOCH. The EPOCH is set to 50 and steps per epoch is set to length of training data to batch size.

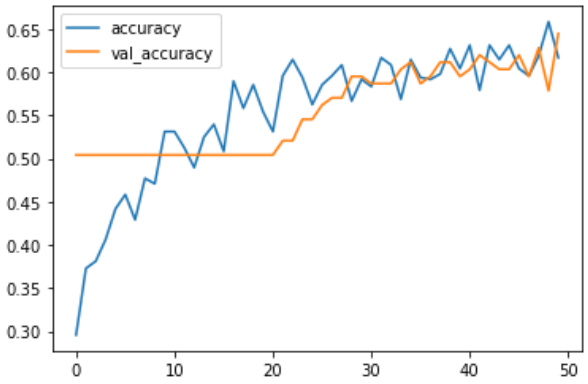

Figure 45

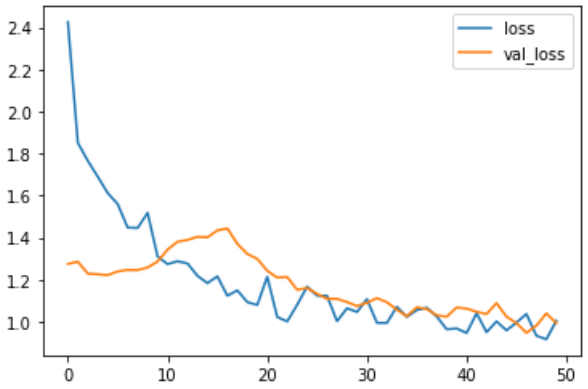

Figure 46