Configuration Manual

MSc Research Project
Data Analytics

Mayuresh Londhe
Student ID: X20137265

School of Computing
National College of Ireland

Supervisor: Dr. Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	**Mayuresh Londhe**
Student ID:	**X20137265**
Programme:	**Msc. Data Analytics** **Year: 2020/2021**
Module:	**Research Project**
Lecturer:	**Dr. Christian Horn**
Submission Due Date:	**16/08/2021**
Project Title:	**Classification of Eye Diseases using Hybrid CNN-RNN Models.**
Word Count:	**1542** **Page Count: 12**

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

Signature: ……

Date: 16/08/2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

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

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

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

Configuration Manual

Mayuresh Londhe
Student ID: X20137265

1 Introduction

This Configuration Manual consists of all the perquisites required to reproduce the research and its outcomes on individual environment. The software and the hardware requirement along with a snapshot of code for Data Import and Pre-processing, Data Augmentation, Exploratory Data Analysis, all the models-built K- Fold Cross Validation and Evaluation are included.

The structure of the report is as follows, Section 2, gives the information about environment configuration. Section 3, provides detail about data collection. Section 4 is data exportation consists of Data Pre-processing and Exploratory Data Analysis. Data Augmentation is explained in section 5. Section 6 provides the details about Data splitting to get equal images in each class for training, validation and testing phase, data transformations and feature label separation performed. Section 7 provides the details about the models built. Section 8, explains the implementation of K-Fold Validation. Section 9, explains how results are computed and visualized.

2 Environment

This section provides the details of Software and Hardware requirements to implement the research done.

2.1 Hardware Requirements

Below Figure 1, provides the hardware specifications required. Intel i5-1135G7 is the 11th Generation Intel Core CPU @ 2.42 GHz, 16 GB installed DDR4 RAM Memory at speed of 3200 Mhz, 64 Bit Windows 10 operating System, 512 GB SSD.

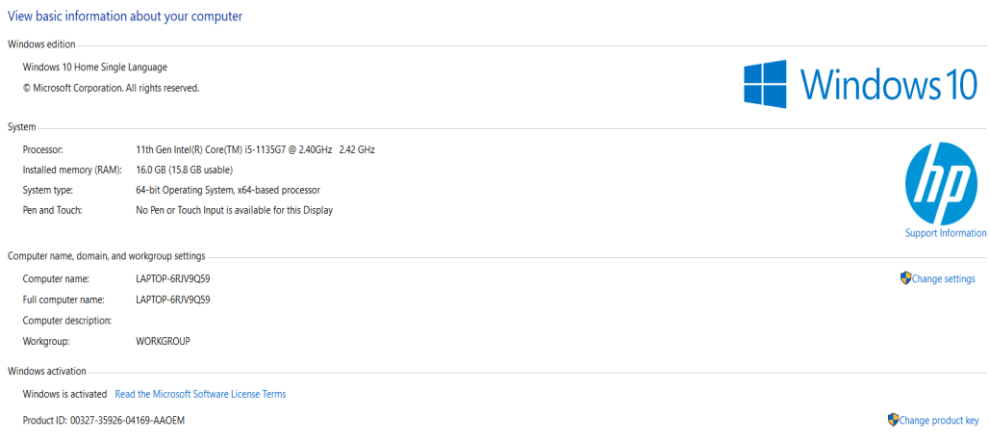

Figure 1: Hardware Requirements

2.2 Software Requirements

- Anaconda Navigator for Windows (Version 1.9.12)
- Jupyter Notebook (Version 6.0.3)
- Python (Version 3.8.3)

3 Data Collection

The dataset is taken from Kaggle public repository. https://www.kaggle.com/jr2ngb/cataractdataset is the link for the dataset. There are 601 images in dataset there is one extra image in the Glaucoma class which is removed therefore there would 300 images in Normal and 100 images each in Cataract, Glaucoma and Retina diseases class. Link given below can also be used.

https://studentncirl-my.sharepoint.com/:f:/g/personal/x20137265_student_ncirl_ie/EtKlBNkXNIlOl7rKmHSqc30BbycJ40sThYMa2yjvzK9ufA?e=9iLT3F

4 Data Exportation

4.1 Importing the Libraries

All the Python libraries required to implement the entire project are listed in Figure 2.

Imported Libraries

In [1]:

```
import tensorflow as tf
import keras
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout, BatchNormalization, LSTM, Input, Reshape
from tensorflow.keras.applications import InceptionV3,InceptionResNetV2,VGG19,DenseNet169
from tensorflow.keras.losses import sparse_categorical_crossentropy
from tensorflow.keras.optimizers import RMSprop
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
import sys
import cv2
import os
```

Figure 2: Required Python Libraries

4.2 Exploratory Data Analysis

4.2.1 To Check Individual Image Size

The Figure 3 and 4 represents the code to visualize and count the number of images for each size. There are 3 different size images are available (1632 x 2464), (1224 x 1848) and (1728 x 2592).

```
dataset_folder_path="dataset/folder_path"
folder= ['1_normal','2_cataract','3_glaucoma','4_retina_disease']
count=0
count_1=0
count_2=0
size_count=[]

for folder in folder:
    path_dataset_join =os.path.join(dataset_folder_path, folder)
    for img in os.listdir(path_dataset_join):
        image= cv2.imread(os.path.join(path_dataset_join, img))
        image_name= img


        if (image.shape == (1632, 2464, 3)):
            count= count+1
            size_count.append("1632*2464")
        elif (image.shape == (1728, 2592, 3)):
            count_1=count_1+1
            size_count.append("1728*2592")
        elif (image.shape == (1224, 1848, 3)):
            count_2=count_2+1
            size_count.append("1224*1848")
        else:
            print(image_name)
            print(image.shape)
```

Figure 3: EDA for Checking Individual Image Size

```
print(count,count_1,count_2)
```

```
158 403 40
```

```
sns.set_style('darkgrid')

sns.countplot(size_count,palette='viridis')
```

Figure 4: Print the Count and Visualization

4.3 Data Pre-processing

As seen in Figure 3 there are 158 that have (1632 x 2464) size, 403 images with (1728 x 2592) size and 40 images with size (1224 x 1848). Here the (1224 x 1848) and (1632 x 2464) images are converted to (1728 x 2592). The Figure 5, illustrate the code to convert all the images to one size. Here, original dataset path need to be provided along with that need to create new folder with name dataset_all_equal_size_images which contains four empty folders with the name 1_normal, 2_cataract,3_glaucoma and 4_retina_diseases. Path of these four folders also used to store the resized Images.

```
dataset_folder_path="dataset/folder_path"
folder= ['1_normal','2_cataract','3_glaucoma','4_retina_disease']

height=1728
width=2592

store_path_1="dataset_all_equal_size_image/1_normal"
store_path_2="dataset_all_equal_size_image/2_cataract"
store_path_3="dataset_all_equal_size_image/3_glaucoma"
store_path_4="dataset_all_equal_size_image/4_retina_disease"

for folder in folder:
    path_dataset_join =os.path.join(dataset_folder_path, folder)
    for img in os.listdir(path_dataset_join):
        image= cv2.imread(os.path.join(path_dataset_join, img))
        image_name= img

        if folder == '1_normal':

            if (image.shape == (1632, 2464, 3)):

                image=cv2.resize(image, (width,height))

                cv2.imwrite(os.path.join(store_path_1,image_name),image)

            elif (image.shape == (1728, 2592, 3)):
                cv2.imwrite(os.path.join(store_path_1,image_name),image)


            elif (image.shape == (1224, 1848, 3)):
                image=cv2.resize(image, (width,height))
                cv2.imwrite(os.path.join(store_path_1,image_name),image)

        elif folder == '2_cataract':

            if (image.shape == (1632, 2464, 3)):
                image=cv2.resize(image, (width,height))
                cv2.imwrite(os.path.join(store_path_2,image_name),image)

            elif (image.shape == (1728, 2592, 3)):
                cv2.imwrite(os.path.join(store_path_2,image_name),image)

            elif (image.shape == (1224, 1848, 3)):
                image=cv2.resize(image, (width,height))
                cv2.imwrite(os.path.join(store_path_2,image_name),image)
```

```
    elif folder == '3_glaucoma':

        if (image.shape == (1632, 2464, 3)):
            image=cv2.resize(image, (width,height))
            cv2.imwrite(os.path.join(store_path_3,image_name),image)

        elif (image.shape == (1728, 2592, 3)):
            cv2.imwrite(os.path.join(store_path_3,image_name),image)

        elif (image.shape == (1224, 1848, 3)):
            image=cv2.resize(image, (width,height))
            cv2.imwrite(os.path.join(store_path_3,image_name),image)

    elif folder == '4_retina_disease':

        if (image.shape == (1632, 2464, 3)):
            image=cv2.resize(image, (width,height))
            cv2.imwrite(os.path.join(store_path_4,image_name),image)

        elif (image.shape == (1728, 2592, 3)):
            cv2.imwrite(os.path.join(store_path_4,image_name),image)

        elif (image.shape == (1224, 1848, 3)):
            image=cv2.resize(image, (width,height))
            cv2.imwrite(os.path.join(store_path_4,image_name),image)
```

Figure 5: Code to Convert All Images Equal Size.

all equal size images : https://studentncirl-my.sharepoint.com/:f:/g/personal/x20137265_student_ncirl_ie/EvAXoG3kxslOksAk3-HU01UBdAYWLYXOZNM7CxiA-HbIUQ?e=ViIG7T

5 Data Augmentation:

To accomplish the class balance across all labels. The images of the Cataract, Glaucoma and Retina Diseases need to be augmented using horizontal filliping and horizontally and vertically together. To achieve this, need to create a copy of the dataset_all_equal_size_images dataset and rename it as all_equal_300_images. File path of the copied and renamed dataset along with 3 folder names for which images need to augmented is passed.

```
folder= ['2_cataract','3_glaucoma','4_retina_disease']

dataset_folder_path= "dataset/all_equal_300_images"


for folder in folder:
    path_dataset_join =os.path.join(dataset_folder_path, folder)

    for img in os.listdir(path_train_AUG):
        image= cv2.imread(os.path.join(path_dataset_join, img))
        image_name= img

        image_flipped_vertically= cv2.flip(image, 1)
        cv2.imwrite(os.path.join(path_dataset_join,image_name + '_image_flipped_vertically.png'),image_flipped_vertica
lly)


        image_flipped_horizontally_vertically= cv2.flip(image, -1)
        cv2.imwrite(os.path.join(path_dataset_join,image_name + '_image_flipped_horizontally_vertically.png'),image_fl
ipped_horizontally_vertically)
```

Figure 6: Code to Perform Data Augmentation

After this process, 3 datasets will be present original dataset, dataset_all_equal_size_images dataset that would be used for implementing the models on imbalanced dataset and all_equal_300_images will be used as a balanced dataset.

all equal 300 images: https://studentncirl-my.sharepoint.com/:f:/g/personal/x20137265_student_ncirl_ie/Eg7mkMbUkgFKiPsNsBfMXUQBG8tTdqNX5Fhzc_tnPREP0Q

6 Data Preparation

6.1 Dataset Import for Model building

The get_data() is defined that takes the path of dataset, depending upon whether model is needed to be implemented on balanced or imbalanced dataset. Along with data import, the

function performs the conversion of the images imported in BGR format to RGB, Black background reduction and image reseize to 224 x 224 dimensions.

```
labels = ['1_normal', '2_cataract','3_glaucoma','4_retina_disease']
img_size = 224
def get_data(data_dir):
    data = []

    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img))[...,::-1] #convert BGR to RGB format
                crop_image= img_arr[0:1728,430:2190]
                resized_arr = cv2.resize(crop_image, (img_size, img_size)) # Reshaping images to preferred size
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)
    return np.array(data)
```

```
#function call to get_data function that takes file path of the dataset.
data= get_data('dataset/dataset_all_equal_size_image/')
```

```
#function call to get_data function that takes file path of the dataset.
data= get_data('dataset/all_equal_300_images/')
```

Figure 7: Dataset Import for Model Building

6.2 Data Splitting

The Figure 8, below provides the code for splitting the imported data into four different data object that contains images for the respective class. This is done to ensure that number of images in each class after the training validation and testing phase would be equal.

```
normal= data[0:300]
normal.shape
```

```
(300, 2)
```

```
cataract=data[300:600]
cataract.shape
```

```
(300, 2)
```

```
glaucoma= data[600:900]
glaucoma.shape
```

```
(300, 2)
```

```
retina_disease= data[900:1200]
retina_disease.shape
```

```
(300, 2)
```

```
normal= data[0:300]
normal.shape
```

```
(300, 2)
```

```
cataract=data[300:400]
cataract.shape
```

```
(100, 2)
```

```
glaucoma= data[400:500]
glaucoma.shape
```

```
(100, 2)
```

```
retina_disease= data[500:600]
retina_disease.shape
```

```
(100, 2)
```

Figure 8: Left: Data Splitting for Imbalanced dataset Right: Data Splitting for Balanced data

6.3 Dataset Random Shuffle

Figure 9, represents the code to shuffle the separated dataset for each class. This code block will remain same for both Imbalanced and Balanced datasets.

```
random.seed(15)
np.random.shuffle(normal)
np.random.shuffle(cataract)
np.random.shuffle(glaucoma)
np.random.shuffle(retina_disease)
```

Figure 9: Dataset Random Shuffle

6.4 Feature and Label Separation

The model takes input as images and labels, therefore the function image_label_split() takes the three arguments train, validation and test.

```python
def image_label_split(train,validation,test):

    x_train = []
    y_train = []
    x_val = []
    y_val = []
    x_test = []
    y_test = []

    for feature, label in train:
      x_train.append(feature)
      y_train.append(label)

    for feature, label in validation:
      x_val.append(feature)
      y_val.append(label)

    for feature, label in test:
      x_test.append(feature)
      y_test.append(label)

    y_train = np.array(y_train)
    y_val = np.array(y_val)
    y_test= np.array(y_test)

    return (x_train,y_train,x_val,y_val,x_test,y_test)
```

Figure 10: Feature and Label Separation

6.5 Data Transformation

The transformation method used is normalization, the function takes the image features as input for all three phases. The functions mentioned in Sections 6.3 and 6.4 need to be executed before model compilation.

```python
def normalize(x_train,x_val,x_test):

    x_train = np.array(x_train) / 255
    x_train.reshape(-1, img_size, img_size, 1)

    x_test= np.array(x_test) / 255
    x_test.reshape(-1, img_size, img_size, 1)

    x_val= np.array(x_val) / 255
    x_val.reshape(-1, img_size, img_size, 1)

    return (x_train,x_val,x_test)
```

Figure 11: Normalization

7 Models Architecture Implementation

All the implemented models are used on both Imbalanced and Balanced dataset. All the models are implemented using pre-trained weights.

7.1 InceptionV3-LSTM

```python
def model_build_compile(k):
    baseModel = InceptionV3(weights="imagenet", include_top=False, input_tensor=Input(shape=(224, 224, 3)))
    for layer in baseModel.layers:
            layer.trainable = False

    x = baseModel.output

        # LSTM layer
    x = Reshape((25, 2048))(x)
    x = ((LSTM(512, activation="relu", return_sequences=True, trainable=False)))(x)
    x = BatchNormalization()(x)

        # FC layer
    x = Flatten(name="flatten")(x)

        # fc1 layer
    x = Dense(units=4096, activation='relu')(x)
    x = BatchNormalization()(x)

        # fc2 layer
    x = Dense(units=4096, activation='relu')(x)
    x = BatchNormalization()(x)

        # Output layer
    output = Dense(units=4, activation='softmax')(x)

    model = Model(inputs=baseModel.input, outputs=output)
    opt = RMSprop(learning_rate=0.01, clipvalue=100)
    model.compile(loss='sparse_categorical_crossentropy', optimizer=opt, metrics=["accuracy"])
    k=k+1
    print("model building and compiling for fold",k)
    return model
```

Figure 12: InceptionV3-LSTM Model Implementation

7.2 InceptionResNetV2-LSTM

```
def model_build_compile(k):
    baseModel = InceptionResNetV2(weights="imagenet", include_top=False, input_tensor=Input(shape=(224, 224, 3)))
    for layer in baseModel.layers:
            layer.trainable = False

    x = baseModel.output

        # LSTM layer
    x = Reshape((25, 2048))(x)
    x = ((LSTM(512, activation="relu", return_sequences=True, trainable=False)))(x)
    x = BatchNormalization()(x)

        # FC layer
    x = Flatten(name="flatten")(x)

        # fc1 layer
    x = Dense(units=4096, activation='relu')(x)
    x = BatchNormalization()(x)

        # fc2 layer
    x = Dense(units=4096, activation='relu')(x)
    x = BatchNormalization()(x)

        # Output layer
    output = Dense(units=4, activation='softmax')(x)

    model = Model(inputs=baseModel.input, outputs=output)
    opt = RMSprop(learning_rate=0.01, clipvalue=100)
    model.compile(loss='sparse_categorical_crossentropy', optimizer=opt, metrics=["accuracy"])
    k=k+1
    print("model building and compiling for fold",k)
    return model
```

Figure 13: InceptionResNetV2-LSTM Model Implementation

7.3 DenseNet169-LSTM

```
def model_build_compile(k):
    baseModel = DenseNet169(weights="imagenet", include_top=False, input_tensor=Input(shape=(224, 224, 3)))
    for layer in baseModel.layers:
            layer.trainable = False

    x = baseModel.output

        # LSTM layer
    x = Reshape((25, 2048))(x)
    x = ((LSTM(512, activation="relu", return_sequences=True, trainable=False)))(x)
    x = BatchNormalization()(x)

        # FC layer
    x = Flatten(name="flatten")(x)

        # fc1 layer
    x = Dense(units=4096, activation='relu')(x)
    x = BatchNormalization()(x)

        # fc2 layer
    x = Dense(units=4096, activation='relu')(x)
    x = BatchNormalization()(x)

        # Output layer
    output = Dense(units=4, activation='softmax')(x)

    model = Model(inputs=baseModel.input, outputs=output)
    opt = RMSprop(learning_rate=0.01, clipvalue=100)
    model.compile(loss='sparse_categorical_crossentropy', optimizer=opt, metrics=["accuracy"])
    k=k+1
    print("model building and compiling for fold",k)
    return model
```

Figure 14: DenseNet169-LSTM Model Implementation

8 K-Fold Cross Validation Implementation

The steps taken till now are necessary to start the model compilation and model fitting stage, an additional step required which is not explained till now is explained in model evaluation section that is test_pred() which is explained at section 9.1. This function predicts the test labels and after that test confusion matrix, sensitivity and specificity is derived. Figure 15,16 represents the K fold validation implementation for K=5. Figure 15, contains the empty lists created to store the model derived training and validation phase accuracy and loss along with test phase predicted confusion matrices, accuracy, sensitivity and specificity.

```
CM= []
test_accuracy=[]
test_sensitivity=[]
test_specificity=[]
train_acc = []
val_acc = []
train_loss = []
val_loss = []
```

Figure 15: Variables to Store Model Generate Result

```
for k in range (5):  # for loop to run 5 folds
    n_normal=30      # specifying the number of images for normal class in test phase,calulated as per 10% of total no
rmal class images 300.
    n_rest=10        # specifying the number of images for disease classes in test phase,calulated as per 10% of total
normal class images 100.

    # Adding the images in normal validation set by using k*n_normal to (k+1)*n_normal as index values for normal data
set divided in cell 6.
    test_normal= normal[k*n_normal:(k+1)*n_normal]
    print('------------------------Start of',k+1,'Fold------------------------')
    print('test images for normal class from',k*n_normal,(k+1)*n_normal)

    # Adding the images in cataract validation set by using k*n_rest to (k+1)*n_rest as index values for cataract data
set divided in cell 7.
    test_cataract= cataract[k*n_rest:(k+1)*n_rest]
    print('test images for cataract class from',k*n_rest,(k+1)*n_rest)

    # Adding the images in gluacoma validation set by using k*n_rest to (k+1)*n_rest as index values for gluacoma data
set divided in cell 8.
    test_glaucoma= glaucoma[k*n_rest:(k+1)*n_rest]
    print('test images for glaucoma class from',k*n_rest,(k+1)*n_rest)

    # Adding the images in retina disease validation set by using k*n_rest to (k+1)*n_rest as index values for retina
 disease dataset divided in cell 9.
    test_retina= retina_disease[k*n_rest:(k+1)*n_rest]
    print('test images for retina disease class from',k*n_rest,(k+1)*n_rest)

    # Now for train and validation set of Normal images first adding 0 to k*n_normal images and then adding all the im
ages from (k+1)*n_normal till last image.

    train_validation_normal= normal[:k*n_normal]
    train_validation_normal= np.append(train_validation_normal,normal[(k+1)*n_normal:],axis=0)
    print('train_validation images for normal class from 0 to',k*n_normal,'and',(k+1)*n_normal,'to 300')

    # Now for train and validation set of cataract images first adding 0 to k*n_rest images and then adding all the im
ages from (k+1)*n_rest till last image.

    train_validation_cataract= cataract[:k*n_rest]
    train_validation_cataract= np.append(train_validation_cataract,cataract[(k+1)*n_rest:],axis=0)
    print('train_validation images for cataract class from 0 to',k*n_rest,'and',(k+1)*n_rest,'to 100')

    # Now for train and validation set of glaucoma images first adding 0 to k*n_rest images and then adding all the im
ages from (k+1)*n_rest till last image.
    train_validation_glaucoma= glaucoma[:k*n_rest]
    train_validation_glaucoma= np.append(train_validation_glaucoma,glaucoma[(k+1)*n_rest:],axis=0)
    print('train_validation  images for glaucoma class from 0',k*n_rest,'and',(k+1)*n_rest,'to 100')

    # Now for train and validation set of retina disease images first adding 0 to k*n_rest images and then adding all
 the images from (k+1)*n_rest till last image.
    train_validation_retina= retina_disease[:k*n_rest]
    train_validation_retina= np.append(train_validation_retina,retina_disease[(k+1)*n_rest:],axis=0)
    print('train_validation images for retina disease class from 0 to',k*n_rest,'and',(k+1)*n_rest,'to 100')


    # Splitting the train validation datasets in 80:20 ratio which would eventually give us 70% images in train and 2
0% images in validation and 10% in test.
    normal_train, normal_validation                 = train_test_split(train_validation_normal, test_size=0.20, random
_state=14,shuffle=True)
    cataract_train, cataract_validation             = train_test_split(train_validation_cataract, test_size=0.20, rand
om_state=14,shuffle=True)
    glaucoma_train, glaucoma_validation             = train_test_split(train_validation_glaucoma, test_size=0.20, rand
om_state=14,shuffle=True)
    retina_disease_train, retina_disease_validation = train_test_split(train_validation_retina, test_size=0.20, random
_state=14,shuffle=True)

    # Appending all train set images for all classes
    train= np.append(normal_train,cataract_train,axis=0)
    train= np.append(train,glaucoma_train,axis=0)
    train= np.append(train,retina_disease_train,axis=0)

    # Appending all validation set images for all classes
    validation= np.append(normal_validation,cataract_validation,axis=0)
    validation= np.append(validation,glaucoma_validation,axis=0)
    validation= np.append(validation,retina_disease_validation,axis=0)

    # Appending all test set images for all classes
    test= np.append(test_normal,test_cataract,axis=0)
    test= np.append(test,test_glaucoma,axis=0)
    test= np.append(test,test_retina,axis=0)
    # Shuffling the train validation and test set as they are added sequentially.
    random.seed(6)
    np.random.shuffle(train)
    np.random.shuffle(validation)
    np.random.shuffle(test)

    # Passing the train validation test as argument for image_label_split function that return features and labels sep
arated.
```

```
    x_train,y_train,x_val,y_val,x_test,y_test = image_label_split(train,validation,test)

    # Passing the x_Train x_val and x_test as a argument for normalize function that returns the normalized and reshap
ed sets.
    x_train,x_val,x_test = normalize(x_train,x_val,x_test)

    # model building and model compile is done using a model_build_compile().

    model = model_build_compile(k)
    # passing x_train,y_train and x_val,y_val for model.fit
    history = model.fit(x_train,y_train,epochs =50, validation_data = (x_val,y_val))


    train_acc = np.append(train_acc,history.history['accuracy'])
    val_acc = np.append(val_acc,history.history['val_accuracy'])

    train_loss = np.append(train_loss,history.history['loss'])
    val_loss = np.append(val_loss,history.history['val_loss'])

    x,y,z,c = test_pred(x_test,y_test,k)

    CM.append([c])
    test_accuracy.append(x)
    test_specificity.append(y)
    test_sensitivity.append(z)
```

Figure 16: K-Fold Validation Implementation

Figure 16, includes the function call that passes k as an argument to model_build_compile() function that returns the new model built and compiled for each fold. After that, the Model is fitted using the training and validation datasets.

8.1 Train Test Validation images per label check

The below code block can be used to confirm the number of images in the test phase of the model. To check for train and validation need to replace test in the for loop with train and validation.

```
l = []
for i in test:
    if(i[1] == 0):
        l.append("1_normal")

    elif (i[1] == 1):
        l.append("2_cataract")

    elif (i[1] == 2):
        l.append("3_glaucoma")

    else :
        l.append("4_retina_disease")


sns.set_style('darkgrid')
sns.countplot(l)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2012fc78820>
```

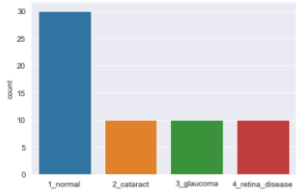

Figure 17: Number of images check in Train Validation and Test

9 Model Evaluation and Visualizations

9.1 Test Predictions, Accuracy, Sensitivity and Specificity

The function defined in this section needs to be executed before section 8.

```
def test_pred(x_val,y_val,k):
    predictions = model.predict(x_val)
    predictions = np.argmax(predictions, axis = -1)

    print('----------------Test accuracy for',k+1,'fold----------------')
    #Confusion matrix, Accuracy, sensitivity and specificity
    cm1 = confusion_matrix(y_val,predictions)
    print('Confusion Matrix : \n', cm1)

    #####from confusion matrix calculate accuracy

    sensitivity_1_normal = (cm1[0,0])/(cm1[0,0]+cm1[0,1]+cm1[0,2]+cm1[0,3])
    #print('Sensitivity_1_normal          : ', sensitivity_1_normal )

    sensitivity_2_cataract = (cm1[1,1])/(cm1[1,0]+cm1[1,1]+cm1[1,2]+cm1[1,3])
    #print('Sensitivity_2_cataract        : ', sensitivity_2_cataract )


    sensitivity_3_glaucoma = (cm1[2,2])/(cm1[2,0]+cm1[2,1]+cm1[2,2]+cm1[2,3])
    #print('Sensitivity_3_glaucoma        : ', sensitivity_3_glaucoma )

    sensitivity_4_retina_disease = (cm1[3,3])/(cm1[3,0]+cm1[3,1]+cm1[3,2]+cm1[3,3])
    #print('Sensitivity_4_retina_disease  : ', sensitivity_4_retina_disease )

    specificity_1_normal = (cm1[1,1]+cm1[1,2]+cm1[1,3]+cm1[2,1]+cm1[2,2]+cm1[2,3]+cm1[3,1]+cm1[3,2]+cm1[3,3])/(cm1[1,0
]+cm1[2,0]+cm1[3,0]+cm1[1,1]+cm1[1,2]+cm1[1,3]+cm1[2,1]+cm1[2,2]+cm1[2,3]+cm1[3,1]+cm1[3,2]+cm1[3,3])
    #print('Specificity : ', specificity_1_normal)

    specificity_2_cataract = (cm1[0,0]+cm1[0,2]+cm1[0,3]+cm1[2,0]+cm1[2,2]+cm1[2,3]+cm1[3,0]+cm1[3,2]+cm1[3,3])/(cm1[0
,1]+cm1[2,1]+cm1[3,1]+cm1[0,0]+cm1[0,2]+cm1[0,3]+cm1[2,0]+cm1[2,2]+cm1[2,3]+cm1[3,0]+cm1[3,2]+cm1[3,3])
    #print('Specificity : ', specificity_2_cataract)

    specificity_3_glaucoma = (cm1[0,0]+cm1[0,1]+cm1[0,3]+cm1[1,0]+cm1[1,1]+cm1[1,3]+cm1[3,0]+cm1[3,1]+cm1[3,3])/(cm1[0
,2]+cm1[1,2]+cm1[3,2]+cm1[0,0]+cm1[0,1]+cm1[0,3]+cm1[1,0]+cm1[1,1]+cm1[1,3]+cm1[3,0]+cm1[3,1]+cm1[3,3])
    #print('Specificity : ', specificity_3_glaucoma)

    specificity_4_retina_disease= (cm1[0,0]+cm1[0,1]+cm1[0,2]+cm1[1,0]+cm1[1,1]+cm1[1,2]+cm1[2,0]+cm1[2,1]+cm1[2,2])/(
cm1[0,3]+cm1[1,3]+cm1[2,3]+cm1[0,0]+cm1[0,1]+cm1[0,2]+cm1[1,0]+cm1[1,1]+cm1[1,2]+cm1[2,0]+cm1[2,1]+cm1[2,2])
    #print('Specificity : ', specificity_4_retina_disease)

    Sensitivity= (sensitivity_1_normal + sensitivity_2_cataract + sensitivity_3_glaucoma + sensitivity_4_retina_diseas
e)/4
    #print(Sensitivity)

    Specificity= (specificity_1_normal + specificity_2_cataract + specificity_3_glaucoma + specificity_4_retina_diseas
e)/4
    #print(Specificity)

    total1=sum(sum(cm1))
    test_accuracy=(cm1[0,0]+cm1[1,1]+cm1[2,2]+cm1[3,3])/total1

    print ('Accuracy    : ', test_accuracy)
    print ('Specificity : ', Specificity)
    print ('Sensitivity : ', Sensitivity)
    print('----------------------------End of',k+1,'Fold-------------------------')
    return test_accuracy,Specificity,Sensitivity,cm1
```

Figure 18: Test Predictions and Confusion Matrix and Sensitivity and Specificity Computation

9.2 Average of All the Evaluation Metrics for Train, Validation and Testing Phase

As the K fold validation generates 5 different results set for each fold, the average of all the evaluation matrices needs to be computed.

```
mean_train_accuracy=np.mean(train_acc)
mean_train_accuracy
```

```
0.8978425928354263
```

```
mean_val_accuracy=np.mean(val_acc)
mean_val_accuracy
```

```
0.49062963223457334
```

```
mean_test_accuracy=np.mean(test_accuracy)
mean_test_accuracy
```

```
0.55
```

Figure 19: Average Accuracy Computation for All Phases

```
mean_train_loss=np.mean(train_loss)
mean_train_loss
```

```
1.2308800502121449
```

```
mean_val_loss=np.mean(val_loss)
mean_val_loss
```

```
13.798260963439942
```

Figure 20: Average Loss for Training and Validation

```
mean_test_specificity= np.mean(test_specificity)
mean_test_specificity
```

```
0.7654947367530492
```

```
mean_test_sensitivity= np.mean(test_sensitivity)
mean_test_sensitivity
```

```
0.4683333333333334
```

Figure 21: Average Sensitivity and Specificity for Test

9.3 Training v/s Validation Accuracy and Loss Plots

```
def plot_print(i,j):
    epochs_range = range(50)

    plt.figure(figsize=(15, 15))
    plt.subplot(2, 2, 1)
    plt.plot(epochs_range, train_acc[i:j], label='Training Accuracy')
    plt.plot(epochs_range, val_acc[i:j], label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(2, 2, 2)
    plt.plot(epochs_range, train_loss[i:j], label='Training Loss')
    plt.plot(epochs_range, val_loss[i:j], label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')

    return plt.show()
```

```
k=1
j=0
for i in range(0,250,50):
    j +=50
    print('Plot for ',k,'cross validation accuracy and loss for Training and Validation phase')
    k +=1
    plot_print(i,j)
```

Figure 22: Training v/s Validation Plots

9.4 Test Phase Confusion Matrices Visualization

```
CM= np.array(CM)
CM.resize(5,4,4)
```

```
def confusionmatrix_vis(i):

        yticklabels=['1_normal', '2_cataract','3_glaucoma','4_retina_disease']
        xticklabels=['1_normal', '2_cataract','3_glaucoma','4_retina_disease']
        plt.figure(figsize=(8, 8))
        hm =sns.heatmap(CM[i], annot=True,annot_kws={"size": 20}, cbar=False,cmap="YlGnBu",yticklabels=yticklabels,xti
cklabels=xticklabels)

        hm.set_xticklabels(hm.get_xticklabels(), rotation=0, fontsize = 12, )
        hm.set_yticklabels(hm.get_yticklabels(), rotation=0, fontsize = 12)

        plt.ylabel("Actual", fontsize = 18)
        plt.xlabel("Predicted",fontsize = 18)

        return plt.show()
```

```
k=1
for i in range(5):
    print('Confusion Matrix for ',k,'Cross Validation Test phase')
    k +=1
    confusionmatrix_vis(i)
```

Figure 23: Confusion Matrices Visualization

9.5 Summarizing the Confusion Matrix for All Folds

```
CM_sum = CM[0]+CM[1]+CM[2]+CM[3]+CM[4]
CM_sum
```

```
array([[107,   0,  25,  18],
       [ 14,  26,   6,   4],
       [ 16,   5,  23,   6],
       [ 28,   7,   6,   9]], dtype=int64)
```

```
yticklabels=['1_normal', '2_cataract','3_glaucoma','4_retina_disease']
xticklabels=['1_normal', '2_cataract','3_glaucoma','4_retina_disease']
plt.figure(figsize=(8, 8))
hm =sns.heatmap(CM_sum, annot=True,annot_kws={"size": 20},fmt='g', cbar=False,cmap="YlGnBu",yticklabels=yticklabels,xt
icklabels=xticklabels)

hm.set_xticklabels(hm.get_xticklabels(), rotation=0, fontsize = 12, )
hm.set_yticklabels(hm.get_yticklabels(), rotation=0, fontsize = 12)

plt.ylabel("Actual", fontsize = 18)
plt.xlabel("Predicted",fontsize = 18)

plt.show()
```

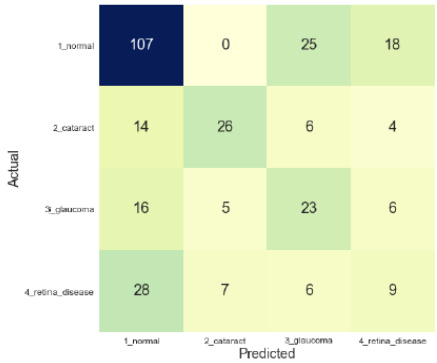

Figure 23: Summarized Confusion Matrix

9.6 Reconfirming the Test Evaluation Results

```
    sensitivity_1_normal = (CM_sum[0,0])/(CM_sum[0,0]+CM_sum[0,1]+CM_sum[0,2]+CM_sum[0,3])
    #print('Sensitivity_1_normal          : ', sensitivity_1_normal )

    sensitivity_2_cataract = (CM_sum[1,1])/(CM_sum[1,0]+CM_sum[1,1]+CM_sum[1,2]+CM_sum[1,3])
    #print('Sensitivity_2_cataract        : ', sensitivity_2_cataract )


    sensitivity_3_glaucoma = (CM_sum[2,2])/(CM_sum[2,0]+CM_sum[2,1]+CM_sum[2,2]+CM_sum[2,3])
    #print('Sensitivity_3_glaucoma        : ', sensitivity_3_glaucoma )

    sensitivity_4_retina_disease = (CM_sum[3,3])/(CM_sum[3,0]+CM_sum[3,1]+CM_sum[3,2]+CM_sum[3,3])
    #print('Sensitivity_4_retina_disease  : ', sensitivity_4_retina_disease )

    specificity_1_normal = (CM_sum[1,1]+CM_sum[1,2]+CM_sum[1,3]+CM_sum[2,1]+CM_sum[2,2]+CM_sum[2,3]+CM_sum[3,1]+CM_sum
[3,2]+CM_sum[3,3])/(CM_sum[1,0]+CM_sum[2,0]+CM_sum[3,0]+CM_sum[1,1]+CM_sum[1,2]+CM_sum[1,3]+CM_sum[2,1]+CM_sum[2,2]+CM
_sum[2,3]+CM_sum[3,1]+CM_sum[3,2]+CM_sum[3,3])
    #print('Specificity : ', specificity_1_normal)

    specificity_2_cataract = (CM_sum[0,0]+CM_sum[0,2]+CM_sum[0,3]+CM_sum[2,0]+CM_sum[2,2]+CM_sum[2,3]+CM_sum[3,0]+CM_s
um[3,2]+CM_sum[3,3])/(CM_sum[0,1]+CM_sum[2,1]+CM_sum[3,1]+CM_sum[0,0]+CM_sum[0,2]+CM_sum[0,3]+CM_sum[2,0]+CM_sum[2,2]+
CM_sum[2,3]+CM_sum[3,0]+CM_sum[3,2]+CM_sum[3,3])
    #print('Specificity : ', specificity_2_cataract)

    specificity_3_glaucoma = (CM_sum[0,0]+CM_sum[0,1]+CM_sum[0,3]+CM_sum[1,0]+CM_sum[1,1]+CM_sum[1,3]+CM_sum[3,0]+CM_s
um[3,1]+CM_sum[3,3])/(CM_sum[0,2]+CM_sum[1,2]+CM_sum[3,2]+CM_sum[0,0]+CM_sum[0,1]+CM_sum[0,3]+CM_sum[1,0]+CM_sum[1,1]+
CM_sum[1,3]+CM_sum[3,0]+CM_sum[3,1]+CM_sum[3,3])
    #print('Specificity : ', specificity_3_glaucoma)

    specificity_4_retina_disease= (CM_sum[0,0]+CM_sum[0,1]+CM_sum[0,2]+CM_sum[1,0]+CM_sum[1,1]+CM_sum[1,2]+CM_sum[2,0]
+CM_sum[2,1]+CM_sum[2,2])/(CM_sum[0,3]+CM_sum[1,3]+CM_sum[2,3]+CM_sum[0,0]+CM_sum[0,1]+CM_sum[0,2]+CM_sum[1,0]+CM_sum[
1,1]+CM_sum[1,2]+CM_sum[2,0]+CM_sum[2,1]+CM_sum[2,2])
    #print('Specificity : ', specificity_4_retina_disease)

    Sensitivity= (sensitivity_1_normal + sensitivity_2_cataract + sensitivity_3_glaucoma + sensitivity_4_retina_diseas
e)/4
    #print(Sensitivity)

    Specificity= (specificity_1_normal + specificity_2_cataract + specificity_3_glaucoma + specificity_4_retina_diseas
e)/4
    #print(Specificity)

    total1=sum(sum(CM_sum))
    test_accuracy=(CM_sum[0,0]+CM_sum[1,1]+CM_sum[2,2]+CM_sum[3,3])/total1

    print ('Accuracy    : ', test_accuracy)
    print ('Specificity : ', Specificity)
    print ('Sensitivity : ', Sensitivity)
```

Figure 24: Test Results computed using Overall Confusion Matrix

References

Data Source:
https://www.kaggle.com/jr2ngb/cataractdataset

Code Reference for Confusion Matrix, Accuracy, Sensitivity and Specificity Computation:
https://statinfer.com/204-4-2-calculating-sensitivity-and-specificity-in-python/

https://towardsdatascience.com/visual-guide-to-the-confusion-matrix-bb63730c8eba