

Configuration Manual

MSc Research Project
MSc Data Analytics

Shahrukh
Student ID: 23135379

School of Computing
National College of Ireland

Supervisor: Dr Chirstian Horn

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Shahrukh
Student ID:	x23135379
Programme:	MSc Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Dr Chirstian Horn
Submission Due Date:	12/08/2024
Project Title:	Multi-Class Eye Disorder Classification using CNN
Word Count:	1405
Page Count:	16

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:	Shahrukh
Date:	16th September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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

Shahrukh
23135379

1 Introduction

This document provides all the necessary information to replicate this research and the results of this research in a personal environment. The report also mentions the hardware and software requirements along with the screenshots of importing necessary libraries with the Exploratory Data Analysis of the dataset. All the preprocessing steps that are performed on the dataset, and model performance on every step, when applied to before and after applying preprocessing.

The next step gives the information about the environment that is used for this research. The Data collection process is mentioned in section 3 of this report. The fourth section is about Data Exportation. The final section explains Model Execution.

2 Enviornmental Setup

The requirements related to the software and hardware are mentioned in this section

2.1 Hardware Requirements

This research is conducted on Intel core m3, the laptop is of 7th generation CPU with the RAM of 8GB . The installed window on the system is Windows 10 Home

[View basic information about your computer](#)

Windows edition

Windows 10 Home

© Microsoft Corporation. All rights reserved.



System

Processor: Intel(R) Core(TM) m3-7Y30 CPU @ 1.00GHz 1.61 GHz

Installed memory (RAM): 8.00 GB

System type: 64-bit Operating System, x64-based processor

Pen and Touch: Touch Support with 10 Touch Points

Computer name, domain, and workgroup settings

Computer name: DESKTOP-CL927FO

Full computer name: DESKTOP-CL927FO

Computer description:

Workgroup: WORKGROUP

[Change settings](#)

Windows activation

Windows is not activated. [Read the Microsoft Software License Terms](#)

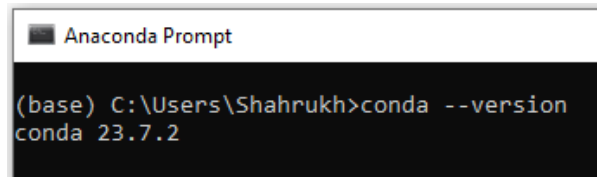
Product ID: 00326-30000-00001-AA300

[Activate Windows](#)

Figure 1: Hardware Requirements

2.2 Software Requirements

- Anaconda Jupyter Notebook



```
Anaconda Prompt

(base) C:\Users\Shahrukh>conda --version
conda 23.7.2
```

Figure 2: Anaconda Version

- Python version 3.11.4
- Keras Version 3.4.1

3 Data Collection

The data set is available on the open-source repository Kaggle. The dataset contains 6392 fundus images of different eye disorders. The diseases are divided into 8 different categories. The Normal class and the Diabetes class are more prominent class in the dataset . The link of the dataset is <https://www.kaggle.com/datasets/andrewmvd/ocular-disease-recognition-odir5K>

4 Data Exportation

4.1 Importing Necessary Libraries

In this section necessary libraries that are used to perform several tasks are imported in Jupyter Notebook.

```
import zipfile
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os
import numpy as np
import random
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from collections import Counter
import shutil
from PIL import Image
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from matplotlib import pyplot as plt
from tensorflow.keras.preprocessing.image import img_to_array, load_img
import pickle
```

Figure 3: Imported Libraries

4.2 Extracting files from dataset

The dataset was present in zip form so all the files in the dataset are imported into the current directory

```
import zipfile
with zipfile.ZipFile('archive (8).zip', 'r') as zip_ref:
    zip_ref.extractall() # Extract all files to the current directory
```

Figure 4: Extracting files into current repository

The dataset contains .csv file of the data frame where all the information of the patients along with the label of images, image names, image path, and the diagnosis keyword is store is extracted and stored in df

```
import pandas as pd

df = pd.read_csv('full_df.csv')
print(df.head())
```

Figure 5: Dataframe

Now the next step is to perfrom the exploratory data analysis on the dataset

4.3 Exploratory Data Analysis

In this step different columns of the data frame are explored, moreover, the count of the images of each class is also done in this stage.

4.3.1 Check for Missing Values Indentification

```
import pandas as pd

df = pd.read_csv('full_df.csv')

# Check for missing values
print(df.isnull().sum())
```

Figure 6: Code to Identify Missing Values

No missing value is identified in the data frame

4.3.2 Check for Gender Distribution

```
# Count the number of male and female patients
patient_counts = df['Patient Sex'].value_counts()

# Create a bar chart
patient_counts.plot(kind='bar', colormap='Set1')
plt.xlabel('Patient Sex')
plt.ylabel('Count')
plt.title('Distribution of Patients by Gender')
plt.xticks(rotation=0)

# Customize the plot further
plt.tight_layout()

plt.show()
```

Figure 7: Gender Distribution Code

4.3.3 Check number of Images in Each Class

The dataset was highly imbalanced because the number of images in N class and the D class are much higher as compare to the other classes

```
label_counts = Counter()
for labels in df['labels']:
    for label in eval(labels): # Use eval to convert string representation of list to actual list
        label_counts[label] += 1
# Print the number of images in each class
print("Number of images in each class:")
for label, count in label_counts.items():
    print(f"{label}: {count}")
```

Figure 8: Code for Counting Images

The results of the above code indicates that two classes have high number of images as compare to other classes

```
Number of images in each class:
N: 2873
D: 1608
O: 708
M: 232
H: 128
C: 293
A: 266
G: 284
```

Figure 9: Output

4.4 Data Preprocessing

Data preprocessing is done to increase the number of images of M, C, A, and G classes through data augmentation. In the data augmentation, these class images were first given a horizontal flip and then a vertical flip. The process followed for this purpose is to take the image of the class give it a flip and then store it in the new directory. Moreover, operations on images, labels, and keywords are also performed

4.4.1 Horizontal Flip

Classes that are selected for the horizontal flip are M, C, A , G , for performing this operation these specific names will be searched in the label column, and after it images all the images are selected from each class of the data frame. A horizontal flip to the images will be given and results will be then saved into the augmentation directory

```

# Load the DataFrame
df = pd.read_csv('full_df.csv')

# Define the file path to the actual images directory
actual_image_dir = 'preprocessed_images'
augmentation_dir = 'new_augmentation/'

# Delete the augmentation directory if it already exists
if os.path.exists(augmentation_dir):
    shutil.rmtree(augmentation_dir)

# Create the augmentation directory
os.makedirs(augmentation_dir)

# Create a new DataFrame for augmented images
augmented_data = []

# Initialize a counter for augmented images by class
class_counts = Counter()

# Filter the DataFrame for the specified classes
specified_classes = {'M', 'C', 'A', 'G'}
filtered_df = df[df['labels'].apply(lambda x: any(label in specified_classes for label in eval(x)))]

# Perform horizontal flip and save the images
for index, row in filtered_df.iterrows():
    filename = row['filename']

    # Load the image
    image_path = os.path.join(actual_image_dir, filename)
    if os.path.exists(image_path):
        image = Image.open(image_path)

    # Perform horizontal flip
    flipped_image = image.transpose(Image.FLIP_LEFT_RIGHT)

    # Save the flipped image with the new naming convention
    flipped_image_filename = f"H-Flip_{filename}"
    flipped_image_path = os.path.join(augmentation_dir, flipped_image_filename)
    flipped_image.save(flipped_image_path)

    # Append the original row data with updated filename to the new DataFrame
    augmented_row = row.copy()
    augmented_row['filename'] = flipped_image_filename
    augmented_data.append(augmented_row)

    # Count the classes in the labels
    for label in eval(row['labels']):
        if label in specified_classes:
            class_counts[label] += 1

# Create a new DataFrame from the augmented data
augmented_df = pd.DataFrame(augmented_data)

# Save the augmented DataFrame to a new CSV file
augmented_df.to_csv('new_augmented_data.csv', index=False)

```

Figure 10: Code for Horizontal Flip

4.4.2 Vertical Flip

The next step is to perform the vertical flip, in the vertical flip same classes for this purpose images are selected from these classes and then after giving the vertical flip to them, images are saved into the same directory

```

# Load the existing augmented DataFrame
augmented_df = pd.read_csv('new_augmented_data.csv')

# Define the file path to the actual images directory
actual_image_dir = 'preprocessed_images'
augmentation_dir = 'new_augmentation/'

# Initialize a counter for augmented images by class
class_counts = Counter()

# Filter the DataFrame for the specified classes
specified_classes = {'M', 'C', 'A', 'G'}
filtered_df = df[df['labels'].apply(lambda x: any(label in specified_classes for label in eval(x)))]

# List to store new rows for the augmented DataFrame
new_rows = []

# Perform vertical flip and save the images
for index, row in filtered_df.iterrows():
    filename = row['filename']

    # Load the image
    image_path = os.path.join(actual_image_dir, filename)
    if os.path.exists(image_path):
        image = Image.open(image_path)

        # Perform vertical flip
        flipped_image = image.transpose(Image.FLIP_TOP_BOTTOM)

        # Save the flipped image with the new naming convention
        flipped_image_filename = f"V-Flip_{filename}"
        flipped_image_path = os.path.join(augmentation_dir, flipped_image_filename)
        flipped_image.save(flipped_image_path)

        # Prepare the augmented row
        augmented_row = row.copy()
        augmented_row['filename'] = flipped_image_filename
        new_rows.append(augmented_row)

        # Count the classes in the labels
        for label in eval(row['labels']):
            if label in specified_classes:
                class_counts[label] += 1

        # Print status
        # print(f"Processed and saved: {flipped_image_filename}")

# Convert new rows to a DataFrame
new_rows_df = pd.DataFrame(new_rows)

# Concatenate the new rows to the existing DataFrame
augmented_df = pd.concat([augmented_df, new_rows_df], ignore_index=True)

# Save the updated DataFrame back to the same CSV file
augmented_df.to_csv('new_augmented_data.csv', index=False)

# Print class counts after augmentation
print("\nClass counts after vertical augmentation:")
for label, count in class_counts.items():
    print(f"{label}: {count}")

print("\nVertical data augmentation complete and appended to 'new_augmented_data.csv'.")

```

Figure 11: Code for Vertical Flip

4.4.3 Image Rotation

One class is added in the rotation of images with the other classes that were used in the vertical and horizontal flip tests, images are rotated to 10 degrees and then saved the images into same directory


```

# Load the original DataFrame
df = pd.read_csv('full_df.csv')

# Load the existing augmented DataFrame
augmented_df = pd.read_csv('new_augmented_data.csv')

# Define the file path to the actual images directory
actual_image_dir = 'preprocessed_images'
new_augmentation_dir = 'new_augmentation/'

# Initialize a counter for augmented images by class
class_counts_before = Counter()
class_counts_after = Counter()

# Filter the DataFrame for the specified classes
specified_classes = {'O', 'M', 'C', 'A', 'G'}
filtered_df = df[df['labels'].apply(lambda x: any(label in specified_classes for label in eval(x)))]

# Count classes before augmentation
for labels in filtered_df['labels']:
    for label in eval(labels):
        if label in specified_classes:
            class_counts_before[label] += 1

# Print class counts before augmentation
print("Class counts before augmentation:")
for label, count in class_counts_before.items():
    print(f"{label}: {count}")

# List to store new rows for the augmented DataFrame
new_rows = []

# Perform slight rotation and save the images
rotation_angle = 10 # Rotation by 10 degrees
for index, row in filtered_df.iterrows():
    filename = row['filename']

    # Load the image
    image_path = os.path.join(actual_image_dir, filename)
    if os.path.exists(image_path):
        image = Image.open(image_path)

        # Perform rotation
        rotated_image = image.rotate(rotation_angle)

        # Save the rotated image with the new naming convention
        rotated_image_filename = f"Rot{rotation_angle}_{filename}"
        rotated_image_path = os.path.join(new_augmentation_dir, rotated_image_filename)
        rotated_image.save(rotated_image_path)

        # Prepare the augmented row
        augmented_row = row.copy()
        augmented_row['filename'] = rotated_image_filename
        new_rows.append(augmented_row)

        # Count the classes in the labels
        for label in eval(row['labels']):
            if label in specified_classes:
                class_counts_after[label] += 1

# Convert new rows to a DataFrame
new_rows_df = pd.DataFrame(new_rows)

# Concatenate the new rows to the existing DataFrame
augmented_df = pd.concat([augmented_df, new_rows_df], ignore_index=True)

# Save the updated DataFrame back to the same CSV file
augmented_df.to_csv('new_augmented_data.csv', index=False)

# Print class counts after augmentation
print("\nClass counts after rotation augmentation:")
for label, count in class_counts_after.items():
    print(f"{label}: {count}")

```

Figure 12: Image Rotation

4.4.4 Defining Image Size and Normalizing the Image pixels values

As the image sizes of 150x150 and 224,224 are used in the preprocessing step how to perform the normalization and how to select the image size the following code is utilized.

4.4.5 Operation on Label

Labels are converted into one hot-encoded labels by this code

```

# Define image size and categories
image_size = (150, 150)
categories = ['N', 'D', 'G', 'C', 'A', 'M', 'O'] # Exclude 'H'

# Create a mapping from category to index
category_to_index = {category: index for index, category in enumerate(categories)}

# Prepare lists to store images and labels
images = []
labels = []

# Iterate through the filtered DataFrame
for index, row in filtered_df.iterrows():
    label_list = row['labels'].strip('[]').replace("'", "").split(',')
    image_path = row['filepath']
    image = load_img(image_path, target_size=image_size)
    image = img_to_array(image) / 255.0

    # Handle multi-label cases
    for label in label_list:
        if label in category_to_index:
            images.append(image)
            labels.append(category_to_index[label])

```

Figure 13: Image Normalization and Resizing

```

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# One-hot encode the labels
labels = to_categorical(labels, num_classes=len(categories))

```

Figure 14: Encoded Label

4.4.6 Down Sampling of Diabetic, Normal class

While training the model the Diabetic and Normal classes were down sampled after the horizontal and vertical flip operation. Figure 16 demonstrates this step

```

# Define the criteria
class_counts = {
    'C': original_df[original_df['labels'].apply(lambda x: 'C' in eval(x))],
    'M': original_df[original_df['labels'].apply(lambda x: 'M' in eval(x))],
    'A': original_df[original_df['labels'].apply(lambda x: 'A' in eval(x))],
    'G': original_df[original_df['labels'].apply(lambda x: 'G' in eval(x))],
    'N': original_df[original_df['labels'].apply(lambda x: 'N' in eval(x))].sample(frac=1/3, random_state=1),
    'D': original_df[original_df['labels'].apply(lambda x: 'D' in eval(x))].sample(frac=0.5, random_state=1)
}

```

Figure 15: Down Sampling N,D after Flips

Down sampling is also performed later on after the rotation operation of the N class. That number of images after this operation are used in the final model

```

# Define the criteria for selecting original images
class_counts = {
    'C': original_df[original_df['labels'].apply(lambda x: 'C' in eval(x))],
    'M': original_df[original_df['labels'].apply(lambda x: 'M' in eval(x))],
    'A': original_df[original_df['labels'].apply(lambda x: 'A' in eval(x))],
    'G': original_df[original_df['labels'].apply(lambda x: 'G' in eval(x))],
    'N': original_df[original_df['labels'].apply(lambda x: 'N' in eval(x))].sample(frac=2/3, random_state=1),
    'D': original_df[original_df['labels'].apply(lambda x: 'D' in eval(x))],
    'O': original_df[original_df['labels'].apply(lambda x: 'O' in eval(x))] # All of 0
}

```

Figure 16: Down Sampling N for final execution

5 Model Execution

5.1 200 Images Each Class (All diseases)

Firstly 200 images are taken to analyze the results on all classes. How the sample size is taken from all the classes is mentioned in Figure 17. As the H class contains less number

of images the total number of images of this class will be included is 128 images

```
class_counts = 200 # Number of samples per class

# Dictionary containing full Label names
label_names = {
    'N': 'Normal',
    'D': 'Diabetes',
    'G': 'Glaucoma',
    'C': 'Cataract',
    'A': 'Age-related Macular Degeneration',
    'H': 'Hypertension',
    'M': 'Pathological Myopia',
    'O': 'Other Diseases'
}

sampled_dfs = []
for label in label_names.keys():
    class_df = df[df['labels'] == label].apply(lambda x: label in x)
    sampled_class_df = class_df.sample(n=min(class_counts, len(class_df)), random_state=42)
    sampled_dfs.append(sampled_class_df)

# Concatenate sampled dataframes
sampled_df = pd.concat(sampled_dfs, ignore_index=True)

# Verify the sample
print(sampled_df.head())
print(sampled_df.shape)
```

Figure 17: Taking a sample of 200 Images

Data is then split into 60, 20, and 20 ratios for training, evaluation, and testing purposes. The CNN model defined in Figure 18 is then trained on the dataset

```
# Define the CNN model
def create_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Figure 18: CNN model

The model is then compiled

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

# Print the model summary
model.summary()
```

Figure 19: Model Compilation

The accuracy value for the testing is printed

```
# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

Figure 20: Accuracy

The graphs for visualization of training and validation accuracy and training and validation loss is then created

```

# Plot training & validation accuracy values
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper left')

plt.tight_layout()
plt.show()

```

Figure 21: Graphically showing Model accuracy, Loss

Now the confusion matrix is created by using the below code and the results of the confusion matrix are also shown in Figure 23

```

y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Compute the confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)

# Visualize the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=categories, yticklabels=categories)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

```

Figure 22: Confusion Matrix code

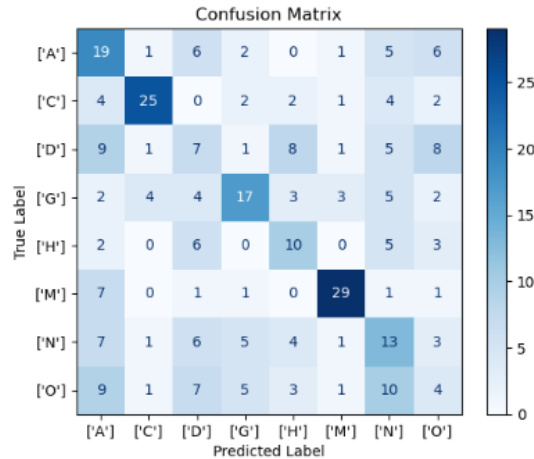


Figure 23: Confusion Matrix Result

5.2 200 Images without H

For doing this process most parts of the code will remain the same as above only from the data frame label H class will be omitted, so that when the images are extracted from the sample of 200 images of each class it will skip the H class. Figure 24, 25 is demonstrate how the class H will be filtered from sample dataframe

Skipping the H class

```
import pandas as pd

# Load your DataFrame
df = pd.read_csv('sampled_200_images.csv')

# Filter out rows where 'Labels' column contains 'H'
filtered_df = df[~df['Labels'].str.contains('H')]

# Verify the filtered DataFrame
print(filtered_df['Labels'].value_counts())
```

Figure 24: Skipping Hypertension Class

```
# Define image size and categories
image_size = (150, 150)
categories = ['N', 'D', 'G', 'C', 'A', 'M', 'O'] # Exclude 'H'

# Create a mapping from category to index
category_to_index = {category: index for index, category in enumerate(categories)}

# Prepare lists to store images and labels
images = []
labels = []

# Iterate through the filtered DataFrame
for index, row in filtered_df.iterrows():
    label_list = row['Labels'].strip('[]').replace("'", "").split(',')
    image_path = row['filepath']
    image = load_img(image_path, target_size=image_size)
    image = img_to_array(image) / 255.0

    # Handle multi-label cases
    for label in label_list:
        if label in category_to_index:
            images.append(image)
            labels.append(category_to_index[label])

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# One-hot encode the labels
labels = to_categorical(labels, num_classes=len(categories))
```

Figure 25: Skipping Hypertension Class

The same model will then be executed on the rest of the classes the results in the form of confusion matrix are mentioned in Figure 26

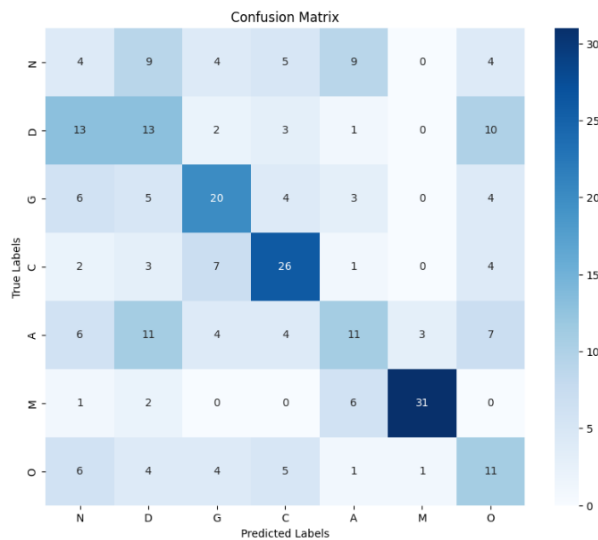


Figure 26: Confusion Matrix

5.3 200 Images without H, O Classes

The model is trained by skipping both of these classes

```

# Define image size and categories
image_size = (150, 150)
categories = ['N', 'D', 'G', 'C', 'A', 'M'] # Exclude 'H' and 'O'

# Create a mapping from category to index
category_to_index = {category: index for index, category in enumerate(categories)}

# Prepare lists to store images and labels
images = []
labels = []

# Iterate through the filtered DataFrame
for index, row in filtered_df.iterrows():
    label_list = row['labels'].strip('[]').replace("'", "").split(',')
    image_path = row['filepath']
    image = load_img(image_path, target_size=image_size)
    image = img_to_array(image) / 255.0

    # Handle multi-label cases
    for label in label_list:
        if label in category_to_index:
            images.append(image)
            labels.append(category_to_index[label])

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

```

Figure 27: Skipping H, O Class

the rest of the logic in the code will remain the same

5.4 Considering whole dataset

The model that is used in this process is described in Figure 18 the criteria used is to first take one-third of images of N class and half of the images of D class Figure 15 and Figure 16 demonstrate this step how these classes are downsampled. In Figure 16 in the final run where all the classes are included, downsampling is only performed on the Normal class.

```

# Load the filtered dataset
filtered_df = pd.read_csv('new_filtered_images.csv')

# Define directories
ORIGINAL_IMAGE_DIR = 'preprocessed_images'
AUGMENTATION_DIR = 'new_augmentation/'

# Define the desired image size
IMG_SIZE = (224, 224)

# Prepare images and labels
images = []
labels = []

# Load images from the directories specified in the filtered dataset
for _, row in filtered_df.iterrows():
    img_path = os.path.join(ORIGINAL_IMAGE_DIR, row['filename'])
    if not os.path.exists(img_path):
        img_path = os.path.join(AUGMENTATION_DIR, row['filename'])

    if os.path.exists(img_path):
        img = Image.open(img_path).resize(IMG_SIZE)
        images.append(np.array(img))
        labels.append(eval(row['labels']))

# Convert to numpy arrays
X = np.array(images)
y = np.array(labels)

```

Figure 28: Without O Class

```

# One-hot encode the labels
unique_labels = sorted(set(label for sublist in y for label in sublist))
label_to_index = {label: index for index, label in enumerate(unique_labels)}
y_encoded = np.array([[label_to_index[label] for label in labels] for labels in y])
y_categorical = to_categorical(y_encoded, num_classes=len(unique_labels))

# Split the dataset
train_df, temp_df = train_test_split(filtered_df, test_size=0.4, random_state=42)
val_df, test_df = train_test_split(temp_df, test_size=0.5, random_state=42)

# Print the total number of images for training, validation, and testing
print(f'Total images for training: {len(X_train)}')
print(f'Total images for validation: {len(X_val)}')
print(f'Total images for testing: {len(X_test)}')

# Verify the counts of images in each class for each set
def count_classes(y_data, unique_labels):
    counts = {label: 0 for label in unique_labels}
    for one_hot in y_data:
        for i, value in enumerate(one_hot):
            if value == 1:
                label = unique_labels[i]
                counts[label] += 1
    return counts

train_counts = count_classes(y_train, unique_labels)
val_counts = count_classes(y_val, unique_labels)
test_counts = count_classes(y_test, unique_labels)

```

Figure 29: Without O Class

```

# Set random seed for reproducibility
SEED = 42
np.random.seed(SEED)
tf.random.set_seed(SEED)
random.seed(SEED)

# Normalize the data
X_train = X_train / 255.0
X_val = X_val / 255.0
X_test = X_test / 255.0

# Build the model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(unique_labels), activation='softmax')
])

model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Create ImageDataGenerators without augmentation
train_datagen = ImageDataGenerator()
val_datagen = ImageDataGenerator()

# Create data generators
train_generator = train_datagen.flow(X_train, y_train, batch_size=32)
val_generator = val_datagen.flow(X_val, y_val, batch_size=32)

```

Figure 30: CNN without O

```

# Fit the model
history = model.fit(
    train_generator,
    steps_per_epoch=len(X_train) // 32,
    validation_data=val_generator,
    validation_steps=len(X_val) // 32,
    epochs=30
)

# Evaluate the model on the test set
test_generator = val_datagen.flow(X_test, y_test, batch_size=32)
test_loss, test_acc = model.evaluate(test_generator, steps=len(X_test) // 32)
print(f'Test accuracy: {test_acc}')

```

Figure 31: CNN without O

Figure 32 represents results on 30 epochs

0.6351
 31/31 [=====] - 68s 605ms/step - loss: 1.0400 - accuracy: 0.6331
 Test accuracy: 0.6330645084381104

Figure 32: Test accuracy

If the O class is added in the same code then the test accuracy reaches 59% the results of the added O class are in the confusion matrix of Figure 33

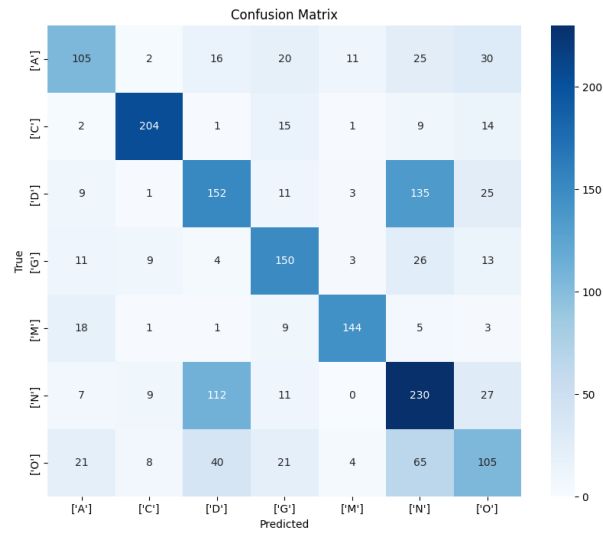


Figure 33: Confusion Matrix with O

For executing the model on keywords first conditions would be matched on the diagnosis keywords column and then images are extracted based on the conditions present in the keyword column.

```
def contains_keyword(keyword, phrase):
    # Return 1 if the keyword is in the phrase, otherwise return 0
    return 1 if keyword in phrase else 0

def check_condition(keywords, condition):
    # Check if a specific condition is present in the diagnostic keywords
    return keywords.apply(lambda x: contains_keyword(condition, x))

def extract_image_ids(data, condition_flag, eye, label, sample_size=None):
    # Extract image IDs based on the condition and label
    if sample_size:
        return data.loc[(data.C == label) & (data['(eye)-Diagnostic Keywords'] == condition_flag)][f'({eye})-Fundus']
    return data.loc[(data.C == label) & (condition_flag == 1)][f'({eye})-Fundus'].values

def Merge_images(*arrays):
    # Concatenate multiple arrays of image IDs
    return np.concatenate(arrays, axis=0)

def process_dataset(data):
    # Define conditions and corresponding column names
    conditions = [
        ("cataract", "left_cataract", "right_cataract"),
        ("non proliferative retinopathy", "left_npr", "right_npr"),
        ("glaucoma", "left_glaucoma", "right_glaucoma"),
        ("myopia", "left_myopia", "right_myopia"),
        ("macular degeneration", "left_md", "right_md"),
        ("drusen", "left_drusen", "right_drusen")
    ]

    # Create columns to indicate the presence of each condition in the left and right eyes
    for cond, left_col, right_col in conditions:
        data[left_col] = check_condition(data['Left-Diagnostic Keywords'], cond)
        data[right_col] = check_condition(data['Right-Diagnostic Keywords'], cond)

    # Extract image IDs for various conditions
    left_cataract_ids = extract_image_ids(data, data.left_cataract, "Left", 1)
    right_cataract_ids = extract_image_ids(data, data.right_cataract, "Right", 1)
```

Figure 34: Processing on keywords


```

left_diabetes_ids = extract_image_ids(data, data.left_npr, "Left", 0)
right_diabetes_ids = extract_image_ids(data, data.right_npr, "Right", 0)

left_glaucoma_ids = extract_image_ids(data, data.left_glaucoma, "Left", 0)
right_glaucoma_ids = extract_image_ids(data, data.right_glaucoma, "Right", 0)

left_myopia_ids = extract_image_ids(data, data.left_myopia, "Left", 0)
right_myopia_ids = extract_image_ids(data, data.right_myopia, "Right", 0)

left_md_ids = extract_image_ids(data, data.left_md, "Left", 0)
right_md_ids = extract_image_ids(data, data.right_md, "Right", 0)

left_drusen_ids = extract_image_ids(data, data.left_drusen, "Left", 0)
right_drusen_ids = extract_image_ids(data, data.right_drusen, "Right", 0)

# Concatenate image IDs for the left and right eyes for each condition
normal_images = merge_images(left_normal_ids, right_normal_ids)
cataract_images = merge_images(left_cataract_ids, right_cataract_ids)
diabetes_images = merge_images(left_diabetes_ids, right_diabetes_ids)
glaucoma_images = merge_images(left_glaucoma_ids, right_glaucoma_ids)
myopia_images = merge_images(left_myopia_ids, right_myopia_ids)
md_images = merge_images(left_md_ids, right_md_ids)
drusen_images = merge_images(left_drusen_ids, right_drusen_ids)

```

Figure 35: Processing on keywords

```

combined_dataset = []

# Process each category of images and append to the combined dataset
print("Generating datasets for each category...")

normal_dataset = generate_dataset(normal_images, 0)
combined_dataset += normal_dataset

cataract_dataset = generate_dataset(cataract_images, 1)
combined_dataset += cataract_dataset

diabetes_dataset = generate_dataset(diabetes_images, 2)
combined_dataset += diabetes_dataset

glaucoma_dataset = generate_dataset(glaucoma_images, 3)
combined_dataset += glaucoma_dataset

myopia_dataset = generate_dataset(myopia_images, 4)
combined_dataset += myopia_dataset

macular_degeneration_dataset = generate_dataset(md_images, 5)
combined_dataset += macular_degeneration_dataset

drusen_dataset = generate_dataset(drusen_images, 6)
combined_dataset += drusen_dataset

```

Figure 36: Processing on keywords

```

# Define image size
image_size = 224

# Convert the combined dataset into predictors (images) and targets (labels)
predictor_images = np.array([item[0] for item in combined_dataset]).reshape(-1, image_size, image_size, 3)
target_labels = np.array([item[1] for item in combined_dataset])

# Split the dataset into training, temporary (testing + validation), and then into testing and validation sets
x_train, x_temp, y_train, y_temp = train_test_split(predictor_images, target_labels, test_size=0.3, random_state=42)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5, random_state=42)

# Convert target labels to categorical format
y_train_categorical = to_categorical(y_train, num_classes=7)
y_val_categorical = to_categorical(y_val, num_classes=7)
y_test_categorical = to_categorical(y_test, num_classes=7)

# Print the shapes of the datasets to verify the splits
print(f"x_train shape: {x_train.shape}, y_train_categorical shape: {y_train_categorical.shape}")
print(f"x_val shape: {x_val.shape}, y_val_categorical shape: {y_val_categorical.shape}")
print(f"x_test shape: {x_test.shape}, y_test_categorical shape: {y_test_categorical.shape}")

```

Figure 37: Training, Testing, Validation

```

# Define the CNN model
def create_cnn_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])
    return model

# Create the model
input_shape = (image_size, image_size, 3)
num_classes = 7
model = create_cnn_model(input_shape, num_classes)

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(
    x_train, y_train_categorical,
    epochs=20,
    validation_data=(x_val, y_val_categorical)
)

```

Figure 38: Training CNN Model

```
56/56 ————— 31s 556ms/step - accuracy: 0.8100 - loss: 0.4880  
Test accuracy: 0.8185890316963196  
56/56 ————— 32s 565ms/step
```

Figure 39: Test Accuracy (20 Epochs)

References

Source of Dataset (Website Link): <https://www.kaggle.com/datasets/andrewmvd/ocular-disease-recognition-odir5k>