Report

1.Title of the project: Ocular Disease recognition (ODR) using Fundus Images

2.Team members:

Suraj Rajeshwar Bhute [21MM61R03]- Implementation of VGG16 model from scratch and through Transfer Learning

Abhishek Naresh Selokar [21MM62R10] - Image Preprocessing and applying Machine Learning models

3. Introduction

At present at least 2.2 billion people around the world have a vision impairment, of whom at least 1 billion have a vision impairment that could have been prevented or is yet to be addressed. The world faces considerable challenges in terms of eye care, including inequalities in the coverage and quality of prevention, treatment and rehabilitation services; a shortage of trained eye care service providers; and poor integration of eye care services into health systems, among others. In ophthalmology, early fundus screening is an economic and effective way to prevent blindness caused by ophthalmic diseases. Clinically, due to the lack of medical resources, manual diagnosis is time-consuming and may delay the condition. With the development of deep learning, some researches on ophthalmic diseases have achieved good results, however, most of them are just based on one disease. Early eye diagnosis is an economical and effective way to prevent blindness caused by diabetes, glaucoma, cataract, age-related macular degeneration (AMD), and many other diseases. Prompt and automatic diagnosis is important and urgent in reducing the activity of the ophthalmologist and preventing visual impairment in patients. Computer vision and deep learning can automatically detect ocular diseases after providing high quality fundus medical images.

A. RETINA

The retina is a thin, soft, transparent sheet of tissue derived from the neuroectoderm. It is the light-sensing tissue that stays back of the eye.

Retinography is a noninvasive diagnostic test that diagnoses eye problems such as Diabetic Retinopathy, Glaucoma,Cataract,Diabetic Macular Edema, Hypertensive Retinopathy and Age-related Macular Depression. Retinographies are images that allow observing the retina.

The fundus structures examined are:

∗ Retinal parenchyma: The retina is an almost transparent membrane. The red color it provides is due to the hue of the Retinal Pigment Epithelium (RPE) and can vary depending on race, age, and skin color. It loses its luster over the years.

∗ Papilla or Optic Disc (OD): the visible part of the optic nerve that is rounded or oval in a vertical direction, 1.5 mm in diameter; The color white-pink, with a central white spot associated with excavation known as the Optical Cup (OC) and has a variable size, not exceeding 30% of the OD.

∗ Retinal vessels: in the center of the optic disc is the vascular bundle composed of the central retinal artery and vein, divided into superior and inferior temporal veins and arteries, and superior and inferior nasal arteries; the branches of the retinal vessels are approaching in all directions, but do not reach the fovea; the vein has a dark red wine color and wavier in trajectory with an artery-vein caliber ratio of 2/3.
∗ Macula: located about two meters (OD) from papilla.

The Structure can be seen in Figure 1.

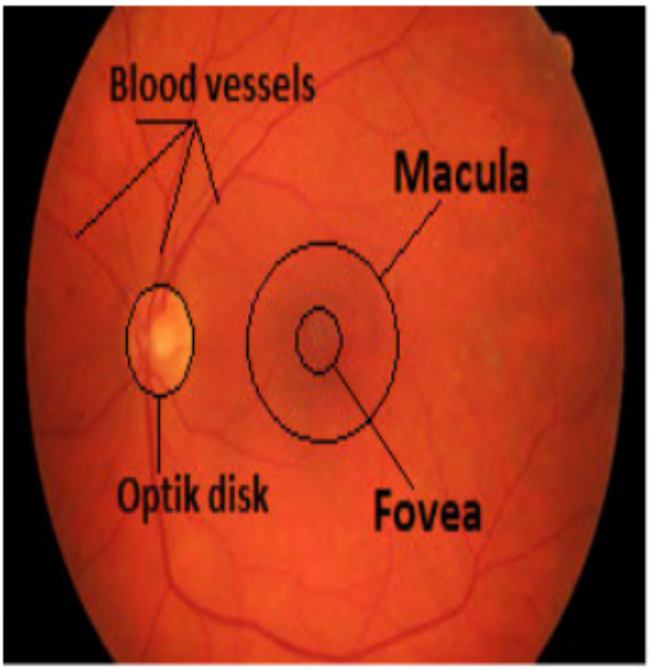


Figure 1. The retina of the eye is the entire orange circle. The OD is the yellowish part, blood vessels, and macula.

A. RETINAL DISEASES

a) GLAUCOMA

There are three main methods used to detect Glaucoma: Intraocular Pressure Measurement (IPM), Function-Based Visual Field Test (FBVFT), and Optic Nerve Head (ONH). The ONH test is an easy paradise to diagnose Glaucoma at its earliest and is done at the same time by professional and trained glaucoma specialists. It uses various morphological parameters such as the vertical Cup-to-Disc Ratio (CDR), Edge-to-Disc Ratio (EDR), Neuro-retinal Border Ratio (NBR), and disk width to detect Glaucoma in retinal images . In other words, the diagnosis of Glaucoma using the ONH method is achieved by examining OD. The eye contains millions of nerve fibers that run from the retina to the optic nerve, located in the center of the OD. When glaucoma is present, the OD becomes empty, and the optic nerve assumes the shape of a cup (OC), exceeding the value

of about 0.5 OD. In Figure 2, the difference between one eye with Glaucoma and one healthy eye is shown.

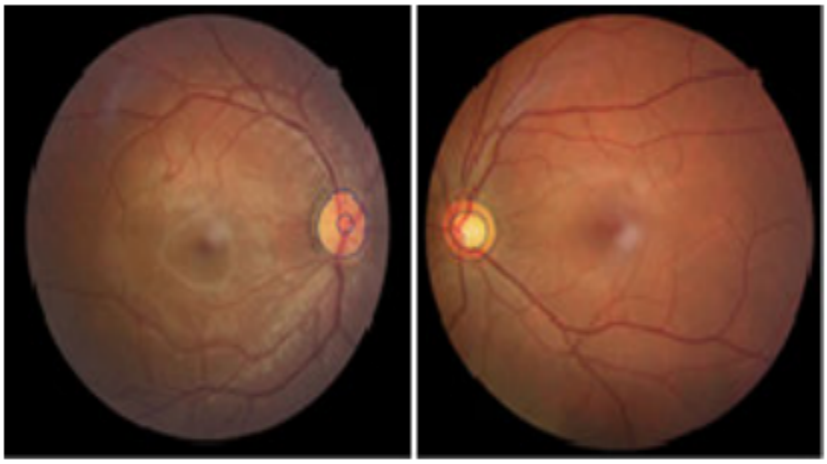


Figure 2. Retinography a) shows an entirely healthy eye and b) has Glaucoma.

b) DIABETIC RETINOPATHY

Diabetes mellitus occurs when the pancreas fails to produce enough insulin, which affects the circulatory system and produces an increased glucose, causing the deterioration of walls of blood vessels to collapse leading to bubbles, blockages, and leaks in the veins. Diabetesic retinopathy is a condition caused by diabetes mellitus. Observing the retinal circulation is very important in controlling systemic diseases before they spread. The lesions that occur in the fundus of the eye as a result of the disease are:

• Micro-Aneurysms (MA): are small dark red spots.

• Exudates are large amounts of cells, and fluid from blood vessels or organs, especially in inflammation.

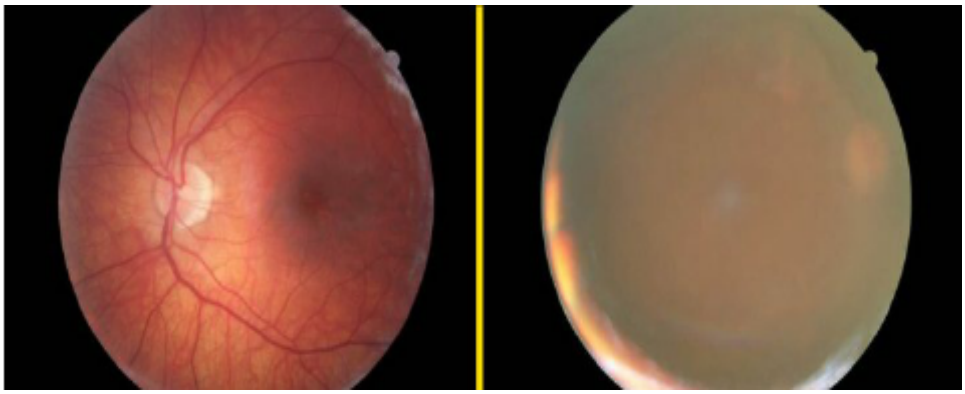
Figure 3 shows the conditions caused by this disease.

(a) (b)

Figure 3. a) Healthy retinography. b) A diabetic retinography

c) CATARACT

A cataract is a cloudy area in the lens of the eye that leads to a decrease in vision.Cataracts often develop slowly and can affect one or both eyes. Symptoms may include faded colors, blurry or double vision, halos around light, trouble with bright lights, and trouble seeing at night.This may result in trouble driving, reading, or recognizing faces. Poor vision caused by cataracts may also result in an increased risk of falling and depression.Cataracts cause half of all cases of blindness and 33% of visual impairment worldwide.



a. b.

Figure 3. Retinography a) shows an entirely healthy eye and b) has Cataract

Eye diseases have become a major problem worldwide, especially in developing countries where technology and finance are limited. Today, the problem is solved because of the task of classification that is part of pattern recognition. Its main purpose is to integrate common elements from any organization, object, event, or event that are real or invisible. Convolutional Neural Networks are a type of Sensitive Network used to distinguish intelligent patterns, Machine Learning, and Data Mining. Also, medicine and ophthalmology use these algorithms to detect diseases in the human body. This work introduces a novel algorithm for clever pattern separation based on the Convolutional Neural network. Two different groups of retinography images are presented: Glaucoma and Diabetic Retinopathy.

First we pre-process the image to enhance the required features for the Deep learning techniques to work, which includes: 1. Green Channel extraction 2. Application of filter

Next, we use this pre-processed image as an input to our CNN architecture and machine learning algorithm

The percentage accuracy rate is closer to 90%. Numeric metrics: Accuracy, Recall, Specificity,Precision and F1 score with values close to 1.

4.Objective of the project -- with clinical motivation:

Studies have shown that many of the leading causes of vision impairment and blindness worldwide are irreversible and cannot be cured.Various ocular diseases are capable of causing permanent and irreversible damage to the patient’s vision, and in extreme cases, it can even lead to blindness. For example, glaucoma, which is the second leading cause of blindness globally, is a chronic and irreversible neurodegenerative disease . However, due to the lack of obvious visual symptoms in glaucoma and the shortage of clinical resources for communal screening, more than 90% of glaucoma cases remain undetected in the population. Early eye diagnosis is an economical and effective way to prevent blindness caused by diabetes, glaucoma, cataract, age-related macular degeneration (AMD), and many

other diseases .Prompt and automatic diagnosis is important and urgent in reducing the activity of the ophthalmologist and preventing visual impairment in patients. Computer vision and deep learning can automatically detect ocular diseases after providing high quality fundus medical images.

In this project we are trying various experiments and methods to build a different model with the help of convolutional neural networks using the TensorFlow library. Also we are trying to analyze fundus images using image processing techniques in matlab and OpenCV python like vessel extraction, detection of edges, detection of optical disc. Apart from this we are trying to implement a few Machine Learning based classification algorithms for classification tasks afterwards we will adapt the algorithm which gives best accuracy.

Pattern Recognition and Machine Intelligence (PRMI) part:
We are trying to classify the fundus images using CNN and Machine Learning algorithms into the aforementioned different classes.

Medical Image analysis part:
Will try to find some insights about the round spots of dark red color in retinopathy images. Apart from this we will try to find some statistical inferences like location,perimeter and area from those spots
Also try to extract boundaries and vessels from fundus images and analyze some patterns from that.

5. Methodology((including all major mathematical equations, image preprocessing, features, classifiers, and parameters clearly described)

5.1 Task Definition

We have a dataset named “Ocular Disease Recognition " wherein the fundus images of an eye is given and are classified into eight categories namely Normal (N),Diabetes (D),Glaucoma (G),Cataract (C),Age related Macular Degeneration (A),Hypertension (H),Pathological Myopia (M),Other diseases/abnormalities (O) .Along with it information such as gender ,age and keywords about the defect in

eyes is provided. Initially all the images were unlabelled , and its classification was evident from the dataset where all information about fundus images of an eye were stored along with its file name. Our first task was to label and differentiate all the fundus images according to their label and for this we took help of a few libraries and the dataset itself. Once the segregation was done, now was the time for image preprocessing .

Image Preprocessing:

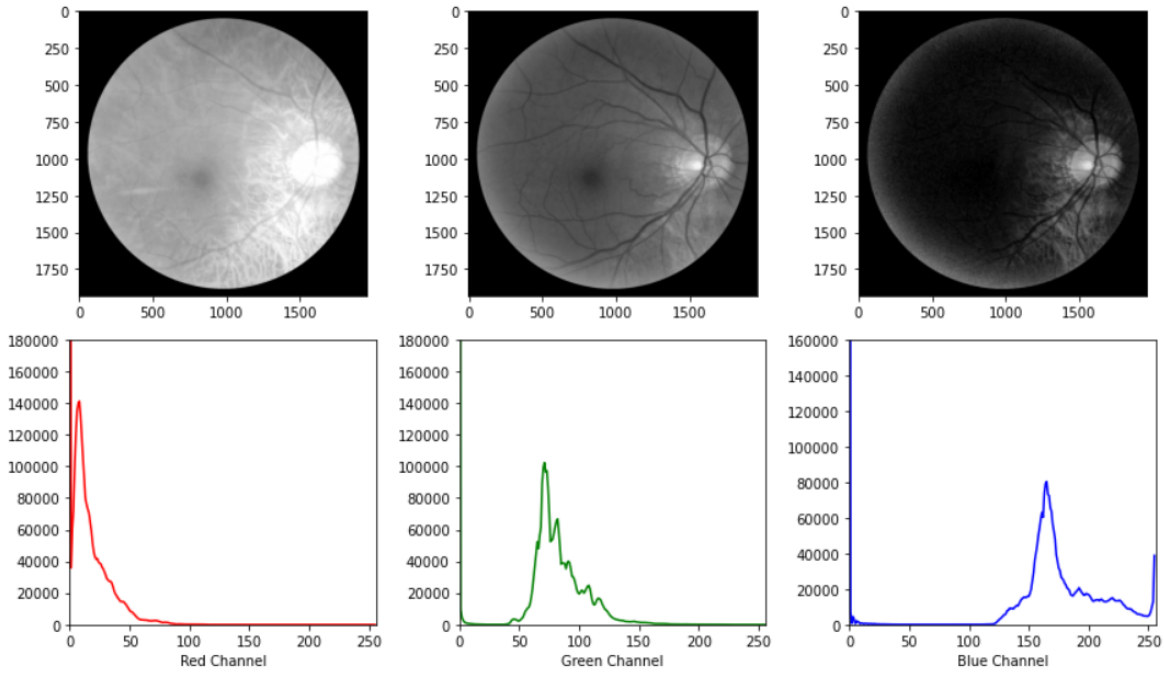
The most important part before feeding raw data to any model is to clean the data,remove noise from it , and extract maximum information out of it.

Image Resizing :As all the images were of high dimension and resolution , we resize the images to 128x128 so that there must be uniformity among each and every image size and computational power decreases as less number of elements are need to be processed now as of earlier.After resizing , each image was normalized .

Data normalization: It is a very vital step which basically ensures that each input pixel has a similar data distribution. This makes convergence faster while training the network. It is done by subtracting the mean from each pixel and then dividing the result by the standard deviation.

Dimensionality reduction: All fundus images have RGB channels which need to be collapsed into a single gray-scale channel. It was observed that the green channel is best for segmenting blood vessels in color fundus images because generally it has the highest contrast between blood vessels and the retinal

background while the red channel is rather saturated and the blue channel is rather dark.



Fundus images are always saturated in the red channel and low contrast in the blue channel we choose the green channel for all our operations

Denoising images via Median filter
Contrast Enhancement : Contrast Limited Adaptive Histogram Equalization (CLAHE) used for distinguishing components of fundus images.
Morphological Operations : Erosion ,Dilation,Bottom hat etc were used

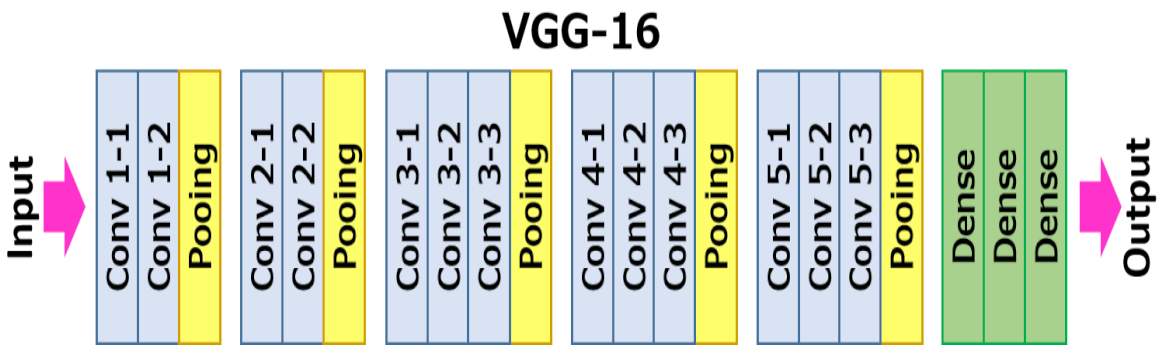
Features

Features are the inputs that are fed to the ML model to output a prediction.

1. Statistical Features such as mean,median ,standard Deviation are used for each image after preprocessing.
2. Ratio of white pixels to total number of pixels is also used over here as a distinguishing feature.
3. Gray level co-occurrence matrix features such as Energy ,Homogeneity ,Dissimilarity ,Correlation and Contrast are also used.

VGG16 Architecture:

VGG-16 NETWORK ARCHITECTURE:



- VGG-16 consists of 16 learnable layers in which 13 are the convolutional layers and the other 3 are fully connected layers.
- After every convolution layer in the network, we have used ReLU functions which help to prevent the exponential growth in the computation required to operate the neural network and it prevents vanishing gradients.
- At the output, the softmax activation function is used as it is a multiclass classification.
- It is considered to be one of the excellent vision model architectures to date.
- Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with stride 1 and always used the same padding and maxpool layer of 2x2 filter of stride 2.
- The input image is resized into the image standard size of 224*224*3 to this VGG-16 model.

The original dataset which we obtained from the Kaggle website consists of 8 categorical datasets; out of 8 classes we are using 4 classes for our model. It is already labeled by themselves. We've designed a model using the standard VGG-16 network which we covered in the Deep learning course and it is a standard network which is used here as it gives good classification of the image data. To this model, datasets pre-processed images have been given as the input to them.

Parameters:

The parameters used in the network in the study;

- Image dimensions: 224*224
- Optimizer: Adam
- Number of revolutions (Epoch): 40

- Number of Batches: 128
- Kernel Size - 3x3
- Max Pooling - 2x2
- Learning Rate : 0.001
- Error function: Categorical Cross Entropy
- Number of training data: 7635
- Number of test data: 955
- Number of validation data: 955
- Monitoring : Validation Loss
- Activation Function : ReLU

Adam Optimizer:

Algorithm 1: Adam Optimizer

Initialize θ_0, $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
While θ_t not converged
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
Bias Correction
$\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\widehat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$
Update
$\theta_t \leftarrow \prod_{\mathcal{F}, \sqrt{\widehat{v_t}}} \left(\theta_{t-1} - \frac{\alpha \widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon} \right)$

Precision:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Recall:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

F1 Score:

$$\text{F1 Score} = \frac{\text{2 x (Precision x Recall)}}{\text{Precision + Recall}}$$

6. Detailed Algorithm -- cite equations for proper understanding

Algorithm Definition (Machine Learning part):

1. Firstly to improve contrast we applied histogram equalization so that we could get a better picture and more information.
2. According to a research paper technique known as “Contrast Limited Adaptive Histogram Equalization (CLAHE)” was applied to images which improves the contrast of the image and makes detection of abnormalities more precise.
3. Canny Edge detection was used in order to detect the blood vessels and distinguish them from the background .
   As it can be seen from the above diagram that Normal fundus images have more visibility of blood vessels than the Cataract ,as the eye becomes cloudy in case of it .So blood vessels are easily detected in white color that means Normal fundus images after applying Canny filter results in more number of white pixels than Cataract which can be used as a features.

4. Bottom Hat transformation is used for vessel extraction . It is used to extract bright features from the darker background. By using Bottom Hat Transform we get the extracted blood vessel from the retinal image.

   Once we applied a Bottom hat transformation on our image ,a threshold was set in order to make the image binary and then measure the count of the total number of white pixels as a feature to distinguish between Cataract and Normal fundus images.

5. Morphological operation such as Erosion was applied on every image and the mean ,std deviation value were measured for each image as a feature,

   Later Gray Level Co-occurrence matrix features were calculated such as Energy,Correlation,Dissimilarity and homogeneity.

6. Various Features were extracted and fed to different Machine Learning models.

Algorithm Definition (Deep Learning part):

The following steps were used to develop the deep learning model to pre-processed images.

1. Obtaining the input dataset:
   In the case of preprocessed images, the dataset which is present at kaggle, we are downloading it through the os environment and unzipping at google colab.

2. Data arrangement:
   In this step, we have a csv file of all image data with their labels. Now we have written a function for splitting data into the particular category. So we are resizing the image data to 224*224 sizes and appending the images of a particular class to that dataset frame. So we have appended all four class images in the ds data frame. The categories used are, 0: Cataract, 1: Glaucoma, 2: Normal and 3: Diabetes

3. Splitting of data to train,test and validation:
   The variable in which the directory of images stored and the respective categories are then splitted to train the dataset as 80%, test dataset and validation dataset each of 10%. Train and validation dataset is given to train the model and check the accuracy of both train and validation to see how the model is getting trained and how it is working for the validation data simultaneously. [8]
4. Model Designing:
   The standard model VGG-16 is used for this classification model. It is used because the model has the ability to highly distinguish the image of different classes.[9]

Apart from that we have also used transfer learning based VGG16 model trained on “Imagenet” dataset as Initializer.

5. Model Training:
   For training the model, to observe how the model behaves when we use Adam as an optimizer. They are used because ADAM is the best optimizer among all.
6. Plotting the Training and validation accuracy of the models.
7. Plotting the Training and validation losses of both the models.
8. Using the 10% of the test data, prediction of the data is done and compared with the actual category and plotting the confusion matrix to obtain the performance metrics of the model performance.
9. Also finding Performance parameter i.e. Precision, Recall and F1 score.

7. Data Description:

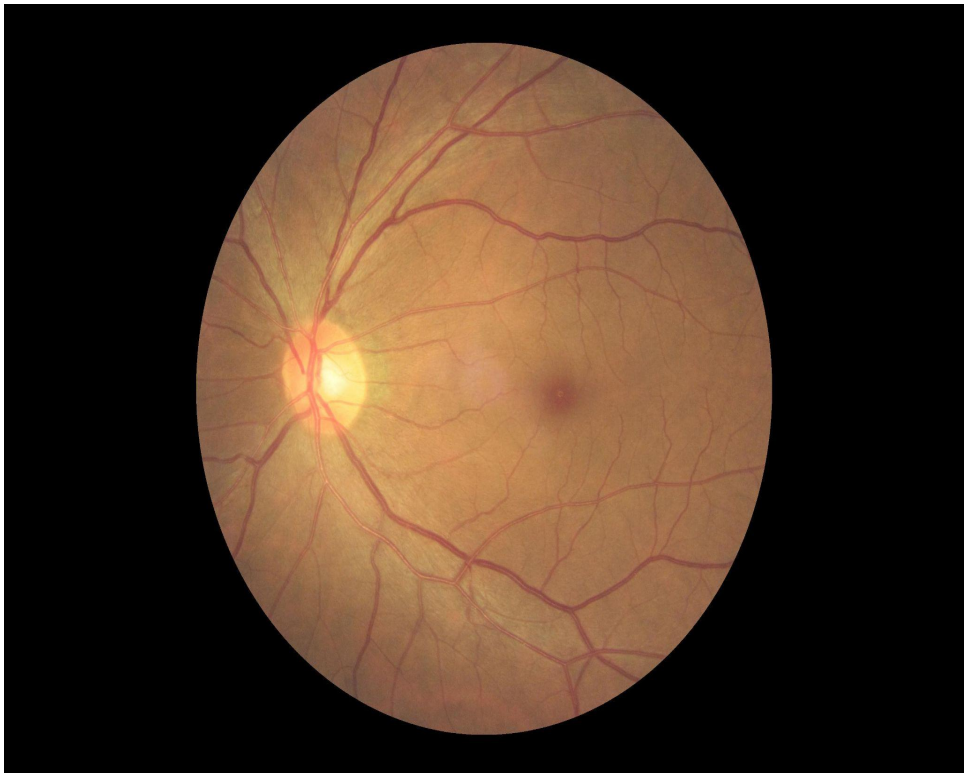
Dataset Used: Ocular Disease Intelligent Recognition (ODIR)

Ocular Disease Intelligent Recognition (ODIR) is a structured ophthalmic database of 5,000 patients with age, color fundus photographs from left and right eyes and doctors' diagnostic keywords from doctors.

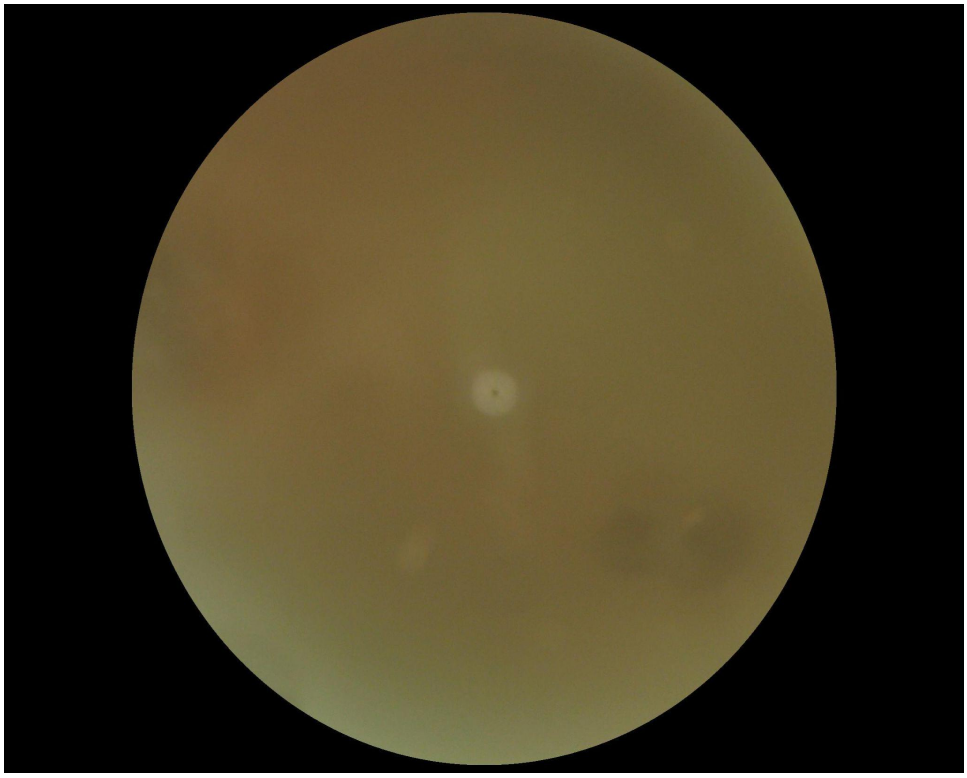
Classes :

A. Normal (N),
B. Diabetes (D),
C. Glaucoma (G),
D. Cataract (C),
E. Age related Macular Degeneration (A),
F. Hypertension (H),
G. Pathological Myopia (M),
H. Other diseases/abnormalities (O)

Images from dataset with their classes:

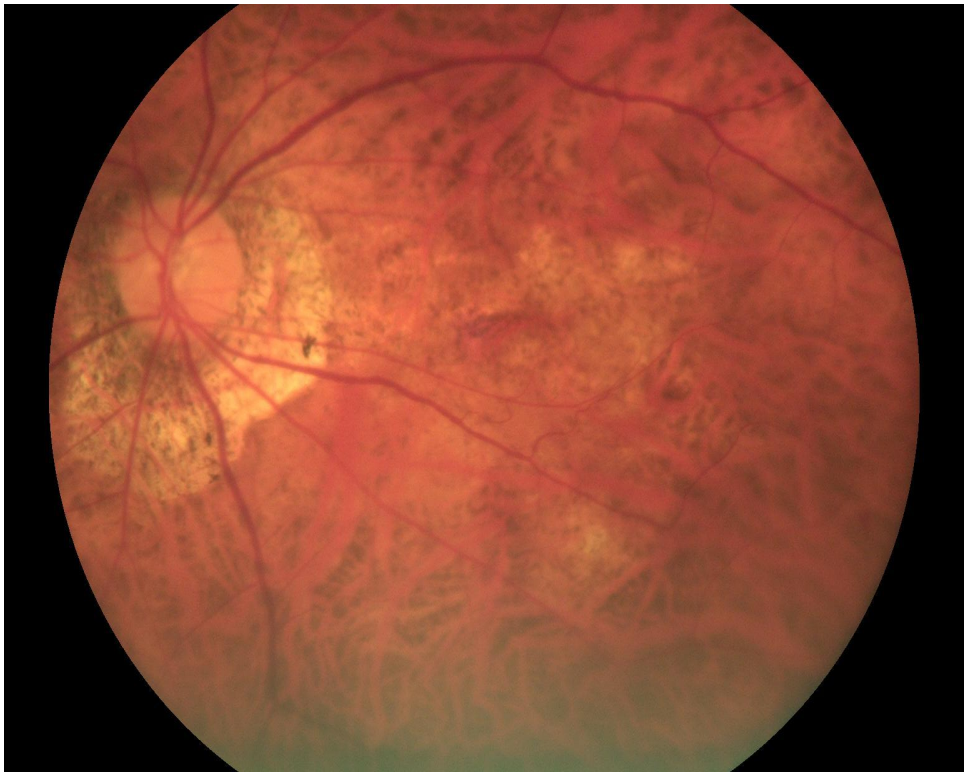


Normal

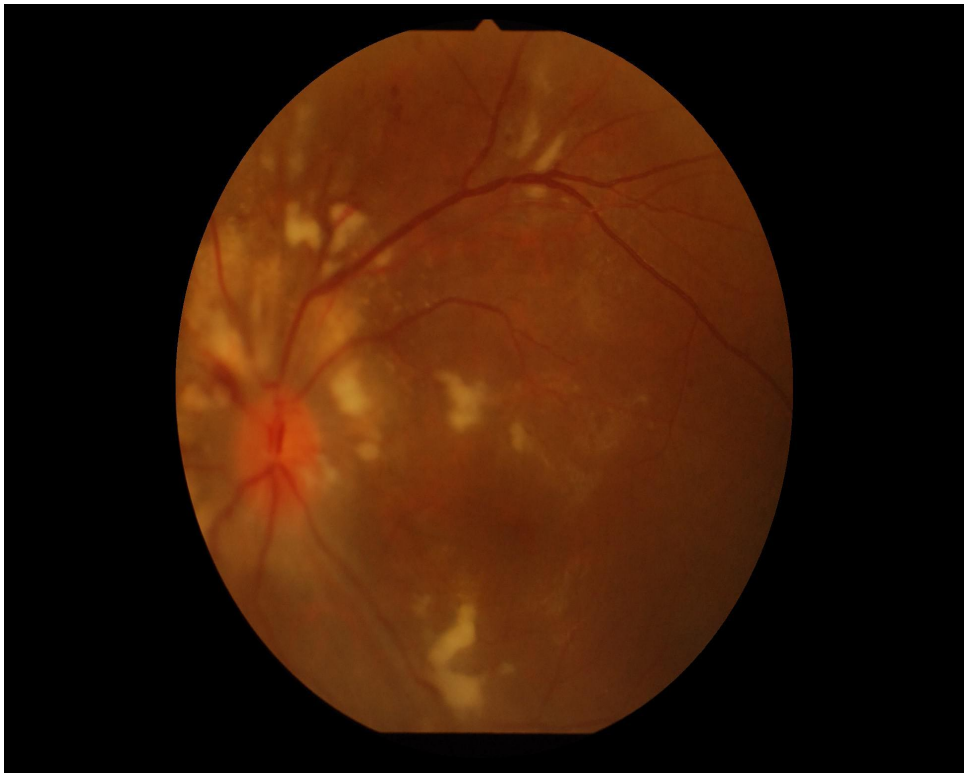


Cataract

Glaucoma



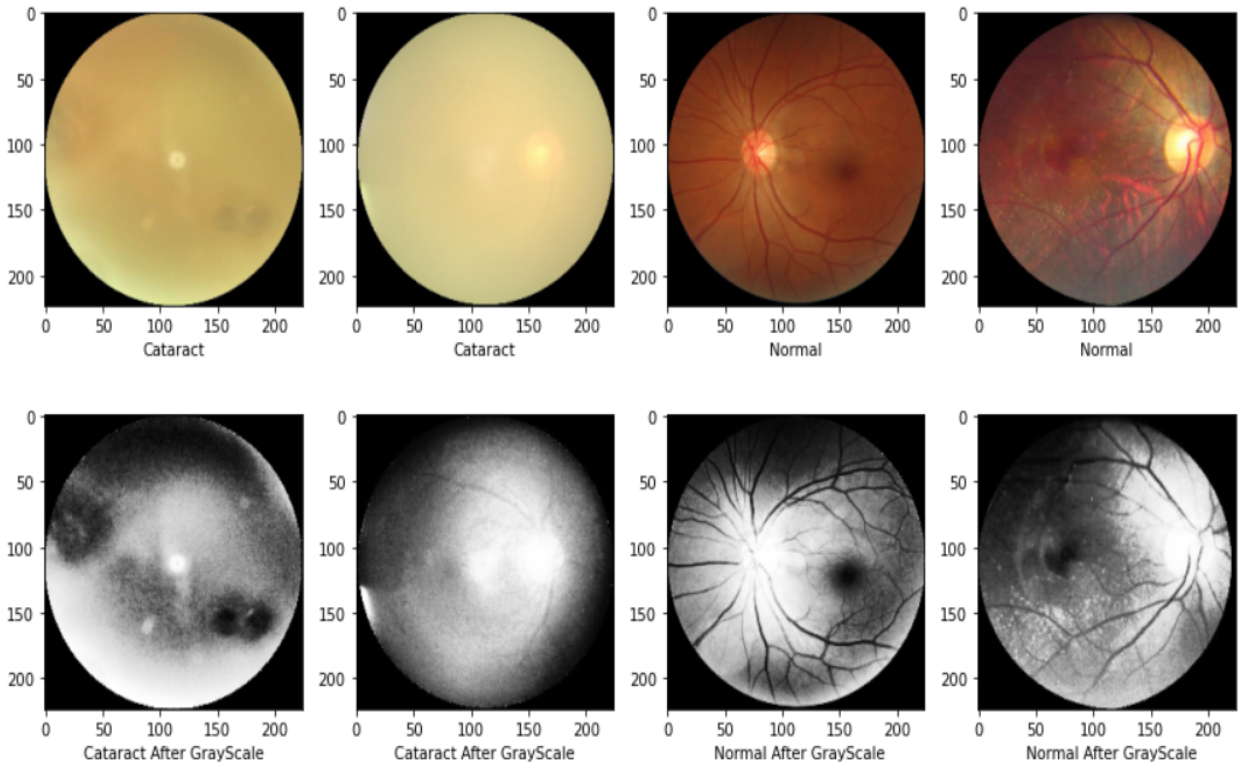
Myopia



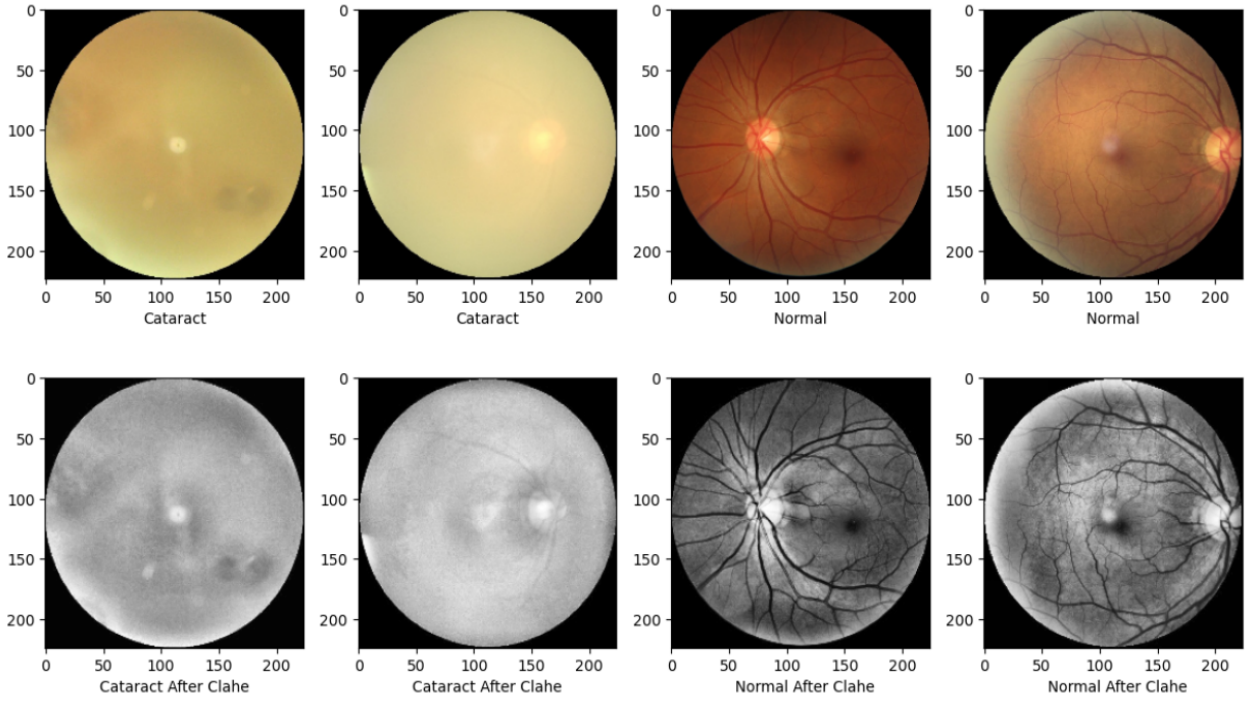
Retinopathy

Results obtained:

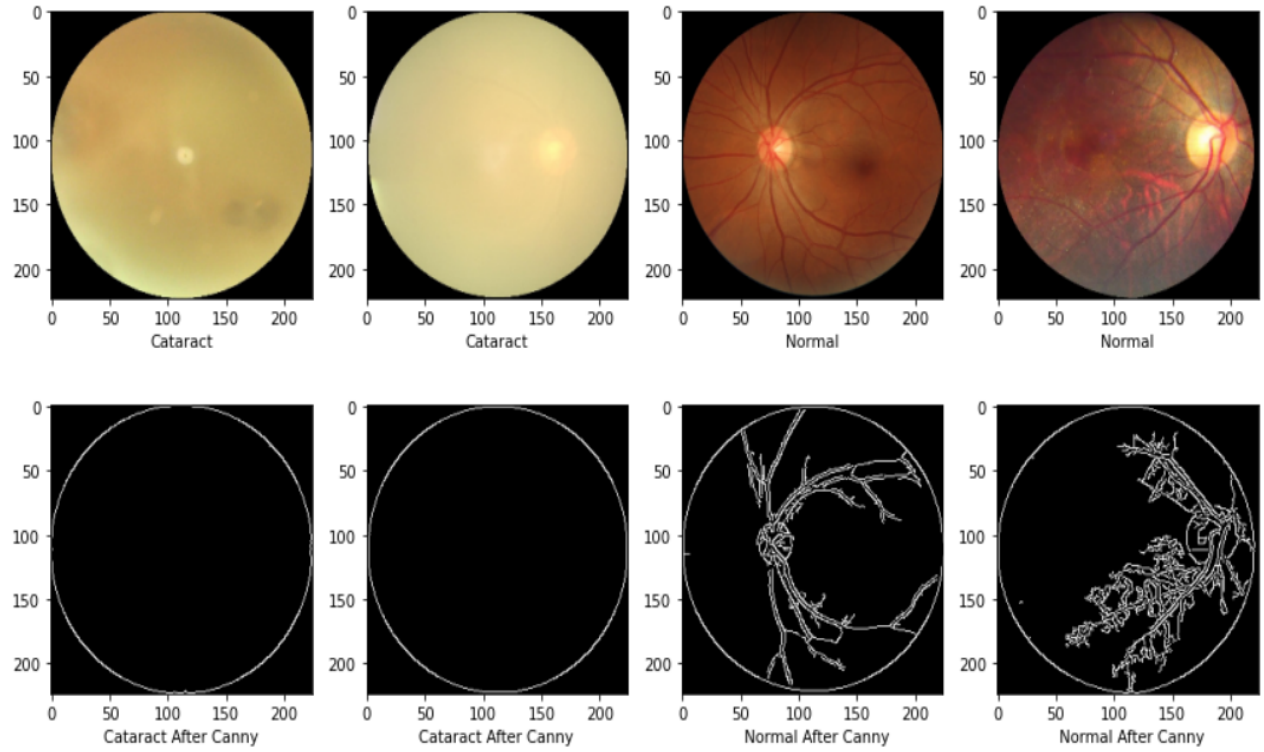
After Grayscaling



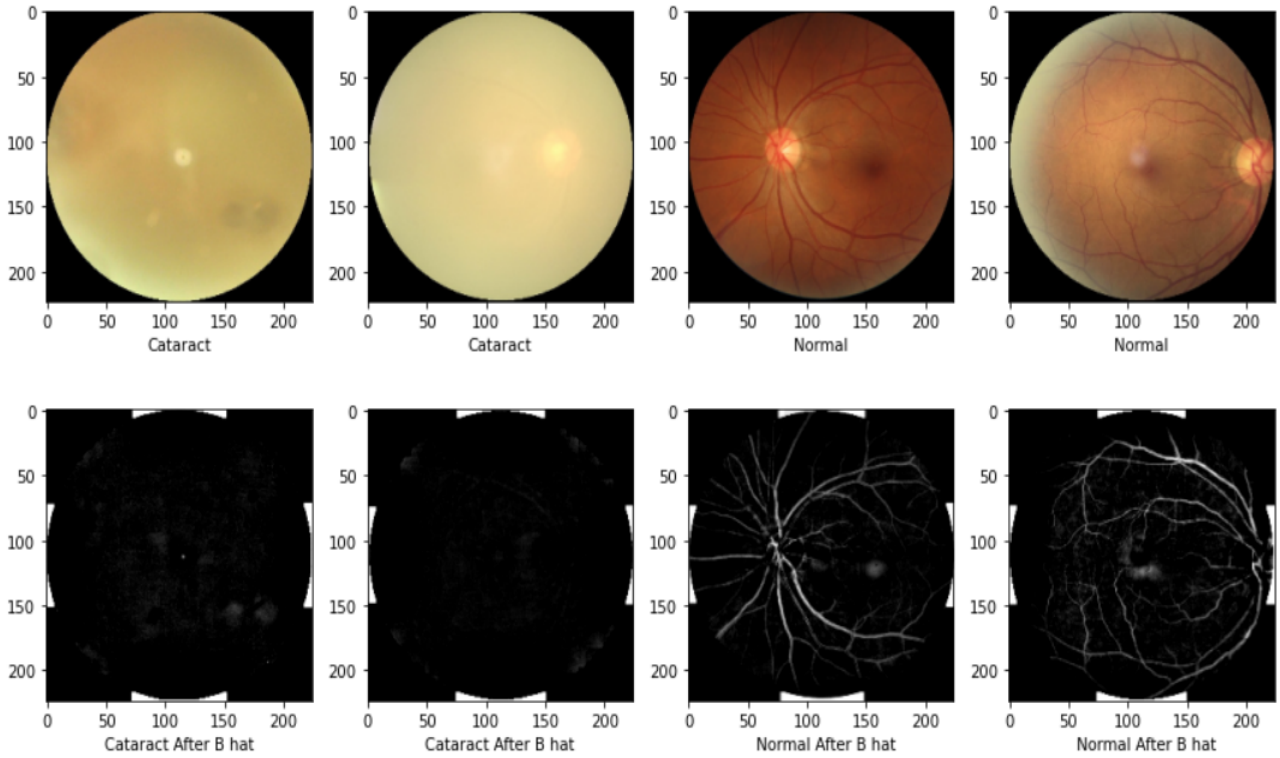
Applying CLAHE



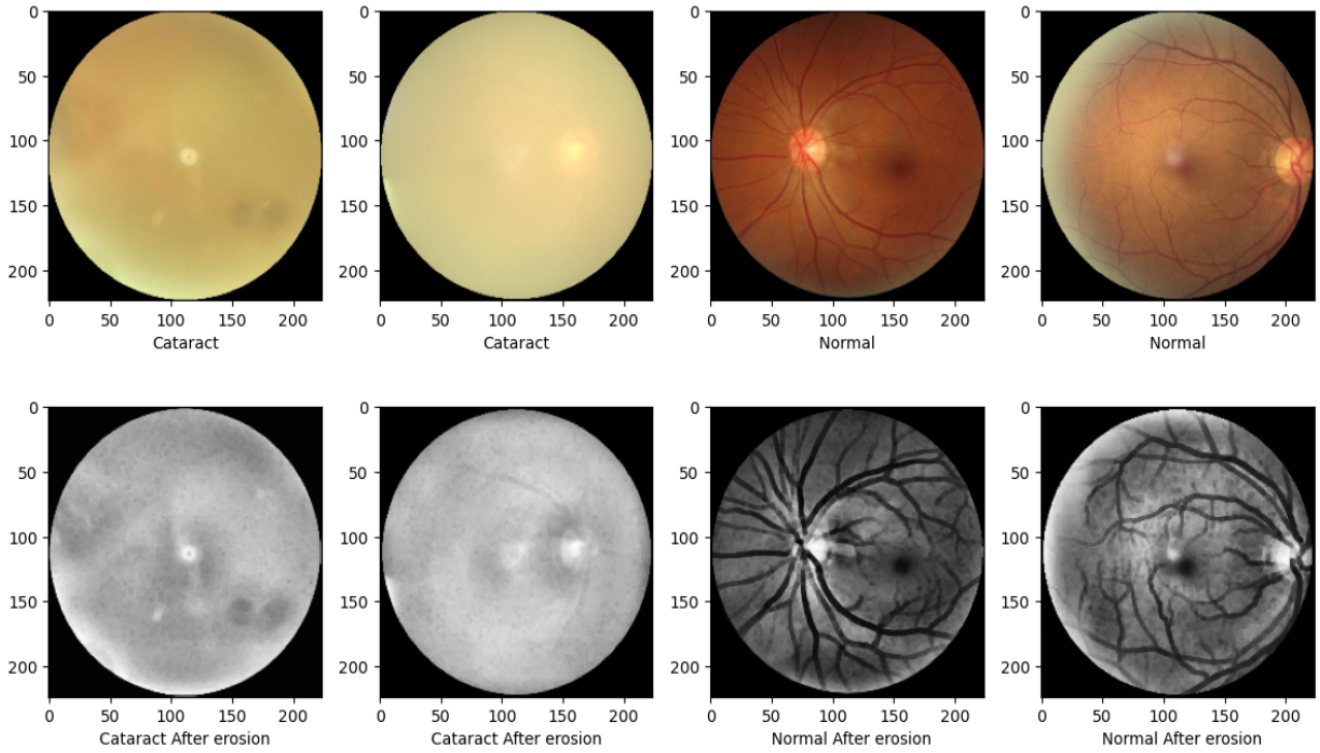
Edge Detection



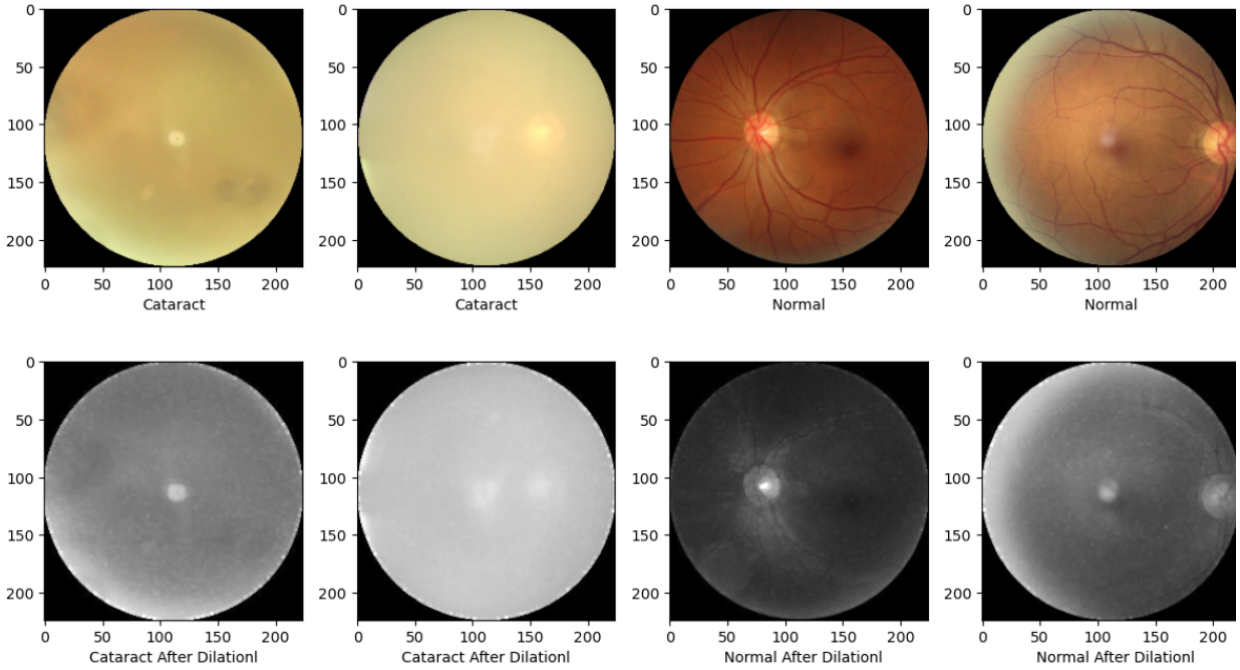
Bottom Hat Transform



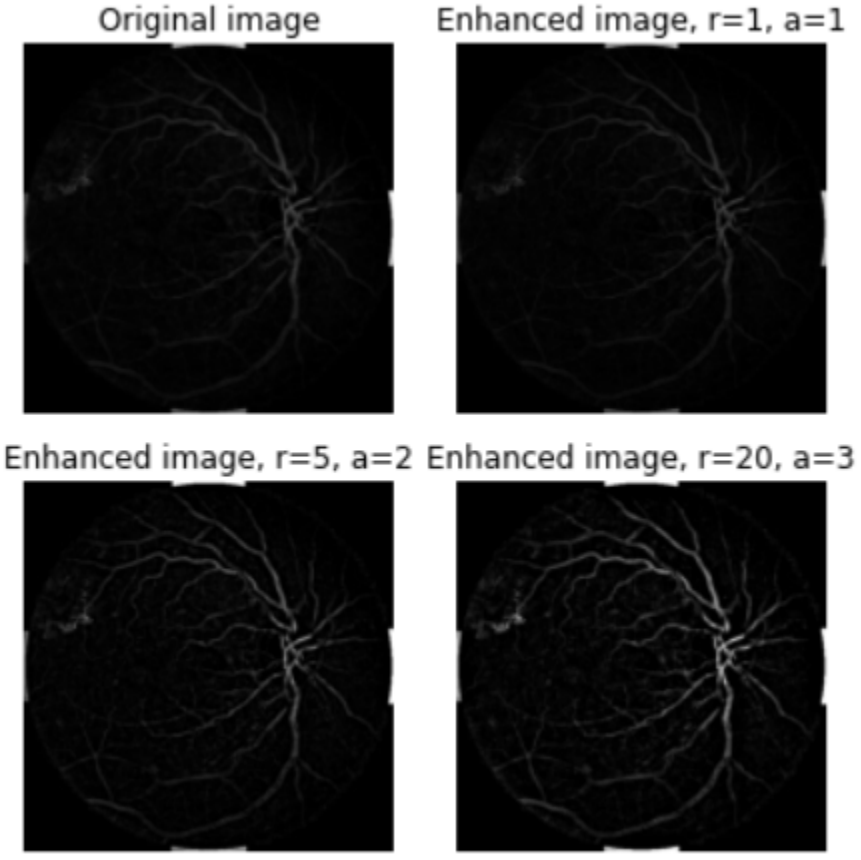
Erosion



Dilation



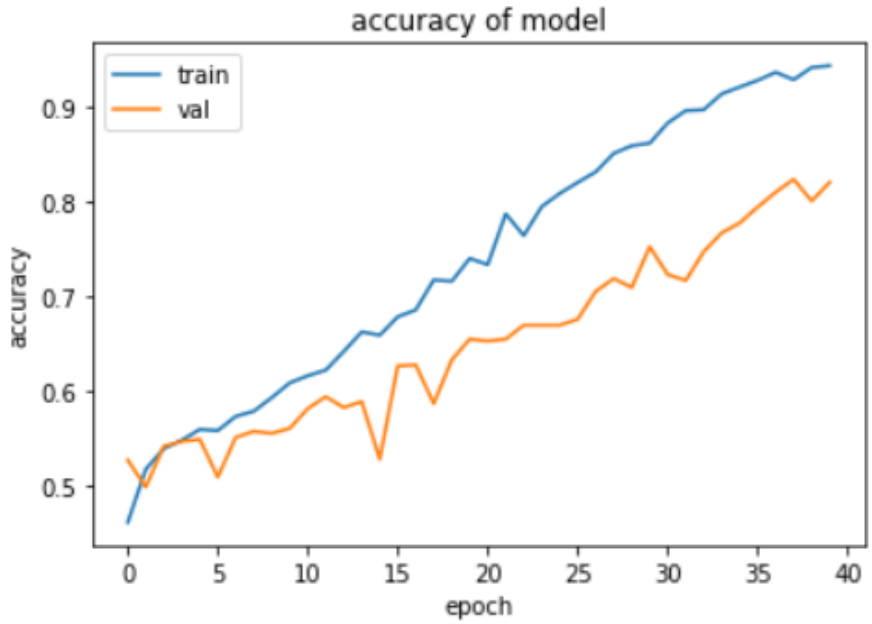
Exudates Detection

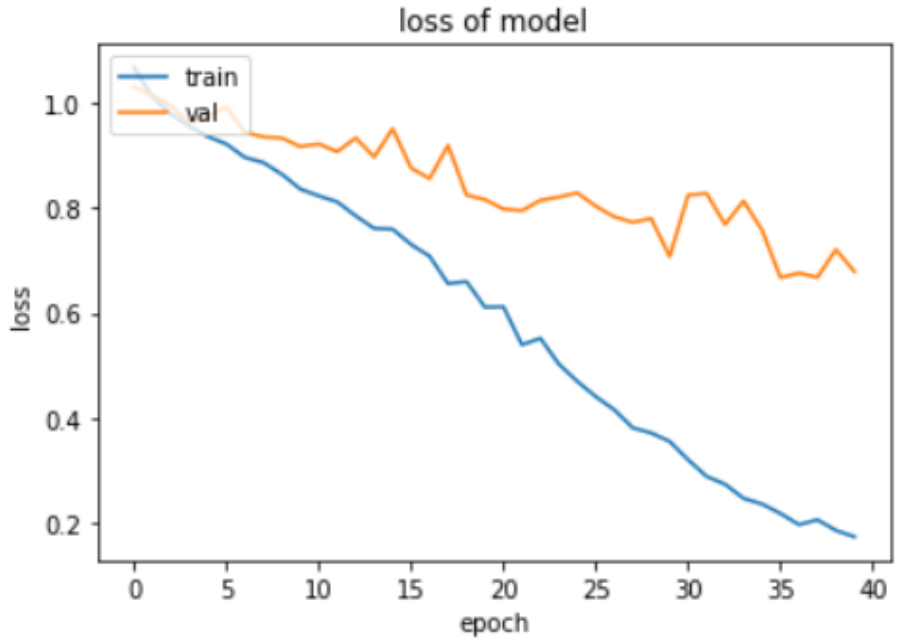


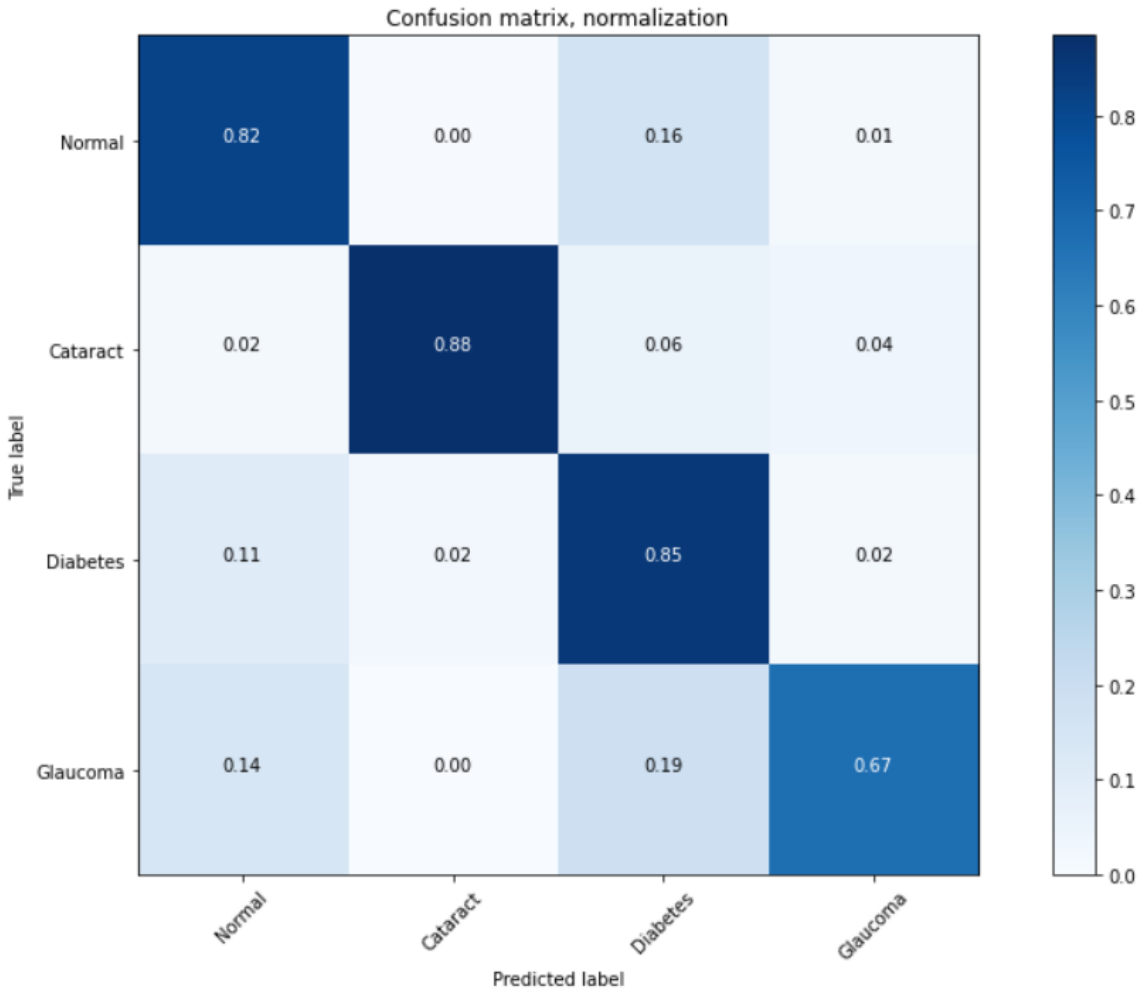
VGG 16 model Result:
All the required observations for both the models are plotted in the form of graphs and in the matrix form for better understanding of data in the visualized form. It is observed that ADAM optimizer shows an increasing slope in the case of Accuracy and decreasing slope in the case of Losses. So, when the training epochs increases, it is sure that Adam optimizer performs very well

The accuracy and the loss graphs for both the models in preprocessed images:

The following graphs, training testing accuracy and loss, confusion matrix, Precision, Recall and F1 Score are for the model built from scratch with Adam as an optimizer:



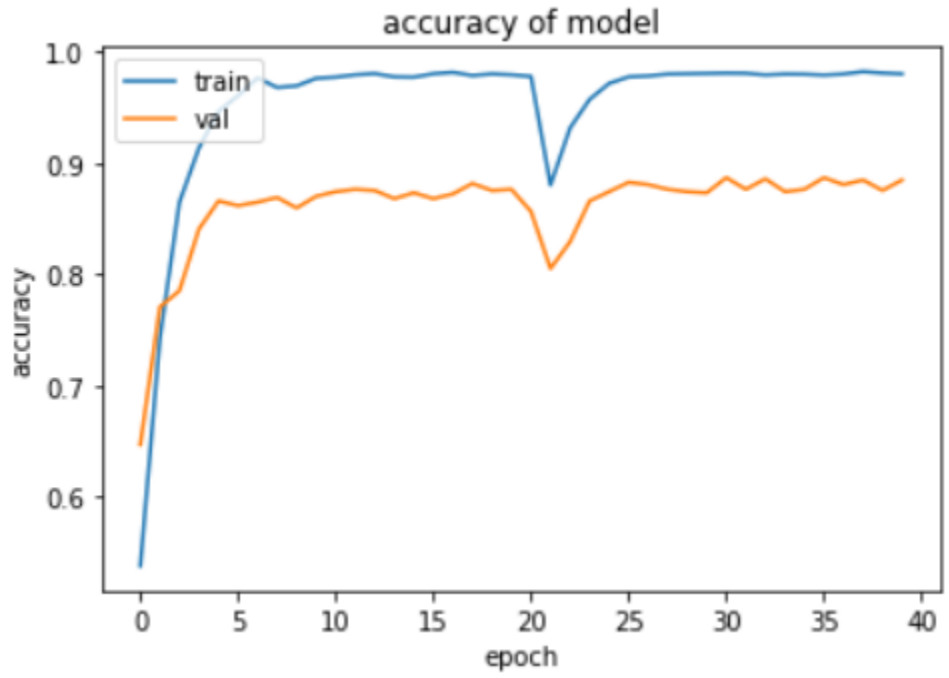


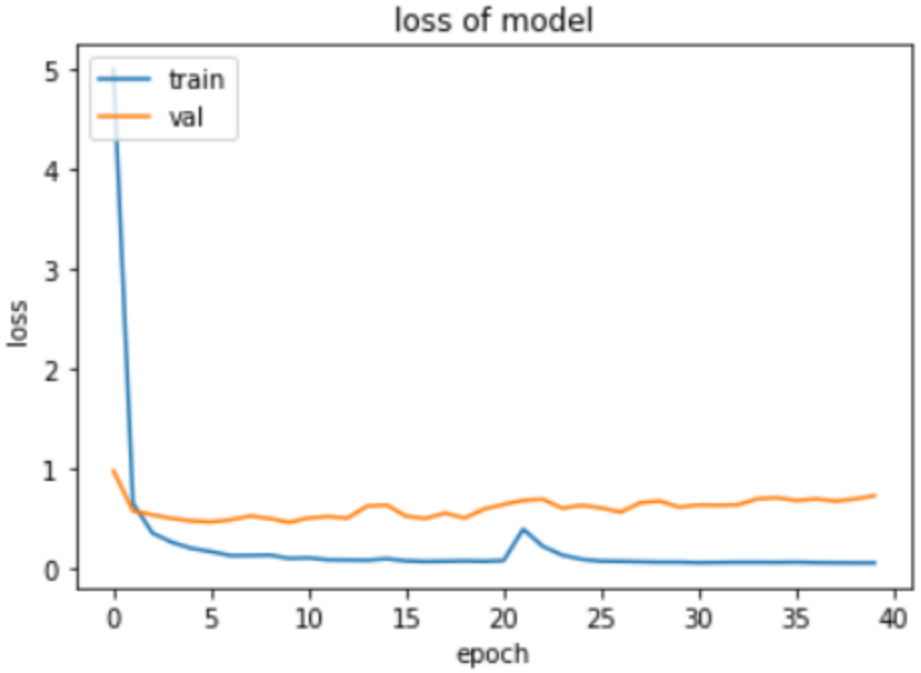


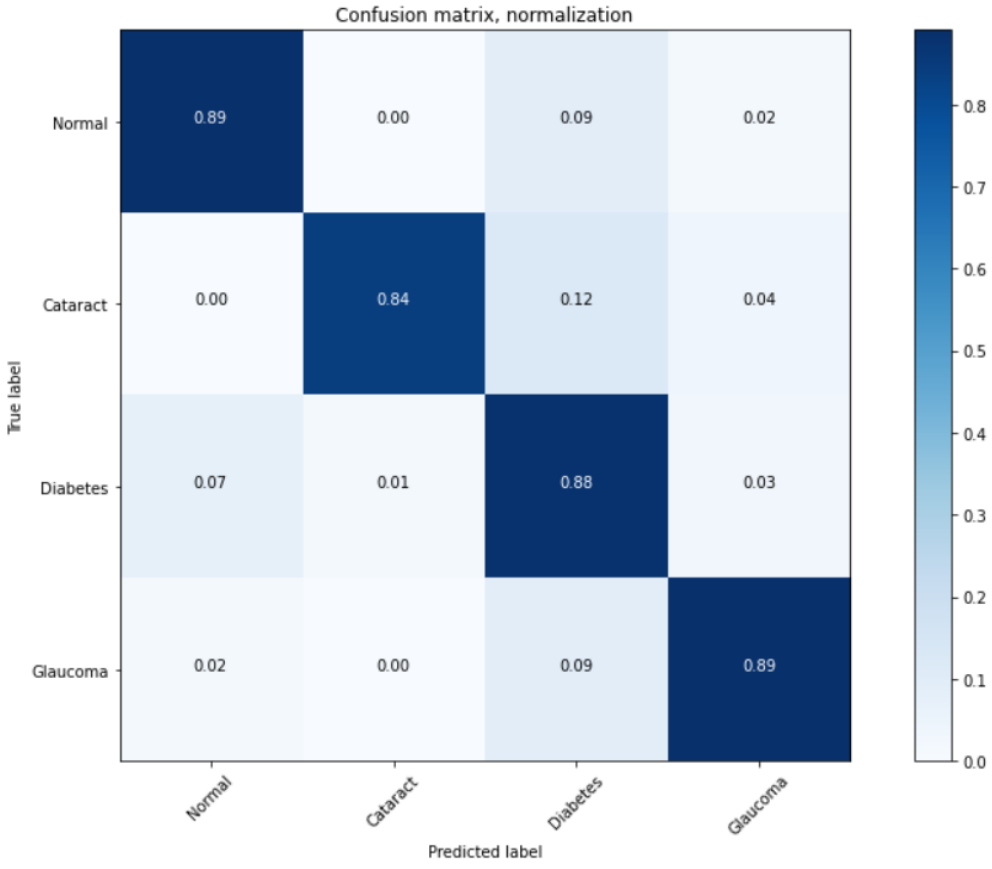
	precision	recall	f1-score	support
Normal	0.86	0.82	0.84	414
Cataract	0.81	0.88	0.84	52
Diabetes	0.81	0.85	0.83	425
Glaucoma	0.73	0.67	0.70	64
accuracy			0.83	955
macro avg	0.80	0.81	0.80	955
weighted avg	0.83	0.83	0.83	955

```
Epoch 35/40
60/60 [==============================] - 292s 5s/step - loss: 0.2355 - accuracy: 0.9202 - val_loss: 0.7580 - val_accuracy: 0.
7767
Epoch 36/40
60/60 [==============================] - 292s 5s/step - loss: 0.2175 - accuracy: 0.9273 - val_loss: 0.6673 - val_accuracy: 0.
7935
Epoch 37/40
60/60 [==============================] - 293s 5s/step - loss: 0.1962 - accuracy: 0.9358 - val_loss: 0.6759 - val_accuracy: 0.
8092
Epoch 38/40
60/60 [==============================] - 292s 5s/step - loss: 0.2055 - accuracy: 0.9280 - val_loss: 0.6677 - val_accuracy: 0.
8229
Epoch 39/40
60/60 [==============================] - 292s 5s/step - loss: 0.1852 - accuracy: 0.9409 - val_loss: 0.7207 - val_accuracy: 0.
7998
Epoch 40/40
60/60 [==============================] - 292s 5s/step - loss: 0.1731 - accuracy: 0.9428 - val_loss: 0.6790 - val_accuracy: 0.
8197
```

The following graphs, training testing accuracy and loss, confusion matrix, Precision, Recall and F1 Score are for the model build through transfer learning trained on “ImageNet” dataset as an Initializer with Adam as an optimizer:




loss of model
train
val
loss
epoch
0
1
2
3
4
5
0
5
10
15
20
25
30
35
40


Confusion matrix, normalization
Normal
Cataract
Diabetes
Glaucoma
True label
0.89
0.00
0.09
0.02
0.00
0.84
0.12
0.04
0.07
0.01
0.88
0.03
0.02
0.00
0.09
0.89
Normal
Cataract
Diabetes
Glaucoma
Predicted label
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

	precision	recall	f1-score	support
Normal	0.92	0.89	0.91	427
Cataract	0.89	0.84	0.86	57
Diabetes	0.87	0.88	0.88	407
Glaucoma	0.72	0.89	0.80	64
accuracy			0.88	955
macro avg	0.85	0.88	0.86	955
weighted avg	0.89	0.88	0.88	955

```
60/60 [==============================] - 70s 1s/step - loss: 0.0803 - accuracy: 0.9716 - val_loss: 0.6227 - val_accuracy: 0.8742
Epoch 26/40
60/60 [==============================] - 70s 1s/step - loss: 0.0628 - accuracy: 0.9775 - val_loss: 0.5964 - val_accuracy: 0.8826
Epoch 27/40
60/60 [==============================] - 70s 1s/step - loss: 0.0597 - accuracy: 0.9783 - val_loss: 0.5558 - val_accuracy: 0.8805
Epoch 28/40
60/60 [==============================] - 70s 1s/step - loss: 0.0558 - accuracy: 0.9800 - val_loss: 0.6488 - val_accuracy: 0.8763
Epoch 29/40
60/60 [==============================] - 70s 1s/step - loss: 0.0516 - accuracy: 0.9802 - val_loss: 0.6664 - val_accuracy: 0.8742
Epoch 30/40
60/60 [==============================] - 70s 1s/step - loss: 0.0524 - accuracy: 0.9805 - val_loss: 0.6043 - val_accuracy: 0.8732
Epoch 31/40
60/60 [==============================] - 70s 1s/step - loss: 0.0463 - accuracy: 0.9807 - val_loss: 0.6274 - val_accuracy: 0.8868
Epoch 32/40
60/60 [==============================] - 70s 1s/step - loss: 0.0488 - accuracy: 0.9806 - val_loss: 0.6235 - val_accuracy: 0.8763
Epoch 33/40
60/60 [==============================] - 70s 1s/step - loss: 0.0511 - accuracy: 0.9792 - val_loss: 0.6288 - val_accuracy: 0.8857
Epoch 34/40
60/60 [==============================] - 70s 1s/step - loss: 0.0511 - accuracy: 0.9800 - val_loss: 0.6881 - val_accuracy: 0.8742
Epoch 35/40
60/60 [==============================] - 70s 1s/step - loss: 0.0495 - accuracy: 0.9798 - val_loss: 0.6992 - val_accuracy: 0.8763
Epoch 36/40
60/60 [==============================] - 70s 1s/step - loss: 0.0521 - accuracy: 0.9790 - val_loss: 0.6720 - val_accuracy: 0.8868
Epoch 37/40
60/60 [==============================] - 70s 1s/step - loss: 0.0470 - accuracy: 0.9800 - val_loss: 0.6870 - val_accuracy: 0.8805
Epoch 38/40
60/60 [==============================] - 70s 1s/step - loss: 0.0456 - accuracy: 0.9822 - val_loss: 0.6664 - val_accuracy: 0.8847
Epoch 39/40
60/60 [==============================] - 70s 1s/step - loss: 0.0438 - accuracy: 0.9809 - val_loss: 0.6859 - val_accuracy: 0.8753
Epoch 40/40
60/60 [==============================] - 70s 1s/step - loss: 0.0449 - accuracy: 0.9801 - val_loss: 0.7191 - val_accuracy: 0.8847
```

It is observed that out of all the algorithms done above, deep learning performed exceptionally well for the eye disease classification and it is observed that the model trained with “ImageNet” dataset through transfer learning as an Initializer performed better than the model built from scratch.

Conclusion & Future direction of research:

Convolutional Neural Networks and Machine learning are the tools for classifying and can help the science of ophthalmology, especially relating to the global problem of Glaucoma and Diabetic Retinopathy
The proposal shows numerical metrics that are almost perfect (Accuracy, Recall, Specificity Precision, and F1 score are close to 1.)
The result of accuracy was 88.38% obtained from VGG 16 trained through Transfer Learning and 82.72 obtained from Scratch Implementation of VGG 16 and 90% above in almost all Machine Learning models.
The Deep learning model in which the entire image without the region of interest given into it with the Adam optimizer performs exceptionally well compared to all other models. So, the Deep learning can be further developed by increasing the total epochs and by the fine tuning of the parameters, so that this model can be developed into the real time prediction and classification so that the early detection of eye diseases with the fundus images is possible which makes the ocular diseases get reduced with proper treatment at the earliest.

References :

1. https://www.sciencedirect.com/topics/engineering/green-channel
2. https://ijisrt.com/wp-content/uploads/2018/05/Enhancement-and-Feature-Extraction-of-Fundus-Images-1.pdf
3. https://reader.elsevier.com/reader/sd/pii/S2352340919312181?token=5743E39134AD964B3FF59190B2F0DBF022A8513A51E419C6A6A3E99B01A975E31B49998C88EB84952B0A4B193C10440C&originRegion=eu-west-1&originCreation=20220411190137
4. https://medium.com/analytics-vidhya/logistic-regression-b35d2801a29c

5. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/#:~:text=The%20batch%20size%20is%20a%20number%20of%20samples%20processed%20before,samples%20in%20the%20training%20dataset.
6. https://www.analyticsvidhya.com/blog/2020/10/the-mathematics-behind-svm/
7. https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8#:~:text=Logistic%20Regression%20is%20a%20Machine,%2C%20failure%2C%20etc.).
8. Classification of Eye Diseases in Fundus Images OMAR BERNABÉ 1, ELENA ACEVEDO 1, ANTONIO ACEVEDO 1, (Member, IEEE), RICARDO CARREÑO 2, AND SANDRA GÓMEZ 3
9. https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c
10. Neural Network and Deep Learning by Michel Nelson

Appendix:

Image Processing + ML code:

```
import numpy as np
import cv2
import os
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
import shutil
from skimage.feature import greycomatrix, greycoprops
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix,
precision_score, recall_score, f1_score, cohen_kappa_score

from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

```
df  = pd.read_excel("data.xlsx")
df.head(10)
df1  = df.loc[(df['N'] == 1) | (df["C"] ==1)| (df["D"] ==1)]
df1.head()
file_names = []

# Loaded data
training_images = []
flags = []

# Features
grayscaled_images = []
inverted_images = []
thresholded_images = []
gray_histogram_of_images = []
RGB_histogram_of_images = []
conny_edged_images = []
laplacian_edged_images = []
x_edged_images = []
y_edged_images = []
Clahe_images = []

threshold_mean = []
threshold_median = []
threshold_std_dev = []

canny_mean = []
canny_median = []
canny_std_dev = []

classes={'Pre_Cataract - Copy':1,"Pre_Normal - Copy":0}
labels = []
file_names.clear()
labels.clear()

for name in classes:
    address='C:/Users/Abhishek/Desktop/PRIMI project/ODIR-5K/' +name

    for add in os.listdir(address):
```

```
            image = cv2.imread(address+'/'+add)
            image = cv2.resize(image, (224,224))
            image = cv2.normalize(image, None, alpha = 0, beta = 255,
norm_type = cv2.NORM_MINMAX, dtype =cv2.CV_8U)
            file_names.append(image)
            labels.append(classes[name])
 print("Data Length =",len(file_names), "files", len(labels), "labels")
plt.bar([0,1], [len([i for i in labels if i == 1]), len([i for i in labels
if i == 0])], color = ['r', 'g'])
plt.xticks([0, 1], ['Cataract', 'Normal'])
plt.show()
training_images=file_names

training_images = (np.array(training_images))
df2 = pd.DataFrame({'label':labels})
print (df2)
import cv2
import matplotlib.pyplot as plt
image = cv2.imread('329_right.jpg')
image  = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
r,g,b = cv2.split(image)

plt.figure(figsize=(12,7))

plt.subplot(2,3,1)
plt.imshow(r,cmap = "gray")


plt.subplot(2,3,2)
plt.imshow(g,cmap = "gray")


plt.subplot(2,3,3)
plt.imshow(b,cmap = "gray")


plt.subplot(2,3,4)
hist = cv2.calcHist([b], [0], None, [256], [0, 256])
plt.plot(hist, color = "r")
plt.xlim([0, 256])
```

```
plt.ylim([0, 180000])
plt.xlabel('Red Channel')


plt.subplot(2,3,5)
hist = cv2.calcHist([g], [0], None, [256], [0, 256])
plt.plot(hist, color = "g")
plt.xlim([0, 256])
plt.ylim([0, 180000])
plt.xlabel('Green Channel')

plt.subplot(2,3,6)
hist = cv2.calcHist([r], [0], None, [256], [0, 256])
plt.plot(hist, color = "b")
plt.xlim([0, 256])
plt.ylim([0, 160000])
plt.xlabel('Blue Channel')




plt.tight_layout()
import numpy as np
img = cv2.imread('8_left.jpg')
median = cv2.medianBlur(img, 5)
# compare = np.concatenate((img, median), axis=1) #side by side comparison


plt.figure(figsize=(12,7))

plt.subplot(2,3,1)
plt.imshow(r,cmap = "gray")

plt.subplot(2,3,2)
plt.imshow(g,cmap = "gray")


plt.subplot(2,3,3)
plt.imshow(b,cmap = "gray")
```

```
plt.subplot(2,3,4)
median = cv2.medianBlur(r, 5)
plt.imshow(median,cmap = "gray")
plt.xlabel('Red Channel')


plt.subplot(2,3,5)
median = cv2.medianBlur(g, 5)
plt.imshow(median,cmap = "gray")
plt.xlabel('Green Channel')

plt.subplot(2,3,6)
median = cv2.medianBlur(b, 5)
plt.imshow(median,cmap = "gray")
plt.xlabel('Blue Channel')




plt.tight_layout()
def vis(images,operated_images,label):
    plt.figure(figsize=(12,7))
    plt.subplot(2,4,1)
    plt.imshow( cv2.cvtColor(images[0], cv2.COLOR_BGR2RGB))
    plt.xlabel('Cataract')

    plt.subplot(2,4,2)
    plt.imshow(cv2.cvtColor(images[8], cv2.COLOR_BGR2RGB))
    plt.xlabel('Cataract')

    plt.subplot(2,4,3)
    plt.imshow(cv2.cvtColor(images[411], cv2.COLOR_BGR2RGB))
    plt.xlabel(' Normal ')

    plt.subplot(2,4,4)
    plt.imshow(cv2.cvtColor(images[788], cv2.COLOR_BGR2RGB))
    plt.xlabel(' Normal ')
```

```
    plt.subplot(2,4,5)
    plt.imshow( operated_images[0],cmap='gray')
    plt.xlabel(f'Cataract After {label}')

    plt.subplot(2,4,6)
    plt.imshow(operated_images[8],cmap='gray')
    plt.xlabel(f'Cataract After {label}')

    plt.subplot(2,4,7)
    plt.imshow(operated_images[411],cmap='gray')
    plt.xlabel(f'  Normal After {label}')

    plt.subplot(2,4,8)
    plt.imshow(operated_images[788],cmap='gray')
    plt.xlabel(f'  Normal After {label}')


    plt.tight_layout()
grayscaled_images.clear()
for idx, image in enumerate(training_images):
    r,g,b = cv2.split(image)
#     g= cv2.equalizeHist(g)
    gray_image = cv2.normalize(src=g, dst=None, alpha=0, beta=255,
norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    grayscaled_images.append(gray_image)

vis(training_images,grayscaled_images,'GrayScale')

gray_histogram_of_images.clear()
for image in training_images:
    r,g,b = cv2.split(image)
    image_histogram = cv2.equalizeHist(g)
    gray_histogram_of_images.append(image_histogram)

vis(training_images,gray_histogram_of_images,'histogram_equal')

Clahe_images.clear()
num_white2 = []
```

```
for idx, image in enumerate(training_images):
    R, G, B = cv2.split(image)

    clahe = cv2.createCLAHE(clipLimit = 5)
    clahe_img = clahe.apply(G)
    Clahe_images.append(clahe_img)
    W = np.sum(image == 255)
    B = np.sum(image == 0)
    num_white2.append((W*1000)/(W+B) )

vis(training_images,Clahe_images,'Clahe')
dfwh = pd.DataFrame({' num_white2': num_white2})
canny_edged_images = []
num_white = []


for image in training_images:
    image = cv2.Canny(image,30,200)
    canny_edged_images.append(image)
    W = np.sum(image == 255)
    B = np.sum(image == 0)
    num_white.append(W/(W+B) )

vis(training_images,canny_edged_images,'Canny')
dfwhite = pd.DataFrame({' num_white': num_white})
ero_images = []
ero_images.clear()
kernel = np.ones((3,3),np.uint8)
for idx, image in enumerate(Clahe_images):

    erosion = cv2.erode(image,kernel,iterations = 1)
    ero_images.append(erosion)

vis(training_images,ero_images,'erosion')
dil_images = []
dil_images.clear()
kernel = np.ones((3,3),np.uint8)
for idx, image in enumerate(training_images):
    R, G, B = cv2.split(image)
```

```
    dilation = cv2.dilate(R,kernel,iterations = 1)
    dil_images.append(dilation)

# showSamples(dil_images, True)
vis(training_images,dil_images,'Dilation')
dl_mean=[]
dl_median=[]
dl_std_dev=[]

for idx, image in enumerate(dil_images):
    mean = np.mean(image)
    median = np.median(image)
    std_dev = np.std(image)

    dl_mean.append(mean)
    dl_median.append(median)
    dl_std_dev.append(std_dev)

dfdl = pd.DataFrame({'dl_mean':dl_mean})
dfdl_median = pd.DataFrame({'dl_std_dev':dl_median})
dfdl_std_dev = pd.DataFrame({'dl_std_dev':dl_std_dev})

i =  cv2.imread("C:/Users/Abhishek/Desktop/PRIMI
project/ODIR-5K/Pre_Diabetic/211_right.jpg")
i = cv2.cvtColor(i,cv2.COLOR_BGR2RGB)
r,g,b = cv2.split(i)
# mean = np.mean(g)
plt.imshow(g,cmap = "gray")
B_images = []
B_images.clear()
for idx, image in enumerate(training_images):

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    rectKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (15, 15))
    blackhat = cv2.morphologyEx(gray, cv2.MORPH_BLACKHAT, rectKernel)
#     blackhat = unsharp_mask(blackhat, radius=20, amount=5)
    B_images.append(blackhat)
vis(training_images, B_images,'B hat')

from skimage import data
```

```
from skimage.filters import unsharp_mask
import matplotlib.pyplot as plt

# image = B_images[788]
i =  cv2.imread("C:/Users/Abhishek/Desktop/PRIMI
project/ODIR-5K/Pre_Diabetic/211_right.jpg")
i = cv2.cvtColor(i,cv2.COLOR_BGR2RGB)
r,g,b = cv2.split(i)
rectKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (15, 15))
g = cv2.morphologyEx(g, cv2.MORPH_BLACKHAT, rectKernel)
# mean = np.mean(g)
plt.imshow(g,cmap = "gray")
result_1 = unsharp_mask(g, radius=1, amount=1)
result_2 = unsharp_mask(g, radius=5, amount=2)
result_3 = unsharp_mask(g, radius=10, amount=3)


fig, axes = plt.subplots(nrows=2, ncols=2,
                         sharex=True, sharey=True, figsize=(5, 5))
ax = axes.ravel()

ax[0].imshow(g, cmap=plt.cm.gray)
ax[0].set_title('Original image')
ax[1].imshow(result_1, cmap=plt.cm.gray)
ax[1].set_title('Enhanced image, r=1, a=1')
ax[2].imshow(result_2, cmap=plt.cm.gray)
ax[2].set_title('Enhanced image, r=5, a=2')
ax[3].imshow(result_3, cmap=plt.cm.gray)
ax[3].set_title('Enhanced image, r=20, a=3')

for a in ax:
    a.axis('off')
fig.tight_layout()
plt.show()
T_images = []
T_images.clear()
for idx, image in enumerate(training_images):

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    rectKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (13, 5))
```

```
    tophat = cv2.morphologyEx(image, cv2.MORPH_TOPHAT, rectKernel)
    tophat = cv2.cvtColor(tophat, cv2.COLOR_RGB2GRAY)

    T_images.append(tophat)

vis(training_images, T_images,'T hat')
T_images[1].shape
Energy = []
Correlation = []
Dissimilarity = []
Homogenity = []
Contrast = []


def energy(img):

            img = np.array(img)
            GLCM = greycomatrix(img, [1], [0])
            return greycoprops(GLCM, 'energy')[0]

def correlation(img):
#
            img = np.array(img)
            GLCM = greycomatrix(img, [1], [0])
            return greycoprops(GLCM, 'correlation')[0]

def dissimilarity(img):

            img = np.array(img)
            GLCM = greycomatrix(img, [1], [0])
            return  greycoprops(GLCM, 'dissimilarity')[0]

def homogenity(img):

            img = np.array(img)
            GLCM = greycomatrix(img, [1], [0])
            return  greycoprops(GLCM, 'homogeneity')[0]

def contrast(img):
```

```
            img = np.array(img)
            GLCM = greycomatrix(img, [1], [0])
            return greycoprops(GLCM, 'contrast')[0]


for img in training_images:
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    Energy.append(energy(img)[0])
    Correlation.append(correlation(img)[0])
    Dissimilarity.append(dissimilarity(img)[0])
    Homogenity.append(homogenity(img)[0])
    Contrast.append(contrast(img)[0])



dfe = pd.DataFrame({'Energy':Energy})
dfco = pd.DataFrame({'Correlation':Correlation})
dfd = pd.DataFrame({' Dissimilarity': Dissimilarity})
dfh = pd.DataFrame({'Homogenity':Homogenity})
dfc = pd.DataFrame({'Contrast':Contrast})
er_mean=[]
er_median=[]
er_std_dev=[]
for idx, image in enumerate(ero_images ):
    mean = np.mean(image)
    median = np.median(image)
    std_dev = np.std(image)

    er_mean.append(mean)
    er_median.append(median)
    er_std_dev.append(std_dev)

dfer= pd.DataFrame({'er_mean':er_mean})
dfer_median= pd.DataFrame({'er_median':er_median})
dfer_std_dev = pd.DataFrame({'er_std_dev':er_std_dev})
clahe_mean=[]
clahe_median=[]
clahe_std_dev=[]

clahe_mean.clear()
```

```
clahe_median.clear()
clahe_std_dev.clear()

for image in Clahe_images:
    mean = np.mean(image)
    median = np.median(image)
    std_dev = np.std(image)

    clahe_mean.append(mean)
    clahe_median.append(median)
    clahe_std_dev.append(std_dev)

dfcl= pd.DataFrame({'clahe_mean':clahe_mean})
dfclahe_median= pd.DataFrame({'clahe_median':clahe_median})
dfclahe_std_dev = pd.DataFrame({'clahe_std_dev':clahe_std_dev})
canny_edged_images = []


for idx, image in enumerate(training_images):
    mean = np.mean(image)
    median = np.median(image)
    std_dev = np.std(image)

    canny_mean.append(mean)
    canny_median.append(median)
    canny_std_dev.append(std_dev)

dfca= pd.DataFrame({'canny_mean':canny_mean})
dfconny_median= pd.DataFrame({'conny_median':canny_median})
dfconny_std_dev= pd.DataFrame({'canny_std_dev':canny_std_dev})
```

```
pdList = [dfwhite,dfer,dfer_median,dfer_std_dev,dfdl
          ,dfcl,dfe,dfc,dfd,dfco,dfh,df2]

new_df = pd.concat(pdList,axis=1)
new_df.head()
df = pd.DataFrame({
    'Classes': ["Cataract","Normal"],
    'Mean': [124.510662, 80.555983],
    'Median': [153.0, 69.0],
    "num_white*1000":[159.44,108.658]
})

# plotting graph
df.plot(x="Classes", y=["Mean", "Median","num_white*1000"], kind="bar")
c = new_df.iloc[:, :-1]
X = new_df.iloc[:, :-1].values
y = new_df.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, np.array(y),
test_size=0.4, random_state=42)
X_val, X_test1, y_val, y_test1 = train_test_split(X, np.array(y),
test_size=0.2, random_state=42)

stdSc = StandardScaler()
X_train = stdSc.fit_transform(X_train)

X_val = stdSc.transform(X_val)

X_train = np.asarray(X_train).astype('float32')
X_val = np.asarray(X_val).astype('float32')
y_train = np.asarray(y_train).astype('float32')
y_val = np.asarray(y_val).astype('float32')

X_test1 = np.asarray(X_test).astype('float32')
y_test1 = np.asarray(y_test).astype('float32')

from sklearn.linear_model import LogisticRegression
```

```
LR = LogisticRegression(penalty= 'l2',random_state=0).fit(X_train,
y_train)
# LR.fit(X_train, y_train)
y_pred = LR.predict(X_test);
LRaccuracy = round(accuracy_score(y_test, y_pred) * 100, 2)
print(accuracy)
```

```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier(max_depth=5,n_estimators=70 ,criterion='gini'
, random_state=0)
RF.fit(X_train, y_train)
y_pred = RF.predict(X_val);
RFaccuracy = round(accuracy_score(y_val, y_pred) * 100, 2)
print(accuracy)
from sklearn.svm import SVC
SV = SVC(C=1)
SV.fit(X_train, y_train)
y_pred = SV.predict(X_val);
SVaccuracy = round(accuracy_score(y_val, y_pred) * 100, 2)
SVaccuracy
from sklearn.tree import DecisionTreeClassifier
DT = DecisionTreeClassifier(max_depth= 4,random_state=0)
DT.fit(X_train, y_train)
y_pred = DT.predict(X_test);
DTaccuracy = round(accuracy_score(y_test, y_pred) * 100, 2)
print(DTaccuracy)
from sklearn.neighbors import KNeighborsClassifier
NN = KNeighborsClassifier(n_neighbors=3,metric = "minkowski")
NN.fit(X_train,y_train)
y_pred = NN.predict(X_test);
NNaccuracy = round(accuracy_score(y_test, y_pred) * 100, 2)
print(NNaccuracy)
import numpy as np
import matplotlib.pyplot as plt
# Dataset generation
```

```
data_dict = {'LR':LRaccuracy, 'RF':RFaccuracy, 'SV':SVaccuracy
,'DT':DTaccuracy, 'NN':NNaccuracy}
courses = list(data_dict.keys())
values = list(data_dict.values())
fig = plt.figure(figsize = (5, 5))
#  Bar plot
plt.bar(courses, values, color ='red',
        width = 0.5)
plt.xlabel("Classifiers")
plt.ylabel("Accuracy")
plt.title("Accuracy of Different Classifder")
plt.show()
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

print("True Negative =",tn)
print("False Positive =",fp)
print("False Negative =",fn)
print("True Positive =",tp)
classifier = Sequential()

layer_info = Dense(activation='relu', input_dim=11, units=50)
classifier.add(layer_info)

# layer_info = Dense(activation='relu', units=6)
# classifier.add(layer_info)

layer_info = Dense(activation='relu', units=30)
classifier.add(layer_info)

layer_info = Dense(activation='sigmoid',units=1)
classifier.add(layer_info)

classifier.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

classifier.fit(X_train, y_train, batch_size=50, epochs=570)

flag_prediction = classifier.predict(X_test).round()

tn, fp, fn, tp = confusion_matrix(y_test, flag_prediction).ravel()
```

```
print("True Negative =",tn)
print("False Positive =",fp)
print("False Negative =",fn)
print("True Positive =",tp)


print(confusion_matrix(y_test, flag_prediction))
print(accuracy_score(y_test, flag_prediction)*100)
```

VGG16 built from Scratch and through Transfer Learning:

```
#Importing required libraries
import os
import seaborn as sns
import pandas as pd
import numpy as np
import cv2 as cv
from tqdm import tqdm
import random
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img,img_to_array
from tensorflow.keras.layers import Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.utils import to_categorical
from sklearn import metrics
import matplotlib.pyplot as plt

os.environ['KAGGLE_USERNAME'] = "surajrbhute"
os.environ['KAGGLE_KEY'] = "36b5cb5030f3f6736f71941d4fff546e"
!kaggle datasets download -d andrewmvd/ocular-disease-recognition-odir5k

# unzipping the dataset file
```

```
!unzip ocular-disease-recognition-odir5k.zip

# Reading the csv file using pandas
dataset = pd.read_csv("full_df.csv")
dataset
dataset.head(10)
# Show value counts for a single categorical variable
sns.countplot(x=dataset['labels'], data=dataset)
sns.countplot(x=dataset['Patient Sex'], data=dataset)
# The hist() function will read the array and produce a histogram:
plt.figure(figsize= (14,7))
plt.hist(dataset['Patient Age'])

def has_normal_fundus(text):
    if "normal fundus" in text:
        return 1
    else:
        return 0

dataset['left_normal'] = dataset['Left-Diagnostic Keywords'].apply(lambda
x: has_normal_fundus(x))
dataset['right_normal'] = dataset['Right-Diagnostic
Keywords'].apply(lambda x: has_normal_fundus(x))

left_normal = dataset.loc[(dataset.N == 1) & (dataset['left_normal'] ==
1)]["Left-Fundus"].values
right_normal = dataset.loc[(dataset.N == 1) & (dataset["right_normal"] ==
1)]["Right-Fundus"].values
normal = np.concatenate((left_normal, right_normal), axis=0)

def has_cataract(text):
    if "cataract" in text:
        return 2
    else:
        return 0

dataset['left_cataract'] = dataset['Left-Diagnostic
Keywords'].apply(lambda x: has_cataract(x))
dataset['right_cataract'] = dataset['Right-Diagnostic
Keywords'].apply(lambda x: has_cataract(x))
```

```
left_cataract = dataset.loc[(dataset.C == 1) & (dataset['left_cataract']
== 2)]['Left-Fundus'].values
right_cataract = dataset.loc[(dataset.C == 1) & (dataset['right_cataract']
== 2)]['Right-Fundus'].values
cataract = np.concatenate((left_cataract, right_cataract), axis=0)

def has_retinopathy(text):
  if ('mild nonproliferative retinopathy') or ('proliferative
retinopathy') or ('diabetic retinopathy') or ('moderate non proliferative
retinopathy') in text:
    return 3
  else:
    return 0

dataset['left_diabetes'] = dataset['Left-Diagnostic
Keywords'].apply(lambda x: has_retinopathy(x))
dataset['right_diabetes'] = dataset['Right-Diagnostic
Keywords'].apply(lambda x: has_retinopathy(x))

left_diabetes = dataset.loc[(dataset.D == 1) & (dataset['left_diabetes']
== 3)]['Left-Fundus'].values
right_diabetes = dataset.loc[(dataset.D == 1) & (dataset['right_diabetes']
== 3)]['Right-Fundus'].values
diabetes = np.concatenate((left_diabetes, right_diabetes), axis=0)

def has_glaucoma(text):
  if 'glaucoma' in text:
    return 4
  else:
    return 0

dataset['left_glaucoma'] = dataset['Left-Diagnostic
Keywords'].apply(lambda x: has_glaucoma(x))
dataset['right_glaucoma'] = dataset['Left-Diagnostic
Keywords'].apply(lambda x: has_glaucoma(x))

left_glaucoma = dataset.loc[(dataset.G == 1) & (dataset['left_glaucoma']
== 4)]['Left-Fundus'].values
```

```
right_glaucoma = dataset.loc[(dataset.G == 1) & (dataset['right_glaucoma']
== 4)]['Right-Fundus'].values
glaucoma = np.concatenate((left_glaucoma, right_glaucoma), axis=0)

print(len(cataract), len(normal), len(glaucoma), len(diabetes))

dataset_dir = "/content/preprocessed_images/"
image_size = 224
labels = []
ds = []
def create_dataset(image_category,label):
    for img in tqdm(image_category):
        image_path = os.path.join(dataset_dir,img)
        try:
            image = cv.imread(image_path,cv.IMREAD_COLOR)
            image = cv.resize(image,(image_size,image_size))
        except:
            continue

        ds.append([np.array(image), np.array(label)])
    random.shuffle(ds)
    return ds
ds = create_dataset(cataract, 1)
ds = create_dataset(glaucoma, 3)
ds = create_dataset(normal, 0)
ds = create_dataset(diabetes, 2)

len(ds)

# Shape of the new dataset
np.shape(ds)

# Taking a look at new dataset
ds

plt.figure(figsize=(12,7))
for i in range(10):
    sample = random.choice(range(len(ds)))
    image = ds[sample][0]
    label = ds[sample][1]
```

```
    if label == 0:
      label = "Normal"
    elif label == 1:
      label = "Cataract"
    elif label == 2:
      label = "Diabetes"
    elif label == 3:
      label = "Glaucoma"
    plt.subplot(2,5,i+1)
    plt.imshow(image)
    plt.xlabel(label)
plt.tight_layout()

# Because of the way the labels were created, it's better to categorize
them to prevent deceiving the model
x = np.array([i[0] for i in ds]).reshape(-1,image_size,image_size,3)
y = to_categorical([i[1] for i in ds])

# Splitting the dataset into train, validation and test sets

from sklearn.model_selection import train_test_split
X_train,X_val,Y_train,Y_val = train_test_split(x,y,test_size=0.2)
X_val,X_test,Y_val,Y_test = train_test_split(X_val, Y_val, test_size=0.5)

print(X_train.shape)
print(X_test.shape)
print(X_val.shape)
print(Y_train.shape)
print(Y_test.shape)
print(Y_val.shape)
```

Code for Model built from Scratch

```
import keras,os
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten
from keras.preprocessing.image import ImageDataGenerator
import numpy as np

from tensorflow.keras.layers import Flatten
```

```
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Model

model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padd
ing="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Flatten())
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=4, activation="softmax"))
```

```
from tensorflow.keras.optimizers import Adam
opt = Adam(lr=0.00001)
model.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy,
metrics=['accuracy'])
model.summary()
```

Code for Model using Transfer Learning:

```
# Importing a trained model.

from tensorflow.keras.applications.vgg16 import VGG16
vgg = tf.keras.applications.VGG16(
    include_top=False,
    weights="imagenet",
    input_shape= (224,224,3)
)
vgg.trainable = False

from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Model
x = Flatten()(vgg.output)
dense1 = Dense(units=128,activation='relu')(x)
dense2 = Dense(units=128, activation='relu')(dense1)
output = Dense(units=4, activation='softmax')(dense2)
A = Model(inputs = vgg.input, outputs = output)
A.summary()

optim = tf.keras.optimizers.Adam(learning_rate = 0.001)
A.compile(optimizer= optim, loss= 'categorical_crossentropy',
metrics=['accuracy'])

history = A.fit(x=X_train, y=Y_train, batch_size=128, epochs=40,
validation_data=(X_val, Y_val))

accuracy = A.evaluate(X_test, Y_test)

# printing history of loss
print(history.history['loss'])
```

```
# printing history of accuracy
print(history.history['accuracy'])

# printing history of validation loss
print(history.history['val_loss'])

# printing history of validation accuracy
print(history.history['val_accuracy'])

# Plotting the accuracy of model
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('accuracy of model')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# Plotting the loss of model
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('loss of model')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

Y_pred = A.predict(X_test)
print(Y_pred)

Y_pred = np.argmax(Y_pred, axis=1)
Y_test = np.argmax(Y_test, axis=1)

cm = confusion_matrix(Y_test, Y_pred, labels= [0,1,2,3])

def plot_confusion_matrix(cm, classes,
                          normalize=True,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
```

```
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    import itertools
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

plt.figure(figsize=(12,8))
plot_confusion_matrix(cm, classes=['Normal', 'Cataract', 'Diabetes',
'Glaucoma'], title='Confusion matrix, normalization')

from sklearn.metrics import classification_report
print(classification_report(Y_test, Y_pred, target_names=['Normal',
'Cataract', 'Diabetes', 'Glaucoma']))

accuracy_score(Y_test, Y_pred, normalize=True)
```

```
plt.figure(figsize=(12,10))
for i in range(10):
    sample = random.choice(range(len(X_test)))
    image = X_test[sample]
    category = Y_test[sample]
    pred_category = Y_pred[sample]

    if category== 0:
        label = "Normal"
    elif category == 1:
        label = "Cataract"
    elif category == 2:
        label = "Diabetes"
    elif category == 3:
        label = "Glaucoma"

    if pred_category== 0:
        pred_label = "Normal"
    elif pred_category == 1:
        pred_label = "Cataract"
    elif pred_category == 2:
        pred_label = "Diabetes"
    elif pred_category == 3:
        pred_label = "Glaucoma"

    plt.subplot(4,5,i+1)
    plt.imshow(cv.cvtColor(image , cv.COLOR_BGR2RGB))
    plt.xlabel("Actual:{}\nPrediction:{}".format(label,pred_label))
plt.tight_layout()

# Loading in new image
import cv2
import numpy as np
new_image = cv2.imread('/content/0_left.jpg')
new_image = cv2.cvtColor (new_image, cv2.COLOR_BGR2RGB)
new_image = cv2.resize(new_image, (224,224))

from google.colab.patches import cv2_imshow
cv2_imshow(new_image)
```

```
# Running model on new image

copy_new_image = np.array(new_image).reshape(-1,224,224,3)
new_prediction = A.predict(copy_new_image)

np = np.argmax(new_prediction, axis=1)

result = ""
if np == [0]:
  result = "Normal"
elif np == [1]:
  result = "Cataract"
elif np ==[2]:
  result = "Diabetes "

elif np==[3]:
  result = "Glaucoma "



print(result)

# Printing out prediction
np


# save it as a h5 file


from tensorflow.keras.models import load_model

A.save('model.h5')
```