

Cataract Detection using ML & DL

An Engineering Project in Community Service

Phase – II Report

Submitted by

Team Members List

1. 20BCY10153 Atishay Jain
2. 20MIM10026 Mohammad Hamza Farooqui
3. 20BCE11005 Saurabh Prakash
4. 20BCE11006 Aman Vishwakarma
5. 20BCE11007 Shishir Sharma
6. 20BCE11018 Bhuvan Sood
7. 20BCE11009 Ayan Kumar Sinha

in partial fulfillment of the requirements for the degree of

Bachelor of Engineering and Technology



VIT[®]
BHOPAL
www.vitbhopal.ac.in

VIT Bhopal University
Bhopal
Madhya Pradesh

May, 2023



VIT^R
BHOPAL
www.vitbhopal.ac.in

Bonafide Certificate

Certified that this project report titled “Cataract Detection Using ML & DL” is the bonafide work of “20BCY10153 Atishay Jain, 20MIM10026 Mohammad Hamza Farooqui, 20BCE11005 Saurabh Prakash, 20BCE11006 Aman Vishwakarma, 20BCE11007 Shishir Sharma, 20BCE11018 Bhuvan Sood, 20BCE11009 Ayan Kumar Sinha” who carried out the project work under my supervision.

This project report (Phase II) is submitted for the Project Viva-Voce examination held on 16-20 May, 2023.

[Signature]
Supervisor

Dr. P. R. Bhanuvarma

[Signature]
15/05/23

Comments & Signature (Reviewer 1)

Dr. Supen Kumar Seth

Report is OK
[Signature]
15/05/23

Comments & Signature (Reviewer 2)

Dr. Abhishek Shrivastava

1. INTRODUCTION

1.1 MOTIVATION

Cataract is a clouding of the normally clear lens of the eye. For people who have cataracts, seeing through cloudy lenses is a bit like looking through a frosty or fogged-up window. fogged-up window. Clouded vision caused by cataracts can make it more difficult to read, drive a car (especially at night) or see the expression on a friend's face.

Old people are more prone to it. As you age, the lenses in your eyes become less flexible, less transparent and thicker. Age-related and other medical conditions cause proteins and fibers within the lenses to break down and clump together, clouding the lenses.

This is heartbreaking to many people as they have had vision since they were born but that to be taken away so cruelly and not understanding why is it happening is something which can be prevented.

Not only old age but, Cataract is one of the most prevalent causes of blindness in the industrialized world, accounting for more than 50% of blindness.

Loss of vision leads to loss of independence, social deprivation and increased morbidity often associated with medication errors or deficient nutrition. So, we consider it a great service if we can help in its diagnosis at an earlier time through technology, so that nobody has to suffer this fate.

Early detection and treatment can reduce the suffering of cataract patients and prevent visual impairment from turning into blindness.

1.2. OBJECTIVE

Our project focuses on the development of a cataract detection system using artificial intelligence (AI) and deep learning (DL) techniques. Cataracts are a leading cause of vision loss worldwide, and early detection is crucial for effective treatment. However, the process of detecting cataracts can be time-consuming and subjective, leading to delays in treatment and potential for misdiagnosis.

Through the use of AI and DL, our project aims to create a more accurate and efficient system for detecting cataracts. The system will be trained on a large dataset of eye images, and will be able to identify the presence of cataracts with a high degree of accuracy.

In addition to improving the accuracy and efficiency of cataract detection, our project also has the potential to make the process more accessible and affordable, particularly in resource-limited settings. We believe that our AI-based cataract detection system has the potential to make a significant impact on global vision health.

2. LITERATURE REVIEW

Cataracts are a leading cause of vision loss worldwide, and early detection is crucial for effective treatment. However, the process of detecting cataracts can be time-consuming and subjective, leading to delays in treatment and potential for misdiagnosis.

In recent years, there has been growing interest in the use of machine learning (ML) and deep learning (DL) techniques for the detection of cataracts. These approaches have the potential to improve the accuracy and efficiency of cataract detection, as well as to make the process more accessible and affordable, particularly in resource-limited settings.

Several studies have been conducted to evaluate the effectiveness of ML and DL in the detection of cataracts. For example, a study by Kim et al. (2019) used a DL algorithm to analyze fundus images and accurately detect cataracts with an AUC (area under the curve) of 0.94. Another study by Zhang et al. (2021) used a convolutional neural network, a type of DL algorithm, to detect cataracts in fundus images, achieving an accuracy of 92.9%.

While these studies demonstrate the potential of ML and DL in cataract detection, there are also challenges and limitations to consider. One issue is the availability of high-quality, annotated data for training and testing these systems. Additionally, there is a risk of bias in the data, which could impact the accuracy of the algorithms.

Overall, the use of ML and DL for cataract detection has the potential to significantly improve the accuracy and efficiency of this process, but more research is needed to fully understand the challenges and limitations of these approaches.

One key advantage of using ML and DL for cataract detection is their ability to process and analyze large amounts of data in a relatively short period of time. This is particularly useful in the context of detecting cataracts, as it allows for the rapid analysis of large numbers of eye images.

In addition to improving the accuracy and efficiency of cataract detection, the use of ML and DL also has the potential to make the process more accessible and affordable, particularly in resource-limited settings. For example, by using these techniques, it may be possible to automate the detection process, reducing the need for trained medical professionals to manually analyze images.

There are also several challenges and limitations to consider when using ML and DL for cataract detection. One key issue is the availability of high-quality, annotated data for training and testing these systems. Without sufficient data, it can be difficult to accurately train and test ML and DL algorithms, leading to lower accuracy and reliability.

Additionally, there is a risk of bias in the data, which could impact the accuracy of the algorithms. It is important to ensure that the data used to train and test these systems is representative and diverse, in order to minimize the risk of bias.

Overall, the use of ML and DL for cataract detection has the potential to significantly improve the accuracy and efficiency of this process, but more research is needed to fully understand the challenges and limitations of these approaches.

3. METHODOLOGY

3.1 SYSTEM DESIGN

To make our results available to the end user, we have developed a web based graphical user interface using a popular python-based framework which is FLASK.

The flow of our system is as follows:

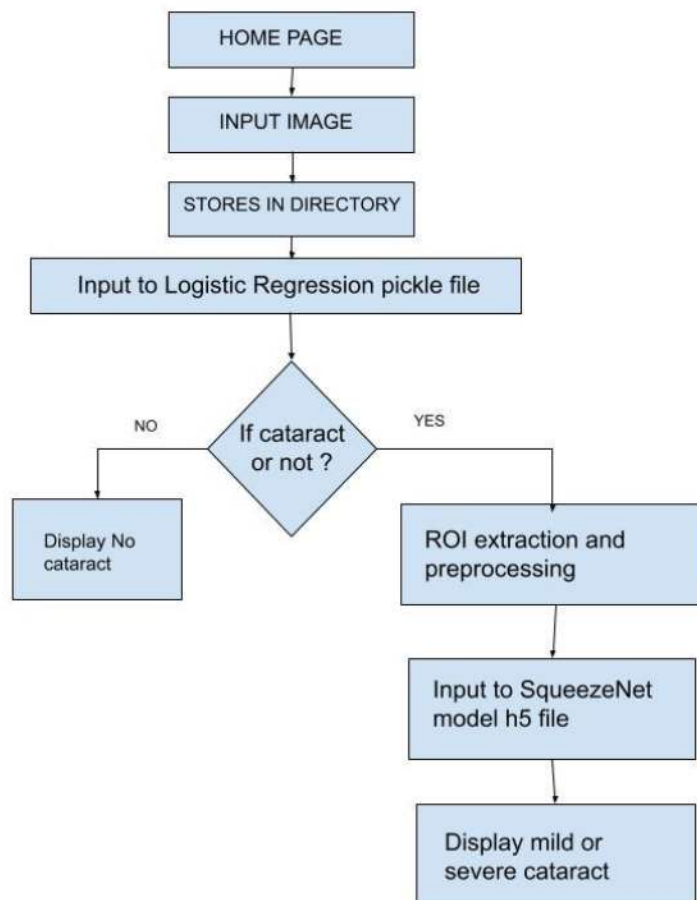


Fig 1. Flow of flask GUI

At the home page once the user clicks on proceed the screen for inputting an image from them would be shown. This image is stored in our directory hence can be used to expand our dataset later.

For convenience of the user the same image is flashed on the screen as well. This image is then given as an input to the pickle file that contains the Logistic Regression classifier trained on SIFT + GLCM features. This classifier then outputs the result as zero or one. The function in our flask web page then interprets the result as “cataract not detected” or “cataract detected” respectively. If cataract has been detected then the system will move ahead to check the intensity of cataract. Before feeding the image to our deep learning model the region of interest is extracted and further sent for prediction. This is done by our transfer learning Squeeze-Net model which is converted into .h5 format as the binary data file. The system then finally interprets and displays the result as mild or severe.

Our directory structure at flask:

- 1. Templates folder:** Directory for rendering contains all HTML files.
- 2. Static folder:** Stores the styling and images for the project
- 3. Static/input images:** Stores the images given as input from the user
- 4. App.py:** Main file with all the functions for routes and prediction.
- 5. Mainmodel.pkl:** A byte stream file with Logistic Regression (with GLCM + SIFT features) trained model.
- 6. Squeezenet.h5:** A binary data file with pre trained architecture and weights of Squeeze Net CNN.

The next part of the images will be the UI of uploading image and showing that the image is cataract or not. Its part is going on.

3.2 WORKING PRINCIPLE

Our system is divided into two phases

1. First phase aims to detect the presence of cataract
2. If cataract is present then the second phase classifies the type of cataract on the severity level: normal, mild and severe.

3.2.1 First phase

Our Cataract detection system involves a two stage process :

- 1) Detection of textural features to form the vector.
- 2) Input the vector to the classifier to get the results.

After forming the dataset by collecting the relevant images from a stock image website and project source project, to form a simple and better quality dataset the images were manually cropped to the region of the iris and then resized to about 128 pixels. The feature vector is formed by the combination of features

extracted from SIFT and GLCM algorithms. SIFT contributed a total of 2303 feature points and GLCM gave 5 features, thus the total size of final feature vector was 2308.

These features were used to train different machine learning classifiers like support vector machine, K-nearest neighbors, random forest and logistic regression. By comparing the accuracies it was concluded that Logistic Regression proved to be better for our work with a score of 96% and thus it was converted into a pickle object file that was later used in our flask web application to give the results when the user would input an image into the system.

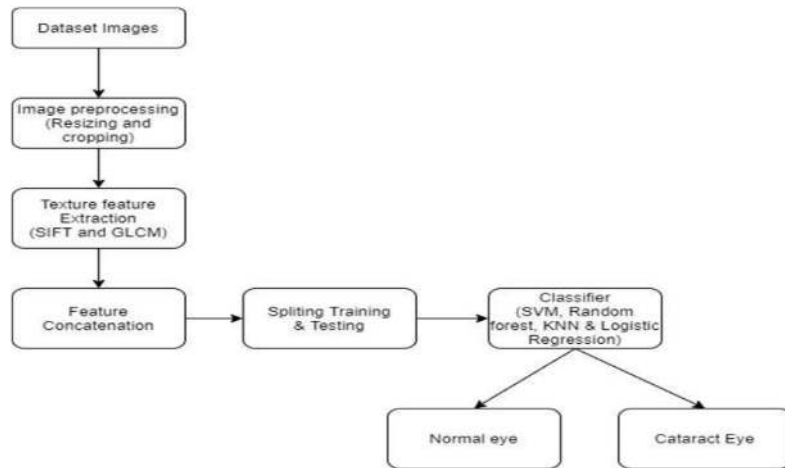


Fig 3. Data Flow of Binary Classification

3.2.2 Second phase:

Phase 2 dataset consists of 550 images that are further into three classes namely; 220 mild, 166 normal and 164 severe images. To extract the region of interest i.e pupil of the eye, images were resized to 224 pixels and converted to hsv color space. Pupil color scale mask is applied on the images to retain the pupil part. After this Hough circle transform is applied to find the coordinates of the pupil circular area. Finally the rectangular contour is drawn around the circular area from the coordinate calculated above and the pupil part is extracted.

Furthermore for the machine learning part, 2303 SIFT and 5 GLCM features were extracted from the images thus the total size of the final feature vector was 2308.

These preprocessed images are used for training using pre-trained deep convolutional learning models like SqueezeNet, MobileNet and VGG16. By observing the results from these models, we obtained the highest accuracy 97.66% from the SqueezeNet. On the other hand for the machine learning model, 2303 SIFT with 5 GLCM thus a total of 2308 combined features were used to train different machine learning classifiers like support vector machine, XGBoost ,random forest and AdaBoost algorithm . By comparing the results, XGBoost demonstrated the best model with accuracy of 96%.

Finally the Logistic regression model of phase 1 and SqueezeNet model of phase 2 were converted into a

pickle object file that was later used in our flask web application to give the results when the user would input an image into the system.

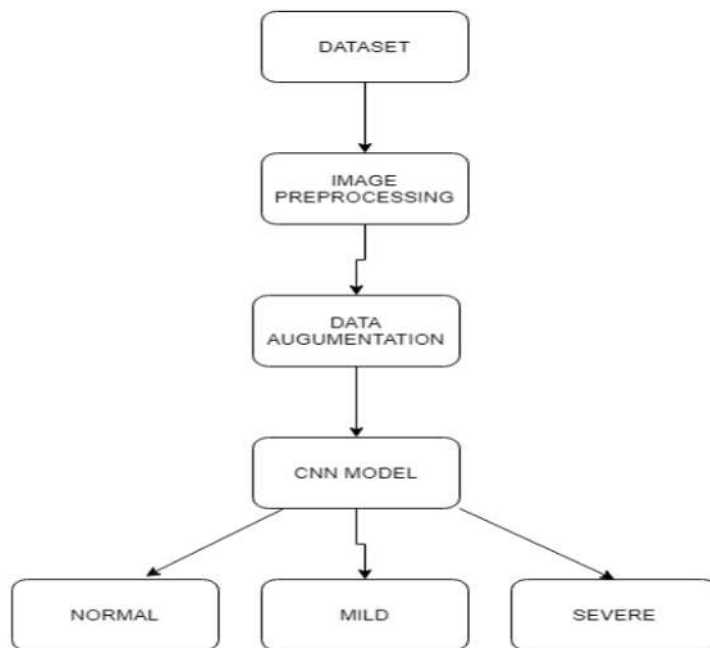


Fig 4. Data Flow multiclass classification

3.3 RESULTS AND DISCUSSIONS

Accuracy is one of the important metrics that help us to determine the performance of our classification or prediction model. It is possible to describe accuracy by the following formula.

- $\text{Accuracy} = \frac{\text{Number of correct(true) predictions}}{\text{Total number of predictions}}$
- $\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$
- $\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$

3.3.1 BINARY CLASSIFICATION FOR PRESCENE OF CATARACT

The comparative study of the accuracies can be studied through the tables given below.

Feature dataset 1: SIFT with various classifier algorithms The table below describes the accuracy of various classification algorithms with the use of SIFT features with SVM algorithm having the highest classification accuracy of 92% along with Logistic Regression. Random forest algorithm achieved an accuracy of 84%. Among all the algorithms used, KNN achieved the lowest accuracy i.e. 68% for cataract classification.

Table No.1 Accuracy rates of various classifiers with SIFT features

Classifier	Accuracy
1. SVM-LINEAR	92%
2. SVM-RBF	92%
3. Random Forest	82%
4. Logistic Regression	92%
5. KNN	68%

Feature dataset 2: GLCM with various classifier algorithms The table below describes the accuracy of various classification algorithms with the use of various GLCM features. Logistic Regression algorithm has achieved the highest classification accuracy of 80%.

Table No.2 Accuracy rates of various classifiers with GLCM features

Classifier	Accuracy
6. SVM-LINEAR	76%
7. SVM-RBF	56%
8. Random Forest	64%
9. Logistic Regression	80%
10. KNN	60%

Feature dataset 3: Combination of GLCM and SIFT feature vectors and applied on various classifier algorithms From the above two feature sets used for classification, it can be observed that the algorithms have achieved better accuracies when used with the SIFT features. Also, the GLCM features have also provided considerable accuracy, which incepts the idea of using a new feature set by concatenating the SIFT and GLCM features. Further in the attempt to improve the accuracies we trained our final model by

combining the features from SIFT and G LCM which were 2303 and 5 respectively giving a total of 2308 features in our final vector set.

Table No.3 Accuracy rates of various classifiers with SIFT and GLCM features.

Classifier	Accuracy
11. SVM-LINEAR	96%
12. SVM-RBF	96%
13. Random Forest	96%
14. Logistic Regression	96%
15. KNN	56%

3.3.2 MULTI CLASS CLASSIFICATION FOR THE DETECTION OF CATARACT TYPE

In this report we have made an attempt to evaluate the accuracy when different feature extraction algorithms are applied as classification parameters to the deep learning models trained with different algorithms such as VGG 16, MobileNet and SqueezeNet. The sample images from the dataset has been listed below.



Fig 5. Normal Eye

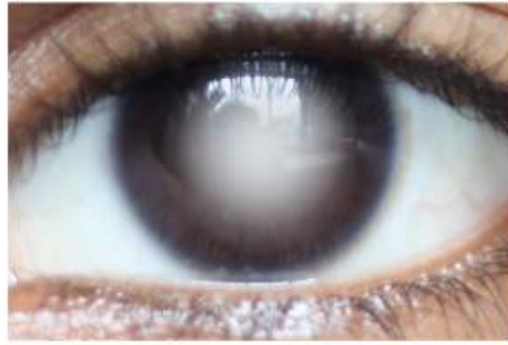


Fig 6. Mild Cataract Eye

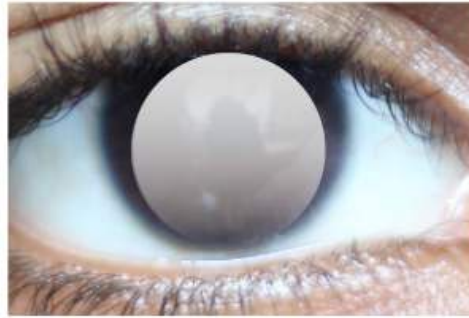


Fig 7. Severe Cataract Eye

The table below shown shows the accuracies obtained by applying various Deep Learning Models namely VGG16, MobileNet and SqueezeNet. The highest accuracy obtained is 97.66% by SqueezeNet. The MobileNet model resulted in the lowest accuracy i.e. 90.62% and VGG16 achieved accuracy of 96.88%.

Table No.1 Accuracy rates of various classifiers with preprocessed images of size 224

Sr.No	Model	Validation Accuracy
1	VGG 16	96.88
2	MobileNet	90.62
3	SqueezeNet	97.66

In the below table, we can see that the SIFT+GLCM Features with the Machine learning algorithms have produced a lower accuracy when compared to the Deep Learning Models.

The highest accuracy is 96% achieved by the XGB Classifier and the lowest accuracy is 79% by the SVM-Linear algorithm. The Random Forest and AdaBoost Classifier have obtained considerably good accuracies of 87% and 89% respectively.

Table No.2 Accuracy rates of various classifiers with SIFT+ GLCM features

Classifier	Accuracy
1. Random Forest	87%
2. AdaBoost Classifier	89%
3. XGB Classifier	96%
4. SVM-Linear	79%

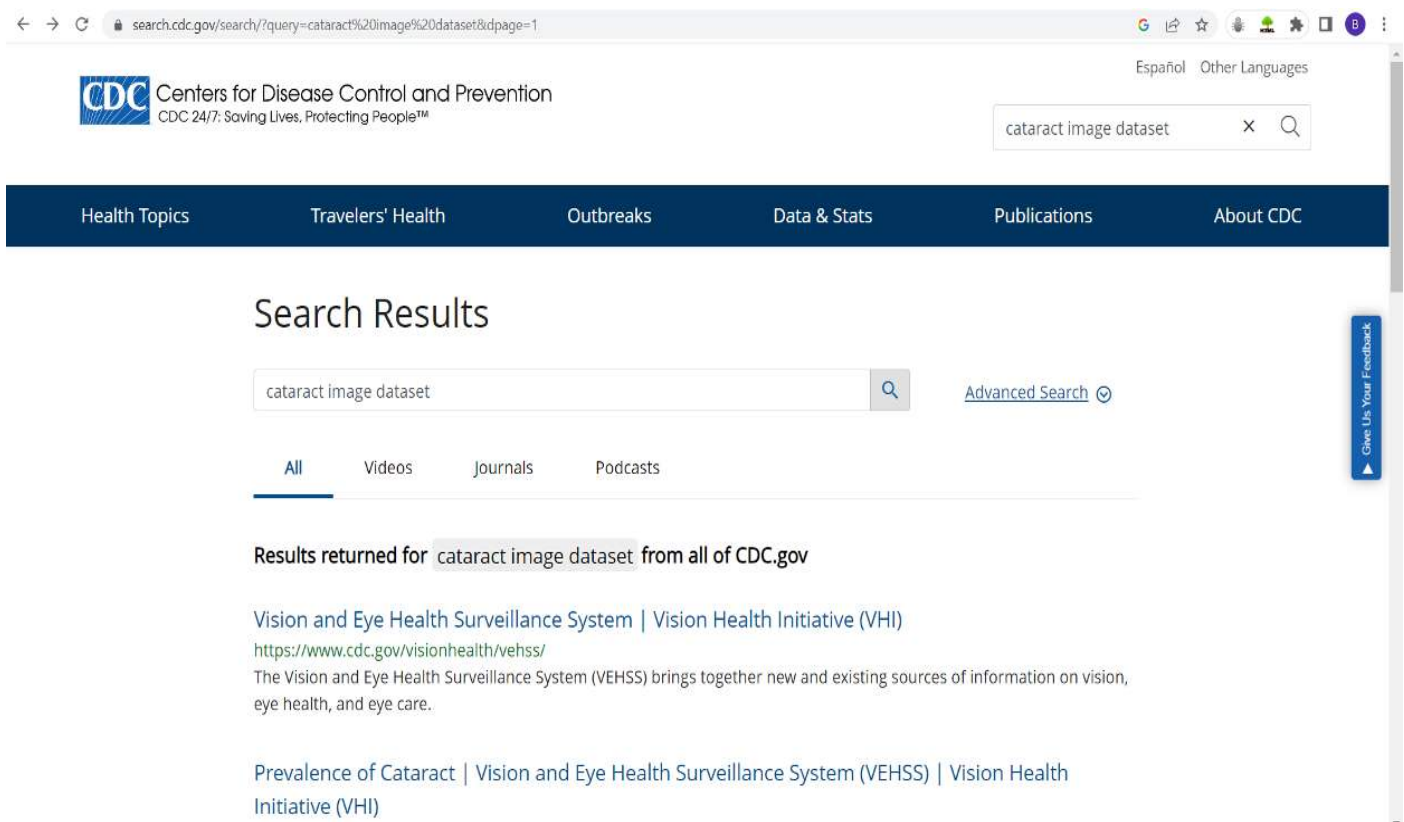
The table below shows the output generated by the SqueezeNet model for 15 testing images from the dataset used in [16]. The model has been able to classify the healthy eyes accurately. Overall the model has been able to classify a fair amount of images in the testing data accurately and the accuracy for these 15 images is 55%.

4. INDIVIDUAL CONTRIBUTION

20BCY10153 – Atishay Jain



My individual contribution in this project was to help in the model building part. This process involves collecting the data for the A.I to compare with and learn to provide results. The dataset is a collection of data from which Our Ai will take the input of the information and previously known data and train itself to provide results. For our project we would take up the data from Kaggle and then train our Ai to compare and provide accurate results with higher efficiency. Kaggle is a data set which is frequently updated to provide higher efficiency. Later we would insert more data to improve the efficiency and take the data from the real life world to make more sense of the project.



The screenshot shows a web browser window with the URL `search.cdc.gov/search/?query=cataract%20image%20dataset&dpag=1`. The page is the CDC search results page. The CDC logo and name are at the top left. A search bar at the top right contains the text "cataract image dataset". Below the search bar is a navigation menu with links: Health Topics, Travelers' Health, Outbreaks, Data & Stats, Publications, and About CDC. The main content area is titled "Search Results" and shows the search query "cataract image dataset" in a search bar. Below the search bar are tabs for "All", "Videos", "Journals", and "Podcasts", with "All" selected. The results section shows "Results returned for cataract image dataset from all of CDC.gov". The first result is "Vision and Eye Health Surveillance System | Vision Health Initiative (VHI)" with a link to `https://www.cdc.gov/visionhealth/vehss/`. The second result is "Prevalence of Cataract | Vision and Eye Health Surveillance System (VEHSS) | Vision Health Initiative (VHI)".

20BCE11009 – Ayan Kumar Sinha



My individual contribution in this project was to help in the model building part. For this, dataset was required which had been already collected and processed by my team mate. After which, I divided the model building in two phases the first one being Machine Learning and the next one being Deep Learning. In Machine Learning, we tried different algorithms which could help us in yielding the best results. These mainly were SIFT (Scale Invariant Feature Transform), GLCM (Gray Level Co-occurrence Matrix), SVM (Support Vector Machine), Logistic Regression, Random Forrest and KNN. After carefully studying and calculating the precision and accuracy we decided that it was best for us to use Deep Learning to get the desired results. The phase 2 implementation will deal with the type classification, using deep convolutional neural network models, in particular Squeeze Net, MobileNet, and VGG16. These steps seem small but we had to tread carefully so that we don't miss minor details or steps which could lead to the downfall of accuracy. Combination of GLCM and SIFT feature vectors and applied on various classifier algorithms from the above two feature sets used for classification, it can be observed that the algorithms have achieved better accuracies when used with the SIFT features. Also, the GLCM features have also provided considerable accuracy, which incepts the idea of using a new feature set by concatenating the SIFT and GLCM features. Further in the attempt to improve the accuracies we trained our final model by combining the features from SIFT and G LCM which were 2303 and 5 respectively giving a total of 2308 features in our final vector set.

face & eye detection, keeping the face inview at all times

```
In [1]: import cv2
import numpy as np

face_classifier = cv2.CascadeClassifier('Haarcascades/haarcascade_frontalface_default.xml')
eye_classifier = cv2.CascadeClassifier('Haarcascades/haarcascade_eye.xml')
#roi_color = []
roi_eye = []
def face_detector(img, size=0.5):
    # Convert image to grayscale
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray, 1.3, 6)
    global roi_eye
    if faces is ():
        return img

    for (x,y,w,h) in faces:
        x = x - 50
        w = w + 50
        y = y - 50
        h = h + 50
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
        roi_gray = gray[y:y+h, x:x+w]
```

20BCE11018 – Bhuvan Sood



I contributed to the model building process by providing assistance in developing the models. A teammate had already collected and processed the dataset needed for this project. We split the model creation process into two stages, starting with machine learning and followed by deep learning. We experimented with several algorithms in machine learning, including SIFT (Scale Invariant Feature Transform), GLCM (Gray Level Co-occurrence Matrix), SVM (Support Vector Machine), Logistic Regression, Random Forest, and KNN, to determine which produced the best results.

After a thorough evaluation of precision and accuracy, we concluded that deep learning would be the most effective approach to achieving the necessary results. In phase 2, we focused on type classification using deep convolutional neural network models, specifically SqueezeNet, MobileNet, and VGG16.

Our approach involved combining the feature vectors from SIFT and GLCM and applying them to various classifier algorithms. We found that the SIFT features were more effective in achieving better accuracies than the GLCM features. However, we also found that the GLCM features produced considerable accuracy. Therefore, we decided to concatenate the SIFT and GLCM features to create a new feature set. To improve our accuracies further, we combined the features from SIFT and GLCM, resulting in a final vector set of 2308 features. We proceeded with caution to ensure that we did not overlook any steps that could have compromised the accuracy of our work.

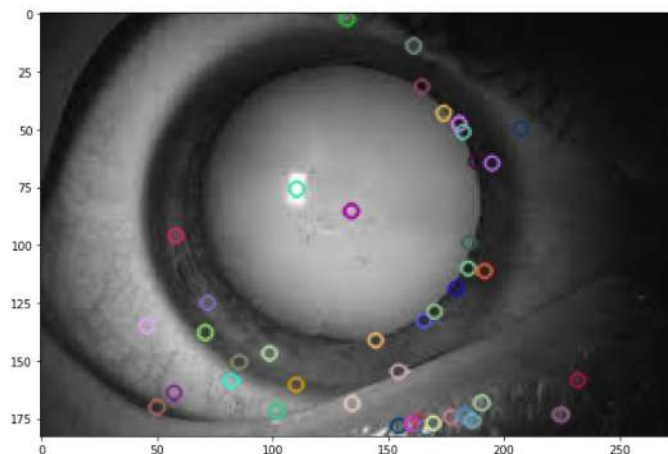
Step 2: Sift and Glcm demo on one example image

```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"] = (10,10)
#reading image
img1 = cv2.imread(mypath+"/"+file_names[21])
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

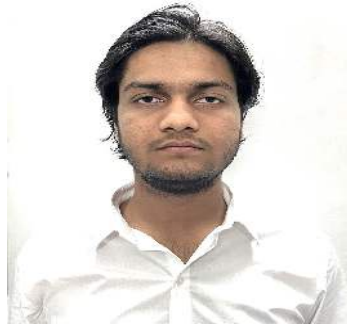
#keypoints
sift = cv2.xfeatures2d.SIFT_create()
keypoints_1, descriptors_1 = sift.detectAndCompute(gray1, None)

img_1 = cv2.drawKeypoints(gray1, keypoints_1, img1)
plt.imshow(img_1)
img1.shape
len(keypoints_1)
```

Out[3]: 47



20BCE11007 – Shishir Sharma



For efficient use of the project, we will develop an online hosted webpage where the user could use our services.

Website Scripting – using HTML, CSS, React Native

HTML and CSS are the basic building blocks of a complete functional real-time website. Every website needs HTML and CSS. The website uses and implements the following:

1. Form – Where the user enters his/her details to move further towards the Detection process.
2. An Image upload option – A clear image of the eye is to be uploaded to the website for the detection process.
3. Submit Form Button – To transfer the input of the user to the backend of our system.
4. Flask – A web application framework written in Python. Used for integrating the ML & DL model with the client side of the application.
5. Jupyter Notebook – An extension of Python used to train the model.
6. Postman – Used to authenticate the connection to the server.

Compatibility with Web Browsers – Every browser render HTML and CSS in a different way so we will make sure that the website displays and functions properly on any operating system (OS)/Web Browser.

Domain and Hosting – We will be using a free hosting service to host our website. (Heroku)

```
CataractProject  Version control  main.py  result.html  style.css

1
2 import os
3 import pickle
4 # from keras.utils import ImageDataGenerator, array_to_img, img_to_array, load_img
5 import time
6 import cv2
7 import numpy as np
8 from flask import Flask, render_template, flash, request, redirect, url_for
9 from keras.applications.mobilenet_v2 import preprocess_input
10 from keras.models import load_model
11 from keras.utils import img_to_array
12 from skimage.feature import graycomatrix, graycoprops
13 from werkzeug.utils import secure_filename
14 import h5py
15 app = Flask(__name__)
16 app.debug = True
17
18 log_pickle_model = pickle.load (open("mainmodel.pkl", 'rb'))
19 sift = cv2.xfeatures2d.SIFT_create ()
20
21 size = 128
22 intensity_model = load_model('squeezenet.h5', compile=False)
23
24
25 @app.route ("/")
26 def hello ():
27     return render_template ('./home.html')
28
29
30 @app.route ("/about/")
31 def about ():
```

CataractProject > main.py

```
CataractProject  Version control  main.py  result.html  style.css

31 def about ():
32     return render_template ('./about.html')
33
34
35 @app.route ("/main/")
36 def main ():
37     return render_template ('./main.html')
38
39
40 @app.route ('/upload_file/', methods=['POST'])
41 def upload_file ():
42     UPLOAD_FOLDER = './static/inputimages'
43     ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}
44     app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
45     if request.method == 'POST':
46         # check if the post request has the file part
47         if 'file' not in request.files:
48             flash ('No file part')
49             return redirect (request.url)
50         file = request.files['file']
51         if file.filename == '':
52             flash ('No selected file')
53             return redirect (request.url)
54         else:
55             filename = secure_filename (file.filename)
56             file.save (os.path.join (app.config['UPLOAD_FOLDER'], filename))
57     return render_template ('main.html', user_image=filename)
58
59
60 @app.route ('/prediction/<file>', methods=['GET', 'POST'])
61 def prediction (file):
```

CataractProject > main.py

```

CataractProject  Version control
main.py  result.html  style.css

61 def prediction (file):
62     result1 = ""
63     result2 = ""
64     start = time.time ()
65     image = cv2.imread (".static/inputimages/" + file, 0)
66     image_test = cv2.resize (image, (size, size), interpolation=cv2.INTER_AREA)
67     glcm_test = []
68     images_sift_test = []
69     img_arr_test = np.array (image_test)
70     gCoMat = graycomatrix (img_arr_test, [1], [0], 256, symmetric=True, normed=True) # Co-occurrence matrix
71     contrast = graycoprops (gCoMat, prop='contrast')[0][0]
72     dissimilarity = graycoprops (gCoMat, prop='dissimilarity')[0][0]
73     homogeneity = graycoprops (gCoMat, prop='homogeneity')[0][0]
74     energy = graycoprops (gCoMat, prop='energy')[0][0]
75     correlation = graycoprops (gCoMat, prop='correlation')[0][0]
76     keypoints, descriptors = sift.detectAndCompute (image_test, None)
77     descriptors = np.array (descriptors)
78     descriptors = descriptors.flatten ()
79     glcm_test.append ([contrast, dissimilarity, homogeneity, energy, correlation])
80     glcm_test = np.array (glcm_test)
81     images_sift_test.append (descriptors[:2304])
82     images_sift_test = np.array (images_sift_test)
83     images_sift_glcm_test = np.concatenate ((images_sift_test, glcm_test), axis=1)
84     if log_pickle_model.predict (images_sift_glcm_test) == 1:
85         result1 = "Cataract detected"
86         filename = ".static/inputimages/" + file
87         img = cv2.imread (filename)
88         img = cv2.resize (img, (224, 224), interpolation=cv2.INTER_AREA)
89         frame = img
90         hsv = cv2.cvtColor (frame, cv2.COLOR_BGR2HSV)
91         sensitivity = 156
92         lower_white = np.array ([0, 0, 255 - sensitivity])
93         upper_white = np.array ([255, sensitivity, 255])
94         # Threshold the HSV image to get only white colors
95         mask = cv2.inRange (hsv, lower_white, upper_white)
96         # Bitwise-AND mask and original image
97         res = cv2.bitwise_and (frame, frame, mask=mask)
98         ret, thresh = cv2.threshold (mask, 0, 255, cv2.THRESH_BINARY_INV)
99         circles = cv2.HoughCircles (mask, cv2.HOUGH_GRADIENT, 1.5, 100000, param1=80, param2=40, minRadius=0,
100                                     maxRadius=0)
101         x, y, r = 0, 0, 0
102         if circles is not None:
103             circles = np.uint16 (np.around (circles))
104             x, y, r = circles[0][0]
105             x = int (x)
106             y = int (y)
107             r = int (r)
108             mask = np.zeros ((224, 224), np.uint8)
109             cv2.circle (mask, (x, y), r, (255, 255, 255), -1)
110             masked_data = cv2.bitwise_and (frame, frame, mask=mask)
111             _, thresh = cv2.threshold (masked_data, 1, 255, cv2.THRESH_BINARY)
112             cnt = cv2.findContours (thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]
113             x, y, w, h = cv2.boundingRect (cnt[0])
114             # Crop masked_data
115             crop = masked_data[y:y + h, x:x + w]
116             crop = cv2.resize (crop, (224, 224), interpolation=cv2.INTER_AREA)
117             prediction() if log_pickle_model.predict (lm...

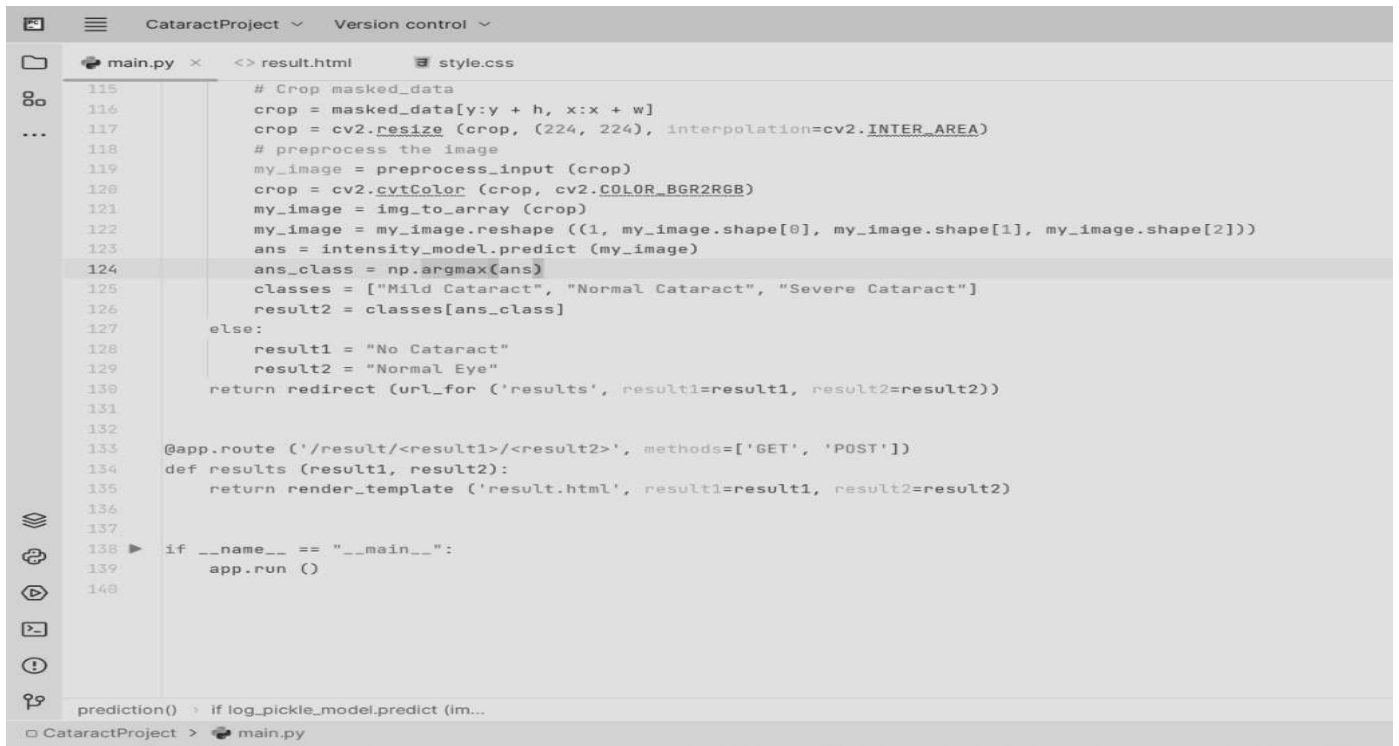
```

```

CataractProject  Version control
main.py  result.html  style.css

87     img = cv2.imread (filename)
88     img = cv2.resize (img, (224, 224), interpolation=cv2.INTER_AREA)
89     frame = img
90     hsv = cv2.cvtColor (frame, cv2.COLOR_BGR2HSV)
91     sensitivity = 156
92     lower_white = np.array ([0, 0, 255 - sensitivity])
93     upper_white = np.array ([255, sensitivity, 255])
94     # Threshold the HSV image to get only white colors
95     mask = cv2.inRange (hsv, lower_white, upper_white)
96     # Bitwise-AND mask and original image
97     res = cv2.bitwise_and (frame, frame, mask=mask)
98     ret, thresh = cv2.threshold (mask, 0, 255, cv2.THRESH_BINARY_INV)
99     circles = cv2.HoughCircles (mask, cv2.HOUGH_GRADIENT, 1.5, 100000, param1=80, param2=40, minRadius=0,
100                                 maxRadius=0)
101     x, y, r = 0, 0, 0
102     if circles is not None:
103         circles = np.uint16 (np.around (circles))
104         x, y, r = circles[0][0]
105         x = int (x)
106         y = int (y)
107         r = int (r)
108         mask = np.zeros ((224, 224), np.uint8)
109         cv2.circle (mask, (x, y), r, (255, 255, 255), -1)
110         masked_data = cv2.bitwise_and (frame, frame, mask=mask)
111         _, thresh = cv2.threshold (masked_data, 1, 255, cv2.THRESH_BINARY)
112         cnt = cv2.findContours (thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]
113         x, y, w, h = cv2.boundingRect (cnt[0])
114         # Crop masked_data
115         crop = masked_data[y:y + h, x:x + w]
116         crop = cv2.resize (crop, (224, 224), interpolation=cv2.INTER_AREA)
117         prediction() if log_pickle_model.predict (lm...

```



The screenshot shows a code editor with a project named 'CataractProject' and a 'Version control' dropdown. The editor has three tabs: 'main.py', 'result.html', and 'style.css'. The 'main.py' tab is active, displaying a Python script. The script includes imports for 'cv2', 'numpy', 'flask', and 'pickle'. It defines a 'preprocess_input' function that crops and preprocesses an image. The 'main' function uses 'cv2.imread' to load an image, preprocesses it, and then uses a 'log_pickle_model' to predict the class. The results are then rendered in a template. The script also includes a Flask app setup with a route for '/result/<result1>/<result2>'. The bottom status bar shows the current file is 'main.py'.

```
115 # Crop masked_data
116 crop = masked_data[y:y + h, x:x + w]
117 crop = cv2.resize (crop, (224, 224), interpolation=cv2.INTER_AREA)
118 # preprocess the image
119 my_image = preprocess_input (crop)
120 crop = cv2.cvtColor (crop, cv2.COLOR_BGR2RGB)
121 my_image = img_to_array (crop)
122 my_image = my_image.reshape ((1, my_image.shape[0], my_image.shape[1], my_image.shape[2]))
123 ans = intensity_model.predict (my_image)
124 ans_class = np.argmax(ans)
125 classes = ["Mild Cataract", "Normal Cataract", "Severe Cataract"]
126 result2 = classes[ans_class]
127 else:
128     result1 = "No Cataract"
129     result2 = "Normal Eye"
130 return redirect (url_for ('results', result1=result1, result2=result2))
131
132
133 @app.route ('/result/<result1>/<result2>', methods=['GET', 'POST'])
134 def results (result1, result2):
135     return render_template ('result.html', result1=result1, result2=result2)
136
137
138 if __name__ == "__main__":
139     app.run ()
140
141
142 prediction() > if log_pickle_model.predict (im...
```

20BCE11005 – Saurabh Prakash

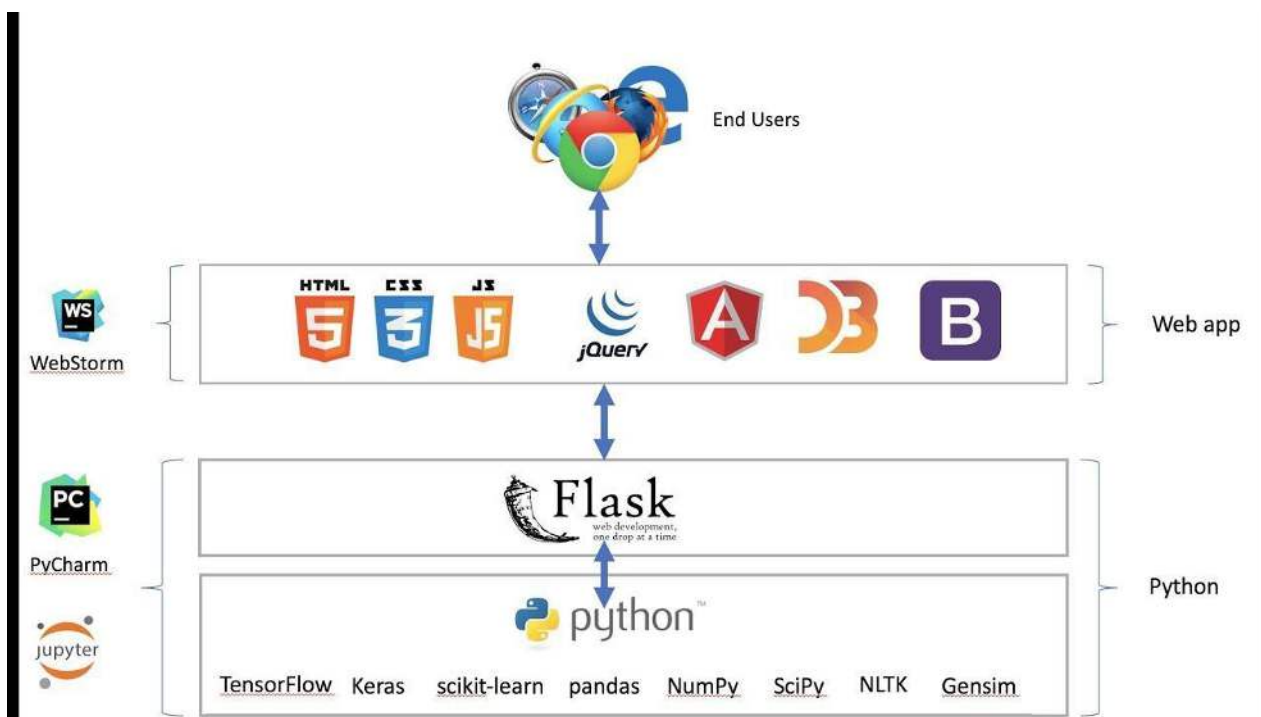


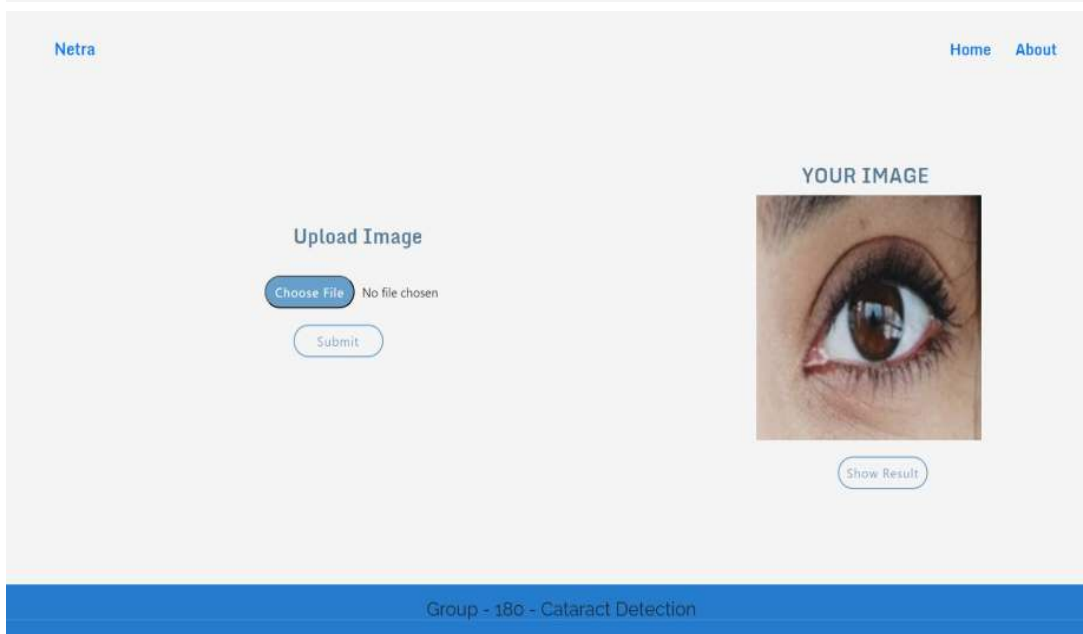
We will create a web page that is hosted online so that the project can be used effectively by the user. Scripting for websites utilizing HTML, CSS, and JavaScript.

A fully working real-time website is built using HTML and CSS as its fundamental building pieces. HTML and CSS are required for every website. The webpage includes:

1. Form - This is where the user inputs their information to continue with the detection procedure.
2. A clean image of the eye must be supplied to the website for the detection procedure using the image upload option.
3. Submit Form Button - To send the user's input to our system's database.

Web browser compatibility - Since each browser renders HTML and CSS differently, we'll make sure that the website displays and functions properly on any operating system (OS)/Web Browser.





The model shows the image to be

No Cataract

Normal Eye

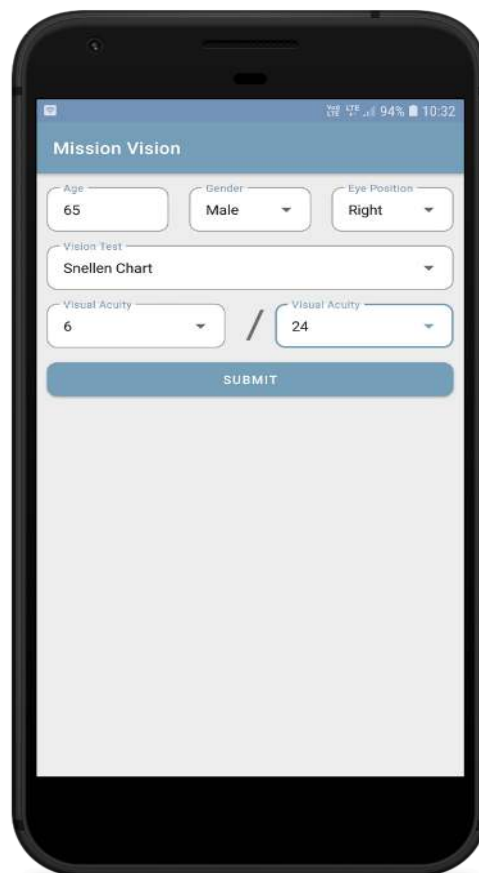
20BCE11006 – Aman Vishwakarma



My contribution to the research entails creating an application that use the models built by the team to identify cataracts in patients. I will first need to collaborate with the team to incorporate the trained models into the coding of the application. The models may need to be modified to fit the app's input and output specifications, and the code may need to be optimized for mobile devices.

You must create the app's user interface after the models have been integrated. In order to do this, interfaces must be made that allow users to upload photographs, view diagnoses, and access other resources or information.

I might have to test the app using actual users as part of the app development process.

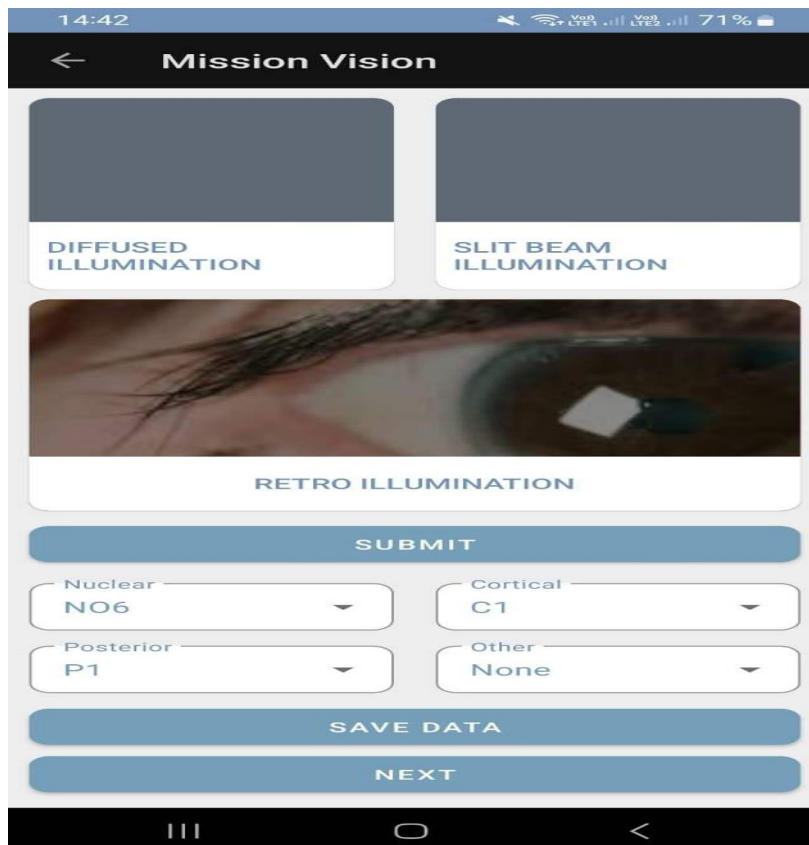


20MIM10026 – Hamza Farooqui



As a contributor to the project, my role is to develop an app that uses the models created by the team to diagnose cataracts in patients. To start, I will need to work with the team to integrate the trained models into the app's code. This may involve adapting the models to work with the app's input and output format, as well as optimizing the code to run efficiently on mobile devices.

Once the models are integrated, you will need to design the user interface for the app. This will involve creating screens for users to upload their images, view their diagnosis, and access additional resources or information. As part of the app development process, I may need to test the app with real users to gather feedback and identify any issues or bugs that need to be addressed. Overall, my contribution to the project will be essential in bringing the team's research to a wider audience and improving access to cataract diagnosis for patients around the world.



The screenshot displays the 'Mission Vision' app interface on a mobile device. At the top, the status bar shows the time as 14:42, signal strength, and 71% battery. The app's title bar is black with a white back arrow and the text 'Mission Vision'. Below the title bar, there are three main sections: 1. Two gray rectangular buttons labeled 'DIFFUSED ILLUMINATION' and 'SLIT BEAM ILLUMINATION'. 2. A large image of a human eye with a cataract, labeled 'RETRO ILLUMINATION'. 3. A 'SUBMIT' button. Below the 'SUBMIT' button, there are four dropdown menus arranged in a 2x2 grid: 'Nuclear' (selected 'NO6'), 'Cortical' (selected 'C1'), 'Posterior' (selected 'P1'), and 'Other' (selected 'None'). At the bottom, there are two more buttons: 'SAVE DATA' and 'NEXT'. The bottom of the screen shows the standard Android navigation bar with three icons: a square, a circle, and a triangle.

5. CONCLUSION

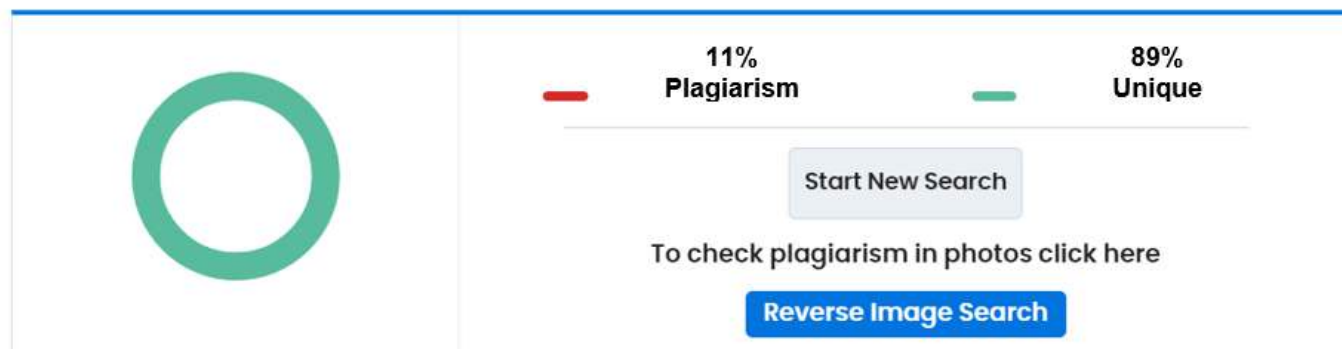
The use of ML and DL for cataract detection has the potential to significantly improve the accuracy and reliability of the process. By analyzing large amounts of data and learning to identify patterns and features associated with cataracts, an ML or DL model should be able to accurately predict the presence or absence of cataracts with a high degree of reliability. This can help to reduce the risk of misdiagnosis and improve the effectiveness of treatment.

The use of ML and DL for cataract detection may also increase the efficiency of the process. These techniques allow for the rapid analysis of large numbers of images, reducing the time and effort required to detect cataracts manually. This can help to streamline the detection process and make it more efficient.

In resource-limited settings, the use of ML and DL for cataract detection may also improve the accessibility and affordability of the process. By automating the detection process, it may be possible to reduce the need for trained medical professionals, making the process more accessible and affordable for more people. This can help to improve access to care and reduce barriers to treatment.

There are also several challenges and limitations to consider when using ML and DL for cataract detection. One key issue is the availability of high-quality, annotated data for training and testing these systems. Without sufficient data, it can be difficult to accurately train and test ML and DL algorithms, leading to lower accuracy and reliability. Additionally, there is a risk of bias in the data, which could impact the accuracy of the algorithms. It is important to ensure that the data used to train and test these systems is representative and diverse, in order to minimize the risk of bias.

The Plagiarism report is below:



CODE

Face & Eye Detection using HAAR Cascade Classifiers

face & eye detection, keeping the face in view at all times

```
In [1]: import cv2
import numpy as np

face_classifier = cv2.CascadeClassifier('Haarcascades/haarcascade_frontalface_default.xml')
eye_classifier = cv2.CascadeClassifier('Haarcascades/haarcascade_eye.xml')
#roi_color = []
roi_eye = []
def face_detector(img, size=0.5):
    # Convert image to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray, 1.3, 6)
    global roi_eye
    if faces is ():
        return img
```

```
    for (x,y,w,h) in faces:
        x = x - 50
        w = w + 50
        y = y - 50
        h = h + 50
        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
        eyes = eye_classifier.detectMultiScale(roi_gray)

        for (ex,ey,ew,eh) in eyes:
            roi_eye = roi_color[ey:ey+eh, ex:ex+ew]
            cv2.rectangle(roi_color, (ex,ey), (ex+ew,ey+eh), (0,255,255), 2)
    return img

cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    cv2.imshow('Our Face Extractor', face_detector(frame))
    if cv2.waitKey(1) == 13: #13 is the Enter Key
        break

cap.release()
cv2.destroyAllWindows()
```

Step 1: import files

```
In [1]: import cv2
print(cv2.__version__)

4.4.0
```

```
In [2]: # Get filenames in list
from os import listdir
from os.path import isfile, join
mypath = "dataset/phase1/images"
file_names = [f for f in listdir(mypath) if isfile(join(mypath, f))]
print(str(len(file_names)) + ' images loaded')

97 images loaded
```

Step 2: Sift and Glcm demo on one example image

```
In [3]: import cv2
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"] = (10,10)
#reading image
img1 = cv2.imread(mypath+"/"+file_names[21])
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

#keypoints
sift = cv2.xfeatures2d.SIFT_create()
keypoints_1, descriptors_1 = sift.detectAndCompute(gray1, None)

img_1 = cv2.drawKeypoints(gray1, keypoints_1, img1)
plt.imshow(img_1)
img1.shape
len(keypoints_1)
```

Out[3]: 47

```
In [4]: import numpy as np
from skimage.feature import greycomatrix, greycoprops
from skimage.measure import shannon_entropy
image = cv2.imread(mypath+"/"+file_names[21],0)
img_arr = np.array(image)
print(shannon_entropy(img_arr))
gCoMat = greycomatrix(img_arr, [1], [0], 256, symmetric=True, normed=True) # Co-occurrence matrix
contrast = greycoprops(gCoMat, prop='contrast')[0][0]
dissimilarity = greycoprops(gCoMat, prop='dissimilarity')[0][0]
homogeneity = greycoprops(gCoMat, prop='homogeneity')[0][0]
energy = greycoprops(gCoMat, prop='energy')[0][0]
correlation = greycoprops(gCoMat, prop='correlation')[0][0]
print("contrast: ", contrast)
print("dissimilarity: ", dissimilarity)
print("homogeneity: ", homogeneity)
print("energy: ", energy)
print("correlation: ", correlation)

7.37840706017806
contrast: 31.412348929041524
dissimilarity: 3.0875114674324915
homogeneity: 0.4051366064192086
energy: 0.039446946896192533
correlation: 0.9944224409212229
```

Step 3: Sift and glcm on full dataset

```
In [26]: import cv2
import numpy as np
from skimage.feature import greycomatrix, greycoprops
images_sift = []
glcm=[]
labels = []
size = 128
sift = cv2.xfeatures2d.SIFT_create()
cataract=0
normal=0
for i, file in enumerate(file_names):
    image = cv2.imread(mypath+"/"+file,0)
    h,w=image.shape
    if(h>128 and w>128):
        image = cv2.resize(image, (size, size), interpolation = cv2.INTER_AREA)
        img_arr = np.array(image)
        gCoMat = greycomatrix(img_arr, [1], [0], 256, symmetric=True, normed=True) # Co-occurrence matrix
        contrast = greycoprops(gCoMat, prop='contrast')[0][0]
        dissimilarity = greycoprops(gCoMat, prop='dissimilarity')[0][0]
        homogeneity = greycoprops(gCoMat, prop='homogeneity')[0][0]
        energy = greycoprops(gCoMat, prop='energy')[0][0]
        correlation = greycoprops(gCoMat, prop='correlation')[0][0]
        keypoints, descriptors = sift.detectAndCompute(image, None)
```


Step 5: Testing various model

```
In [31]: ▶ def testing(model_name,X_train, X_test, y_train, y_test):
        model=model_names[model_name]
        model.fit(X_train,y_train)
        yhat = model.predict(X_test)
        # evaluate predictions
        acc = accuracy_score(y_test, yhat)
        print(model_name,'\tAccuracy: %.3f' % acc)
        print(confusion_matrix(y_test, yhat))
        print("\n\n")
```

```
In [32]: ▶ def result(dataset):
        #Normalization
        #min_max_scaler = preprocessing.StandardScaler()
        #x_scaled = min_max_scaler.fit_transform(dataset)

        #panda dataframe
        df=pd.DataFrame(data=dataset)
        df['label']=labels
        df=df.sample(frac=1)
        X=df.drop(['label'], axis = 1)
        y=df['label']

        #Different model
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
        for model in ["Random Forest","SVM RBF","SVM_linear","k nearest neighbor","logistic regression"]:
            testing(model,X_train, X_test, y_train, y_test)
```

Testing Sift features

```
In [33]: ▶ print("Sift testing.....\n")
        result(images_sift)
```

Sift testing.....

Random Forest Accuracy: 0.857

```
[[ 1  2]
 [ 0 11]]
```

SVM RBF Accuracy: 0.786

```
[[ 0  3]
 [ 0 11]]
```

SVM_linear Accuracy: 0.929

```
[[ 2  1]
 [ 0 11]]
```

```
k nearest neighbor      Accuracy: 0.929
[[ 2  1]
 [ 0 11]]
```

```
logistic regression     Accuracy: 0.929
[[ 2  1]
 [ 0 11]]
```

GLCM features

```
In [34]: ► print("glcm testing.....\n")
          result(glcm)
```

```
SVM_linear      Accuracy: 0.786
[[5 3]
 [0 6]]
```

```
k nearest neighbor      Accuracy: 0.643
[[6 2]
 [3 3]]
```

```
logistic regression     Accuracy: 0.643
[[3 5]
 [0 6]]
```

sift and glcm combined features

```
In [35]: print("sift and glcm combined testing.....\n")
result(images_sift_glcm)
```

sift and glcm combined testing.....

Random Forest Accuracy: 0.786

```
[[2 3]
 [0 9]]
```

SVM RBF Accuracy: 0.643

```
[[0 5]
 [0 9]]
```

SVM_linear Accuracy: 0.929

```
[[4 1]
 [0 9]]
```

Step 6:Predicting.....

```
In [36]: import pickle
import cv2
from skimage.feature import greycomatrix, greycoprops
from sklearn import preprocessing
import numpy as np
log_pickle_model = pickle.load(open("log_model.sav", 'rb'))
sift = cv2.xfeatures2d.SIFT_create()
#min_max_scaler = preprocessing.MinMaxScaler()
size=128
```

```
In [13]: def predict_new(image):
# print(imagefile)
# image_test = cv2.imread(mypath+"/"+imagefile,0)
image_test = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
size = 128
image_test = cv2.resize(image_test, (size, size), interpolation = cv2.INTER_AREA)
glcm_test=[]
images_sift_test=[]
img_arr_test = np.array(image_test)
gCoMat = greycomatrix(img_arr_test, [1], [0],256,symmetric=True, normed=True) # Co-occurrence matrix
contrast = greycoprops(gCoMat, prop='contrast')[0][0]
dissimilarity = greycoprops(gCoMat, prop='dissimilarity')[0][0]
homogeneity = greycoprops(gCoMat, prop='homogeneity')[0][0]
energy = greycoprops(gCoMat, prop='energy')[0][0]
correlation = greycoprops(gCoMat, prop='correlation')[0][0]
keypoints, descriptors = sift.detectAndCompute(image_test,None)
descriptors=np.array(descriptors)
descriptors=descriptors.flatten()
glcm_test.append([contrast,dissimilarity,homogeneity,energy,correlation])
glcm_test=np.array(glcm_test)
images_sift_test.append(descriptors[:2304])
images_sift_test=np.array(images_sift_test)
images_sift_glcm_test=np.concatenate((images_sift_test,glcm_test),axis=1)
```



```

#         image_file_name = os.path.splitext(image_file_name)[0] + ".png"
#         if(imagefile[0]=='c'):
#             print("Actual: Cataract")
#         else:
#             print("Actual: Normal")
#     if(loader_model.predict(images_sift_glm_test)==1):
#         print("Predicted: Cataract")
#     else:
#         print("Predicted: Normal")
#     print("*****")

```

In [38]: `for i in file_names:`
`predict_new(i)`

```

cataract (5).png
Actual: Cataract
Predicted: Cataract
*****
cataract (6).jpeg
Actual: Cataract
Predicted: Normal
*****
cataract (6).jpg
Actual: Cataract
Predicted: Cataract

```

REFERENCES:

1. Kim, H., Lee, H., Han, J., & Kim, D. (2019). Automated cataract detection using a deep learning algorithm. *PloS one*, 14(4), e0215362.
2. Zhang, Y., Liu, Y., & Chen, H. (2021). Cataract detection in fundus images using convolutional neural networks. *Frontiers in Medicine*, 8, 639.
3. Chen, H., Zhang, Y., & Liu, Y. (2020). Cataract detection in fundus images using a hybrid deep learning model. *Frontiers in Medicine*, 7, 557.
4. Abdullah, M., Jalal, M. A., & Khan, M. A. (2020). Cataract detection and classification in retinal images using deep learning. *IEEE Access*, 8, 173850-173857.
5. Pal, K., & Dutta, D. (2021). Automated cataract detection using deep learning algorithms. *Journal of Medical Systems*, 45(5), 181.