Rohith Yogi (IMT2016072)
Sumanth (IMT2016081)
Nikhil Sai (IMT2016091)

Visual Recognition

Assignment 3 (REPORT)

Overview

In this report, we will summarize the various steps involved in face and gender recognition. This project is divided into five parts.

PART I: IIITB Face Dataset

PART II: Pre-Processing

PART III: Face Recognition

PART IV: Explore an open source solution and compare

PART V: Gender recognition

PART I consists of finding the dataset useful for performing experiments on Face and Gender recognition. The dataset contains about 43 folders, one for each person and each folder having about 20 images of the same people in different lighting conditions, expression and pose variations.

These variations will help the model, in turn, to be robust and recognize the face and gender in any given circumstances.

PART II consists of preprocessing the images by exploring various functions for manipulating and extracting features from them.

In this part, we loaded the images from the dataset and split them into train and test in the ratio of 4:1. Thereafter these images are used in feature extraction by performing operations on them.

The images were preprocessed and then resized 256x256.

We used two different approaches for face detection. The first approach is using haar cascading. The XML files used for haar cascading were taken from the open source.

https://github.com/opencv/opencv/tree/master/data/haarcascades

haarcascade_frontalface_default.xml

Haar cascade helps in detecting the faces. It detects the faces with almost 90% accuracy. Few images were needed to be discarded as the pose is too much away from frontal.

Haarcascade_eye_tree_eyeglasses.xml

Similarly, the eyes can be detected using this. There are several other files that can be used in detecting eyes. The file used by us detects both eyes and can also be used if the person in the image is wearing spectacles. So we preferred this over the other haar cascade files.

Haarcascade_mcs_nose.xml

This XML file, in particular, did not provide us with good results as the nose was detected in few images and was not in other images so we had to change this file constantly to get better and better results.

After the eyes were detected they were separated out to find the center for each eye. To do this we converted the grayscale image to binary image to calculate moments of the binary image. After which we calculated x and y coordinates of the center.

This was the process followed to detect the center of the eyes.

Then the next task was to Rotate and Scale the image so that the distance between the two eyes is 128 pixels. Most of the images were perfectly aligned so we focused more on the scaling of the images so that the distance between the two eyes is 128 pixels. To do this we calculated the distance between the eyes and then resized the image to that margin.

cv2.resize(image, (0,0), fx=128/distance, fy=128/distance)

So now the distance between the two eyes is 128 pixels.

The other significant feature we found was the tip of the nose. The haar cascade was helpful in detecting nose for most of the images and after which the center of the nose was simple enough to find as the same method that we used for detecting centroid for the eyes was used here.

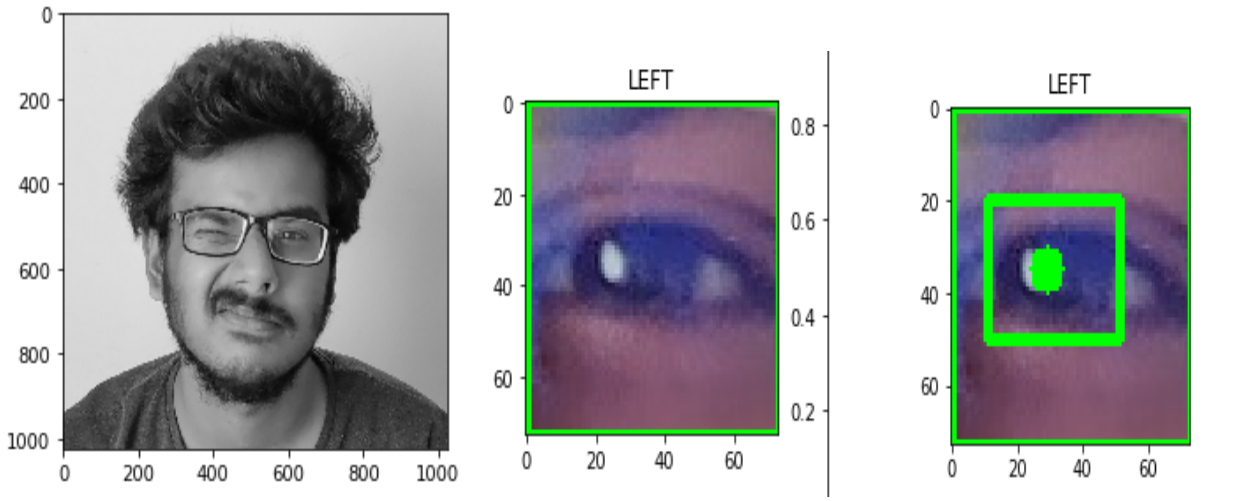
The final task in the preprocessing stage was to find the oval region corresponding to the face and reset all pixels outside the oval region to a constant value. This task was accomplished using the OpenCV function ellipse.

The process was first to create a mask image of the same shape as an input image, filled with 0s (black color) and create a white filled

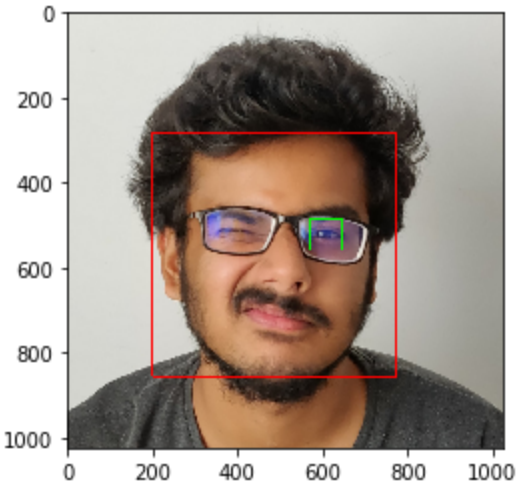
ellipse. Simply using the Bitwise AND operation to blackout regions outside the mask to a constant value. The plots can be seen below

Preprocessed Images:

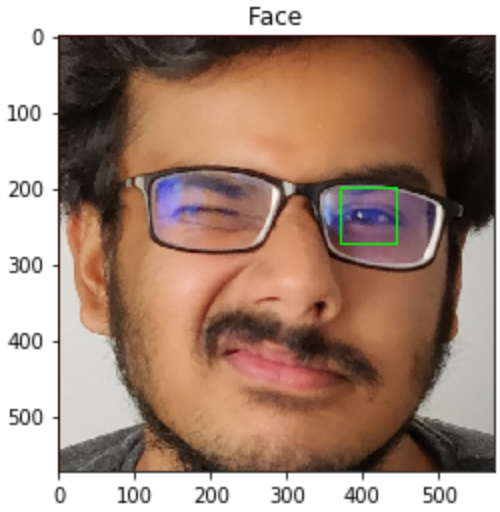
INPUT IMAGE **EYE Detection** **EYE Centroid Detection**



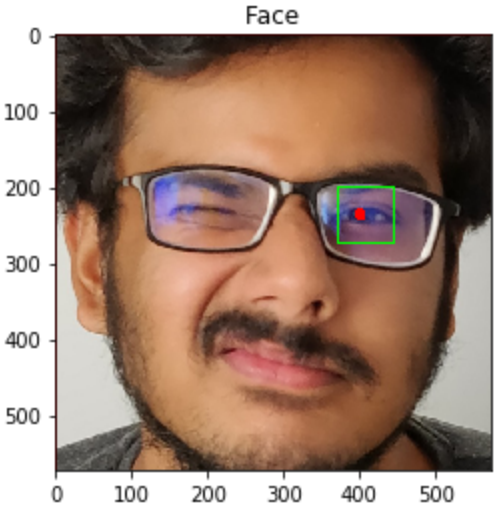
Boundary around detected face

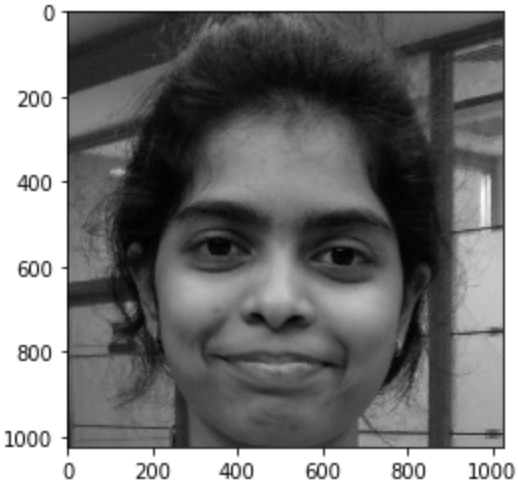


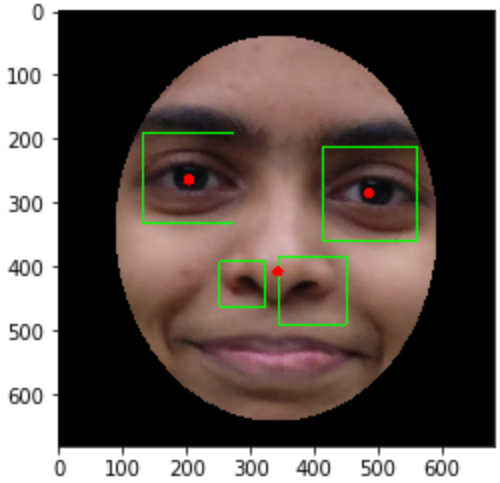
Boundary around detected eye

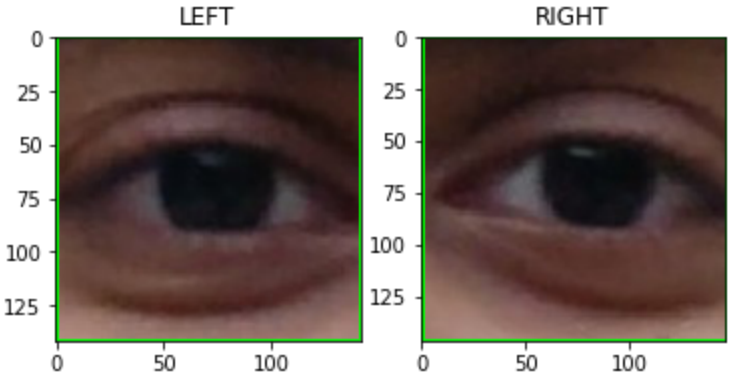


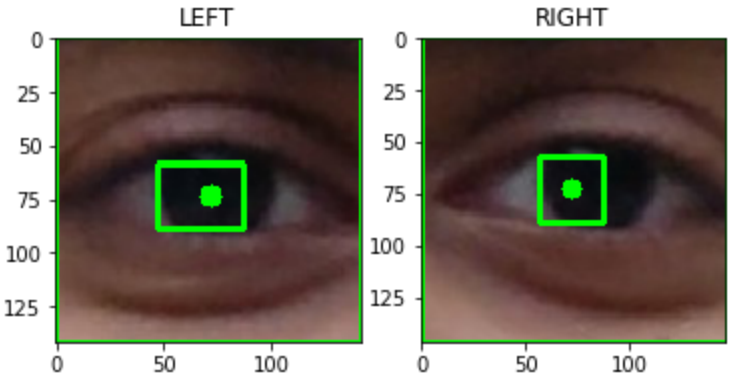
Plotting the centroid of eye with the boundary

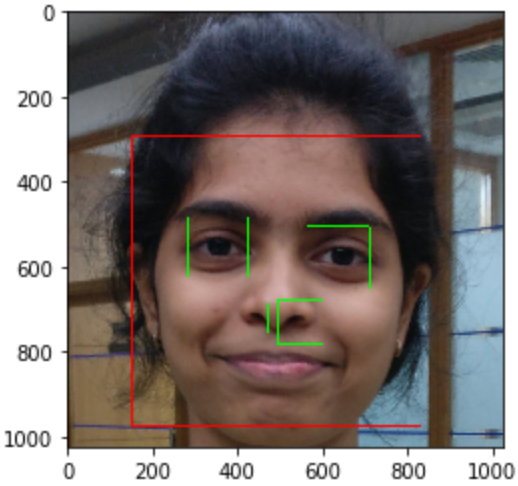


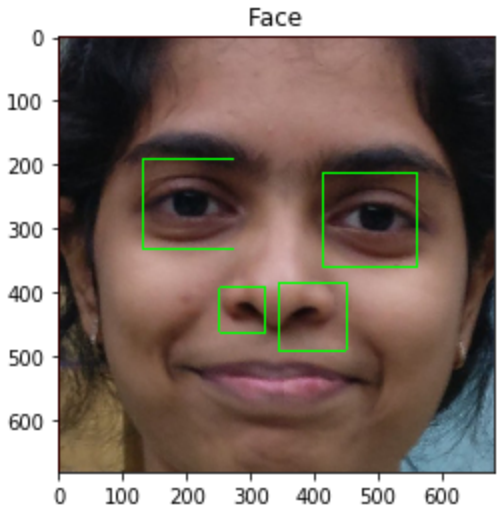

0
200
400
600
800
1000
0
200
400
600
800
1000

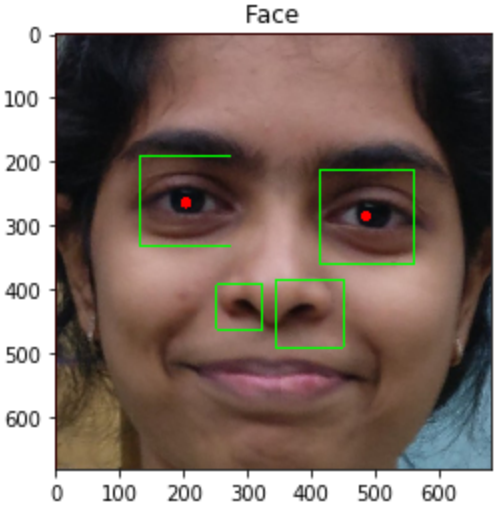

0
100
200
300
400
500
600
0
100
200
300
400
500
600

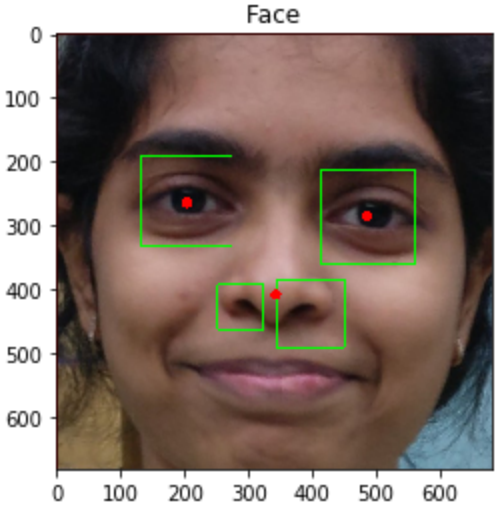

LEFT
RIGHT
0
25
50
75
100
125
0
50
100

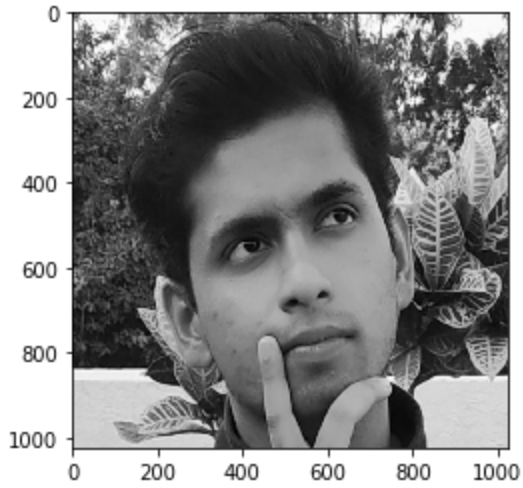

LEFT
RIGHT
0
25
50
75
100
125
0
50
100

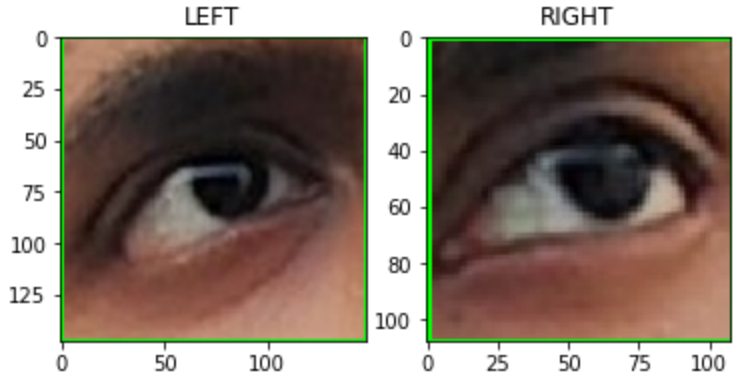

0
200
400
600
800
1000
0
200
400
600
800
1000

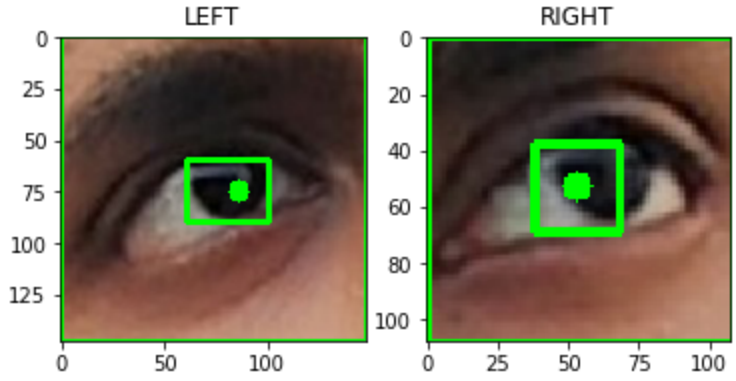

Face
0
100
200
300
400
500
600
0
100
200
300
400
500
600

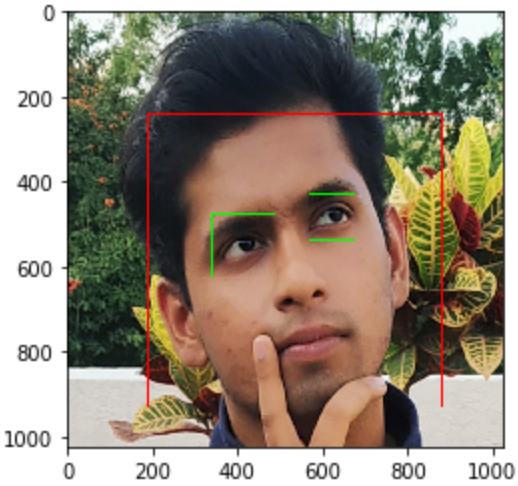

Face
0
100
200
300
400
500
600
0
100
200
300
400
500
600

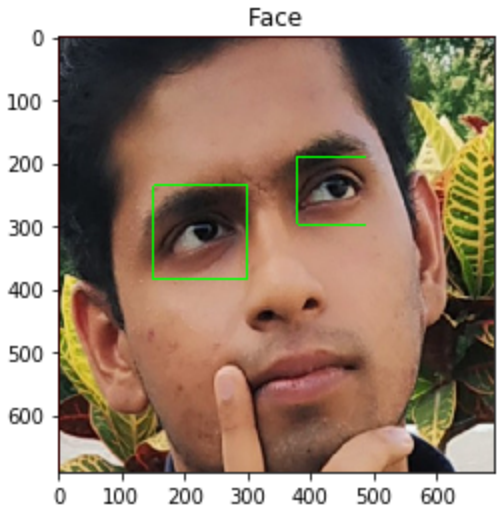

Face
0
100
200
300
400
500
600
0
100
200
300
400
500
600

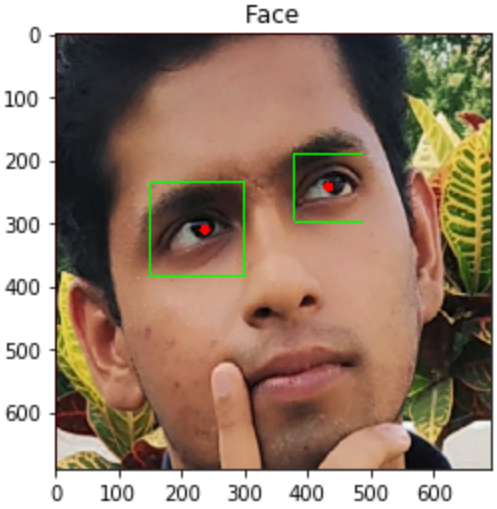

0
200
400
600
800
1000
0
200
400
600
800
1000

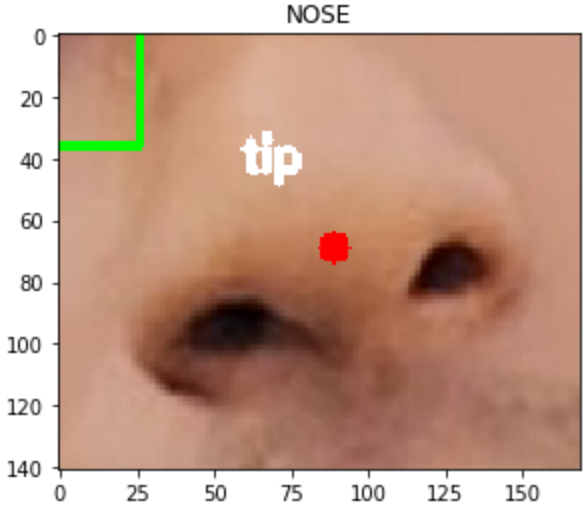

LEFT
RIGHT
0
25
50
75
100
125
0
50
100
0
20
40
60
80
100
0
25
50
75
100

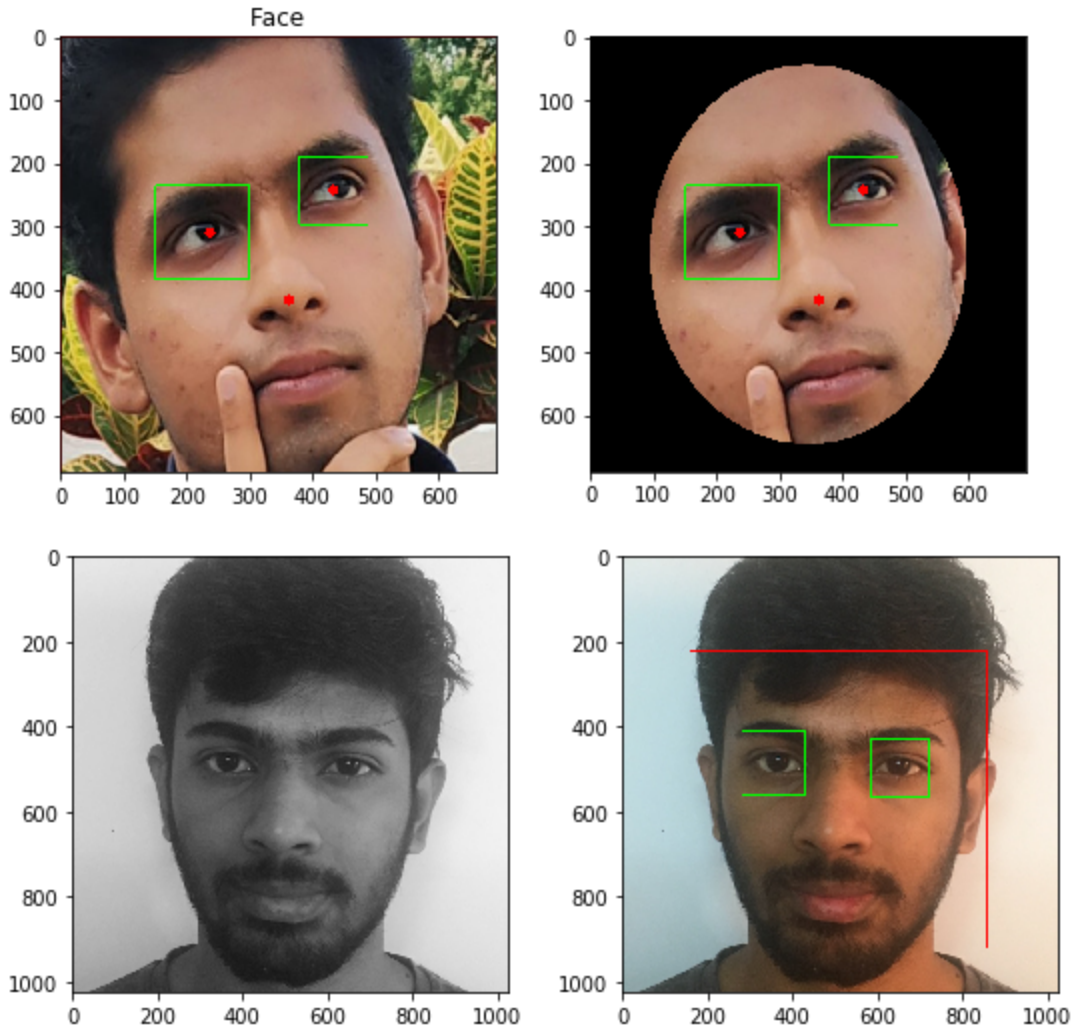

LEFT
RIGHT
0
25
50
75
100
125
0
50
100
0
20
40
60
80
100
0
25
50
75
100

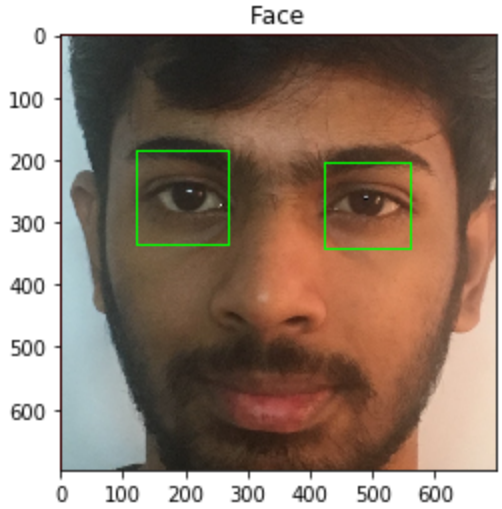


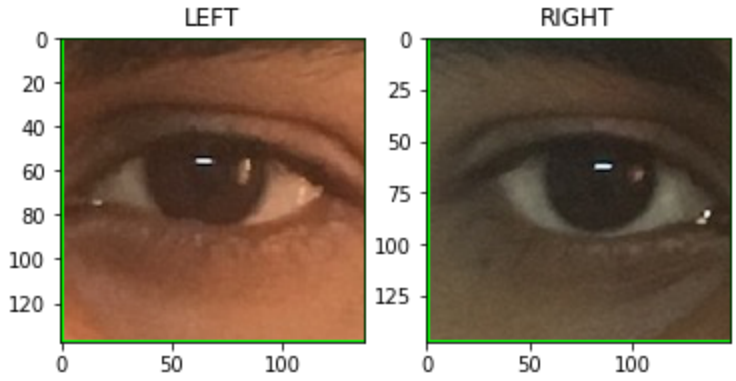

Face

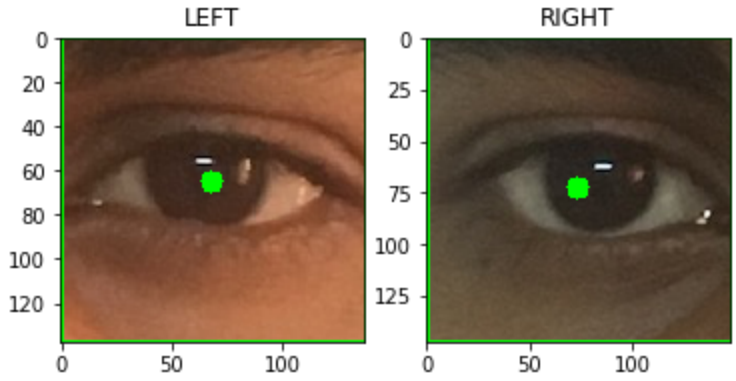

Face

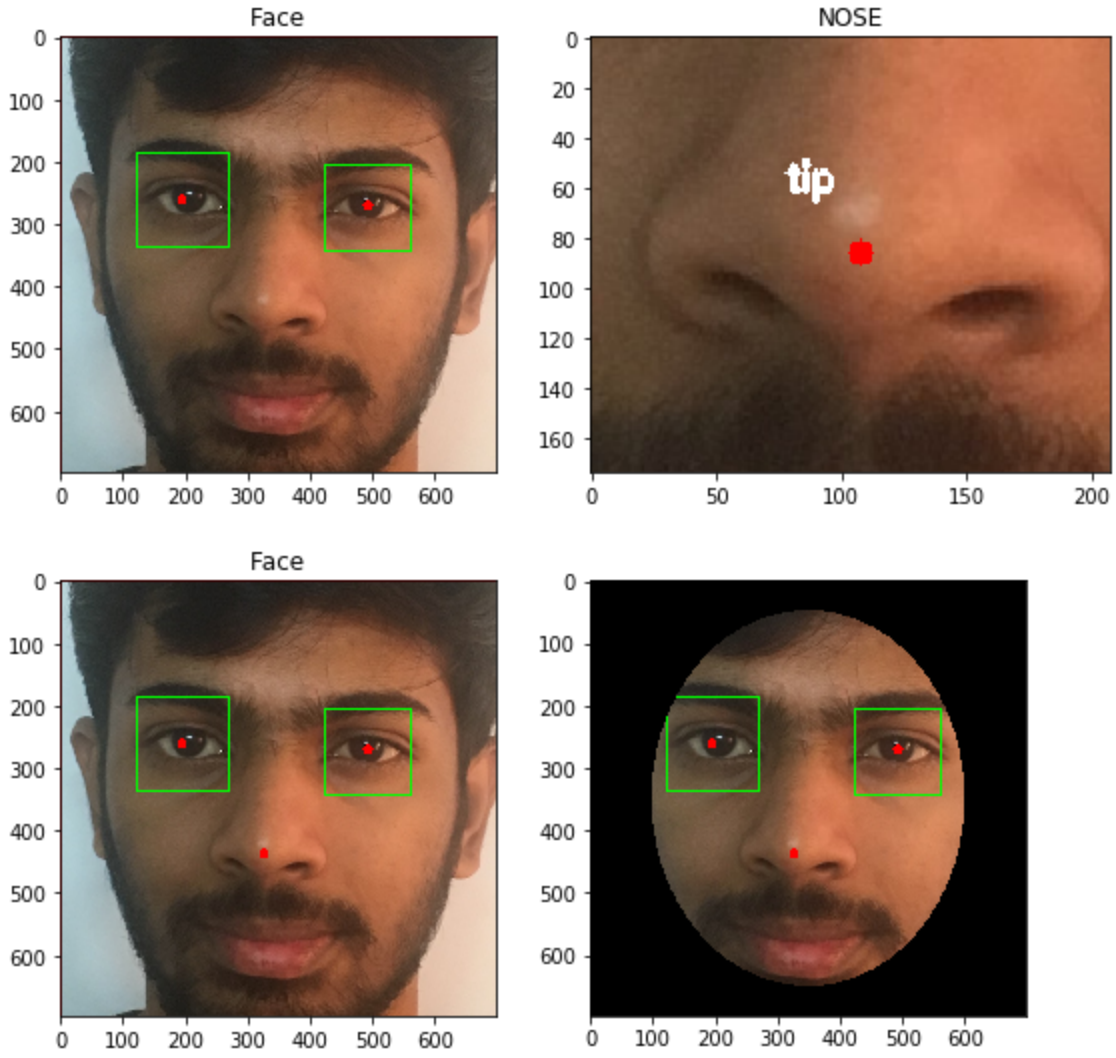

NOSE
tip


Face
0
100
200
300
400
500
600
0
100
200
300
400
500
600
0
100
200
300
400
500
600
0
100
200
300
400
500
600
0
200
400
600
800
1000
0
200
400
600
800
1000
0
200
400
600
800
1000
0
200
400
600
800
1000


Face
0
100
200
300
400
500
600
0 100 200 300 400 500 600


LEFT
RIGHT
0
20
40
60
80
100
120
0 50 100
0
25
50
75
100
125
0 50 100


LEFT
RIGHT
0
20
40
60
80
100
120
0 50 100
0
25
50
75
100
125
0 50 100



PART III

Face Recognition is done on IIITB Face Dataset (Processed and also on unprocessed data) using PCA algorithm.

- The data is split into Traning and Testing in the ratio of 80 to 20. Splitting of data is done to each students folder.

- Then all the training images are stacked in the form of a matrix where each column represents a linearized image.
- Then mean shifted images are calculated (A)
- Eigen vectors of covariance Matrix (AA^T) need to be calculated but due to memory constrain eigen vectors A^TA are calculated using inbuild numpy function.
- We get eigen vectors of AA^T by matrix multiplication of A^T with the eigen vectors these eigen vectors are called as EigenFaces.
- Then we transpose the training and testing images into eigen space.
- Then the label of the testings image is calculated using Euclidean distance to the training images.
- The label of The training image which has the least distance to the test image is assigned as the label to the test image.
- We have Experimented with different no of eigenvectors and the results and observations are presented.

Results and Observations:

Results of PCA on unprocessed data :

No of Eigen Faces = 685

Top1 accuracy = 65.5 %

Top 10% accuray =1.5 %

Top third accuracy = 2.931

No of Eigen Faces = 500

Top1 accuracy = 64.36%

Top 10% accuray =1.505%

Top third accuracy = 2.931%

No of Eigen Faces = 300

Top1 accuracy = 63.21%

Top 10% accuray =1.517%

Top third accuracy = 2.95%

No of Eigen Faces = 60

Top1 accuracy = 64.36%

Top 10% accuray =1.425%

Top third accuracy = 2.844%

Results of PCA on processed data :

No of Eigen Faces = 685

Top1 accuracy = 43.67%

Top 10% accuracy =1.05%

Top third accuracy = 1.90%

No of Eigen Faces = 500

Top1 accuracy = 43.67%

Top 10% accuray =1.057%

Top third accuracy = 1.89%

No of Eigen Faces = 100

Top1 accuracy = 41.95%

Top 10% accuray = 1.045%

Top third accuracy = 1.90%

No of Eigen Faces = 15

Top1 accuracy = 44.8%

Top 10% accuray = 1.086%

Top third accuracy = 2.103%

What I observed was that the accuracy reaches a threshold value and saturate from there on it doesn't matter how many eigenfaces you use,the important features are contained only in some basic limited no of eigenfaces.

Another thing is that for processing i.e detecting images, we tried different methods such as using the haar cascade, dlib library. The results were not great. So the accuracy of Face Recognition has decreased a lot.

Face recognition on IIITB data set using LDA and LBP.

- **Linear discriminant analysis**(LDA) is a classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule.
- The model fits a Gaussian density to each class, assuming that all classes share the same covariance matrix.
- In this Assignment, we used two types of input data to the LDA and LBP.
    - Raw input images without any pre-processing are vectorized and given as input.
    - The resultant images after pre-processing were sent as input to LDA.
- Used **sklearn.discriminant_analysis.LinearDiscriminantAnalysis** to apply LDA to the image data
- Fit_transform is used to Fit data, then transform it.
- Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.
- One of the important parameter in LDA is the number of components for the dimensionality reduction.
- This transformed X is used in Logistic regression to get the model.

- **Local Binary Patterns**(LBP) is one of the methods to extract features from the images.
- It labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number.
- Used open-cv functions to apply LBP on the images.

Results for LDA and LBP:

- **LDA without preprocessed data**
  - Number of components = 100
  - Logistic Regression accuracy = 0.69540022988505747
- **LDA with preprocessing**
  - Number of components = 100
  - Logistic regression accuracy = 0.4770
- **LBP without preprocessing**
  - Acccuracy = 0.68965
- **LBP with preprocessing**
  - Accuracy = 0.58620

PART IV

The open source solution of the face recognition problem is known as open face. OpenFace is a Python and Torch implementation of face recognition with deep neural networks.

The code is available on GitHub at cmusatyalab/openface.

The code shows the instructions on how to install and run the open face module. And thereafter we installed the dlib library in python. The dlib

library contains an implementation of the face_recognition module which in turn is used on the dataset given to us.

After the test and train images are generated by splitting the dataset into 4:1 ratio the labels of those images were also generated.

The face_recognition module contains face_locations and face_encodings which were used on the train and test images. First, the train images were converted into RGB images and then these images were passed into face_locations which detects the faces and the face_encodings detect the encodings. These are used as the features in the train images and a then similar thing is done on the test images and all the features there extracted are compared using compare_face operation in the face detection module. All those images that are matched are separated and the accuracy score is calculated. These are the results obtained:

We got an accuracy of 55.747%. when compared to PCA in part 3 It is very less.

Part V

Below is the basic idea/architecture for gender recognition using for IIITB dataset

1. Collect all the images and resize them into standard size format.
2. Use algorithms like CLACHE for HistogramEquilizationto balance the light Intensity of the input images.
3. Use HAAR Classifiers to detect the faces from GRAY scale input image which returns the region of interest of face.
4. Histogram Oriented Gradients helps us to find the features of the image which mainly focus on the shape and texture of extracted faces.This is used for gender detection.

5. Use SVM to train on the above features to classify the images according to gender.

Resources:

https://github.com/opencv/opencv/tree/master/data/haarcascades

https://www.learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/

http://stackoverflow.com/

https://docs.opencv.org/