Visual Recognition

Assignment 3 - Mini Project

Pavan Bharadiya

MT2018023

M.Tech CSE - 1st Year

Vasu Bansal

MT2018130

M.Tech CSE - 1st Year

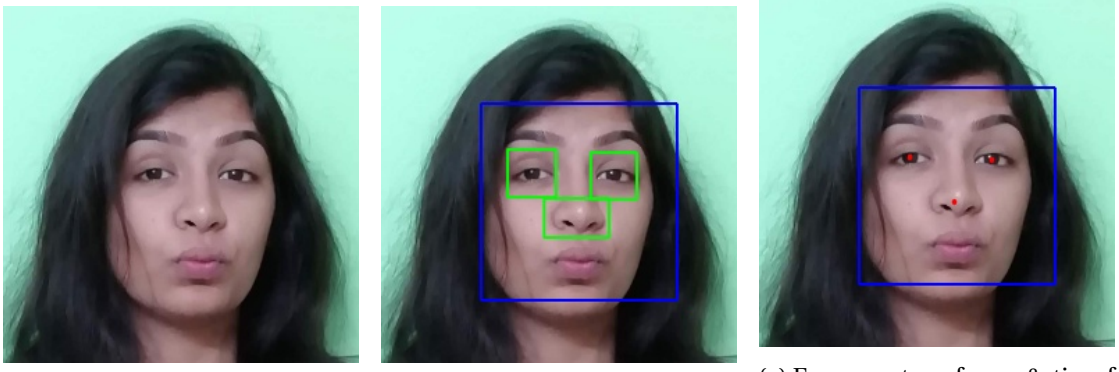
March 29, 2019

1 QUESTION 1

Pre-Processing

Explore functions for manipulating images and extracting features.

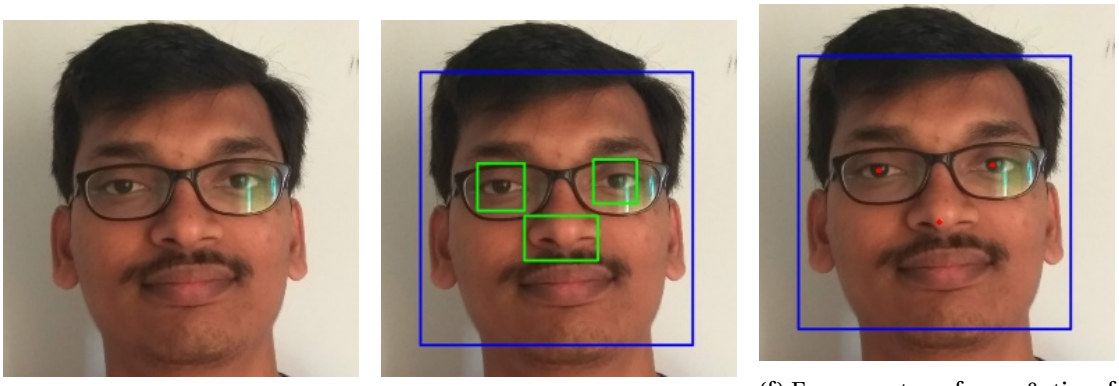
1. **Haar feature based classifier for face, eyes and nose detection**
   a) Load the image and convert it to grayscale.
   b) In this method we are using Haar feature based cascade classifier for face, eyes and nose detection.
   c) We have downloaded the .xml files for face, eyes, and nose using which we are loading a classifier as follows:
   face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
   d) Using the cascade classifier we are detecting faces from input image. The detected faces are returned as a list of rectangles which is represented by four points x,y,w,h.
   e) x,y are the coordinates of upper left corner of face,w is width of rectangle in the face, and h is height of rectangle in the face.
   f) Using x,y,w,h we have drawn a rectangle around which contains the face.
   g) For eyes detection only face area is considered as eyes canâĂŹt be outside face.
   h) Using same steps as above we have drawn a rectangle around eyes.
   i) Also to detect center of each eye, we used same coordinate and drawn circle at (x+w//2,y+h//2).
   j) In similar way we have detected nose and tip of nose.
   k) Then we have Resized and saved output images in 256 X 256 pixels format.



(a) Original

(b) Face, eyes, & nose detection

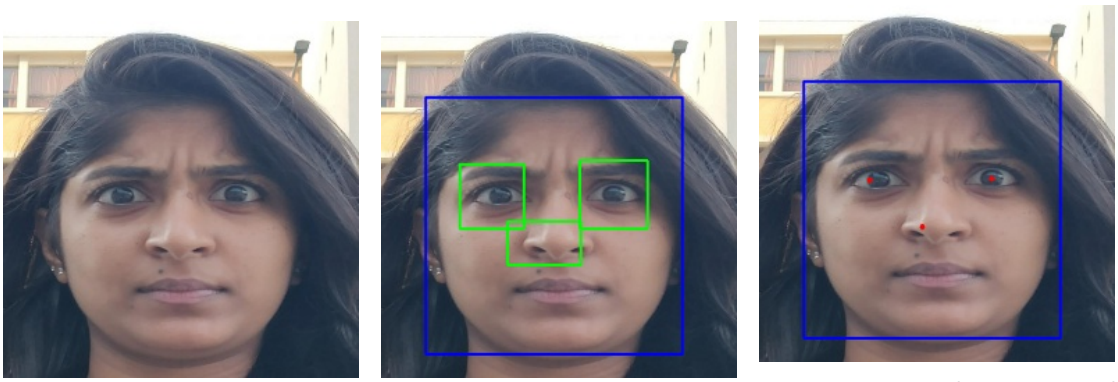
(c) Face, center of eye, & tip of nose detection



(d) Original

(e) Face, eyes, & nose detection

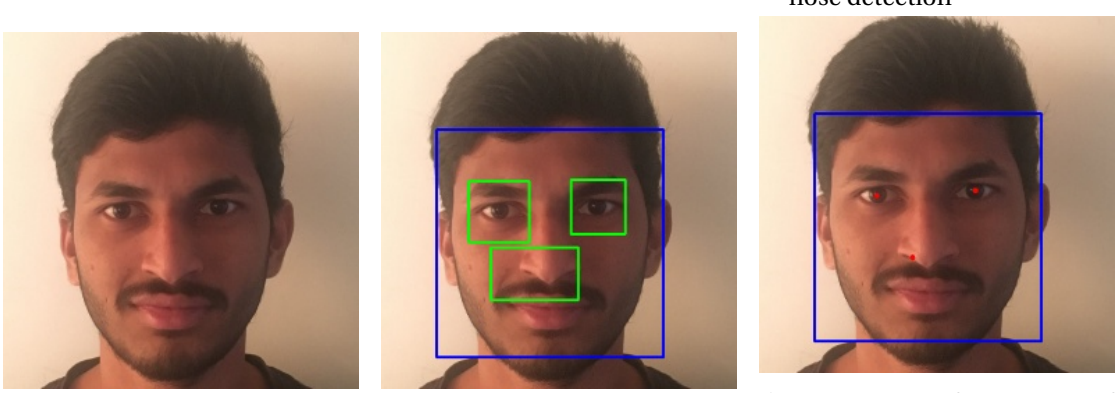
(f) Face, center of eye, & tip of nose detection



(g) Original

(h) Face, eyes, & nose detection

(i) Face, center of eye, & tip of nose detection



(j) Original

(k) Face, eyes, & nose detection

(l) Face, center of eye, & tip of nose detection

Figure 1.1: Face and facial features detection using Haar Cascade

2. **Facial landmarks and eyes alignment with dlib, OpenCV**

   a) The facial landmark detector implemented inside dlib produces 68 (x, y)-coordinates that map to specific facial structures.

   b) These 68 point mappings were obtained by training a shape predictor on the labeled iBUG 300-W dataset.

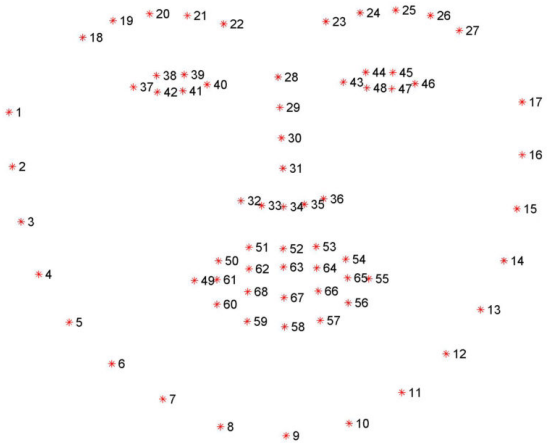


Figure 1.2: 68 facial landmark coordinates

   c) Examining the image, we can see that facial regions can be accessed via simple Python indexing:

      i. The mouth can be accessed through points [48, 68].

      ii. The right eyebrow through points [17, 22].

      iii. The left eyebrow through points [22, 27].

      iv. The right eye using [36, 42].

      v. The left eye with [42, 48].

      vi. The nose using [27, 35].

      vii. And the jaw via [0, 17].

   d) Initializes dlibâĂŹs pre-trained face detector based on a modification to the standard Histogram of Oriented Gradients + Linear SVM method for object detection.

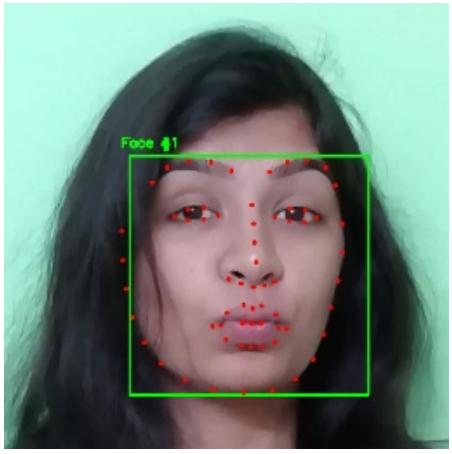
   e) Load input image from disk via OpenCV, then pre-processes the image by resizing to have a width of 500 pixels and convert it to grayscale.

   f) Then loads the facial landmark predictor.

   g) Draw the bounding box surrounding the detected face.

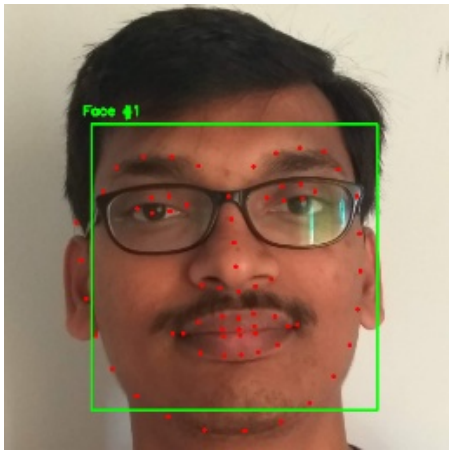
   h) Loop over the detected facial landmarks and draw each of them individually.

(a) Original



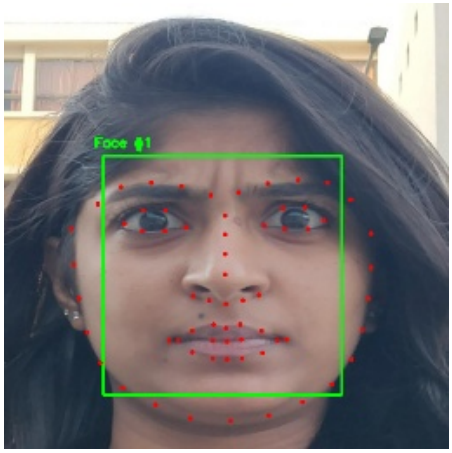
(b) Facial feature detection

(c) Original



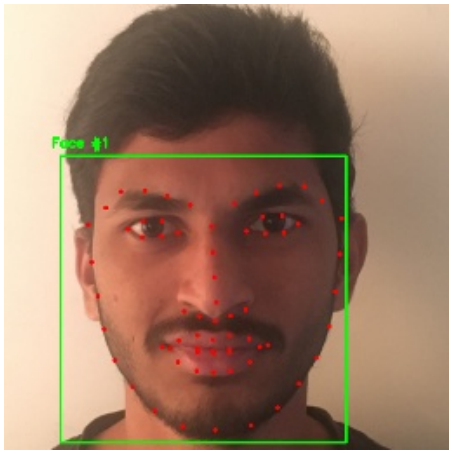
(d) Facial feature detection

(e) Original



(f) Facial feature detection

(g) Original



(h) Facial feature detection

Figure 1.3: Face and facial features detection using dlib, openface

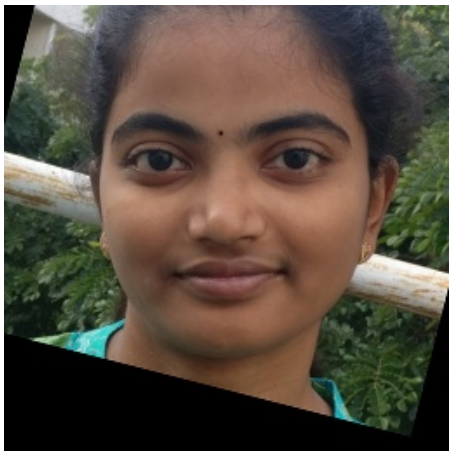
(a) Original

(b) Face Alignment

(c) Original

(d) Face Alignment

(e) Original



(f) Face Alignment

(g) Original

(h) Face Alignment

Figure 1.4: Face and facial features detection using dlib, openface

2 QUESTION 2

Face Recognition using Principal Component Analysis (PCA)

To solve the problem of Face Recognition we write PCA algorithm to find "eigen faces" i.e. eigen vectors. These eigen faces form a basis set of all images that are used to construct the co-variance matrix. This leads to dimensionality reduction because now we can use this smaller set of basis images to represent the original training images by projecting these training images on the eigen faces space.

A new image is classified by first projecting it onto the eigen faces space, and then finding the closest projection in the projection set of training images gives the label for the test image.

We follow the below steps to do face recognition:

1. **Data preparation**

   In this step we read in all the images in the gray scale format, flatten them to 1D array and then vertically stack the images to form (N X D) 2D array where N = Number of images and D = Dimension of the images = Height * Width.

2. **The PCA algorithm**

   a) *Finding the mean*

   In this step we find the mean across every dimension i.e. we get a 1D mean vector of length M.

   $$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

   b) *Computing the covariance matrix*

   $$S = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T$$

   c) *Finding the eigen values and vectors of S*

   $$Sv_i = \lambda_i v_i \quad i = 1, 2, ..., n$$

   d) *Reducing Dimensions*

   We select the K best eigen vectors (based on decreasing order of eigen values).

3. **Training**

   We project all training samples into the PCA subspace using the below equation, where x represents the training image and W represents the eigen vector matrix.

   $$y = W^T(x - \mu)$$

4. **Testing**

   We first project the query image unto the PCA subspace using the below equation. Then we find the nearest neighbor between the projected training images and the projected query image.

$$x = Wy + \mu$$

Results

Dataset used for testing is IIITB image dataset. We followed two different approaches for extracting faces from the dataset:

1. **Using Haar Cascade**

   - Eigen Faces extracted are shown in Figure 2.3
   - Below table in Figure 2.1 summarizes the results obtained

Number Of Eigen Faces Used	Accuracy (in %)	Rank 1	Rank 3	Rank 10
50	Training		100	
	Test	68.96	79.31	88.5
100	Training		100	
	Test	70.11	77.01	88.5
150	Training		100	
	Test	70.11	77.01	88.5
200	Training		100	
	Test	68.96	78.16	88.5
250	Training		100	
	Test	70.11	78.16	88.5
300	Training		100	
	Test	70.11	78.16	88.5

Figure 2.1: Summary Result - PCA

2. **Using Openfaces**

   - Eigen Faces extracted are shown in Figure 2.4
   - Table Figure 2.2 summarizes the results obtained for different number of eigen faces used.

Number Of Eigen Faces Used	Accuracy (in %)	Rank 1	Rank 3	Rank 10
10	Training		100	
	Test	59.21	67.1	85.52
20	Training		100	
	Test	64.47	76.31	86.84
30	Training		100	
	Test	65.78	77.63	89.47
40	Training		100	
	Test	68.42	77.63	89.47
50	Training		100	
	Test	68.42	77.63	89.47
60	Training		100	
	Test	67.1	78.94	88.15
70	Training		100	
	Test	67.1	80.26	88.15
80	Training		100	
	Test	68.42	78.94	88.15
90	Training		100	
	Test	65.78	80.26	88.15

Figure 2.2: Summary Result - PCA

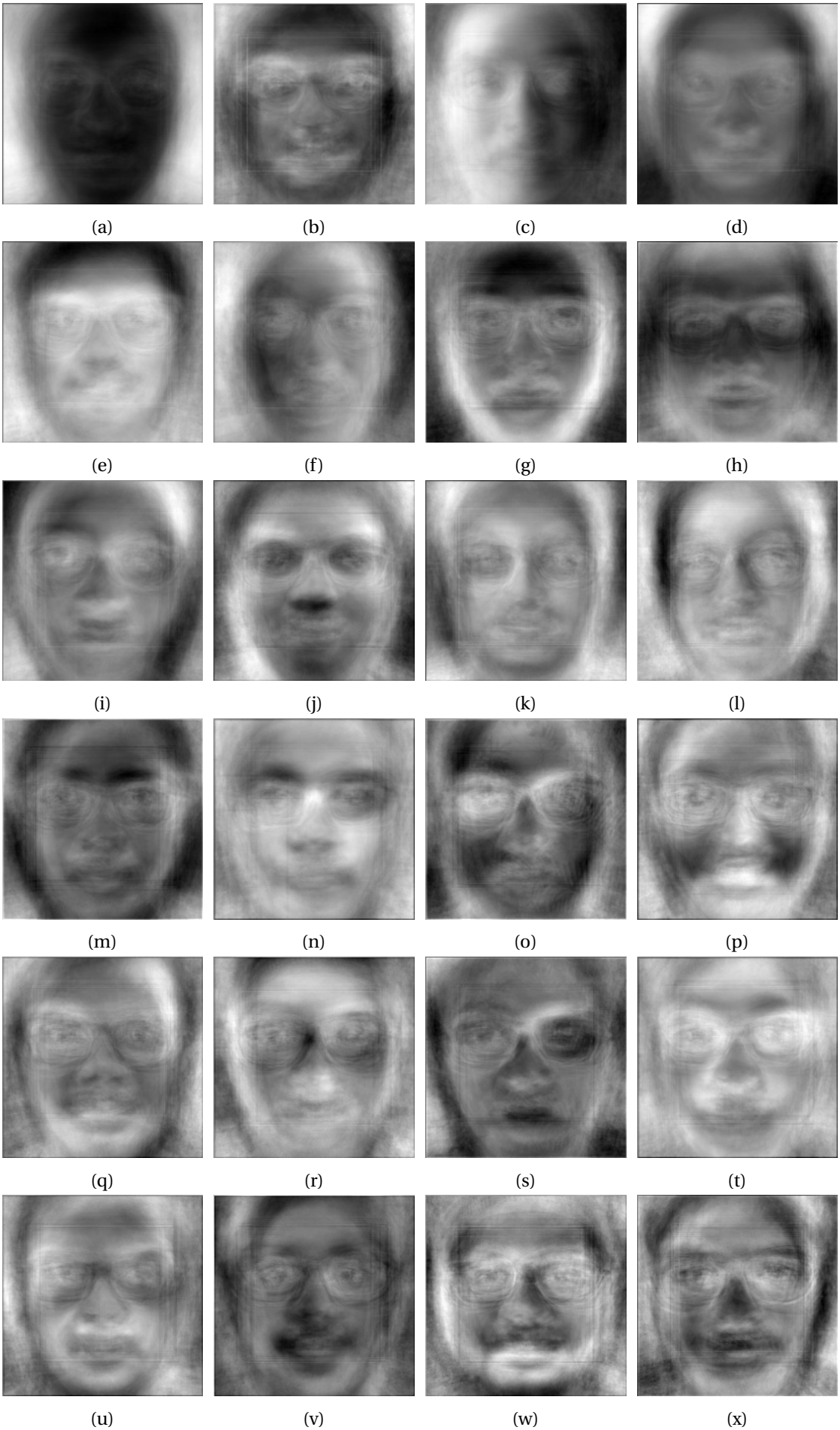


Figure 2.3: Eigen Faces detected in IIITB Face Dataset Using Haar Cascade as the face detector

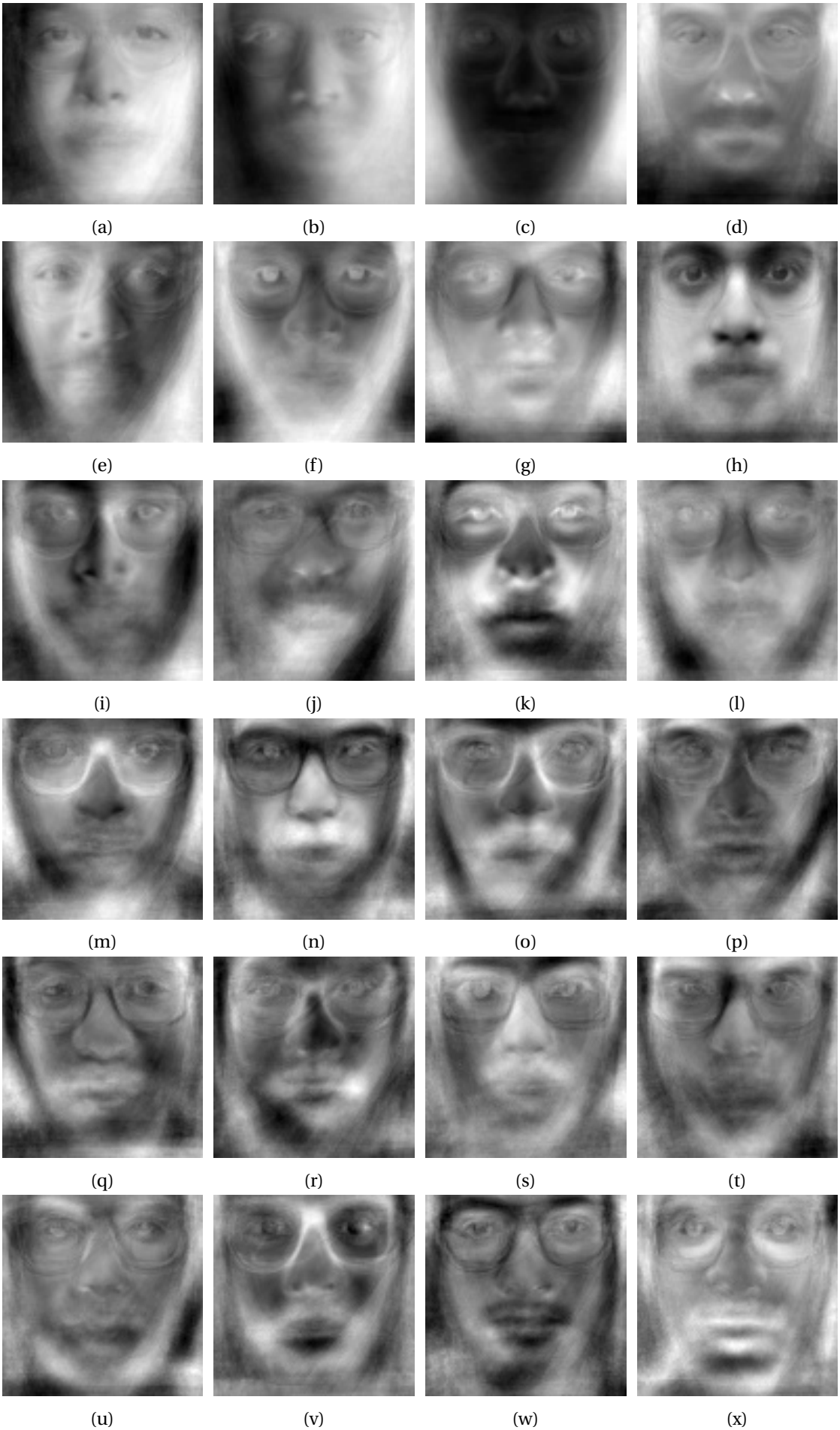


Figure 2.4: Eigen Faces detected in IIITB Face Dataset Using Openfaces Face Detector

3 QUESTION 3

Face Recognition using Linear Discriminant Analysis (LDA)

LDA works on the same principle as the eigenface method approach discussed before i.e. using PCA. It performs dimensionality reduction while preserving as much of the class discriminatory information as possible.

PCA doesn't use the concept of face classes, whereas LDA does and tries to increase the inter class variance.

Results

Dataset used for testing is IIITB image dataset. We followed two different approaches for extracting faces from the dataset:

1. **Using Haar Cascade**

   Below table in Figure 3.1 summarizes the results obtained

Accuracy (in %)	Rank 1	Rank 3	Rank 10
Training		100	
Test	36.36	41.81	58.18

Figure 3.1: Summary Result - LDA using Haar Cascade

2. **Using Openfaces**

   Below Table Figure 3.2 summarizes the results obtained

Accuracy (in %)	Rank 1	Rank 3	Rank 10
Training		100	
Test	78.8	85.43	95.36

Figure 3.2: Summary Result - LDA using Openfaces

4 QUESTION 4

Face Recognition using Local Binary Patterns (LBP)

LBP works as a texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number.

Algorithm

The LBP feature vector, in its simplest form, is created in the following manner:

1. Divide the examined window into cells (e.g. 16x16 pixels for each cell).
2. For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.
3. For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.
4. Where the center pixel's value is greater than the neighbor's value, write "0". Otherwise, write "1". This gives an 8-digit binary number (which is usually converted to decimal for convenience).
5. Compute the histogram, over the cell, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the center). This histogram can be seen as a 256-dimensional feature vector.
6. Optionally normalize the histogram.
7. Concatenate (normalized) histograms of all cells. This gives a feature vector for the entire window.

Results

Dataset used for testing is IIITB image dataset. We followed two different approaches for extracting faces from the dataset:

1. **Using Haar Cascade**

   Below table in Figure 4.1 summarizes the results obtained

Accuracy (in %)	Rank 1	Rank 3	Rank 10
Test	44.89	67.34	91.83

Figure 4.1: Summary Result - LBP using Haar Cascade

2. **Using Openfaces**

   Below Table Figure 4.2 summarizes the results obtained

Accuracy (in %)	Rank 1	Rank 3	Rank 10
Test	36.36	41.8	58.18

Figure 4.2: Summary Result - LBP using Openfaces

5 QUESTION 5

Explore an open source solution and compare

1. OpenFace is a Python and Torch implementation of face recognition with deep neural networks.
2. To setup the environment for running this method, we used docker container provided on their website.
3. We created our own data set inside docker.
4. This method follows four steps:
   a) Detect the biggest face
   b) Detect the facial landmarks (outer eyes, nose and lower lip)
   c) Warp affine to a canonical face
   d) Save output (96x96) to a file in an easy to access format
5. First we run align-dlib.py to get aligned images of extracted face from original face.
6. In the next step we extract feature from these aligned images using main.lua.
7. Then we train the model from generated features using classifier.py. This will create a file called classifier.pkl
8. We test the model by using classifier.pkl for an image. It gives label for given image with some confidence.
9. We changed original code to get top 1%, top 3%, top 10% accuracy.

Results
Dataset used for testing is IIITB image dataset.

Accuracy (in %)	Rank 1	Rank 3	Rank 10
Training		100	
Test	96.82	100	100

Figure 5.1: Openface Result

6 QUESTION 6

Gender Classification - Literature Review

After extensive search, we found that Local Binary Patterns (LBP) is the best method to do gender recognition (leaving out the deep learning methods).

Apart from LBP we also tried solving the problem using PCA and LDA and results shown below shows that LDA is the best method to do gender classification.

Other methods

Other methods for gender recognition are based on deep learning.

Dataset

For this we used the IIITB image dataset. We extracted appx. 700 faces using openface and took 20% of the dataset as the test data set.

Results Obtained

Following table summarizes the results obtained. And we can clearly see that LBP performs the best among all three classifiers.

Accuracy (in %)	Training	Test
PCA	100	88.78
LDA	100	89.71
LBP	100	97.19

Figure 6.1: Gender Recognition Results