Group Project 1: Face Recognition

Brahma Kulkarni (IMT2017011)
Kaustubh Nair (IMT2017025)
Sarthak Khoche (IMT2017038)
Vaishnavi Dhulipalla (IMT2017514)

2 Apr, 2020

1 Introduction

In today's era, face recognition is present everywhere, where a person looks at a camera, and the camera scans his/her face to recognize the person. Essentially, we can formulate face recognition as a classification task, where the inputs are images and the outputs are people's names(class labels). We're going to approach this problem using a popular technique for face recognition called **eigenfaces**, which underlies on **PCA**, an unsupervised dimensionality reduction technique.

Eigenfaces is a method that is useful for face recognition by determining the variance of faces in a collection of face images and use those variances to encode and decode a face in a machine learning way without the full information, reducing computation and space complexity.

2 Preprocessing

The dataset given to us consisted of 859 images. We applied various preprocessing techniques to make it easier to use for face recognition.

2.1 Detecting faces

We used a HAAR-feature based cascade classifier to detect faces in the images. Once the face was detected, we cropped the surrounding region to make it easy to detect other features such as eye, nose, jaw, etc. (Fig. 2)

2.2 Detecting eyes

We extended a similar approach to detect eyes in images. We used the HAAR-feature detector provided in the OpenCV library to find coordinates of eyes. By comparing the X coordinates of the eyes, we determined which one was the left eye and which was the right.

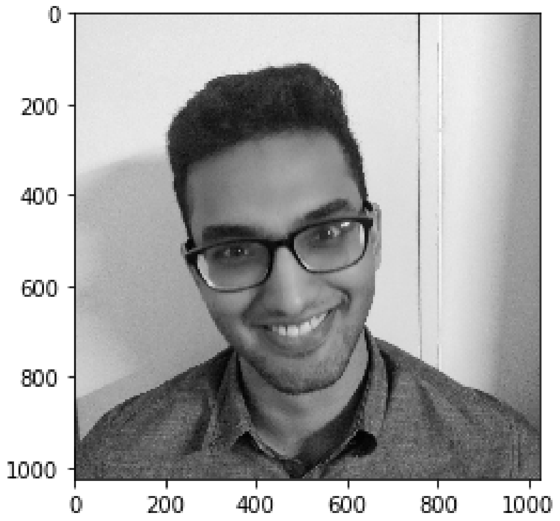


Figure 1: Base image (Taken from IIITB dataset; do not circulate.)

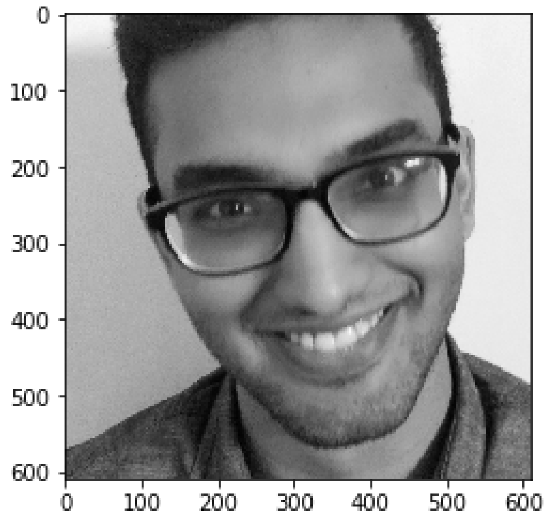


Figure 2: Face detection

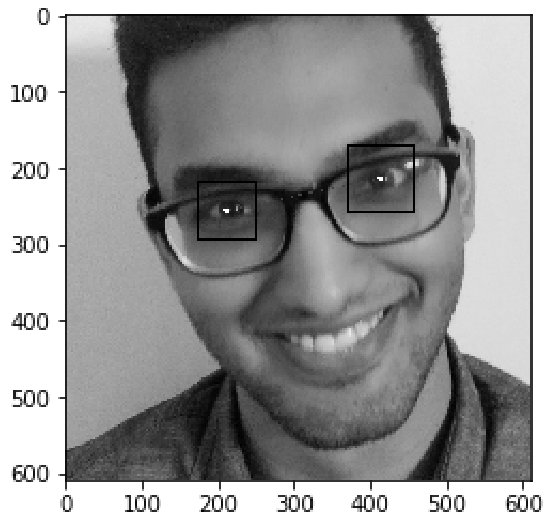


Figure 3: Eye detection

2.3 Locating center of eyes

The center of the eyes was determined using the bounding values of the rectangle obtained after eye detection. This center will be used for orienting the image so that both centers have the same Y coordinates.

2.4 Rotating image using eye orientation

We applied trigonometery to determine the angle by which the image needs to be rotated. Then, we rotated the image using OpenCV's 'image.rotate(angle)' function (Fig. 4)

2.5 Scaling image

We scaled all the images so that the distance between the eyes is 128 pixels. This makes it easier for the model to differentiate between the faces, since all the features of the same size for each face.

2.6 Drawing oval surrounding face

We detected additional facial landmarks like the bounds of the jaw to make it easier to detect the shape of the face. We used the facial landmark detector provided by dlib to determine the coordinates of various endpoints of the jaw. Then we computed the ellipse using these features which was applied as a mask on the image using a bitwise and operation. All the pixel outside this mask was set to black. (Fig. 5)

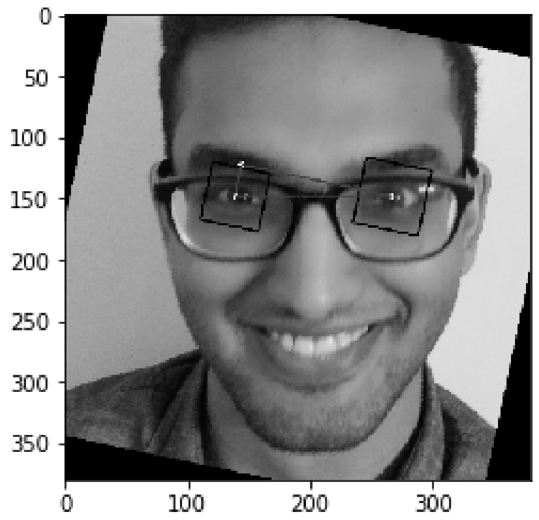


Figure 4: Rotating image based on eye coordinates

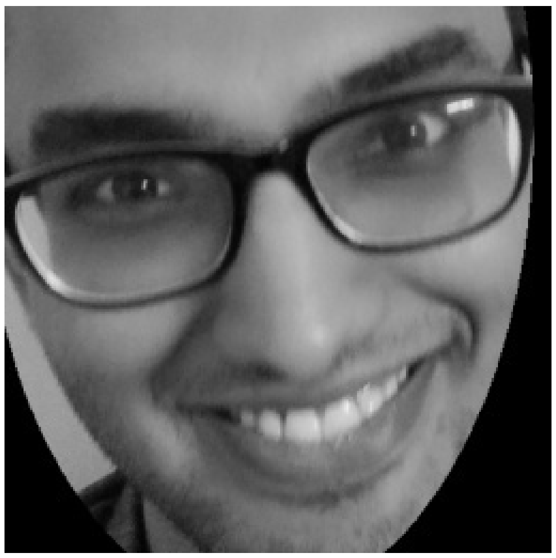


Figure 5: Drawing oval around face

3 Approach

The crux of our approach involves taking a high-dimensional image and projecting it down to a lower-dimensional space while retaining the essence, and then classifying the image(face) based on the smaller intermediate image. The essence of our approach lies in dimensionality reduction. Here we will describe the three popular dimensionality reduction algorithms.

3.1 PCA - Principal Component Analysis

We have used EigenFaces for Face Recognition(Section 5). At the heart of EigenFaces lies Principal Component Analysis or PCA. The idea behind PCA is to select the hyperplane such that when all the points are projected onto it, they are maximally spread out. Each of the principal components is chosen in such a way so that it would describe most of the still available variance and all these principal components are orthogonal to each other. In all, the first principal component is the one with maximum variance.

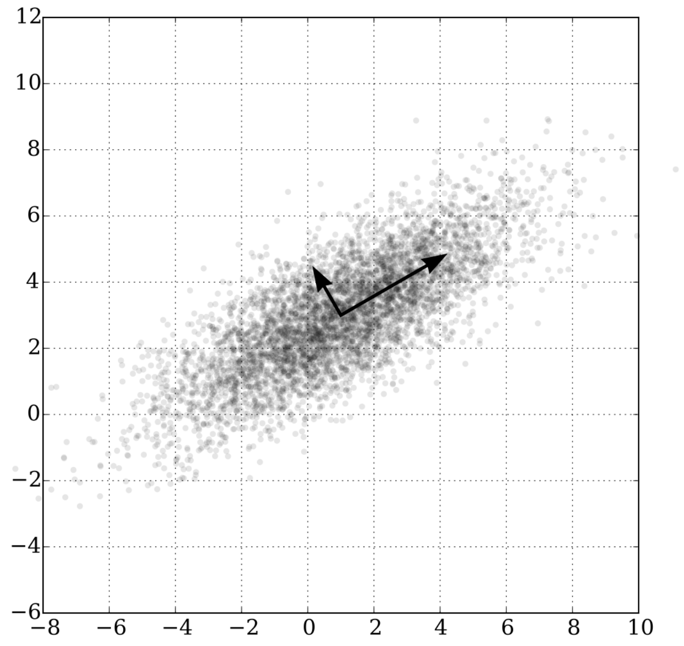
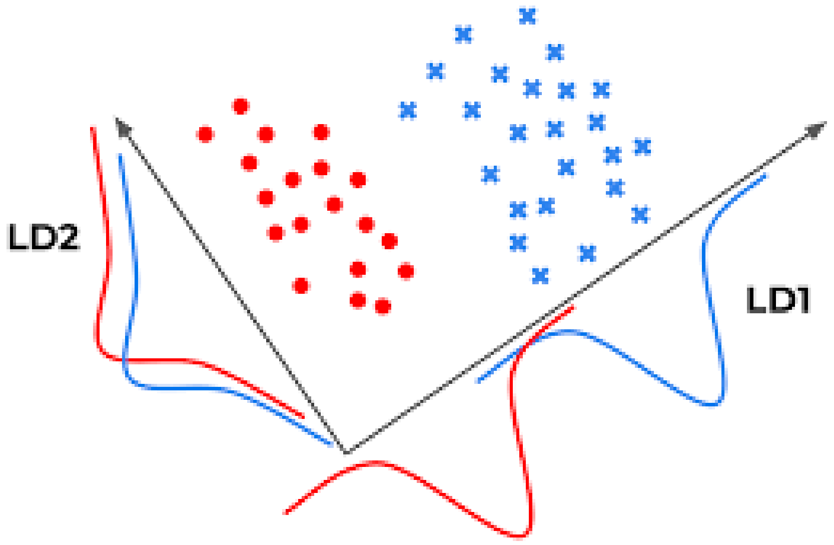


Figure shows the centered plot for some random data. In PCA, we want to find the axis with maximal variance. In the plot above, a potential axis is the x-axis or y-axis, but, in both cases, that's not the best axis(one with maximum variance). However, if we pick a line that cuts through our data diagonally, that will be the axis where the data would be most spread. If we were to project our points onto this axis, they would be maximally spread. We get this axis using eigenvectors. We start by computing the covariance matrix of our data and then compute eigenvectors of this covariance matrix. Then we pick the eigenvectors corresponding to the maximum eigen values. Those will be our principal axes and the axes that we project our data onto to reduce dimensions, by matrix multiplying it with the original data matrix. Using this approach, we can take high-dimensional data and reduce/project it down to a lower dimension by selecting the largest eigenvectors of the covariance matrix and projecting using those eigenvectors. Since we're computing the axes of maximum spread, we're retaining the most important aspects of our data.

Relating this to our problem of face recognition, we can conceptualize our processed n $\times$ n images as points in a n^2 dimensional space. Then, we can use PCA to reduce our space from n^2 into something much smaller. This will help speed up our computations and be robust to noise and variation

3.2 LDA - Linear Discriminant Analysis

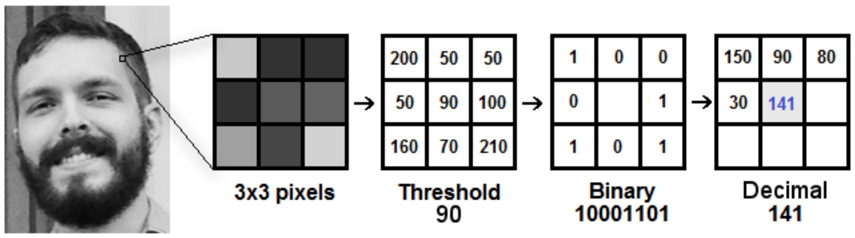
Linear Discriminant Analysis or Fisher's Discriminant Analysis is a supervised dimension reductionality algorithm. It projects the features in higher dimension space onto a lower dimensional space by preserving the class information as well, using the following steps:



1. It first calculates the separability between different classes(i.e the distance between the mean of different classes) also called as between-class variance/inter-class variance.
2. It then calculates the distance between the mean and sample of each class,which is called the within class variance/intra-class variance.
3. It the computes for a lower dimensional space which maximizes the between class variance and minimizes the within class variance, using an approach similar to PCA.

3.3 LBP - Local Binary Pattern

Local Binary Pattern is 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. The first step in constructing the LBP texture descriptor is to convert the image to grayscale. For each pixel in the grayscale image, we select a neighborhood of size r surrounding the center pixel. A LBP value is then calculated for this center pixel and stored in the output 2D array with the same width and height as the input image.



In the above figure we take the center pixel (90) and threshold it against its neighborhood of 8 pixels. If the intensity of the center pixel is greater-than-or-equal to its neighbor, then we set the value to 1 ; otherwise, we set it to 0. With 8 surrounding pixels, we have a total of $2^8 = 256$ possible combinations of LBP codes. From the binary value assigning, we calculate the LBP value for the center pixel(in a clockwise manner starting from top-left corner, as can be seen in the figure). This process of thresholding, accumulating binary strings, and storing the output decimal value in the LBP array is then repeated for each pixel in the input image. The last step is to compute a histogram over the output LBP array. Since a 3×3 neighborhood has $2^8 = 256$ possible patterns, our LBP array will have a minimum value of 0 and a maximum value of 255, allowing us to construct a 256-bin histogram of LBP codes as our final feature vector.

4 Implementation - Approach 1

In this approach, we have used PCA for feature extraction and then used various classification algorithms like Logistic Regression, KNN Classifier, Random Forest and SVM with linear kernel to predict faces(class label) associated with images of people. The implementation can be broken into the following steps :

- Preprocessing
- Preparing and Reading the Data
- Implementing PCA
- Transforming data using PCA (or other algorithms like LDA or LBP)
- Training the model
- Testing the model

Details of all the steps and the observations can be found in *approach1.ipynb*.

5 Implementation - Approach 2

In this approach, we have directly used EigenFaces, FisherFaces and LBPHFaces recognizer provided by opencv to predict faces(class label) associated with images of people. Let us first briefly describe each of these face recognizer.

5.1 EigenFace Recognizer

This algorithm considers the fact that not all parts of a face are equally important for face recognition. When it looks at multiple images, it tries to look for areas with maximum variation among faces, which'll help it differentiate one face from the other. Basically, it tries to extract the components which are relevant and useful (called the principal component) and discards the remaining components. And based on these principal components, it tries to find the best class label associated with the best match component.

5.2 FisherFaces Recognizer

FisherFaces recognizer is based on LDA (as seen in section 3.2), which tries to extract principal components from the image that helps differentiate one person from another. Doing so ensures that one's individual components do not dominate over others.

5.3 LBPHFaces Recognizer

LBPHFaces Recognizer is based on LBP (as seen in section 3.3), where we get a histogram of all decimal values based on the local binary patterns of all pixels. Hence for n images in our test set, we'll have n histograms for training. And hence for a new image in the test set, it compares it's histogram with the stored set of histograms to return the label associated with the best match.

The implementation of this approach can be broken into the following steps :

- Preprocessing
- Preparing and Reading the Data
- Detecting Faces
- Training using existing implementations of the said recognizers
- Testing and Computing top accuracies

Details of all the steps and the observations can be found in *approach2.ipynb*.

6 Future Works

A small application that came to our mind while implementing was to extend it to make it Real Time. This would then be beneficial for an application like a 'walk-in attendance system' making the entire experience seamless, time efficient and less "proxy" prone.