Executive Summary

We evaluate different face recognition techniques, for full frontal face images. These images are taken from the public domain and we form our training and testing data from such freely available images.

To achieve robust classification we do:

1. Preprocessing: Extraction of Face using a Face Detector and Histogram Equalization for balancing the contrast
2. We select features using the Subspace technique PCA and LDA.
3. Finally we classify using KNN and SVM classifier.

Apart from that we improved upon the basic algorithm in two new ways

1. Taking LDA of the PCA data and using these as the features
2. Pairwise-PCA, where we build a classifier by voting smaller classifiers which differentiate between each pair of persons in PCA space generated for those two persons only.

Our experiments reveal the LDA and PCA combined approach to be the best approach and gave an accuracy of 92% on the training set.

Table of Contents:

1. Aims and Objectives

The aim of this project is to identify known persons, from news pages, which have articles with images of famous celebrities and people in them. Our goal is to try out different methods of face recognition in this project, select the best ones and try to improve upon the result further.

The following objectives were laid down to meet above end:

- The initial objective is to find a method that achieves at least a 50% recognition rate to start with.
- This and other techniques will be further analysed and improved upon for a consistent performance upon images in the public domain.
- Also the scope of this project limits us to confine to full frontal faces

To realize the said objectives, we will be using a lot of the Opensource libraries and resources in the implementation and experiments.

2. Introduction

The history of Face Recognition dates back to as far 1960's, when the initial effort was done by marking the various points in a face, and it was a semi-automated process. The markers could be the different components of a human face – eyes, nose, lips, etc. Those days have long gone and with the advancement of computing and relentless research, it has become an essential genre of the pattern recognition exercise. And so advanced techniques and algorithm have been employed to achieve even 100% face recognition.

The main focus of our paper is News Images, since technology has made a lot of celebrity images are freely available on the internet. These images provide us a resource and a chance to explore what Face Recognition could do in the real world.

What does it take to recognize a Face?

When an image is given for computerized face recognition, how will a computer let us know if it contains a face and if it does how can we get an identification of who that face can be. Let us start with what the image means to the computer, or how the computer understands the image as.

A digital photograph or an image is a discretized version of the real world data. It is discrete in two respects:

1. It is made of pixels (***Sampling***) – discrete points of colour or gray-level.
2. It is measured in fixed levels of intensity (***Quantization***) – d discrete levels of intensity value.

An image is therefore an $(H\mathbf{x}\mathrm{W})$ dimensional vector where H and W are the height and width of the image. $I(x, y)$ gives the intensity at any point (x, y).

Now that we know what a digital image means, it is now possible to illustrate how we can take this data and read a known face from it, if present.

1.1 The Steps of Face Recognition

The following represents the scheme of things that a typical face recognizer does:

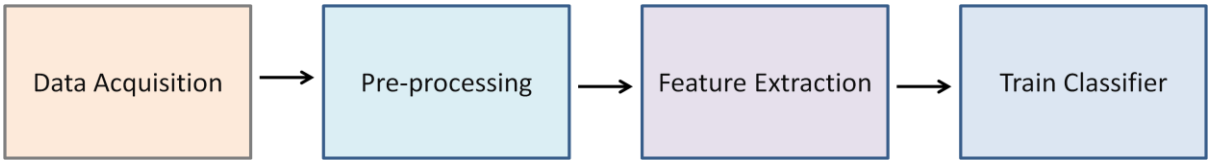


Figure1: Training Phase

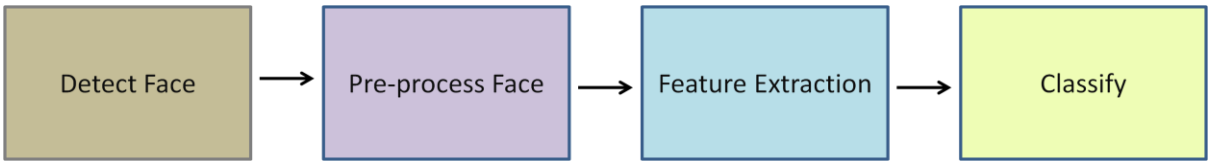


Figure2: Testing Phase

Training Phase:

First to build a model of the information used for classification, we select **data** which contains images of the person(s) that needs to be recognized. These raw images contain various noise and defects which can be unfavourable for a good classification. They have to be **pre-processed** to eliminate such discrepancies in the data. To operate on the raw pixels is cumbersome, so we need **features**, to get the right type information for efficient classification. Finally the classifier is **trained**, using the selected features for the data.

Testing Phase:

Now the trained classifier is ready to be **tested** any new data and classify it as positive or negative example, in this case reveal the identity (class) of the person. The first and foremost is the **face detection** step. Once the face is found and extracted, it needs to **preprocessed**, and then the features are extracted from it. It is these **features** which are then sent to the classifier for final **classification**.

1.2 Different Approaches

To start with, there are an ever increasing number of feature types, and a range of pattern recognition algorithms (classifier) to choose from. But they can be broadly put into the following groups:

1. **Global Approach**: The entire image is considered as the training vector(subspace methods –PCA, LDA, etc)
2. **Local Approach**: The components of the images are extracted and used separately (Local Features – LBP, Gabor, etc.)

In this thesis we have focussed mainly upon the **Global** or the Subspace methods, while select only one local method (Local Binary Patterns) as a baseline for comparison.

The above two approaches are assigned by considering the features used in the face recognition.

Once the features are selected, the classification can be done by using different algorithms which provide solution to the classification problem. We have chosen two approaches to study - Support Vector Machines and K Nearest Neighbour algorithm. Support Vector Machines claim to be more robust than KNN, by the very fact that they can be **kernelized** to form better decision boundary.

We have implemented these algorithms, and designed experiments around them, to analyse and improve the performance. The various parameters in the classifiers need to be optimized and we show how those values are arrived at. Finally an implementation using the URL of a news-page, for the task of face recognition is built. Let us begin by discussing the theoretical background of the core algorithms used.

3. Theoretical Background

In this section we discuss the theoretical aspects of the various approaches selected. First, we start off with the Face Detection Algorithm of Viola Jones used in the OpenCV Face Detector. Then we elaborate upon some of the principles of preprocessing step used – Histogram Equalization. Further we discuss the Feature Selection mechanism of Principal Component Analysis, Linear Discriminant Analysis and Local Binary Patterns. And finally we present the different classifiers, their theory and what are the meaning of the parameters used in them.

3.1 Face Detection - Viola Jones Algorithm

We use OpenCV's Face Detector which is based upon the **Viola Jones Algorithm** for Object Detection using features known as **Haar Features**, which are computationally very inexpensive and fast

The Viola Jones algorithm for object detection was introduced by Paul Viola and Michael Jones in 2001. The original Viola Jones paper came up with the following three new concepts. [1][2]

1. **Integral Images**: An intermediate representation of the original pixel image, in which each pixel value in the new Integral Image is calculated by summing up the intensity values of all pixels to the left and on top of it. The main advantage of using Integral Images is that they allow the fast computation of the Haar-like features which are used for the object detection.

2. **AdaBoost**: Also known as adaptive boosting involves building a strong classifier by combining numerous weak classifiers, and giving a weighted sum of the weak classifiers as the output. Adaboost is used not only to train the classifier but also for a greedy selection of the best features

3. **Attentional Cascade**: The attentional cascade helps increase the performance of the classifier by focussing on the probable object region and rejecting the non-face region. The word "cascade" in the classifier name means that the resultant classifier consists of several simpler classifiers (stages) that are applied subsequently to a region of interest until at some stage the candidate is rejected or all the stages are passed. It forms a degenerate decision tree in which the negative sub-windows are rejected as early as possible in the decision process.

3.1.1 HAAR – like features

The motivation of using features instead of pixels lies in the fact "that the feature-based system operates much faster than a pixel-based system". [2]

The Haar Features are based on Haar Wavelets. Wavelets are functions which integrate to zero. The Haar Wavelet has the following mother wavelet (function).

$$\psi(t) = \begin{cases} 1 & 0 \leq t < 1/2 \\ -1 & \frac{1}{2} \leq t < 1 \\ 0 & otherwise \end{cases}$$

Haar Wavelet Function (1)

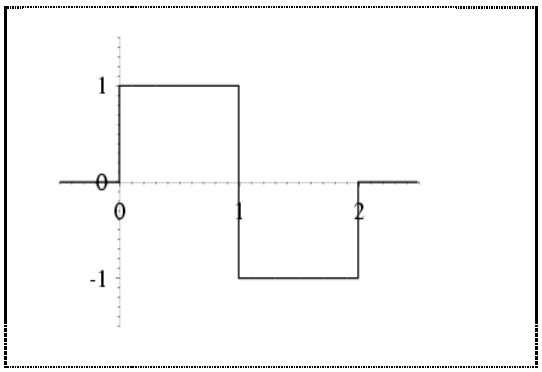


Figure 3:One dimensional Haar Wavelet

The Haar wavelets are a natural set of basis functions which encode differences in average intensities between different regions. [5]. Since we need better features for object recognition, the features used in Viola Jones are not true two dimensional Haar Wavelets, i.e., not just simple adjacent rectangular blocks but of different combinations of blocks. Hence they are known as ***Haar-like*** Features.

The figure below shows the different Haar-like Features used in the original Viola Jones approach.

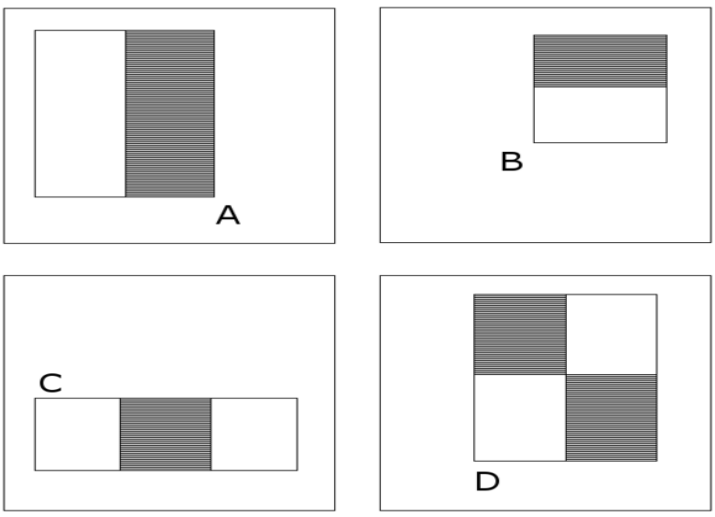


Figure 4: The Haar Features used by Viola Jones.(Adapted from[6])

The value of a two-rectangle feature is the **difference** between the **sums of the pixels** within two rectangular regions. A three-rectangle feature computes the sum within two outside rectangles subtracted from the sum in a centre rectangle. Finally a four-rectangle feature computes the difference between diagonal pairs of rectangles. [2].

The current implementation of Viola Jones algorithm in OpenCV uses the following set of Haar-like Features, shown in the following figure below. This is the set of **Extended** Haar-like Features which was introduced by [8], shown in Figure 5.

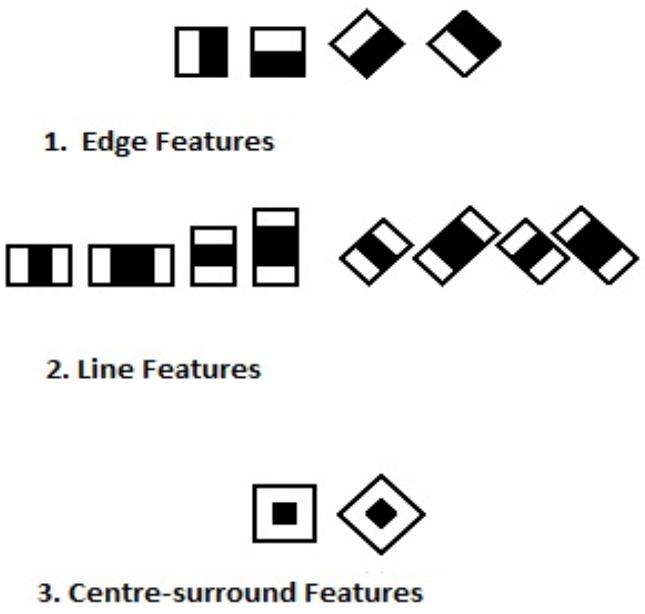


Figure 5: Extended Haar Features used by OpenCV Face Detector (Adapted from [7])

And unlike the Haar Wavelets which the two dimensional Haar features is an $over - complete1$, which means that they are more features than the number of elements (pixels) in the image space and also they are not a linearly independent set. Given that the base resolution of the detector is 24x24, the exhaustive set of rectangle features is quite large, over 180,000. [1].

In [5] two important advantages of working with the Haar Features, are described. One is that, Haar Features help encode the presence **of visual features such as boundaries,** etc., at different scales and orientation. This is easily known by the difference in the intensity levels as is represented in the Haar basis. The second observation is that the Haar Features, as illustrated above being over-complete helps to **capture complex patterns**, not just the ones given by the rectangular features.

For a detailed analysis of the Haar Features and its application the reader is referred to [5] and [8].

3.1.2 Integral Images

The use of **Integral Images** in Viola Jones is breakthrough in itself, because by the using the Integral Images the computation of the Haar Features can be achieved in **constant time**.

Before the Haar Features are calculated on the pixel images, the input images are converted into an **intermediate** representation known as Integral Images. An Integral Image is nothing but a pixel represented as a **sum of all the pixel values** which exist on its left and above it, as shown in the shaded region in the figure below.

$$ii(x,y) = \sum_{i \le x, j \le y} i(i,j) \qquad \textbf{(2)}$$

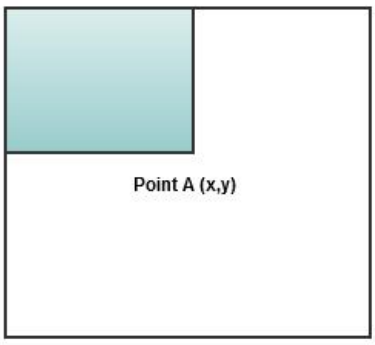


Figure7: Region used in calculation of the Integral Image

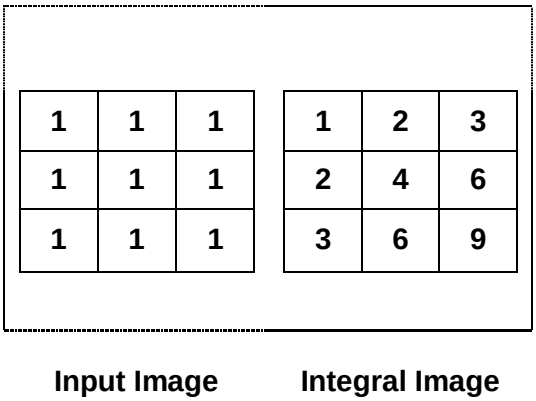
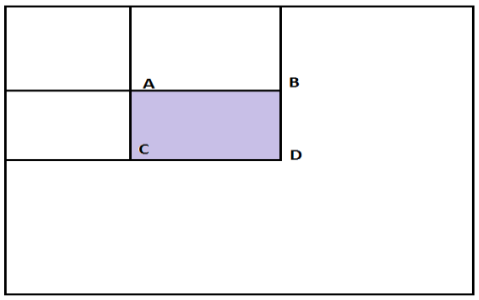


Figure8: Calculation of an Integral Image

And by using this Integral Image representation the sum of all the pixel values in any rectangular area of the image can be easily calculated by just using the memory access of only four locations at the most. This is explained as shown in Figure 9.



In the figure, the sum of the pixel values in the coloured region is given by the following calculation:

$$Sum\ of\ Rectangle = D - (B + C) + A$$

Figure9: Calculation of the sum of a rectangular region.

Thus by accessing the values of the integral image at four corners A, B, C and D, respectively, it is possible to arrive at the sum of the intensity values of any given rectangular region of the image.

"Clearly the difference between two rectangular sums can be computed in eight references. Since the two-rectangle features defined above involve adjacent rectangular sums they can be computed in six array references, eight in the case of the three-rectangle features, and nine for four-rectangle features." **[1].**

3.1.3 AdaBoost

AdaBoost or Adaptive Boosting, as a technique for classification was first introduced by Yoav Freund and Robert E. Schapire in [9].

Boosting refers to the general problem of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb [9].

The following algorithm is the modified version of AdaBoost which is used in the Viola Jones face detector. [2]

- Given example images(x_i, y_i): $i = 1, \ldots, n$, and $y_i = 0, 1$ for negative and positive examples, respectively.
- Initialize $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$, where m and l are the numbers of positive and negative examples, respectively.
- For $t = 1, \ldots \ldots, T$:
    1. Normalize the weights, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} wt,j}$ { so that w_t is a probability distribution }
    2. For each feature j, train a weak classifier h_j, then the error evaluated is
    $$\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|.$$
    3. Choose classifier h_t with the lowest error ϵ_t.
    4. Update the weights:$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$
    where $e_i = 0$ if example x_i is correctly classified, while $e_i = 1$, if otherwise
    and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$
- The final strong classifier is:
$$h(x) = \begin{cases} 1, -- \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0, ---------- \boldsymbol{otherwise} \end{cases}$$
where, $\alpha_t = log \frac{1}{\beta_t}$

Algorithm 1: Modified Adaboost in Viola Jones [2]

Adaboost assigns **weight** to each data point. Upon subsequent iteration of the classifier, this weight is modified such that the weights of the correctly classified instances are reduced, while those of the incorrectly classified ones remain unchanged. This method is used to **increase the relative penalty for incorrectly classified instances** as against the correctly classified ones.

The AdaBoost is not only used for training the face detector but also, used to evaluate **the best features** out of the many (160000 for a 24x24 pixel image), through a brute force comparison.

3.1.4 Attentional Cascade

In Viola Jones AdaBoost is used to train several classifiers which are arranged in a rejection cascade as shown in the figure below. Each classifier in the cascade **rejects** as many negative examples as it can. It is based upon the observation that majority of the sub-windows are negative. This acts like a degenerate decision tree of some kind in which the **positive example need to be accepted by all the nodes of the decision tree**.

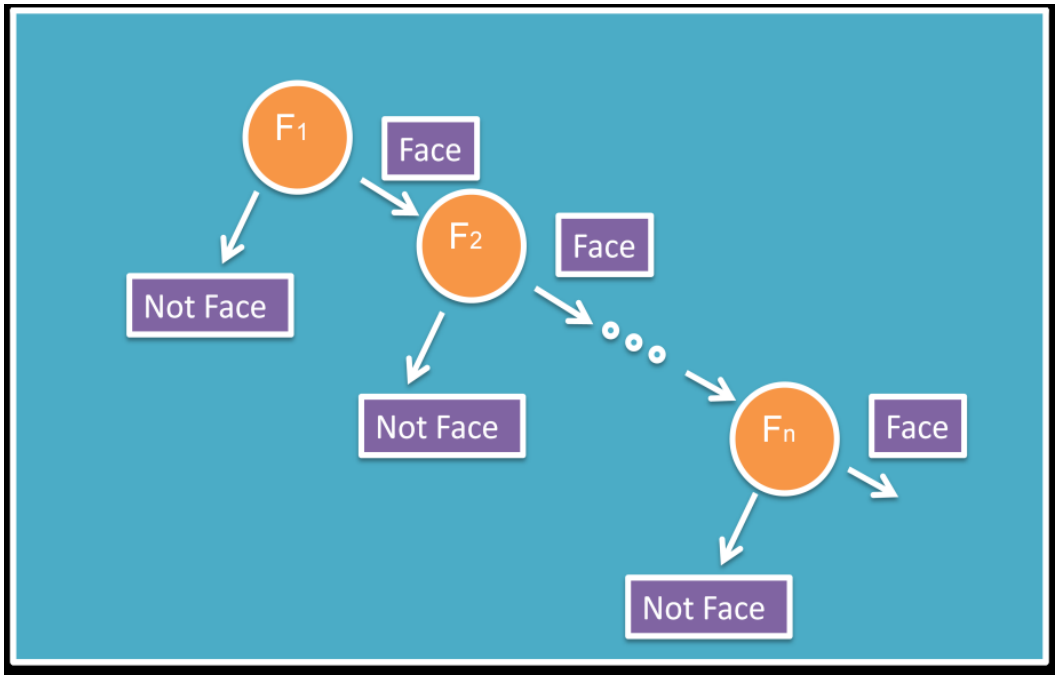


Figure 10: The detection cascade

3.2 Histogram Equalization

Preprocessing is a very important step in Face Recognition. And especially since we use the whole image as a vector in our subspace classification technique, it can greatly affect the closeness of the data points within each class.
Histogram Equalization redistributes the image intensity over the entire image more uniformly.

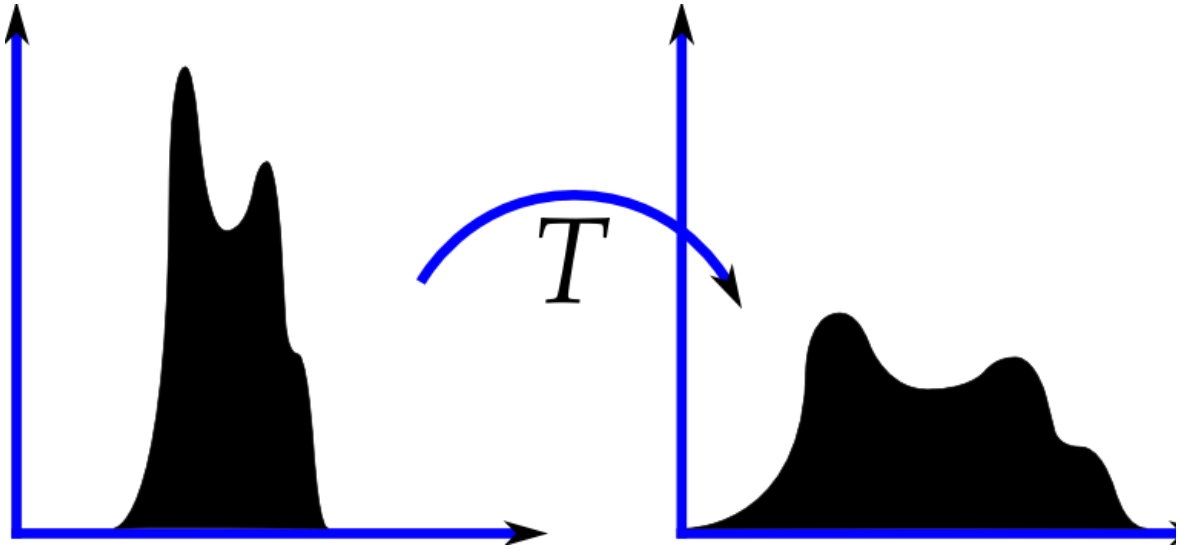


Figure 11: Image Histogram before and after the transform.(Courtesy [11])

"The histogram manipulation, which automatically minimizes the contrast in areas too light or too dark of an image, consists of a nonlinear transformation that it considers the accumulative distribution of the original image; to generate a resulting image whose histogram is approximately uniform."[12]

Grayscale

Histogram Equalized

Figure 12: Grayscale Image and its Histogram equalized version

The above figure shows a greyscale image against its histogram equalized counterpart.

3.3 Principal Component Analysis (PCA)

Principal Component Analysis, also called discrete Karhunen-Loeve Transform, is a technique of dimensionality reduction which had been used in many areas of Image Processing. It was first used as a mechanism of Face Recognition in 1991, by Turk and Pentland in [13].

PCA works on the logic that the maximum information about the data vectors (images) lies on the first few axes generated by the Principal Components Analysis.
Let us see how the PCA axes are obtained. Supposing our Training Set $\mathbf{X}$ consists of n vectors

$$\mathbf{X} = \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_1} \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{x_n} \end{bmatrix}$$

Let the mean of the data set be denoted by $\boldsymbol{\mu_x} = \sum_{i=1}^{n} \mathbf{x_i}$.

The Covariance Matrix is given by $\boldsymbol{C_X} = \mathbf{E}\{(\mathbf{x} - \boldsymbol{\mu_x})(\mathbf{x} - \boldsymbol{\mu_x})^{\mathbf{T}}\}$. This Scatter matrix is a measure of the variance in the data set. Therefore upon doing an Eigen Decomposition of this covariance matrix we get the basis vectors (the Eigen Vectors) along which the data **variance is maximum** for the highest Eigen Values

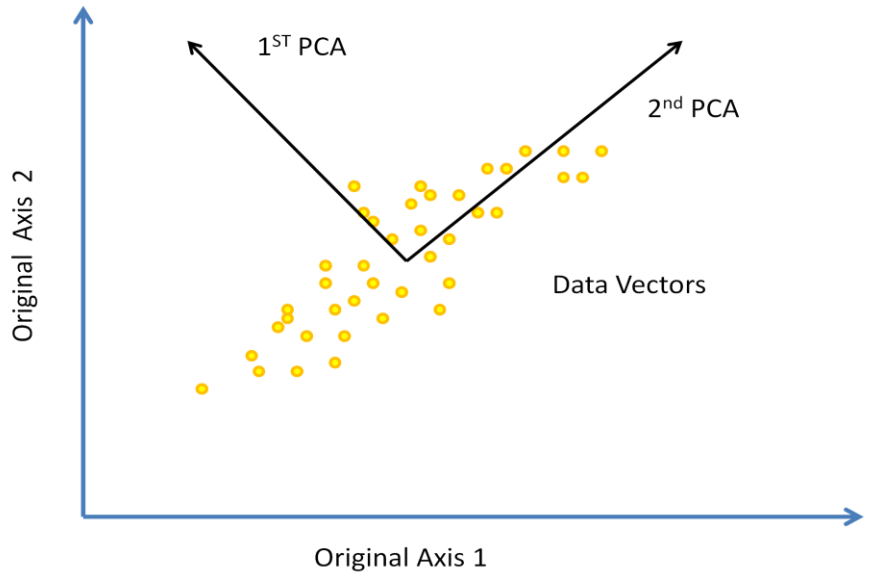


Figure 13: Principal Component Analysis

The first Principal Component is the Eigen Vector with the highest Eigen Value and the second the next highest and so on. The data is projected into a much higher dimension than

the original image, but **data compression** is achieved by selecting only the first few Eigen Vectors, and the rest can be discarded.

This axes generated by the PCA is used in Face Recognition and are known as **EigenFaces,** since they resemble human faces when rendered into gray-scale. They are thus defined as the "Eigen Vectors of the Covariance Matrix of the probability distribution of the vectors of human faces"[17]. Some of the First Eigen Faces generated in our experiments are shown here.

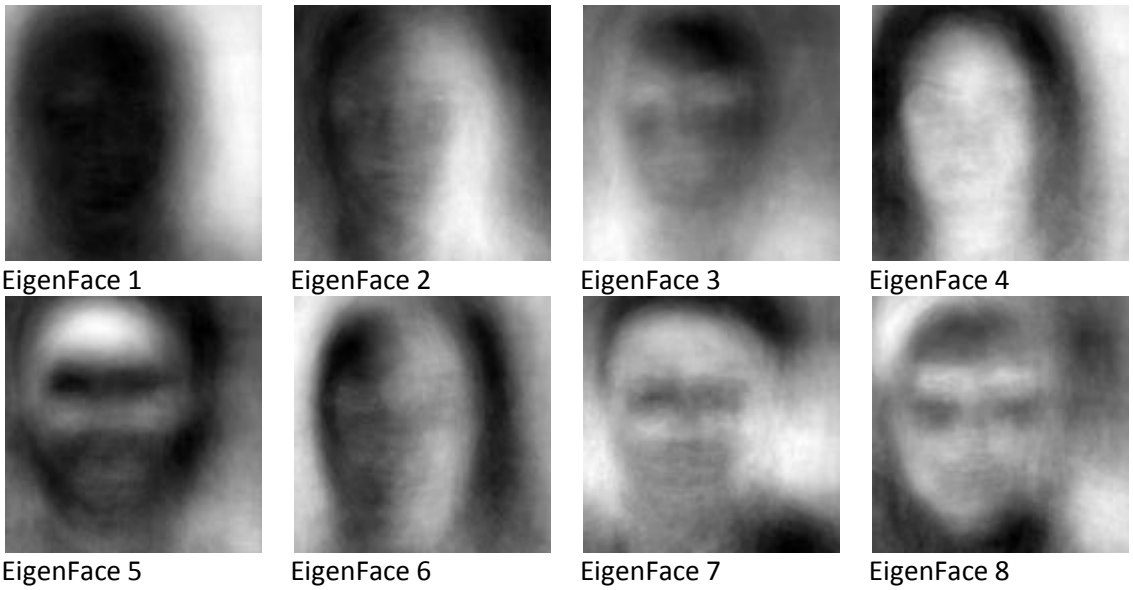


Figure 14: The first eight Eigen Faces obtained in Training.

The Average Image
Returned by the OpenCV PCA class

Example Input Image
In Gray-Scale with Histogram Equalization

The Example Projection
Projected onto the Eigen Space

Figure 15: The Average Image and a Projection of a face.

The image on the left is the average image returned by OpenCV PCA.

The one on the right is an image in gray-scale and its projection in PCA.

3.4 Linear Discriminant Analysis (LDA)

LDA is a subspace method in which the training data is projected into a space where the maximum discriminatory information lies on the axes of the LDA subspace.

The difference between PCA and LDA is that LDA maximises the **separability** between the classes whereas PCA finds the axes which contain maximum information.

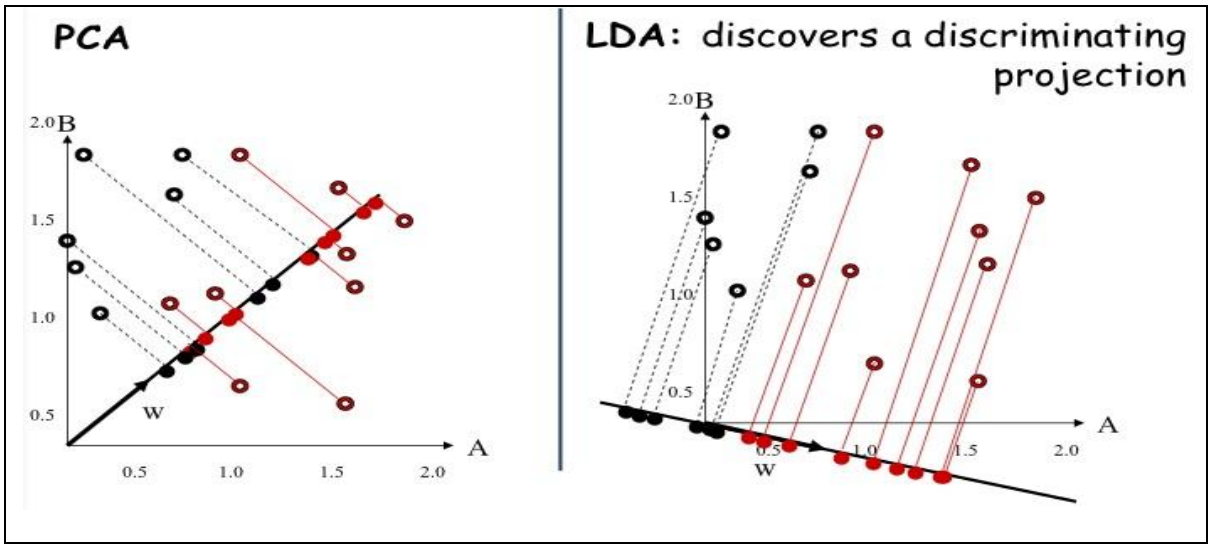


Figure 16: Comparison between PCA and LDA (Adapted from [18])

Linear Discriminant Analysis is also known as **Fisher's linear discriminant.**
LDA is particularly beneficial when the classes are very well defined. Through LDA we find the hyperplanes along which the separation between the classes is maximized. And then when the data-points are projected onto these hyperplanes, they are scattered far as apart as possible, yielding high discrimination between members of the different classes.

The number of LDA axes is generally **(*N-1*)** where ***N*** is the number of classes in the dataset. Looking at this LDA can be used for pattern recognition in the data, while at the same time achieving a dimensionality reduction. A few explanations of the process of LDA will make it clear.

In Linear Discriminant Analysis, unlike PCA, the aim is to **maximize** the between-class scatter (co-variance) of the training set, against the within-class scatter (co-variance). It is a matter of making the **ratio of the between-class scatter to that of the within-class**, as large as possible.

Here is a brief discussion how we calculate the LDA axes from the dataset.

In the simple case of a Two-class Problem, the ratio or the criterion that is needed to be maximized is the following.

$$J(w) = \frac{|\widetilde{\mu_1} - \widetilde{\mu_2}|^2}{\widetilde{s_1}^2 + \widetilde{s_1}^2} \quad \textbf{(3)}$$

Where $|\widetilde{\mu_1} - \widetilde{\mu_2}|^2$ is a measure of the distance between the projected means, in the new vector space, and $\widetilde{s_1}^2 + \widetilde{s_1}^2$ is the total within class variance of the two classes in the projected space.

This condition will be further generalized in the case of problems of multi-class nature, such as ours. And so the multi-class criterion is similarly arrived, as demonstrated below.

Each image vector *'x_i'* belongs to a class out of *'c'* classes, and here again the required objective is to find the projection space with axes *'w'*, which satisfy a condition similar to the above, in the multi-class environment.
The Within Class scatter or per-class scatter is computed as follows: It is an approximation of the co-variance among the members of the same class, of how far apart the data is held together in its own group.

$$S_W = \sum_c \sum_{i \in c} (x_i - \mu_c)(x_i - \mu_c)^T \quad \textbf{(4)}$$

Where x_i represents each image vector, and μ_c the corresponding the mean vector of the class the image belongs to.

Subsequently the Between Class Scatter is also calculated as follows:

$$S_B = \sum_c (\mu_c - \bar{x})(\mu_c - \bar{x})^T \quad \textbf{(5)}$$

Where μ_c is again the mean vector belonging to each class of the images (for each person), and $\bar{x}$ is the average vector of the whole set.
Likewise the goal of Linear Discriminant Analysis is then to find the space of the hyperplanes ***W*** maximize the following quantity.

$$J(w) = \frac{w^T S_B w}{w^T S_w w} \quad \textbf{(6)}$$

Where S_B is the Between Class Scatter and S_w Within Class scatter of the dataset, as given by the equations (4) and (5), above.

The process of evaluating the LDA axes $\boldsymbol{w}$, is straight forward from now on, as equation (6) reduces to a simple generalized Eigen value problem.

$$\mathbf{S_B W} = \boldsymbol{\lambda} \mathbf{S_w W} \quad \textbf{(7)}$$

This is easily computed in our OpenCV implementation by using the Singular Value Decomposition Function "*cvSVD()*"**.**

3.5 Local Binary Patterns (LBP)

We choose one of the Local Features methods, just as baseline to compare it against the Global Image approaches that we have discussed and used in this project. The local feature technique chosen is the Local Binary Pattern or LBP for short. These are nothing but a scale-invariant local texture function, originally developed by Ojala *et al.* in [20][21]. I have used the technique specified in [19], and have implemented the most basic version of it. Our aim was not to fully implement it, but to contrast it's performance against the Global Features approach that we have been following.

What is LBP?

LBP as we have said is a local texture operator. The operator labels the pixels of an image by thresholding the 3x3-neighbourhood of each pixel with the centre value and considering the result as a binary number. Then the histogram of the labels can be used as a texture descriptor.[19].

10	11	2
7	8	1
9	5	1

1	1	0
0		0
1	0	0

Binary: 11000010

Decimal: 194

Figure17: Calculation of LBP

The newly generated texture based LBP images look like the following, next is the corresponding histogram of the whole LBP image.

Original Gray-Scale Image

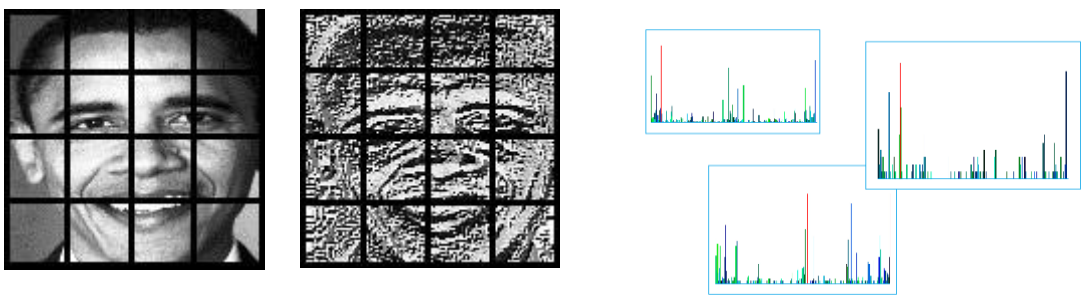
After Applying the LBP Transform

What the Histogram of the LBP looks like

Figure 18: Original and LBP image with Histogram

For an LBP, we need to compute the Local Histogram, not the overall histogram as shown above, but the local histogram looks similar and the picture is just to give the reader an idea about what the histogram data looks like.

The next step in getting the LBP descriptor of the image is to divide the image into blocks and then we calculate the Histogram values of each block.
Implementation: The image size of 130x130 returned a 128x128 LBP Image, as shown in the pictures (1) and (2). And it is further divided into blocks of 32x32, yielding 64 in all.



The image and Its LBP version with the blocks

Compute LBP Histogram for each of the blocks

Figure 19: Computation of block LBP Histogram

The Histogram of all the blocks for each image is stored, as the model, during the training phase.
The testing phase involved finding the nearest neighbour among the histogram of the LBP images.
The test image has to again go through the entire process of LBP generation and subsequent Histogram creation. After the pre-processing, it is weighted (or not weighted) and classified using the nearest neighbour approach for each block. The resulting class is obtained by voting among the nearest neighbour class of each block.

3.5.1 Chi-Square Histogram Distance

There are different methods or metrics to measure the distance between two histograms, in finding the nearest neighbour among the LBP Histogram of the images – Log-likelihood, Histogram Intersection, etc. We have used the Chi-Squared Distance metric, as it compares each point of information on the histogram, using a squared difference technique, as shown below.

$$\chi^2(\mathbf{X}, \mathbf{Y}) = \sum_i \frac{(X_i - Y_i)^2}{X_i + Y_i} \quad \textbf{(8)}$$

3.6 KNN Classification

KNN or K Nearest Neighbour is one of the few **Lazy** Learner classification algorithms that has been used widely in the field of Pattern Recognition.

The KNN classifier does not build any explicit model of the data, but is an **instance based learner** and needs to evaluate all points in the vector space of the data to arrive at the final classification.

The KNN algorithm, takes only one parameter "***k"*** which is the number of nearest neighbour, according to the calculated distance in the vector space. The training phase is apparently the optimization of this parameter "***k"*** – the number of neighbours.

1. Train by keeping all instances of the data set in the feature space
2. For each test image, evaluate the k nearest neighbour based on the distance metric used
3. Find a class to assign to the test data, by weighted voting of the k nearest neighbour

Algorithm 2: KNN

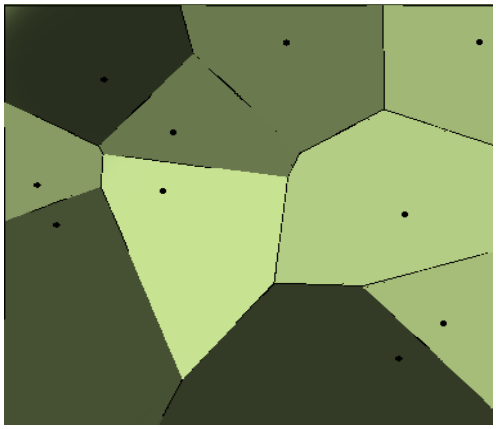


Figure 20: Voronoi tessellation – decision boundary of 1NN(Adapted from[22])

KNN classification determines the decision boundary locally, by taking into consideration each data point in the training set. In the case of 1 Nearest Neighbour or 1NN, the shape of this decision surface is given by a **Voronoi Diagram** or **Voronoi Tessellation**, as shown here.

The **Voronoi** diagram divides the entire vector(metric) space into regions(subsets of points), where each instance(object, in our case images) is associated with a Voronoi cell in which all points in the cell has the least distance to the object associated with cell when compared other objects in the metric space. When Euclidean distance is applied, the region is a convex polygon, surrounding the image point.

By default **1NN** is highly susceptible to **noise** in the training data, cannot perform as a robust classifier since the classification relies on the class of a single training document, which may be incorrectly labelled or atypical. [23]

This is improved upon by using K neighbours instead of just 1. The common method is to decide upon the winner by finding the **majority class**. Also there is a probabilistic approach which gives probability ratio to each class within the neighbourhood [23]. OpenCV does this by voting using calculated weighted sum.[25]. One way to do this **weighting** by cosine similarities.[23]. In the case of text documents similarity based upon the cosine distance (the angular distance) between the vectors gives a good result.

While selecting **k**, in order to get a fairer vote, generally k value is odd. Typically chosen to be **k = 3** or **k = 5**.

3.6.1 Does KNN achieve good classification?

In [25], the authors have demonstrated by experimental analysis, that there are cases in which KNN performs well and cases where failure takes place. Most importantly, they have established that the **Curse Dimensionality** (which is so famous in machine learning) can deliberately undermine the effectiveness of the KNN model. They show that as the number of dimensions or features increase the *contrast* [25] in the distance of any input point to its nearest neighbour against other points (images) in the image space becomes non-existent. In other words the probability of any point being a neighbour converges at higher dimensions.

Experimentally, in[25], they have shown that by the 10^{th} dimension, this contrast in distance between the nearest neighbour and all other points is reduced by a factor of 6 and at 20 by almost 4, at which point the algorithm is degenerative.
To overcome this, they emphasize, that the data points of different classes should be diversely located –as far apart. We can achieve this in the image data by highly effective pre-processing techniques.

3.7 Support Vector Machines (SVM)

Support Vector Machines belongs to the group of **maximum margin classifier**. Support Vector Machines maximizes this margin distance from the test data point to the classifier boundary. Generally SVM's are used for a two-class problem (+1 vs. -1), but they have also been extended to be used in the problems of multi-class nature.

The decision boundary of SVM is an $\boldsymbol{n-1}$ **dimensional hyper-plane** which is selected to separate the classes in an n dimensional space. In the case of 2 dimensions, it reduces to the problem of finding a line (1 dimensional hyperplane which is a straight line) from which the perpendicular distance of the margin points are the largest. By **margin points** it is referred to the points on the two different classes which lie **nearest** to the decision boundary. The margin points are shown, in following image, by the circled dots in dark and light regions (indicative of the separate classes) on either side of the margin lines.

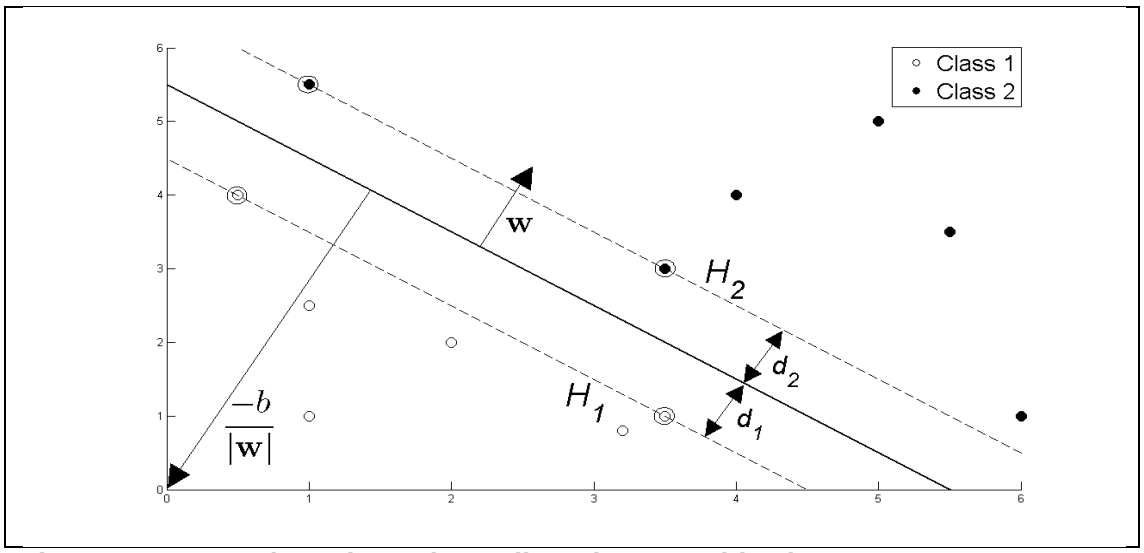


Figure 21: Hyperplane through two linearly separable classes

The n dimensional hyperplane is given by the equation:

$$\mathbf{w}.\mathbf{x} + \mathbf{b} = \mathbf{0} \qquad \textbf{(9)}$$

Where $\mathbf{w}$ is the normal to the hyperplane $\mathbf{b}$ is the offset on the origin, rather $\frac{\mathbf{b}}{\|\mathbf{w}\|}$ is the perpendicular distance from the origin to the maximal hyperplane. **[8].**

For the two different classes (+1 and -1), the margin points are exactly the points which satisfy

$$\mathbf{x_i w} + \mathbf{b} = +\mathbf{1} \quad \text{And} \quad \mathbf{y_i} = +\mathbf{1} \qquad \textbf{(10)}$$

$$\mathbf{x_i w} + \mathbf{b} = -\mathbf{1} \quad \mathbf{y_i} = -\mathbf{1} \qquad \textbf{(11)}$$

And these are the points that lie on the hyperplanes H_1 and H_2, while all other point lie farther away.
For all other points in two sets

$$\mathbf{x_i w + b} > +1 \quad \text{And} \quad \mathbf{y_i} > +1 \tag{12}$$

$$\mathbf{x_i w + b} < -1 \quad \mathbf{y_i} < -1 \tag{13}$$

Combining both equations above gives

$$\mathbf{y_i(x_i w + b)} \geq +\mathbf{1} \tag{14}$$

The aim of SVM is to maximize the margin distance between H_1 and H_2 which is equal to $\frac{1}{\|\mathbf{w}\|}$. With this objective, the equation is similar to minimizing the cost $\frac{1}{2}\|\mathbf{w}\|^2$.

3.7.1 Introducing the Soft margin

To allow for a few misclassified items, a margin error penalty, $\boldsymbol{\xi_i}$, so now we treat the data as

$$\mathbf{x_i w + b} > +1 - \boldsymbol{\xi_i} \quad \text{And} \quad \mathbf{y_i} > +1 \tag{15}$$

$$\mathbf{x_i w + b} < -1 + \boldsymbol{\xi_i} \quad \mathbf{y_i} < -\mathbf{1} \tag{16}$$

The above gives us the training condition:

$$\mathbf{y_i(x_i w + b) - 1 + \boldsymbol{\xi_i} \geq 0} \tag{17}$$

With the new soft margin our new objective for the SVM classifier, will be to find

$$min\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{i}\boldsymbol{\xi_i} \tag{18}$$

[The above has been adapted from **[26] [27]** and **[29]**]
In **[29]**, again it is demonstrated how the optimal hyperplane $\mathbf{w}$ is actually the linear combination of the support vector (points on the hyperplane $\mathbf{w}$).

$$\mathbf{w} = \sum_{i=0}^{l}\alpha_i y_i \boldsymbol{s_i} \tag{19}$$

Where s_i are the support vectors.
And for each vector to be tested the evaluation is done by

$$sgn(w^T\mathbf{x} + b) \quad \textbf{(20)}$$

3.7.2 C-Support Vector Classification

The equation yielded above is nothing but a formulation of **C-Support Vector Classification**, which is that, given training vectors (images) $x_i \epsilon R^n, i = 1, \dots, . l$ and two classes $y_i \epsilon \{+1, -1\}$[28]

$$min \frac{1}{2}\mathbf{W^T W} + C\sum_{i=1}^{i} \boldsymbol{\xi_i} \quad \textbf{(21)}$$

Subject to the condition given in equation **(17).**

The equation **(21)**, is a typical Quadratic Programming problem and is easily solvable by a Quadratic Programming (QP) Solver.

3.7.3 The Kernel Trick

In the hope that the data vectors will be more separable in a higher dimensional space, the input vectors can be mapped to some such space (F) using a function $\phi(x_i)$

$$\phi: x_i \rightarrow \phi(x_i) \epsilon \mathrm{F} \quad \textbf{(22)}$$

By mapping x_i with $\phi(x_i)$, there is no need to evaluate such s function directly, because of what we call a **Kernel Trick**. The Kernel Trick can be applied to functions which involve only the inner products. Wherever there is an inner product the kernel function evaluates its outcome directly:

$$K(x_i, x_j) = \phi(x_i)^T \ \phi(x_J) \quad \textbf{(23)}$$

When Kernel is used then by Representer's Theorem, the decision function becomes:
(As shown in **[28]**).

$$sgn(w^T\phi(x) + b) = sgn(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x_j) + b) \quad \textbf{(24)}$$

3.7.4 Radial Basis Function (RBF) Kernel

The Radial Basis Function given by (25) and (26), are real valued functions which depend upon the Euclidean distance between the two vectors, one of the vectors is the Support Vector and the other is the testing data vector. The shape and the area covered by the support vector boundary is a function of σ-width of the Gaussian Kernel, or subsequently γ - the inverse width parameter.

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \quad \textbf{(25)}$$

Or

$$K(x_i, x_j) = \exp\left(-\gamma\|x_i - x_j\|^2\right) \quad \textbf{(26)}$$

Where $\gamma = 2\sigma^2$

3.7.5 Effect of Gamma

The SVM boundary given by $(x) = \sum_{i=1}^{l} \alpha_i y_i K(x_i, x_j) + b$, can be easily seen as a sum of **Gaussian Bumps** centered around each support vector [30]. As γ is increased the classifier becomes more and more biased towards. When $\gamma = 0$, it is a Linear Kernel.

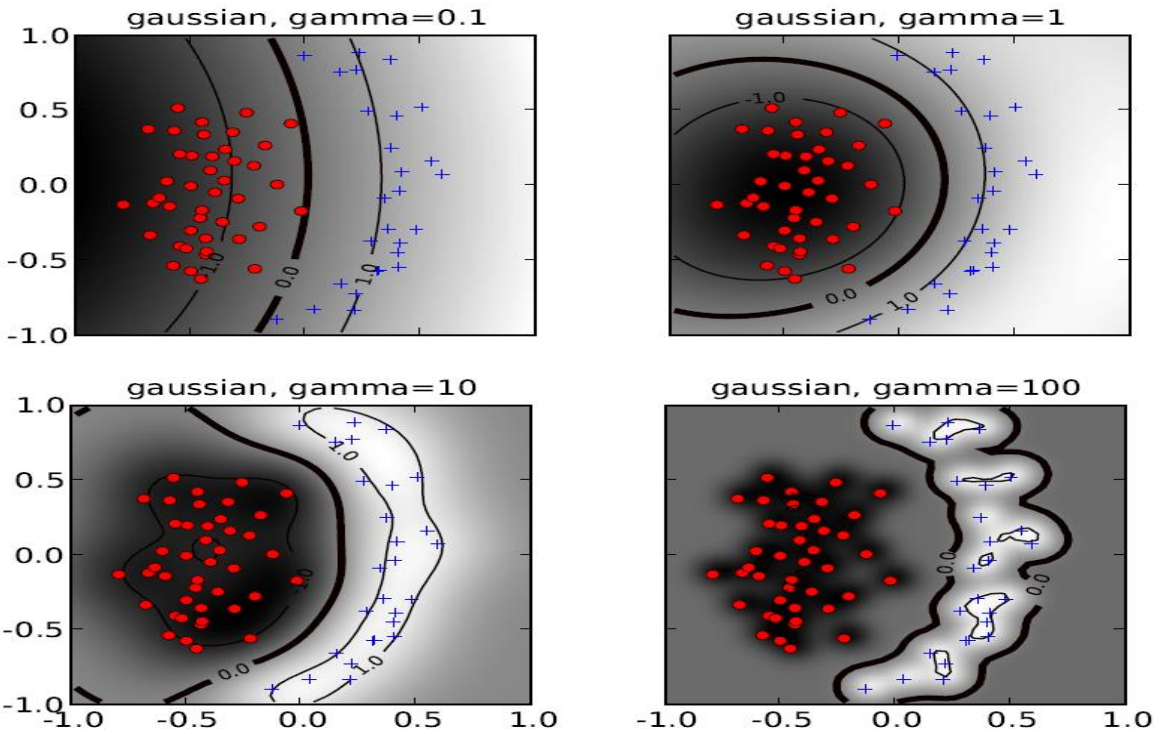


Figure22: Change in SVM Decision Boundary with change in gamma of the RBF Kernel. (Courtesy [30])

Figure shows how the value of γ affects the shape of the decision boundary. Large γ causes a narrower fit, where as lower value of γ ensures a smoother curvature and greater region of influence of the Support Vectors.RBF kernel should not be used when Number of instances is very much less than the number of features

3.7.6 Parameter C

Large C value means higher penalty for outliers, whereas lower C value means higher number of outliers, and hence inaccurate classification for the training data.

3.7.7 Cross-Validation for Parameter Selection

Cross-Validation on the training set is used only to select the optimum parameters for the SVM model. Generally it is a v –fold cross validation. A $v-$ fold cross validation is where, the training set is divided into v equal sub-sets, one set is used for testing on the SVM which is trained using the remaining $v-1$ sets. The cross-validation accuracy is therefore measured by testing each training point once, checking whether it can be correctly classified, while using the $v-1$ subset to train each iteration. This prevents overfitting the SVM model to the training data, and makes it a more robust classifier.

The cross-validation is performed for each possible combination of the values of **C** and $\boldsymbol{\gamma}$. The accuracy of the test is reported as described above.

3.7.8 Grid Search

To select combinations of parameters **C** and $\boldsymbol{\gamma}$, the technique used id the "**Grid Search**". It is an exhaustive search technique where all combinations of the value of **C** and $\boldsymbol{\gamma}$ are tried and the one with the highest accuracy is chosen.

LIBSVM could do this using the python script "grid.py". In "grid.py" as stated in **[14]**, it is a good practice to vary the parameter values exponentially. By default it takes the power or the exponent of "2" as the argument. And the default values observed ranges between **C** = 2^{-5} to 2^{15} and $\boldsymbol{\gamma}$ = 2^{-15} to 2^{3}**[14]**

The script "grid.py" gives the option to search the parameters in steps; we can achieve a finer search by giving smaller steps to vary the parameters. It obviously takes a long time to do cross validation on each combination, but good parameters with higher accuracy can be obtained by using smaller steps. Also increasing the number of v (folds) in the cross validation step.

3.7.9 Linear Kernel

When the number of training classes is very small compared to the feature dimensions, we can avoid using a kernel to map to a higher dimensional space because the data is already put in a separable space.

3.7.10 Multi-class classification

Last but definitely not the least, another issue with SVM is multi-class classification. We use LIBSVM – an opensource implementation for SVM, (both in Java and C++). As mentioned in the Interim Report Literature Review, there are two ways to do this.

1. The One vs. All.
2. The One vs. One.

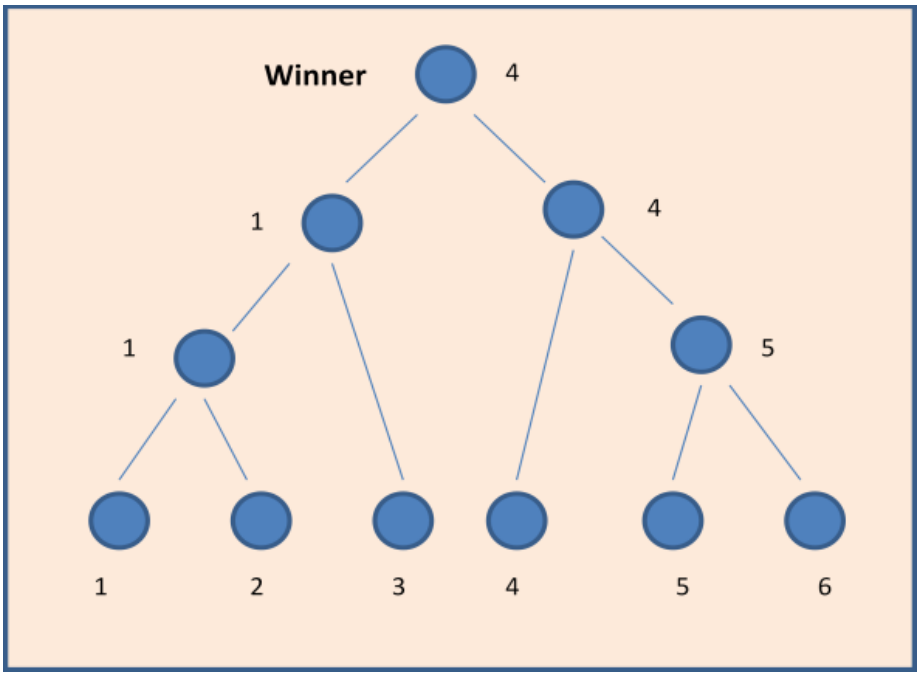


Figure 23: Pairwise SVM classification tree for multiple classes

LIBSVM uses One vs. One, on the grounds that it is faster to train. It is based upon an experimental study done in [31], where they found One vs. One to be more practical.

4. Implementation

There were four main steps to the implementation done:

1. Data Acquisition for preparing training and test sets.
2. Face Detection and Extraction of Faces
3. Feature Selection based on the above described approaches
4. Classifier evaluation on the test set, and parameter selection.

4.1 Data Preparation

For proper face recognition, we require a comprehensive database of faces – spanning across different races and sexes. In our project, we decided to use the face images of celebrities which are commonly available on the internet. This was done in view of deploying the project in a real world environment with real life people's images instead of images taken in a controlled environment.

The program to parse and download the internet images was implemented in Java. A list, of such URL (Uniform Resource Locator) of probable celebrity images, was given in a file as input. From this input file the program first found out only the image URL, which have file extensions ending in ".jpg" or ".png" or ".gif", since we intend to use only pictures that were in JPG, GIF or PNG format. The new list of the filtered image URLs of JPG, GIF and PNG images is generated as a file to be used as an input for the Image Downloader program.

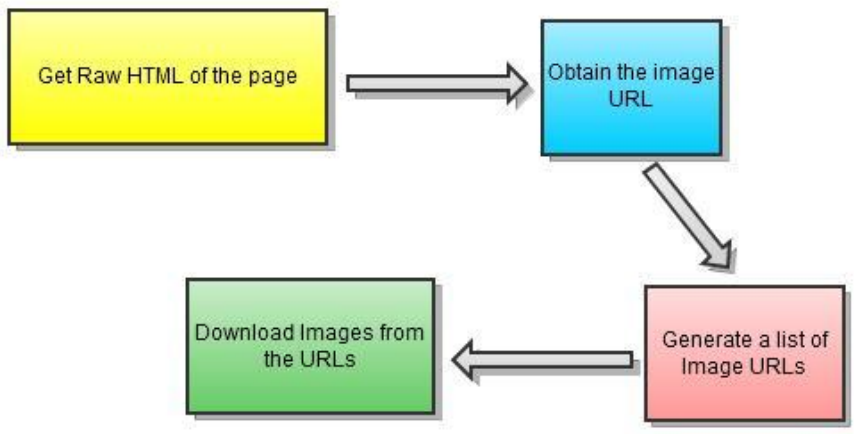


Figure 24: Process Flow of the Image Download Program

The following is a part of the file for the person "David Cameron". In the list of URLs generated each Image URL is extracted on a new line of the file.

Figure 25: Snapshot of the Image URL list file

The images were downloaded into the folder which was named for the specific person. These images contain the person to be identified but the image needs pre-processing before it could be used for training the classifier.

4.2 Face Detection – Extraction of the Faces

A critical part of face recognition is the correct detection of the faces.

OpenCV comes with a pre-trained classifier based on the haar-cascades of frontal human faces. The trained model "haarcascade_frontalface_alt.xml" is loaded into the classifier.

The function which does the face detection is

```
cvHaarDetectObjects(
        IntPtr image,
        IntPtr cascade,
        IntPtr storage,
        double scaleFactor,
        int minNeighbors,
        int flags,
        MCvSize minSize
)
```

Here among above parameters, we do tuning for the parameters scaleFactor, minNeighbors and minSize, in order to achieve better accuracy at detecting faces.

- minSize – the scale at which the image sub-window checks for a face
- scaleFactor - the rate at which the scale is ugraded
- minNeighbors – the neighbourhood of a face region will give many positive responses from the face detector, this parameter sets the minimum number of neighbours to be evaluated to check the region as a correct face region.

The figures below show the plot of all these three parameters against their precision values.

The test was done on the data (images) downloaded for a single person. The plots are at scale factors 10%, 15% and 20%. ; minimum window size was varied between 20x20 to 50x50; each line on the graph represents, the precision/recall at the minimum neighbours of 3,4, or 5.

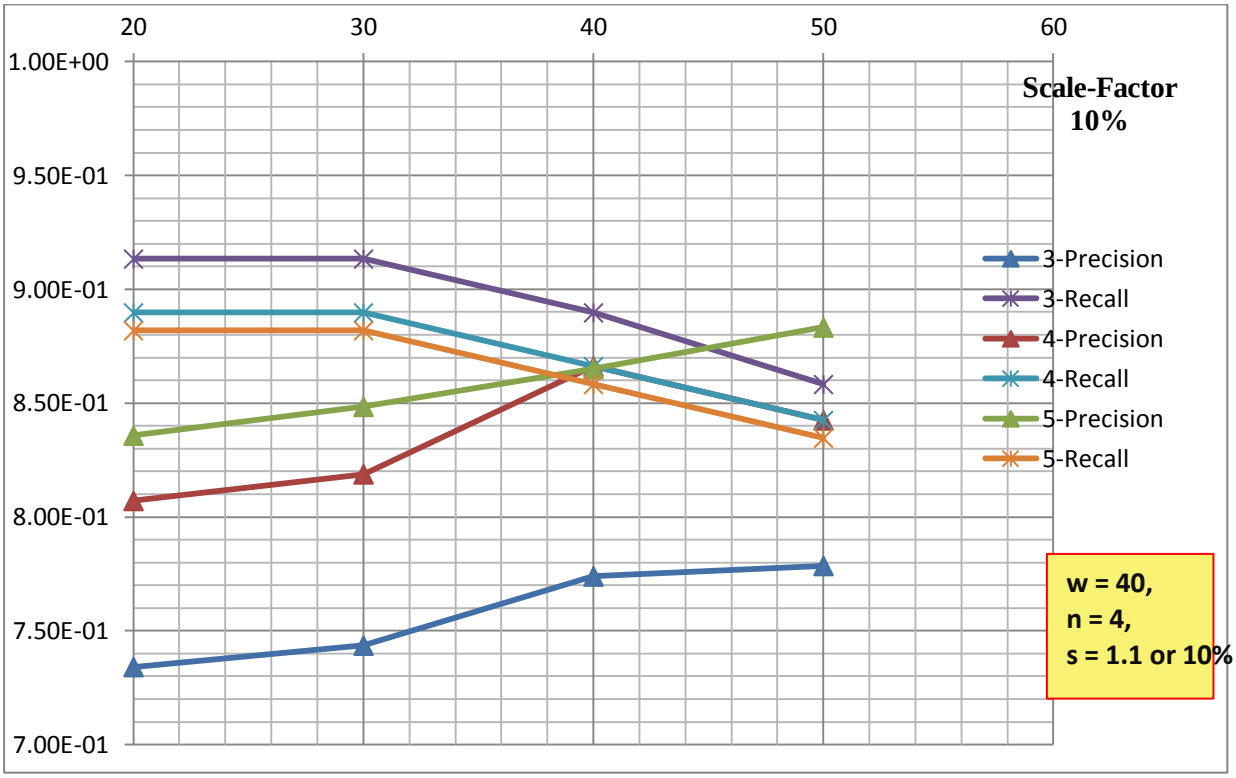


Figure26: Plot of Precision/Recall at Scale Factor 10%

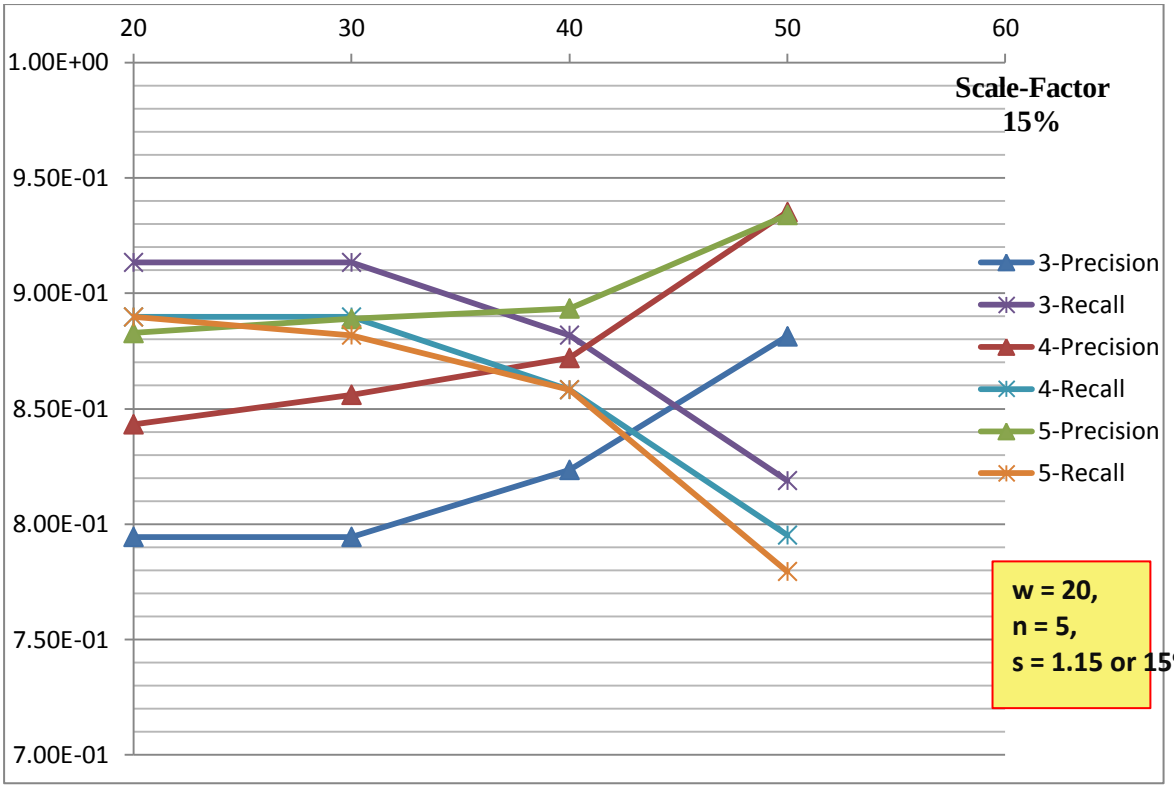


Figure27: Plot of Precision/Recall at Scale Factor 15%

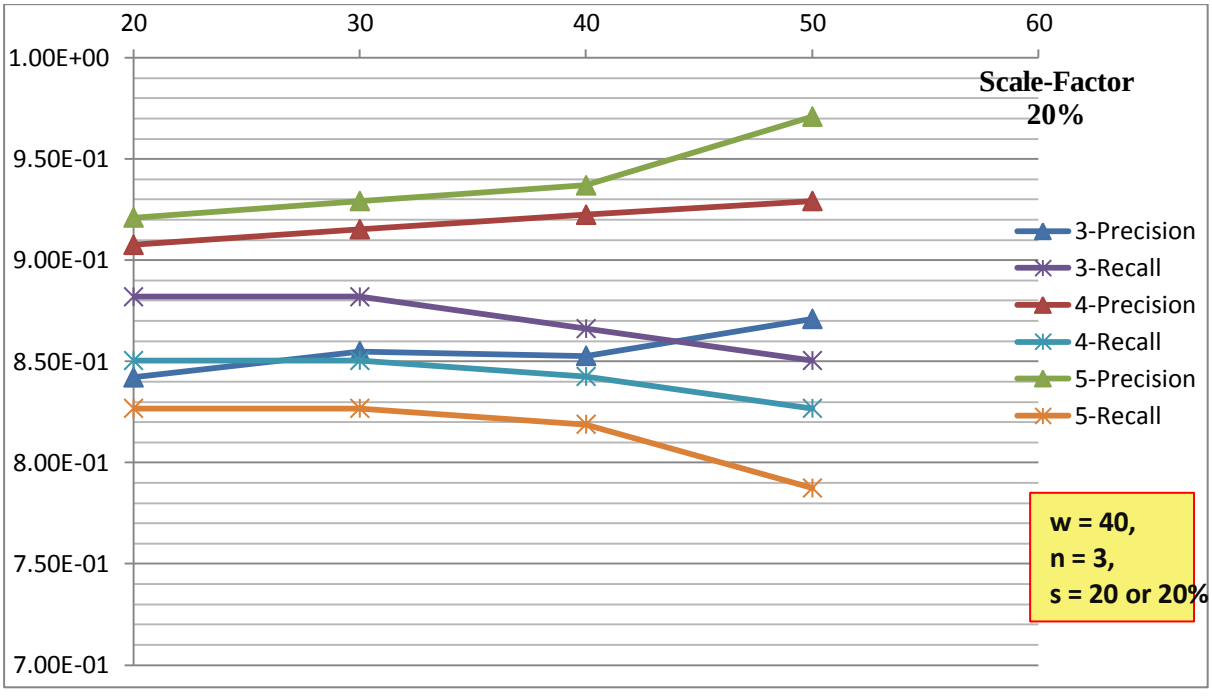


Figure: Plot of Precision/Recall at Scale Factor 20%

Each graph shows the best value of the parameters in the yellow box. We select the parameter values obtained in the first graph (topmost), with values:

- minNeighbors – 4
- scaleFactor - 10%
- minSize – 40.

These parameters were found to more robust.

4.3 Feature Selection

4.3.1 PCA

Training: For using the Eigen Faces in Face Recognition the Steps involved are:

1. Get all the training images – preprocess them
2. Generate the Eigen Vectors also known as Eigen Faces
3. Project the training vectors into the Eigen Space and save them. This forms the training model of our classifier

We use OpenCV, PCA class to construct the **PCA** object, which then generates the required EigenVectors and also the Average Image.

Testing: The testing phase is simply to project the test image onto the EigenVectors, and use this projection in the search of the nearest neighbour among all the projections learned during training. Also SVM can be used to find the decision boundary in the space of all the Projected Training points.

Here is a graph for the selection of the number of PCA Eigen Vectors.

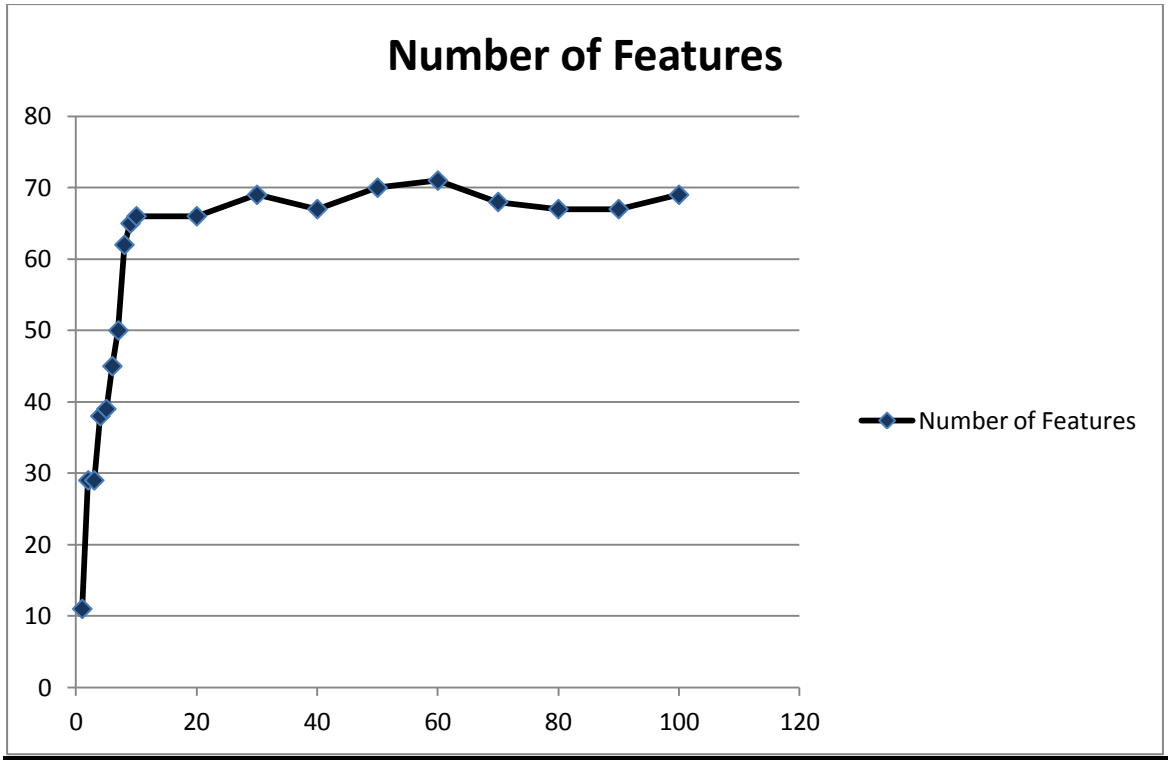


Figure28: PCA Eigen Vectors and their accuracy

These data is obtained by taking three nearest neighbour in KNN (k = 3). We see that at very low number of Eigen Vectors the accuracy is very low. However it stabilises at around 40 – 50. We select 50 as the number of Eigen Vectors to be used because as seen here the performance does not improve much beyond 50, indicating the remaining Eigen Vectors are redundant.

4.3.2 LDA

Linear Discriminant Analysis was implemented in OpenCV and for this a new Object Oriented Class called LDA has been written in C++.

Three overloaded constructors are used to define a new LDA class object.

`LDA(Mat input, Mat class_input, int ntrain);`	- Takes input matrix of image data, class_id of the images and number of training data.
`LDA(Mat input, Mat class_input, int ntrain, char* LDAfilename);`	- Same as above, but also saves the learned classifier in a file.
`LDA(char* loadFromFile);`	- Create a new LDA object from an already trained file.

Internally the LDA class computes the parameters for the generation of the Discriminant Axes by using the following functions within the class.

`static CvMat* calc_mean(Mat input, int rows);`	- Calculates mean vector of the whole dataset - $\bar{x}$
`static CvMat* calc_class_mean(Mat input, int rows, Mat classid_mat, int classid, int *ntrain);`	- Calculates mean vector of each class - μ_c
`CvMat* within_class_scatter(Mat input, int numtrain, Mat class_mat);`	- Calculates the Within Class Scatter of the classes in the dataset - S_w
`CvMat* between_class_scatter(Mat input, int numtrain, Mat class_mat);`	- Calculates the Between Class Scatter among the different classes in the dataset - S_B

Eigen Vectors for the Discriminant Axes are obtained using a Singular Value Decomposition.

Also the functionality to project the data onto the LDA axes is provided by the function – **"`Mat project_LDA(Mat img);`"**

The Training:

During the training phase, all of the data in the training set, associated with their class information is processed, to give the LDA Axes, by the method described above. The learned

axes and the projections of the training images obtained during this phase are saved. They are later to be used as the **model** for the classification of testing data.

The Testing:

The said model which has been trained in the previous phase is loaded and the LDA axis is generated. Each of the training images to be classified is projected on to the LDA subspace. The projections are then compared with KNNsearch or the single nearest neighbour class is returned as the out-come. SVM were also tested on the LDA subspace data.

In LDA, we select N-1 vectors as the LDA axis for classification, where N is the number of the classes (persons) in the training set.

4.3.3 LBP

Test and future directions:

The LBP implemented, as stated earlier was only for comparison purposes only, and did not yield good results from the start.

We tried LBP at different resolutions of the input image, but did not improve.

However Local Descriptors have proven to be very effective features in different researches in the related fields of Face Expression Verification and Object Detections. But in all cases the authors have used various extensions to the simple LBP approach and have improved upon the results quite considerably. Take a look at papers **[34]** and **[35]** for extended analysis and techniques.

4.3.4 KNN

KNN in all the cases, when used with PCA, LDA, or both together, has been done using the OpenCV "**CvKNearest**" class and its function "`CvKNearest::find_nearest`".

Also while finding the 1 nearest neighbour, a manually written function "findNN()" using the Euclidean Distance is used.

Euclidean Distance:

The Euclidean Distance is simply the sum of the squared difference between the image feature values, as given by $D(X,Y)$, and x_i and y_i are the i^{th} dimensional feature value.

$$D(X,Y) = \sum_{i=1}^{n} (x_i - y_i)^2 \quad \textbf{(27)}$$

The following graph is obtained by evaluating the accuracy of PCA KNN at 50 Eigen Vectors

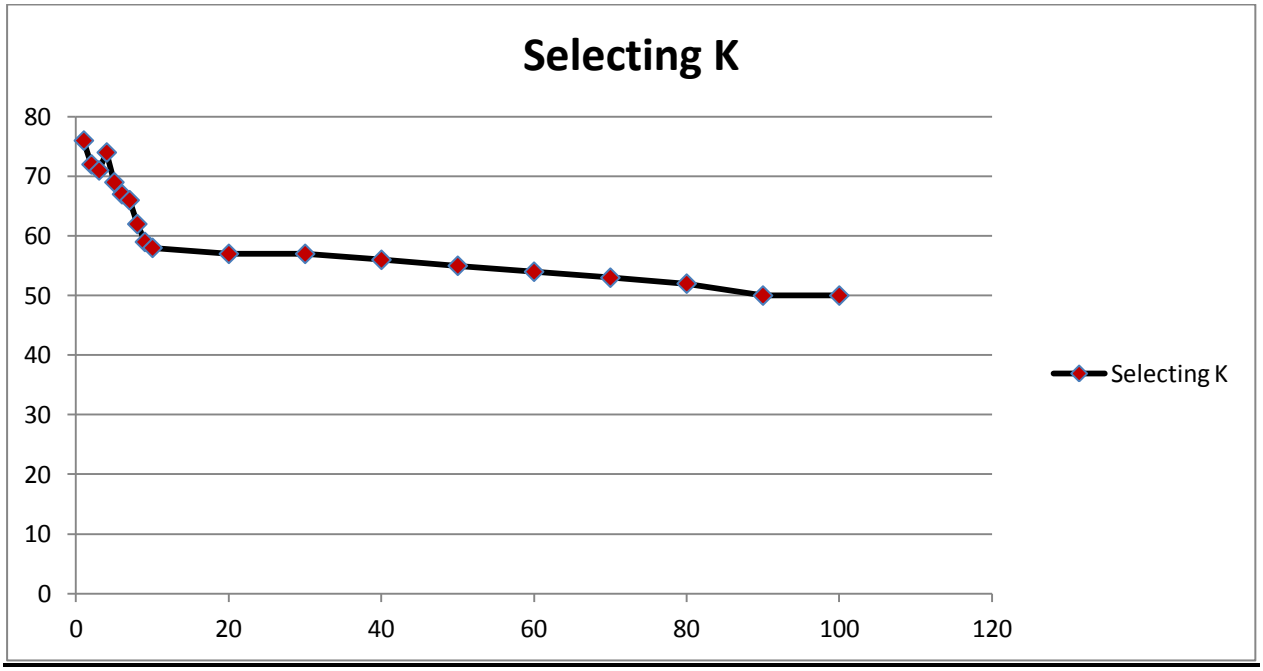


Figure29: Selection of the Parameter K

We see that at low K value, the accuracy is quite high, but this may not be the general case when considering the set of all possible test vectors. As a matter of fact, 1NN performs the best here.

To make it more robust we need to take a value which can have a representation of the surrounding neighbourhood of the test data, not just one point which can be random as well as susceptible to noise in the training set.

So we chose a number of k = 3, as the parameter for KNN.

4.3.5 SVM

C-Support Vector Classification was the type of SVM we used for our training set. It was done using the opensource implementation of the SVM classifier –LIBSVM [28].

In LIBSVM, we used the RBF kernel and did 10 fold cross validation, using the "grid.py" python script for doing grid search of the optimum parameters. "grid.py" by default does a 5 fold cross validation. The 10 fold cross validation takes time but the results are more accurate.

We used the grid search in the feature space generated by the PCA, LDA and the PCA+LDA hybrid. The following images are the contours yielded by the grid search, for the first 50 Principal Components, first 50 LDA axes, and the first 10 LDA+PCA features.

The cross-validation accuracy is reported below

	Classifier	Accuracy
1	PCA	77.5%
2	LDA	100%
3	LDA+PCA	100%

Figure30: Cross Validation Accuracy

It is important to note that these are the cross-validation on the training set and cannot be generalized to the test evaluation.

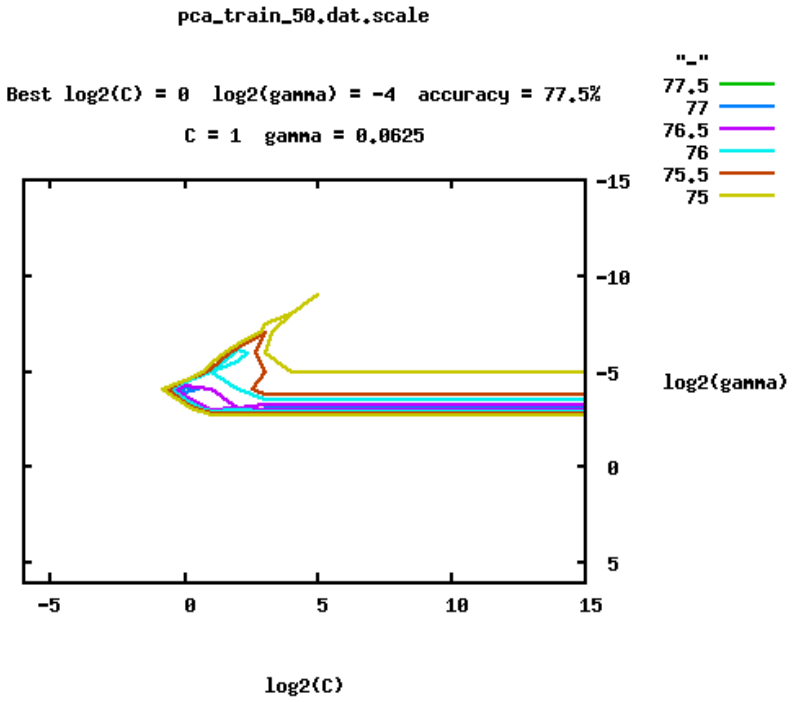


Figure31: Contour for the first 50 PCA components in parameter search

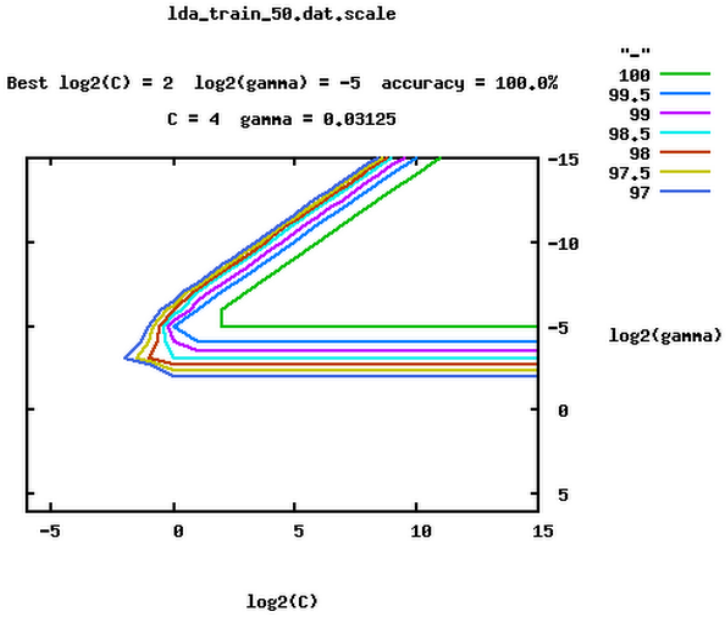


Figure31: Contour for the first 50 LDA components in parameter search

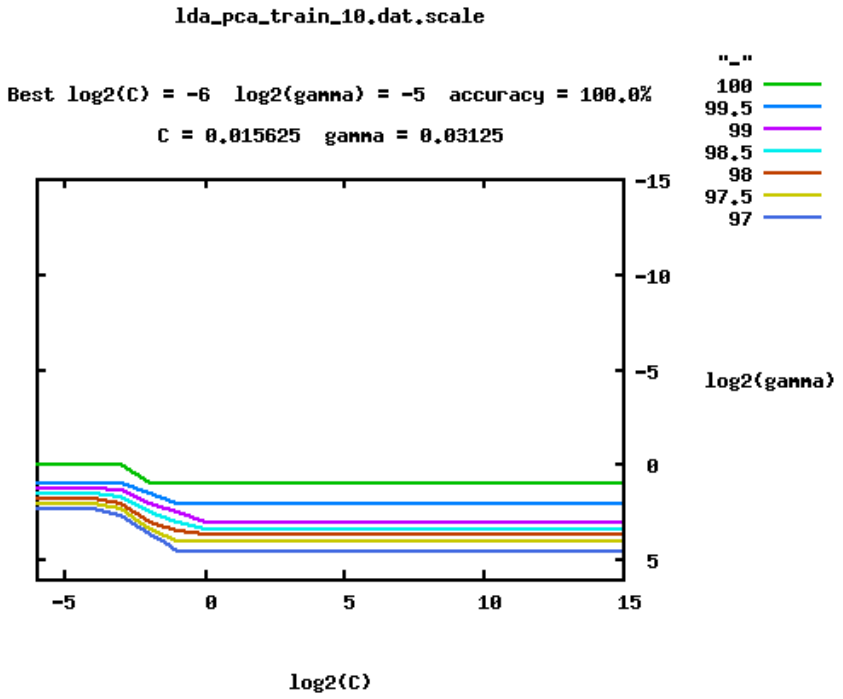


Figure32: Contour for the first 50 PCA+LDA components in parameter search

5. Improvements

Upon the basic classification and feature selection techniques the following improvements were made and we came up with two more approaches. The first is the PCA+LDA approach, where we take LDA of the PCA projections and use them as features for the classification and the second one is a novel Pair-wise PCA approach, where we build a new classifier by voting the classification done in the PCA space of the PCA generated by taking each pair of classes(persons here).

5.1 LDA on PCA – A Hybrid Approach

While exploring further improvements in the subspace approaches of face recognition, we take inspiration from **[27]**, which was particularly aimed at getting a better result than the normal LDA approach using direct gray-scale images, and also to improve generalizations.

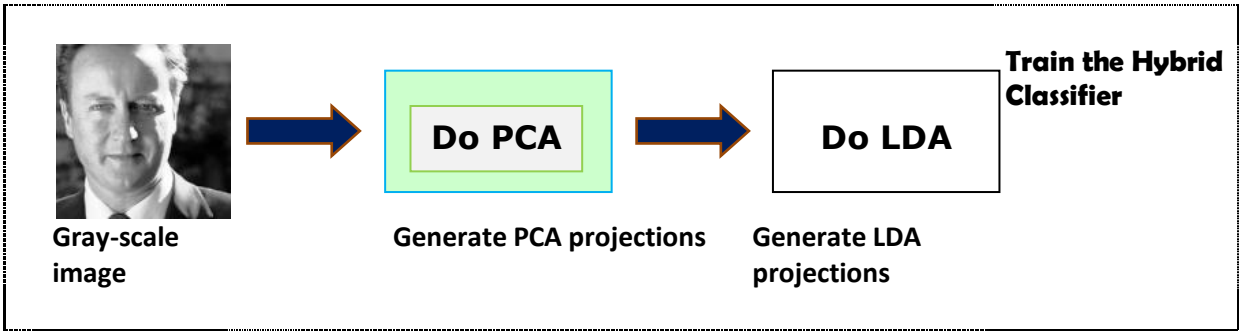


Figure33: LDA+PCA Approach

Training:

The steps involved is not that complicated; the training phase is shown above learn the PCA projections first, from the training images. Then use these PCA projections to train the LDA. The LDA projections obtained from these operations are stored as the learned model.

Testing:

Any test image is first projected on-to the PCA subspace, using the PCA EigenVectors, obtained during training. This PCA projection is then passed into the LDA, where the new LDA projection is compared against all the projections learned in training phase. We can use KNN or SVM to do that.

5.2 Pair-wise PCA – A novel approach

The general PCA Eigen Space, using the entire set of classes, yielded information which did not have that high a discriminating factor as among the different classes. And considering LDA alone, it suffered from overfitting the training data. We chose to devise a novel way of improving the class separating without overfitting the data. It is an experimental method, which will have many challenges if it needs to be scaled to a deployable algorithm.

In the Pair-wise PCA, we take each person or class of image and compare it against each of the remaining classes, to train and generate a model. The model compares 1 image against only 1 other image. In this smaller PCA space, created between two image classes, there is a better chance of separating one class from the other.

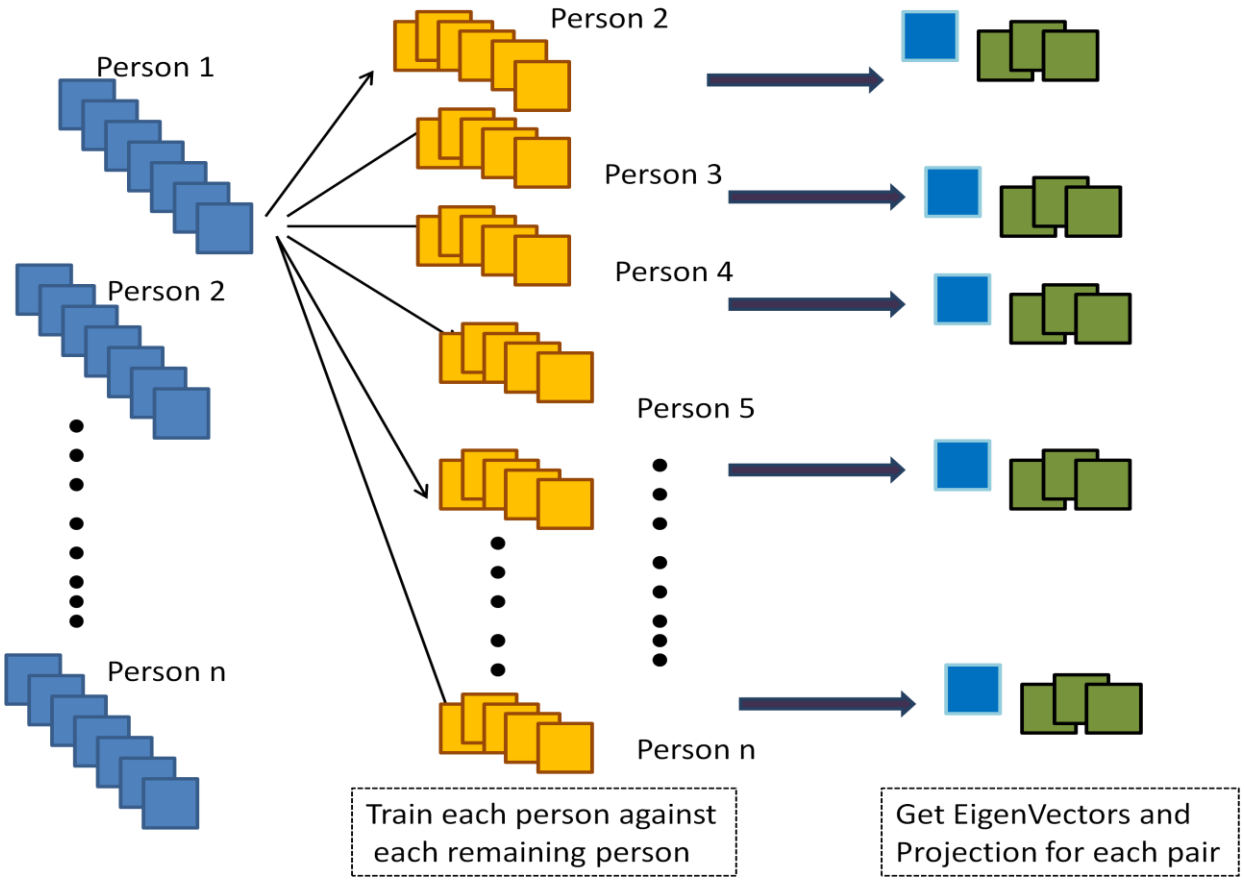


Figure34: The Pair-wise PCA approach

Since each person is trained against every other person, there are $\frac{n(n-1)}{2}$ combinations or pairs of PCA training that is required, where n is the number of classes, or persons. This can be a challenge because the time complexity to train is not linear, but $O(n^2)$. The same goes for testing each test image has to be evaluated against all $\frac{n(n-1)}{2}$ trained PCA models. The final result is of course decided by voting among the PCA results of all pairs.

There can be algorithmic and heuristics based approaches to reduce the operational complexity of this approach, but in the short time for the thesis, work we scaled down the time of training by considerable amount just by reducing the training vector size from 128x128 pixels to 50x50, and also using fewer training samples for each person.

6. Results

The training set composed of 10 persons, and 20 images of each person. All images were preprocessed using Histogram Equalization and saved as training set. The same was done for the test set, 10 persons with 10 images for each.

The following result was obtained.

Classifier	Accuracy (%)
PCA KNN	71
LDA KNN	55
LDA+PCA	92
LDA SVM	48
PCA SVM	86
LDA + PCA SVM	85
Pairwise PCA	73

The above results were obtained after extensive experimentation of parameter optimization of each feature and classifier. Initially the test was done using 128x128 images. The results were very poor. However reducing the scale to 50x50 helped improve the results as well as getting a faster time of operation.

We can see that LDA+PCA outperformed all other methods evaluated so far, with an accuracy of 92%. While Pairwise method was a little above average.

LDA on the other hand gives 100% cross-validation results but fails to generalize on the test data. It is highly prone to overfitting.

PCA with KNN does an average job with 71%. SVM with RBF kernel, brings out better classification in the same PCA space with an accuracy of 86%.

7. Conclusion

We implemented various algorithms for Feature extraction and Classification for the Face Recognition problem. Also detail steps of optimizing the different parameters are evaluated and the outputs are shown. We can conclude that LDA+PCA has been the best in the subspace methods, that we have seen. We have chosen this method in the final implementation of the GUI, which will get the data from the news URL, Extract Faces (through Haar Detector) and identify the person from that image. Although LDA sounded promising initially, it had not performed at par, due to overfitting. The new LDA+PCA, still lets use the LDA but in a different vector space, altogether. And while KNN gets results better than SVM in the experiments, SVM with RBF kernel will be highly robust and more useful when the training classes become large.

8. Future Direction

In the genre of Face Recognition, one of the definite ways forward will be the approach discussed in [33]. The method in [33] is a component based one, compared against general global approaches, which use the image as a single vector. In a typical scenario of the component based approaches, the first thing to do is the detection of the components. Highly accurate results can be obtained by combining classifiers which have been trained on specific components. In this method the proper extraction of the facial components is critical to the performance of the overall classifier.

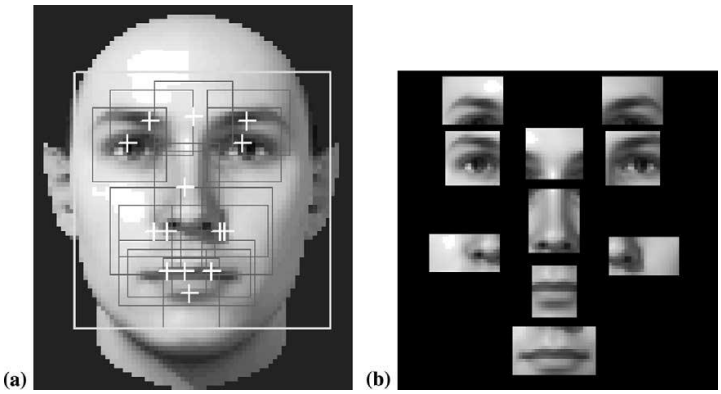


Figure: Component Based Face Recognition[33]

A method to improve the general face recognition without using components the PCA and LDA could be the use sparse representation as shown in [32]. But care should be taken while exploring new directions so that the method is scalable and fast, while giving a reliable accuracy.

9. Bibliography

1. Viola, P. and M. Jones (2001). Rapid object detection using a cascade of simple features. CVPR.

2. Viola, P. and M. Jones (2001). Robust Real-time Object Detection. IJCV.

3. Bradski, G. (2000). "The OpenCV Library." Dr. Dobb's Journal of Software Tools.

4. " Haar Feature-based Cascade Classifier for Object Detection." http://opencv.willowgarage.com, n.d. Web. 22 Mar. 2012.

5. Papageorgiou, C. and T. Poggio (2000). A trainable system for object detection. IJCV.

6. Adapted from the Wikimedia Commons file "Image: Prm VJ fig1 featureTypesWithAlpha.png" URL - http://en.wikipedia.org/wiki/File:Prm_VJ_fig1_featureTypesWithAlpha.png

7. Adapted from the OpenCV WIKI file "Image: haarfeatures.png" URL - http://opencv.willowgarage.com/documentation/c/objdetect_cascade_classification.html

8. Lienhart, R. and J. Maydt (2002). "An extended set of Haar-like features for rapid object detection." CVPR.

9. Freund, Y. and R. E. Schapire (1995). "A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting."

10. Starovoitov, V. V., D. I. Samal, et al. (2003). Image Enhancement for Face Recognition. International Conference on Iconics.

11. Adapted from the Wikimedia Commons file "Image: Histogrammspreizung.png" URL - http://en.wikipedia.org/wiki/File:Histogrammspreizung.png

12. RAMÍREZ-GUTIÉRREZ, K., D. CRUZ-PÉREZ, et al. "Face Recognition and Verification using Histogram Equalization." SELECTED TOPICS in APPLIED COMPUTER SCIENCE.

13. Turk, M. A. and A. P. Pentland (1991). Face recognition using EigenFaces. IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Maui, HI: 586 – 591

14. M. Turk and A. Pentland, EigenFaces for recognition Journal of Cognitive Neuroscience (1991), 71-86.

15. Kirby, M. and L. Sirovich (1990). "Application of the Karhunen-Loeve Procedure for the Characterization of Human Faces." IEEE Trans. Pattern Anal. Mach. Intell. 12(1): 103-108.

16. - Zhao, W., R. Chellappa, et al. (1998). Discriminant analysis of principal components for face recognition. Third IEEE International Conference on Automatic Face and Gesture Recognition.

17. Andrey Finayev , "Eigenface", Catalogue of Artificial Intelligence Techniques, URL - http://aicat.inf.ed.ac.uk/entry.php?id=636, n.d. Web. 27 Mar. 2012

18. PCA vs LDA, http://www.csee.wvu.edu/~timm/cs591o/old/FSS.html. West Virginia University, , Web. 21 Mar. 2012. <http://www.lcsee.cemr.wvu.edu/>.]

19. T. Ahonen, et al. (2004). `Face Recognition with Local Binary Patterns'. pp. 469-481.

20. T. Ojala, M. Pietikäinen, and D. Harwood (1994), "Performance evaluation of texture measures with classification based on Kullback discrimination of distributions", Proceedings of the 12th IAPR International Conference on Pattern Recognition (ICPR 1994), vol. 1, pp. 582 - 585.

21. T. Ojala, M. Pietikäinen, and D. Harwood (1996), "A Comparative Study of Texture Measures with Classification Based on Feature Distributions", Pattern Recognition, vol. 29, pp. 51-59.

22. 10 Shops in a Flat City and Their Voronoi Cells (euclidean Distance). Digital image. Http://en.wikipedia.org/wiki/Voronoi_diagram. Wikipedia, 15 Jan. 2012. Web. 21 Mar. 2012. http://www.wikipedia.org/

23. Christopher D. Manning, Prabhakar Raghavan & Hinrich Schütze (2008). *Vector space classification*. In *Introduction to Information Retrieval* (Ch: 14). Retrieved from http://nlp.stanford.edu/IR-book/pdf/14vcat.pdf

24. "K-Nearest Neighbors" URL - http://opencv.itseez.com/modules/ml/doc/k_nearest_neighbors.html, Web. 29 Mar. 2012

25. Beyer, K., J. Goldstein, et al. (1999). When Is "Nearest Neighbor" Meaningful?

26. Fletcher, T. (2008). Support Vector Machines Explained. London, University College London. PhD: 19. http://www.tristanfletcher.co.uk/SVM%20Explained.pdf

27. Boser, B. E., I. M. Guyon, et al. (1992). "A Training Algorithm for Optimal Margin Classifiers." Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory: 144--152.

28. Chang, C.-C. and C.-J. Lin (2011). "LIBSVM: A library for support vector machines." ACM Transactions on Intelligent Systems and Technology 2(3): 27:21--27:27.

29. Cortes, C. and V. Vapnik (1995). "Support-Vector Networks." Machine Learning: 273--297.

30. A. Ben-Hur & J. Weston (2010). `A User's Guide to Support Vector Machines Data Mining Techniques for the Life Sciences'. vol. 609 of Methods in Molecular Biology, chap. 13, pp. 223-239. Humana Press, Totowa, NJ

31. C.-W. Hsu and C.-J. Lin. A comparison of methods for multi-class support vector machines , IEEE Transactions on Neural Networks, 13(2002), 415-425.

32. - Li, H., P. Wang, et al. (2010). Robust Face Recognition via Accurate Face Alignment and Sparse Representation. Proceedings of the 2010 International Conference on Digital Image Computing: Techniques and Applications.

33. B. Heisele, et al. (2003). `Face recognition: component-based versus global approaches'. Computer Vision and Image Understanding 91(1-2):6-21.

34. Anish Mittal , P. B. (2008). "ROBUST FACIAL EXPRESSION RECOGNITION USING LBP VARIANTS BASED SVM ENSEMBLE." 254-258.

35. Smith, R. S. and T. Windeatt (2010). "Facial Expression Detection using Filtered Local Binary Pattern Features with ECOC Classifiers and Platt Scaling." Journal of Machine Learning Research 11: 111-118.

10. Codes

"cv_lda.h"

```
//File:           cv_lda.h
//Author:         Bina Ningthoujam/bn0887
//Course:         COMSM3100
//Date:           17/03/2012
//Assignment:     MSc Advanced Project
//File Purpose:   Declaration of LDA class.
//Program Purpose:  Create a declaration for the LDA class.

#include <iostream>
#include "cv.h"
#include "cvaux.h"
#include "highgui.h"

using namespace cv;
using namespace std;

#ifndef CV_LDA_H
#define CV_LDA_H

class LDA
{
	public:
		//CvMat* LDA_mat;//lda mat
		//CvMat cLDA_mat;//need a global variable for loading from file
		Mat LDAmat;
		Mat inputMat;//row mat input
		Mat classMat;//classid of the training samples
		int numTrain;//number of the training samples
		int numClasses;//number of classes
		int numFisher;//rows of the LDA mat

		LDA(Mat input, Mat class_input, int ntrain);
		LDA(Mat input, Mat class_input, int ntrain, char* LDAfilename);
		LDA(char* loadFromFile);

		Mat project_LDA(Mat img);
		int save_LDA(char*filename);
		Mat loadLDA(char* filename);
		Mat getLDA();
	private:
		static vector<int> find_unique_classes(Mat class_mat);
		static CvMat* calc_mean(Mat input, int rows);
		static CvMat* calc_class_mean(Mat input, int rows, Mat classid_mat, int
classid, int *ntrain);
		CvMat* within_class_scatter(Mat input, int numtrain, Mat class_mat);
		CvMat* between_class_scatter(Mat input, int numtrain, Mat class_mat);
};

#endif
```

"cv_lda.cpp"

```
//File:           cv_lda.cpp
//Author:         Bina Ningthoujam/bn0887
//Course:         COMSM3100
//Date:           17/03/2012
//Assignment:     MSc Advanced Project
//File Purpose:   Definition of LDA class.


#include "stdafx.h"
#include "cv_lda.h"

using namespace cv;
using namespace std;

  //Constructors
  LDA::LDA(Mat input, Mat class_input, int ntrain)
{
	inputMat = input;//
	classMat = class_input;//
	numTrain = ntrain;//

	vector<int> Classes = find_unique_classes(classMat);
	numClasses = Classes.size();//

	//Calculate the LDA
	Mat temp = inputMat.reshape(1,numTrain);
	int num_components = temp.cols;

	CvMat* Sw = within_class_scatter(inputMat, numTrain, classMat);

	CvMat* Sb = between_class_scatter(inputMat, numTrain, classMat);

	CvMat* invSw = cvCreateMat(num_components,num_components,CV_32F);
	for(int r=0;r<num_components;r++)
		for(int c=0;c<num_components;c++)
			cvmSet(invSw,r,c,0.0);//initialize

	cvInvert(Sw,invSw);
	cvReleaseMat(&Sw);
	CvMat *Sratio = cvCreateMat(num_components,num_components,CV_32F);
	cvMatMul(invSw,Sb,Sratio);//generalized mat mul
	cvReleaseMat(&Sb);
	cvReleaseMat(&invSw);

	CvMat* EigenVectors = cvCreateMat(num_components, num_components, CV_32F);
	CvMat* EigenValues = cvCreateMat(num_components, 1, CV_32F );
	cvSVD(Sratio, EigenValues, EigenVectors, 0, CV_SVD_U_T + CV_SVD_MODIFY_A);//see
opecv svd
	cvReleaseMat(&Sratio);
	cvSave("C:\\WORKSPACE\\LDA\\LDA_Axis\\EigenVector.xml", EigenVectors );

	numFisher = EigenVectors->rows;
	Mat lda(EigenVectors);
	lda.copyTo(LDAmat);
}

LDA::LDA(Mat input, Mat class_input, int ntrain, char* LDAfilename)
{
	inputMat = input;
	classMat = class_input;
	numTrain = ntrain;
```

```
	vector<int> Classes = find_unique_classes(classMat);
	numClasses = Classes.size();
	//Calculate the LDA
	Mat temp = inputMat.reshape(1,numTrain);
	int num_components = temp.cols;

	CvMat* Sw = within_class_scatter(inputMat, numTrain, classMat);

	CvMat* Sb = between_class_scatter(inputMat, numTrain, classMat);

	CvMat* invSw = cvCreateMat(num_components,num_components,CV_32F);
	for(int r=0;r<num_components;r++)
		for(int c=0;c<num_components;c++)
			cvmSet(invSw,r,c,0.0);//initialize

	cvInvert(Sw,invSw);
	cvReleaseMat(&Sw);
	CvMat *Sratio = cvCreateMat(num_components,num_components,CV_32F);
	cvMatMul(invSw,Sb,Sratio);//generalized mat mul
	cvReleaseMat(&Sb);
	cvReleaseMat(&invSw);

	CvMat* EigenVectors = cvCreateMat(num_components, num_components, CV_32F);
	CvMat* EigenValues = cvCreateMat(num_components, 1, CV_32F );
	cvSVD(Sratio, EigenValues, EigenVectors, 0, CV_SVD_U_T + CV_SVD_MODIFY_A);//see
opecv svd
	cvReleaseMat(&Sratio);

	cvSave("C:\\WORKSPACE\\LDA\\LDA_Axis\\EigenVector.xml", EigenVectors );//saving
	numFisher = EigenVectors->rows;

	//Save the LDA
	Mat lda(EigenVectors);
	lda.copyTo(LDAmat);
	FileStorage fs(LDAfilename, FileStorage::WRITE);
			if (!fs.isOpened()){
				fs.open(LDAfilename, FileStorage::WRITE);
			}
			fs << "numClasses" << numClasses;
			fs << "numFisher" << numFisher;
			fs << "numTrain" << numTrain;
			fs << "LDA" << Mat(LDAmat);
			fs.release();

}

LDA::LDA(char* loadFromFile)
{
	Mat temp = loadLDA(loadFromFile);
}

//private//
vector<int> LDA::find_unique_classes(Mat class_mat)
{
	//class_mat - the classes of each sample in the input
	bool checked;
	int isize = class_mat.rows;
	vector<int> the_classes;
	int size = -1;

	for(int i=0;i< isize;i++)
```

```
	{
		checked=false;
		for(int j=0;j<the_classes.size();j++)
		{
			if(the_classes[j]==class_mat.at<int>(i,0))
			{
				checked=true;
			}
		}
		if(!checked)
		the_classes.push_back(class_mat.at<int>(i,0));
	}

	  size = the_classes.size();

	  return the_classes;
}

CvMat* LDA::calc_mean(Mat input, int rows)
{
	  //input - the input mat
	  //rows - number of rows in the input mat - which should be
	  Mat temp;
	  input.reshape(1, rows).convertTo(temp,CV_32FC1,1.0,0.0);
	  int n_components = temp.cols;
	  CvMat* mean = cvCreateMat(1,n_components,CV_32F);
	  for(int i = 0; i < rows; i++)
	  {
		CvMat temp1r = temp.row(i);
		cvAdd(mean, &temp1r, mean);
	  }
    cvConvertScale(mean,mean,1.0/rows,0);

	  return mean;
}

CvMat* LDA::calc_class_mean(Mat input, int rows, Mat classid_mat, int classid, int
*ntrain)
{
	  //input - the input mat
	  //rows - number of rows in the input mat - which should be
	  //classid_mat - the mat containing the class_id of the respective row in the
input mat
	  //classid - id for which mean is to be calculated
	  //*ntrain - passed in to return the number of training samples in the required
class
	  //int nclasses =  find_number_of_unique_classes(classid_list);
	  Mat temp;
	  input.reshape(1, rows).convertTo(temp,CV_32FC1,1.0,0.0);
	  int n_components = temp.cols;
	  int n_rows = 0;
	  CvMat* c_mean = cvCreateMat(1,n_components,CV_32F);
	  for(int i = 0; i < rows; i++)
	  {
		if(classid==classid_mat.at<int>(i,0))
		{
			CvMat temp1r = temp.row(i);
			cvAdd(c_mean, &temp1r, c_mean);
			n_rows++;
		}
	  }
    cvConvertScale(c_mean,c_mean,1.0/n_rows,0);
```

```
	(*ntrain) = n_rows;//the number of training sample in the class

	return c_mean;//return the mean of the class
}

CvMat* LDA::within_class_scatter(Mat input, int numtrain, Mat class_mat)
{
	//input - the input mat
	//numtrain - number of rows or training samples in the input mat
	//class_mat - the linked class id for each input
	typedef struct class_mean_data
	{
		int classid;
		Mat c_mean;
		int n_train;
	}class_mean;
	vector<class_mean> Class_mean_array;

	vector<int> Classes = find_unique_classes(class_mat);

	int n_classes = Classes.size();
	Mat temp = input.reshape(1,numtrain);
	int num_components = temp.cols;

	CvMat* mean_a = calc_mean(input,numtrain);
	//per class
	for(int i = 0; i < n_classes; i++)
	{
		int classid = Classes[i];
		int ntrain = 0;
		CvMat* c_mean = calc_class_mean(input,numtrain,class_mat,classid,
&ntrain);
		class_mean temp_data;
		temp_data.classid = classid;
		temp_data.n_train = ntrain;
		Mat t_c_mean(c_mean);
		t_c_mean.copyTo(temp_data.c_mean);
		Class_mean_array.push_back(temp_data);
	}

	//Sw part
	CvMat* Sw = cvCreateMat(num_components,num_components,CV_32F);
	for(int r=0;r<num_components;r++)
		for(int c=0;c<num_components;c++)
			cvmSet(Sw,r,c,0.0);
	for(int i = 0; i < Class_mean_array.size(); i++)
	{
		int classid = Class_mean_array[i].classid;
		CvMat* cSw = cvCreateMat(num_components,num_components,CV_32F);
		for(int r=0;r<num_components;r++)
			for(int c=0;c<num_components;c++)
				cvmSet(cSw,r,c,0.0);
		int num_c = 0;

		for(int j = 0; j < input.rows; j++)
		{
			if(classid==(class_mat.at<int>(j,0)))
			{
				CvMat* mdiff = cvCreateMat(1,num_components,CV_32F);
				CvMat* mdiff_squared =
cvCreateMat(num_components,num_components,CV_32F);
```

```
                        CvMat input_r_d = input.row(j);
                        CvMat* input_r = &input_r_d;
                        CvMat mean_c = Class_mean_array[i].c_mean;
                        cvSub(input_r,&mean_c,mdiff);
                        cvMulTransposed(mdiff,mdiff_squared,1);
                        cvAdd(cSw,mdiff_squared,cSw);
                        cvReleaseMat(&mdiff);
                        cvReleaseMat(&mdiff_squared);
                        num_c++;
                    }
                }
                double scale = (double)num_c/(double)numtrain;
                cvConvertScale(cSw,cSw,scale);//weighted
                cvAdd(Sw,cSw,Sw);
                cvReleaseMat(&cSw);
        }
        return Sw;
}

CvMat* LDA::between_class_scatter(Mat input, int numtrain, Mat class_mat)
{
        typedef struct class_mean_data
        {
                int classid;
                Mat c_mean;
                int n_train;
        }class_mean;
        vector<class_mean> Class_mean_array;

        vector<int> Classes = find_unique_classes(class_mat);

        int n_classes = Classes.size();
        Mat temp = input.reshape(1,numtrain);
        int num_components = temp.cols;

        CvMat* mean_a = calc_mean(input,numtrain);

        //per class
        for(int i = 0; i < n_classes; i++)
        {
                int classid = Classes[i];
                int ntrain = 0;
                CvMat* c_mean = calc_class_mean(input,numtrain,class_mat,classid,
&ntrain);
                class_mean temp_data;
                temp_data.classid = classid;
                temp_data.n_train = ntrain;
                Mat t_c_mean(c_mean);
                t_c_mean.copyTo(temp_data.c_mean);
                Class_mean_array.push_back(temp_data);
        }
        CvMat* Sb = cvCreateMat(num_components,num_components,CV_32F);
        for(int r=0; r<num_components; r++)
                for(int c=0; c<num_components; c++)
                        cvmSet(Sb,r,c,0.0);
        for(int i = 0; i < Class_mean_array.size(); i++)
        {
                int classid = Class_mean_array[i].classid;

                CvMat* mdiff = cvCreateMat(1,num_components,CV_32F);
                CvMat* mdiff_squared =
cvCreateMat(num_components,num_components,CV_32F);
```

```
			CvMat mean_c = Class_mean_array[i].c_mean;
			int num_c = Class_mean_array[i].n_train;
			cvSetZero(mdiff);
			cvSetZero(mdiff_squared);

			cvSub(&mean_c,mean_a,mdiff);
			cvMulTransposed(mdiff,mdiff_squared,1);
			double m_scale = (double)num_c/(double)numtrain;
			cvConvertScale(mdiff_squared,mdiff_squared,m_scale);
			cvAdd(Sb,mdiff_squared,Sb);
			cvReleaseMat(&mdiff);
			cvReleaseMat(&mdiff_squared);
	}
	return Sb;
}

//public//
Mat LDA::project_LDA(Mat img)
{
	//lda axes are the rows
	int n_components = LDAmat.cols;
	CvMat lda = LDAmat;
	CvMat imgmat = img;
		CvMat *imagT = cvCreateMat(n_components,1,CV_32F);//transpose of input
		CvMat *LDAProjectionT = cvCreateMat(numFisher,1,CV_32F);
		CvMat *LDAProjection = cvCreateMat(1,numFisher,CV_32F);
		cvTranspose(&imgmat,imagT);
		cvMatMul(&lda,imagT,LDAProjectionT);
		cvTranspose(LDAProjectionT,LDAProjection);
	Mat plda(LDAProjection, false);
	return plda;
}

int LDA::save_LDA(char* filename)
{
	FileStorage fs(filename, FileStorage::WRITE);
			if (!fs.isOpened()){
				fs.open(filename, FileStorage::WRITE);
			}
			fs << "numClasses" << numClasses;
			fs << "numFisher" << numFisher;
			fs << "numTrain" << numTrain;
			fs << "LDA" << Mat(LDAmat);
			fs.release();
	return 0;
}

Mat LDA::loadLDA(char* loadFromFile)
{
	FileStorage fs(loadFromFile, FileStorage::READ);
				if (!fs.isOpened()){
					fs.open(loadFromFile, FileStorage::READ);
				}
					fs ["LDA"] >> LDAmat;
					fs ["numClasses"] >> numClasses;
					fs ["numFisher"] >> numFisher;
					fs ["numTrain"] >> numTrain;
	return LDAmat;
}
```

```
Mat LDA::getLDA()
{
	return LDAmat;
}
```

Function Generating LBP – LBP(cv::Mat input)

```
Mat LBP(Mat imat)//calculate and return the LBP
{
	int pixel, n1,n2,n3,n4,n5,n6,n7,n8,value;
	int v1,v2,v3,v4,v5,v6,v7,v8;
	Mat lbp_mat = Mat::zeros(imat.rows-2,imat.cols-2,CV_8UC1);//create the LBP
		for(int i=1;i<(imat.rows-1);i++){
			for(int j=1;j<(imat.cols-1);j++){
	//GET the pixels in the 8 neighbourhood.
	//starting from the top left corner and going clock-wise
				pixel = imat.at<uchar>(i,j);
				n1 = imat.at<uchar>(i-1,j-1);
				n2 = imat.at<uchar>(i-1,j);
				n3 = imat.at<uchar>(i-1,j+1);
		n4 = imat.at<uchar>(i,j+1);
				n5 = imat.at<uchar>(i+1,j+1);
				n6 = imat.at<uchar>(i+1,j);
				n7 = imat.at<uchar>(i+1,j-1);
		n8 = imat.at<uchar>(i,j-1);

		(n1>=pixel)?(v1=128):(v1=0);
		(n2>=pixel)?(v2=64):(v2=0);
		(n3>=pixel)?(v3=32):(v3=0);
		(n4>=pixel)?(v4=16):(v4=0);
		(n5>=pixel)?(v5=8):(v5=0);
		(n6>=pixel)?(v6=4):(v6=0);
		(n7>=pixel)?(v7=2):(v7=0);
		(n8>=pixel)?(v8=1):(v8=0);

		 value = v1 + v2 + v3 + v4 + v5 + v6 + v7 + v8;
				lbp_mat.at<uchar>(i-1,j-1) = value;
	    }
	 }
	 return lbp_mat;
}
```