Tribhuvan University
Institute of Engineering - Pulchowk Campus
Department of Electronics and Computer Engineering

Final Year Project Report

on

Real Time Face Tracking and Recognition (RTFTR)

http://rtftr.sourceforge.net

[EG777CT]

March 25, 2009

Submitted by:

Abhishek Dutta Anjan Nepal Bibek Shrestha Lakesh Kansakar

{ adutta.np, anjan.nepal, bibekshrestha, lakesh.kansakar } @ gmail.com

Institute of Engineering - Pulchowk Campus

Lalitpur, Nepal

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

The undersigned certify that they have read, and recommended to the Institute of Engineering for acceptance, a project entitled "Real Time Face Tracking and Recognition (RTFTR)" by Mr. Abhishek Dutta, Mr. Anjan Nepal, Mr. Bibek Shrestha and Mr. Lakesh Kansakar in partial fulfillment of the requirements for the degree "Bachelor of Computer Engineering".

Supervisor, Dr. Jyoti Tandukar
Associate Professor
Department of Electronics and Computer Engineering, Pulchowk Campus

Supervisor, Mr. Sharad Ghimire
Assistant Professor
Department of Electronics and Computer Engineering, Pulchowk Campus

Internal Examiner, Mr. Rajeev Kumar Kanth
Assistant Professor
Department of Electronics and Computer Engineering, Pulchowk Campus

External Examiner, Mr. Roshan Chitrakar
Assistant Professor
Nepal College of Information Technology (NCIT), Pokhara University, Nepal

Date: March 25, 2009

TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS
DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

Ananda Niketan, Pulchowk, Lalitpur, P.O.Box 4175, Kathmandu, Nepal.
Tel : 5534070, 5521260 extn. 311, Fax : 977-1-5553946, E-mail : ece@ioe.edu.np

Our Ref :

DEPARTMENT ACCEPTANCE

The project entitled "Real Time Face Tracking and Recognition (RTFTR)", in partial requirements of "Bachelor of Computer Engineering" degree of the Institute of Engineering - Pulchowk Campus has been submitted by Mr. Abhishek Dutta, Mr. Anjan Nepal, Mr. Bibek Shrestha and Mr. Lakesh Kansakar and has been accepted as an authentic work conducted in our department.

S Joshi

Prof. Dr. Shashidhar Ram Joshi
Head of the Department
Department of Electronics and Computer Engineering
Institute of Engineering - Pulchowk Campus
Lalitpur, Nepal

Date: March 25, 2009

Table of Contents

List of Figures

Acknowledgment

We offer our gratitude to our project supervisors, Associate Prof. Dr. Jyoti Tandukar and Assistant Prof. Sharad Ghimire, for providing a lot of independence and intellectual guidance while being very encouraging and supportive.

We acknowledge Dr. Shashidhar Ram Joshi for encouraging us to carry out a research based final year project. We also acknowledge Mr. Subash Dhakal, our lecturer for the course *"Image Processing and Pattern Matching"*, for introducing us to the wonderful world of computer vision. We are truly grateful to Assistant Prof. Rajeev Kumar Kanth, our lecturer for the course *"Digital Signal Analysis and Processing"*, for helping us understand the intricacies of Gabor Wavelets.

Last, but not least, we want to gratefully acknowledge the generous support of the following students of IOE-Pulchowk Campus for volunteering in creation of IOE Face Database. (Niraj Sapkota, Sushil Shilpakar, Sabbir Kumar Manandhar, Saroj Gautam, Bibek Raj Dhakal, Subash Basnet, Madhusudan Ram Joshi, Ruchin Singh, Bibek Karmacharya)

Abstract

Real Time Face Tracking and Recognition (RTFTR) is a computer vision project that performs the task of locating human faces in a video stream and recognizing those faces by matching them against the database of known faces. Most face recognition system have a fixed processing pathway consisting of face extraction/recognition (fe/fr) modules. So it is difficult to incorporate new algorithms in those systems. Hence a modular approach to the design of RTFTR system has been used to allow for easy incorporation of new fe/fr algorithms to the system. A flexible 6 stage processing pathway, for visual data, has been defined to realize this modular architecture. It also allows assessment of the collective performance of two or more algorithms when they work in unison.

Keywords : Real Time Face Detection, Real Time Face Recognition, Gabor Wavelets, Neural Network, Adaboost, Subspace LDA, PCA, Cascade Detector

Objectives

The objectives of this project are:

- Develop a system that is capable of realtime face tracking/recognition
- Study the effect of using more than one algorithm for face extraction and face recognition procedure on the overall accuracy of face tracking/recognition
- Apply modular approach to build a face tracking/recognition system that is capable of incorporating new algorithms easily

Chapter 1: Introduction

Real Time Face Tracking and Recognition (RTFTR) is a computer vision project that aims to achieve the following two tasks:

- real time ($\sim$ 25 frame per second) tracking of human faces in a video stream
- recognize those human faces by matching them against the training face database

It also involves studying the performance when existing face extraction(FE) and face recognition(FR) algorithms work in unison. RTFTR implements a modular architecture[1] that forms a processing pathway, for visual data, containing 6 stages as shown in Figure 1.1. This type of design has following advantages:

- It is capable of incorporating new FE/FR algorithms into the existing system, without affecting its operation, by providing a pluggable module environment for the modules implementing new algorithms. A standard interface for each stage has been defined which allows the development of new modules without requiring the developers to have knowledge of the internal operational details of RTFTR.
- Computer vision algorithms consume a large chunk of processing resource available in a general purpose computer. Implementation of such systems design in distributed computing[2] environment is straight forward and involves less complexity. Moreover, achieving realtime operation on a single computer requires huge amount of processing power and memory. Hence with present design of RTFTR, real time operation can be achieved using a distributed computing environment.
- Such modular design will also provide a flexible environment to study the collective performance of different FE/FR algorithms working in unison.

The following aliases have been used to refer to each algorithm implemented in Face Extraction and Recognition stages.

- Neural Network Based Face Extraction - **rowleynn**
- Face Extraction using AdaBoost and Cascaded detector- **adaboost**

[1] the inspiration for such design came from the flowgraph based architecture used by gnuradio applications, `http://www.gnuradio.org`, for radio signal processing. RTFTR implements similar architecture for computer vision application (ie: for 2D discrete signal processing)

[2] OpenMPI can be used to realize a version of RTFTR that distributes FE/FR tasks to different computers and aggregates the result to achieve real time tracking and recognition of human faces. Refer to "Future Work" section 4.2 for details

- Face Recognition using Subspace LDA - **lda**
- Face Recognition using Gabor Wavelet Transform - **kepenekci**

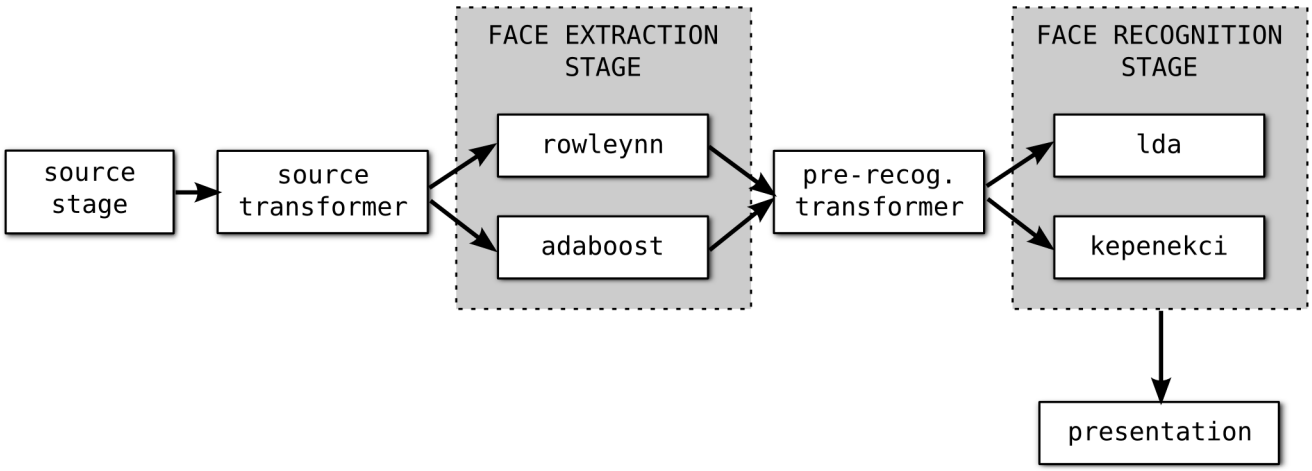


Figure 1.1: System block diagram of RTFTR

The present C++ implementation of RTFTR is based on the system design illustrated in Figure 1.1 and the next sections describe each block of this system diagram in detail.

1.1 Source Stage

This stage (STAGE 1) refers to the source of visual data for the system. This stage can fetch data from different types of visual data sources like CCD camera, video file, sequence of images, etc. Most face recognition systems are tightly coupled with their sources, like a CCD camera, which limits the capability of the software. The source stage in RTFTR allows the rest of the system to fetch visual data from a variety of sources through a standard interface that hides the complexity involved in handling these data sources.

1.2 Source Transformer Stage

Face extraction algorithms usually process raw visual data. However, most of them have good performance when supplied with visual data that has been preprocessed or modified in some desirable way. This stage (STAGE 2) handles the task of transforming the visual data as required by the next stage algorithm. IDENTITY SOURCE TRANSFORMATION, which makes this stage transparent between STAGE 1 and STAGE 3, will be applied to those algorithms that do not require any kind of source transformation.

To illustrate the functioning of this stage, let us consider a hypothetical scenario in which the visual source, a CCD camera, is operating at 30 fps. The algorithm in Face Extraction Stage (STAGE 3) is not capable of processing this huge amount of data and hence only desires to receive the frame numbers 1,5,10,15,20,25,30. So a module in Source

Transformation Stage will filter out the frames that are not required and only pass the desired frames to the next stage. The present implementation of RTFTR uses the following source transformers:

RGB2GRAY Transformer It converts the RGB coded video frames from the source into gray scale images. This module is required as all the algorithms in FE/FR stages do not use color information of a image during processing.

SizeTransformer This module re-sizes the video frame size from 640×480 pixels to 256× 192 pixels. Cubic interpolation technique is used for scaling operation (provided by OpenCV's cvResize() method).

1.3 Face Extraction Stage

This stage performs the task of detecting human faces in an image and extracting the facial regions to pass it to the next stage (Pre Recognition Transformer) for further processing. The two face extraction algorithms implemented in RTFTR are:

1.3.1 Neural Network Based Face Extraction (rowleynn)

The face detection algorithm proposed in the PhD.D thesis "Neural Network-Based Face Detection" [7] has been implemented as a one of the face extraction modules in RTFTR.

The set of neural networks are trained with a large set of face and non-face images. Once the neural networks are trained to give acceptable result, image frames are passed onto them. The three stage detection process first extracts all possible 20×20 pixel sub-images from the original image and its scaled copies, performs window rejection to these sub-images by variance thresholding and then passes the non-rejected sub-images into the neural network. The single real valued output from the neural network between -1 and 1 classifies non-face and face respectively. Finally accuracy of the neural network is improved by arbitration, ie. multiple detections around a single face are clustered into single detection and output from two different neural networks are also combined together. A complete description of this algorithm is presented in section 2.1.

1.3.2 Face Extraction using AdaBoost (adaboost)

A rapid face detection algorithm based on the papers published by Paul Viola and Michael Jones[9] in 2001 has been implemented as the other face extraction module in RTFTR. The algorithm is among the fastest existing face detection algorithms. It uses a technique of performing detections in several stages (cascade) and removing much of the non face containing regions at the early stage. Each stage consists of features which are used for comparing the subwindow as a face or a nonface. The process of calculation of the feature value is rapid by the use of the technique called integral image and the selection of the

features is done by the AdaBoost algorithm. A complete description of this algorithm is presented in section 2.2.

1.4 Pre Recognition Transformer

This stage is similar to Source Transformation Stage except that it performs transformation as required by the face recognition algorithms in STAGE 5.

To illustrate the functioning of this stage, let us consider a hypothetical scenario in which Principal Component Analysis (PCA) [8] based recognition scheme has been implemented in Face Recognition Stage (STAGE 5). PCA algorithm cannot recognize faces having size different than the training image size ie: PCA algorithm is not scale invariant. Hence, the face extracted by STAGE 3 algorithm must be scaled to match the size of training images used for the training of the PCA algorithm. Hence a PRE-RECOG Transformation module, that performs scaling, can be used in this stage to increase the robustness of algorithm implemented in STAGE 5.

1.5 Face Recognition Stage

This stage uses the facial images extracted by FE stage to recognize the human faces by matching against the training face database. This stage implements the hybrid approach for face recognition just as human perception system uses local features and the whole face region to recognize a face [12].

1.5.1 Face Recognition using Subspace LDA (lda)

Linear Discriminant Analysis (LDA) is a holistic face recognition method based on the paper "Discriminant analysis for recognition of human faces images" [3]. It is a statistical approach for classifying samples of unknown classes based on training samples with known classes. This technique aims to maximize between-class (i.e., across users) variance and minimize within-class (i.e., within user) variance. The performance of LDA is improved by Subspace LDA [13] that unifies the concept of Principal Components Analysis - PCA [8] (the Eigenfaces approach pioneered by Kirby and Sirivich in 1988) and LDA.

A complete description of this algorithm is presented in section 2.3.

1.5.2 Face Recognition using Gabor Wavelet Transform (kepenekci)

It is a feature based face recognition approach which implements the algorithm proposed by Bruce Kepenekci in MSc thesis titled "Face Recognition using Gabor Wavelet Transform" [5]. This algorithm was implemented in RTFTR instead of Elastic Bunch Graph Matching (EBGM) technique (as specified in the RTFTR project proposal) because it reduces the complexity of training and face matching procedure by automatic localization

of facial features. The difficulty with EBGM is the requirement of accurate landmark localization (eyes, nose, etc) [1]. Moreover, this technique is capable of using diverse facial features (like mole, scar, etc) for the recognition process. A complete description of this algorithm is presented in section 2.4.

1.6 Presentation Stage

This stage presents the result of face tracking/recognition stages. The present implementation of RTFTR includes a presentation stage that displays the video stream, from source stage, in a simple Graphical User Interface. The information supplied by Face Extraction (regions of the frame where human faces are located) and Recognition (name of the person for each human face) stages is overlaid on the display area as shown in Figure 1.2

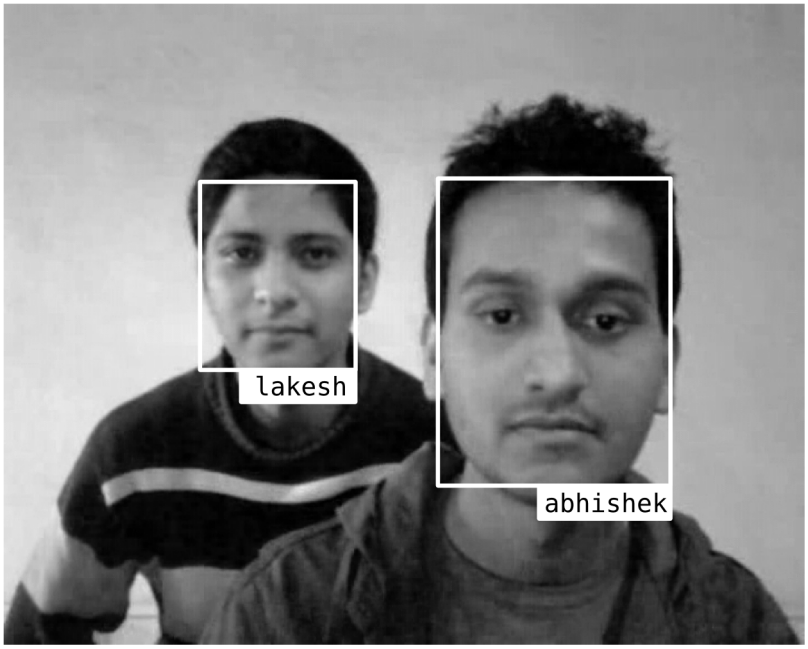


Figure 1.2: Output of the presentation stage

Chapter 2: Literature Review

2.1 Neural Network Based Face Extraction

In this chapter, a neural network-based face detection system inspired by the work of H. A. Rowley [7] is presented. Faces are detected in an unprocessed input image. The system processes and normalizes small windows extracted from the input image. Digital image processing techniques such as normalization of size, position, improvement of light conditions and contrast are used here. A set of neural networks is used in the system and it decides whether the window contains a face or not. In this case, a multilayer perceptron is used. A face detection neural network uses the method of distributing the decision among multiple subnetworks. The result of the face detection system is in the form of a set containing locations of human faces.

2.1.1 Introduction

Face detection is a challenging task. Consider the complexities that are present in an image due to factors like complex backgrounds, many variations in the scene containing face, the randomness of location of face in the scene, variety of sizes, poses and illuminations, etc.

Most face detection algorithms can be categorized into two different groups.

First Group consists of a process where sub-images are extracted from the main image and these are scanned for face like patterns. The process which discriminates face like and non face like regions in the image can be simple techniques like template matching, gray-level dependencies, eigen-space decomposition.

Second Group contains algorithms based on image segmentation. Often color information is available. The first step is pixel-based color segmentation in order to detect skin-colored regions. Then region shape is taken into consideration to distinguish face region from the rest.

The neural network based approach belongs to the first group. Sub windows are extracted from the main image and image enhancement algorithms are applied to them and finally passed into a neural network that distinguishes face containing windows and non face containing windows.

Advanced techniques have been found that outperform the neural network based technique in speed as well as accuracy. However we implement this algorithm to understand

some of the underlying principles of neural networks, image processing and perform comparative analysis with the new Cascaded Detector using AdaBoost algorithm presented in Chapter 2.2.

2.1.2 Multilayer Perceptron

The basic multilayer perceptron (MLP) building unit is a model of an articial neuron. This unit computes the weighted sum of the inputs plus the threshold weight and passes this sum through the activation function (usually sigmoid or sigmoid symmetric)

$$v_j = \theta_j + \sum_{i=1}^{p} w_{ji}\, x_i = \sum_{i=1}^{p} w_{ji}\, x_i \tag{2.1}$$

$$y_j = \varphi_j(v_j) \tag{2.2}$$

where, v_j is a linear combination of inputs $x1, x2, ..., xp$ of neuron , $w_{j0} = {}_j$ is the threshold weight connected to special input $x_0 = 1, y_j$ is the output of neuron j and $\varphi(\Sigma)$ is its activation function. Herein we use a special form of sigmoidal (non-constant, bounded, and monotone-increasing) activation function - logistic function

$$y_j = \frac{1}{1 + \mathrm{e}^{-v_j}} \tag{2.3}$$

and the symmetric sigmoidal activation function given by $2\, y_j - 1$

In a multilayer perceptron, the outputs of the units in one layer form the inputs to the next layer. The weights of the network are usually computed by training the network using the backpropagation (BP) algorithm. A multilayer perceptron represents a nested sigmoidal scheme. Its form for single output neuron is

$$F(x, w) = \varphi\Big(\sum_j w_{0j}\varphi\big(\sum_k w_{kj}\varphi\big(\cdots\varphi\big(\sum_i w_{li}x_i\big)\cdots\big)\big)\Big) \tag{2.4}$$

where $\varphi(\Sigma)$ is a sigmoidal activation function, w_{oj} is the synaptic weight from neuron j in the last hidden layer to the single output neuron o , and so on for the other synaptic weights, x_i is the i^{th} element of the input vector x . The weight vector w denotes the entire set of synaptic weights ordered by layer, then neurons in a layer, and then the number in a neuron.

In this work, also a bootstrap procedure is used. A bootstrap algorithm for face detection can be used as follows: It initializes the training set with examples of the face and non-face classes, then the neural network is trained, new images not containing faces are submitted to the network, false face detections are collected, and a part of the false detections is selected and incorporated to the non-face class of the training set.

2.1.3 Training Data Preparation

A face detector must classify whether a given sub-window belongs to a set of images of face. Variations in the images of the face increase the complexity of the decision boundary to distinguish faces from non faces. Couple of techniques that can be used to reduce the variability in face images are explained.

The Training Set

The FERET Database for the training of faces, we have used the FERET data set. It is publicly available dataset and contains a large number of face images. Two or more faces of more than 1200 different people can be downloaded from `http://www.itl.nist.gov/iad/humanid/feret/`.

Figure 2.1: Sample images from FERET Database.

Face Alignment

The face images in the training set needs to be of the same size and aligned with each other. This alignment helps to reduce the variation in the position, orientation and scale of the faces. For the FERET database, ground truth information are available for all face images. The ground truth information are used to align images with eachother. Alignment is done such that the squared distance between pair of corresponding feature points is minimum. It is done with the help of rotation, scaling and translation.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s\cos\theta & -s\sin\theta \\ s\sin\theta & s\cos\theta \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} = \begin{pmatrix} a & -b & t_x \\ b & a & t_y \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

If we have several corresponding sets of coordinates, this can be further rewritten as follows:

$$\begin{pmatrix} x_1 & -y_1 & 1 & 0 \\ y_1 & x_1 & 0 & 1 \\ x_2 & -y_2 & 1 & 0 \\ y_2 & x_2 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ t_x \\ t_y \end{pmatrix} = \begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \end{pmatrix}$$

Generally, this algorithm converges within five iterations, yielding for each face the transformation which maps it close to a standard position, and aligned with all the other faces. Once the parameters for each images are known, images can be re-sampled using cubic interpolation to produce a cropped and aligned image. The averages and distributions of the feature locations for frontal faces are shown in Figure 2.2.

Algorithm for aligning manually labelled face images

Figure 2.2: (a) Average of upright face examples (b) Positions of average facial feature locations (white circles), and the distribution of the actual feature locations (after alignment) from all the examples (black dots)

1. Initialize F_{avg}, a vector which will be the average positions of each labelled feature over all the faces, with some initial feature locations. In the case of aligning frontal faces, these features might be the desired positions of the two eyes in the input window.

2. For each face i, use the alignment procedure to compute the best rotation, translation, and scaling to align the faces features F_i with the average feature locations F_{avg}. Call the aligned feature locations F'_i.

3. Update F_{avg} by averaging the aligned feature locations F'_i for each face i

4. The feature coordinates in F_{avg} are rotated, translated, and scaled (using the alignment procedure described earlier) to best match some standardized coordinates. These standard coordinates are the ones used as initial values used for F_{avg}.

5. Go to step 2 (ie: repeat steps 2, 3, 4 for each image i).

Preprocessing for Brightness and Contrast

The variations in face image caused by different lighting conditions can be removed using several image enhancement techniques as discussed in Rowley's paper. The preprocessing technique first attempts to equalize the intensity values in across the window. We fit a function which varies linearly across the window to the intensity values in an oval region inside the window as shown in Figure 2.3. Pixels outside the oval may represent the background, so those intensity values are ignored in computing the lighting variation across the face.

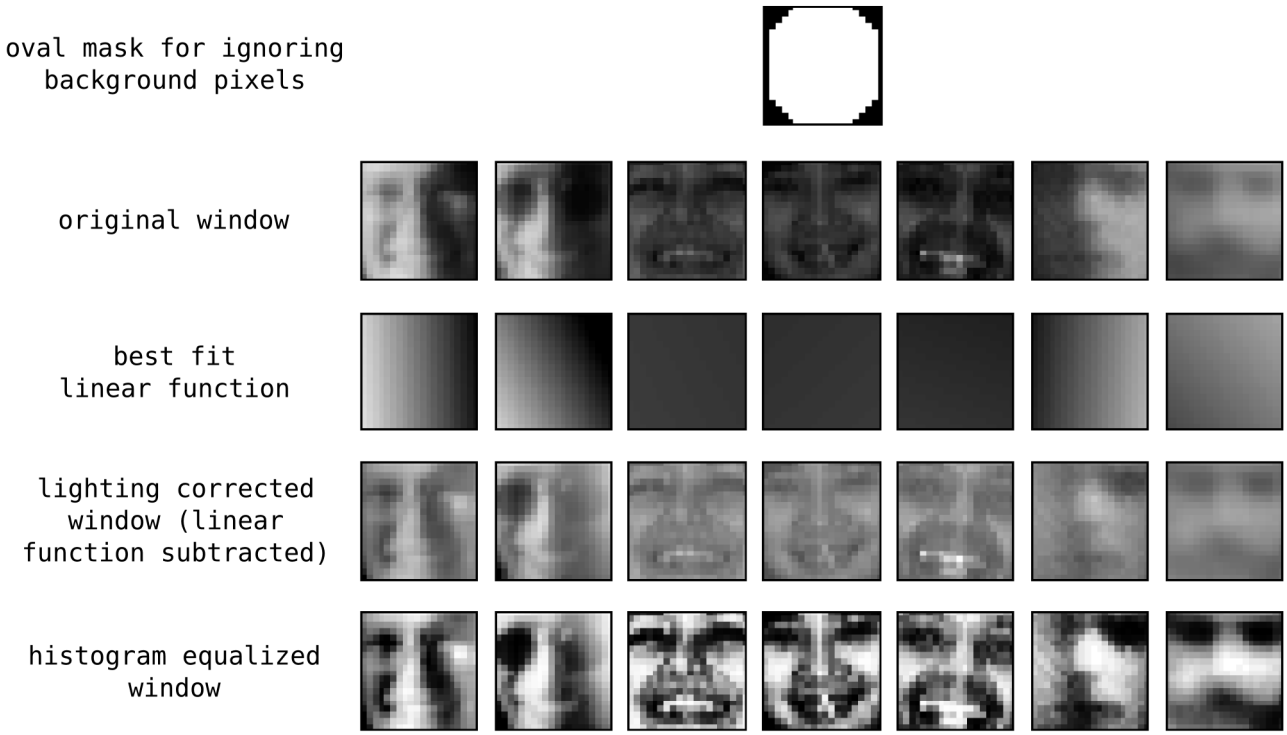


Figure 2.3: The steps in preprocessing a window. First, a linear function is t to the intensity values in the window, and then subtracted out, correcting for some extreme lighting conditions. Then, histogram equalization is applied, to correct for different camera gains and to improve contrast. For each of these steps, the mapping is computed based on pixels inside the oval mask, while the mapping is applied to the entire window

If the intensity of a pixel at (x, y) is $I(x, y)$ then we want to fit this linear model parametrized by a, b, c to the image:

$$\begin{pmatrix} x & y & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = I(x, y)$$

This linear function will approximate the overall brightness of each part of the window,

and can be subtracted from the window to compensate for a variety of lighting conditions.

Next, standard histogram equalization is performed on the image. Histogram equalization spreads a localized histogram of the image to a wider range, the result is contrast adjustment of the image. It helps to enhance lighter and darker regions in the image [4].

Generation of non-faces

A large number of nonface images is required to train the neural network. This is because the variety of nonface images is much greater than that of face images. The nonface images were easily generated from two different techniques.

1. Multiple scenic images not containing human face were fed into a randomly initialized neural network. The output images, sure to contain no human faces were re-sized into 20×20 pixel sized images and put in the nonface images collection.

2. A partially trained network without accurate detection was fed with images containing faces. The detected sub-images from the network were analyzed and false detections containing both nonfaces as well as partial face regions were then added into the nonface images collection. This collection was further used to train the neural network to increase its accuracy.

2.1.4 Structure of Detector System

Individual Face Detection Networks

There are two stages of operation for the system. First it applies a set of neural network-based detectors to an image, and then uses an arbitrator to combine the output. The individual detectors examine each location in the image at several scales, looking for locations that might contain a face. The arbitrator then merges detections from individual networks and eliminate overlapping detections.

The first component of our system is a neural network that receives as input a 20×20 pixel region of the image, and generates an output ranging from 1 to -1, signifying the presence or absence of a face, respectively. To detect faces anywhere in the input, the network is applied at every location in the image. To detect faces larger than the window size, the input image is repeatedly reduced in size (by sub-sampling), and the detector is applied at each size. This network must have some invariance to position and scale. The amount of invariance determines the number of scales and positions at which it must be applied. For the work presented here, we apply the network at every pixel position in the image, and scale the image down by a factor of 1.1 to 1.2 for each step in the pyramid. This image pyramid is shown at the left of Figure 2.4

After a 20×20 pixel window is extracted from a particular location and scale of the input image pyramid, it is preprocessed using the affine lighting correction and histogram equalization steps described in Section 2.1.3. The preprocessed window is then passed to

a neural network. The network has retinal connections to its input layer; the receptive fields of hidden units are shown in Figure 2.4. The input window is broken down into smaller pieces, of four 10×10 pixel regions, sixteen 5×5 pixel regions, and six overlapping 20×5 pixel regions. Each of these regions will have complete connections to a hidden unit, as shown in the Figure 2.4. The network has a single, real-valued output, which indicates whether or not the window contains a face.

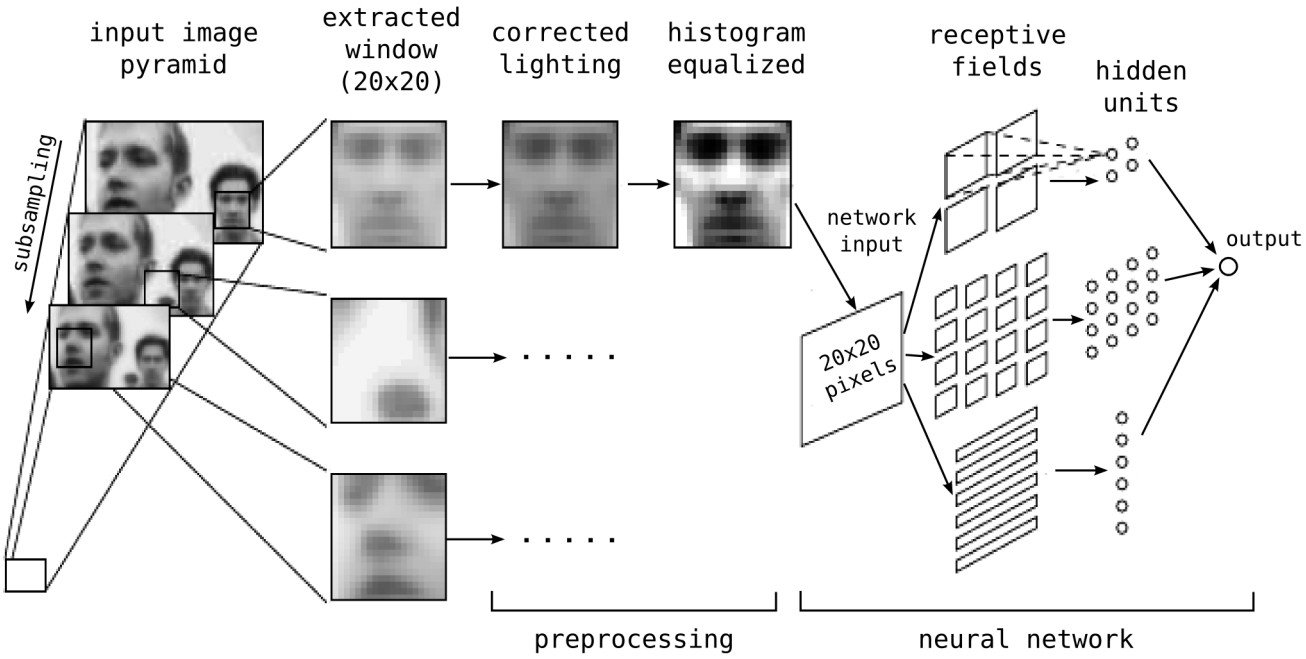
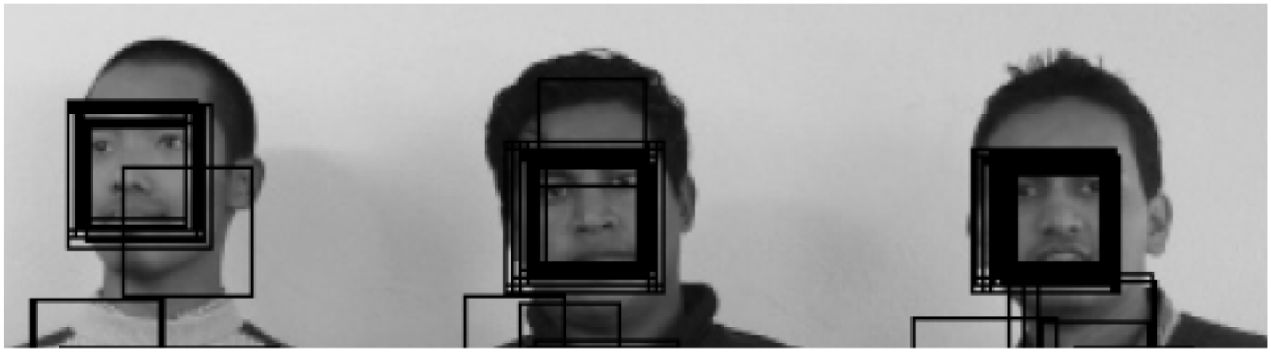


Figure 2.4: The basic algorithm used for face detection

Merging Overlapping Detections

In Figure 2.5, the raw output from a single network will contain a number of false detections. The result can be improved by merging overlapping detections from a the network. Most faces are detected at multiple nearby positions or scales, while false detections often occur with less consistency. This observation leads to a heuristic which can eliminate many false detections. For each location and scale at which a face is detected, the number of detections within a specified neighborhood of that location can be counted. If the number is above a threshold, then that location is classified as a face. The centroid of the nearby detections defines the location of the detection result, thereby collapsing multiple detections. In the experiments section, this heuristic will be referred to as thresholding.

If a particular location is correctly identified as a face, then all other detection locations which overlap it are likely to be errors, and can therefore be eliminated. Based on the above heuristic regarding nearby detections, we preserve the location with the higher number of detections within a small neighborhood, and eliminate locations with fewer detections. Later, in the discussion of the experiments, this heuristic is called overlap



(a)

(b)

Figure 2.5: Image with sub-images above a threshold marked in square box (a), and the result of clustering the detected rectangles (b)

elimination. There are relatively few cases in which this heuristic fails; however, one such case is illustrated in the left two faces in Figure 2.4, in which one face partially occludes another.

If a particular location is correctly identified as a face, then all other detection locations which overlap it are likely to be errors, and can therefore be eliminated. Based on the above heuristic regarding nearby detections, we preserve the location with the higher number of detections within a small neighborhood, and eliminate locations with fewer detections. Later, in the discussion of the experiments, this heuristic is called overlap elimination. There are relatively few cases in which this heuristic fails.

The implementation of these two heuristics is illustrated in Figure 2.6. Each detection by the network at a particular location and scale is marked in an image pyramid, labelled the "output" pyramid. Then, each location in the pyramid is replaced by the number of detections in a specified neighborhood of that location. This has the effect of "spreading out" the detections. Normally, the neighborhood extends an equal number of pixels in the dimensions of scale and position, but for clarity in Figure 2.6 detections are only spread out in position. A threshold is applied to these values, and the centroids (in both position and scale) of all above threshold regions are computed.

All detections contributing to the centroids are collapsed down to single points. Each centroid is then examined in order, starting from the ones which had the highest number of detections within the specied neighborhood. If any other centroid locations represent

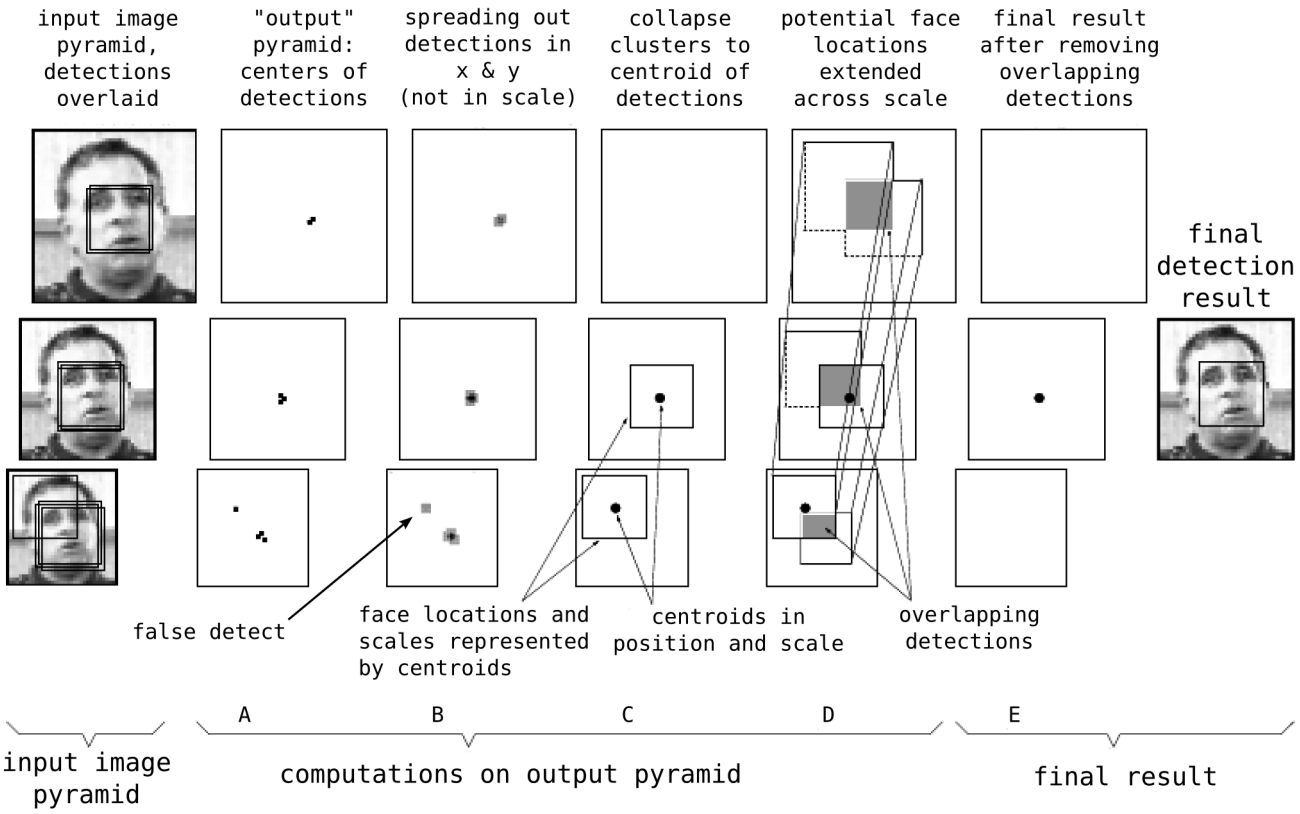


Figure 2.6: The framework used for merging multiple detections from a single network : (A) The detections are recorded in an image pyramid. (B) The detections are spread out and a threshold is applied. (C) The centroids in scale and position are computed, and the regions contributing to each centroid are collapsed to single points. In the example shown, this leaves only two detections in the output pyramid. (D) The nal step is to check the proposed face locations for overlaps, and (E) to remove overlapping detections if they exist. In this example, removing the overlapping detection eliminates what would otherwise be a false positive.

a face overlapping with the current centroid, they are removed from the output pyramid. All remaining centroid locations constitute the final detection result.

2.2 Face Extraction using AdaBoost and Cascaded detector

Cascaded face detector by the use of AdaBoost algorithm is based on the paper by Paul Viola and Michael J. Jones [9]. The paper describes a face detection framework that is capable of processing images extremely rapidly[1] while achieving high detection rates. There are three key contributions. The first is the introduction of integral image for rapid computation of the features used in the detector. The second is the AdaBoost algorithm for selection of efficient classifiers from a large population of potential classifiers. Third is the method for combining the classifiers generated by AdaBoost algorithm into a cascade, which has the property of removing most of the non-face images in the early stage by simple processing, and focus on complex face like regions in the later stages which take higher processing time.

This algorithm is selected for the face detection because of its property of detecting faces extremely rapidly which would help in our project in processing real time videos.

2.2.1 Features

AdaBoost algorithm classifies images based on the value of simple features. The simple features are the reminiscent of Haar basis functions as shown in Figure 2.8. In this method of face detection, we have used three kinds of features: two rectangle feature, three rectangle feature and four rectangle feature. Two rectangle feature is the difference between the sum of the pixels within two rectangular regions. The regions have the same size and are horizontally or vertically adjacent. Similarly three rectangle feature is the value obtained by subtracting the sum of pixels of outer rectangles from the center rectangle. Finally, four rectangle feature is the difference between the diagonal pairs of rectangles. In short, the feature value is the difference between the sum of pixels of the white region to the dark region of the features as shown in the Figure 2.7

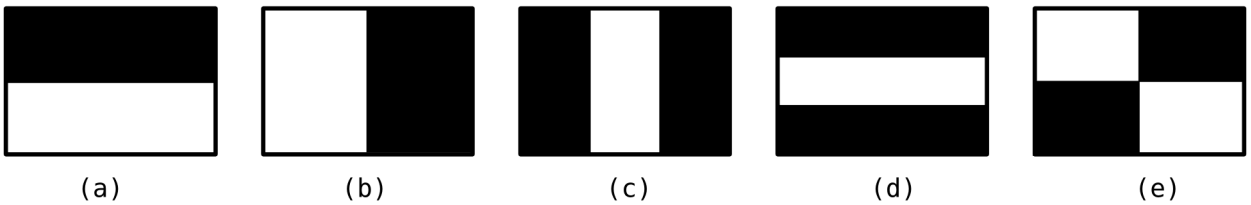


Figure 2.7: Features used in AdaBoost algorithm. (a) and (b) are two rectangle features, (a) being two vertically stacked rectangular feature. Similarly, (c) and (d) are three rectangle feature and (e) is the four rectangle feature

[1] as described in the paper, detection rate of 15 frames per second on a 384×288 pixel images on a 700MHz Intel Pentium III

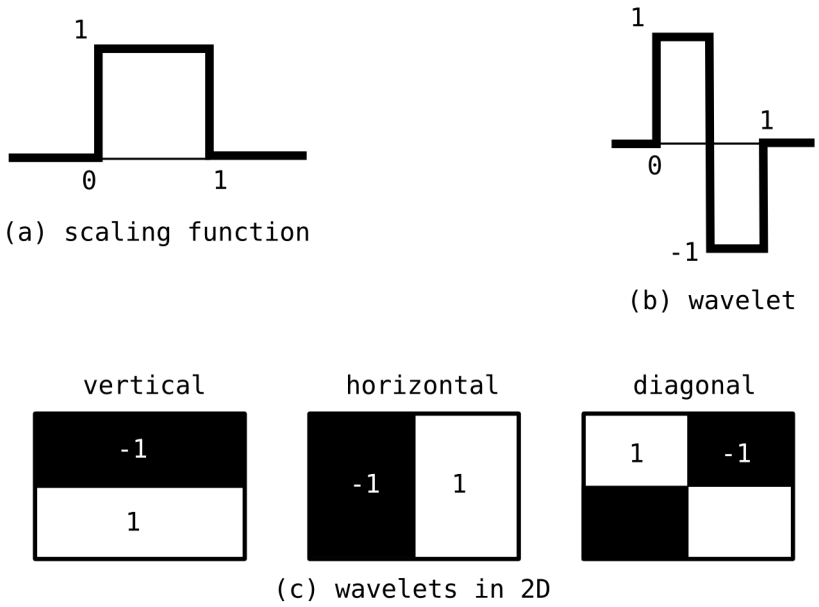


Figure 2.8: The Haar wavelet framework: (a) and (b) the Haar scaling function and wavelet, (c) the three types of 2-dimensional non-standard Haar wavelets: vertical, horizontal and diagonal

Computing the Features

Given the base resolution of the detector sub-window is 24x24, the exhaustive set of rectangle features is quite large, more than 160,000. In simple words, the features have to be generated such that it starts from every possible pixel of the detector sub-window and also should cover all the widths and heights possible. This should be done for all types of features (two rectangle, three rectangle and four rectangle features). The algorithm for the feature generation is given below:

1. Start from the initial position of the sub-window, $(1, 1)$. Create a rectangular feature of size such that 1 pixel represents the black region and 1 pixel represents the white region

2. Move the feature throughout the sub-window

3. Increase the size of the feature such that both the black region and white region size increases by 1 in horizontal direction. Goto step 2, until the width of the feature equals the width of the sub-window

4. Increase the size of the feature such that both the black region and dark region size increases by 1 in vertical direction. Goto step 2 until feature height equals the height of the detector sub-window

5. For all the starting position and all the width and height of the feature (from steps 1 to 4), store the co-ordinates as (x, y, w, h) where x= starting x-coordinate, y = starting y-coordinate, w = width and h = height of the feature

6. Repeat steps 1 to 5 for all types of rectangular features (i.e. two rectangle, three rectangle and four rectangle feature)

Total number of features generated is 162,336. The feature count of different features are given below.

- Two vertically stacked rectangle feature : 43,200
- Two horizontally stacked rectangle feature: 43,200
- Three vertically stacked rectangle feature : 27,600
- Three horizontally stacked rectangle feature: 27,600
- Four rectangle feature: 20,736

Generation of three horizontally stacked features is illustrated in Figure 2.9 and achieved by the pseudo-code given below:

```
H=24; W=24;
for (h=1; h<=H; h++)
    for (w=3; w<=W; w+=3)
        for (y=0; y<=H-h; y++)
            for (x=0; x<=W-w; x++)
                Save(x, y, w, h)
```

This pseudo-code saves only the starting position with the width and height. To distinguish between the type of feature (like two vertically stacked, two horizontally stacked, three horizontally stacked etc), it is distinguished by saving a type as well. So, during the calculation, the type shows that this feature is a horizontally stacked three rectangle feature and the middle rectangle has to be subtracted from the outer rectangles.

2.2.2 Integral Image

For the fast computation of feature values, integral image of an image can be used. Integral image size is the same size of the image, but the value at any location x,y contains the sum of the pixels above and to the left of x, y, inclusive as shown in Figure 2.10

$$ii(x, y) = \sum_{x' \leq x \, y' \leq y} i(x', y')$$

where, $ii(x, y)$ is the integral image and $i(x, y)$ is the original image. So, for the computation of $ii(x, y)$, following pair of recurrences can be used:

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{2.5}$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{2.6}$$

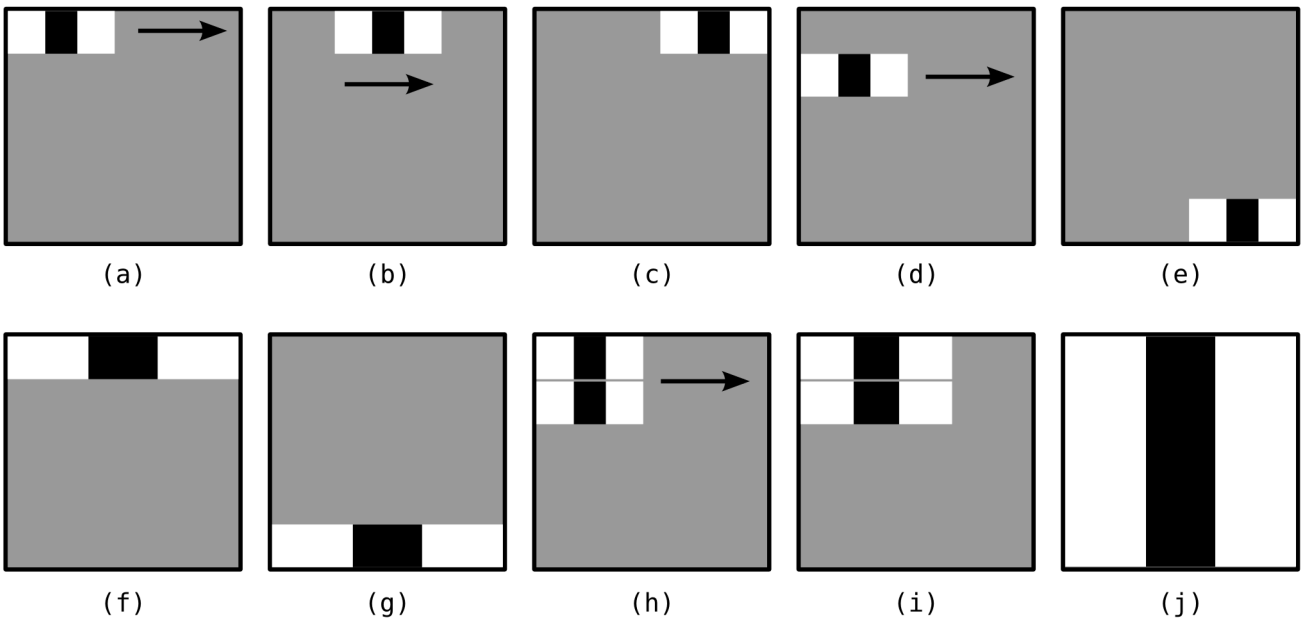


Figure 2.9: Process of generation of three horizontally stacked rectangle features : (a) Initially 1 pixel each of white and black region is at the initial position (1, 1). It is then moved throughout the window (a to e). The feature is then increased in width by one pixel each time and moved throughout the window until the width of the feature become equal to the width of the window (f to g). Then the feature height is increased by one pixel (h) each time and moved throughout the window until the width of the feature become equal to the height of the window (i). This process continues until the whole feature covers the window (j)

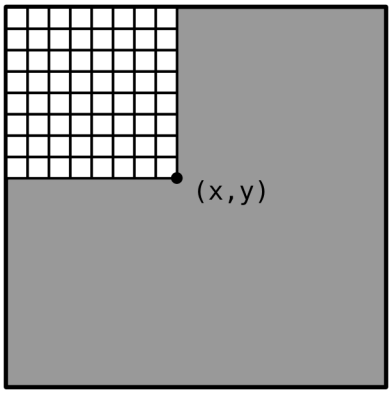


Figure 2.10: Integral image is the sum of the pixels at the left and above of the point (x,y) inclusive

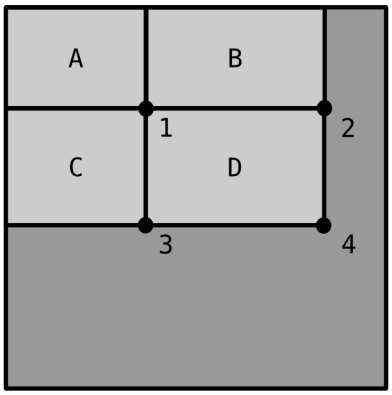


Figure 2.11: The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can be computed as $4 + 1 - (2 + 3)$

where $s(x, y)$ is the cumulative row sum and $s(x, -1) = 0$, and $ii(-1, y) = 0$. For the computation of features using integral image, only few array references has to be done. For two rectangle feature, 6 array references has to be done. For three rectangle feature, 8 array references has to be done and for four rectangle features, 9 array references has to be done. Few simple operations like addition and subtraction has to be done for the feature evaluation, so the feature value computation is very fast using the technique of integral image. Initial computation of the integral image takes more time, but it has to be only computed once, and after its computation any feature can be computed rapidly using the same integral image with the maximum of 9 array references per feature. The computation of feature value for three horizontally stacked rectangle feature is shown in the Figure 2.12. For the three horizontally stacked rectangle feature, eight points are

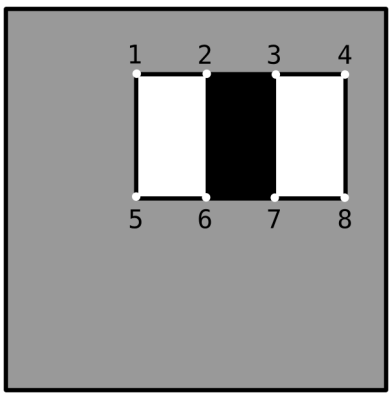


Figure 2.12: Feature evaluation using integral image

computed as shown. Then the integral image value at that point $ii(x, y)$ is found at each point. The feature value is computed as $p1+p8-(p5+p4)-2*(p2+p7-(p3+p6))$, where pN is the integral image value at point N. This value is computed using the technique shown in Figure 2.11 for computing value for a single rectangle.

For the face detection faces at multiple levels, other techniques use the method of image pyramid. Using the method of integral image, the detection of faces at multiple level is much faster than the image pyramid method. This is because in integral image method, the feature rectangle can be increased in size instead of computing the image pyramid. The same integral image can be used for the feature evaluation requiring same computation time as that at the original feature scale. But in case of image pyramid method, the scaling of the image itself takes a lot of time and further the detection time is high in all level of image pyramids.

2.2.3 AdaBoost Algorithm

In a single subwindow of size 24×24, there are more than 160,000 features. Computation of all these features in detection can be very expensive in time. But there are efficient classifiers among these large number of features which when efficiently combined gives a robust classifier. The selection and combination of these efficient classifiers from among the large set is done by AdaBoost algorithm. Viola and Jones have given a variant of AdaBoost algorithm to train the classifiers.

AdaBoost algorithm is given a feature set with large number of features and the training images (labeled faces and non-faces, that is supervised learning) . AdaBoost learning algorithm is used to boost the classification performance of simple learning algorithm (eg. a simple perceptron). It does this by combining a collection of weak classification functions to form a strong classifier. In the language of boosting the weak classification functions are called weak learners, because we do not expect even the best weak learner to classify the training data well (that is, for a given problem even the best perceptron may classify the training data correctly only 51% of time). The selection of the best classifier for each round is the one which gives lowest classification error in the training data.

Boosting is the method of combining the weak classifiers to form a strong classifier. In the later round of learning, AdaBoost finds the new weak classifier after re-weighting the training examples such that it emphasizes on the examples incorrectly classified by the previous weak classifiers.

The weak classification function selects a single rectangular feature which best separates the positive and negative examples, determines the threshold and gives a weak classifier. So, weak classifier ($h_j(x)$) consists of a rectangular feature (f_j), threshold (θ_j) and a polarity (p_j) indicating the direction of inequality:

$$h_j(x) = \begin{cases} 1 & \text{if } p_j \cdot f_j(x) < p_j \cdot \theta_j \\ 0 & \text{otherwise} \end{cases}$$

Boosting algorithm which when run for T number of rounds selects T number of best weak classifiers and combines these to form a strong classifier also finding the threshold for the strong classifier. The boosting algorithm used for the selection of the rectangular

feature and combining those to form a strong classifier is given below:

1. Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively

2. Initialize weights $W_{1,i} = \frac{1}{2\,m}, \frac{1}{2\,l}$ for $y_i = 0, 1$ respectively, where m and l are the numbers of negatives and positives respectively

3. For $t = 1, \ldots, T$:

   (a) Normalize the weights
   $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$ so that w_t is a probability distribution

   (b) For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t
   $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$

   (c) Choose the classifier, h_t, with the lowest error ϵ_t

   (d) Update the weights:
   $w_{t+1,i} = w_{t,i}\, \beta_t^{1-\epsilon_i}$

4. The final strong classifier is:

$$h(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t\, h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

   where $\alpha_t = \log \frac{1}{\beta_t}$

The description of how the algorithm works is now presented. Labeled faces and non faces (labeled 1 and 0) are provided. The initial weight of all the examples (training faces and non faces) is calculated . Then the boosting iteration starts. In each iteration, the weights are normalized and for each feature a threshold is computed which then gives a weak classifier (computation of threshold is explained later). For each weak classifier, its error of classification in the training examples is computed by weighted sum. Among the more than 160,000 features, the classifier with the minimum of the errors is evaluated, and this is the weak classifier selected by AdaBoost algorithm for this round of boosting. For the next round of boosting, the weights are updated such that the weights for the examples wrongly classified by previous classifier is higher than the ones that are rightly classified. This way, in the next round the classifier is selected which focuses on these examples for finding the efficient classifier and the classifiers selected for different round of boosting will be different. The final strong classifier is created by combining the weak classifiers generated and setting the threshold as half the weight given to the classifiers.

For the computation of the threshold for each of the features, a table has to be maintained. The examples are sorted according to the feature value. For each element in the

Table 2.1: Table maintained to compute the threshold for each feature

error	weight	example (+/-)	feature value	S^+	S^-
0.028	0.017	1	-30	0.028	0.031
0.008	**0.02**	**1**	**23**	**0.008**	**0.031**
0.0235	0.012	0	89	0	0.019
0.031	0.019	0	97	0	0
0.031	0.008	1	105	0	0

sorted list, four sums are maintained and evaluated: the total sum of positive example weights T^+, the total sum of negative example weights T^-, the sum of positive weights below the current example S^+ and the sum of negative weights below the current example S^-. The error for a threshold which splits the range between the current and previous example in the sorted list is:

$$e = min(S^+ + (T^- - S^-), S^- + (T^+ - S^+))$$

or the minimum of the error of labeling all examples below the current example negative and labeling the examples above positive versus the error of the converse. Then the feature value is selected as threshold for that feature which has the minimum of these error values. After finding the threshold, we also need to find the polarity of the classifier. Polarity gives the direction of inequality, that is if the feature value of an example is greater than the threshold, should it be labeled as a face or a non face. To find the polarity, following relation is used for the example from which threshold is found:

$$p = \begin{cases} 1 & \text{if } S^+ + (T^- - S^-) < S^- + (T^+ - S^+) \\ -1 & \text{otherwise} \end{cases}$$

A sample of the table maintained to compute the threshold for each feature is shown in Table 2.1. All the examples are sorted according to the feature value. The values of T^+ = 0.045 and T^- = 0.031 can be computed only once for each feature, and hence are not shown in the table. Here, the threshold for this feature is computed as 23 and the polarity as -1. Figure 2.13 shows one of the first features that is selected by AdaBoost algorithm.

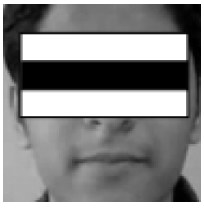


Figure 2.13: One of the first features selected by AdaBoost algorithm

The difference between the intensity of the region of eyes and the sum of region of forehead and cheek is higher in most of the faces and not always high in non faces. So, this feature gives lower classifying error and AdaBoost selects this as a feature.

AdaBoost algorithm takes a very long time for the training. This is because more than 160,000 features are to be processed in selecting one of the features in one round of boosting. A lot of time is taken for finding the threshold and evaluating error of each feature. Also, the number of training examples is also very large.

2.2.4 Cascade of classifiers

AdaBoost algorithm is capable of making a good detector with the use of many features which can result in high detection rate. But a monolithic detector made by using AdaBoost only has low detection speed. This is because the feature values have to be evaluated and classified for all the sub-windows, regardless of its complexity. To increase the detection speed, several stages of the classifiers are made which reject most of the sub-windows at the early stage and give more time on less number of sub-windows which are complex and have face-like regions. The cascade is made to reject the sub-windows because the number of non face sub-windows is far greater than the number of face sub-windows in an image. Figure 2.14 illustrates of the cascade of classifiers for the face

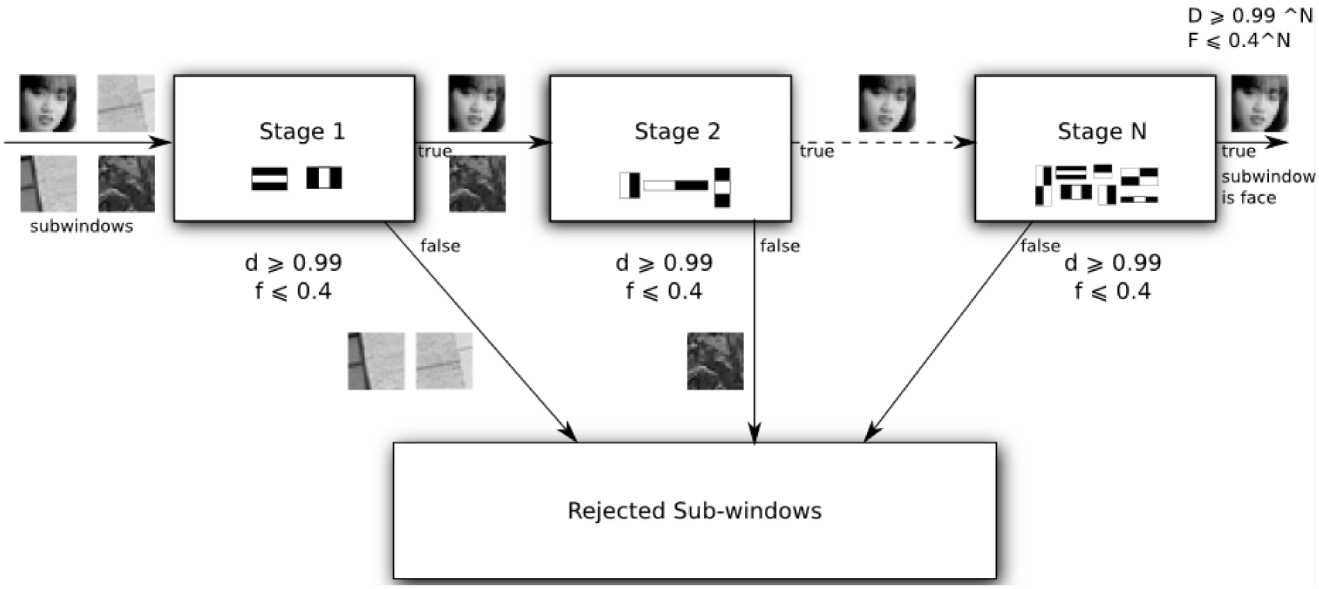


Figure 2.14: Cascade of classifiers for the face detection

detection. Each stage (strong classifier) is trained by AdaBoost algorithm. When a sub-window is passed to the cascade, most of the simple sub-windows not containing faces are rejected by simple processing of two or three features. Computation time increases as the stage number increases because of the complexity of the sub-window. More and more non face sub-windows are rejected as the sub-window moves further into the cascade. The sub-windows which are classified as face by all the stages are the final detections.

Training of the cascade of classifiers

For the training of the cascade of classifiers, each stage is generated by using AdaBoost algorithm. To achieve high detection rate and low false positive rate of the final detector, each stage is made such that it has the detection rate greater or equal to a given value and false positive rate less than or equal to a given value. If this condition is not met, the stage is again trained by specifying larger number of classifiers than the previous. The final false positive rate and the detection rate of the cascade is given as:

$$F = \prod_{i=1}^{K} f_i$$

where, F is the false positive rate of the cascaded classifier, f_i is the false positive rate of the ith stage and K is the number of stages. The detection rate is:

$$D = \prod_{i=1}^{K} d_i$$

Where D is the detection rate of the cascaded classifier, d_i is the detection rate of ith stage and K is the number of stages. Every face detection system requires high detection rate and a low false positive rate. Given the concrete goal for overall false positive and detection rates, target rates can be determined for each stage in the cascade process. For example, detection rate of 0.9 can be achieved by 10 stages with the detection rate of each stage $d_i = 0.99(0.99^{10} = 0.9)$. The detection rate of 0.99 for each stage can be found for a high false positive rate. But for 10 stages, if the false detection rate of each stage is 0.4, then the final false detection rate will be $0.000105(0.4^{10})$.

For the training of the cascade, initially faces and non-faces for the training and faces and non-faces for the validation are provided. AdaBoost algorithm is designed to find a strong classifier which best classifies the faces from the non-faces and is not designed to produce high detection rate with high false detection rate. So, to achieve high detection rate of say 0.99, the stage has to be checked with the validation image to find the current detection rate, and the threshold of the strong classifier has to be reduced to achieve the detection rate of 0.99. In doing so, the false positive rate also increases, and if it is greater than the desired false positive rate (say 0.4), then the stage has to be trained again with more number of features included.

After the generation of one stage, the non-face training images for the next stage are generated by running the detector up to the previous stage on a set on non-face containing images. These false detections are then used as non-face training images for the training of the new stage. This way, the complexity of the non-face increases as the number of stages increases, resulting in more and more face-like non-face training images in later stages. So, the number of classifiers in the stage also increases as the stage number increases.

Training algorithm for cascade detector

The training algorithm is as follows:

- User selects values of f, the maximum acceptable false positive rate per layer and d, the minimum acceptable detection rate per layer
- User selects target overall false positive rate, F_{target}
- P = set of positive examples
- N = set of negative examples
- $F_0 = 1.0; D_0 = 1.0$
- $i = 0$
- while $F_i > F_{target}$
  - $i \leftarrow i + 1$
  - $n_i = 0; F_i = F_{i-1}$
  - while $F_i > f \times F_{i-1}$
    - $n_i \leftarrow n_i + 1$
    - Use P and N to train a classifier with n_i features using AdaBoost
    - Evaluate current cascade classifier on validation set to determine F_i and D_i
    - Decrease threshold for the i^{th} classifier until the current cascade classifier has a detection rate of at least $d \times D_{i-1}$ this also affects F_i)
  - $N \leftarrow \emptyset$
  - If $F_i > F_{target}$ then evaluate the current cascade detector on the set of non-face images and put any false detections into the set N

The algorithm for the generation of cascade of classifiers is shown in the figure above. First, the maximum acceptable false positive rate per stage f, minimum acceptable detection rate per layer d and overall target false positive rate of the final cascade F_{target} has to be defined. Also, the training set of positive P and negative examples N , set of validation positive and negative examples, and the non face source images from which the non faces for the later stages of the cascade will be generated, has to be provided. Then the stages of the cascade are generated using AdaBoost algorithm. Once the stage is generated, its threshold is reduced to meet the requirement of minimum detection rate. If reduction of threshold in doing so does not meet the requirement for the maximum false positive rate, more classifiers are added to the stage using AdaBoost algorithm. After these conditions are met, all the non face training images are removed and new ones are generated from the non face source images using the cascade generated so far. Then the new stage generation starts. This continues until the final false positive rate is less than or equal to the target false positive rate.

2.3 Face Recognition using Subspace LDA

There are basically three approaches to face recognition :

Holistic approach This approach works on the whole face region as the raw input to the recognition system. It doesn't consider individual features like eye, nose, mouth etc but works on the whole face for comparing similarity. Algorithms like PCA (Principle Component Analysis) and LDA(Linear Discriminant Analysis) are examples of holistic approach based face recognition algorithms.

Feature based approach Unlike holistic approach, feature based approach considers individual features like eyes, nose, mouth, mole etc and compares the similarity between the individual features for a match. Algorithms like Elastic Bunch Graph Matching (EBGM), Face recognition using Gabor Wavelet Transform (discussed in Chapter 2.4 are examples of feature based face recognition algorithms.

Hybrid approach Hybrid approach is the mixture of holistic approach and feature based approach. Compared to the individual holistic and feature based approach, hybrid approach is considered to have the best performance.

Subspace Linear Discriminant Analysis (LDA) is a hybrid algorithm for face recognition. It was initially proposed in [13] followed by [11]. It consists of two separate face recognition algorithms which are:

- Principle Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)

We will discuss about these two algorithms in sections

2.3.1 Principle Component Analysis (PCA)

PCA is one of the most primitive algorithms used for face recognition proposed by Pentland and Turk [8]. It is a holistic approach based face recognition algorithm. It is also known as Eigenfaces. Any image (suppose of dimension $N_x x N_y$) can be thought of as a point in a $P(= Nx \times Ny)$ dimensional image space having values in the range of pixel values. For the case of gray scale images, in each dimension the image could have a value in between 0 and 255. An image can be thought as a point in the image space by converting the image to a long vector by concatenating each column of the image one after the other. The Eigenface method tries to find a lower dimensional space for the representation of the face images by eliminating the variance due to non-face images; that is, it tries to focus on the variation just coming out of the variation between the face images. Eigenface method is the implementation of Principal Component Analysis (PCA) over images. In this method, the features of the studied images are obtained by looking for the maximum

deviation of each image from the mean image. This variance is obtained by getting the eigenvectors of the covariance matrix of all the images.

The Eigenface space is obtained by applying the eigenface method to the training images. Later, the training images are projected into the Eigenface space. Next, the test image is projected into this new space and the distance of the projected test image to the training images is used to classify the test image. The train image projection with the minimum distance from the test image projection is the correct match for that test face.

Training operation in PCA

Let I be an image of size (N_x, N_y) pixels, then the training operation of PCA algorithm can be expressed in mathematical terms as

1. Convert the training image matrix I of size (N_x, N_y) pixels to the image vector Γ of size $(P \times 1)$ where $P = N_x \times N_y$ (ie: the train image vector Γ is constructed by stacking each column of train image matrix I)

2. Create a training set of training image vectors such that its size is $P \times M_t$ where M_t is the number of training images
   $\Gamma_{P \times M_t} = [\Gamma_1 \Gamma_2 \dots \Gamma_{M_t}]$ where, Γ_i represents the image vector of i^{th} training images

3. Compute arithmetic average (mean face Ψ) of the training image vectors at each pixel point given by:

$$\Psi_{P \times 1} = \frac{1}{M_t} \sum_{i=1}^{M_t} \Gamma_i$$

4. Obtain mean subtracted vector (Φ) by subtracting the mean face from the each training image vector as given below:

$$\Phi_i = \Gamma_i - \Psi$$

5. Create the difference matrix (A) which is the matrix of all the mean subtracted vectors (Φ) and is given by:

$$A_{P \times M_t} = [\Phi_1 \Phi_2 \dots \Phi_{M_t}]$$

6. Compute the covariance matrix (X) given by:

$$X_{P \times P} = A \cdot A^T$$

7. Compute the eigenvector and eigenvalue of the covariance matrix (X). The dimension of covariance matrix (X) is $P \times P = (N_x \cdot N_y) \times (N_x \cdot N_y)$. For an image of typical size, the task of computing eigenvector of a matrix of such huge dimension is a computationally intensive task. However a way around this problem exists and is described below: Let us define a matrix $Y_{M_t \times M_t}$ such that

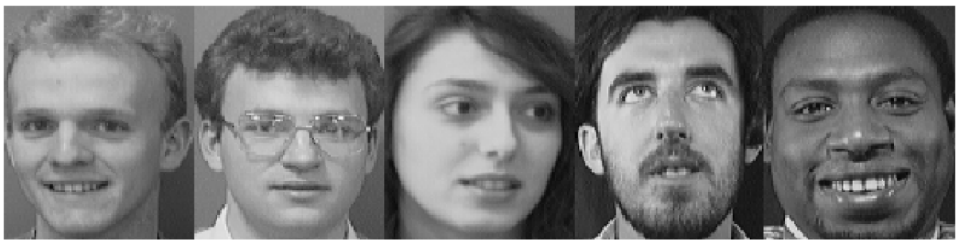
$$Y = A^T \cdot A$$

$$\begin{aligned}
Y \cdot \nu_i &= \mu_i \cdot \nu_i \\
(A^T \cdot A) \cdot \nu_i &= \mu_i \cdot \nu_i && (\because Y = A^T \cdot A) \\
(A \cdot A^T \cdot A) \cdot \nu_i &= A \cdot \mu_i \cdot \nu_i && \text{(premultiplying both sides by matrix A)} \\
A \cdot A^T \cdot A \cdot \nu_i &= \mu_i \cdot (A \cdot \nu_i) && \text{(as } \mu_i \text{ is scalar, we can rearrange the terms)} \\
X \cdot (A \cdot \nu_i) &= \mu_i \cdot (A \cdot \nu_i) && \text{(where } X = A \cdot A^T\text{)} \\
X \cdot \vartheta_i &= \mu_i \cdot \vartheta_i && \text{(where } \vartheta_i = (A \cdot \nu_i)\text{)}
\end{aligned}$$

which is of the form $B \cdot I = \lambda I$ where I and λ are the eigenvector and eigenvalue of B. **Hence, we conclude that ϑ and μ_i is one of the the eigenvectors and eigenvalues of matrix $X = A \cdot A^T$ respectively**.

Thus we obtain the eigenvector $\vartheta_i (= A \cdot \nu_i)$ of covariance matrix $X = A \cdot A^T$ by first computing the eigenvector (ν_i) of matrix $Y = A^T \cdot A$. This formulation brings substantial computational efficiency.

Some example images and mean of the images from the ORL database are given below. The eigenfaces are in fact $(P \times 1)$ vectors for the computations; in order to see what they look like, they are rearranged as $(Nx \times Ny)$ matrices. Instead of using M_t



(a) Example images from the ORL database

(b) Mean image

Figure 2.15: Face images in ORL database and a sample mean face

of the eigenfaces, $M' \leq M_t$ of the eigenfaces can be used for the eigenface projection. This is achieved to eliminate some of the eigenvectors with small eigenvalues, which contribute less variance in the data. Eigenvectors can be considered as the vectors

Figure 2.16: some examples of eigenfaces

pointing in the direction of the maximum variance and the value of the variance the eigenvector represents is directly proportional to the value of the eigenvalue (i.e, the larger the eigenvalue indicates the larger variance the eigenvector represents). Hence, the eigenvectors are sorted with respect to their corresponding eigenvalues. The eigenvector having the largest eigenvalue is marked as the first eigenvector, and so on. In this manner, the most generalizing eigenvector comes first in the eigenvector matrix. In the next step, the training images are projected into the eigenface space and thus the weight of each eigenvector to represent the image in the eigenface space is calculated. This weight is simply the dot product of each image with each of the eigenvectors.

8. Determine the projection of a training image on each of the eigenvectors as given below:
   $\omega_k = \vartheta_k^T \cdot \Phi = \vartheta_k^T \cdot (\Gamma - \Psi) \qquad k = 1, 2, \ldots, M_t$

9. Determine the weight matrix Ω - which is the representation of the training images in eigenface space
   $\Omega_{M' \times 1} = [\omega_1 \, \omega_2 \, \ldots \, \omega_{M'}]^T$

The training operation of PCA algorithm ends with the computation of the weight matrix. At this point, the images are just composed of weights in the eigenface space, simply like they have pixel values in the image space. The important aspect of the eigenface transform lies in this property. Each image is represented by an image of size $(N_x \times N_y)$ in the image space, whereas the same image is represented by a vector of size $(M' \times 1)$ in the eigenface space.

Figure 2.17: representation of training face images as weighed sum (given by Ω) of eigenfaces

Recognition operation in PCA

When a new probe(test) image is to be classified, it is also mean subtracted and projected onto the eigenface space and the test image is assumed to belong to the nearest class by calculating the Euclidean distance of the test image projections to that of the training image projections.

Let T be an image of size (N_x, N_y) pixels, then the recognition operation of PCA algorithm can be expressed in mathematical terms as:

1. Convert the test image matrix T of size (N_x, N_y) pixels (*the size of test image must be same as that of the training images*) to the image vector Γ_T of size $(P \times 1)$ where $P = N_x \times N_y$ (ie: the test image vector Γ_T is constructed by stacking each column of test image matrix T)

2. Obtain mean subtracted test image vector (Φ_T) by subtracting the mean face (computed during the training session) from the test image vector as given below:
   $\Phi_T(P \times 1) = \Gamma_T - \Psi$

3. Determine the projection of a test image on each of the eigenvectors as given below:
   $\omega_k = \vartheta_k^T \cdot \Phi_T = \vartheta_k^T \cdot (\Gamma_T - \Psi) \qquad k = 1, 2, \ldots, M'$

4. Determine the weight matrix Ω - which is the representation of the test image in eigenface space
   $\Omega_{T\,(M' \times 1)} = [\omega_1\, \omega_2\, \ldots\, \omega_{M'}]^T$

5. Compute the value of similarity function of given test image for each training image. Similarity of test image with i^{th} training image is defined as:
   $\delta_i = \|\Omega_T - \Omega_{\Psi_i}\| = \sqrt{\sum_{k=1}^{M'} (\Omega_{T_k} - \Omega_{\Psi_{i_k}})^2}$
   where (δ_i) is the Euclidean distance (L2 norm) between projections of images in

face space. **The training face image which has the minimum distance (δ) is the face that is closely matching with the test face**.

2.3.2 Linear Discriminant Analysis (LDA)

LDA is another holistic approach based face recognition method proposed by Etemad and Chellapa [3]. However unlike Eigenfaces which attempts to maximize the scatter of the training images in face space, it attempts to maximize the between class scatter, while minimizing the within class scatter. In other words, moves images of the same face closer together, while moving images of different faces further apart. Overall it tries to increase the ratio of the between class scatter to within class scatter. The difference in the space chosen by LDA and PCA [2] is illustrated in Figure 2.18. In mathematical terms

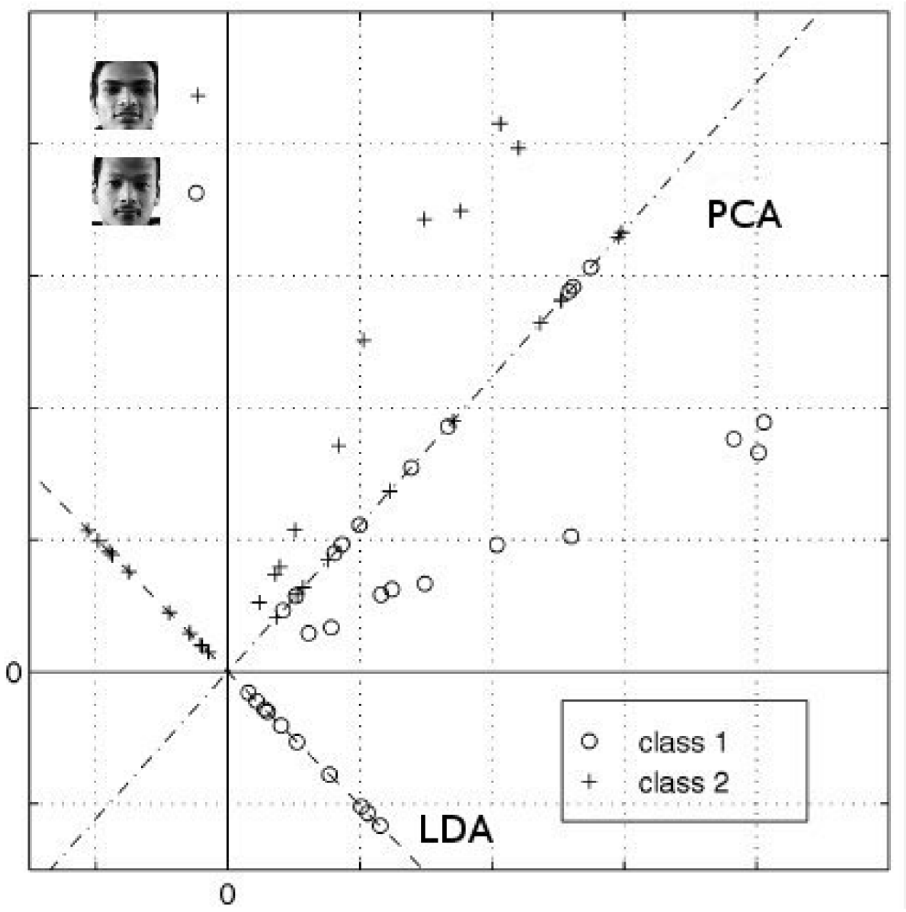


Figure 2.18: Difference in the space chosen by LDA and PCA

$$J(T) = \frac{|T^T \cdot S_b \cdot T|}{|T^T \cdot S_w \cdot T|} \tag{2.7}$$

where, S_b and S_w are between-class and within-class scatter matrix

Training operation in LDA

Let I be an image of size (N_x, N_y) pixels, then the training operation of PCA algorithm can be expressed in mathematical terms as

1. Convert the training image matrix I of size (N_x, N_y) pixels to the image vector Γ of size $(P \times 1)$ where $P = N_x \times N_y$ (ie: the train image vector Γ is constructed by stacking each column of train image matrix I)

2. Create a training set of training image vectors such that its size is $P \times M_t$ where M_t is the number of training images
   $\Gamma_{P \times M_t} = [\Gamma_1 \Gamma_2 \ldots \Gamma_{M_t}]$ where, Γ_i represents the image vector of i^{th} training images

3. Compute class mean face Ψ_{C_i} which is the arithmetic average of the training image vectors corresponding to the same individual at each pixel point for each class; and its size is $(P \times 1)$. In LDA, mean face images are also calculated for each face class; this is due to need for the calculation of each face classes inner variation. Hence, for each of C_L individuals having q_i training images in the database, the class mean face is given by.
   $\Psi_{C_i} = \frac{1}{q_i} \sum_{k=1}^{q_i} \Gamma_k \qquad i = 1, 2, \ldots, C_L$ (total no. of classes)

4. Compute arithmetic average (mean face Ψ) of the training image vectors at each pixel point given by:
   $\Psi_{P \times 1} = \dfrac{1}{M_t} \sum_{k=1}^{M_t} \Gamma_k$

5. Obtain mean subtracted vector (Φ) by subtracting the class mean face from the each training image vector of that class as given below:
   $\Phi_i = \Gamma_i - \Psi_{C_i}$

6. Compute within class scatter matrix S_w for C_L individuals each having q_i training images. S_w represents the average scatter Σ_i of the image vectors of different individuals C_i around their respective class means Ψ_{C_i}.

$$S_{w\,(P \times P)} = \sum_{i=1}^{C_L} P(C_i)\, \Sigma_i \tag{2.8}$$

   where, $P(C_i) = \dfrac{1}{C_L}$
   Average scatter $(\Sigma_i) = E[\Phi_i \cdot \Phi_i^T] = E\lfloor (\Gamma_i - \Psi_{C_i}) \cdot (\Gamma_i - \Psi_{C_i})^T \rfloor$

7. Compute the "between class scatter matrix" S_b which represents the scatter of each class mean Ψ_{C_i} around the overall mean vector Ψ.

$$S_{b\,(P\times P)} = \sum_{i=1}^{C_L} P(C_i)(\Psi_{C_i} - \Psi)\cdot(\Psi_{C_i} - \Psi)^T$$

The objective is to maximize J(T) (given in equation 2.7); in other words finding an optimal projection W which maximizes between-class scatter and minimizes within-class scatter.

$$W = arg\ \max_T (J(T)) \Rightarrow max(J(T)) = \left.\frac{|T^T \cdot S_b \cdot T|}{|T^T \cdot S_w \cdot T|}\right|_{T=W}$$

W can be obtained by solving the generalized eigenvalue problem

$S_b \cdot W = S_w \cdot W \cdot \lambda_w$

From the generalized eigenvalue equation, only $C_L - 1$ or less of the eigenvalues come out to be nonzero. Only the eigenvectors coming out with these nonzero eigenvalues can be used in forming the $W_{P\times(C_L-1)}$ matrix.

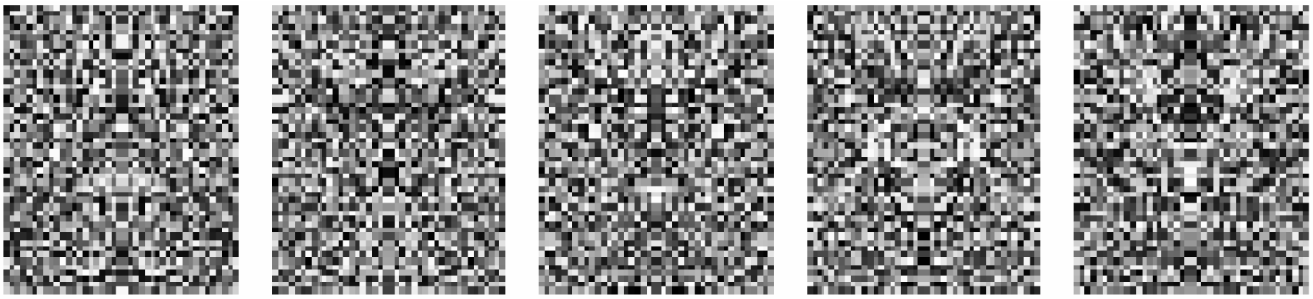


Figure 2.19: LDA bases

8. Project training images vector to the classification space by the dot product of the optimum projection W and the image vector as follows:
   $g(\Phi_i)_{(C_L-1)\times 1} = W^T \cdot \Phi_i \qquad i = 1, 2, \ldots, M_t$

Recognition operation in LDA

The test image vector (Γ_T) is projected to the classification space in a similar way

$g(\Phi_T)_{(C_L-1)\times 1} = W^T \cdot \Phi_T$

Finally, the distance between the projections is calculated by the Euclidean distance between the training and test classification space projections. Distance measure is given by:

$$d_{T_i} = \|g(\Phi_T) - g(\Phi_i)\| = \sqrt{\sum_{k=1}^{C_L-1}\left(g_k(\Phi_T) - g_k(\Phi_i)\right)^2} \qquad i = 1, 2, \ldots, M_t$$

The test image is assumed to be in the class whose distance is the minimal among all other class distances.

2.3.3 Subspace LDA

Subspace LDA is a hybrid algorithm. It uses both PCA and LDA. However only LDA is used as the ultimate classifier; PCA is only used as a dimension reduction step. So in

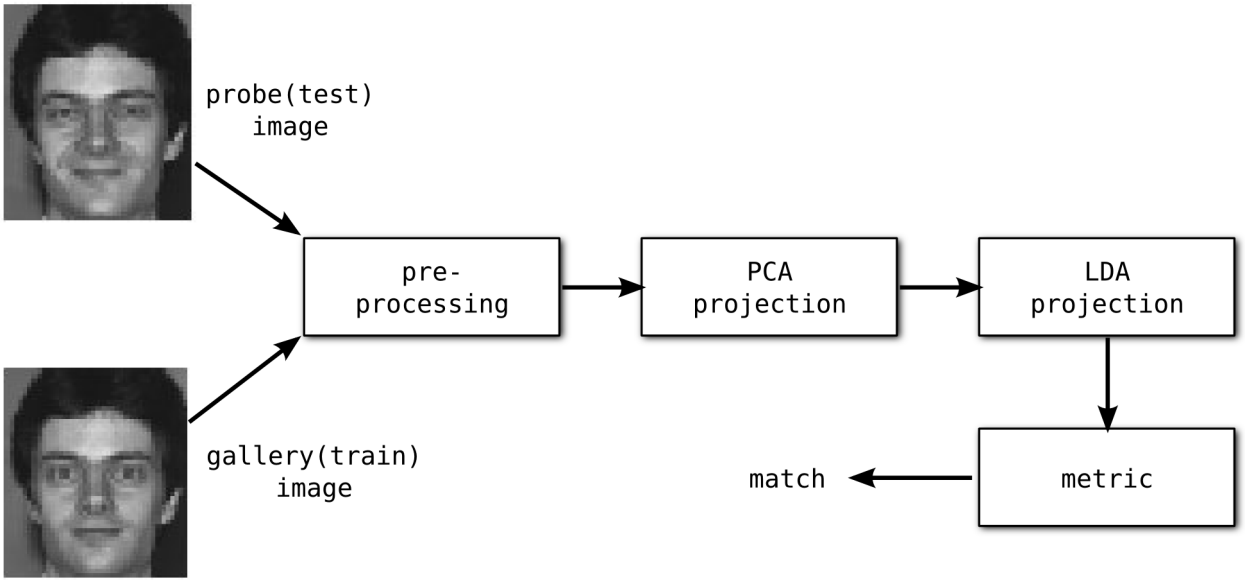


Figure 2.20: Basic block diagram of subspace LDA implementation for face recognition

subspace LDA algorithm we initially create a PCA subspace using the training images as described above in the PCA section. All the training images are then projected into the PCA subspace. These projections are then input to the LDA classifier. LDA again creates a new subspace from these PCA projections as described above in the LDA section. The PCA projections are again projected into the LDA subspace. For classification, a test face is projected first into the PCA subspace followed by LDA subspace. Then a suitable distance metric can be used to compare the distance between the individual projections and classify the test face.

Figure 2.21: Subspace LDA bases

2.4 Face Recognition using Gabor Wavelet Transform

This algorithm was proposed by Bruce Kepenekci in the thesis titled "FACE RECOGNITION USING GABOR WAVELET TRANSFORM"[5] published on Sep. 2001. We will use the alias "Kepenekci" to refer to this algorithm in this report. It presents a novel technique of automatic localization of facial features based on Gabor wavelet response of a facial image. Gabor wavelets are popular for their ability to approximate the response of simple cells in Human Visual System. The simple cells are tuned to orientation and spatial frequency for which they produce high response. This algorithm is more robust to illumination changes as Gabor wavelet transform of images (whose DC component is eliminated) is used for feature detection instead of directly using graylevel values of each pixel.

2.4.1 Introduction to Gabor Wavelets

Gabor wavelet (or Gabor filter) refers to a complex sinusoidal carrier enveloped by a Gaussian. Mathematically:

$$g(x, y) = w_r(x, y)s(x, y) \tag{2.9}$$

where, $w_r(x, y)$ = Gaussian envelop and $s(x, y)$ = complex sinusoidal carrier

Complex sinusoidal carrier - $s(x, y)$

The complex sinusoid is defined as

$$s(x, y) = \mathrm{e}^{\jmath\left(2\pi(u_0 x + v_0 y) + P\right)} \tag{2.10}$$

where,

u_0, v_0 are frequency of horizontal and vertical sinusoids respectively

P is an arbitrary phase shift

Gaussian envelop - $w_r(x, y)$

The Gaussian envelop is defined as

$$w_r(x, y) = K\,\mathrm{e}^{-\pi\left(a^2(x-x_0)_r^2 + b^2(y-y_0)_r^2\right)} \tag{2.11}$$

where,

K : scaling constant

(a, b) : envelop axis scaling constant

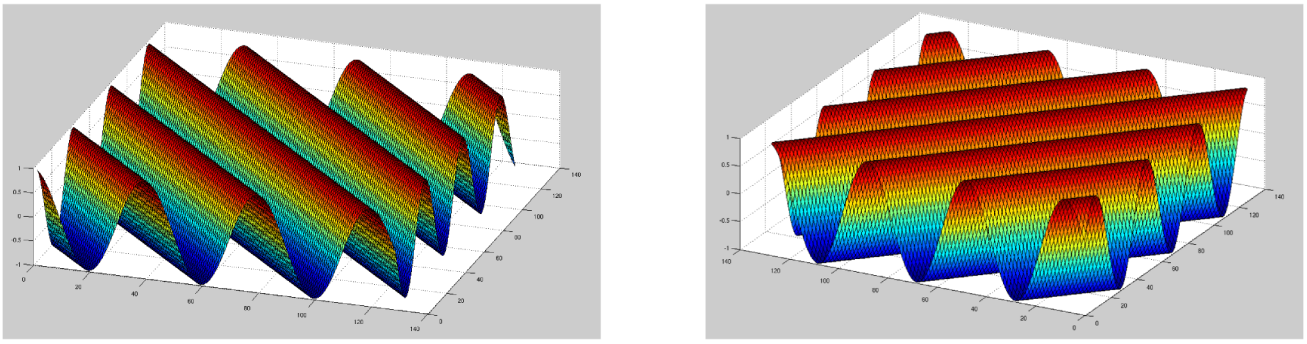
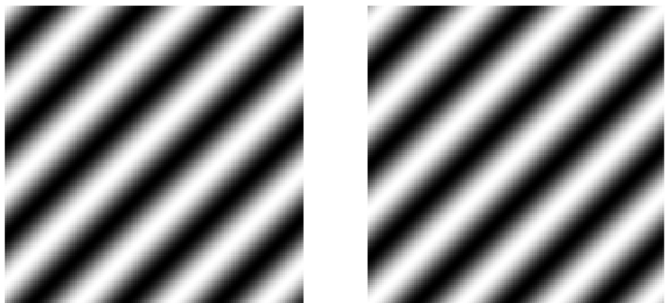


Figure 2.22: 3D visualization of complex sinusoid $s(x, y)$



(a) real part of $s(x, y)$ (b) imaginary part of $s(x, y)$

Figure 2.23: 2D gray level visualization of complex sinusoid $s(x, y)$ for $u_0 = v_0 = 1/40,\ P = 0,\ x = 1, 2 \ldots 128\ y = 1, 2 \ldots 128$

θ : envelop rotation constant
(x_0, y_0) : Gaussian envelop peak
$(x - x_0)_r = (x - x_0)\cos\theta + (y - y_0)\sin\theta$
$(y - y_0)_r = -(x - x_0)\sin\theta + (y - y_0)\cos\theta$

Gabor Wavelet - $g(x, y)$

Gabor wavelet, modeling the responses of simple cells in the primary visual cortex, are simply plane waves restricted by a Gaussian envelope function. Gabor wavelet obtained from by equation 2.9 is shown in Figure 2.25 and 2.26. Each complex Gabor wavelet consists of two functions in quadrature (out of phase by 90 °) conveniently located in real and imaginary parts of complex function.

Substituting equation 2.11 and 2.10 in 2.9, we get the equation for complex Gabor wavelet in spatial domain given by:

$$g(x, y) = K\, \mathrm{e}^{-\pi\left(a^2(x-x_0)_r^2 + b^2(y-y_0)_r^2\right)}\, \mathrm{e}^{\jmath\left(2\pi(u_0 x + v_0 y) + P\right)} \tag{2.12}$$

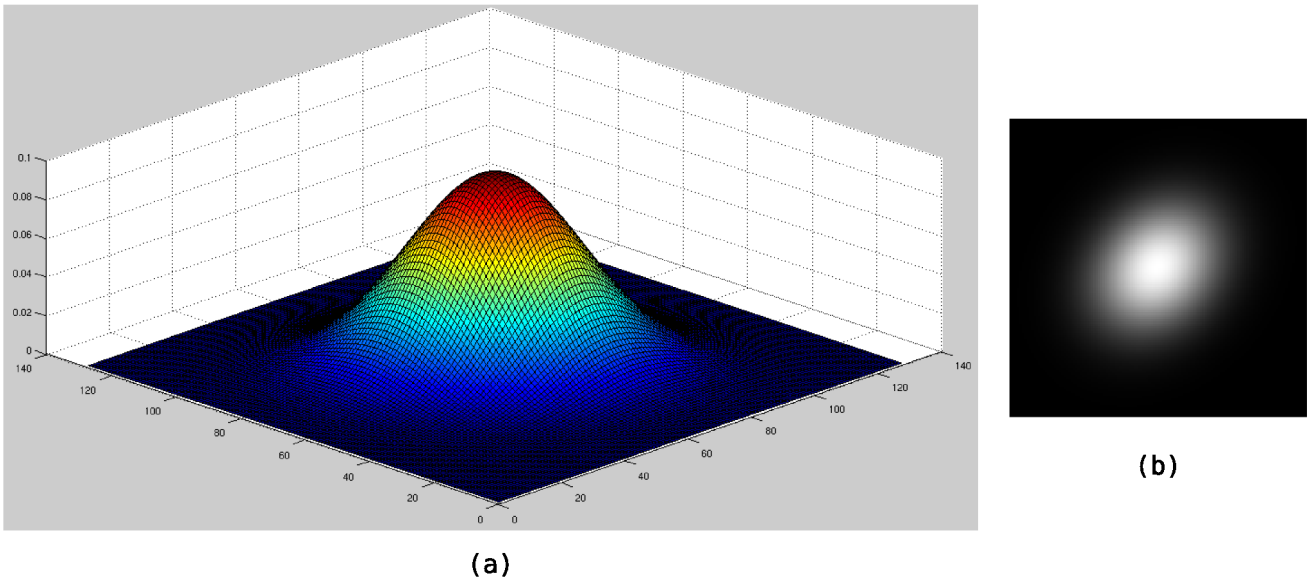


Figure 2.24: (a) 3D (b) 2D gray level - visualization of Gaussian $w_r(x, y)$ for $K = 0.1,\ a = 1/50, b = 1/40,\ \theta = \pi/4,\ (x_0, y_0) = (64, 64),\ x = 1, 2 \ldots 128\ y = 1, 2 \ldots 128$

where,

K : scaling constant

(a, b) : envelop axis scaling constant

θ : envelop rotation constant

(x_0, y_0) : Gaussian envelop peak

u_0, v_0 : frequency of horizontal and vertical sinusoids respectively

P : an arbitrary phase shift

If $a = b = \sigma$ then rotation angle $\theta = 0$ has no effect. Also if we express the spatial frequency of complex sinusoid in polar coordinate as magnitude F_0 and direction ω_0 and we restrict the magnitude of spatial frequency of sinusoid carrier F_0 to satisfy the equation

$$F_0 = \sqrt{u_0^2 + v_0^2} = \frac{\sigma^2}{\sqrt{2\pi}} \tag{2.13}$$

where, $u_0 = F_0 \cos\omega_0$, $v_0 = F_0 \sin\omega_0$ and σ = standard deviation of the Gaussian envelop

then the equation 2.12 reduces to (taking $P = 0$)

$$g(x, y) = K\, \mathrm{e}^{-\pi\sigma^2(x^2+y^2)}\, \mathrm{e}^{\jmath\left(2\pi F_0(x\cos\omega_0 + y\sin\omega_0)\right)} \tag{2.14}$$

The 2D Fourier transform of equation 2.12 is:

$$\hat{g}(u, v) = \frac{K}{ab}\, \mathrm{e}^{-\pi\left(\frac{(u-u_0)^2}{a^2} + \frac{(v-v_0)^2}{b^2}\right)}\, \mathrm{e}^{\jmath\left(2\pi\{x_0(u-u_0) + y_0(v-v_0)\}\right)} \tag{2.15}$$

An offset parameter will be introduced in the second exponential term of Equation 2.12 in next section to offset the DC component of Gabor wavelet[6].

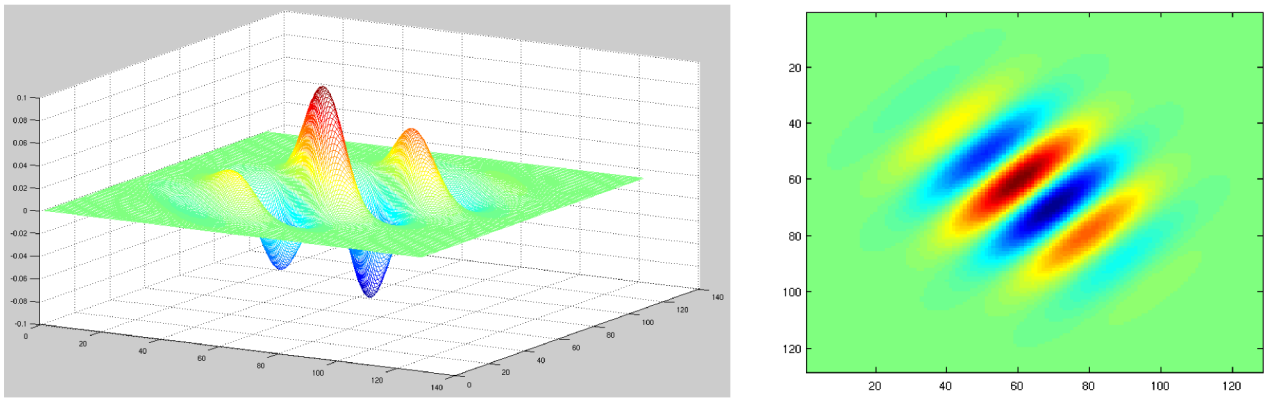


Figure 2.25: (a) 3D visualization of real part of $g(x, y)$, (b) Top view of 3D plot in (a)

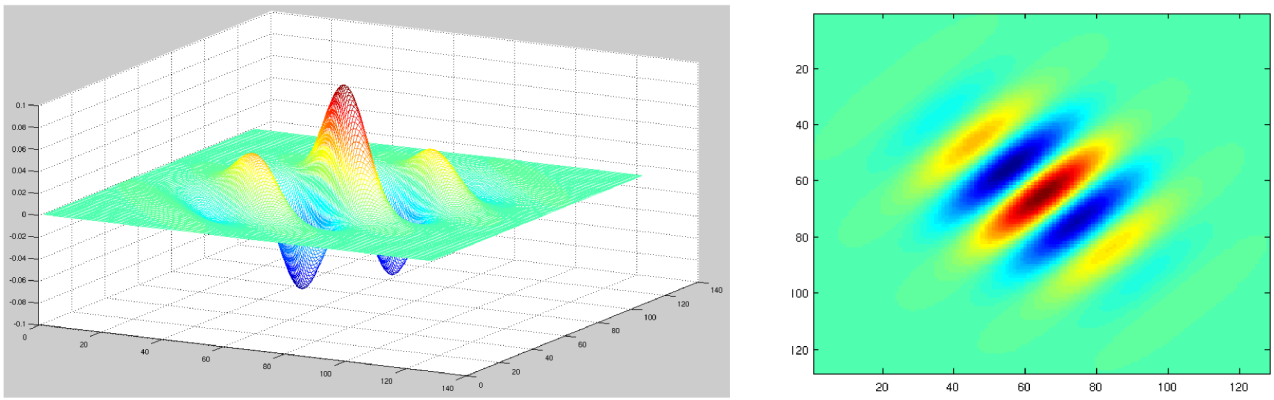


Figure 2.26: (a) 3D visualization of imaginary part of $g(x, y)$, (b) Top view of 3D plot in (a)

Making the Gabor wavelet illumination invariant

The Gabor wavelets will be used for detection of features (like eyes, nose, mole, dimple, etc) in grayscale images of human faces. Facial feature detection can be severely effected by variation in illumination and we may not want the Gabor wavelet to respond to the absolute intensity of an image. Hence we subtract the DC response from equation 2.12.

From the theory of Fourier analysis, we know that the DC response of a 2D signal $f(x, y)$ is given by the value of $F(0, 0)$ where $F(u, v)$ is the Fourier transform of $f(x, y)$.The DC response of 2D Gabor kernel is obtained from equation 2.15. Subtracting this complex constant from the complex sinusoid carrier of Equation 2.12 gives a 2D Gabor kernel, given in equation 2.16, that produces a response that is not effected by changes in illumination of a image[6].

$$g(x, y) = K\, \mathrm{e}^{-\pi\sigma^2(x^2+y^2)} \left(\mathrm{e}^{\jmath\left(2\pi F_0(x\cos\omega_0 + y\sin\omega_0)\right)} - \mathrm{e}^{-\left(\frac{\sigma^2}{2}\right)} \right) \tag{2.16}$$

2.4.2 Description of the algorithm

The algorithm proposed by Kepenekci has two stages:

Feature extraction This stage is responsible for automatic identification of facial features like eyes, nose, mouth, etc and generating the feature vector for each feature point. It selects peaks (high energized points) of Gabor wavelet response as feature points. Hence it is capable of identifying diverse facial characteristics of different faces such as dimples, moles, scars, etc. The feature extraction procedure is shown in Figure 2.27

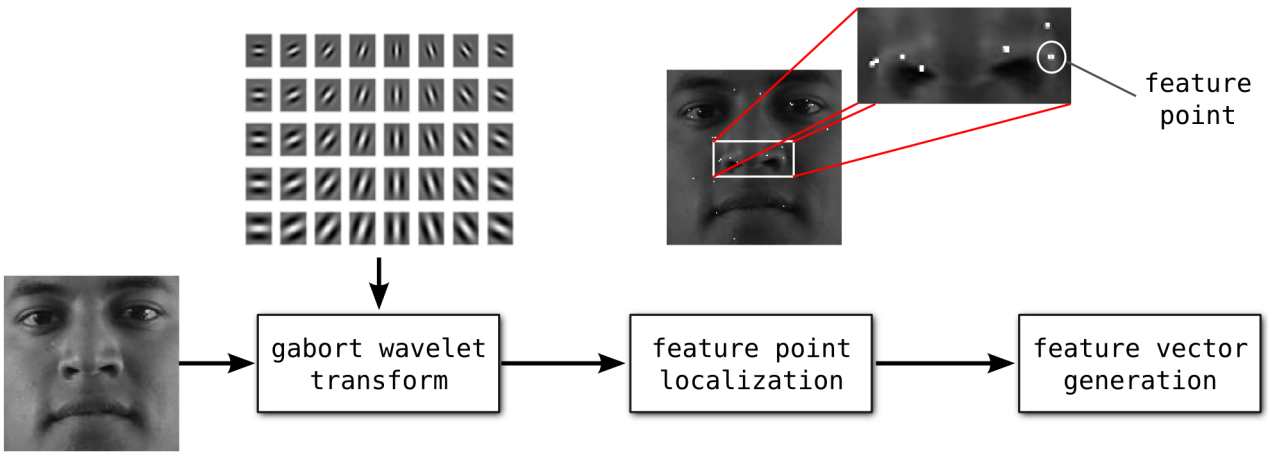


Figure 2.27: Block diagram representing the Feature extraction procedure

The function of each block in Figure 2.27 (the feature extraction procedure) will be explained in next sections

Feature matching Feature matching procedure involves comparing the features vectors of a probe(test) image to the feature vectors of training images. The feature vectors

(FV) of training images and probe image is obtained by Feature Extraction stage discussed in section 2.4.2. The feature matching procedure is shown in Figure 2.28

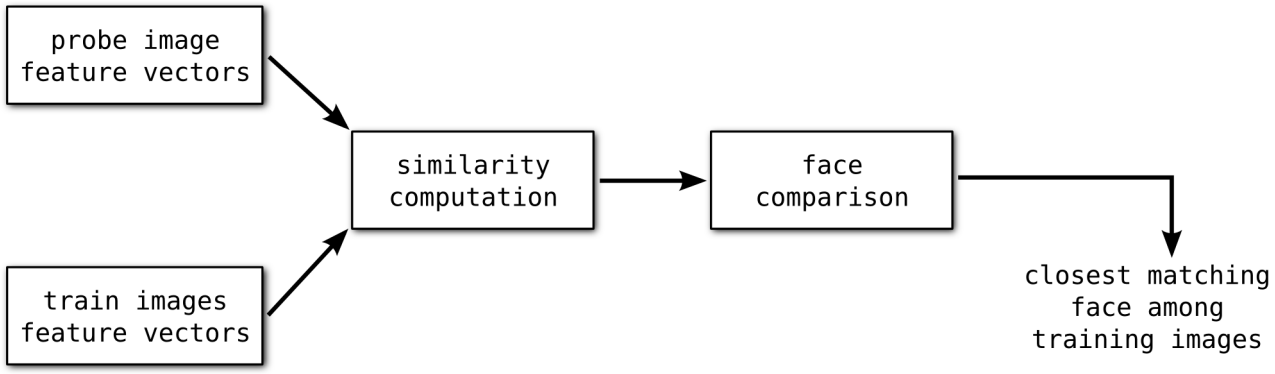


Figure 2.28: Block diagram representing the Feature matching procedure.

The function of each block in Figure 2.27 (the feature matching procedure) will be explained in next sections

The task of training is accomplished by Feature extraction stage which locates the feature points and stores the respective feature vectors. The task of face recognition involves the use of both stages. Training and Recognition operation can be visualized as a combination of these two stages as shown in Figure 2.29 and Figure 2.30 respectively.

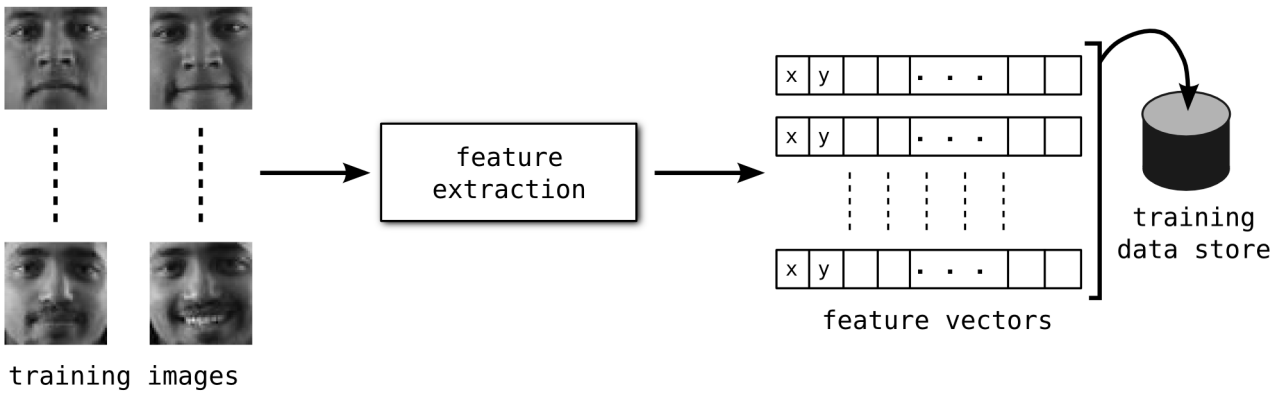


Figure 2.29: Training process involves use feature extraction stage

Gabor Wavelet Transform

A set of 40 Gabor filters is generated from mother Gabor wavelet by varying the magnitude of spatial frequency(F_0) and orientation(ω_0) in equation 2.14 as shown in Figure 2.31a. Convolution of an input image with this set of Gabor filter gives response as shown in Figure 2.32

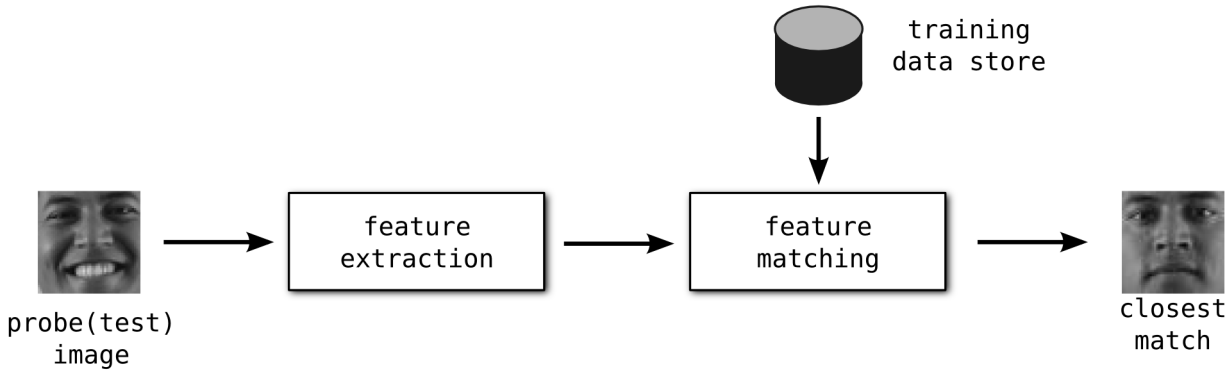
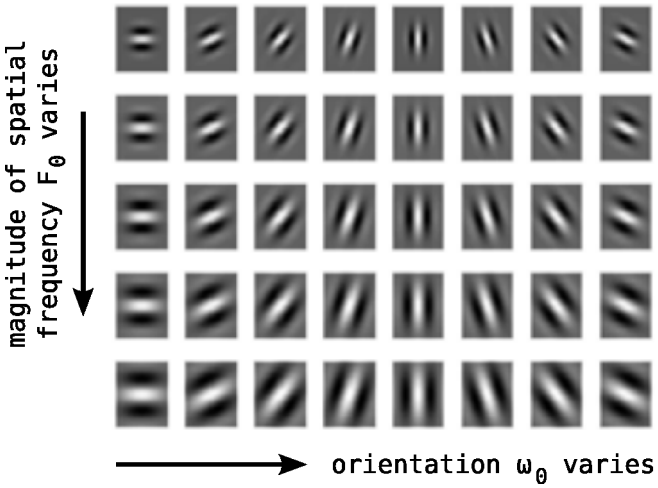
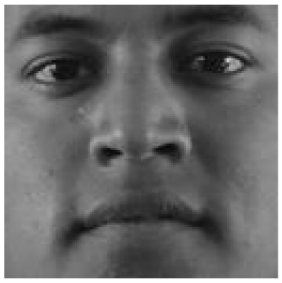


Figure 2.30: Recognition process involves use of both feature extraction and feature matching stages



(a) A set of 40 Gabor filters corresponding to 5 spatial frequency and 8 orientation



(b) Input face image

Figure 2.31: A set of 40 Gabor filters and a sample input image

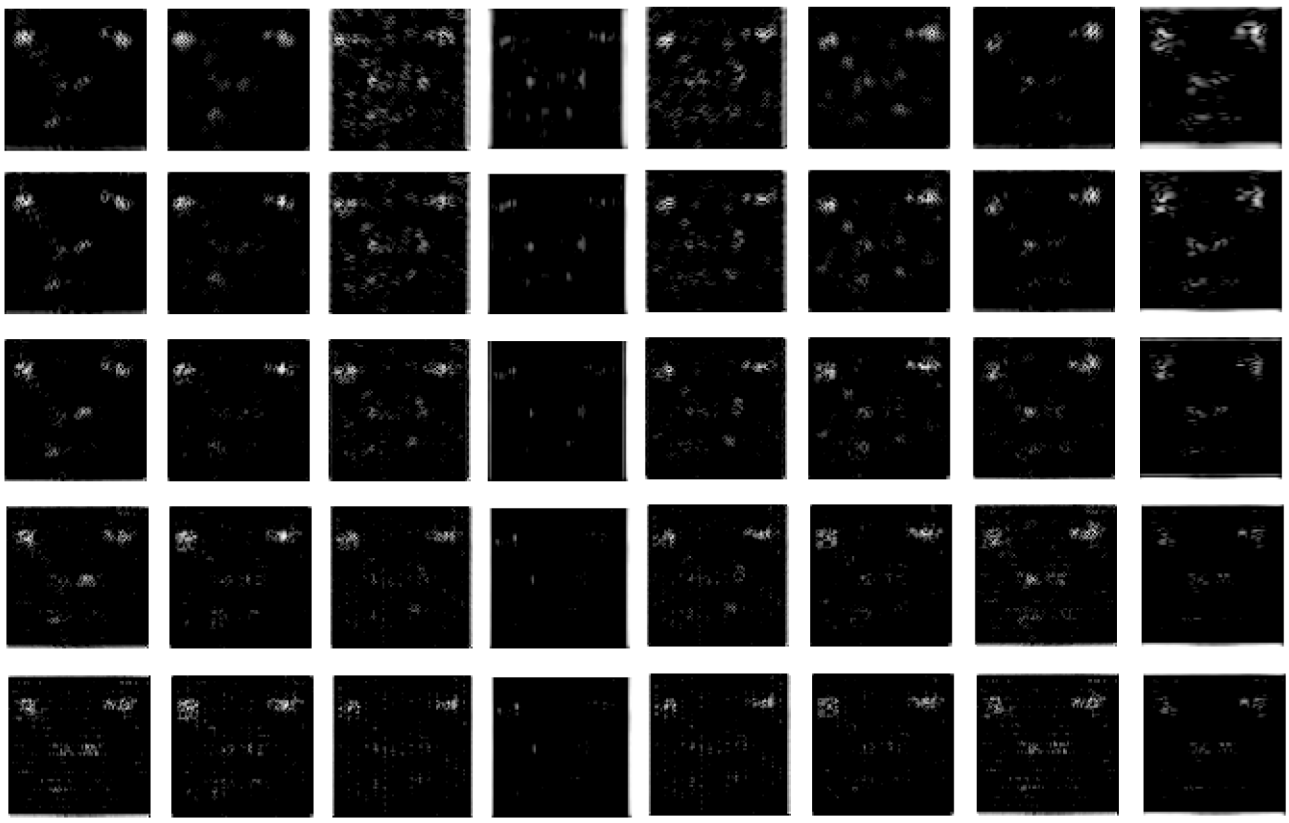


Figure 2.32: Response of gabor filters in Figure 2.31a when applied to images in Figure 2.31b

Feature point localization

From the responses (Figure 2.32) of face image to the Gabor filter, peaks (high energy points) are found by searching all the locations in a window W_0 of size $W \times W$ using the following procedure: Let,

$M \times N$ be the size of face image

(x_0, y_0) be the center of window W_0

R_j be the response of the face image to j^{th} Gabor filter (for $j = 1, 2, \ldots, 40$)

Then, a feature point is located at (x_0, y_0) , if

$$R_j(x_0, y_0) = \max_{(x,y) \in W_0} (R_j(x, y)) \tag{2.17}$$

$$R_j(x_0, y_0) > \frac{1}{MN} \sum_{x=1}^{M} \sum_{y=1}^{N} R_j(x, y) \tag{2.18}$$

for $j = 1, 2, \ldots, 40$

NOTE:

- Window size $W \times W$ is an important parameter in this algorithm and it must be chosen small enough to capture the important facial features and large enough to avoid redundancies.

- Equation 2.18 avoids getting stuck in a local maximum instead of finding the peaks of responses.

After applying the above procedure, we get a set feature points (depicting facial features) as shown in Figure

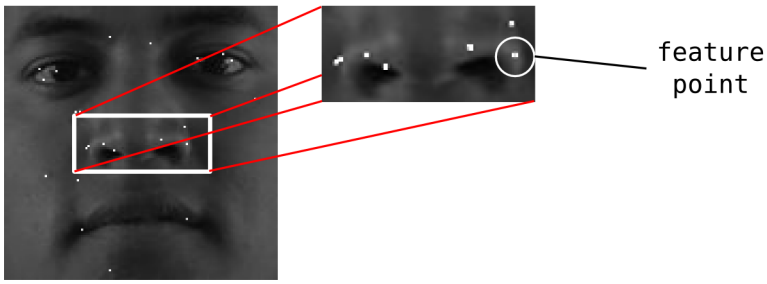


Figure 2.33: Result of feature point localization procedure applied to the Gabor filter responses of Figure 2.32

Feature vector generation

Feature vector corresponding to a feature point (determined by *Feature point localization* procedure) is stored in a user defined data structure whose first two elements is the (x,y) coordinate of the feature point's spatial location. The remaining 40 elements are the coefficients of each Gabor filter response at that spatial coordinate position (ie: (x, y)). The k^{th} feature vector of i^{th} face image is defined as:

$v_{i,k} = \{ x_k,\ y_k,\ R_{i,j}(x_k, y_k) \qquad \text{for } j = 1, 2, \ldots, 40\}$

where, $R_{i,j}(x_k, y_k)$ is the coefficient at (x_k, y_k) location of the response of j^{th} Gabor filter to i^{th} face image

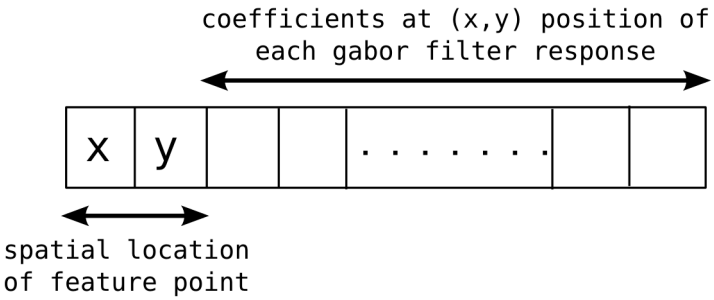


Figure 2.34: Data structure used for storage of feature vector corresponding to each feature point

Similarity Computation

The similarity of two complex valued feature vector is computed using the following similarity function (which ignores the phase):

$$S_i(k,j) = \frac{\sum_l |v_{i,k}(l)| \cdot |v_{t,j}(l)|}{\sqrt{\sum_l |v_{i,k}(l)|^2 \sum_l |v_{t,j}(l)|^2}}, \qquad l = 3, 4, \ldots, 42 \tag{2.19}$$

where,

l represents the index of feature vector array(of length 42)

$v_{i,k}$ is the k^{th} feature vector of i^{th} training image

$v_{t,j}$ is the j^{th} feature vector of t^{th} probe(test) image

$S_i(k,j)$ is the similarity of j^{th} feature vector of probe image to k^{th} feature vector of i^{th} training image

Equation 2.19 computes the angle ($\cos\theta = \frac{\vec{a}\cdot\vec{b}}{|\vec{a}|\,|\vec{b}|}$) between two complex feature vectors. Thus the value of S_i is bounded ($0 < S_i < 1$) as the value of $\cos\theta$ is bounded in that range. Also, if i^{th} training image is used as probe image:

$S_i(j,j) = 1$

Face Comparison

After the completion of Similarity Computation stage (discussed in section 2.4.2, the following data is available to the Face Comparison block:

- Feature vectors of all the training face images (ie: $v_{i,k}$ gives the k^{th} feature vector of i^{th} training image)
- Feature vectors of the probe (test) image (ie: $v_{t,j}$ gives the j^{th} feature vector of t^{th} probe image)
- $S_i(j,j)$ which gives the similarity of each feature vector of probe image with respect to the all the feature vectors of each training images.

For the Face Comparison block, we will define few quantities as given below:
OS_i = overall similarity such that it represents the similarity of probe image to i^{th} training images ($0.0 < OS_i < 1.0$)

The algorithm to determine the overall similarity OS_i (a value between 0.0 and 1.0 which gives a measure of similarity

1. Eliminate the feature vectors of training images that are not close enough to feature vectors of probe image in terms of location and similarity. The feature vectors that meet the two criterion, (a) and (b), given below should be examined in next step:

(a) $\sqrt{(x_r - x_t)^2 + (y_r - y_t)^2} < th_1$: $th1$ is the approximate radius of the area that contains either eye, mouth or nose. (x_r, y_r) and (x_t, y_t) represents the feature vector location of training image and probe image respectively.

This criteria ensures that feature vectors in a particular region of the probe image are compared to feature vectors located in the same region of training images (ie: feature vectors near eyes are only compared with feature vectors near the eye region and not with the feature vectors in mouth region of training images.

(b) $S_i(k, j) > th_2$: $th2$ is the standard deviation of similarities of all feature vectors of training images. This condition helps reduce the redundancies in feature vectors that are being considered.

2. After completion of Step 1, we are left with a set(can be an empty set) of training image feature vectors ($N_{k,j}$) for each j^{th} feature vector of probe image. From each such set ($N_{k,j}$), we obtain a feature vector that has maximum similarity (S_i) with respect to corresponding feature vector of probe image.

$$Sim_{i,j} = \max_{l \in N_{k,j}} (S_i(l, j)) \tag{2.20}$$

where, $Sim_{i,j}$ gives the similarity of i^{th} training image to probe image based on j^{th} feature vector of probe image.

3. The overall similarity (OS_i = the similarity of probe image to i^{th} training images) is computed as the mean of all the similarities of i^{th} training images that passed the tests of step 1 and 2.

$$OS_i = mean\{Sim_{i,j}\} \tag{2.21}$$

The training image for which the value of OS_i is maximum is the face image having closest match with the probe image.

Chapter 3: Performance and Analysis

In this chapter we will discuss about the performance of C++ implementation of RTFTR. The two face recognition modules were tested on some standard face databases and the performance of the two face extraction modules was tested on a video shot by webcam at 640 × 480 resolution. Testing was performed on a Dell laptop with the following specifications :

- Processor : Intel(R) Core(TM)2 Duo CPU T7250 @ 2.00GHz
- Memory : 2GB
- Operating System : Ubuntu 8.10 / Fedora Core 8
- C++ compiler : g++ (GCC) 4.1.2

The nature of several parameters were studied in order to assess the performance of RTFTR when it is implemented and used in a real world scenario. All the performance related data presented here relate to natural lighting condition, almost no facial occlusion, large amount of facial expression variation and very small out of plane rotation (10^o) face. The following aliases have been used to refer to each algorithm in all the performance plots and description:

- Neural Network Based Face Extraction - **rowleynn**
- Face Extraction using AdaBoost and Cascaded detector– **adaboost**
- Face Recognition using Subspace LDA - **lda**
- Face Recognition using Gabor Wavelet Transform - **kepenekci**

We used OpenCV's cvGetTickCount() and cvGetTickFrequency() methods to measure the time elapsed while performing a task. Typical code used for this purpose is given below:

```
int64 t1 = cvGetTickCount();
// Perform the task whose CPU time is to be measured
int64 t2 = cvGetTickCount();
double time_elapsed = (t2-t1)/(cvGetTickFrequency()*1e06);  // in seconds
```

The "number of actual faces" count, for each video frame, used in section 3.2 was fed manually by counting number of faces in each frame. Same video file was used to test the performace of both face extraction modules (adaboost and rowleynn).

3.1.1 Improvements in the training algorithm for cascaded detector

The training of the cascade (as discussed in section 2.2.4) takes a very long time to complete. We have made mainly four modifications in the algorithm to speed up the training process:

- instead of starting from one classifier per stage and adding one each time the detection and false positive rate are not met, we have specified the number of classifier to start with and the number to add each time the condition is not met. This is specified for all the stages, and few number of classifiers are defined for the initial stages and larger for the later stages.

- instead of training the stage with AdaBoost from the initial condition after the addition of the classifier, we have made modifications so that the training continues from the new classifiers. This is done by saving the weights of the examples for the final round of boosting. If the new classifiers have to be added, these weights are retrieved and the training continues.

- the modification has been done such that if there is any problem which ceases the training operation, the training starts from the last saved period and does not need to be restarted from the beginning.

- because the training operation takes a long time (more than a week), OpenMP [1] has been used to parallelize the training code. For each of the five feature types, five threads have been created and individual thread calculates the feature value, threshold and error of classification of each feature type in parallel. Before the use of OpenMP, only one of the two cores of the processor was used for the training and after the use of OpenMP, both the cores were used, which improved in the training speed of the algorithm.

3.1.2 Training dataset

For the training of the face detector, faces and non faces of size 24×24 were used. Total features on this size of sub-window is $162,336$. The face and non-face images used by Voila and Jones in their training were collected for the training purpose. There were more than 4000 faces and more than 7000 non-face images. For the training purpose, 500 of the faces were used and around 1200 non-face images were used initially. For the validation, 2000 faces and 2000 non-faces different than that used in training were used.

[1] OpenMP is an API that supports multi-platform shared-memory parallel programming in C/C++. `http://www.openmp.org`

Table 3.1: Detection time of Cascaded AdaBoost Detector for various parameters

S.N.	Starting scale	Scale factor	Average Detection Time (secs)
1	1	1.1	0.36
2	1	1.25	0.17
3	1.25	1.1	0.22
4	1.25	1.25	0.11
5	1.5	1.25	0.055
6	2	1.25	0.027

The approximate training time for each classifier selection for these number of training and validation images was one hour. Stage one has three classifiers, so the training time for stage one was around 3 hours. Similarly, the classifiers selected for the higher stages are $3, 2, 4, 6, 12, 12, 10, 10, 26, 58$ and 22 up to stage 11.

The non-face source images which were used to generate non-faces for the higher stages were collected at random and also contained the environment in which the detector would be run. The use of the non-face source image containing the environment on which the face detector would be run helps to decrease the false-positive rate significantly.

3.1.3 Implementation of the detector

The frames from the webcam were scaled down to 256×192 resolution before the detection process. The scaling factor increases the size of the scanning subwindow and is used to detect the faces in the images at different scales. The step was used to scan the whole image for the faces. For the detection, the step of $[\delta \times s]$ was used where δ is the initial step size, s is the current scaling factor and [] is the rounding operation. The detection speed can be increased by the scaling factor and the step size, but the detection rate decreases in doing so. The detections time for the frames of size 256 x 192 with initial starting step size of 1.5 and varying other parameters is shown in table 3.1. There are many detections of the same face due to the detection at multiple scales. This was removed by clustering the multiple detections into one, which also helped in reducing the false detections.

3.1.4 Results of detection

The detection on the webcam images are better than the images selected at random because the non-face source images used for the training was of the environment on which the detector would be run. The detections of other images are good for the closeup images, but for other images there can be many false detections. This can be improved by training the detector on a larger training set and training the detector for larger stage. Also, for some other images there can be no detections at all on a face containing images. This was because the minimum detection rate per stage was set to a lower value ($d = 0.96$). This can be improved by setting the minimum detection rate per stage very close to one (say $d = 0.99$).

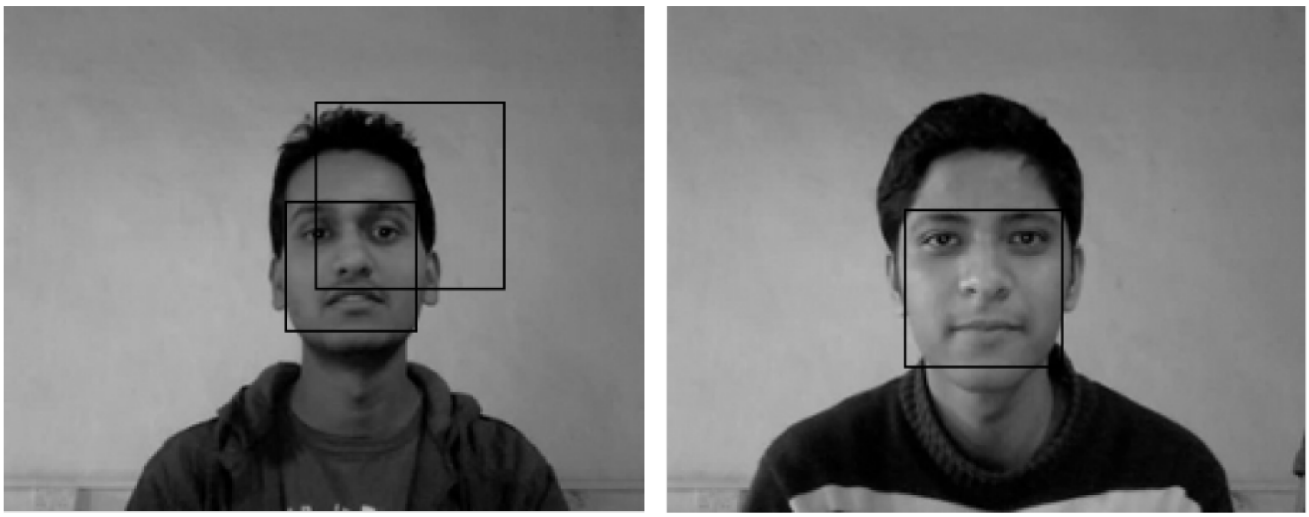


Figure 3.1: Result of detection procedure applied to the webcam images

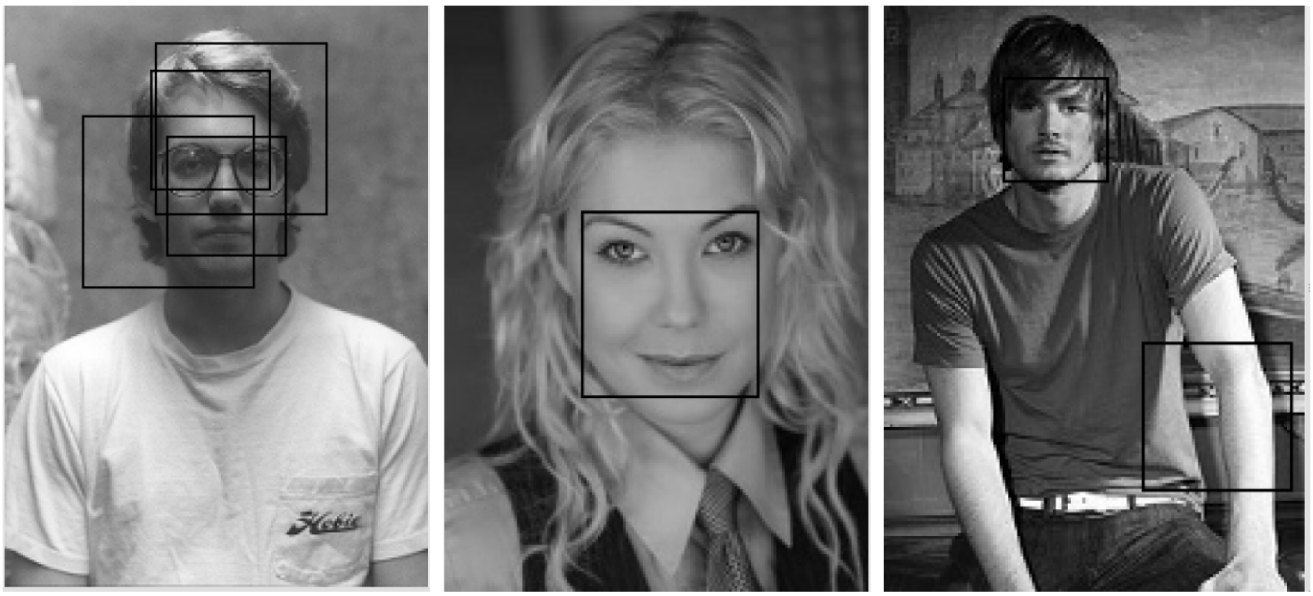


Figure 3.2: Result of detection procedure applied to some random images

In this project, for the generation of the cascade, the minimum detection rate per stage, d was taken to be 0.96 and the maximum false detection per stage, f was taken to be 0.6. Selection of low value of d caused the detector to skip some of the faces in the test image. Cascaded detector was generated up to 11 stages containing 140 features. Because there are very small number of stages and the features used, the number of false detections in the test images is higher. To achieve the comparable false detection rate with the Viola and Jones detector (containing 6061 features), the training time required by our cascade would be approximately 6061 hours (36 weeks) because the training code is not fully optimized. Training for this long time was not possible due to time constraint. So we fixed the environment on which we would be running our face detector and performed the training according to that.

Different image frames from the video of same size (256x192) with different count of faces were fed to the detectors. The number of faces were manually counted. Both the face extraction modules (rowleynn and adaboost) were fed the same video frame for the generation of plots. The performance was measured in several ways described below.

3.2.1 number of faces in a frame v/s face extraction time

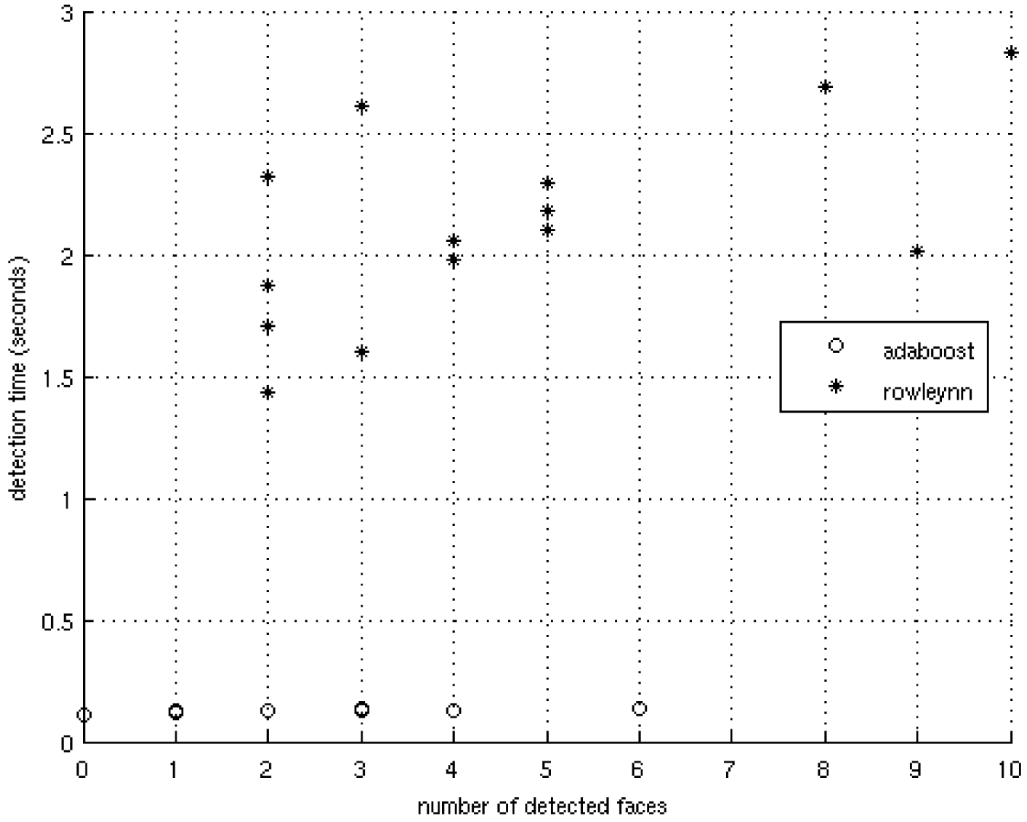


Figure 3.3: Plot of number of faces detected by adaboost and rowleynn modules v/s the corresponding detection time

The plot of number of faces detected by adaboost and rowleynn modules and the corresponding detection time is shown in Figure 3.3. The extraction time is independent of the number of faces in a frame, both for the neural network based detector and the AdaBoost detector. This is because most of the detection time is taken in scanning the whole image with subwindow and processing it, and the time taken to store a window as a face is very less in comparison to the total detection time.

3.2.2 actual/reported number of faces v/s frame-id

For the AdaBoost detector, as shown in the Figure 3.5 the detection rate was higher for the video frames in which the face of the subject was directly facing the camera and

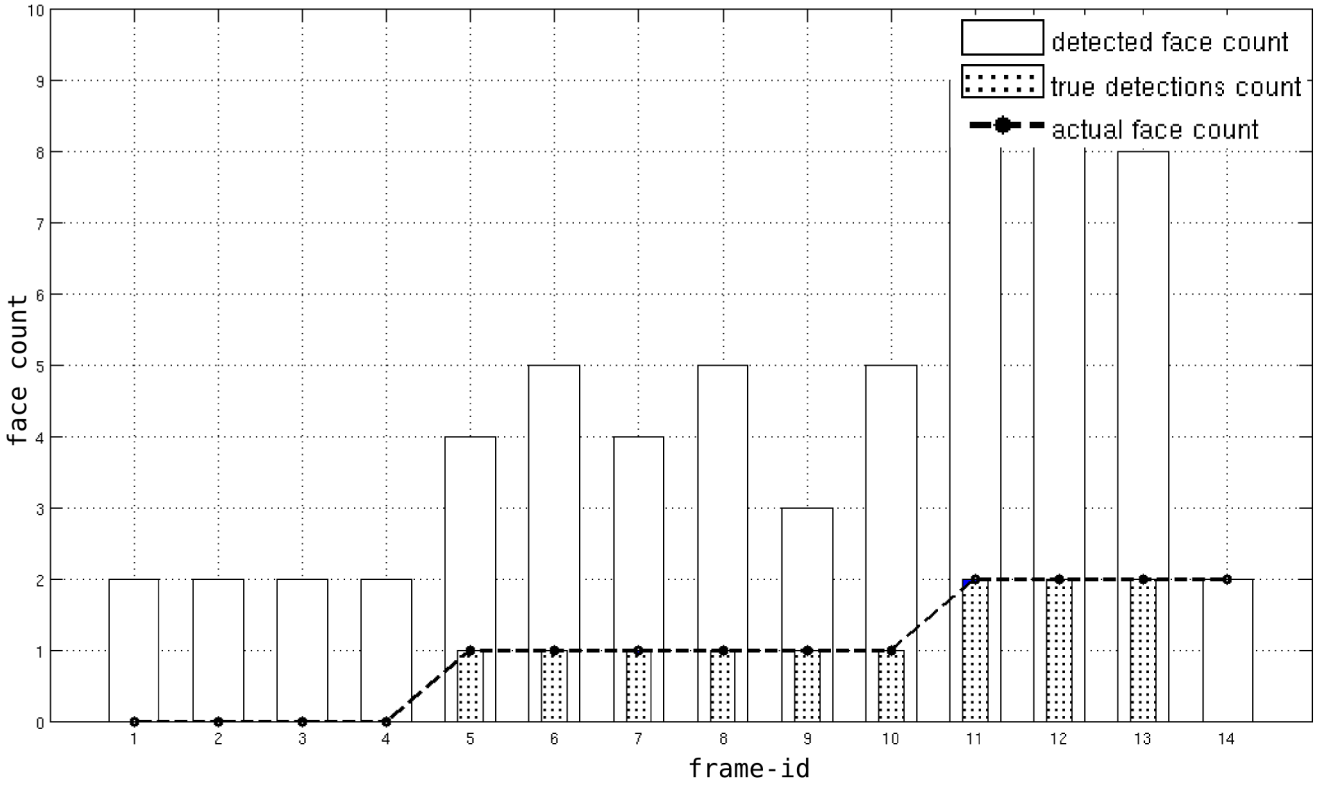


Figure 3.4: Plot for actual and detected number of faces by rowleynn module in different frames of a test video

when the video frame contained only one subject. There are higher false detections in many video frames because the cascade was trained only up to 11 stages(due to long time required for the training). So, for reducing the number of false positives, the detector has to be trained to higher stages.

For the neural network detector, the detection rate and the number of false detection is high. However the implementation in [7] has appreciable level of accuracy. The reason for high number of false detections and its improvements are:

- The number of false detections can be reduced by training the neural network with large number of face images and a comparable number of non face images. The number of face images used for the training, in our implementation, was around 500 and the non face images was 6000 images. Hence our implementation could not realize high level of accuracy.

- The training faces features were roughly aligned and the detection rate of faces can be improved with proper alignment of the features.

3.2.3 Training time

For rowleynn, training time was 40 minutes with an acceptable error rate of 0.003 (0.3%) for 500 faces and 6000 non faces. For AdaBoost cascade, the training time for 11 stages(total of 140 classifiers selected by AdaBoost) for 500 faces and 1200 non faces

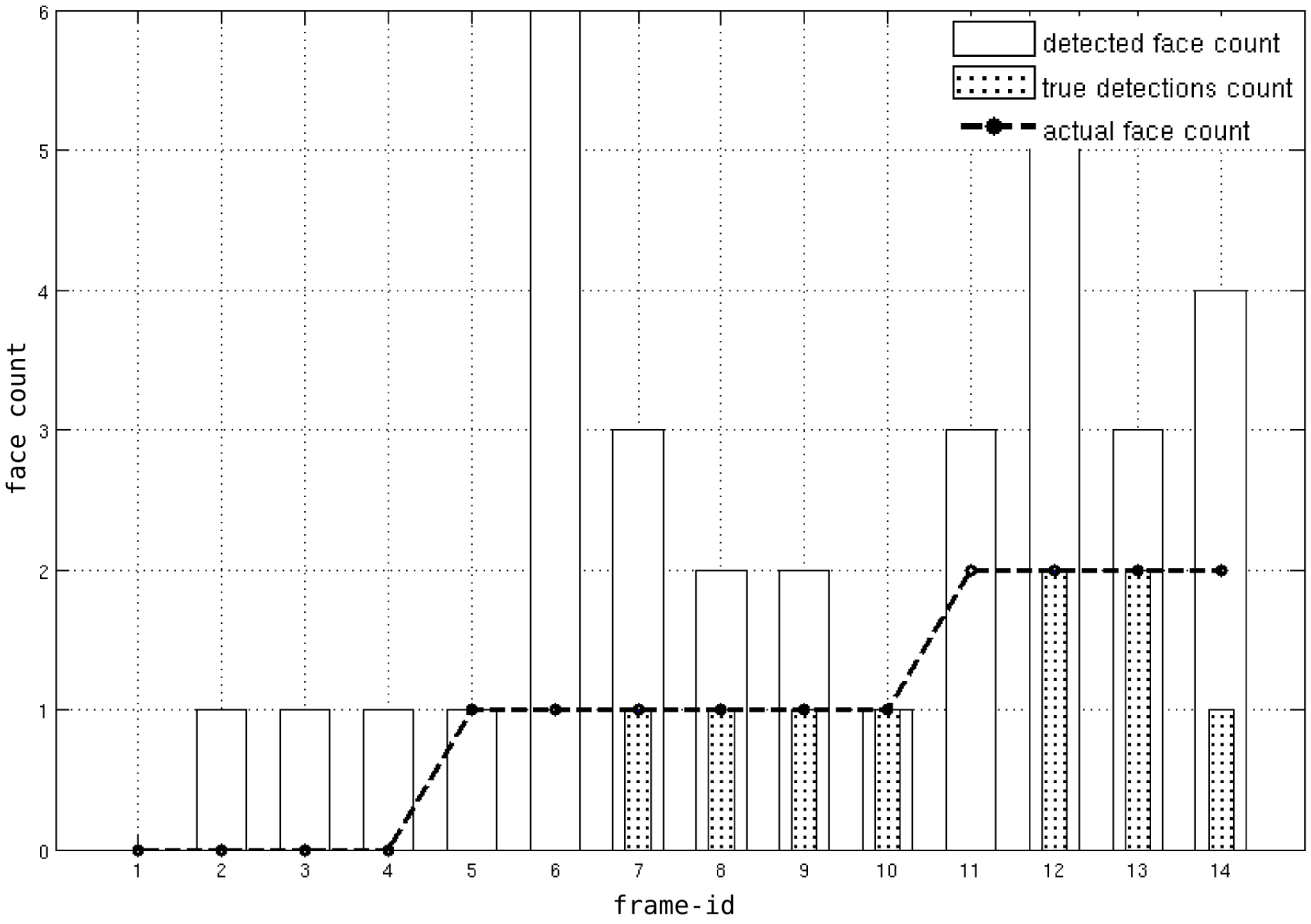


Figure 3.5: Plot for actual and detected number of faces by adaboost module in different frames of a test video

was around 140 hours. The detection rate and false detection rate up to stage 11 is 0.7797 and 0.00825 respectively

It contains 165 grayscale images in GIF format of 15 individuals. There are 11 images per subject, one per different facial expression or configuration: center-light, w/glasses, happy, left-light, w/no glasses, normal, right-light, sad, sleepy, surprised, and wink. However all the faces are frontal and there is no out of plane rotation. The ground truths of these faces were manually obtained. Using the ground truth, the images were geometrically normalized or registered to a template size of 60x60 pixels with the eyes at the specified location in the template size. Further some preprocessing techniques like median filtering and histogram equalization was performed to remove the noise and reduce the illumination variation effects. Out of the available images per person, five of them were used to train the system while some of the rest were used for testing purpose.

Figure 3.6: Yale Face Database snapshot

3.3.1 Performance of Subspace LDA algorithm on Yale Face Database

The performance of Subspace LDA was compared using different distance metrics like Cosine and Euclidean. Further it was tested by dropping different number of eigen vectors. The best result obtained was correct matches of around 85% with false matches of around

15%. The details of the results can be depicted in Figure 3.7.

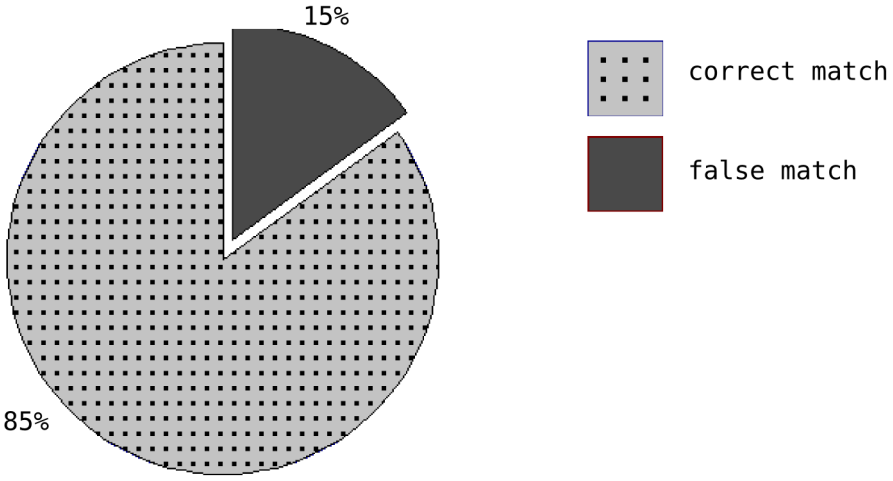


Figure 3.7: Performance of subspace LDA on Yale Face Database

Performance of Kepenekci algorithm on Yale Face Database

The training and probe image's size in Yale Face Database is 60×60 pixels. Due to this low resolution, we observe an 18% false match as shown in Figure 3.8. We also observed that the probe images that resulted in false match, for kepenekci algorithm, were successfully recognized by subspace LDA algorithm. Also the probe images that resulted in false match, for subspace LDA algorithm, were successfully recognized by kepenekci algorithm. This observation can be attributed to the fact that subspace LDA and kepenekci algorithms are holistic and feature based approach respectively. Hence, it proves that use of using hybrid approach we can cover larger portion of the total face space.

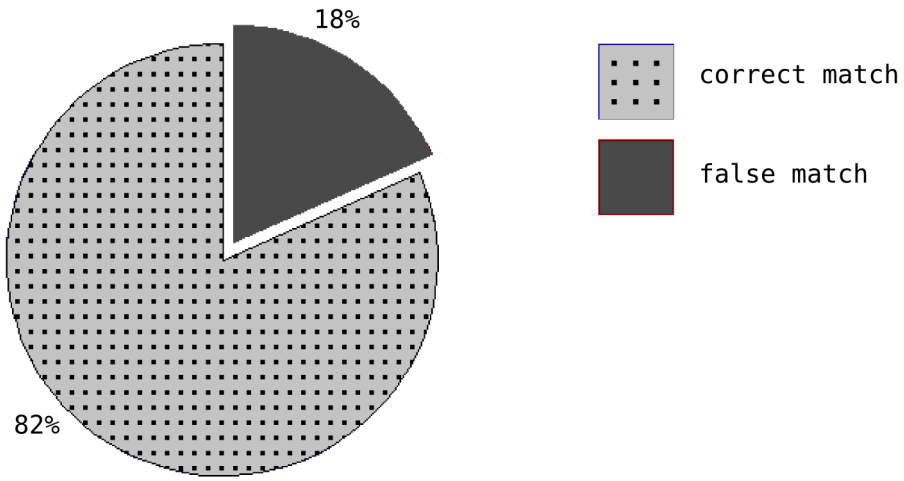


Figure 3.8: Plot showing the performance of kepenekci algorithm when tested using Yale Face Database

3.4 Performance of Face Recognition modules on IOE Face Database

The training and probe (test) images, used to gather performance data, is given in Appendix C. A 2 minute video containing the faces of four developers of RTFTR was captured using a webcam at 640 × 480 resolution. OpenCV's face detector module was used to extract faces from this video file. We used these extracted faces for training and probe image set as shown in Appendix C. We used this process (instead of using standard digital camera to capture images) because we wanted to test the performance in a situation that would arise after integrating face extraction (adaboost and rowleynn) and face recognition modules.

3.4.1 recognition rate (correct/false matches)

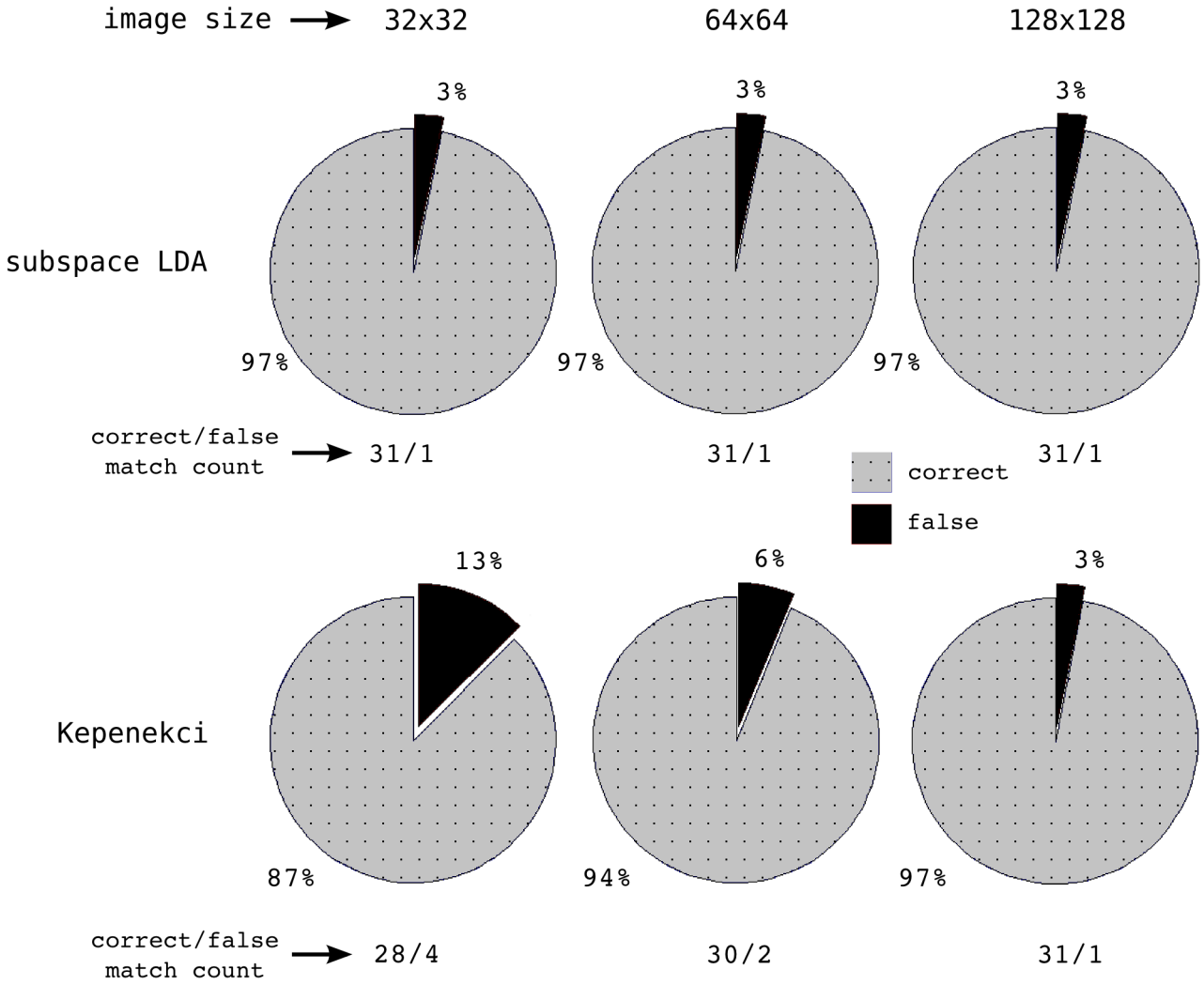


Figure 3.9: Recognition rate for subspace LDA and Kepenekci modules for train/probe images of three different sizes (32x32, 64x64, 128x128)

Subspace LDA, being a holistic approach, is not affected by the resolution of probe images. It can recognize faces as long as the overall facial structure is preserved. This is

a remarkable property of Subspace LDA which can be used to recognize faces in a video stream which usually contains faces of small size present at low resolution. It has been tested in very low resolution images of size 24x24 pixels [12]

On the other hand the performance of recognition of Kepenekci algorithm, a feature based approach to face recognition, increases with higher resolution train/probe face images. Higher resolution images allows Kepenekci algorithm to detect fine facial features (like scar, mole, dimple, etc) and use these features during the comparison process. Large number of features in train/probe images ensures higher accuracy in the recognition process. A drastic performance improvement was observed when a large number of training images ($\sim$ 20 train images per person) was used.

However, the performance of Subspace LDA algorithm degrades when there is considerable illumination variation between the training and probe face images or when profile faces (side looking faces) are present in the probe set. For kepenekci algorithm, the performance depends on the choice of combination (obtained from several test run) of the window (W_0) size $W \times W$ and the frequency of complex sinusoid (ω_0 in equation 2.16 which is used to form the Gabor Wavelets). As no quantitative method to determine the optimal combination of $W \times W$ and ω_0 exists, optimal performance cannot be achieved easily.

The result of recognition rate performance test performed on C++ modules implementing Subspace LDA and Kepenekci algorithm is shown in Figure 3.9.

3.4.2 probe face-id v/s recognition time

Recognition time is the most important parameter for realtime operation of RTFTR. The result of recognition time performance test performed on C++ modules implementing Subspace LDA algorithm is shown in Figure 3.10. It is evident from the plot that the recognition time varies from 0.1 seconds to 1.4 seconds for images of three different size. As Subspace LDA can detect faces with high accuracy even at low resolution (32×32), we can conclude that it can be used for realtime operation with appreciable level of accuracy. This performance can be attributed to the fact that the Subspace LDA bases and the projections generated during the training phase is reused during the recognition phase. Any distance metric, Euclidean or Cosine, can then be used to compare the distance or similarity between the probe and training images.

For the Kepenekci algorithm, the recognition time varies from 0.01 seconds to 0.5 seconds for images of three different sizes shown in Figure 3.11. *This level of performance in recognition time was obtained after the OpenMPI based optimizations in the Kepenekci module.* As Kepenekci algorithm performs better only at high resolution, we have to consider the recognition time of $\sim$ 0.1 seconds (for intermediate image size of 64×64) for appreciable level of accuracy. Hence the realtime operation of Kepenekci algorithm is only possible if the number of faces to be processed per second is very low ($\sim$ 10 faces

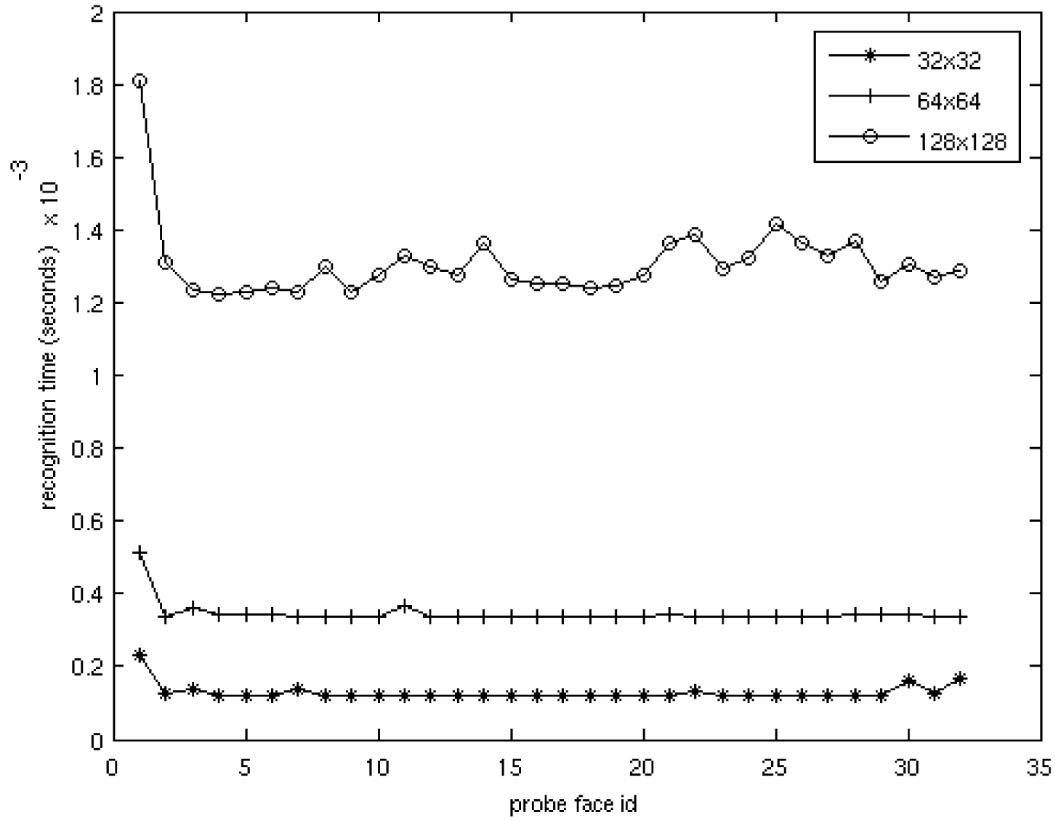


Figure 3.10: Recognition time for subspace LDA module for train/probe images of three different sizes (32x32, 64x64, 128x128)

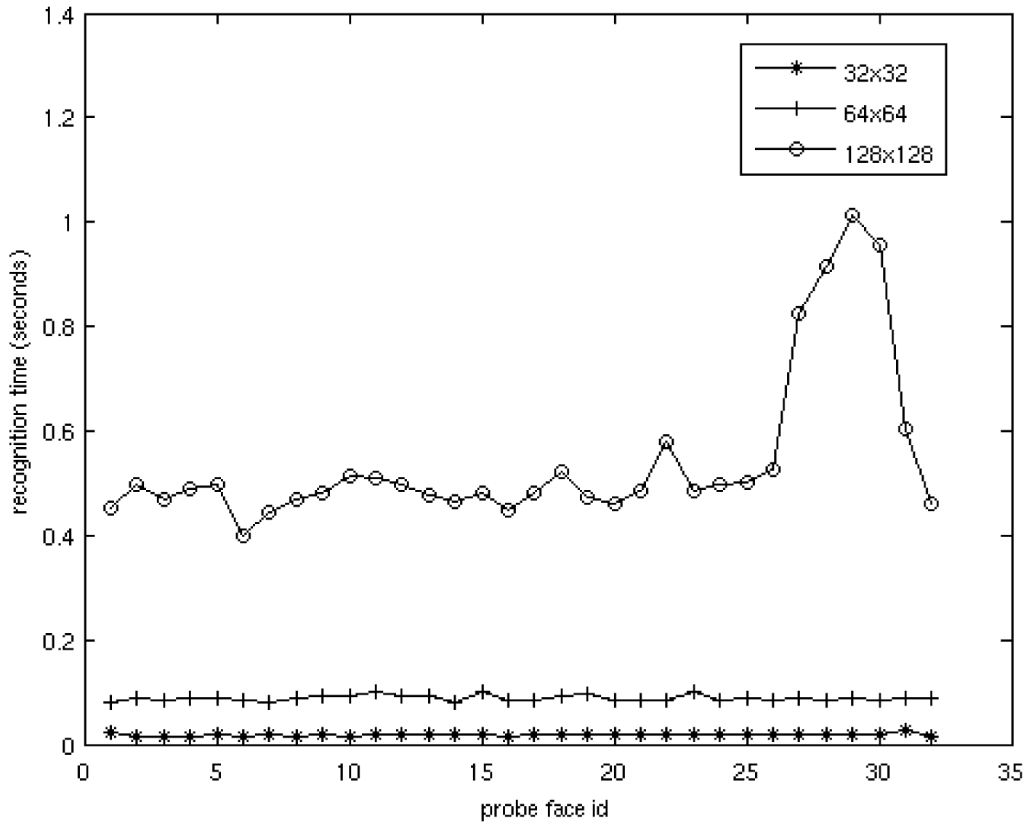


Figure 3.11: Recognition time for kepenekci module for train/probe images of three different sizes (32x32, 64x64, 128x128)

/ second). The computational complexity of Kepenekci algorithm is high due to the fact that it compares a large number of features of a probe image with the features of all the training images. Moreover, the recognition time varies with the amount of feature vectors in the probe image and the training images as shown by increase in recognition time for probe face-id 27 to 32.

Hence Kepenekci algorithm can be used in unison with Subspace LDA algorithm to process only those probe images whose distance, as computed by Subspace LDA algorithm, from each face class is nearly same. In such situation Kepenekci algorithm can be used to resolve this ambiguous situation, of Subspace LDA algorithm, and recognize faces with more accuracy. In that case, the image size should be neither be very low resolution nor be very high resolution (an intermediate resolution of 64×64).

3.4.3 training time v/s number of training images

Although the training time of a face recognition algorithm is not a crucial factor for real time operation it becomes an important factor to consider when the amount of training data is very large (which is usually the case for real world applications). The total training time for Subspace LDA and Kepenekci algorithm is shown in Figure 3.12 and for Kepenekci modules it is shown in Figure 3.13.

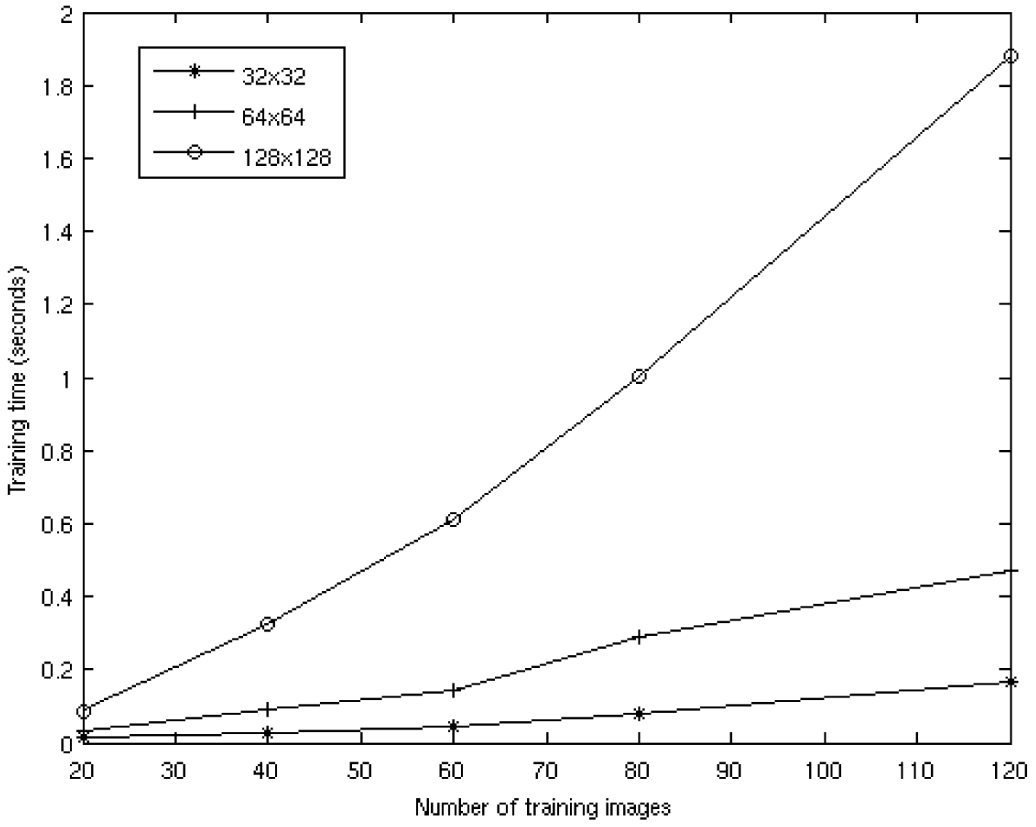


Figure 3.12: Training time for subspace LDA module for train/probe images of three different sizes (32x32, 64x64, 128x128)

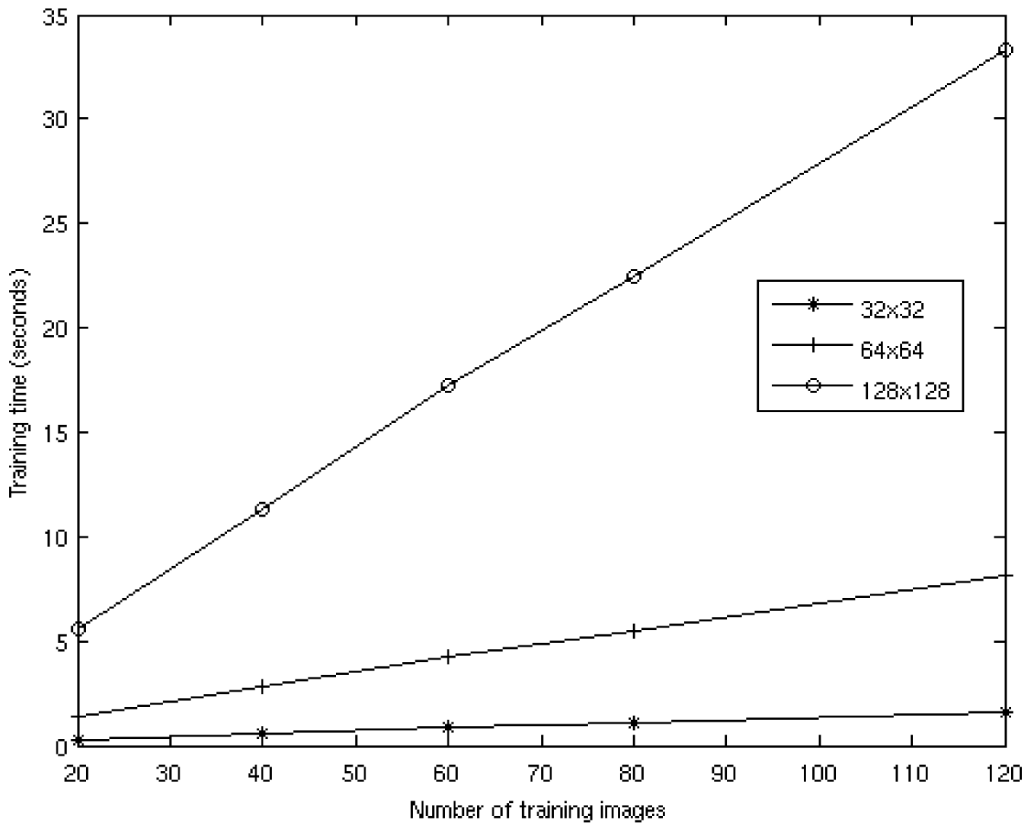


Figure 3.13: Training time for kepenekci module for train/probe images of three different sizes (32x32, 64x64, 128x128)

It is clear from the plot that the Subspace LDA algorithm is a better performer in this case as well. However, we have to consider another important factor called the retraining complexity (time) while evaluating these two algorithms on the basis of training time performance. Retraining time refers to the time required to add new data to existing training data. In almost all real world application, there is a frequent need of adding new facial images to the training database.

For Kepenekci algorithm, the retraining time is very small as it involves computation of features of the new images and appending the new data to existing training database. However, for the Subspace LDA algorithm the retraining operation is complex and time consuming. It requires re-computation of subspace LDA bases when new training images are to be added.

Hence if we consider the overall complexity of training and retraining operation, we conclude that Kepenekci algorithm has better performance.

3.5 Performance of RTFTR

We used a video containing 100 frames to test the performance of RTFTR when different combinations of face extraction (rowleynn and adaboost) and face recognition (subspacelda and kepenekci) modules were used to form the visual data processing pathway of the system.

The detection time refers to the total time it takes to extract and report all the faces in a given frame (of size 256×192). The detected faces does not necessarily refer to the correct detections and we are only considering the time it takes , for the face extraction module, to completely process a given video frame.

Similarly, the recognition time refers to the time elapsed between the instant of supplying an image (of size 64×64) to the face recognition module and the instant when it reports a face-id having closest match in the training database. The reported face-id is not necessarily the correct match and we are only considering the time it takes to complete all the steps in the recognition algorithms.

The total frame processing time is the sum of detection time, recognition time and the preprocessing time (all the overhead added by source, source transformer, pre-recognition transformer stages).

3.5.1 Use of Neural Network for Face Extraction

The face detection time for rowleynn module (that implements Neural Network based face detection technique [7]) varies in the range of 0.9 to 1.0 seconds as shown in Figure 3.15 , 3.17. Hence the task of realtime face detection using rowleynn module is not feasible. The combination of rowleynn module (for face extraction) with subspace lda (as shown in Figure 3.14) and kepenekci module (as shown in Figure 3.16) was tested to compare performance of the two face recognition algorithms when used with rowleynn module.

The recognition time for subspace lda module is very small and varies in the range of 0.0 to 0.01 second as shown in Figure 3.15 (considering an average of 5 face/nonface detections per frame). With this level of performance, we can perform recognition at 20 frames per second assuming each frame contains at most 5 faces. However, when subspace lda module is combined with rowleynn module, the performance degrades due to large face detection time of rowleynn module.

The recognition time for kepenekci module is very high and varies in the range of 0.5 to 1.0 second as shown in Figure 3.17. The recognition time plot for kepenekci module in Figure 3.21 shows relatively small recognition time which varies in the range of 0.1 to 0.15 second. This difference in recognition time for same set of frames can be attributed to the fact that the two face extraction algorithms (rowleynn and adaboost) report different number of detections for a frame. Moreover, rowleynn module has larger number of false detections as compared to adaboost module. Hence the recognition time, which is the total time to recognize all the detections (correct and false detections) of a frame, for the combination of rowleynn and kepenekci module is very high as rowleynn results in large number of false detections.

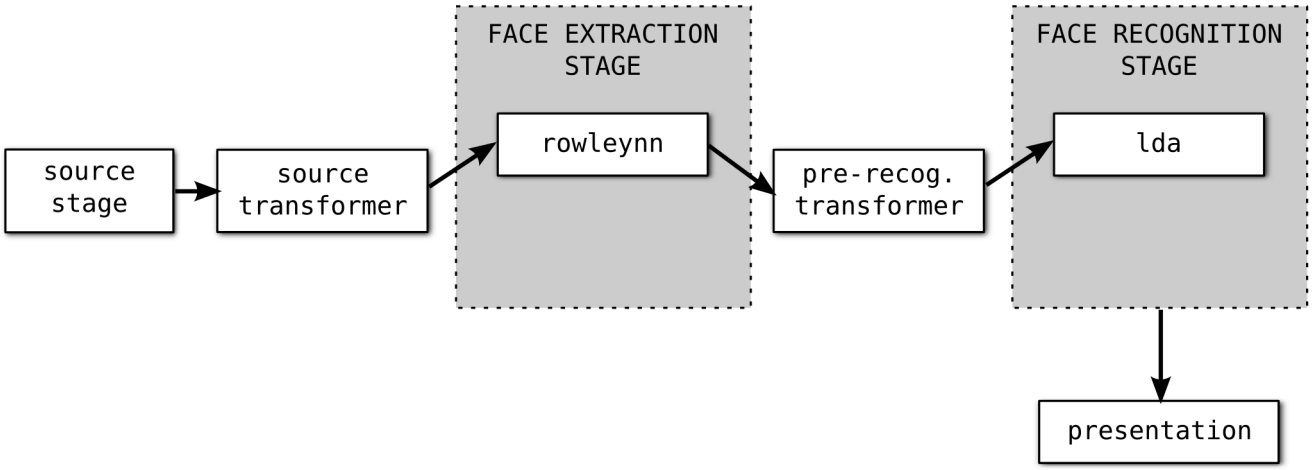


Figure 3.14: Combination of Neural Network (for face detection) and subspace LDA (for face recognition)

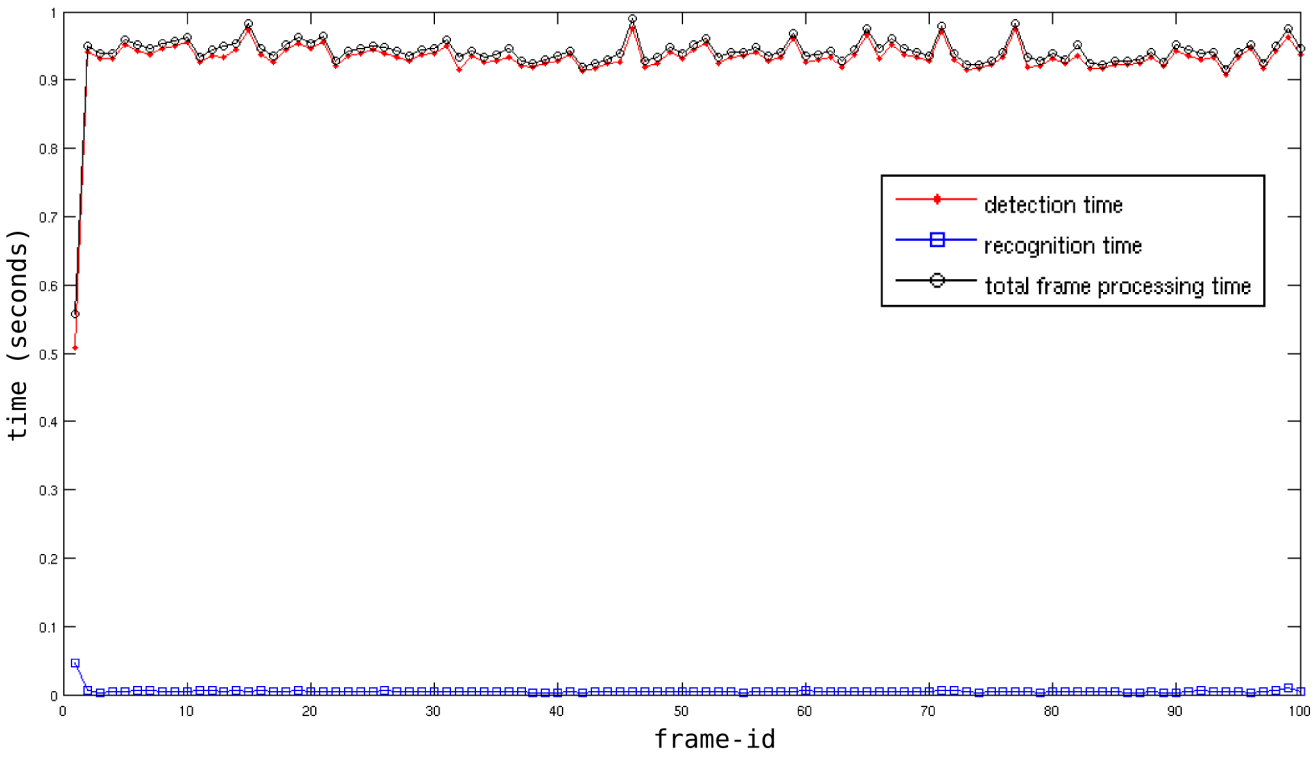


Figure 3.15: Plot showing the performance of RTFTR system realized using the combination shown in Figure 3.14

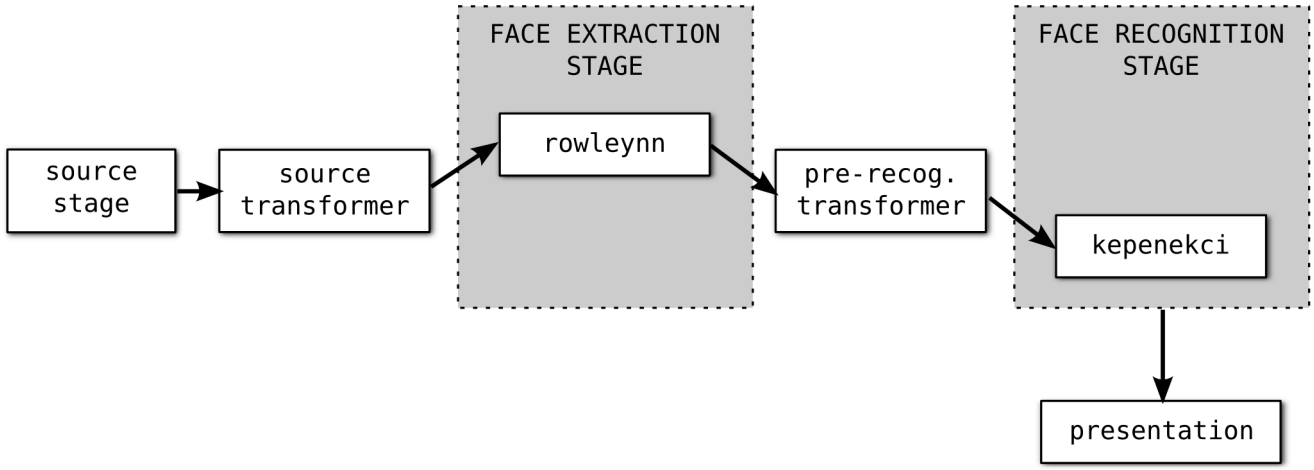


Figure 3.16: Combination of Neural Network (for face detection) and kepenekci (for face recognition)

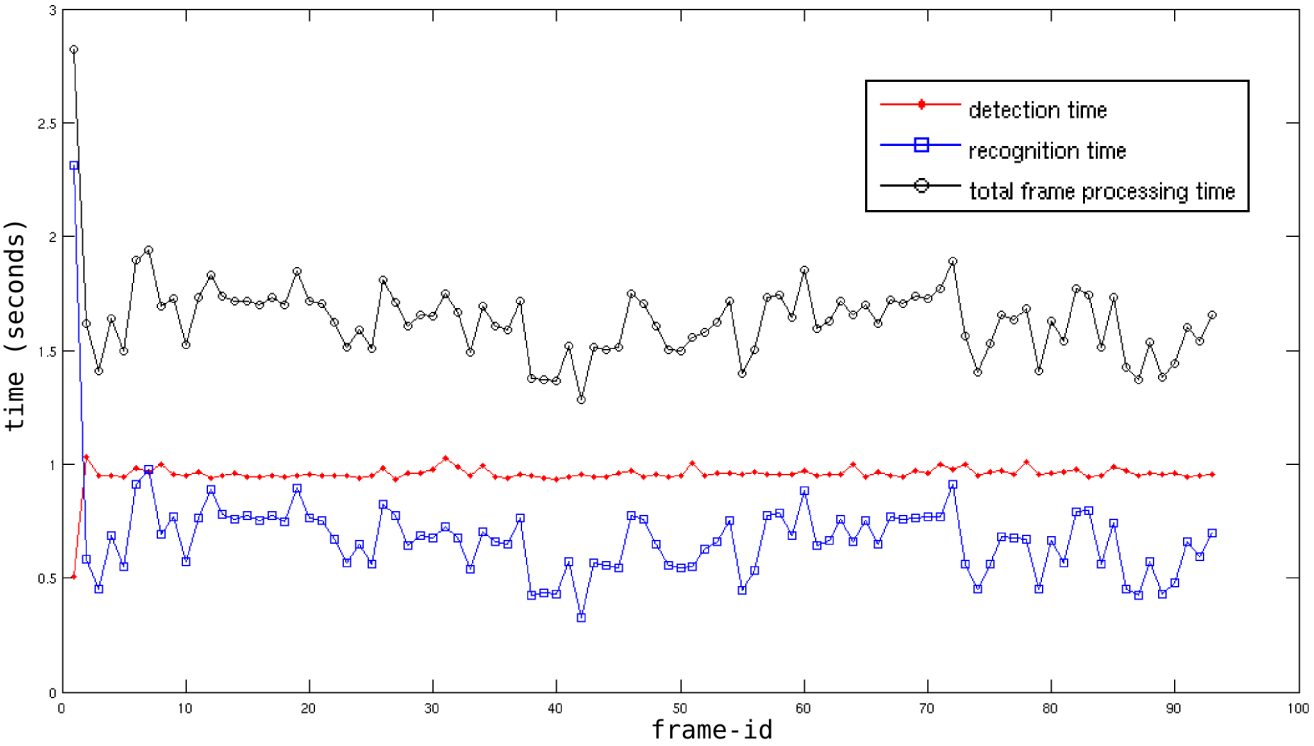


Figure 3.17: Plot showing the performance of RTFTR system realized using the combination shown in Figure 3.16

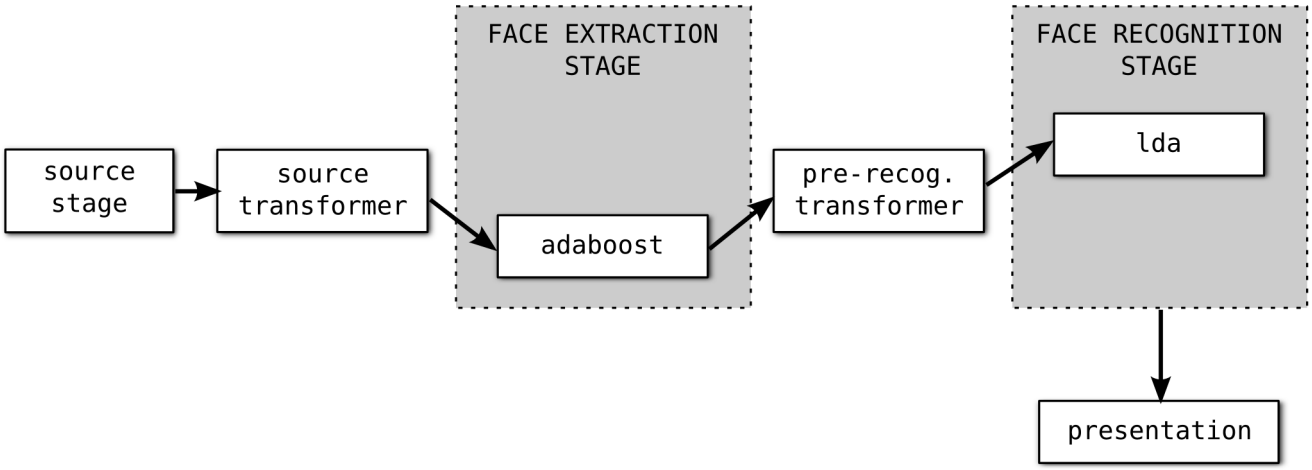


Figure 3.18: Combination of AdaBoost (for face detection) and subspace LDA (for face recognition)

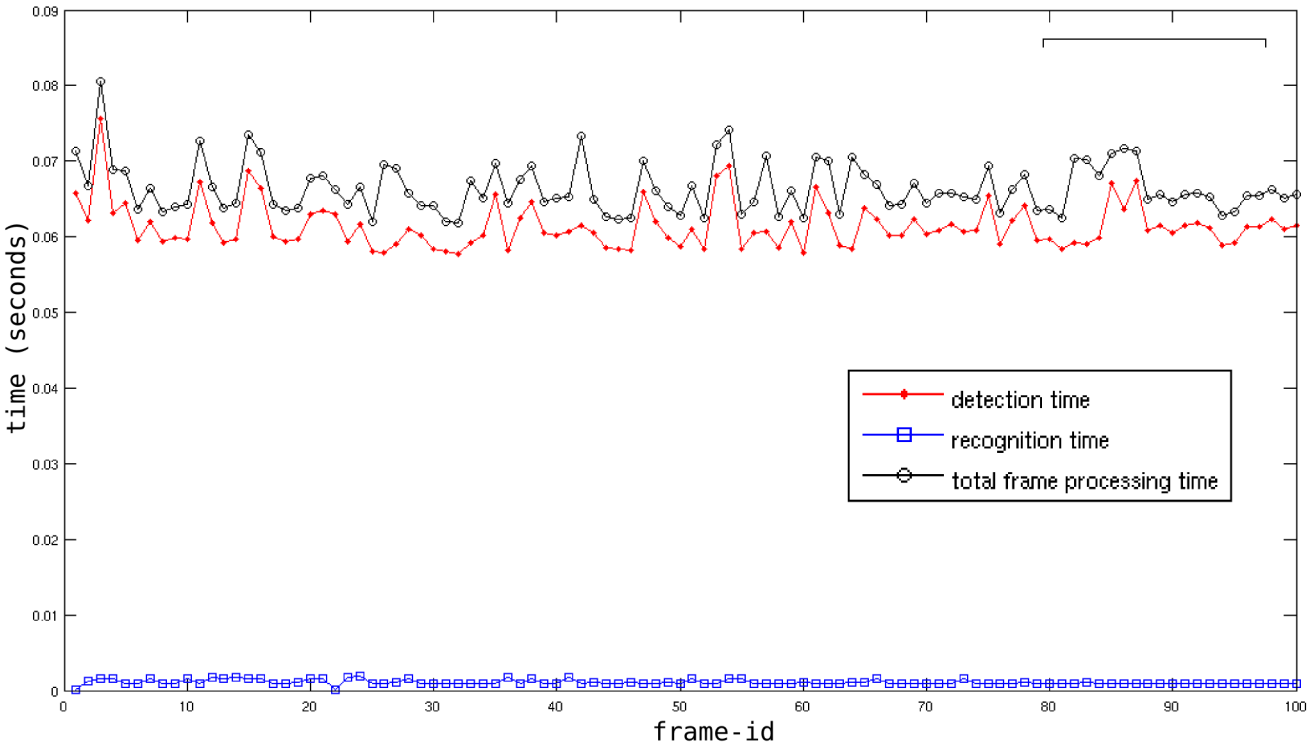


Figure 3.19: Plot showing the performance of RTFTR system realized using the combination shown in Figure 3.18

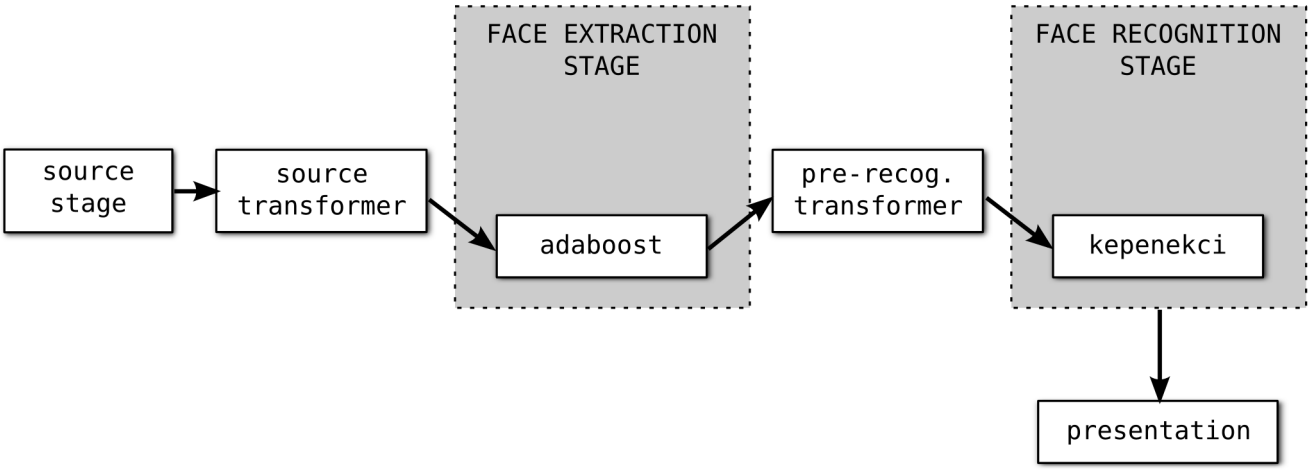


Figure 3.20: Combination of AdaBoost (for face detection) and kepenekci (for face recognition)

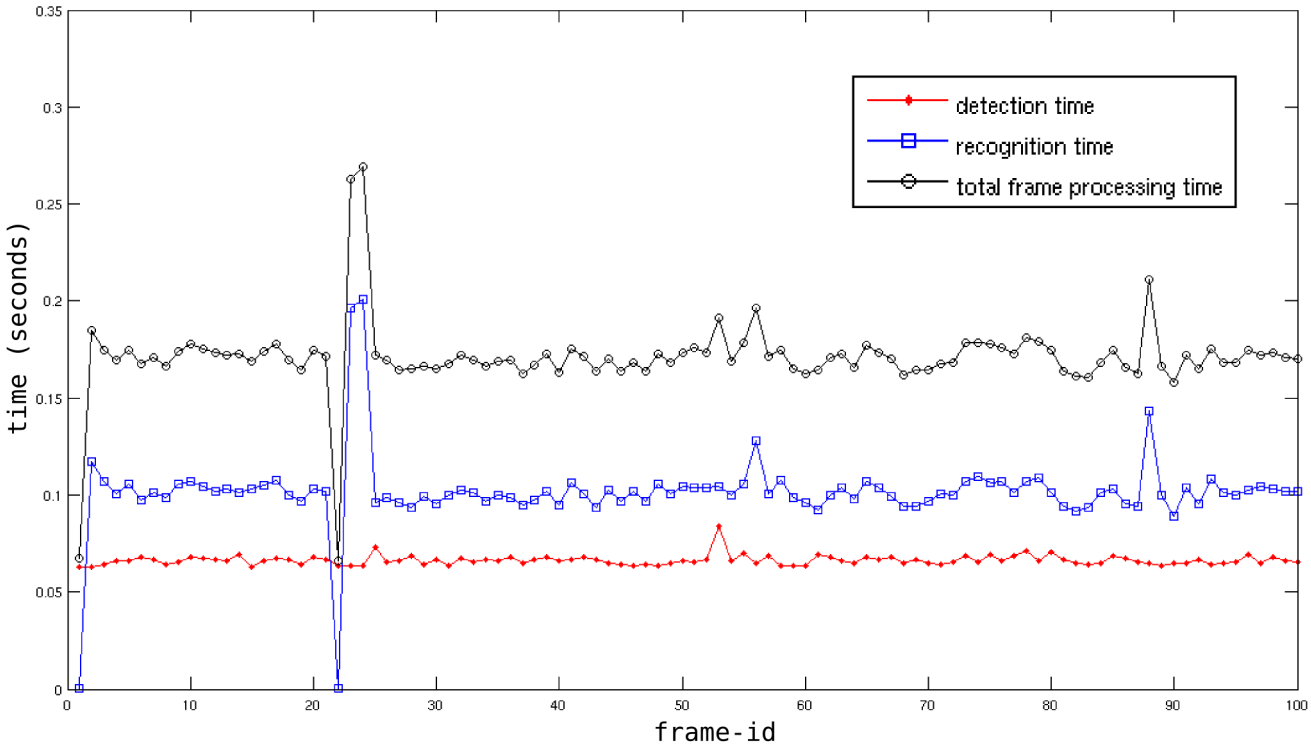


Figure 3.21: Plot showing the performance of RTFTR system realized using the combination shown in Figure 3.20

3.5.2 Use of AdaBoost for Face Extraction

The face detection time for adaboost module (that implements AdaBoost algorithm [9]) varies in the range of 0.06 to 0.08 seconds as shown in Figure 3.19 , 3.21. Hence the task of realtime face detection using adaboost module is feasible. The combination of adaboost module (for face extraction) with subspace lda (as shown in Figure 3.18) and kepenekci module (as shown in Figure 3.20) was tested to compare performance of the two face recognition algorithms when used with adaboost module.

The recognition time for subspace lda module is very small and varies in the range of 0.0 to 0.01 second as shown in Figure 3.19. With this level of performance, we can perform recognition at 20 frames per second assuming each frame contains at most 5 faces.

The recognition time for kepenekci module is very high and varies in the range of 0.5 to 1.0 second as shown in Figure 3.21. The recognition time plot for kepenekci module in Figure 3.17 shows larger recognition time which varies in the range of 0.5 to 1.0 second. This difference in recognition time for same set of frames can be attributed to the fact that the two face extraction algorithms (rowleynn and adaboost) report different number of detections for a frame. Hence the recognition time, which is the total time to recognize all the detections (correct and false detections) of a frame, for the combination of adaboost and kepenekci module is small (as compared to the combination formed with rowleynn) as adaboost results in small number of false detections.

Chapter 4: Conclusions and future work

4.1 Conclusions

For the two Face Extraction(FE) algorithms in RTFTR, we concluded that although Neural Network based approach can achieve high level of accuracy in face detection, its high detection time (1.5 to 3 seconds as shown in Figure 3.3) does not make it fit for realtime operation. Despite the very large training time (which is not a critical factor for realtime operation) of AdaBoost algorithm, it offers good performance with very small detection time (< 0.1 seconds).

> *Instead of using two different algorithms (Neural Network based approach and AdaBoost) for the task of realtime face extraction the implementation of three instances Adaboost algorithm, trained with different dataset, can provide appreciable real time performance. Results from the three instances can be aggregated using voting technique as suggested in [10].*

Processing each frame in a video ($\sim$ 30 fps) by AdaBoost algorithm to detect faces requires large amount of processing power and resources to achieve realtime operation. Use of algorithms like camshift, optical flow (for only detecting motion), etc can reduce the number of frames to be processed by the AdaBoost. These algorithm analyzes each video frame to detect presence of new faces in the video.

> *Use of techniques like camshift, optical flow (for only detecting motion) can improve the performance of the face extraction stage.*

The two Face Recognition(FR) algorithms working in unison results in a hybrid approach to face recognition. Research[12] has shown that visual processing pathway of human brain (an exemplar for artificial face recognition systems) also employs a hybrid approach for face recognition. Both algorithms have high accuracy for recognition of faces in two different regions (there exists some overlap between the two regions) of the face space. Subspace LDA performs well even for low resolution images but the performance degrades for images containing illumination variation and out of plane rotation of faces. However, Kepenekci algorithm is an illumination invariant technique having appreciable level of performance only for high resolution images.

> *An arbitrator function can be designed to assign appropriate weights (based on the nature of input image and face space region that an algorithm recognizes with high accuracy) to the result of recognition by each face recognition algorithm. A well chosen arbitrator function can improve the overall accuracy*

of the face recognition process. The absence of quantitative techniques to precisely define the nature of input images present a hurdle to the design of an optimal arbitrator function.

Nonlinearity and distortions introducted by image capture device (like a CCD camera) can drastically reduce the accuracy of feature based techniques like Kepenekci algorithm. Such errors have virtually no impact on the performance of holistic techniques like Subspace LDA. Hence, appropriate preprocessing should be performed before supplying the train/probe images. Image registration (an important image preprocessing technique) ensures that overall face structure and facial features location of train/probe images have least amount of variation. Also, a controlled environment(that can control parameters like illumination, possible poses of human face, etc) can improve the accuracy of FR algorithms.

Controlling the environment parameters (like illumination, possible facial poses, image resolution, etc) provides a proven technique for improving the accuracy of face recognition procedure. Preprocessing (histogram equalization, image registration, etc) of train/probe images can also ensure desirable level of accuracy.

4.2 Future Work

4.2.1 OpenMPI based version of RTFTR that utilizes distributed processing

OpenMPI[1] can be used to implement a parallel processing version of RTFTR that distributes task of face extraction/recognition to a number of computers. The results are aggregated by the MASTER node.

4.2.2 Flowgraph based GUI for RTFTR

A user interface that allows formation of custom processing pathway by adding components that implement different stages of RTFTR (similar to the Matlab Simulink models). This type of architecture is also being developed by gnuradio companion[2] for the gnuradio project. Present implementation of RTFTR involves integration of different rtftr modules (modules belonging different stages) using a simple C++ program. Such intuitive user interface would hide the complexity by allowing users to form processing pathway without requiring users to have the knowledge of interface used by each module for communication.

[1] `http://www.open-mpi.org` - Message Passing Interface (MPI) is a standardized API typically used for parallel and/or distributed computing

[2] GNU Radio Companion (GRC) is a graphical tool for creating signal flow graphs and generating flow-graph source code

4.2.3 Use of motion tracking to improve the performance

Present implementation of RTFTR does not use the information gained from motion tracking algorithms like camshift. It processes each frame of the video and the relationship between consecutive frames in not used during the face extraction/recognition procedure. Use of motion tracking algorithms can reduce the number of frames to be processed and hence improve the overall performance of RTFTR.

Bibliography

[1] Face recognition. http://www.biometrics.gov/Documents/FaceRec.pdf, August 2006. Subcommittee on Biometrics, National Science and Technology Council.

[2] Peter Belhumeur N., Joao Hespanha P., and Kriegman J. David. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 19(7), 1997.

[3] Kamran Etemad and Rama Chellapa. Discriminant analysis for recognition of human faces images. volume 14, August 1997.

[4] R. C. Gonzales and R. E. Woods. *Digital Image Processing*. Addison-Wesey Publishing Comp., 1992.

[5] Burcu Kepenekci. Face recognition using gabor wavelet transform, September 2001.

[6] Javier Movellan R. Tutorial on gabor filters. http://mplab.ucsd.edu/tutorials/gabor.pdf.

[7] Henry Rowley A. *Neural Network-Based Face Detection*. PhD thesis, School of Computer Science, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, May 1999.

[8] Matthew Turk A. and Alex Pentland P. Face recognition using eigenfaces. In *Journal of Cognitive Neuroscience*, volume 3, page 7286, 1991.

[9] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, December 2001.

[10] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, page 7, 2001.

[11] W. Zhao, R. Chellappa, and P. J. Phillips. Subspace linear discriminant analysis for face recognition. Technical Report CAR-TR-914, Center for Automation Research, University of Maryland, College Park, MD., 1999.

[12] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. Face recognition: A literature survey. In *ACM Computing Survey*, volume 35, pages 399–458, December 2003.

[13] Wenyi Zhao, Arvindh Krishnaswamy, Rama Chellapa, Daniel Swets L, and John Weng. Discriminant analysis of principle components for face recognition. 1998.

Appendix A: IOE Face Database

We photographed few volunteers from our college to create a small face database. This face database has been created for the testing purpose of the final year project "Real Time Face Tracking and Recognition (rtftr)" (http://rtftr.sourceforge.net). We had 13 subjects with around 12 pose variations for each subject. We also captured video for the whole photo session event. This video will also be used for future testing purpose. The IOE Face Database version 2.0 is available at
`http://rtftr.sourceforge.net/data/face_db/ioe_face_db_ver2.zip`.

A.1 Details of IOE Face Database

```
Photographs taken on  : 12:00 midday , 3 December 2008
Place                 : IOE Library
Number of subjects    : 13
Last Updated          : 7 December 2008
Total no. of images   : 227

# Train Image Naming convention #
      f0010v009
        |    |________ pose id (can vary from 001 to 999)
        |_____________ face id (can vary from 0001 to 9999)

# Probe Image Naming convention #
      t0011v003
        |    |________ pose id (can vary from 001 to 999)
        |_____________ face id (can vary from 0001 to 9999)

# Description of contents of Face Database
  .
  |-- images        --------> training face images
  |   |-- f0001v001.jpg
  |   |-- f0001v002.jpg
  |   |-- ......
  |   |-- f0013v016.jpg
  |   `-- f0013v017.jpg
  |-- probe_images  --------> group photos that can be used for testing (probe)
```

```
  |   |-- dsc04448.jpg
  |   |-- dsc04450.jpg
  |   `-- dsc04451.jpg
  `-- readme.txt    --------> detailed information about the ioe face database
```

Details of subjects in the face database

Face Id	Name	Class Roll No	No. of pose variations
0001	Niraj Sapkota	062BCT521	015
0002	Sushil Shilpakar	062BCT547	023
0003	Sabbir Kumar Manandhar	062BCT537	013
0004	Saroj Gautam	062BCT541	014
0005	Bibek Raj Dhakal	062BCT506	016
0006	Subash Basnet	062BCT544	018
0007	Madhusudan Ram Joshi	062BCT518	020
0008	Ruchin Singh	062BCT536	020
0009	Bibek Karmacharya	061BCT508	016
0010	Bibek Shrestha	061BCT509	021
0011	Lakesh Kansakar	061BCT521	017
0012	Anjan Nepal	061BCT502	017
0013	Abhishek Dutta	061BCT501	017

Appendix B: Video used for performance evaluation of FE modules

Figure B.1: Some probe (test) frames used for obtaining the results presented in section 3.2 of Chapter 3

Appendix C: Train and Probe images used for performance evaluation of FR modules

Figure C.1: Probe (test) images used for obtaining the results presented in section 3.4 of Chapter 3

Figure C.2: Training images used for obtaining the results presented in section 3.4 of Chapter 3