

# HUMAN FACE DETECTION SYSTEM



by

GC AZHAR (Leader)

GC USMAN

GC ADEEL

GC UBAID

A thesis submitted in partial fulfillment of the  
requirements for the degree of B.E Computer  
Software

National University of Sciences and  
Technology, Rawalpindi

APRIL 2004

# **ABSTRACT**

## **HUMAN FACE DETECTION SYSTEM**

by

GC AZHAR (Leader)

GC USMAN

GC ADEEL

GC UBAID

Object detection is a fundamental problem in computer vision. Detection of faces, in particular, is a critical part of face recognition and critical for systems which interact with users visually.

We present a view-based approach implemented with artificial neural networks for face detection. A retinally connected neural network examines small windows of an image, and decides whether each window contains a face. The system arbitrates between multiple networks to improve performance over a single network. We present a straightforward procedure for aligning positive face examples for training. To collect negative examples, we use a bootstrap algorithm, which adds false detections into the training set as training progresses. This eliminates the difficult task of manually selecting nonface training examples, which must be chosen to span the entire space of nonface images. Simple heuristics, such as using the fact that faces rarely overlap in images, can further improve the accuracy.

In the end we performed sensitivity analysis on the networks, and present empirical results on a large test set.

## **DECLARATION**

No portion of the work presented in this dissertation has been submitted in support of another award of qualification either at this institution or elsewhere.

## **DEDICATION**

This document is dedicated to our beloved parents who have been a source of constant encouragement for us.

## ACKNOWLEDGMENTS

We arrived at MCS on a New Year night of 2001, more than three years ago. We had a lot to learn, how to find our way around in new college and how to do computer science degree. Fortunately our instructors, friends and colleagues made this easy, teaching us about life and studies in too many ways to list them all. Let us just mention a few examples.

Thanks to our advisor, Lt.Col Rashid Satti, for teaching us about artificial intelligence, neural networks and its applications and also for his encouragement and advice.

Thanks to Dr.Usama Hasan for showing us that professors can be friends too.

Thanks to my course mates over the years: Sheraz Cheema for introducing us to basketball; Waleed Mansoor, for showing compassion in every action; Bilal Anwar for reminding us of the enthusiasm we had as a first year student; Haris, for many conversations and his bike on which we used to wave through the city. Thanks to our colleagues in the Computer Science Department who taught us so much. Above all, our poor computers who worked for our project and companions of our loneliness.

And finally, thanks to our parents. Their constant love, support, and encouragement made finishing this impossible task possible.

GC Azhar  
GC Usman  
GC Adeel  
GC Ubaid

# CONTENTS

<b>Abstract</b>	<b>i</b>
<b>Declaration</b>	<b>ii</b>
<b>Dedication</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>1 Project Overview</b>	
1.1 Introduction	
1.2 Challenges in Face Detection . . . . .	2
1.3 A View-Based Approach using Neural Networks . . . . .	4
1.4 Evaluation . . . . .	6
<b>2 Requirement Analysis</b>	
2.1 Resource Requirements . . . . .	7
2.2 Project Boundaries & Constraints . . . . .	8
2.3 Project Assumptions . . . . .	8
2.4 System Features . . . . .	9
2.5 Project Deliverables . . . . .	10
<b>3 System Design . . . . .</b>	<b>11</b>
3.1 Schematic Diagram . . . . .	11
3.2 Data Flow Diagrams . . . . .	12
<b>4 Data Preparation</b>	
4.1 Introduction . . . . .	16
4.2 Facial Feature Labeling and Alignment . . . . .	16
4.3 Preprocessing for Brightness and Contrast . . . . .	20
4.4 Face-Specific Lighting Compensation . . . . .	23
4.4.1 Linear Lighting Models . . . . .	23
4.4.2 Neural Networks for Compensation . . . . .	25
4.4.3 Quotient Images for Compensation . . . . .	27
4.5 Summary . . . . .	29
<b>5 Upright Face Detection</b>	
5.1 Introduction . . . . .	31
5.2 Individual Face Detection Networks . . . . .	32
5.2.1 Face Training Images . . . . .	33
5.2.2 Non-Face Training Images . . . . .	34
5.2.3 Active Learning . . . . .	34
5.2.4 Exhaustive Training . . . . .	36
5.3 Analysis of Individual Networks . . . . .	37
5.3.1 Sensitivity Analysis . . . . .	37
5.3.2 ROC (Receiver Operator Characteristic) Curves . . . . .	38
5.4 Refinements . . . . .	41
5.4.1 Clean-Up Heuristics . . . . .	41
5.4.2 Arbitration among Multiple Networks . . . . .	44
5.5 Evaluation . . . . .	47
5.5.1 Upright Test Set . . . . .	47
5.5.2 Example Output . . . . .	50
5.5.3 Effect of Lighting Variation . . . . .	50

<b>6 Tilted Face Detection</b> .....	<b>54</b>
6.1 Introduction .....	54
6.2 Algorithm .....	55
6.2.1 Derotation Network .....	56
6.2.2 Detector Network .....	58
6.3 Arbitration Scheme .....	58
6.3 Analysis of the Networks .....	59
6.4 Evaluation .....	60
6.4.1 Derotation Network with Upright Face Detectors .....	60
6.4.2 Proposed System .....	61
6.4.3 Exhaustive Search of Orientations .....	66
6.4.4 Upright Detection Accuracy .....	68
6.5 Summary .....	70
<b>7 Non-Frontal Face Detection</b> .....	<b>71</b>
7.1 Introduction .....	71
7.2 Geometric Distortion to a Frontal Face .....	72
7.2.1 Labelling the 3D Pose of the Training Images .....	72
7.2.2 Representation of Pose .....	77
7.2.3 Training the Pose Estimator .....	78
7.2.4 Geometric Distortion .....	80
7.3 View-Based Detector .....	81
7.3.1 View Categorization and Derotation .....	81
7.3.2 View-Specific Face Detection .....	84
7.4 Evaluation of the View-Based Detector .....	86
7.4.1 Non-Frontal Test Set .....	86
7.4.2 Experiments .....	86
7.5 Summary .....	90
<b>8 Speedups</b> .....	<b>92</b>
8.1 Introduction .....	92
8.2 Fast Candidate Selection .....	92
8.2.1 Candidate Selection .....	92
8.2.2 Candidate Localization .....	93
8.2.3 Candidate Selection for Tilted Faces .....	95
8.3 Change Detection .....	97
8.4 Skin Color Detection .....	98
8.5 Evaluation of Optimized Systems .....	100
8.6 Summary .....	102
<b>9 User Manual</b> .....	<b>103</b>
9.1 Pre-Requisites .....	103
9.2 Quick Start Instructions .....	103
9.3 Advanced Settings .....	104
9.3.1 Windows Interval .....	106
9.3.2 Threshold .....	106
9.3.3 The Scaling Properties .....	106
9.4 Training A Neural Network for Face Detection .....	106
<b>Appendices</b> .....	<b>111</b>
Appendix A .....	111
Appendix B .....	115

# Chapter 1

## Project Overview

### 1.1 Introduction

The goal of our project is to show that the face detection problem can be solved efficiently and accurately using a view-based approach implemented with artificial neural networks. Specifically, we will demonstrate how to detect upright, tilted, and non-frontal faces in cluttered images, using multiple neural networks whose outputs are arbitrated to give the final output.

Object detection is an important and fundamental problem in computer vision, and there have been many attempts to address it. The techniques which have been applied can be broadly classified into one of two approaches: matching two- or three-dimensional geometric models to images or matching view-specific image-based models to images. Previous work has shown that view-based methods can effectively detect upright frontal faces and eyes in cluttered backgrounds.

In developing a face detector that uses machine learning, three main sub problems arise. First, images of objects such as faces vary considerably, depending on lighting, occlusion, pose, facial expression, and identity. The detection algorithm should explicitly deal with as many of these sources of variation as possible, leaving little unmodelled variation to be learned. Second, one or more neural-networks must be trained to deal with all remaining variation in distinguishing faces from non-faces. Third, the outputs from multiple detectors must be combined into a single decision about the presence of a face.

Often, face recognition systems work by first applying a face detector to locate the face, then applying a separate recognition algorithm to identify the face. Other face recognition system sometimes use the hypothesize and verify technique, in which they



first generate a hypothesis of which object is present (recognition), then use a more precise algorithm to verify whether that object is actually present (detection).

## **1.2 Challenges in Face Detection**

Object detection is the problem of determining whether or not a sub-window of an image belongs to the set of images of an object of interest. Thus, anything that increases the complexity of the decision boundary for the set of images of the object will increase the difficulty of the problem, and possibly increase the number of errors the detector will make.

Suppose we want to detect faces that are tilted in the image plane, in addition to upright faces. Adding tilted faces into the set of images we want to detect increases the set's variability, and may increase the complexity of the boundary of the set. Such complexity makes the detection problem harder. Note that it is possible that adding new images to the set of images of the object will make the decision boundary becomes simpler and easier to learn. One way to imagine this happening is that the decision boundary is smoothed by adding more images into the set. However, the conservative assumption is that increasing the variability of the set will make the decision boundary more complex, and thus make the detection problem harder.

There are many sources of variability in the object detection problem, and specifically in the problem of face detection. These sources are outlined below.

### **Variation in the Image Plane:**

The simplest type of variability of images of a face can be expressed independently of the face itself, by rotating, translating, scaling, and mirroring its image. Also included in this category are changes in the overall brightness and contrast of the image, and occlusion by other objects. Examples of such variations are shown in Figure 1.1.

### **Pose Variation:**

Some aspects of the pose of a face are included in image plane variations, such as rotation and translation. Another source of variation is the distance of the face from the camera. Examples of such variations are shown in Figure 1.1.



**Figure 1.1:** Examples of how face images between poses and between different individuals.

**Lighting and Texture Variation:**

Up till now, we have described variations due to the position and orientation of the object with respect to the camera. Now we come to variation caused by the object and its environment, specifically the object's surface properties and the light sources. Changes in the light source in particular can radically change a face's appearance. Examples of such variations are shown in Figure 1.2.



**Figure 1.2:** Examples of how images of faces change under extreme lighting conditions.

**Background Variation:**

When an object has a predictable shape, it is possible to extract a window which contains only pixels within the object, and to ignore the background. However, for profile faces, the border of the face itself is the most important feature, and its shape

varies from person to person. Thus the boundary is not predictable, so the background cannot be simply masked off and ignored. A variety of different backgrounds can be seen in the example images of Figures 1.1 and 1.2.

### **Shape Variation:**

A final source of variation is the shape of the object itself. For faces, this type of variation includes facial expressions, whether the mouth and eyes are open or closed, and the shape of the individual's face, as shown in some of the examples of Figure 1.1.

The next section will describe the approach to the face detection problem, and show how each of the above sources of variation can be addressed.

## **1.3 A View-Based Approach using Neural Networks**

The face detection system is based on four main steps:

### **1. Localization and Pose Estimation:**

Use of a machine learning approach, specifically an artificial neural network, requires training examples. To reduce the amount of variability in the positive training images, they are aligned with one another to minimize the variation in the positions of various facial features.

At runtime, we do not know the precise facial feature locations, and so we cannot use them to locate potential face candidates. Instead, we use exhaustive search over location and scale to find all candidate locations.

It is at this stage rotation of the face, both in- and out-of-plane, are handled. A neural network analyzes the potential face region, and determines the pose of the face. This allows the face to be rotated to an upright position (in-plane) and selects the appropriate detector network for the particular out-of-plane orientation.

### **2. Preprocessing:**

To further reduce variation caused by lighting or camera differences, the images are preprocessed with standard algorithms such as histogram equalization to improve the overall brightness and contrast in the images. We also examine the possibility of

lighting compensation algorithms that use knowledge of the structure of faces to perform lighting correction.

### 3. Detection:

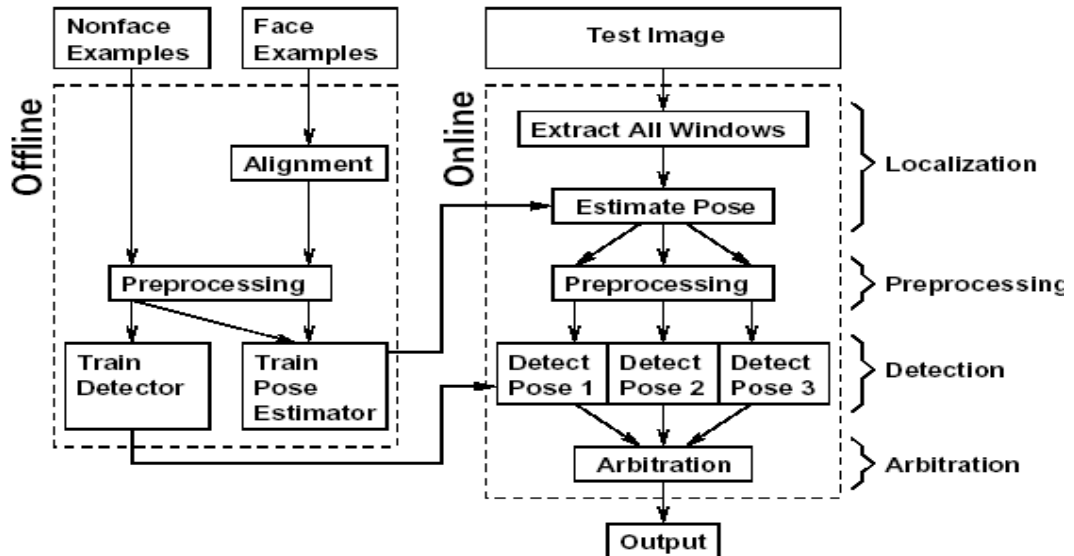
The potential faces which are already normalized in position, pose, and lighting in the first two steps are examined to determine whether they are really faces. This decision is made by neural networks trained with many face and non-face example images. This stage

Handles all sources of variation in face images not accounted for the in the previous two steps. Separate networks are trained for frontal, tilted, and non frontal faces.

### 4. Arbitration:

In addition to using three separate detector, one for each class of poses of the face, multiple networks are also used within each pose. Each network learns different thing from the training data, and makes different mistakes. Their decisions can be combined using some simple heuristics, resulting in reinforcement of correct face detections and suppression of false alarms.

Together these steps attempt to account for the sources of variability described in the previous section. These steps are illustrated schematically by Figure 1.3.



**Figure 1.3:** Schematic diagram of the main steps of the face detection algorithm.

## 1.4 Evaluation

We evaluated the accuracy of the algorithms developed. A number of test sets were used, with images collected from a variety of sources, including the World Wide Web, scanned photographs and newspaper clippings, and digitized video images.

Each test set is designed to test one aspect of the algorithm, including the ability to detect faces in cluttered backgrounds, the ability to detect a wide variety of faces of different people, and the detection of faces of different poses. An overview of the results is given in Table 1.4. We will see that the upright detector is able to detect 36.0% of faces on a test set containing mostly upright faces, while the tilted face detector has comparable detection rates. Both of these systems have fairly low false alarm rates. The detection rate for the non-frontal detector is significantly lower, reflecting the relative difficulty of these problems.

**Table 1.4:** Overview of the results from the systems

System	Test Set	Detection Rate	False Alarms
Upright Detector (Chapter 3)	<i>Upright Test Set</i> (130 images, 507 upright faces)	86.0%	31
Tilted Detector (Chapter 4)	<i>Tilted Test Set</i> (50 images, 223 frontal faces)	85.7%	15
Non-Frontal Detector (Chapter 5)	<i>Non-Frontal Test Set</i> (53 images, 96 faces)	56.2%	118

# Chapter 2

## Requirement Analysis

### 2.1 Resource Requirements

Following are the resources which are utilized in our project

a) Personnel

The personnel required to complete the project consist of following.

- GC Azhar Mehmood (Ldr)
- GC Usman Arif
- GC Adeel Sajjad
- GC Ubaid Rafiq

b) Hardware

The basic hardware required to accomplish this project is listed as under.

- Personal Computers
- Digital camera
- Scanner

c) Software

Following softwares were used in the project.

- Matlab
- C++
- Visual Studio.net
- MS Word
- MS paint

d) Office Requirements

The place where we will actually implement the project. It will be the Computer Lab of the college as well as our residences.

## **2.2 PROJECT BOUNDARIES & CONSTRAINTS**

- Project is limited to detect faces (not recognition) with an acceptable number of false positives.
- Facial Features should be clearly visible for detection of face (Frontal Face Detection).
- System works effectively with 640 X 500 resolution images. Larger resolution images lead to greater time for detection which becomes potentially inefficient.
- Distance from the camera is also one limitation.
- System suffers if the contrast and the lighting conditions are not balanced even after preprocessing.
- Skin threshold below one detect faces but also increases error rates.
- To improve quality of detection windows interval should be decreased but at the same time it will increase searching time.

## **2.3 PROJECT ASSUMPTIONS**

There are few assumptions which were taken at the start of the project to make the detection task simpler.

- Faces rarely overlap in the images.
- Images with incomplete facial features are not catered for face detection.
- Faces that appear far in the background with little information will not be taken into account.
- Test images are taken assuming that they are fair in lighting and contrast.
- The effectiveness and efficiency of the system is dependent on training the neural network.
- Heuristics such as detecting face with height to width ratio can be used where possible.
- The detection rate of the system cannot be 100%.

## **2.4 SYSTEM FEATURES**

- **Flexibility:-**

System is flexible to cater for user requirements. Any format of the input image such as bmp, jpeg, png, gif etc can be given to the system. The output of the system can be saved for future reference.

- **Training Functions:-**

Various training functions are available such as linear, sigmoid and Gaussian .We can select according to our requirements.

- **Searching Parameters:-**

. Searching parameters can be varied to make the system more effective. Windows interval, skin threshold and scaling parameters can be set individually by the user.

- **Active Learning:-**

The system is provided with the facility that the system can be trained as the training progresses. From the output user can select manually the windows and add them to the training set as positive or negative examples. This improves the accuracy of the system.

- **Neural Network Editor**

User can edit the neural network settings. New neural network can be created with 30 x 30 input masks. Any of the training functions and training images can be provided for this new network. The user can also train the system offline.

- **Time Estimation**

By looking at the searching parameters user can guess the time required to scan the test image. The “searching progress bar” gives the total windows and the



windows scanned so far by the system. Also it provides the total number of patterns found in the image.

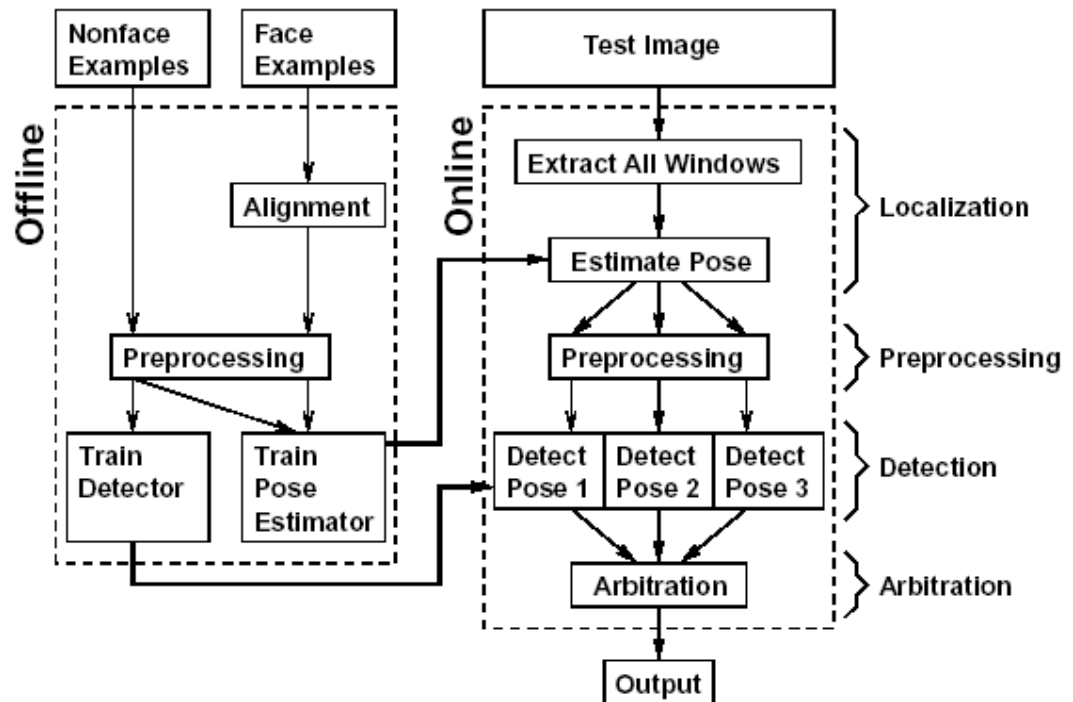
## **2.5 PROJECT DELIVERABLES**

- Test Image Database
- Face Database for Training
- Non-Face Database for Training
- Implemented Neural Network
- Neural Network Editor
- Trained Neural Network
- Detection System
- User Manual

# Chapter 3

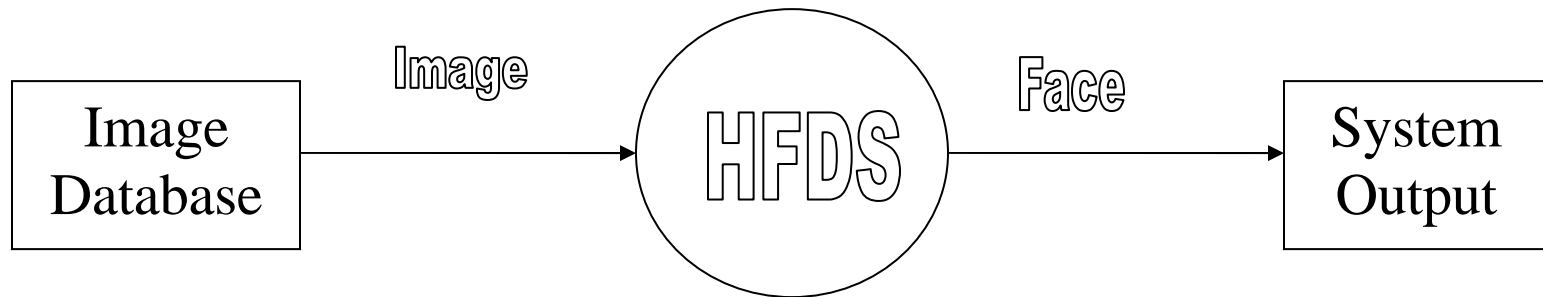
## SYSTEM DESIGN

### 3.1 Schematic Diagram

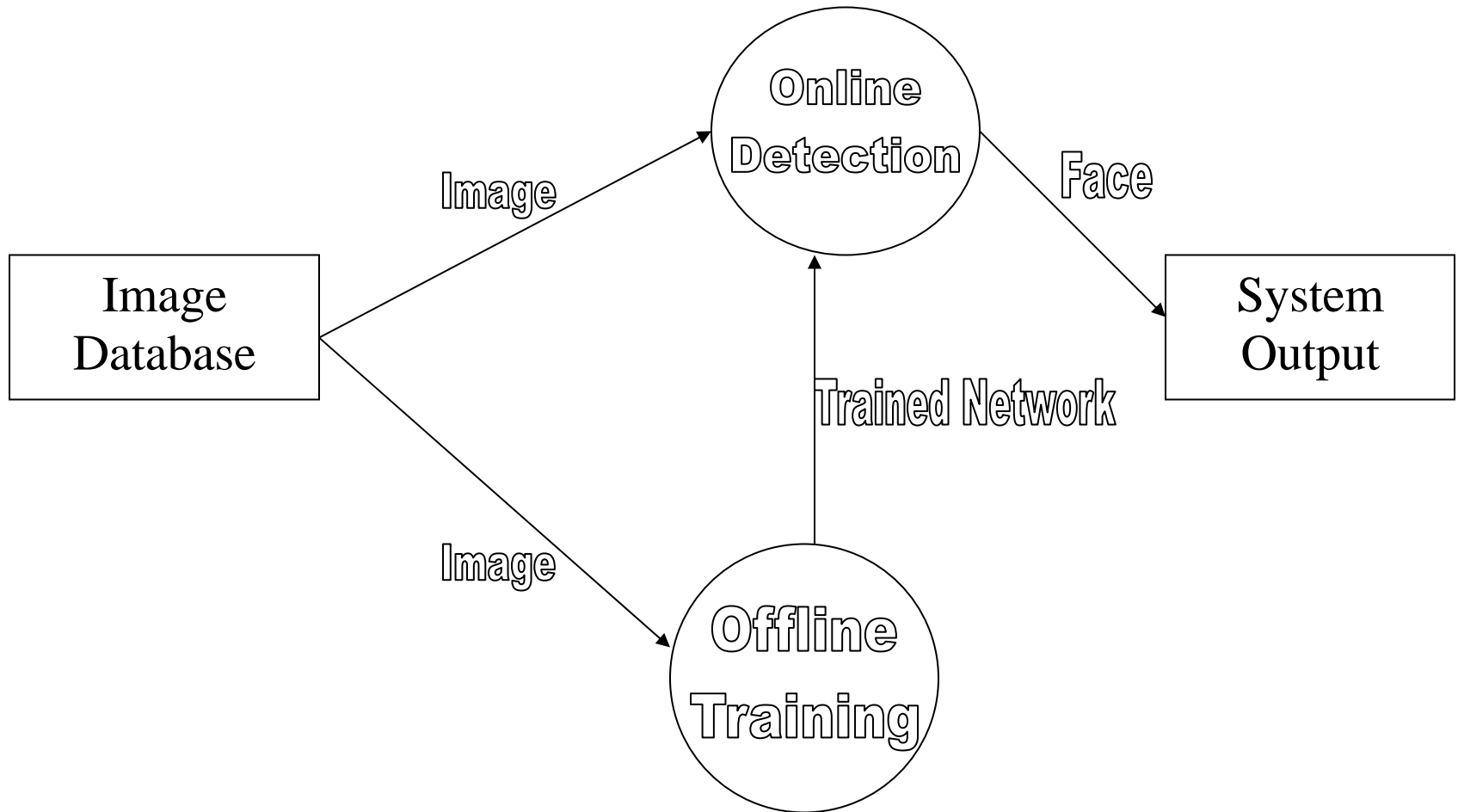


### 3.2 Data Flow Diagrams

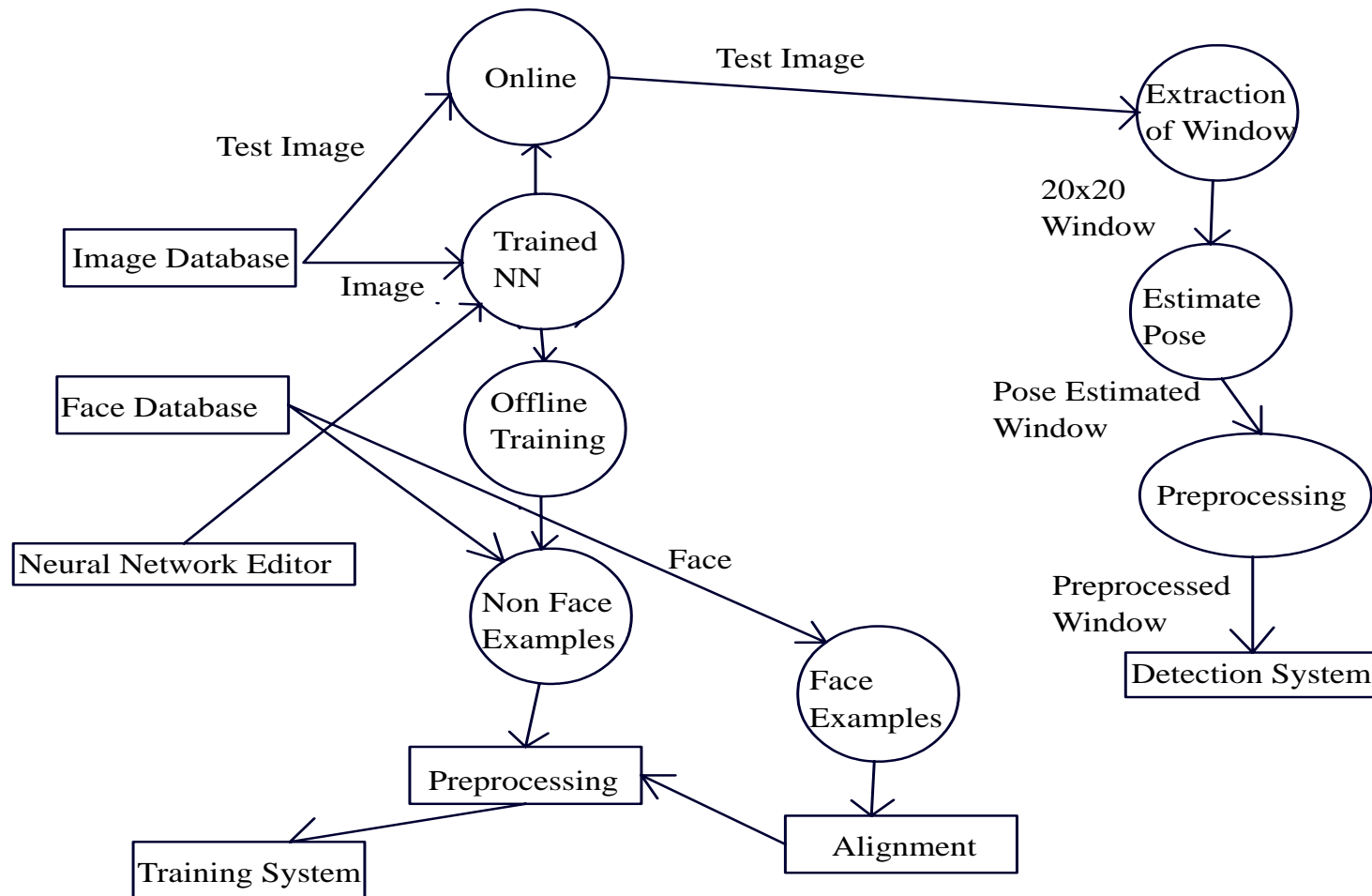
LEVEL 0 DFD



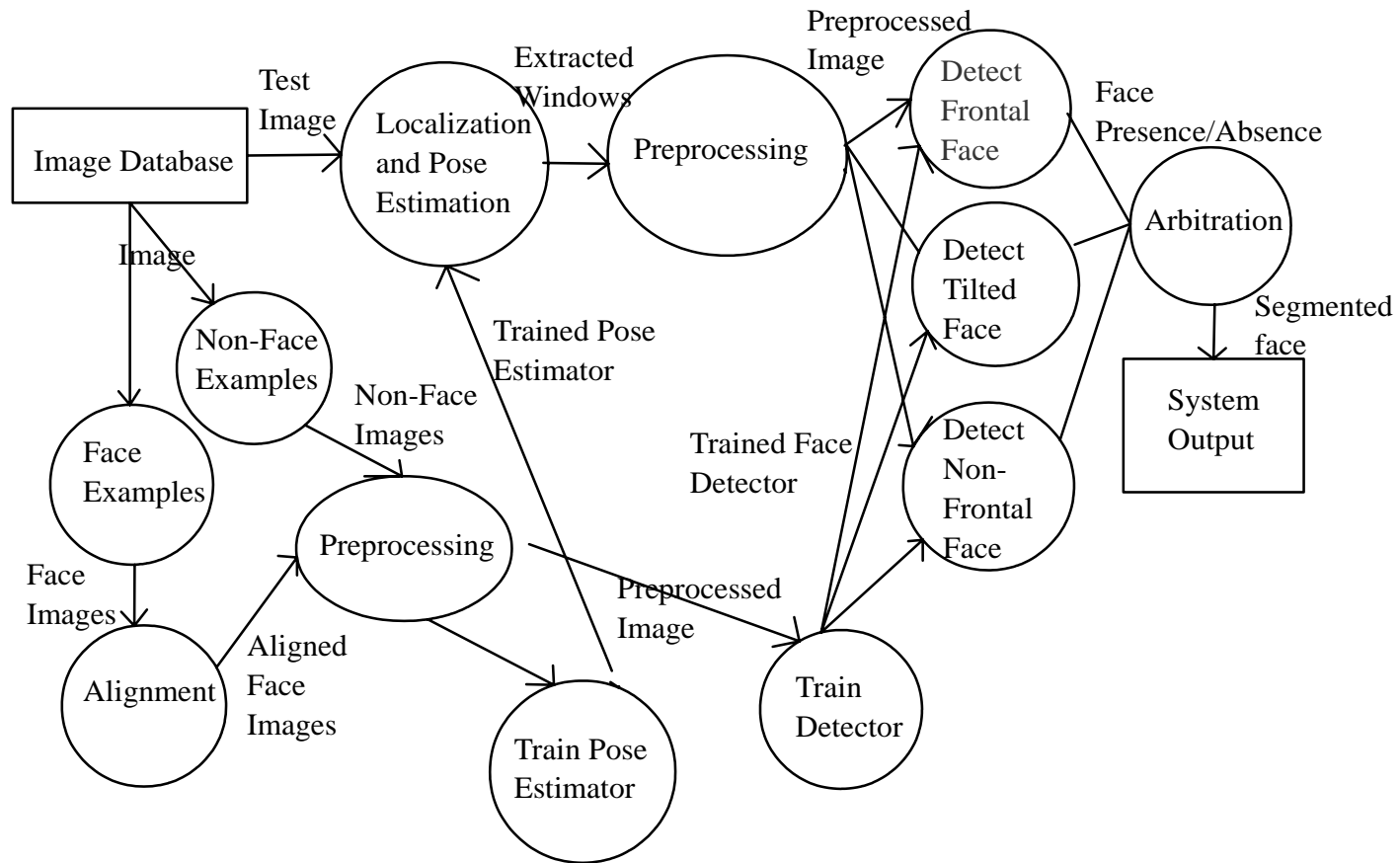
# LEVEL 1 DFD



# LEVEL 2 DFD



# LEVEL 3 DFD



# Chapter 4

## Data Preparation

### 4.1 Introduction

This project will utilize a view-based approach to face detection, and will use a statistical model (an artificial neural network) to represent each view. A view-based face detector must determine whether or not a given sub-window of an image belongs to the set of images of faces. Variability in the images of the face may increase the complexity of the decision boundary to distinguish faces from non-faces, making the detection task more difficult.

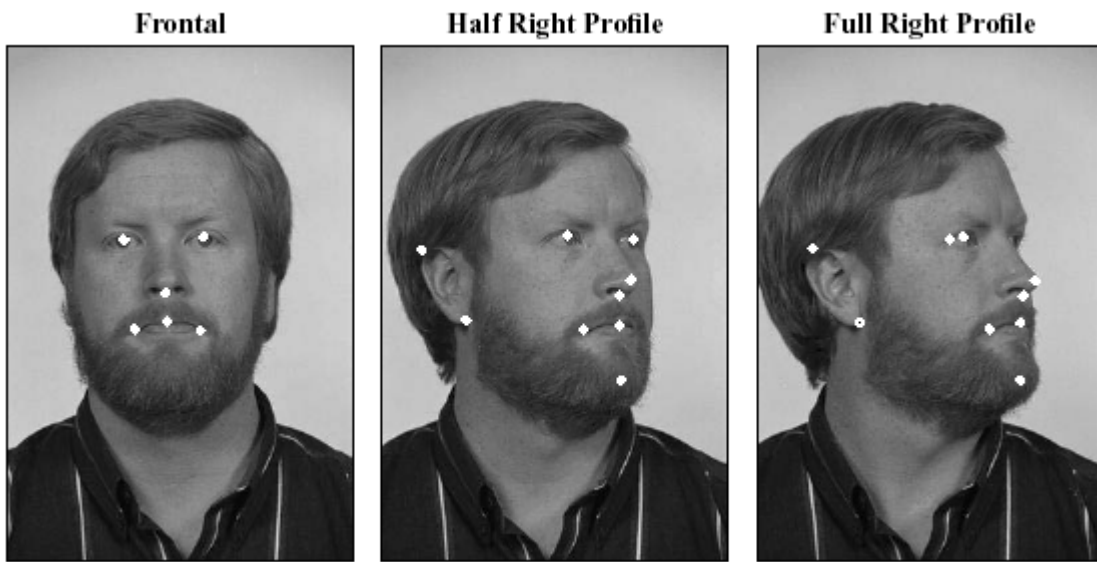
### 4.4 Facial Feature Labeling and Alignment

The first step in reducing the amount of variation between images of faces is to align the faces with one another. This alignment should reduce the variation in the two dimensional position, orientation, and scale of the faces. Ideally, the alignment would be computed directly from the images, using image registration techniques. This would give the most compact space of images of faces. However, the image intensities of faces can differ quite dramatically, which would make some faces hard to align with each other, but we want every face to be aligned with every other face.

The solution used for this project is manual labeling of the face examples. For each face, a number of feature points are labeled, depending on the three-dimensional pose of the head, as listed in Figure 4.1.

The next step is to use this information to align the faces with one another. First, we must define what is meant by alignment between two sets of feature points. We define it as the rotation, scaling, and translation which minimize the sum of squared distances between pairs of corresponding features. In two dimensions, such a coordinate transformation can be written in the following form:

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



**Figure 4.1:** Features points manually labeled on the face, depending on the three dimensional pose of the face. The left profile views are mirrors of the right profiles.

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

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



When there are two or more pairs of distinct feature points, this system of linear equations can be solved by the pseudo-inverse method. Renaming the matrix on the left hand side as  $\mathbf{A}$ , the vector of variables  $(a, b, t_x, t_y)^T$  as  $\mathbf{T}$ , and the right hand side as  $\mathbf{B}$ , the pseudo-inverse solution to these equations is:

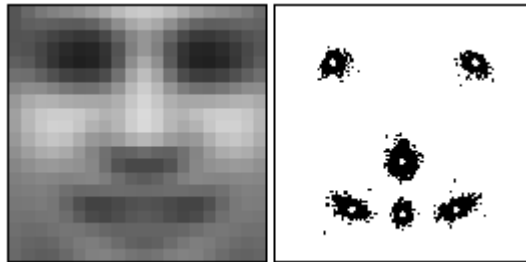
$$\mathbf{T} = (\mathbf{A}^T \mathbf{A})^{-1} (\mathbf{A}^T \mathbf{B})$$

The pseudo-inverse solution yields the transformation  $\mathbf{T}$  which minimizes the sum of squared differences between the sets of coordinates  $x_{-i}, y_{-i}$  and the transformed versions of  $x_i, y_i$ , which was our goal initially.

Now that we have seen how to align two sets of labeled feature points, we can move on to aligning sets of feature points. The procedure is described in following algorithm.

1. Initialize  $\bar{\mathbf{F}}$ , a vector which will be the average positions of each labeled 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. For faces of another pose, these positions might be derived from a 3D model of an average head.
4. For each face  $i$ , use the alignment procedure to compute the best rotation, translation, and scaling to align the face's features  $\mathbf{F}_i$  with the average feature locations  $\bar{\mathbf{F}}$ . Call the aligned feature locations  $\mathbf{F}_{-i}$ .
3. Update  $\bar{\mathbf{F}}$  by averaging the aligned feature locations  $\mathbf{F}_{-i}$  for each face  $i$ .
4. The feature coordinates in  $\bar{\mathbf{F}}$  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  $\bar{\mathbf{F}}$ .
5. Go to step 4.

Empirically, this algorithm converges within five iterations, yielding for each face the transformation which maps it to close to a standard position, and aligned with all the other faces. Once the parameters needed to align the faces are known, the image can be resampled using bilinear interpolation to produce a cropped and aligned image. The averages and distributions of the feature locations for frontal faces are shown in Figure 4.4, and examples of images that have been aligned using this technique are shown in Figure 4.3.



**Figure 4.2:** Left: Average of upright face examples. Right: Positions of average facial feature locations (white circles), and the distribution of the actual feature locations (after alignment) from all the examples (black dots).

In training a detector, obtaining a sufficient number of examples is an important problem. One commonly used technique is that of virtual views, in which new example images are created from real images. In this project, this has taken the form of randomly rotating, translating, and scaling example images by small amounts



**Figure 4.3:** Example upright frontal face images aligned to one another.



**Figure 4.4:** Example upright frontal face images, randomly mirrored, rotated, translated, and scaled by small amounts.

Once the faces are aligned to have a known size, position, and orientation, the amount of variation in the training data can be controlled. The detector to be made more or less robust to particular variations in a desired degree. Some example images in which random amounts rotation (up to  $10^\circ$ ), random translations of up to half a pixel, and random scalings up to 10% are shown in Figure 4.4.

### 4.3 Preprocessing for Brightness and Contrast

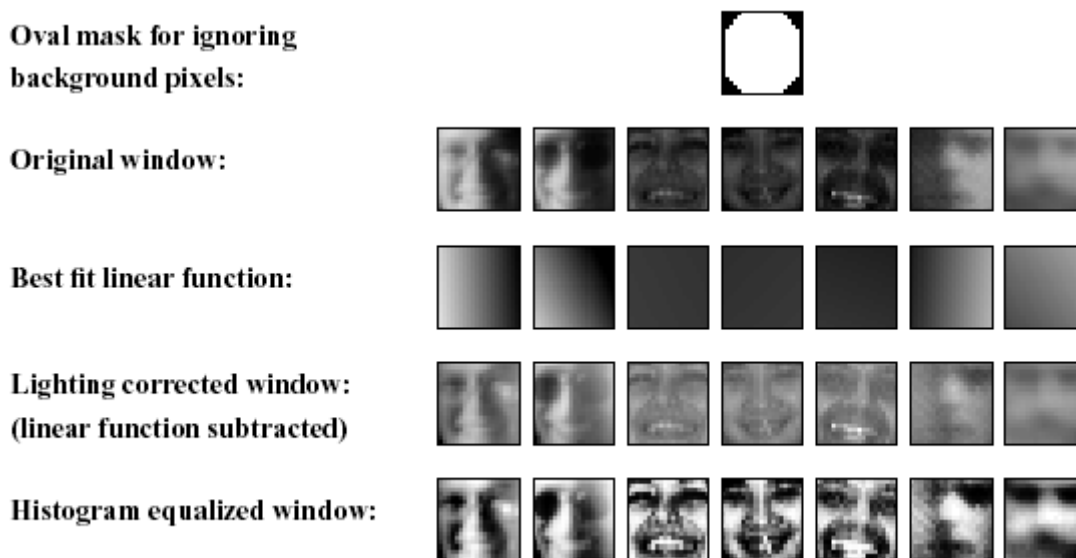
After aligning the faces, there is one remaining major source of variation (apart from intrinsic differences between faces). This variation is caused by lighting and camera characteristics, which can result in brightly or poorly lit images, or images with poor contrast.

We first address these problems by using a simple image processing approach. This 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 (shown in Figure 4.5a). Pixels outside the oval may represent the background, so those intensity values are ignored in computing the lighting variation across the face. If the intensity of a pixel  $x, y$  is  $I(x, y)$ , then we want to fit this linear model parameterized by  $a, b, c$  to the image:

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

The choice of this particular model is somewhat arbitrary. It is useful to be able to represent brightness differences across the image, so a non-constant model is useful. The variation is limited to linear to keep the number of parameters low and allow them to be fit quickly. Collecting together the contributions for all the pixels in the oval window gives an over-constrained matrix equation, which is solved by the pseudo-inverse method. 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, histogram equalization is performed, which non-linearly maps the intensity values to expand the range of intensities in the window. The histogram is computed for pixels inside an oval region in the window. This compensates for differences in camera input gains, as well as improving contrast in some cases. Some sample results from each of the preprocessing steps are shown in Figure 4.5. The algorithm for this step is as follows. We first compute the intensity histogram of the window, where each intensity level is given its own bin. This histogram is then converted to a cumulative histogram, in which the value at each bin says how many pixels have intensities less than or equal to the intensity of the bin.

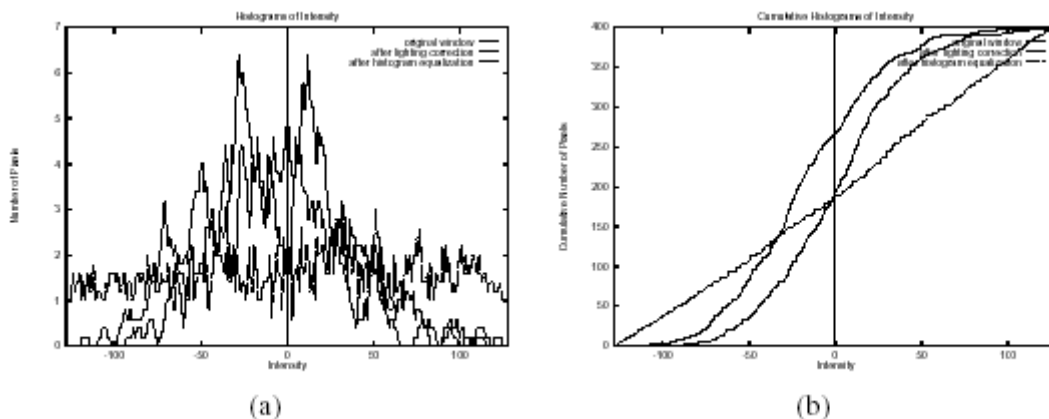


**Figure 4.5:** The steps in preprocessing a window. First, a linear function is fit to the intensity values in the window, and then subtracted out, correcting for some extreme lighting conditions. Then, histogram equalization is applied, to correct for different camera gains and to improve contrast. For each of these steps, the mapping is computed based on pixels inside the oval mask, and then applied to the entire window.

The goal is to produce a flat histogram, which is an image in which each pixel intensity occurs an equal number of times. The cumulative histogram of such an image will have that property that the number of pixels with an intensity less than or equal to a given intensity is proportional to that intensity.

Given an arbitrary image, we can produce an image with a linear cumulative histogram by adjusting the pixel intensities. Each intensity will be mapped to the value of the cumulative histogram for that bin. This guarantees that the number of pixels matches the intensity, which is the property we want. In practice, it is impossible to get a perfectly flat histogram (for example, the input image might have a constant intensity), so the result is only an approximately flat intensity histogram. To see how the histograms change with each step of the algorithm, see Figure 4.6.

In some parts of this project, only histogram equalization with subtracting the linear model is used. This is done when we do not know which pixels in the input window are likely to be foreground or background, and cannot apply the linear correction to just the face. Instead, we just apply the histogram equalization to the whole window, hoping that it will reduce the variability somewhat, without the background pixels having too much effect on the appearance of the face in the foreground.



**Figure 4.6:** (a) Smoothed histograms of pixel intensities in a  $40 \times 40$  window as it is passed through the preprocessing steps. Note that the lighting correction centers the peak of intensities at zero, while the histogram equalization step flattens the histogram. (b) The same three steps shown with cumulative histograms. The cumulative histogram of the result of lighting correction is used as a mapping function, to map old intensities to new ones.

## 4.4 Face-Specific Lighting Compensation

Part of the motivation of the preprocessing steps in the previous section is to have robustness to variations in the lighting conditions, for instance lighting from the side of the face which changes its overall appearance. However, there are limits to what “dumb” corrections, with no knowledge of the structure of faces, can accomplish. In this section, I will present some preliminary ideas on how to intelligently correct for lighting variation.

### 4.4.1 Linear Lighting Models

The ideas in this section are based on the illumination models, in which the range of appearances has been explored that an object can take on under differently lighting conditions. One assumption used in it is that adding light sources to a scene results in an image which is a sum of the images for each individual light source. This means that the brightness of a point on the object depends only on the reflectivity of the object (its albedo) and the angle between the object’s surface and the direction to the light source, according to the following formula (assuming there are no shadows):

$$I(x, y) = A(x, y)\mathbf{N}(x, y) \cdot \mathbf{L}$$

where  $I(x, y)$  is the intensity of pixel  $x, y$ ,  $A(x, y)$  is the albedo of the corresponding point on the object,  $\mathbf{N}(x, y)$  is the normal vector of the object’s surface (relative to a vector pointing toward the camera) and  $\mathbf{L}$  is a vector from the object to the light source, which is assumed to cast parallel rays on the object.

As the light source direction  $\mathbf{L}$  is varied,  $I(x, y)$  also varies, but the surface shape and albedo are fixed. Since the equation is linear, and  $L$  has three parameters, the space of

images of the object (without shadows) is a three dimensional subspace. This subspace can be determined from (at least) example images of the object, by using principal components analysis (PCA). This subspace is related by a linear transformation to the set of normal vectors  $\mathbf{N}(x, y)$ . If we want to recover the true normal vectors, we need to know the actual light source directions. If these directions are available, the system can be treated as an over-constrained set of equations and solved directly for  $\mathbf{N}(x, y)$  without performing principal components analysis. Actually, we will solve for the product  $A(x, y)\mathbf{N}(x, y)$ , but since  $\mathbf{N}(x, y)$  have unit length, it is possible to separate the albedo  $A(x, y)$ .

An example result is shown in Figure 4.4.

With  $A(x, y)$  and  $\mathbf{N}(x, y)$  in hand, which are essentially the color and shape of the face, we can then generate new images of the face under any desired lighting conditions. Some examples of images which can be generated are shown in Figure 4.3.

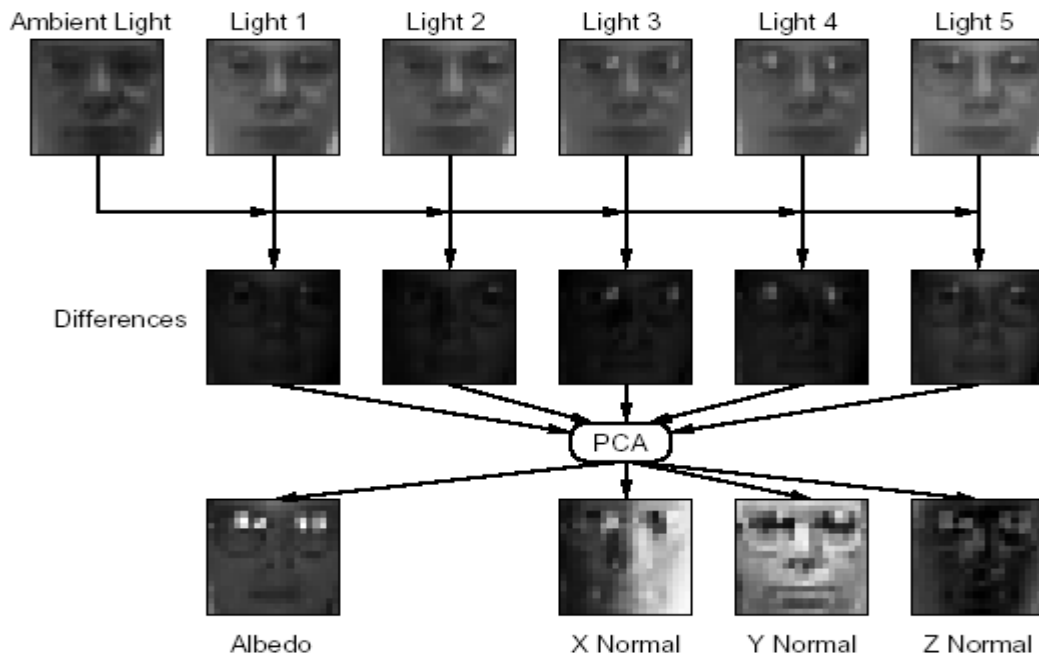
Such images can be used directly for training a face detector, and such experiments will be reported on in the next chapter. It is however quite time consuming to capture images of faces under multiple lighting conditions, and this limits the amount of training data. Ideally, we would like to learn about how images of faces change with different lighting, and apply that to new images of faces, for which we only have single images.

#### **4.4.4 Neural Networks for Compensation**

Given a new input window to be classified as a face or non-face, we would like to apply a lighting correction which will remove any artifacts caused by extreme lighting conditions. This lighting correction must not change faces to non-faces and vice-versa. The architecture we tried is shown in Figure 4.9.

The architecture feeds the input window to a neural network, which has been trained to produce a lighting correction that is an image to add to the input which will make the lighting appear that it is coming from the front of the face. Some example training data is shown in Figure 4.10.

This data was prepared using the lighting models described above. This lighting correction is then added back into the original input window to get the corrected window. To prevent the neural network from applying arbitrary corrections (which could turn any non-face into a face), the network architecture contains a bottleneck, forcing the network to parameterize the correction using only four activation levels. The output layer essentially computes a linear combination of four images based on these activations.



**Figure 4.4:** Example images under different lighting conditions, such as these, allow us to solve for the normal vectors on a face and its albedo.

Some results from this system for faces and non-faces are shown in Figure 4.9. As can be seen, most of the results for faces are quite good (one exception is the fifth face from the left). Most of the strong shadows are removed, and the brightness levels of all parts of the face are similar. However, the results for non-faces are troubling. Many of the non-faces now look very face-like. The reason for this can be seen by considering the types of corrections that must be performed. When the lighting is very extreme, say from the left side of the face, the right side of the face will have intensity values of zero. Thus the corrector must “construct” the entire right half of the face. This construction capability makes it create faces when given relatively uniform non-faces as input.



One potential solution to this problem would be to measure how much work the lighting correction network had to do. If it made large changes in the image, then the result of the face detector applied to that window should be more suspect. This has not yet been explored.



**Figure 4.3:** Generating example images under variable lighting conditions.

### 4.4.3 Quotient Images for Compensation

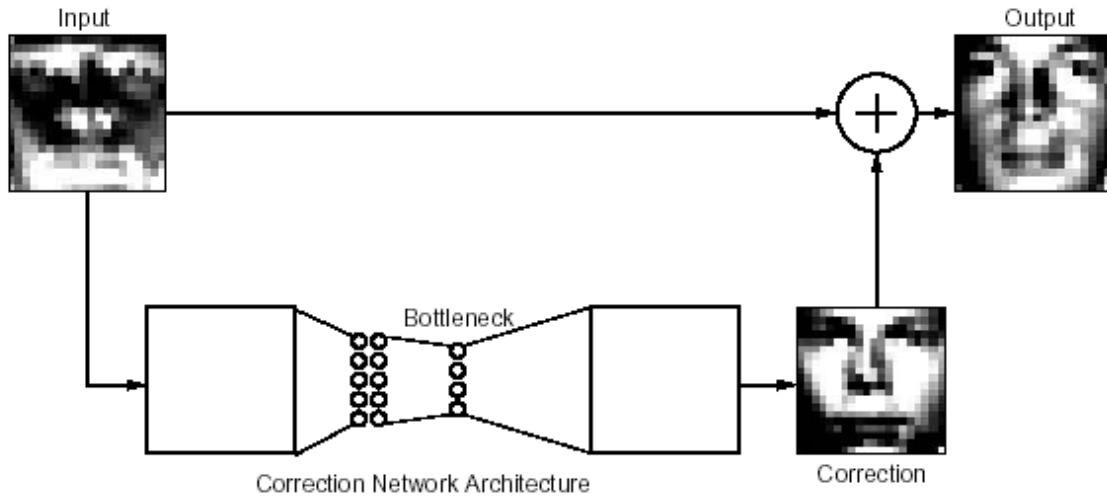
The idea in this work is again to use linear lighting models. A technique is used where an input image can be simultaneously projected into the linear lighting spaces of a set of linear models. The simultaneous projection finds the  $\mathbf{L}$  which minimizes the following quantity:

$$\sum_{i=1}^n \sum_{x,y} (I(x,y) - A_i(x,y)(\mathbf{N}_i(x,y) \cdot \mathbf{L}))^2$$

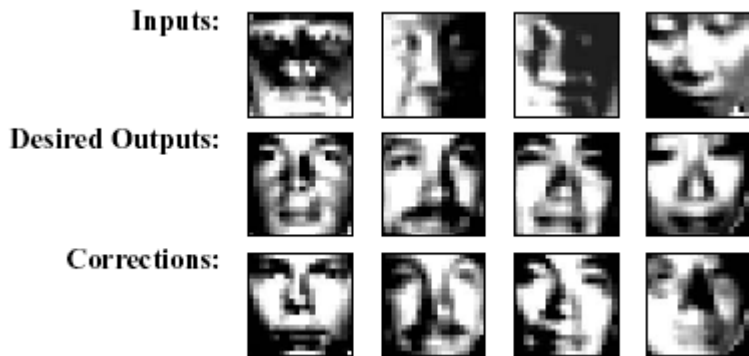
Where  $I(x,y)$  is the input image,  $i$  is summed over all  $n$  lighting models, and, and  $A_i(x,y)$  and  $\mathbf{N}_i(x,y)$  are the corresponding albedo and normal vectors for lighting model  $i$  at pixel

$(x, y)$ . The result of this optimization is a vector  $\mathbf{L}$  representing the lighting conditions for the face in the input image. Using a set of linear models allows for some robustness to differences in the albedos and shapes of individual faces. Using the collection of face lighting models, they then compute an image of the average face under the same conditions using the following equation:

$$\frac{1}{n} \sum_{i=1}^n A_i(x, y) (N_i(x, y) \cdot \mathbf{L})$$



**Figure 4.9:** Architecture for correcting the lighting of a window. The window is given to a neural network, which has a severe bottleneck before its output. The output is a correction to be added to the original image.

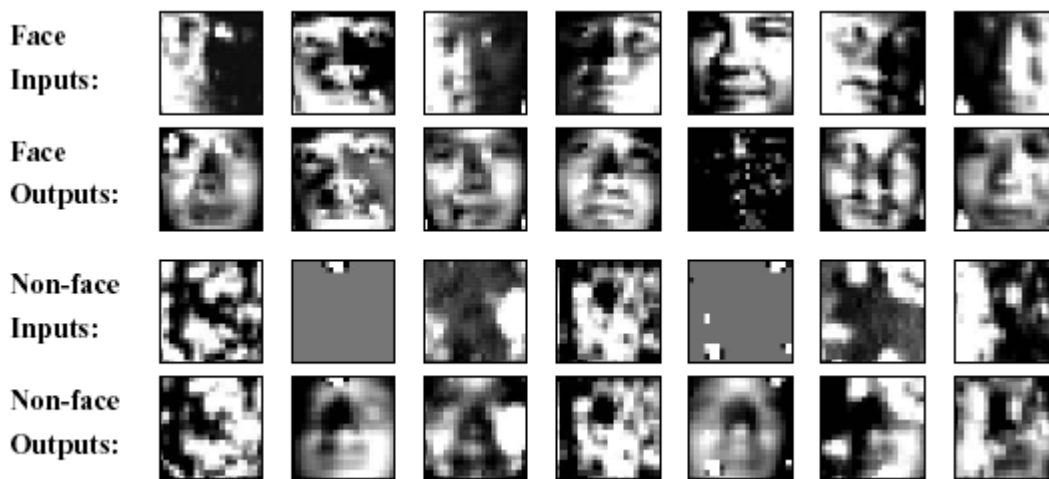


**Figure 4.10:** Training data for the lighting correction neural network.

The input image is divided by this synthetic image, yielding a so called “quotient image”. Mathematically, the quotient image contains only the ratio of the albedos of the new face and the average face, assuming that the faces have similar shapes.

The original work on this technique used the quotient image for face recognition, because it removes the effects of lighting and allows recognition with fewer example images. The same approach can be used to normalize the lighting of input windows for face detection. Instead of just dividing by the average face under the estimated lighting conditions, we can go a step further, multiplying by the average face under frontal lighting:

$$I'(x, y) = I(x, y) \frac{\sum_{i=1}^n A_i(x, y) (N_i(x, y) \cdot L)}{\sum_{i=1}^n A_i(x, y) (N_i(x, y) \cdot (1 \ 0 \ 0))}$$



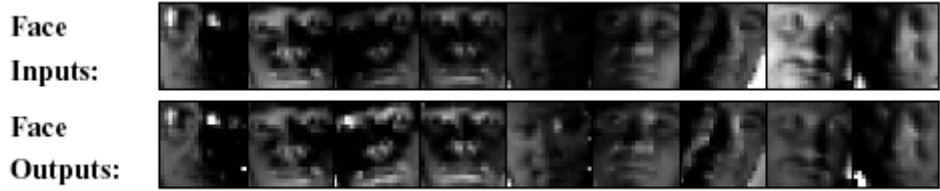
**Figure 4.11:** Result of lighting correction system. The lighting correction results for most of the faces are quite good, but some of the non-faces have been changed into faces.

This should ideally give an image of the original face but with frontal lighting. Some examples are shown in Figure 4.40. It is not clear that this approach will work well for face detection. As can be seen, while the overall intensity has been roughly normalized, the brightness differences across the face have not been improved. In some cases, bright spots have been introduced into the output image, probably because of the specular reflections in the images used to build the basis for the face images. Finally, since the lighting model does not incorporate shadows, the shadows cast by the nose or brow will cause problems.

## 4.5 Summary

The first part of this chapter described the training and test databases used throughout this project. The major focus however was some methods for segmenting face regions from

training images, aligning faces with one another, and preprocessing them to improve contrast. The chapter ended with some speculative results on how to intelligently correct for extreme lighting conditions in images. Together these techniques will be used to generate training data for the detectors to be described later.



**Figure 4.14:** Result of using quotient images to correct lighting.

# Chapter 5

## Upright Face Detection

### 5.1 Introduction

In this chapter, a neural network-based algorithm is presented to detect upright, frontal views of faces. The algorithm works by applying one or more neural networks directly to portions of the input image, and arbitrating their results. Each network is trained to output the presence or absence of a face.

Training a neural network for the face detection task is challenging because of the difficulty in characterizing prototypical “non-face” images. Unlike face *recognition*, in which the classes to be discriminated are different faces, the two classes to be discriminated in face *detection* are “images containing faces” and “images not containing faces”. It is easy to get a representative sample of images which contain faces, but much harder to get a representative sample of those which do not. We avoid the problem of using a huge training set for non-faces by selectively adding images to the training set as training progresses. This “bootstrap” method reduces the size of the training set needed. The use of arbitration between multiple networks and heuristics to clean up the results significantly improves the accuracy of the detector.

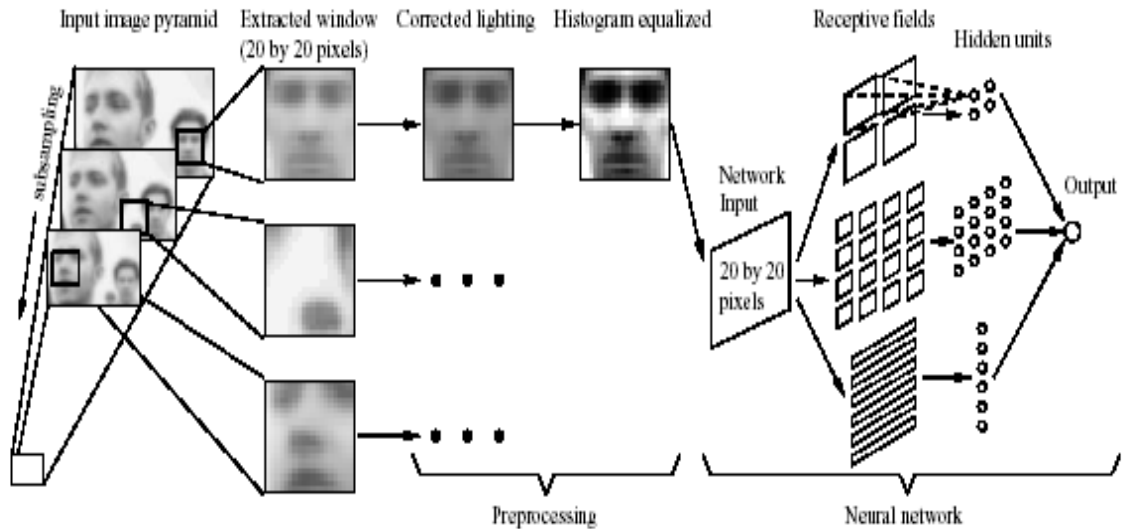
The architecture of the system and training methods for the individual neural networks which make up the detector are presented in Section 5.2. Section 5.5 examines how these individual networks behave, by measuring their sensitivity to different parts of the input image, and measuring their performance on some test images. Methods to clean up the

results and to arbitrate among multiple networks are presented in Section 5.4. The results in Section 5.5 show that the system is able to detect 90.5% of the faces over a test set of 150 complex images, with an acceptable number of false positives.

## 5.2 Individual Face Detection Networks

The system operates in two stages: it first applies a set of neural network-based detectors to an image, and then uses an arbitrator to combine the outputs. 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 eliminates overlapping detections.

The first component of our system is a neural network that receives as input a  $20 \times 20$  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. We apply the network at every pixel position in the image, and scale the image down by a factor of 1.2 for each step in the pyramid. This image pyramid is shown at the left of Figure 5.1.



**Figure 5.1:** The basic algorithm used for face detection.

After a  $20 \times 20$  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 2.5. 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 5.1. The input window is broken down into smaller pieces, of four  $10 \times 10$  pixel regions, sixteen  $5 \times 5$  pixel regions, and six overlapping  $20 \times 5$  pixel regions. Each of these regions will have complete connections to a hidden unit, as shown in the figure. Although the figure shows a single hidden unit for each subregion of the input, these units can be replicated. For the experiments which are described later, we use networks with two and three sets of these hidden units. The shapes of these subregions were chosen to allow the hidden units to detect local features that might be important for face detection. In particular, the horizontal stripes allow the hidden units to detect such features as mouths or pairs of eyes, while the hidden units with square receptive fields might detect features such as individual eyes, the nose, or corners of the mouth. Similar input connection patterns are commonly used in speech and character recognition tasks. The network has a single, real-valued output, which indicates whether or not the window contains a face.

### **5.2.1 Face Training Images**

In order to use a neural network to classify windows as faces or non-faces, we need training examples for each set. For positive examples, we use the techniques presented in Section 2.5 to align example face images in which some feature points have been manually labelled. After alignment, the faces are scaled to a uniform size, position, and orientation within a  $20 \times 20$  pixel window. The images are scaled by a random factor and translated by a random amount up to 0.5 pixels. This allows the detector to be applied at each pixel location and at each scale in the image pyramid, and still detect faces at intermediate locations or scales. In addition, to give the detector some robustness to slight variations in the faces, they are rotated by a random amount (up to 10 degrees). In our experiments, using larger amounts of rotation to train the detector network yielded too many false positive to be usable. There are a total of 1046 training examples in our training set, and 15 of these randomized training examples are generated for each original face. The next sections describe methods for collecting negative examples and training.

### **5.2.2 Non-Face Training Images**

We needed a large number of non-face images to train the face detector, because the variety of non-face images is much greater than the variety of face images. One large class of images which do not contain any faces are pictures of scenery, such as trees, mountains, and buildings.

Collecting a “representative” set of non-faces is difficult. Practically any image can serve as a non-face example; the space of non-face images is much larger than the space of face images. The statistical approach to machine learning suggests that we should train the neural networks on precisely the same distribution of images which it will see at runtime. For our face detector, the number of face examples is 15,000, which is a practical number. However, our representative set of scenery images contains approximately 150,000,000 windows, and training on a database of this size is very difficult. The next two sections describe two approaches to training with this amount of data.



## 5.2.5 Active Learning

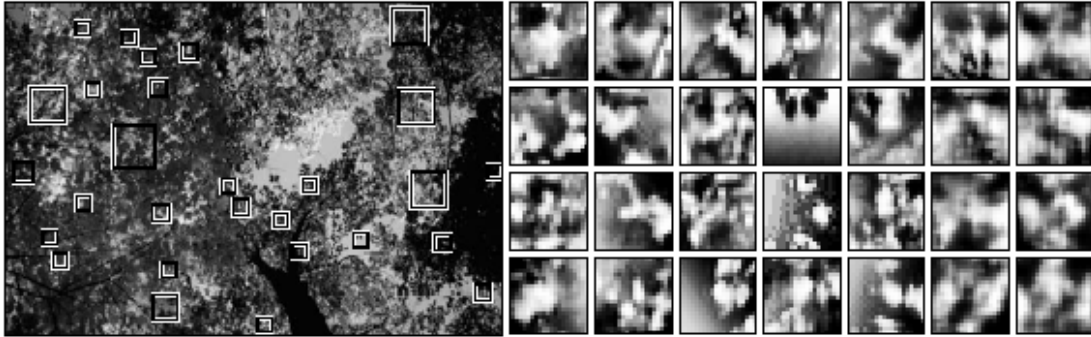
Because of the difficulty of training with every possible negative example, we utilized an algorithm. Instead of collecting the images before training is started, the images are collected during training, in the following manner:

1. Create an initial set of non-face images by generating 1000 random images. Apply the preprocessing steps to each of these images.
2. Train a neural network to produce an output of 1 for the face examples, and -1 for the non-face examples. On the first iteration of this loop, the network's weights are initialized randomly. After the first iteration, we use the weights computed by training in the previous iteration.
5. Run the system on an image of scenery *which contains no faces*. Collect subimages in which the network incorrectly identifies a face (an output activation  $> 0$ ).
4. Select up to 250 of these subimages at random, apply the preprocessing steps, and add them into the training set as negative examples. Go to Step 2.

The training algorithm used in Step 2 is the standard error backpropagation algorithm. The neurons use the  $\tanh$  activation function, which gives an output ranging from -1 to 1, hence the threshold of 0 for the detection of a face.

Since the number of negative examples is much larger than the number of positive examples, uniformly sampled batches of training examples would often contain only negative examples, which would be inappropriate for neural network training. Instead, each batch of 100 positive and negative examples is drawn randomly from the entire training sets, and passed to the backpropagation algorithm as a batch. We choose the training batches such that they have 50% positive examples and 50% negative examples. This ensures that initially, when we have a much larger set of positive examples than negative examples, the network will actually learn something from both sets. Note that this changes

the distribution of faces and non-faces in the training sets compared with what the network will see at run time.



**Figure 5.2:** During training, the partially-trained system is applied to images of scenery which do not contain faces (like the one on the left). Any regions in the image detected as faces (which are expanded and shown on the right) are errors, which can be added into the set of negative training examples.

Some examples of non-faces that are collected during training are shown in Figure 5.2. Note that some of the examples resemble faces, although they are not very close to the positive examples shown in Figure 2.2. The presence of these examples forces the neural network to learn the precise boundary between face and non-face images. We used 120 images of scenery for collecting negative examples in the bootstrap manner described above.

### 5.2.4 Exhaustive Training

Neural network training usually requires training the network many times on its training images; a single pass through 150,000,000 scenery windows not only requires a huge amount of storage, but also takes nearly a day on a four processor SGI supercomputer. Additionally, a network usually rains on images in batches of about 100 images; by the time we reach the end of 150,000,000 examples, it will have forgotten the characteristics of first images. As in the previous section, to insure that the the neural network learns about both faces and non-faces, we select the batches of negative examples to have approximately equal numbers of positive and negative examples. However, this changes

the apparent distribution of positive and negative examples, so that it no longer matches the real distribution.

It is possible to compensate for this using Bayes' Theorem. If we denote  $P(\text{face}|\text{window})$  as the probability that a given window is a face, and  $P'(\text{face})$  and  $P'(\text{non-face})$  as the prior probability of faces and non-faces in the training sets (both 0.5), then Bayes' Theorem says:

$$\text{NN Output} = P'(\text{face}|\text{window}) = \frac{P(\text{window}|\text{face}) \cdot P'(\text{face})}{P(\text{window}|\text{face}) \cdot P'(\text{face}) + P(\text{window}|\text{nonface}) \cdot P'(\text{nonface})}$$

Neural networks will learn to estimate the left hand side of this equation, and since we know  $P'(\text{face})$ ,  $P'(\text{non-face})$ , and that  $P(\text{window}|\text{non-face}) = 1 - P(\text{window}|\text{face})$ , this equation simplifies, giving:

$$P(\text{window}|\text{face}) = \text{NN Output}$$

Let us denote the true probability of faces is  $P(\text{face})$ , and non-faces is  $P(\text{non-face})$ . Then we can use Bayes' Theorem in the forward direction to get the true probability of a face given the image:

$$P(\text{face}|\text{window}) = \frac{\text{NN Output} \cdot P(\text{face})}{\text{NN Output} \cdot P(\text{face}) + (1 - \text{NN Output}) \cdot P(\text{nonface})}$$

We would like to classify a window as a face if  $P(\text{face}|\text{window}) > 0.5$ , which is equivalent to setting a threshold of:

$$\text{NN Output} > 1 - P(\text{face})$$

Since we are using neural networks with  $\tanh$  activation functions, the output range is -1 to 1, so this threshold is adjusted as follows:

$$\text{NN Output} > 1 - 2P(\text{face})$$

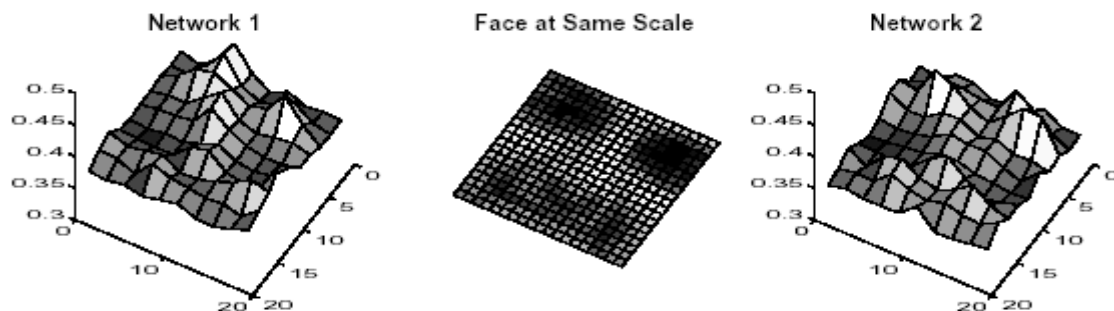
Thus we need to determine the prior probability of faces, which will be discussed in Section 5.5.2.

## 5.3 Analysis of Individual Networks

This section presents some analysis of the performance of the networks described above, beginning with a sensitivity analysis, then examining the performance on the *Upright Test Set*.

### 5.31 Sensitivity Analysis

In order to determine which part of its input image the network uses to decide whether the input is a face, we performed sensitivity analysis. We collected a positive test set based on the training database of face images, but with different randomized scales, translations, and rotations than were used for training. The negative test set was built from a set of negative examples collected during the training of other networks. Each of the  $20 \times 20$  pixel input images was divided into 100  $2 \times 2$  pixel subimages. For each subimage in turn, we went through the test set, replacing that subimage with random noise, and tested the neural network. The resulting root mean square error of the network on the test set is an indication of how important that portion of the image is for the detection task. Plots of the error rates for two networks we trained are shown in Figure 5.5. Network 1 uses two sets of the hidden units illustrated in Figure 5.1, while Network 2 uses three sets.

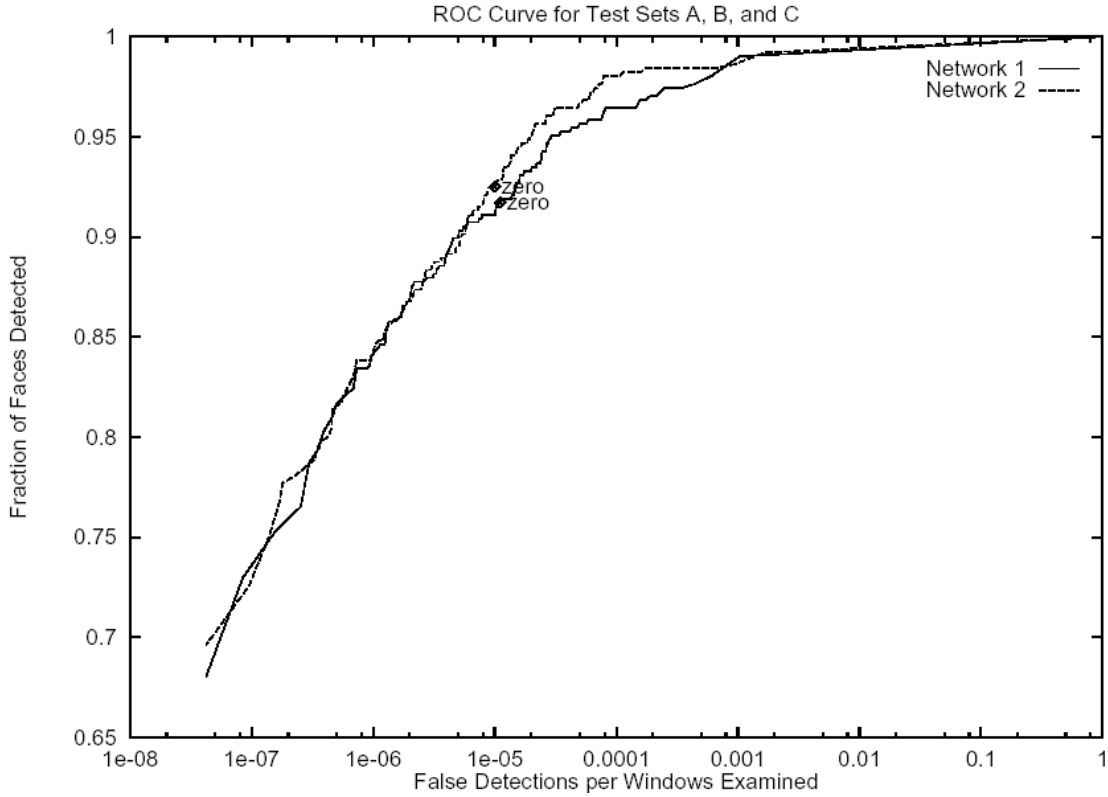


**Figure 5.5:** Error rates (vertical axis) on a test created by adding noise to various portions of the input image (horizontal plane), for two networks. Network 1 has two copies of the hidden units shown in Figure 5.1 (a total of 55 hidden units and 2905 connections), while Network 2 has three copies (a total of 25 hidden units and 4552 connections).

The networks rely most heavily on the eyes, then on the nose, and then on the mouth (Figure 5.5). We have seen this behavior on several real test images. In cases in which only one eye is visible, detection of a face is possible, though less reliable, than when the entire face is visible. The system is less sensitive to the occlusion of the nose or mouth.

### **5.3.2 ROC (Receiver Operator Characteristic) Curves**

The outputs from our face detection networks are not binary. The neural networks produce real values between 1 and -1, indicating whether or not the input contains a face. A threshold value of zero is used during *training* to select the negative examples (if the network outputs a value of greater than zero for any input from a scenery image, it is considered a mistake). Although this value is intuitively reasonable, by changing this value during *testing*, we can vary how conservative the system is. To examine the effect of this threshold value during testing, we measured the detection and false positive rates as the threshold was varied from 1 to -1. At a threshold of 1, the false detection rate is zero, but no faces are detected. As the threshold is decreased, the number of correct detections will increase, but so will the number of false detections.



**Figure 5.4:** The detection rate plotted against false positive rates as the detection threshold is varied from -1 to 1, for the same networks as Figure 5.5. The performance was measured over all images from the Upright Test Set. The points labelled “zero” are the zero threshold points which are used for all other experiments.

This tradeoff is presented in Figure 5.4, which shows the detection rate plotted against the number of false positives as the threshold is varied, for the two networks presented in the previous section. This is measured for the images in the *Upright Test Set*, which consists 150 images with 502. The false positive rate is in terms of the number of  $20 \times 20$  pixel windows that must be examined. This number can be approximated from the number of pixels in the image and the scale factor between different resolutions in the image pyramid (1.2):

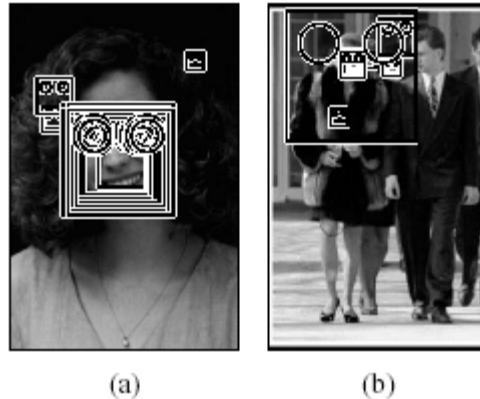
$$\text{number of windows} \approx \text{width} \cdot \text{height} \cdot \left( \sum_{l=0}^{\infty} (1.2 \cdot 1.2)^{-l} \right) = \frac{\text{width} \cdot \text{height}}{1 - 1.2^{-2}} \approx 3.27 \cdot \text{width} \cdot \text{height}$$

To give an intuitive idea about the meaning of the numbers in Figure 5.4 (with a zero threshold), some examples of the output on the two images in Figure 5.5 are shown in Figure 5.6. In the figure, each box represents the position and size of a window to which Network 1 gave a positive response. The network has some invariance to position and scale, which results in multiple boxes around some faces. Note also that there are quite a few false detections; the next section presents some methods to reduce them.

The above analysis can be used with the probabilistic analysis in Section 5.2.4 to determine the threshold for detecting faces in that scheme. Suppose that for a true face, windows one pixel either side of its location, and windows either side of its scale can be detected, then each face contributes about  $5 \cdot 5 \cdot 5 = 22$  face windows. In the training database, there are 1046 faces ( $22 \times 1046 = 25242$  face windows) and 592,624,545  $20 \times 20$  windows, giving a probability of faces equal to  $1/20954$ . This is the value that will be used later in testing.



**Figure 5.5:** Example images on to test the output of the upright detector.



**Figure 5.6:** Images from Figure 5.5 with all the above threshold detections indicated by boxes. Note that the circles are drawn for illustration only, they do not represent detected eye locations.

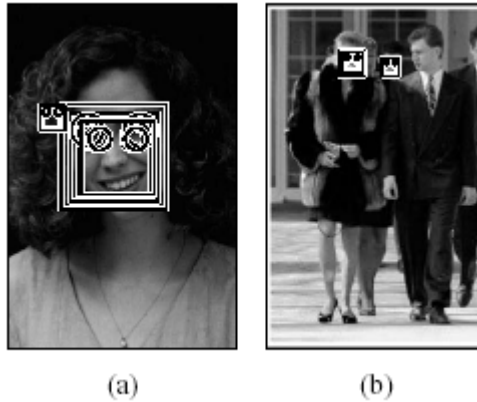
## 5.4 Refinements

The examples in Figure 5.6 showed that the raw output from a single network will contain a number of false detections. In this section, we present two strategies to improve the reliability of the detector: cleaning-up the outputs from an individual network, and arbitrating among multiple networks

### 5.4.1 Clean-Up Heuristics

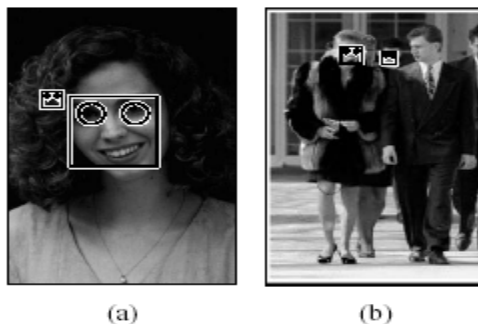
Note that in Figure 5.6a, the face is detected at multiple nearby positions or scales, while false detections often occur with less consistency. The same is true of Figure 5.6b, but since the faces are smaller the overlapping detections are not visible. These observations lead to a heuristic which can eliminate many false detections. For each detection, the number of other detections within a specified neighborhood of that detection 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(size,level)*, where *size* is the size of the neighborhood, in both pixels and pyramid steps, on either side of the detection in question, and *level* is the total number of detections which must appear in that neighborhood. The result of applying *threshold(4,2)* to the images in Figure 5.6 is shown in Figure 5.2.



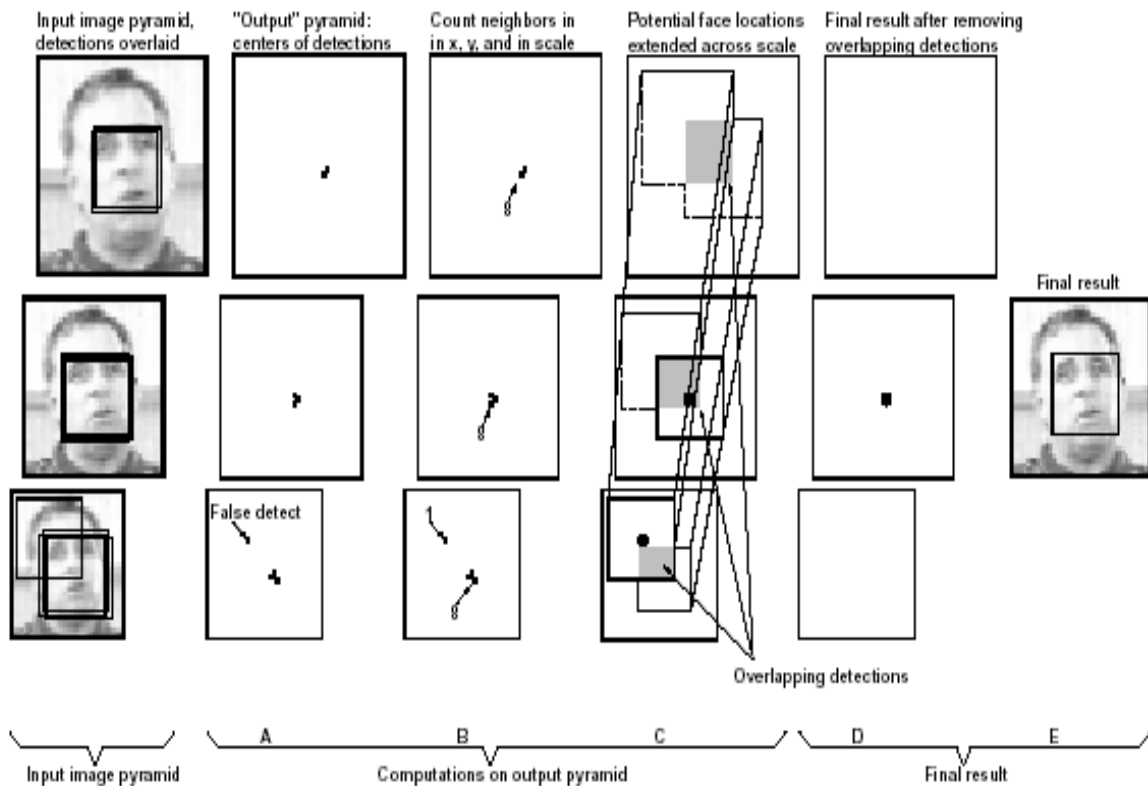


**Figure 5.2:** Result of applying  $threshold(4,2)$  to the images in Figure 5.6.

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. In the discussion of the experiments, this heuristic is called *overlap*. There are relatively few cases in which this heuristic fails; however, one such case is illustrated by the left two faces in Figure 5.5b, where one face partially occludes another, and so is lost when this heuristic is applied. These arbitration heuristics are very similar to, but computationally less expensive than, those presented in my previous paper [Rowley *et al.*, 1995].



**Figure 5.5:** Result of applying *overlap* to the images in Figure 5.2.



**Figure 5.9:** The framework for merging multiple detections from a single network: A) The detections are recorded in an “output” pyramid. B) The number of detections in the neighborhood of each detection are computed. C) The final step is to check the proposed face locations for overlaps, and D) to remove overlapping detections if they exist. In this example, removing the overlapping detection eliminates what would otherwise be a false positive.

The implementation of these two heuristics is illustrated in Figure 5.9. Each detection at a particular location and scale is marked in an image pyramid, called the “output” pyramid. Then, each detection is replaced by the number of detections within its neighborhood. A threshold is applied to these values, and the centroids (in both position and scale) of all above threshold detections are computed (this step is omitted in Figure 5.9. Each centroid is then examined in order, starting from the ones which had the highest number of detections within the specified neighborhood. If any other centroid locations represent a face overlapping with the current centroid, they are removed from the output pyramid. All remaining centroid locations constitute the final detection result. In the face detection work described in [Burel and Carel, 1994], similar observations about the nature of the outputs were made, resulting in the development of heuristics similar to those described above.

## 5.4.2 Arbitration among Multiple Networks

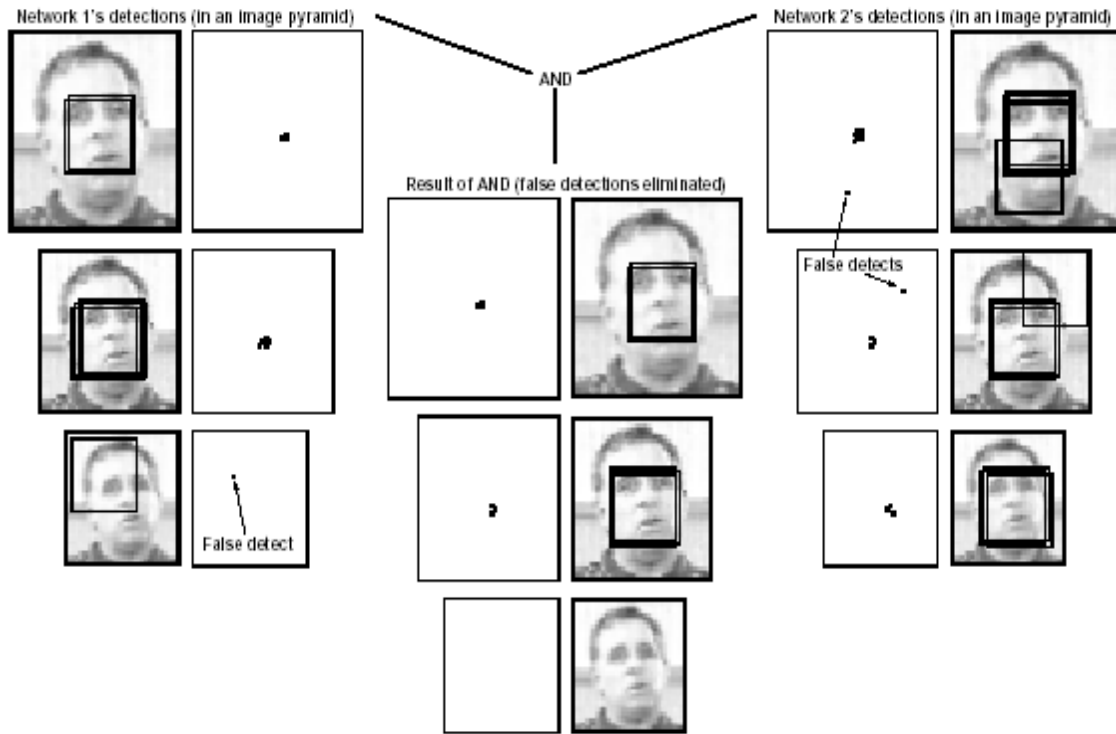
[Sung, 1996] provided some formalization of how a set of identically trained detectors can be used together to improve accuracy. He argued that if the errors made by a detector are independent, then by having a set of networks vote on the result, the number of overall errors will be reduced.

To further reduce the number of false positives, we can apply multiple networks, and arbitrate between their outputs to produce the final decision. Each network is trained using the same algorithm with the same set of face examples, but with different random initial weights, random initial non-face images, and permutations of the order of presentation of the scenery images. As will be seen in the next section, the detection and false positive rates of the individual networks will be quite close. However, because of different training conditions and because of self-selection of negative training examples, the networks will have different biases and will make different errors.

The arbitration algorithm is illustrated in Figure 5.10. Each detection at a particular position and scale is recorded in an image pyramid, as was done with the previous heuristics. One way to combine two such pyramids is by ANDing them. This strategy signals a detection only if both networks detect a face at precisely the same scale and position. Due to the different biases of the individual networks, they will rarely agree on a false detection of a face. This allows ANDing to eliminate most false detections. Unfortunately, this heuristic can decrease the detection rate because a face detected by only one network will be thrown out. However, we will see later that individual networks can all detect roughly the same set of faces, so that the number of faces lost due to ANDing is small.

Similar heuristics, such as ORing the outputs of two networks, or voting among three networks, were also tried. In practice, these arbitration heuristics can all be implemented with variants of the *threshold* algorithm described above. For instance, ANDing can be implemented by combining the results of the two networks, and applying *threshold(0,2)*,

ORing with  $threshold(0,1)$ , and voting by applying  $threshold(0,2)$  to the results of three networks.

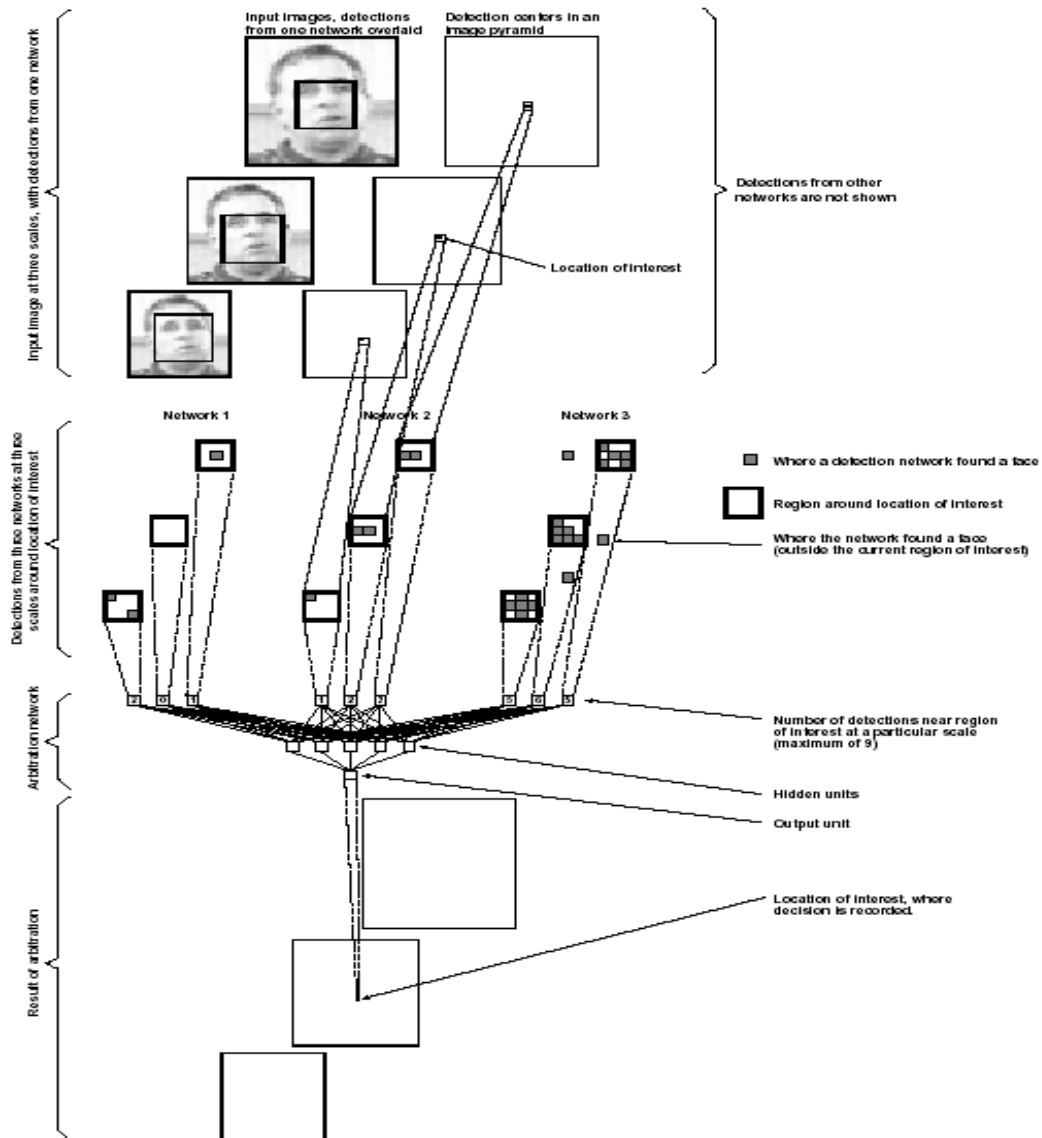


**Figure 5.10:** ANDing together the outputs from two networks over different positions and scales can improve detection accuracy.

Each of these arbitration methods can be applied before or after the clean-up heuristics. If applied afterwards, we combine the centroid locations rather than actual detection locations, and require them to be within some neighborhood of one another rather than precisely aligned, by setting the *size* parameter of the *threshold* which implements the arbitration to a 4 rather than 0. These are denoted  $AND(4)$  and  $AND(0)$  in the experiments.

Arbitration strategies such as ANDing, ORing, or voting seem intuitively reasonable, but perhaps there are some less obvious heuristics that could perform better. To test this hypothesis, we applied a separate neural network to arbitrate among multiple detection networks, as illustrated in Figure 5.11. For every location, the arbitration network examines a small neighborhood surrounding that location in the output pyramid of each individual network. For each pyramid, we count the number of detections in a  $5 \times 5$  pixel region at each of three scales around the location of interest, resulting in three numbers for each

detector, which are fed to the arbitration network. The arbitration network is trained (using the images from which the positive face examples were extracted) to produce a positive output for a given set of inputs only if that location contains a face, and to produce a negative output for locations without a face. As will be seen in the next section, using an arbitration network in this fashion produced results comparable to (and in some cases, slightly better than) those produced by the heuristics presented earlier, at the expense of extra complexity.



**Figure 5.11:** The inputs and architecture of the arbitration network which arbitrates among multiple face detection networks.

## 5.5 Evaluation

A number of experiments were performed to evaluate the system. We first show an analysis of which features the neural network is using to detect faces, then present the error rates of the system over two large test sets, and finally show some example output.

### 5.5.1 Upright Test Set

The first set of test images is for testing the capabilities of the upright face detector. To evaluate the accuracy of their system, a test database of 25 images from various sources is collected, which we also use for testing purposes.

A number of these images were chosen specifically to test the tolerance to clutter in images, and did not contain any faces. Others contained large numbers of upright, frontal faces, to test the detector's tolerance of different types of faces. A few example images are shown in Figure 5.12. In the following, this test set will be called the *Upright Test Set*.



**Figure 5.12:** Example images from the *Upright Test Set*, used for testing the upright face detector.

Table 5.15 shows the performance of different versions of the detector on the *Upright Test Set*. The four columns show the number of faces missed (out of 502), the detection rate, the total number of false detections, and the false detection rate (compared with the number of  $20 \times 20$  windows examined).

Type	System	Missed faces	Detect rate	False detects	False detect rate
One network, no heuristics	1) Network 1 (2 copies of hidden units (52 total), 2905 connections)	44	91.3%	928	1/89546
	2) Network 2 (3 copies of hidden units (78 total), 4357 connections)	37	92.7%	853	1/97419
	3) Network 3 (2 copies of hidden units (52 total), 2905 connections)	47	90.7%	759	1/109485
	4) Network 4 (3 copies of hidden units (78 total), 4357 connections)	40	92.1%	820	1/101340
One network, with heuristics	5) Network 1 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	50	90.1%	516	1/161044
	6) Network 2 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	44	91.3%	453	1/183441
	7) Network 3 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	51	89.9%	422	1/196917
	8) Network 4 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	42	91.7%	452	1/183847
Arbitrating among two networks	9) Networks 1 and 2 $\rightarrow$ AND(0)	66	87.0%	156	1/532687
	10) Networks 1 and 2 $\rightarrow$ AND(0) $\rightarrow$ threshold(4,3) $\rightarrow$ overlap	92	81.9%	8	1/10387401
	11) Networks 1 and 2 $\rightarrow$ threshold(4,2) $\rightarrow$ overlap $\rightarrow$ AND(4)	71	86.0%	31	1/2680619
	12) Networks 1 and 2 $\rightarrow$ threshold(4,2) $\rightarrow$ overlap $\rightarrow$ OR(4) $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	50	90.1%	167	1/497600
Arbitrating among three networks	13) Networks 1, 2, 3 $\rightarrow$ voting(0) $\rightarrow$ overlap	55	89.2%	95	1/874728
	14) Networks 1, 2, 3 $\rightarrow$ network arbitration (5 hidden units) $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	85	83.2%	10	1/8309921
	15) Networks 1, 2, 3 $\rightarrow$ network arbitration (10 hidden units) $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	86	83.0%	10	1/8309921
	16) Networks 1, 2, 3 $\rightarrow$ network arbitration (perceptron) $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	89	82.4%	9	1/9233245

**Table 5.15:** Detection and error rates for the Upright Test Set, which consists of 150 images and contains 502 frontal faces. It requires the system to examine a total of 55,099,211  $20 \times 20$  pixel windows.

The table begins by showing the results for four individual networks. Networks 1 and 2 are the same as those used in Sections 5.5.1 and 5.5.2. Networks 5 and 4 are identical to Networks 1 and 2, respectively, except that the negative example images were presented in a different order during training. The results for ANDing and ORing networks were based on Networks 1 and 2, while voting and network arbitration were based on Networks 1, 2,

and 5. The neural network arbitrators were trained using the images from which the face examples were extracted. Three different architectures for the network arbitrator were used. The first used 5 hidden units, as shown in Figure 5.11. The second used two hidden layers of 5 units each, with complete connections between each layer, and additional connections between the first hidden layer and the output. The last architecture was a simple perceptron, with no hidden units.

As discussed earlier, the *threshold* heuristic for merging detections requires two parameters, which specify the size of the neighborhood used in searching for nearby detections, and the threshold on the number of detections that must be found in that neighborhood. In the table, these two parameters are shown in parentheses after the word *threshold*. Similarly, the ANDing, ORing, and voting arbitration methods have a parameter specifying how close two detections (or detection centroids) must be in order to be counted as identical.

Systems 1 through 4 in the table show the raw performance of the networks. Systems 5 through 5 use the same networks, but include the *threshold* and *overlap* steps which decrease the number of false detections significantly, at the expense of a small decrease in the detection rate. The remaining systems all use arbitration among multiple networks. Using arbitration further reduces the false positive rate, and in some cases increases the detection rate slightly. Note that for systems using arbitration, the ratio of false detections to windows examined is extremely low, ranging from 1 false detection per 492, 600 windows to down to 1 in 10, 552, 401, depending on the type of arbitration used. Systems 10, 11, and 12 show that the detector can be tuned to make it more or less conservative. System 10, which uses ANDing, gives an extremely small number of false positives, and has a detection rate of about 51.9%. On the other hand, System 12, which is based on ORing, has a higher detection rate of 90.1% but also has a larger number of false detections. System 11 provides a compromise between the two. The differences in performance of these systems can be understood by considering the arbitration strategy. When using ANDing, a false detection made by only one network is suppressed, leading to a lower false positive rate. On the other hand, when ORing is used, faces detected correctly by only one network will be preserved, improving the detection rate.



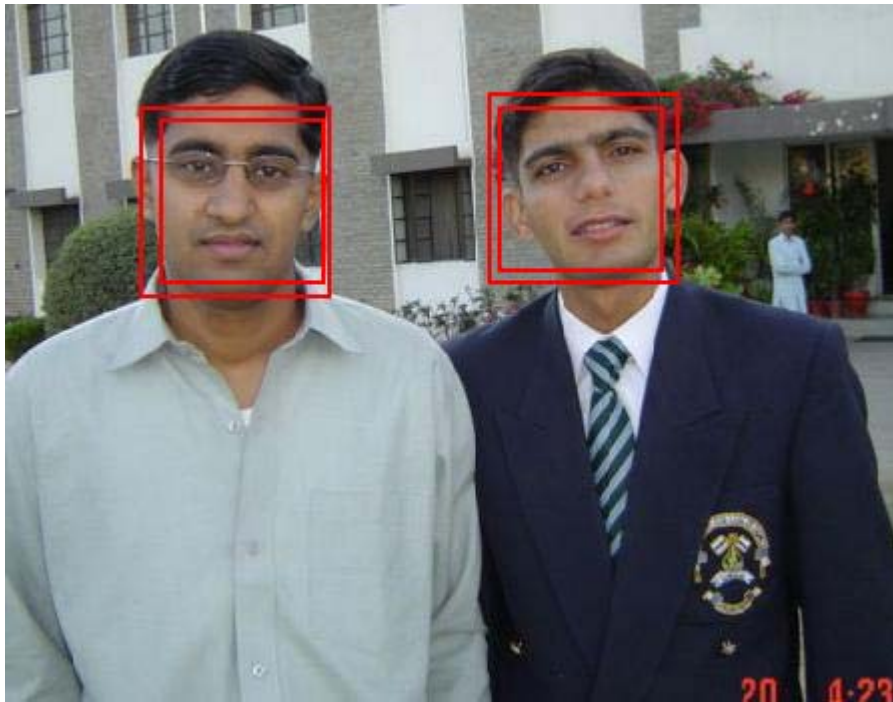
Systems 14, 15, and 16, all of which use neural network-based arbitration among three networks, yield detection and false alarm rates between those of Systems 10 and 11. System 15, which uses voting among three networks, has an accuracy between that of Systems 11 and 12.

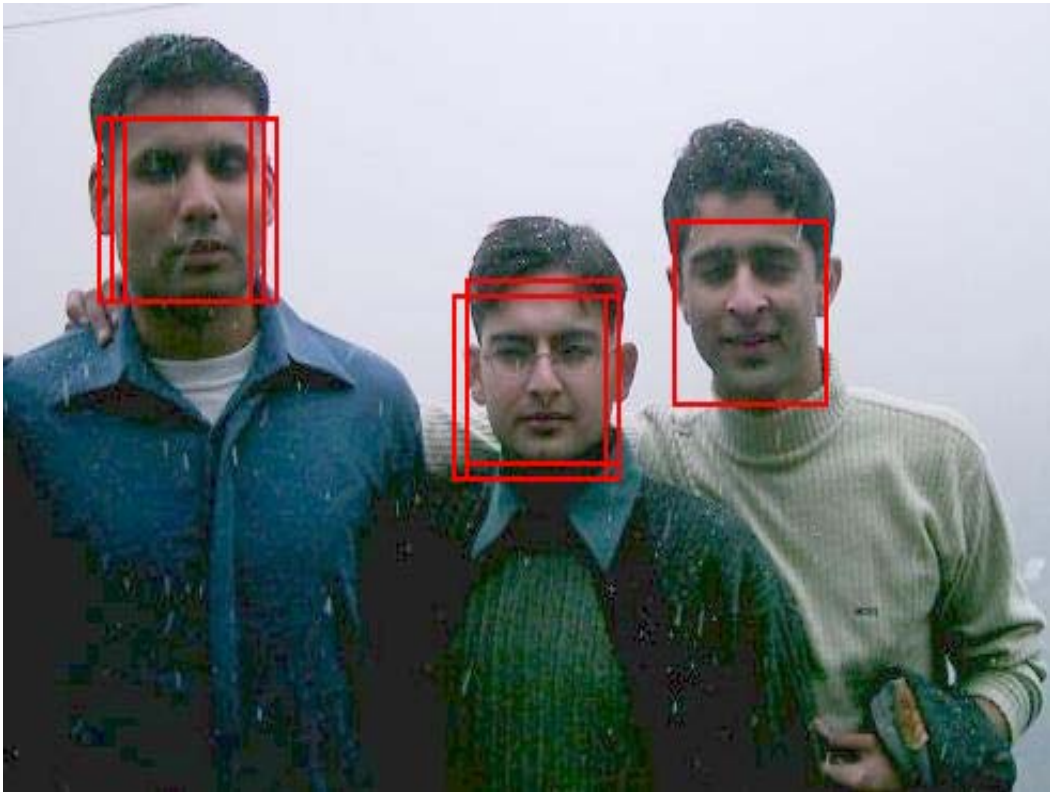
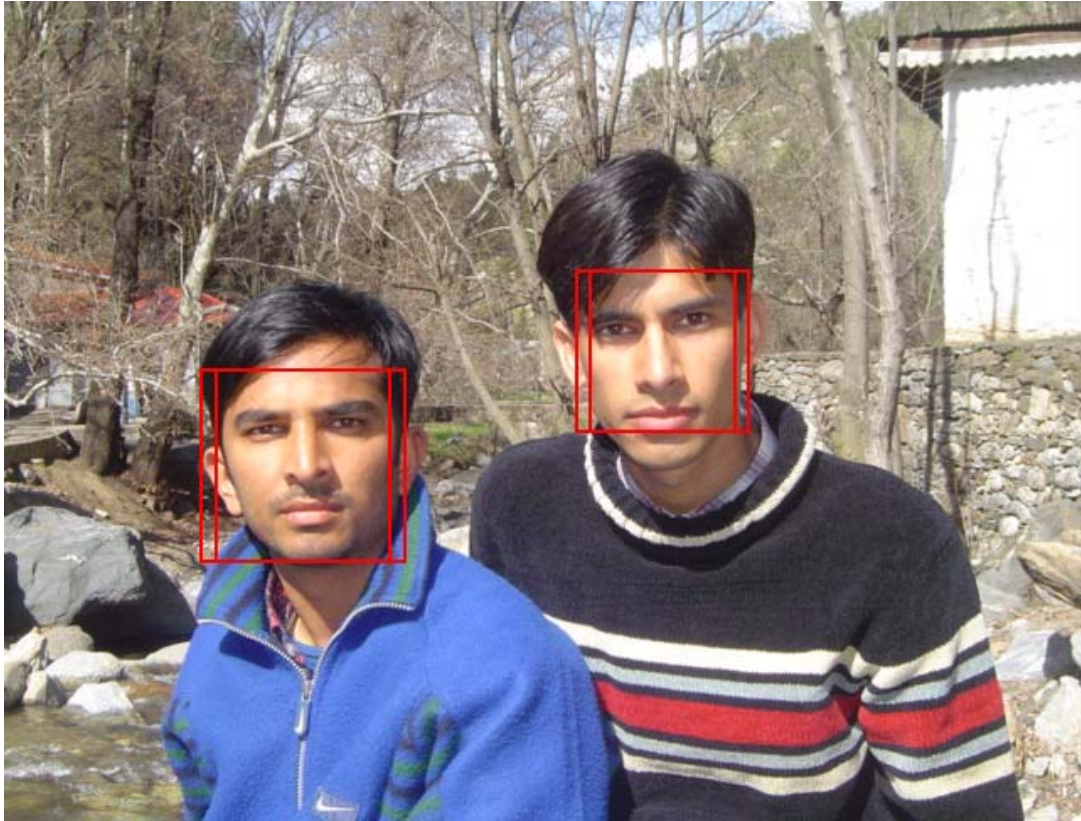
### 5.5.2 Example Output

Figs. 5.16 from System 11 on images from the *Upright Test Set*.

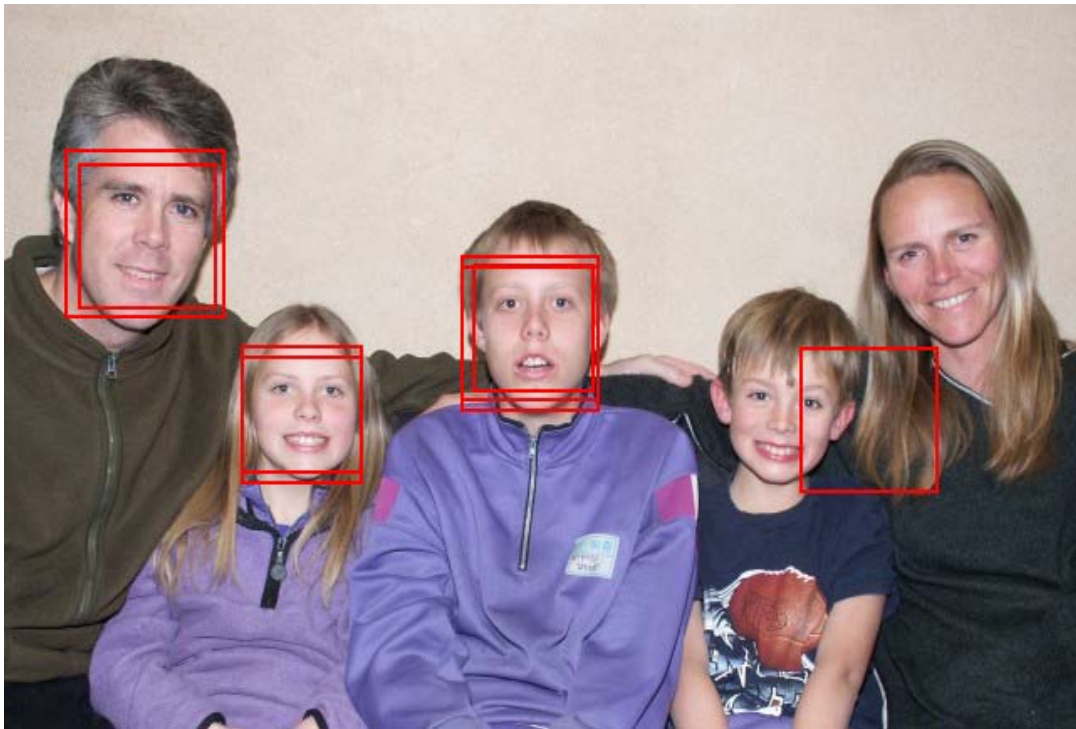
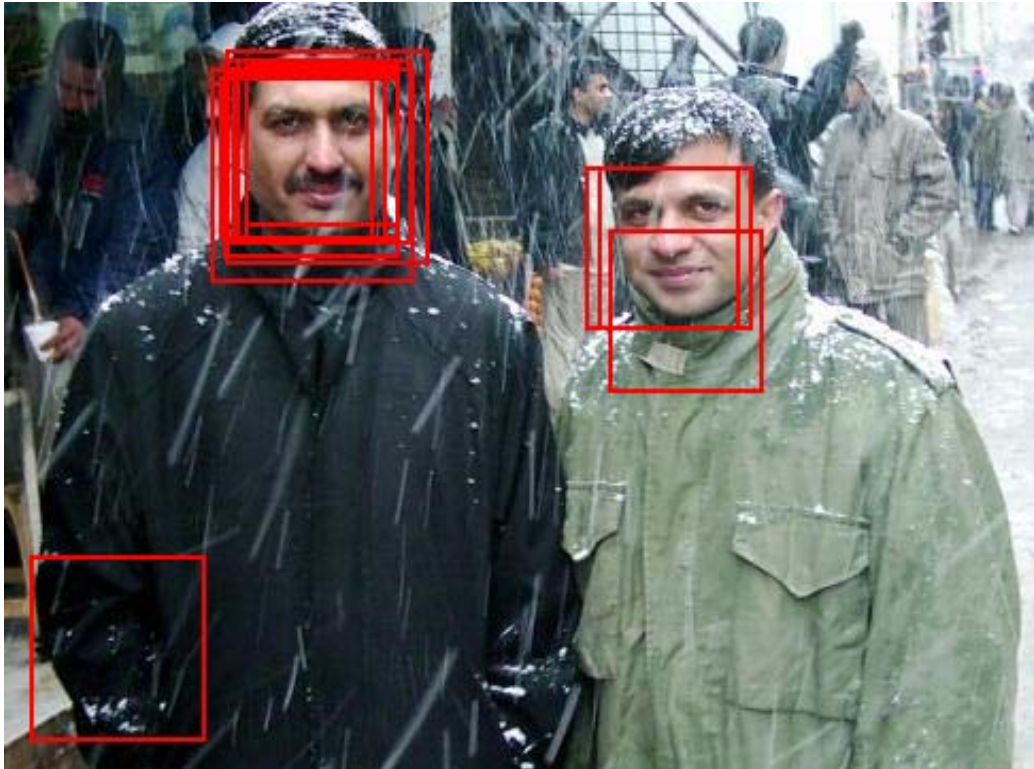
### 5.5.3 Effect of Lighting Variation

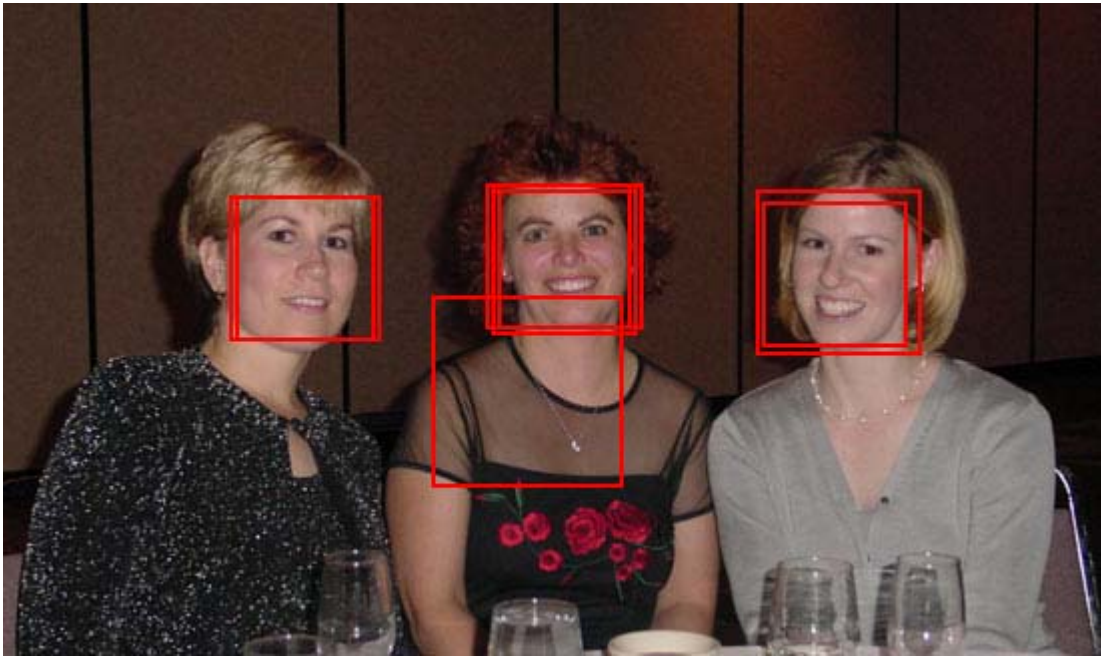
Section 2.6 discussed methods to use linear lighting models of faces to explicitly compensate for variations in lighting conditions before attempting to detect a face. These models can also be used to generate training data for a face detector, so that the neural network can implicitly learn to handle lighting variation.











**Figure 5.16:** Output from System 11 in Table 5.15.

# Chapter 6

## Tilted Face Detection

### 6.1 Introduction

In demonstrating the system described in the previous chapter, the people watching the demonstration would expect faces to be detected at any angle, as shown in Figure 6.1. In this chapter, we present some modifications to the upright face detection algorithm to detect such tilted faces. This system efficiently detects frontal faces which can be arbitrarily rotated within the image plane.



**Figure 6.1:** People expect face detection systems to detect rotated faces.

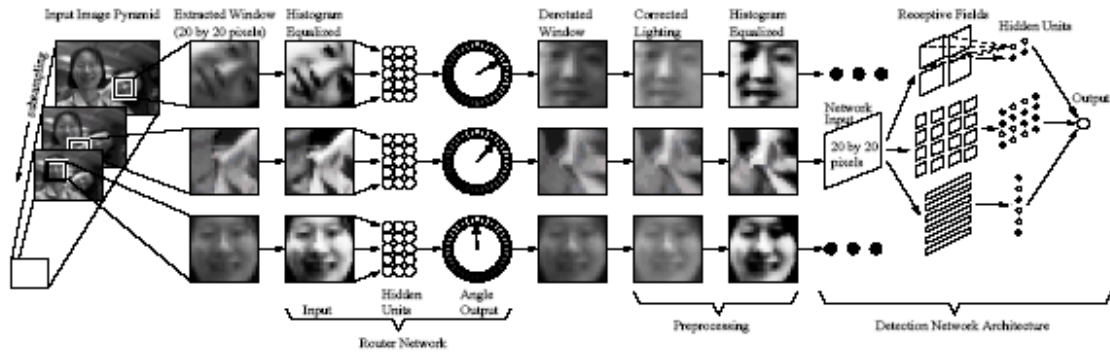
There are many ways to use neural networks for rotated-face detection. The simplest would be to employ the upright face detection, by repeatedly rotating the input image in small increments and applying the detector to each rotated image. However, this would be an extremely computationally expensive procedure. The system described in the previous chapter is invariant to approximately  $10^\circ$  of tilt from upright (both clockwise and counterclockwise). Therefore, the entire detection procedure would need to be applied *at least* 13 times to each image, with the image rotated in increments of  $20^\circ$ .

An alternate, significantly faster procedure uses a separate neural network, termed a “derotation network”, to analyze the input window before it is processed by the face detector. The derotation network’s input is the same region that the detector network will receive as input. If the input contains a face, the derotation network returns the angle of the face. The window can then be “derotated” to make the face upright. Note that the derotation network *does not* require a face as input. If a non-face is encountered, the derotator will return a meaningless rotation. However, since a rotation of a non-face will yield another non-face, the detector network will still not detect a face. On the other hand, a rotated face, which would not have been detected by the detector network alone, will be rotated to an upright position, and subsequently detected as a face. Because the detector network is only applied once at each image location, this approach is significantly faster than exhaustively trying all orientations.

Detailed descriptions of the algorithm are given in Section 6.2. We then analyze the performance of each part of the system separately in Section 6.3, and test the complete system in Section 6.6. We will see that the system is able to detect 29.6% of the faces over the *Upright Test Set* and *Tilted Test Set*, with a very small number of false positives.

## 6.2 Algorithm

The overall structure of the algorithm, shown in Figure 6.2, is quite similar to the one presented in the previous chapter. Starting from the input image, an image pyramid is built, with scaling steps of 1.2.  $20 \times 20$  pixel windows are extracted from every position and scale in this input pyramid, and passed to a classifier.



**Figure 6.2:** Overview of the algorithm.

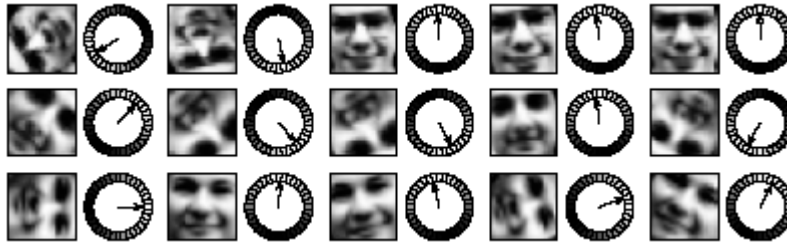
First, the window is preprocessed using histogram equalization (Section 2.5), and then given to a *derotation network*. The tilt angle returned by the derotation network is then used to rotate the window with the potential face to an upright position. Finally, the *derotated window* is preprocessed with linear lighting correction and histogram equalization, and then passed to one or more upright face detection network, like those in the previous chapter, which decide whether or not the window contains a face.

The system as presented so far could easily signal that there are two faces of very different orientations at adjacent pixel locations in the image. To counter such anomalies, and to reinforce correct detections, clean up heuristics and multiple detection networks are employed. The design of the derotation network and the heuristic arbitration scheme are presented in the following subsections.

## 6.2.1 Derotation Network

The first step in processing a window of the input image is to apply the derotation network. This network assumes that its input window contains a face, and is trained to estimate its orientation. The inputs to the network are the intensity values in a  $20 \times 20$  pixel window of the image (which have been preprocessed by histogram equalization, Section 2.5). The output angle of rotation is represented by an array of 36 output units, in which each unit  $i$  represents an angle of  $i * 10^\circ$ . To signal that a face is at an angle of  $\theta$ , each output is trained to have a value of  $\cos(\theta - i * 10^\circ)$ . Examples of the training data are given in Figure 6.3.





**Figure 6.3:** Example inputs and outputs for training the derotation network.

Previous algorithms using Gaussian weighted outputs inferred a single value from them by computing an average of the positions of the outputs, weighted by their activations. For angles, which have a periodic domain, a weighted sum of angles is insufficient. Instead, we interpret each output as a weight for a vector in the direction indicated by the output number  $i$ , and compute a weighted sum as follows:

$$\left( \sum_{i=0}^{35} \text{output}_i * \cos(i * 10^\circ), \sum_{i=0}^{35} \text{output}_i * \sin(i * 10^\circ) \right)$$

The direction of this average vector is interpreted as the angle of the face.

As with the upright face detector, the training examples are generated from a set of manually labelled example images containing 1063 faces. After each face is aligned to the same position, orientation, and scale, they are rotated to a random known orientation to generate the training example. Note that the training examples for the upright detector had small random variations in scale and position for robustness; the derotation network performed better without these variations.

The architecture for the derotation network consists of four layers: an input layer of 600 units, two hidden layers of 15 units each, and an output layer of 36 units. Each layer is fully connected to the next. Each unit uses a hyperbolic tangent activation function, and the network is trained using the standard error backpropagation algorithm.

## 6.2.2 Detector Network



After the derotation network has been applied to a window of the input, the window is derotated to make any face that may be present upright. Because the input window for the derotation network and detection network are both  $20 \times 20$  square windows, and are an angle with respect to one another, their edges may not overlap. Thus the derotation must resample the original input image.

The remaining task is to decide whether or not the window contains an upright face. For this step, we used the algorithm presented in the previous chapter. The resampled image, is preprocessed using the linear lighting correction and histogram equalization procedures described in Section 2.5. The window is then passed to the detector, which is trained to produce 1 for faces, and  $-1$  for non-faces. The detector has two sets of training examples: images which are faces, and images which are not. The positive examples are generated in a manner similar to that of the derotation network; however, the amount of rotation of the training images is limited to the range  $-10^\circ$  to  $10^\circ$ .

Some examples of non-faces that are collected during training were shown in Figure 3.2. At runtime, the detector network will be applied to images which have been derotated, so it may be advantageous to collect negative training examples from the set of derotated non-face images, rather than only non-face images in their original orientations. In Section 6.6, both possibilities are explored.

### 6.2.3 Arbitration Scheme

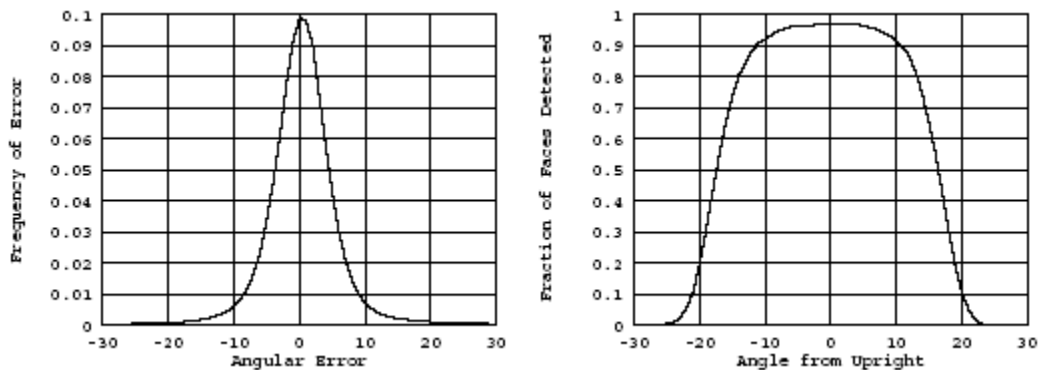
As mentioned earlier, it is possible for the system described so far to signal faces of very different orientations at adjacent pixel locations. As with the upright detector, we use some simple cleanup and arbitration heuristics to improve the results. These heuristics are restated below, with the changes necessary for handling rotation angles in addition to positions and scales. Each detection is first placed in a 6-dimensional space, where the dimensions are the  $x$  and  $y$  positions of the center of the face, the scale in the image pyramid at which the face was detected, and the angle of the face, quantized in increments of  $10^\circ$ . For each detection, we count the number of detections within 6 units along each dimension (6 pixels, 6 pyramid scales, or  $60^\circ$ ). This number can be interpreted as a confidence measure, and a threshold is applied. As before, this heuristic is denoted

*threshold(distance,level)*. Once a detection passes the threshold, any other detections in the 6-dimensional space which would overlap it are discarded. This step is called *overlap* in the experiments section.

To further reduce the number of false detections, and reinforce correct detections, we arbitrate between two independently trained detector networks. To use the outputs of these two networks, the postprocessing heuristics of the previous paragraph are applied to the outputs of each individual network, and then the detections from the two networks are ANDed. The specific preprocessing thresholds used in the experiments will be given in Sections 6.6.

### 6.3 Analysis of the Networks

In order for the system described above to be accurate, the derotator and detector must perform robustly and compatibly. Because the output of the derotator network is used to normalize the input for the detector, the angular accuracy of the derotator must be compatible with the angular invariance of the detector. To measure the accuracy of the derotator, we generated test example images based on the training images, with angles between  $-30^\circ$  and  $30^\circ$  at  $1^\circ$  increments. These images were given to the derotation network, and the resulting histogram of angular errors is given in Figure 6.6 (left). As can be seen, 92% of the errors are within  $\pm 10^\circ$ .



**Figure 6.6:** Left: Frequency of errors in the derotation network with respect to the angular error (in degrees). Right: Fraction of faces that are detected by a detection network, as a function of the angle of the face from upright.

The detector network was trained with example images having orientations between  $-10^\circ$  and  $10^\circ$ . It is important to determine whether the detector is in fact invariant to rotations within this range. We applied the detector to the same set of test images as the derotation network, and measured the fraction of faces which were correctly classified as a function of the angle of the face. Figure 6.6 (right) shows that the detector detects over 90% of the faces that are within  $10^\circ$  of upright, but the accuracy falls with larger angles.

Since the derotation network's angular errors are usually within  $10^\circ$ , and since the detector can detect most faces which are rotated up to  $10^\circ$ , the two networks should be compatible.

Just as we noted in the previous section that the detector network is applied only to non-faces which have been derotated, the same observation can be made about faces. The derotation network does make some mistakes, but those mistakes may be *systematic*; in this case the detector may be able to exploit this to produce more accurate results. This idea will be tested in the experiments section.

## 6.6 Evaluation

### 6.6.1 Derotation Network with Upright Face Detectors

The first system we test employs the derotation network to determine the orientation of any potential face, and then applies two upright face detection networks from the previous chapter, Networks 1 and 2. Table 6.5 shows the number of faces detected and the number of false alarms generated on the three test sets. We first give the results of the individual detection networks, and then give the results of the post-processing heuristics (using a threshold of one detection). The last row of the table reports the result of arbitrating the outputs of the two networks, using an AND heuristic. This is implemented by first post-processing the outputs of each individual network, followed by requiring that both networks signal a detection at the same location, scale, and orientation. As can be seen in the table, the post-processing heuristics significantly reduce the number of false detections,

and arbitration helps further. Note that the detection rate for the *Tilted Test Set* is higher than that for the *Upright Test Set*, due to differences in the overall difficulty of the two test sets.

**Table 6.5:** Results of first applying the derotation network, then applying the standard upright detector networks.

System	Upright Test Set		Tilted Test Set	
	Detect %	# False	Detect %	# False
Network 1	89.6%	4835	91.5%	2174
Network 2	87.5%	4111	90.6%	1842
Net 1 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	85.7%	2024	89.2%	854
Net 2 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	84.1%	1728	87.0%	745
Nets 1,2 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap $\rightarrow$ AND(4) $\rightarrow$ overlap	81.6%	293	85.7%	119

## 6.6.2 Proposed System

Table 6.5 shows a significant number of false detections. This is in part because the detector networks were applied to a different distribution of images than they were trained on. In particular, at runtime, the networks only saw images that were derotated. We would like to match this distribution as closely as possible during training. The positive examples used in training are already in upright positions, and barring any systematic errors in the derotator network, have an approximately correct distribution. During training, we can also run the scenery images from which negative examples are collected through the derotator. We trained two new detector networks using this scheme, and their performance is summarized in Table 6.6. As can be seen, the use of these new networks reduces the number of false detections by at least a factor of 6. The detect rate has also dropped, because now the detector networks must deal with non-faces derotated to look as much like faces as possible. This makes the detection problem harder, and the detection networks more conservative. Of the systems presented in this chapter, this one has the best trade-off

between the detection rate and the number of false detections. Images with the detections resulting from arbitrating between the networks are given in Figure 6.5.

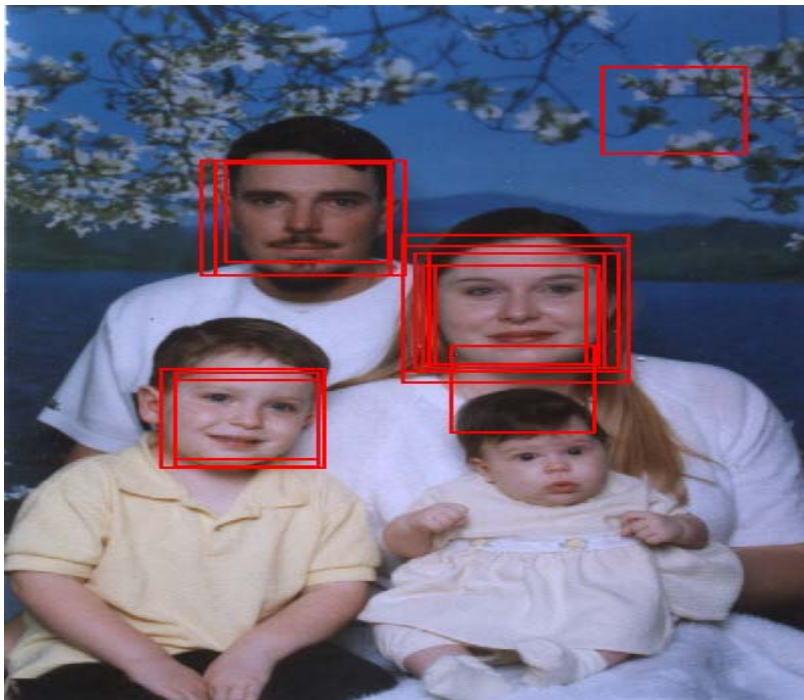
**Table 6.6:** Results of the proposed tilted face detection system, which first applies the derotator network, then applies detector networks trained with derotated negative examples.

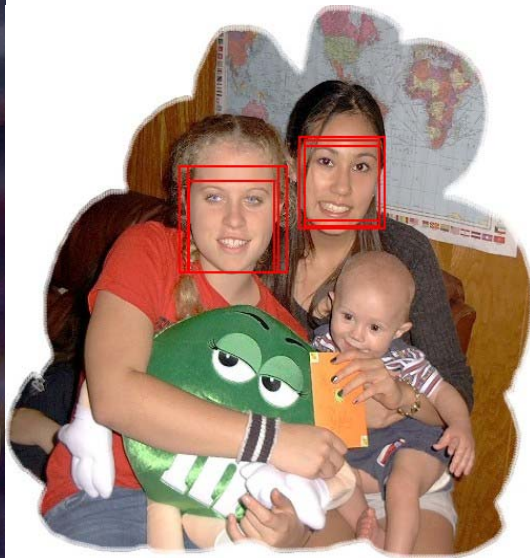
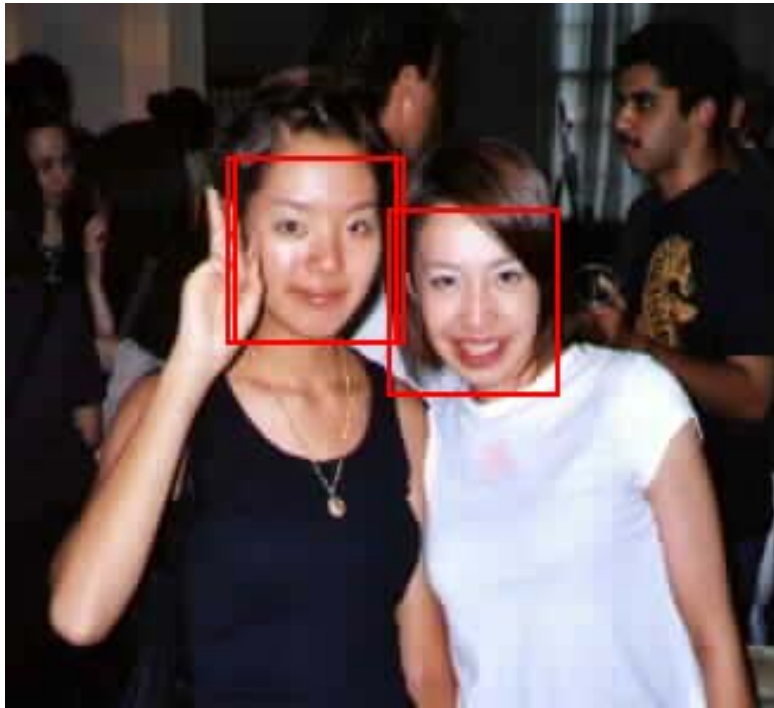
System	Upright Test Set		Tilted Test Set	
	Detect %	# False	Detect %	# False
Network 1	81.0%	1012	90.1%	303
Network 2	83.2%	1093	89.2%	386
Network 1 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	80.2%	710	89.2%	221
Network 2 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	82.4%	747	88.8%	252
Networks 1 and 2 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap $\rightarrow$ AND(4)	76.9%	44	85.7%	15

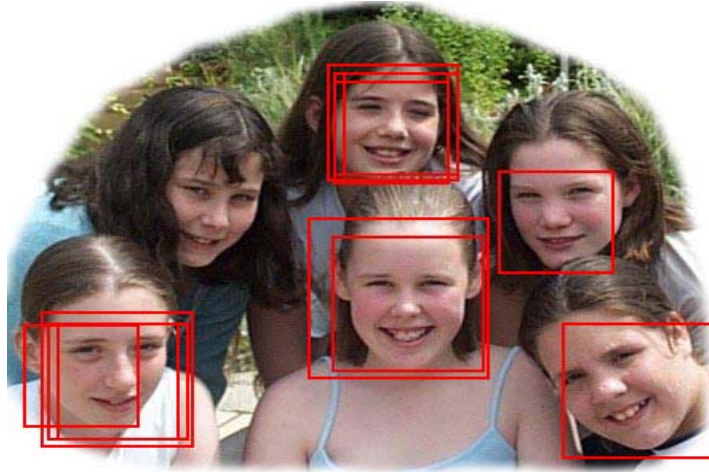
This idea can be carried a step further, to training the detection networks on face examples which have been derotated by the derotation network. If there are any systematic errors made by the derotation network (for example, faces looking slightly to one side might have a consistent error in their angles), the detection network might be able to take advantage of this, and produce better detection results. The results of this training procedure are shown in Figure 6.3. As can be seen, the detection rates are somewhat lower, and the false alarm rates are significantly lower.

One hypothesis for why this happens is as follows: For robustness, the previous detector networks were trained with face images including small amounts of rotation, translation and scaling. However, since the derotation network was more accurate without such variations, it was trained without them. In this experiment, the positive examples had these sources of variation removed. The scale and translation was removed when the randomly rotated faces are created, while the rotation variation is removed the the derotation network. This may have made the detector somewhat brittle to small variations in the faces. However, at the same time it makes the set of face images that must be accepted smaller, making it easier to discard non-faces.

An alternative hypothesis is that the errors made by the derotation network are not systematic enough to be useful. Instead, perhaps they introduce more variability into the face images. Because of the random error in recovery of the angle, important facial







**Figure 6.2:** Result of arbitrating between two networks trained with derotated negative examples.

features are no longer at consistent locations in the input window, making the detection problem itself harder. This hypothesis does not explain the lower false alarm rate, however. Both of these hypotheses deserve further exploration.

**Table 6.3:** Result of training the detector network on both derotated faces and non-faces.

System	Upright Test Set		Tilted Test Set	
	Detect %	# False	Detect %	# False
Network 1	67.3%	294	75.8%	109
Network 2	69.9%	341	79.4%	102
Network 1 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	66.9%	245	75.3%	89
Network 2 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	69.1%	278	79.4%	88
Networks 1 and 2 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap $\rightarrow$ AND(4)	61.1%	10	71.7%	6



### 6.6.3 Exhaustive Search of Orientations

To demonstrate the effectiveness of the derotation network for rotation invariant detection, we applied the two sets of detector networks described above without the derotation network. The detectors were instead applied at 13 different orientations (in increments of  $20^\circ$ ) for each image location. We expect such systems to detect most rotated faces. However, assuming that errors occur independently, we may also expect many more false detections than the systems presented above. Table 6.9 shows the results using the upright face detection networks from the previous chapter, and Table 6.10 shows the results using the detection networks trained with derotated negative examples.

**Table 6.9:** Results of applying the upright detector networks from the previous chapter at 13 different image orientations.

System	Upright Test Set		Tilted Test Set	
	Detect %	# False	Detect %	# False
Network 1	93.7%	17848	96.9%	7872
Network 2	94.7%	15828	95.1%	7328
Network 1 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	87.5%	4828	94.6%	1928
Network 2 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	89.8%	4207	91.5%	1719
Networks 1 and 2 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap $\rightarrow$ AND(4)	85.5%	559	90.6%	259

**Table 6.10:** Networks trained with derotated examples, but applied at all 13 orientations.

System	Upright Test Set		Tilted Test Set	
	Detect %	# False	Detect %	# False
Network 1	90.6%	9140	97.3%	3252
Network 2	93.7%	7186	95.1%	2348
Network 1 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	86.9%	3998	96.0%	1345
Network 2 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	91.8%	3480	94.2%	1147
Networks 1 and 2 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap $\rightarrow$ AND(4)	85.3%	195	92.4%	67

Recall that Table 6.5 showed a larger number of false positives compared with Table 6.6, due to differences in the training and testing distributions. In Table 6.5, the detection networks were trained with false-positives in their original orientations, but were tested on images that were rotated from their original orientations. Similarly, if we apply these detector networks to images at all 13 orientations, we should expect an increase in the number of false positives because of the differences in the training and testing distributions (see Tables 6.9 and 6.10). The detection rates are higher than for systems using the derotation network. This is because any error by the derotator network will lead to a face being missed, whereas an exhaustive search of all orientations may find it. Thus, the differences in accuracy can be viewed as a tradeoff between the detection and false detection rates, in which better detection rates come at the expense of much more computation.

### 6.6.6 Upright Detection Accuracy

Finally, to check that adding the capability of detecting rotated faces has not come at the expense of accuracy in detecting upright faces, in Table 6.11 we present the result of applying the original detector networks and arbitration method from Chapter 3 to the three test sets used in this chapter. The results for the *Upright Test Set* are slightly different from those presented in the previous chapter because we now check for the detection of 6 upside-down faces, which were present, but ignored, in the previous chapter.

**Table 6.11:** Results of applying the upright algorithm and arbitration method from the previous chapter to the test sets.

System	Upright Test Set		Tilted Test Set	
	Detect %	# False	Detect %	# False
Network 1	90.6%	928	20.6%	380
Network 2	92.0%	853	19.3%	316
Network 1 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	89.4%	516	20.2%	259
Network 2 $\rightarrow$ threshold(4,1) $\rightarrow$ overlap	90.6%	453	17.9%	202
Networks 1 and 2 $\rightarrow$ threshold(4,2) $\rightarrow$ overlap $\rightarrow$ AND(4)	85.3%	31	13.0%	11

Table 6.12 shows a breakdown of the detection rates of the above systems on faces that are rotated less or more than  $10^\circ$  from upright, in the *Upright Test Set* and *Tilted Test Set*. As expected, the upright face detector trained exclusively on upright faces and negative examples in their original orientations gives a high detection rate on upright faces. The tilted face detection system has a slightly lower detection rate on upright faces for two reasons. First, the detector networks cannot recover from all the errors made by the derotation network. Second, the detector networks which are trained with derotated negative examples are more conservative in signalling detections; this is because the derotation process makes the negative examples look more like faces, which makes the classification problem harder.

Another way to breakdown the results of the tilted face detector is to look at how each of the two stages, the derotation stage and the detection stage, contribute to the detection rate. To measure this, we extract the  $20 \times 20$  windows in the test sets which contain a face, and compute the derotation angle using two methods: the neural network, and the alignment method used to prepare the training data for this network. By comparing the results of these two methods, we can see how accurate the derotation network is on an independent test set. Next, the faces derotated by these two methods can be passed to the detection network, whose detection rates can be measured

**Table 6.12:** Breakdown of detection rates for upright and rotated faces from the test sets.

System	All Faces	Upright Faces ( $\leq 10^\circ$ )	Rotated Faces ( $> 10^\circ$ )
Tilted detector (Table 4.8)	79.6%	77.2%	84.1%
Upright detector (Chapter 3)	63.4%	88.0%	16.3%

for these two cases. The results of these comparisons, for the *Upright* and *Tilted Test Sets* are shown in Table 6.13.

**Table 6.13:** Breakdown of the accuracy of the derotation network and the detector networks for the tilted face detector.

Stage	Statistic	Test Sets	
		Upright	Tilted
Derotation network output	Angle within $10^\circ$	69.3%	76.7%
Detector output	Manual derotation	61.4%	58.7%
	Automatic derotation	49.3%	56.5%
Complete system	Detection rate	76.9%	85.7%

As can be seen from the table, there is a between 2% and 12% penalty for using the neural network to derotate the images, relative to derotating them by hand. This penalty partly explains the decrease in the detection rate compared with the upright detector. The table shows only the detection rates when applying a single detector network at a single pixel location and scale in the image. In practice, the detectors are applied at every pixel location and scale, giving them more opportunities to find each face. This explains the higher detection rates of the complete system (the last line in Table 6.13 relative to the earlier lines).

## 6.5 Summary

This chapter has demonstrated the effectiveness of detecting faces rotated in the image plane by using a derotation network in combination with an upright face detector. The system is able to detect 29.6% of faces over several large test sets, with a small number of false positives. The technique is applicable to other template-based object detection schemes. The next chapter will examine some techniques for detecting faces that are rotated out of the image plane.

# Chapter 7

## Non-Frontal Face Detection

### 7.1 Introduction

The previous chapter presented a two stage face detection algorithm, in which first analyzes the angle of a potential face, then uses this information to geometrically normalize that part of the image for the detector itself. The same idea can be applied in the more general context of detecting faces rotated out of the image plan. There are two ways in which this could be approached. The first is directly analogous to the approach for tilted faces: by using knowledge of the shape and symmetry of human faces, image processing operations may be able to convert to a profile or half profile view of a face to a frontal view. A second approach, and the one we have explored in more detail, is to partition the views of the face, and to train separate detector networks for each view.

We used five views: left profile, left semi-profile, frontal, right semi-profile, and right profile. A pose estimator is responsible for directing the input window to one of these view-specific detectors. The work presented here only handles two degrees of freedom of rotation: rotation in the image plane, and rotation to the left or right out of the plane. Extending the algorithm to faces rotated up or down should be straightforward.

This Section begins with a discussion of how to estimate the three dimensional pose of a face given an input image, and how to use this pose information to geometrically distort the image to synthesize an upright, frontal view. We will discuss that the results of the procedure are not good enough for use in a face detector. Then we will discuss pose information to select a detector customized to a particular view of the face.

## 7.2 Geometric Distortion to a Frontal Face

To detect faces which are tilted in the image plane, we simply applied some image processing operators to rotate each window to an upright orientation before applying the detector. If we have a 3D model of the head, and information about the orientation of the head in the image, we can use texture mapping to generate an upright, frontal image of the head. Similar techniques have been used in the past to generate frontal images of the face from partial profile views for the face recognition task. These techniques work quite well, but are quite computationally expensive, requiring an iterative optimization procedure to align each face with the model.

Our algorithm computes the orientation of a potential face for every window of the image before running the detector. There are other approaches using eigenspaces and those using three dimensional geometric models of the face. Because this algorithm must be applied so many times, a computationally less expensive technique such as a neural network is more appropriate. The following sections describe the training data for this network, how it was trained, and finally describe how texture mapping was used to synthesize an upright, frontal view of the face.

### 7.2.2 Labelling the 3D Pose of the Training Images

There are several new issues that arise in creating the training data for this problem. We begin by labelling important feature points in images. We aligned the faces with one another by performing a two-dimensional alignment, using translation, rotation, and uniform scaling to minimize the sum of squared distances between the labelled feature locations.



**Figure 7.1:** Generic three-dimensional head model used for alignment.

Since we are now considering out-of-plane rotations, the faces must be aligned with one another in three-dimensions. However, we are given only a two-dimensional representation of each face, and two-dimensional feature locations. We begin with a three-dimensional model of a generic face, shown in Figure 7.1. The feature locations used to label the face images are labelled in 3D on the model. Then, we attempt to find the best three-dimensional rotation, scaling, and translation of the model which, under an orthographic projection, best matches each face. A perspective projection could also be used, but since it has more parameters, their estimates will be less robust. This is similar to the alignment strategy presented earlier, but using a three-dimensional model. Unlike two-dimensional alignment, this least-squares optimization no longer has a closed form solution in terms of an over-constrained linear system. If we denote the locations of feature  $i$  of the face as  $x_i$  and  $y_i$ , and the feature locations of the 3D model as  $x_i, y_i, z_i$ , then optimization problem is to minimize  $E$  in the following equation:

$$E(S, T_x, T_y, q) = \sum_{i=1}^n \left| \begin{pmatrix} x'_i \\ y'_i \end{pmatrix} - \begin{pmatrix} S & 0 & 0 & T_x \\ 0 & S & 0 & T_y \end{pmatrix} \cdot R(q) \cdot \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \right|^2$$

where  $S$  is the scaling factor,  $R(q)$  is a  $4 \times 4$  rotation matrix parameterized by a four dimensional quaternion  $q$ , and  $T_x, T_y$  are the translation parameters. Each  $x_i$  and  $y_i$  gives rise to a term which contributes to the summation, and depends nonlinearly on the parameters. A standard method to optimize such a system is the iterative Levenberg-



Marquardt method [Marquardt, 1963, Press *et al.*, 1993]. For simplicity, the summation above can be rewritten as a matrix equation, as follows:

$$E = \left| \begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \\ x'_n \\ y'_n \end{pmatrix} - f(P) \right|^2 = |X' - f(P)|^2$$

where  $P$  is the vector of parameters  $S$ ,  $T_x$ ,  $T_y$ ,  $q$ , and  $f$  is a vector function generating the coordinates of the 3D head model from the pose specified by those parameters.

The Levenberg-Marquardt method approximates the function  $f(P)$  using a first-order Taylor expansion, as follows:

$$f(P_0 + \Delta P) \approx f(P_0) + \left( \frac{\partial f(P_0)}{\partial P} \right) \Delta P = f(P_0) + J \Delta P$$

Since the error function is quadratic, it can be minimized by setting its derivative to zero using the above Taylor approximation, as follows:

$$\begin{aligned} \frac{\partial E(P_0 + \Delta P)}{\partial \Delta P} &\approx \frac{\partial}{\partial \Delta P} |X' - f(P_0 + \Delta P)|^2 \\ &= \frac{\partial}{\partial \Delta P} |X' - (f(P_0) + J \Delta P)|^2 \\ &= 2J^T (X' - (f(P_0) + J \Delta P)) \\ &= 0 \end{aligned}$$

This is an over-constrained linear system, whose approximate solution is:

$$\Delta P \approx (J^T J)^{-1} J (X' - f(P_0))$$

This solution can only be computed when the initial parameters  $P_0$  are near the minimum, and the Taylor approximation is accurate. Under these conditions, the update to the parameters  $\Delta P$  can be very accurate.

However, dependences between the parameters may make the inversion of  $J^T J$  numerically unstable. In some cases, a better approach is that of gradient descent, in which  $\Delta P$  is computed as follows:

$$\Delta P \propto \frac{\partial E(P_0)}{\partial P} \approx 2 \left( \frac{\partial f(P_0)}{\partial P} \right)^T (X' - f(P_0)) = 2J^T (X' - f(P_0))$$

The Levenberg-Marquardt combines these two methods, as follows:

$$\Delta P = (J^T J + \lambda I) J^T (X' - f(P_0))$$

$\lambda$  is a weight for the contributions of the two methods. When  $\lambda$  is zero, the method follows the Taylor expansion method. When  $\lambda$  is large,  $\Delta P$ 's value is dominated by the gradient descent value. The actual update to the parameters is computed from  $P_- = P_0 + k\Delta P$ , and the method is iterated until the parameters converge.

To apply the Levenberg-Marquardt method to matching a face in three-dimensions, we need to know the  $f(P)$  and  $J$  functions. The  $f(P)$  function is given by the above equation relating  $x_{-i}$ ,  $y_{-i}$  and  $x_i$ ,  $y_i$ ,  $z_i$ . The only non-linear portion of this equation is in the rotation matrix  $R(q)$  which has a non-linear dependence on the four quaternion parameters  $q$ .

Using the Levenberg-Marquardt method, we can rotate, translate, and scale the 3D face model in three-dimensions to align it with each of the training faces. The face images are roughly categorized according to their angle from frontal, but the actual angles of the faces can vary significantly within each category. The approximate angle for each category is used to initialize the Levenberg-Marquardt optimization, which is then run until the parameters converge.

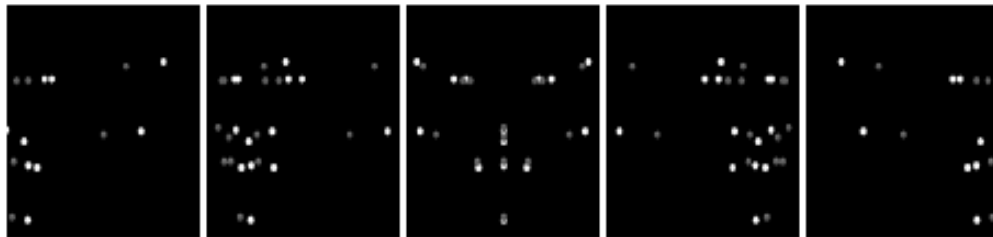
Once the faces are all aligned with the 3D model, the positions of the features on the aligned faces can be averaged to update the feature positions in the 3D model. However, since each facial feature contains only two-dimensional information, they cannot be directly averaged.

Instead, we return to the equation relating  $x_{-i}$ ,  $y_{-i}$  and  $x_i$ ,  $y_i$ ,  $z_i$ , and write it in the following form:

$$\begin{pmatrix} S & 0 & 0 & T_x \\ 0 & S & 0 & T_y \end{pmatrix} \cdot R(q) \cdot \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} = \begin{pmatrix} x'_i \\ y'_i \end{pmatrix}$$

We can see that when the rotation, translation, and scaling parameters of the face are known, this equation describes a linear relationship between the feature locations in the 3D model and the feature locations on each example face. We can make a larger matrix equation which includes all the features of all the faces. This equation will be over constrained, and can be solved by the least-squares method to find the vector of feature locations of the 3D model.

With an updated 3D model, we can go back and update the alignment parameters aligning that model with each face, and iterate the two steps several times until convergence. The resulting 3D model feature locations are shown in Figure 7.2 and the results of the alignment, illustrated by rendering the original 3D model together with the face, are shown in Figure 7.3.



**Figure 7.2:** Refined feature locations (gray) with the original 3D model features (white).



**Figure 7.3:** Rendered 3D model after alignment with several example faces.

### 7.2.3 Representation of Pose

We used quaternions to represent the angles of the 3D model with respect to the face. Quaternions have several nice properties which make them attractive for the type of optimization used to align the models, because they have no singularities. Continuous changes in the four-dimensional unit-quaternion space result in continuous changes in the rotation matrix. The expense of this representation is redundancy; rather than the minimal three parameters needed to describe an orientation, four are used.

One result of this redundancy in the quaternion representation is that a quaternion and its corresponding negative quaternion both represent the same angle. Any attempt to restrict the representation to one of these pairs (say, by restricting the first component to always be positive) will lead to singularities in the representation. The redundancy of this form is not a problem for an optimization procedure using gradient descent, but it is a problem if you want to build a mapping from an input directly to a quaternion using a neural network.

Ideally, we need a representation which has no singularities, and also gives a unique representation for each rotation. One simple representation is to use two orthogonal 3D unit vectors, one pointing from the center of the 3D model to the right ear, the other pointing along its nose. This representation is clearly unique (any change in the unit vectors changes the orientation of the head), and also clearly continuous (any small change in the unit

vectors gives a small change of orientation). Again, this improvement of the representation comes at the cost of redundancy, because we now need to use six parameters to represent the orientation.

## 7.2.4 Training the Pose Estimator

From the previous two sections, we have example face images which are aligned with one another in three dimensions, and a continuous representation of the angle of each face. The scale and translation components of the alignment are used to normalize the size and position of each angle. The image is rotated by a random amount in-plane, and the orientation parameters are adjusted appropriately. This gives example images and outputs like those shown in Figure 7.4. As before, a number of random variations of each example face are used to increase the robustness of the system. It is also important to balance the number of faces at each orientation. For this purpose, we quantize the angle of each face from frontal into increments of  $10^\circ$  and count the number of faces in each category. The number of random examples for each face is inversely proportional to the number of faces in the category, which equalizes the distribution of this angle.



**Figure 7.4:** Example input images (left) and output orientations for the pose estimation neural network. The pose is represent by six vectors of output units (bottom), collectively representing 6

real values, which are unit vectors pointing from the center of the head to the nose and the right ear. Together these two vectors define the three dimensional orientation of the face. The pose is also illustrated by rendering the 3D model at what same orientation as the input face (right).

The outputs from the pose estimation neural network should be two unit vectors, representing the orientation of the head. Each component of these vectors is represented by an array of output units. Their values are computed by a weighted sum of the positions of the outputs within the array, weighted by the activation of the output.

With the training examples in hand, a neural network can be trained. The network has an input retina of  $20 \times 20$  pixels, connected to six sets of hidden units, each of which looks at a  $7 \times 7$  sub-window. These hidden units are completely connected to a layer of 40 hidden units, which is then completely connected to the output layer. The output consists of six arrays of 31 units, each representing a real value between -1 and 1. The results of this network on some test images are illustrated in Figure 7.7. When applying the network, the output vectors' magnitudes are normalized, and the second unit vector is forced to be perpendicular to the first.



**Figure 7.7:** The input images and output orientation from the neural network, represented by a rendering of the 3D model at the orientation generated by the network.

## 7.2.7 Geometric Distortion

The main difficulty in producing a frontal image of a face from a partial profile is that parts of the face in the original image will be occluded. However, if we assume that the left and right sides of the face are symmetric with one another, we can replace the half of the upright, frontal view that contains partial occlusions with a mirror image of the other half.

Some example results are shown in Figure 7.6, for faces which are limited to angles of  $47^\circ$  from frontal. As can be seen, when the pose estimation network is accurate and the face is similar in overall shape to the 3D model, the resulting upright, frontal view of the face can be quite realistic. However, small errors in the pose estimation result in larger artifacts in the frontal faces, and large errors in pose estimation give results that are unrecognizable as faces. Additionally, even if the pose estimation is perfect, errors in the 3D model of the face (which is just a generic model) will lead to errors. Given these potential problems, we decided to use another approach.



**Figure 7.6:** Input windows (left), the estimated orientation of the head (center), and geometrically distorted versions of the input windows intended to look like upright frontal faces (right).

## 7.3 View-Based Detector

Instead of trying to geometrically correct the image of the face, we will try to detect the faces rotated out-of-plane directly. However, we cannot expect a single neural network to be able to detect all views of the face by itself. Even increasing the amount of in-plane rotation for frontal face images dramatically increase the error rate. To minimize the amount of variation in the images the neural network must learn, we partition the views of the face into several categories according to their approximate angle from frontal.

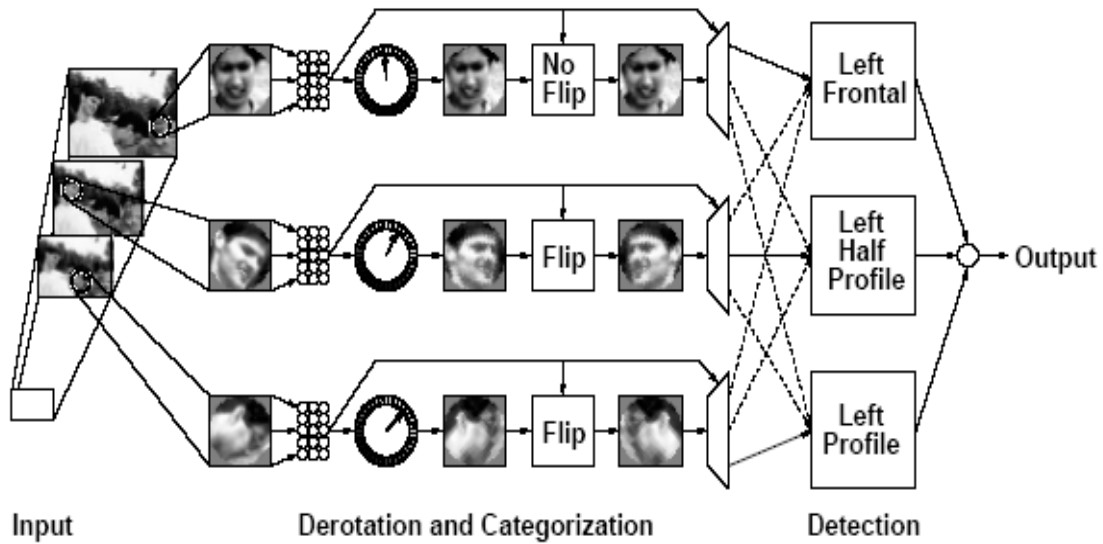
The idea is to use a pose estimation network to first compute the in-plane angle of the face and the category, then rotate the image in-plane to an upright orientation, and finally to apply the appropriate detector network. Note that the detectors for the faces looking to the right are simply mirror images of the networks for the faces looking to the left. This algorithm is illustrated in Figure 7.2.

### 7.3.1 View Categorization and Derotation

The detector networks work by looking for particular features (mostly the eyes) at particular locations in the input window. Imagine a head rotating in the input window. As it rotates, all the feature locations will shift within the input window, meaning that none of the feature locations are stable. This would make the detection problem much harder. It would be better to apply the two-dimensional alignment procedure to the faces within each category. This would allow each view-specific face detector to concentrate on specific features in specific locations.

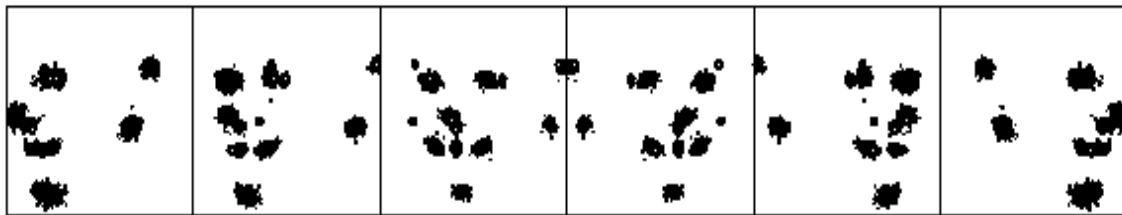
We are still left with the question of how to assign faces to specific categories. In light of the observation that the two-dimensional alignment of feature locations is important, we chose to use a criterion based on how closely the feature locations align with a prototypical example of the category





**Figure 7.2:** View-based algorithm for detecting non-frontal and tilted faces.

. This prototype is constructed from the three-dimensional model, which is rotated to several angles from frontal, as shown in Figure 7.3. Each of the face examples is aligned as well as possible with all of the category prototypes, and assigned to the category it matches closest.

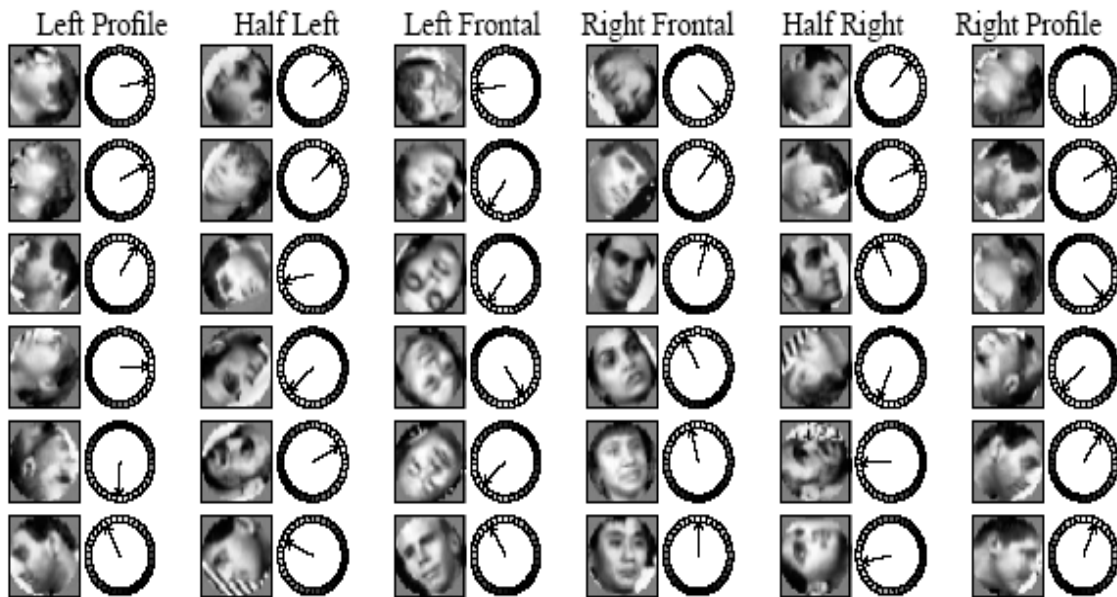


**Figure 7.3:** Feature locations of the six category prototypes (white), and the cloud of feature locations for the faces in each category (black dots).

As before, the actual alignment could be an iterative process, first aligning all the faces in the category with the category prototype, then updating the prototype with the average of all the aligned faces in the category, and repeating. However, in our experiments we found that simply aligning each faces with the original category prototype allowed the category estimation network to work better. This may be because the original prototypes

have geometric relations with one another (such as the eyes always falling on the same scan line) which are disrupted if the prototypes are adjusted to the training examples.

Once the faces are aligned with one another, they are rotated to random in-plane orientations, and the resulting images are recorded as the training examples, as shown in Figure 7.9. Associated with each face example is its in-plane orientation and the category label. As before, we produce several random variations of each training face, and the number of variations is chosen to balance the number of examples in each category.



**Figure 7.9:** Training examples for each category, and their orientation labels, for the categorization network. Each column of images represents one category.

Next we can train a neural network to produce the categorization and in-plane orientation of an input window. The architecture used consists of four layers. The input layer consists of units which receive intensities from the input window, a circle of radius 17 pixels. The first hidden layer has localized connections to this circular input. The second hidden layer has 40 units, with complete connections to the first hidden layer and to the output layer. The output angle is represented by a circle of output units, each representing a particular angle. Each category label has an individual output, and the category with the highest output is considered to be the classification of the window. Note that one difference with

the previous work is that the input window is circular; this makes it possible to rotate the input window in-plane without having to recompute the preprocessing steps. With a square window, the derotated window covers different pixels than the original window, invalidating the histogram equalization that is done. As a further optimization, the rotation is done by sampling pixels at integer coordinators rather than bilinear interpolation.



**Figure 7.10:** Training examples for the category-specific detection networks, as produced by the categorization network. The images on the left are images of random in-plane angles and out-of-plane orientations. The six columns on the right are the results of categorizing each of these images into one of the six categories, and rotating them to an upright orientation. Note that only three category-specific networks will be trained, because the left and right categories are symmetric with one another.

### 7.3.2 View-Specific Face Detection

We can apply this network to all of its training data, which will categorize and derotate the images into the appropriate categories. These images can then be used to train a set of detection networks. Note that we could have created training data based directly on the original images and their categorizations, but by using the view-categorization network to label the training data, we hope to capitalize on any systematic errors that it may make. The

training networks are trained in the same way as those used for the tilted face detector. In particular, the negative examples are also run through the view categorization network, to make sure the detectors are trained on the same type of images they will see at runtime.

As in the previous chapters, two networks are trained for each of three categories, and their results are arbitrated to improve the overall accuracy of the system. The main technique is to examine all the detections within a small neighborhood of a given detection, and the total number is interpreted as a confidence measure for that particular detection. For the upright face detection, the neighborhood was defined in terms of the position and scale of the detection. For the tilted detector, an extra dimension, the in-plane orientation of the head, was added. Small changes in each of these dimensions result in small changes in the image seen by the detector, so the neighborhood of a detection is easily defined in terms of a neighborhood along each dimension.

For the non-frontal detector, yet another dimension is needed, that of the out-of-plane orientation, or category, of the head. Unlike the previous dimensions, this dimension has discrete values. The categories place facial features at different locations; see for example the location of the point between the two eyes for each category prototype in Figure 7.3. To decide whether two detections are in the same neighborhood, the offset between the facial feature locations in the two categories must be taken into account. For each pair of categories, the shift of the point between the two eyes is computed. The following procedure is then used to decide whether detection  $D_2$  is in the neighborhood of detection  $D_1$ :

1. If  $D_1$  and  $D_2$ 's categories differ by more than one, return `false`.
2. If the scales of  $D_1$  and  $D_2$  more than 4 apart, return `false`.
3. Translate the location of  $D_2$  by the offset for the two categories. The translation is along the direction indicated by the in-plane orientation of  $D_2$ .
4. Scale the location of  $D_2$  according to the difference in pyramid levels between the two detections.
7. If the adjusted location of  $D_2$  further than 4 pixels in  $x$  and  $y$  from  $D_1$ , then return `false`.
6. Return `true`.

Using this concept of neighborhood, the actual arbitration procedure used is the same as that used for the tilted faces. Then, the cleaned results of the two networks are ANDed together, again using the neighborhood concept to locate corresponding detections in the outputs of the two networks.

The results of the individual networks and the complete system are reported in the next section.

## 7.4 Evaluation of the View-Based Detector

### 7.4.1 Non-Frontal Test Set

The set of images are intended to have a variety of poses, from frontal to profile, as well as a variety of in-plane angles. In the experiments section, this set is referred to as the *Non-Frontal Test Set*.

### 7.4.3 Experiments

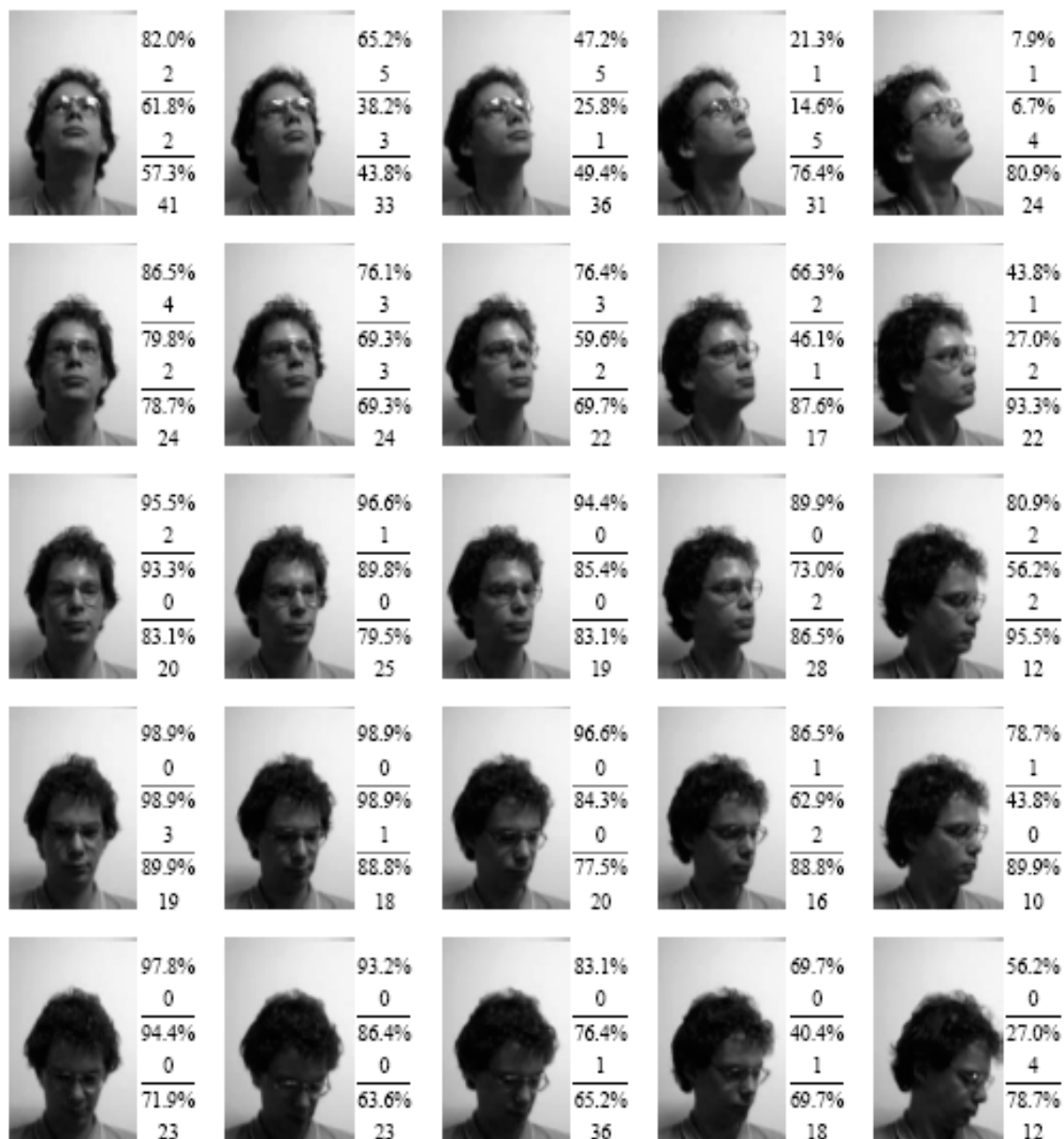
To evaluate the view detector network, I first tested its capabilities compared with the previous two systems. As before, the accuracy of the system will depend on the types of arbitration used. The system has significantly more false alarms than either of the two previous systems, and a slightly lower detection rate. This suggests that for applications needing the detection of only upright or tilted faces, one of the two previous detectors is a better choice.

The performance of the upright, tilted, and non-frontal face detectors is shown in Figure 7.17. Next to each image is a pair of numbers representing the accuracy of the detector in the orientation of the head. The second pair of numbers gives the performance for the tilted face detector. As with the upright detector, it has a high detection rate for frontal images. However, its accuracy drops off more quickly than the upright detector as the face is rotated away from frontal. The last pair of numbers gives the accuracy of the non-frontal

detector for these images. The detection rate of this detector is quite uniform over the test images, detecting both frontal and profile faces. The lowest detection rates are for faces looking above or below the camera, because this type of rotation is not covered by the non-frontal detector. As the faces turn towards profiles, the detection rate improves. This is because up and down motion becomes rotation in the image plane, which the non-frontal detector is able to handle.

To get a better understanding of why the detection rates are lower for the non-frontal face detector than the  $30 \times 30$  circular window in the test sets which contains a face, and the derotation angle and category of the face are computed using two methods: the neural network, and the method used to prepare the training data for this network. By comparing the results of these two methods, we can see how accurate the derotation and categorization network is on an independent test set. Next, the faces derotated and categorized by these two methods can be passed to the detection network, whose detection rates can be measured for these two cases. The results of these comparisons, for the *Upright*, *Tilted*, and *Non-Frontal Test Sets* are shown in Table 7.16.

It is reasonable to assume that the detection rate using the manual classification is the best possible, and that the automatic



**Figure 7.17:** Images used for testing the pose invariant face detector.. Next to each representative image are three pairs of numbers. The top pair gives the detection rate and number of false alarms from the upright face detector of Chapter 3. The second pair gives the performance of the tilted face detector from Chapter 4, and the last pair contains the numbers from the system described in this chapter.

**Table 7.16:** Breakdown of the accuracy of the derotation and categorization network and the detector networks for the non-frontal face detector.

Stage	Statistic	Test Sets		
		Upright	Tilted	Non-Frontal
Derotation and categorization network output	Exact category	49.5%	48.7%	38.5%
	Not categorized	25.1%	20.5%	30.2%
	Angle within $10^\circ$	42.6%	42.4%	21.9%
Detector output	Manual categorization	65.3%	65.2%	39.6%
	Automatic categorization	34.9%	31.7%	16.7%
	Predicted	32.3%	31.8%	15.2%
Complete system	Detect rate	67.2%	69.6%	56.2%

categorization and derotation will be the product of the detection rate for manual categorization/derotation and the fraction of the faces for which the categorization/derotation network returns the right category. This prediction is shown in the “Predicted” line of Table 7.16. Since the prediction accurately matches the actual detection rate when using the neural network for categorization and derotation, we can see that improving the categorization performance will directly improve the overall detection rate.

The table shows only the detection rates when applying a single detector network at a single pixel location and 7.16 relative to the earlier lines.

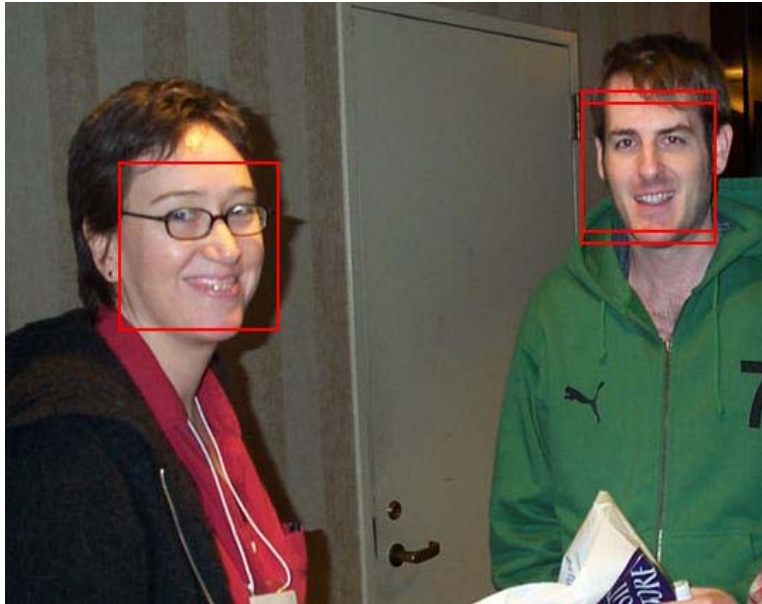
## 7.7 Summary

This chapter has presented an algorithm to detect faces which are rotated out of the image plane. First, by using geometric distortions, it may be possible to transform a face image from a partial profile to an upright frontal view, thereby enabling the use of an upright, frontal detector. However, in the experiments shown, it was found to be difficult to align a



3D model of the face precisely enough with the image to perform the transformation accurately; this made it unusable for detection.

The second approach partitions the out-of-plane rotations of the head into several views and uses separate detection





## **Chapter 8**

### **Speedups**

#### **8.1 Introduction**

In this chapter, we briefly discuss some methods to improve the speed of the face detectors presented in this thesis. This work is preliminary, and not intended to be an exhaustive exploration of methods to optimize the execution time.

## 8.2 Fast Candidate Selection

The dominant factor in the running time of the upright face detection system described thus far is the number of  $20 \times 20$  pixel windows which the neural networks must process. The computational cost of the arbitration steps is negligible in comparison, taking less than one second to combine the results of the two networks over all positions in the image of this size.

### 8.2.1 Candidate Selection

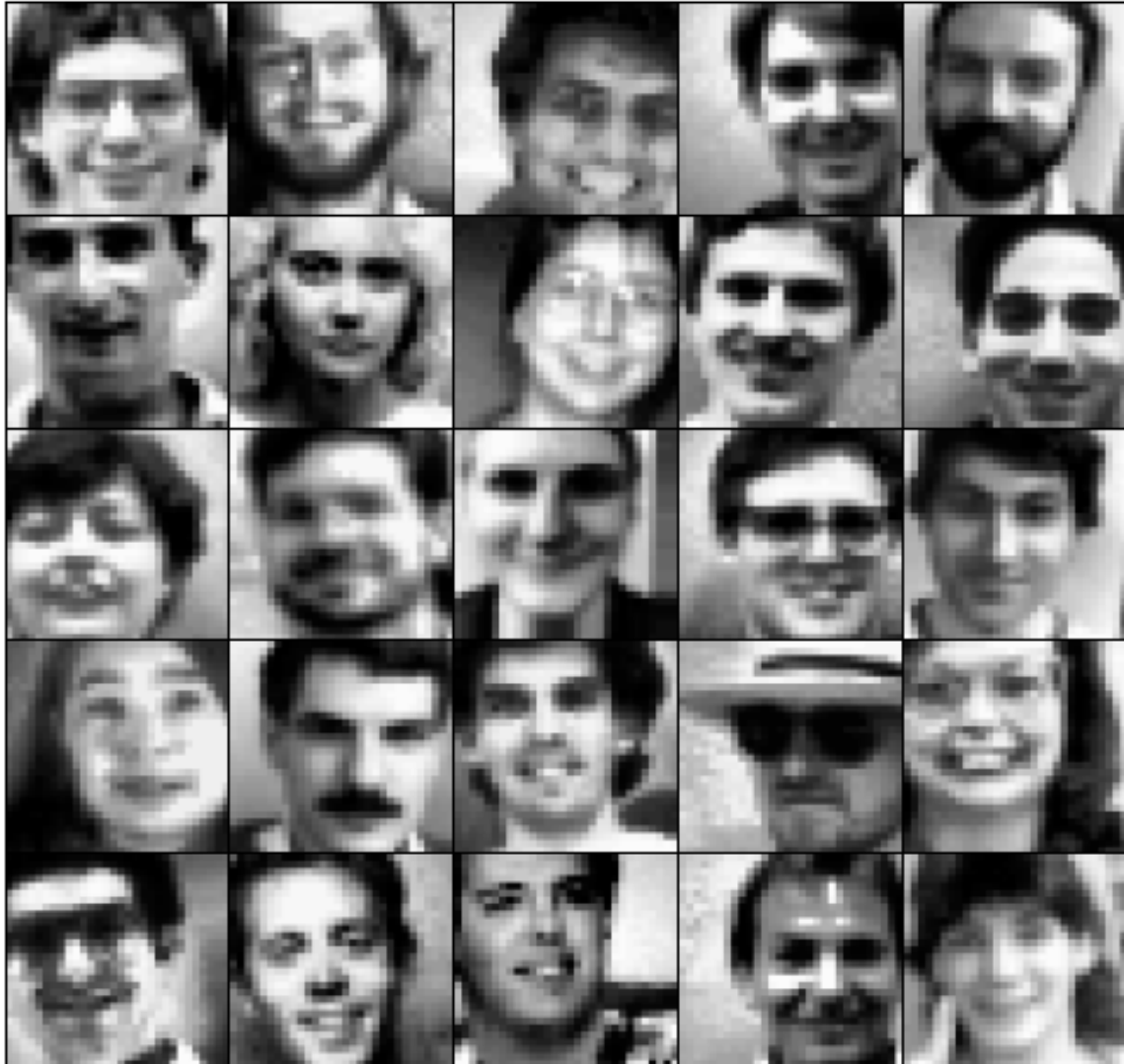
The original detector was trained to detect a  $20 \times 20$  face centered in a  $20 \times 20$  window. We can make the detector more flexible by allowing the same  $20 \times 20$  face to be off-center by up to 5 pixels in any direction. To make sure the network can still see the whole face, the window size is increased to  $30 \times 30$  pixels. Thus the center of the face will fall within a  $10 \times 10$  pixel region at the center of the window, as shown in Figure 8.1. As before, the network has a single output, indicating the presence or absence of a face. This detector can be moved in steps of 10 pixels across the image, and still detect all faces that might be present. The scanning method is illustrated in Figure 8.2, for the upright face detection domain. The figure shows the input image pyramid and the  $10 \times 10$  pixel regions that are classified as containing the centers of faces. An architecture with an image output was also tried, which yielded about the same detection accuracy, but required more computation. The network was trained using the active learning procedure. The windows are preprocessed with histogram equalization before they are passed to the candidate selector network.

As can be seen from the figure, the candidate selector has many more false alarms than the detectors described. A simple arbitration strategy, ANDing, is used to combine the outputs of two verification networks. The heuristic that faces rarely overlap can also be used to reduce computation, by first scanning the image for large faces, and at smaller scales not processing locations which overlap with any

detections found so far. The results of these verification steps are illustrated on the right side of Figure 8.2.

### **8.2.2 Candidate Localization**

Scanning the  $10 \times 10$  locations within the candidate for faces can still take a significant amount of time. This can be reduced by introducing another network, whose purpose is to localize the face more precisely. In the upright face detection domain, this network takes a  $30 \times 30$  input window, and should produce two outputs, the  $x$  and  $y$  positions of the face. These outputs are represented using a standard representation. Each output has a range from  $-5$  to  $5$ , so the output is represented by a vector of 20 outputs, each having an associated value of in the range  $-10$  to  $10$ .

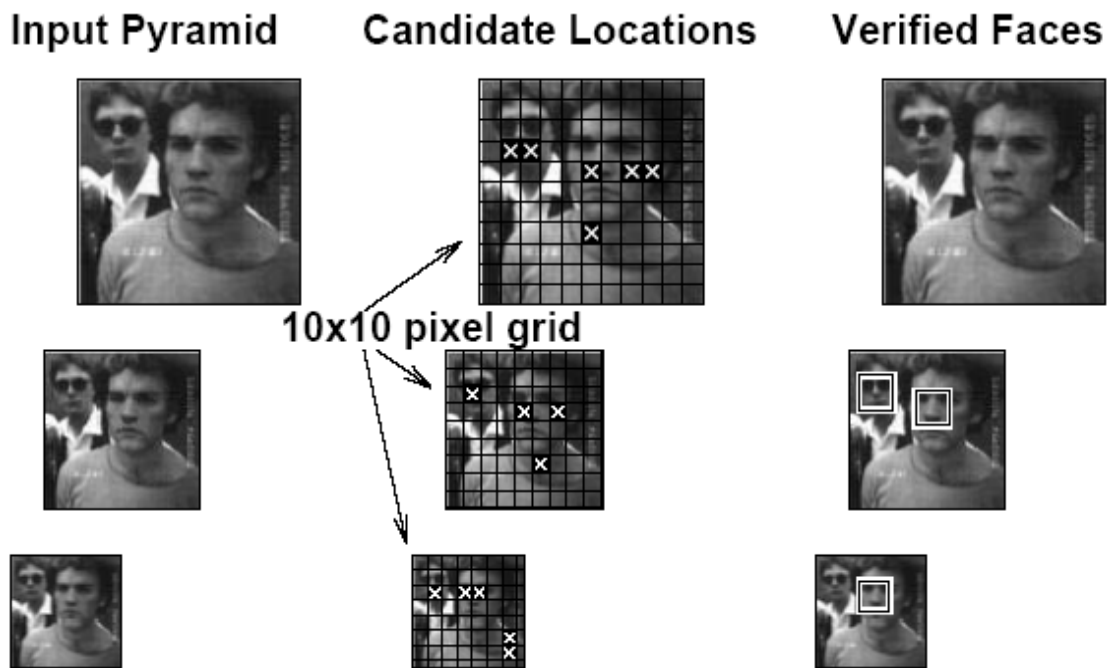


**Figure 8.1:** Example images used to train the candidate face detector. Each example window is  $30 \times 30$  pixels, and the faces are as much as five pixels from being centered horizontally and vertically.

To get the real valued output, a weighted sum of the values represented by each output is computed. The outputs are trained to produce a bell curve with a peak at the desired output. Given these values, the window centered on the face can be extracted and verified. We must check that the localization network is accurate enough, and the detection network is invariant enough, to the location of the face. Figure 8.3 shows an error histogram for the localization network, and the sensitivity of the upright face detector networks to how far off-center the face is. Since the average error of the localization network is quite small, the verification networks need only be applied once, at the estimated location of the face.

### 8.2.3 Candidate Selection for Tilted Faces

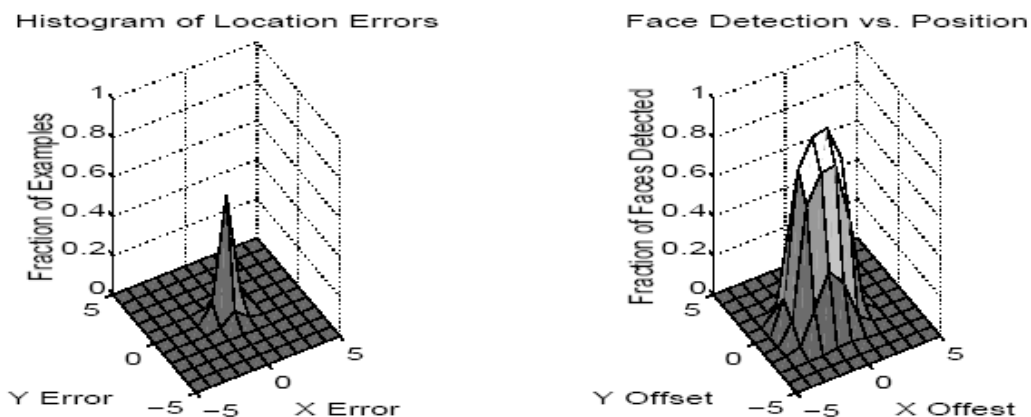
The same idea can be applied in the tilted face detection domain. We begin by training a candidate detector with examples of faces at different locations and orientations in the input window. Like with the upright candidate selector, the idea is that this network should be able to eliminate portions of the input from consideration. However, since the



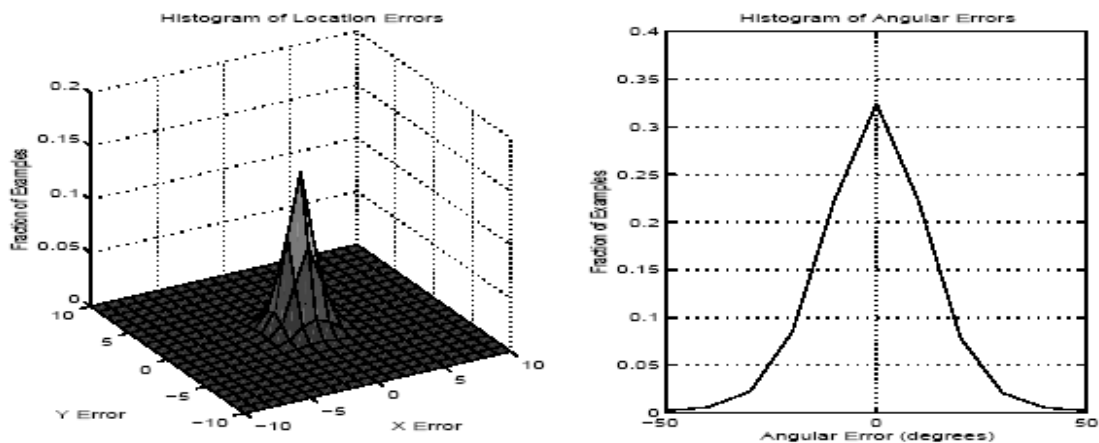
**Figure 8.2:** Illustration of the steps in the fast version of the face detector

set of allowed images is now much more variable, the candidate selector has a harder task, and it cannot eliminate as many areas, so the speedup will not be as large as in the upright case.

In addition to producing the  $x$  and  $y$  locations of the face, the tilted localization network must also produce the angle within an angle of about  $10^\circ$  to be detected accurately. Since the localization of this network is not as accurate as the upright detector, we need to apply the detectors to verify several candidate locations. Specifically, the detector is applied at 3 different  $x$  and  $y$  values and 5 angles around the estimated location and orientation of the face.



**Figure 8.3:** Left: Histogram of the errors of the localization network, relative to the correct location of the center of the face. Right: Detection rate of the upright face detection networks, as a function of how far off-center the face is. Both of these errors are measured over the training faces.



**Figure 8.4:** Left: Histogram of the translation errors of the localization network for the tilted face detector, relative to the correct location of the center of the face. Right: Histogram of the angular errors. These errors are measured over the training faces.

### 8.3 Change Detection

Further performance improvements can be made if one is analyzing many pictures taken by a stationary camera. By taking a picture of the background scene, one can determine which portions of the picture have changed in a newly acquired image, and analyze only those portions of the image.

In practice, changes in the environment, lighting, or automatic adjustments of the iris or gain in the camera can cause false detections. We need an intermediate between using a fixed background image, and using the previous image, to detect changes.

The intermediate I choose was to use a moving average of the input image as the “background”. The background

$$\text{background\_} = 0.95 \cdot \text{background} + 0.05 \cdot \text{input}$$

Then we compute the difference between the background and the input, and apply a threshold of 20. Before applying the threshold, changing pixels are at a border of their face.

## 8.4 Skin Color Detection

If there is a fast way to locate regions of the image containing skin color, then the search for faces can be restricted to those regions of the image.

A fast technique for locating skin color regions first converts the color information to a normalized color space. Under normal imaging and lighting conditions, skin colors for all races form a fairly tight cluster in normalized color space. Recent work using a very large number of images collected from the World Wide Web showed there are slightly differences between the races, but that a simple histogram-based model of skin color can accurately model the distribution of skin color.

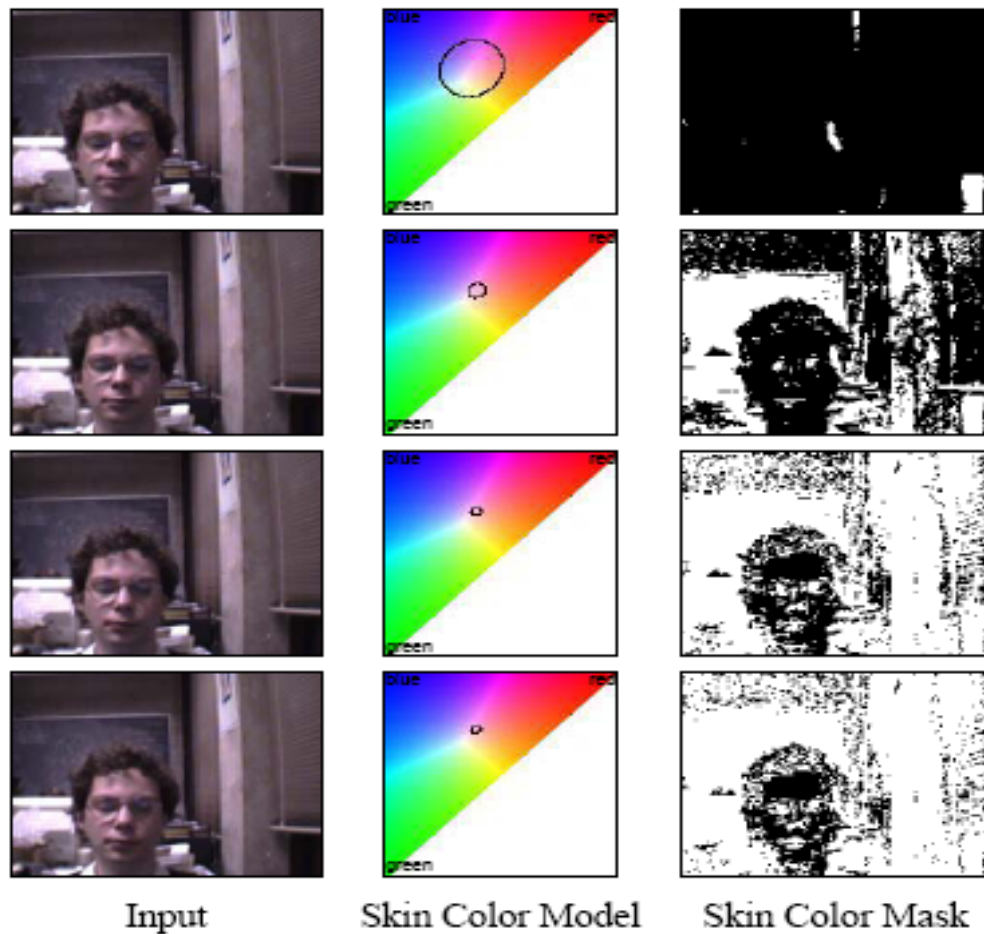
For our work, however, we used the Gaussian model. This model has the advantage of not requiring many training images. It can be implemented on a workstation. Then, before applying the candidate selector to a window of the image, the number of skin color pixels in the region are computed, and if this number is above a threshold, the region is evaluated by the candidate selector.

Note that the observed color associated with skin will depend strongly on the camera and the lighting conditions. A better approach is to model the skin color as the system is running.



We begin by using a very broad color model. Effectively every region of the image will need to be processed and update the Gaussian model. Over time, the Gaussian model will become more precise, allowing the detector to rule out larger areas of the image and run faster. This process is illustrated in Figure 8.8.

If the lighting conditions change, the skin color model may no longer be accurate, and faces will be missed. On



**Figure 8.8:** The input images, skin color models in the normalized color space (marked by the oval), and the resulting skin color masks to limit the potential face regions. Initially, the skin color model is quite broad, and classifies much of the background as skin colored. When the face is detected, skin color samples from the face are used to refine the model, so that it gradually focuses only on the face.

## 8.5 Evaluation of Optimized Systems

Incorporating the skin color and change detection algorithms brings down the processing time. This time depends on the complexity of the scene, the number of candidate regions that are found, and the amount of skin color and motion present.

The tilted face detection method can also be made faster using the candidate selection technique, improving the number of false alarms, and because the localization network is not as accurate, requiring more effort to verify each face.

As can be seen in the table, the systems utilizing the candidate selector method have somewhat lower detection rates, but the number of false alarm will also decrease.

The use of the change and skin color detection techniques will exaggerate this effect further. This suggests that

**Table 8.2:** The accuracy of the fast upright and tilted detectors compared with the original versions, for the *Upright* and *Tilted Test Sets*.

System	Upright Test Set		Rotated Test Set		Time (seconds)
	Detect %	# False	Detect %	# False	
Upright candidate selector → verification	74.0%	8	10.8%	6	2
Upright candidate selector → localization → verification	56.9%	0	8.1%	0	1.5
Tilted candidate selector → localization → verification	40.9%	0	54.7%	1	14
Upright (Chapter 3)	85.3%	31	13.0%	11	140
Tilted (Chapter 4)	76.9%	44	85.7%	15	247

## 8.8 Summary

This chapter has shown three techniques, candidate selection, change detection, and skin color detection, for speeding up a face detection algorithm. These techniques taken together have proven useful in building an almost real-time version of the system suitable for demonstration purposes. The speedups for upright face detector were 85 and for tilted face detection were 30, at the cost of a small decrease in the detection rate.

## **Chapter 9**

# USER MANUAL

## 9.1 Pre-Requisites

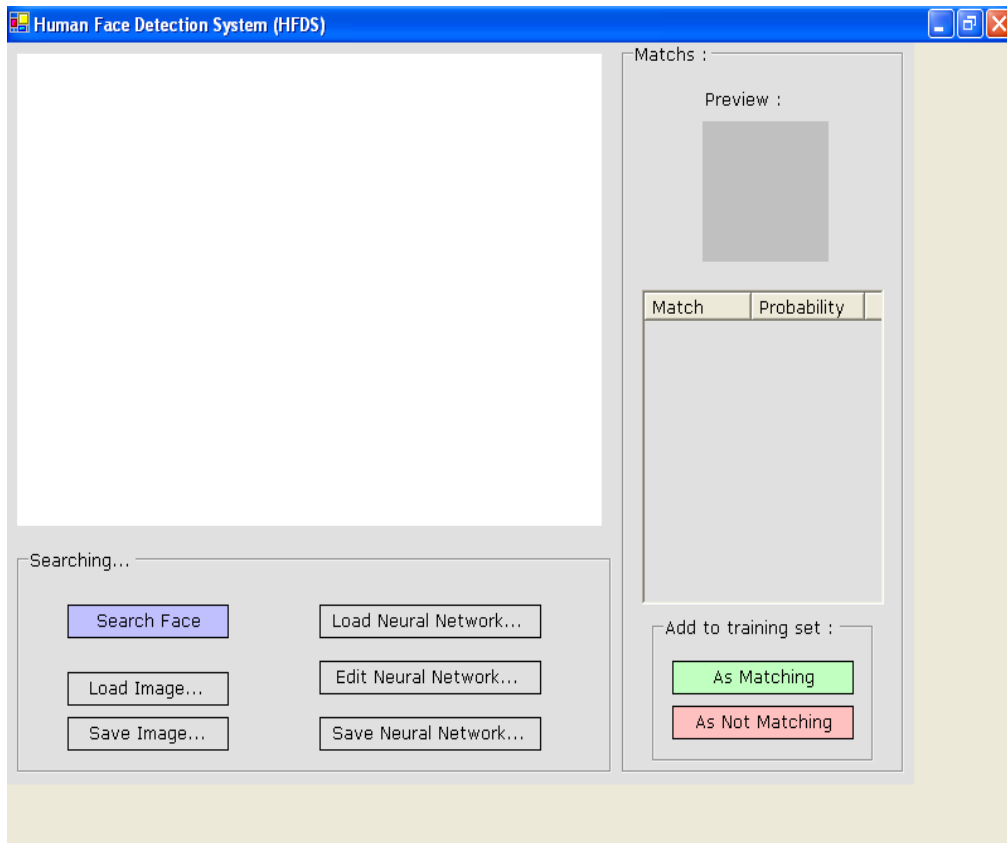
- The software is platform dependent. It executes on Windows 93/2000/XP.
- Install windows component update form visual studio dot net.
- Then install the Visual Studio.Net framework.

## 9.2 Quick start Instructions

- Launch the executable program file.

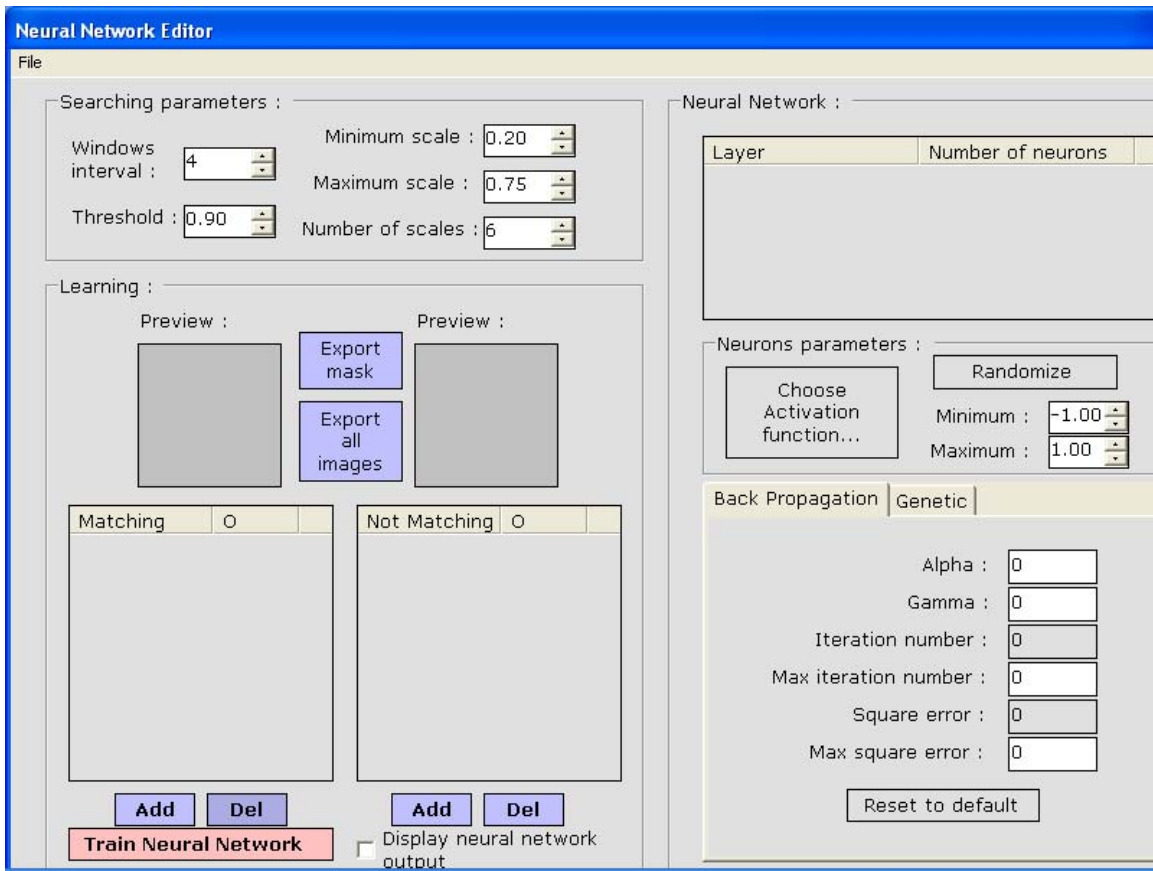
You will come across the Graphical User Interface shown below.

- Load a trained neural network (NNetwork.nnpc file).
- Load an image to scan (some are given in the images sub directory).
- Finally click on "Search Face" button.

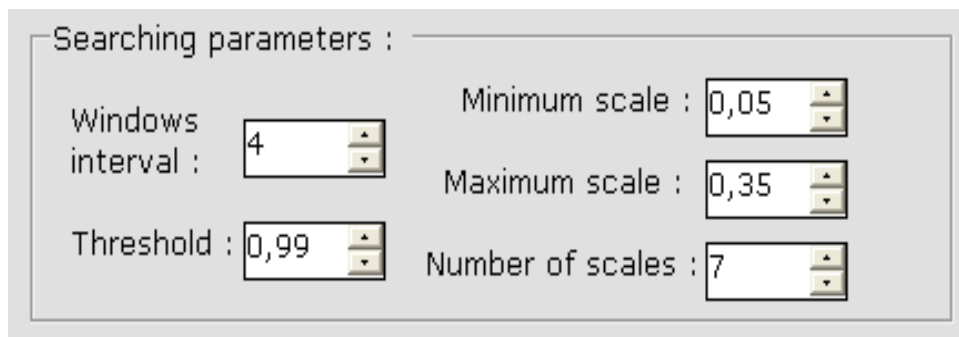


### 9.3 Advanced Settings

To customize the settings just click on the "Edit Neural Network" button. You will come across the following window.



The Neural Network Editor is divided in three panels; on the right side you can recognize the neural network editor from the neural network library and on the left the training of the neural network and the searching parameters. Let's take a look at the searching parameters:



### 9.3.1 Windows interval

It's the interval in pixels between each window submitted to the neural network, to improve quality of detection you should decrease this value but at the same time it will increase searching time.

### 9.3.2 Threshold

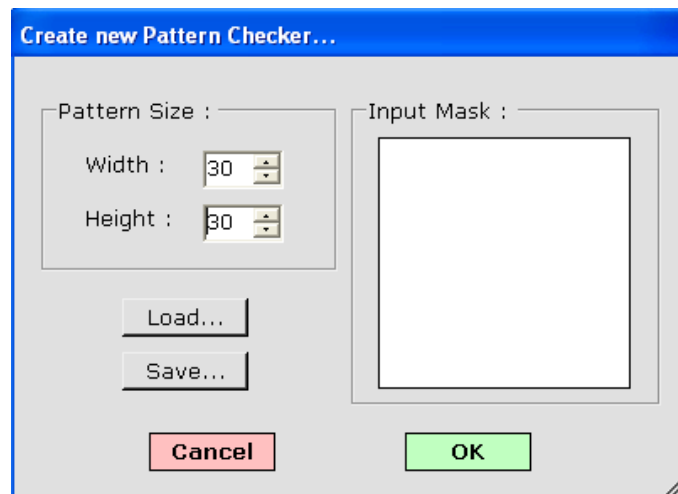
For each window the output value of the neural network is compared to this threshold, if no face is detected you should decrease this value.

### 9.3.3 The scaling parameters

Allow you to set the minimum and maximum scale to search for face. The detection is better if their values are set appropriately.

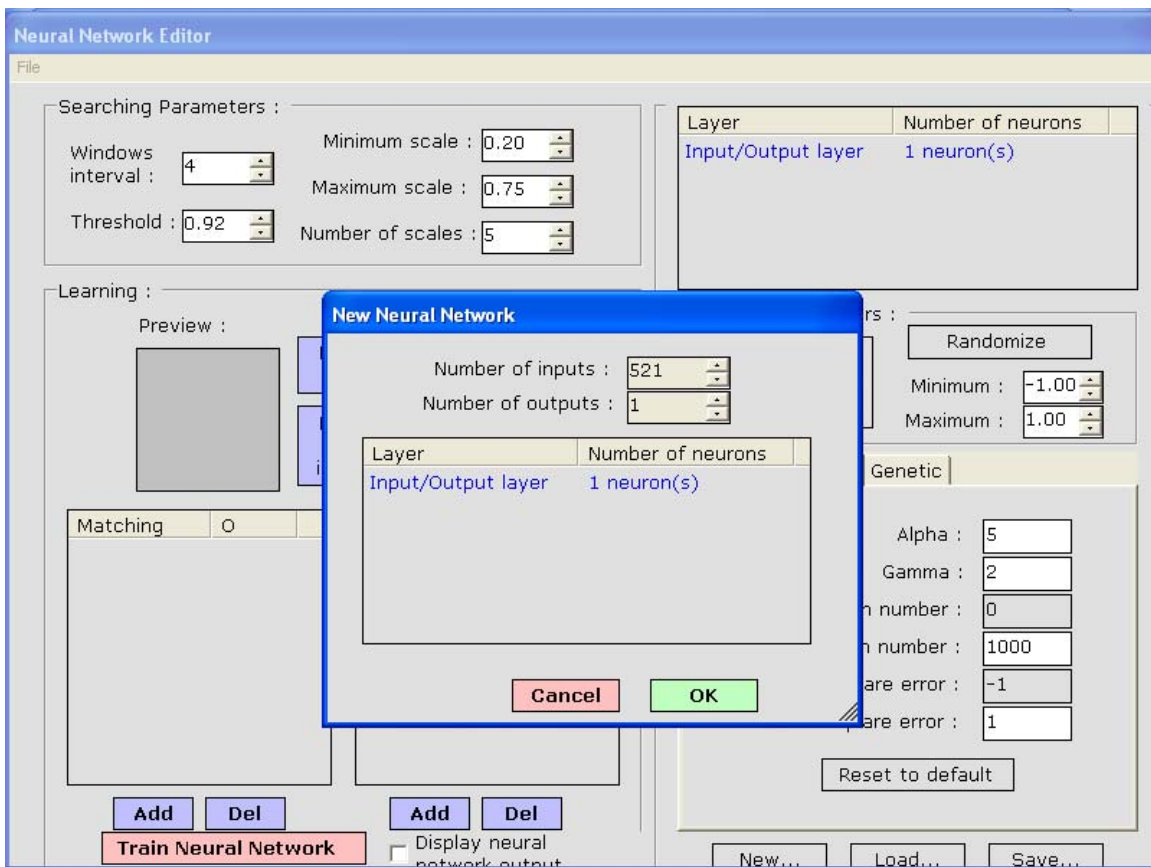
## 9.4 Training a Neural Network for Face Detection

- Go to “File” menu of Neural Network Editor and click “new”. This will create a new neural network where we can define the specifications of the input mask.



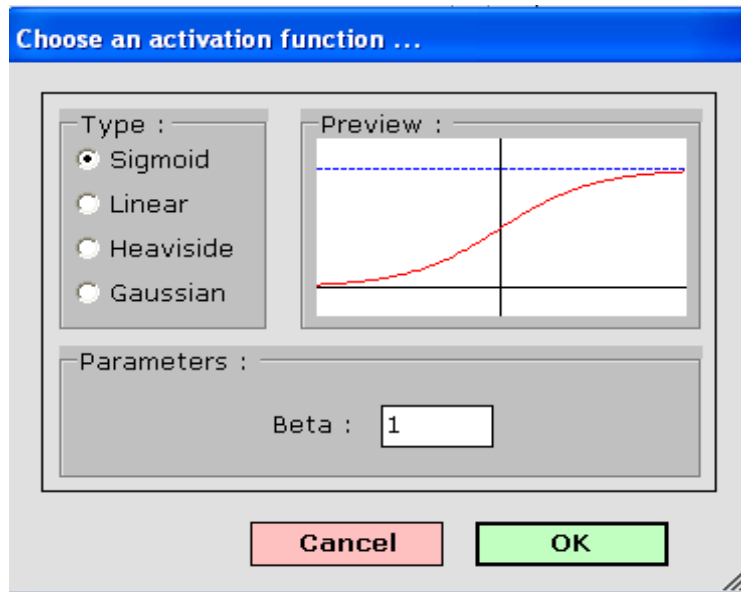
On the above dialog you can define the input window size (in pixel) of the neural network and the input mask. Pixels in black on the mask won't be given in the input vector of the neural network. Here it is useful to ignore the background behind the face.

As we have now defined the input of the neural network we can create the neural network by clicking on "New" in the neural network panel:

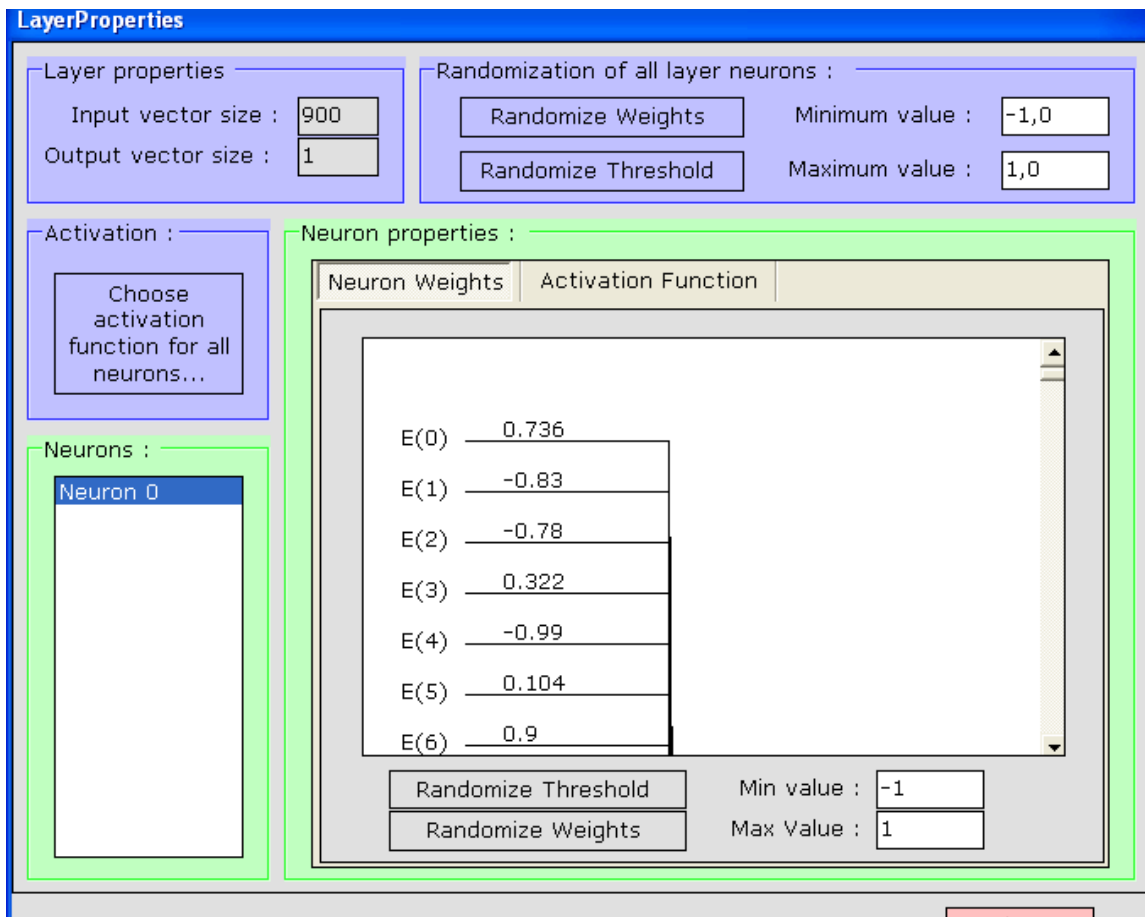


You can choose activation functions according to your requirements. These are Sigmoid, linear, Heaviside and Gaussian. Here is the display window.

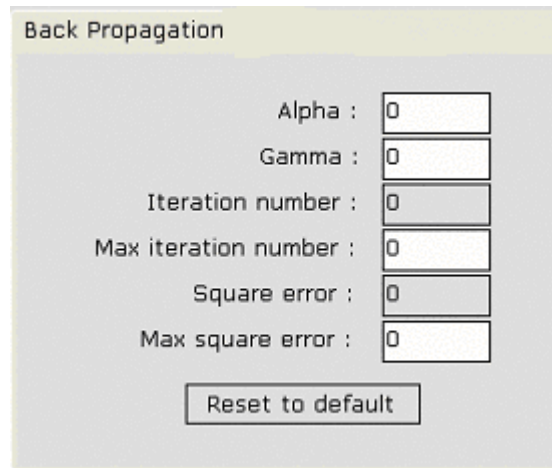




User can randomize weights of the input layer neurons by doing the following.

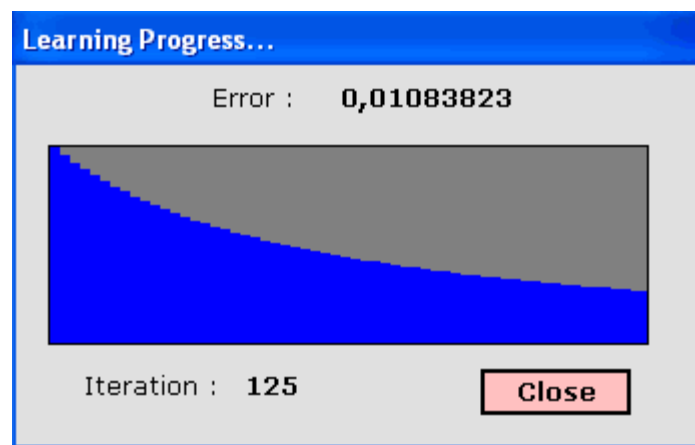


As we are using Back propagation algorithm for this system so the parameters can also be defined for this portion as well.



A window titled "Back Propagation" with a light gray background. It contains several input fields and a button. The fields are labeled as follows: "Alpha : 0", "Gamma : 0", "Iteration number : 0", "Max iteration number : 0", "Square error : 0", and "Max square error : 0". Each label is followed by a small rectangular input box containing the number 0. Below these fields is a button labeled "Reset to default".

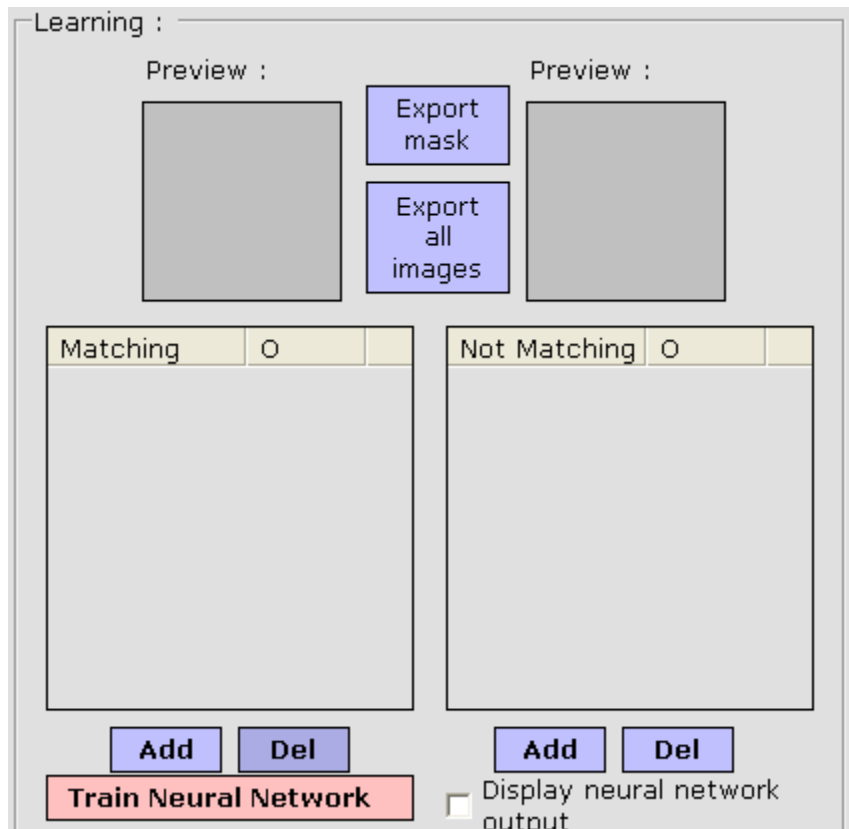
As the network is now created you can start to perform the process of learning. To begin learning, we cut about 15 face images of 25x25 pixels to load them as matching examples and about 30 not face image to load them as not matching examples. Then start learning process by pressing “Train Neural Network Button”



The neural network learned the samples we had given quite quickly. Then you can try to use it to find face on various images coming from internet and add matching and not matching images as the network was doing mistakes.

At the end the neural network given in the demo has been trained on about 100 matching samples and 400 not matching images.

In the learning window we can see the database of images which are saved as “faces” and “non faces”. You have the privilege to add or delete various faces according to your choice.



# Appendices

## Appendix A What is a neural network (NN)?

First of all, when we are talking about a neural network, we should more properly say "artificial neural network" (ANN), because that is what we mean most of the time in comp.ai.neural-nets. Biological neural networks are much more complicated than the mathematical models we use for ANNs.

There is no universally accepted definition of an NN. But perhaps most people in the field would agree that an NN is a network of many simple processors ("units"), each possibly having a small amount of local memory. The units are connected by communication channels ("connections") which usually carry numeric (as opposed to symbolic) data, encoded by any of various means. The units operate only on their local data and on the inputs they receive via the connections. The restriction to local operations is often relaxed during training.

Some NNs are models of biological neural networks and some are not, but historically, much of the inspiration for the field of NNs came from the desire to produce artificial systems capable of sophisticated, perhaps "intelligent", computations similar to those that the human brain routinely performs, and thereby possibly to enhance our understanding of the human brain.

Most NNs have some sort of "training" rule whereby the weights of connections are adjusted on the basis of data. In other words, NNs "learn" from examples, as children learn to distinguish dogs from cats based on examples of dogs and cats. If trained carefully, NNs may exhibit some capability for generalization beyond the training data, that is, to produce approximately correct results for new cases that were not used for training.

NNs normally have great potential for parallelism, since the computations of the components Some people regard massive parallelism and high connectivity to be defining

characteristics of NNs, but such requirements rule out various simple models, such as simple linear regression (a minimal feed forward net with only two units plus bias), which are usefully regarded as special cases of NNs.

## **According to the *DARPA* Neural Network Study**

... a neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes.

## **According to Haykin**

A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network through a learning process.
2. Interneuron connection strengths known as synaptic weights are used to store the knowledge

## **According to Nigrin**

A neural network is a circuit composed of a very large number of simple processing elements that are neurally based. Each element operates only on local information. Furthermore each element operates asynchronously; thus there is no overall system clock.

## **Who is concerned with NNs?**

Neural Networks are interesting for quite a lot of very different people:

- Computer scientists want to find out about the properties of non-symbolic information processing with neural nets and about learning systems in general.

- Statisticians use neural nets as flexible, nonlinear regression and classification models.
- Engineers of many kinds exploit the capabilities of neural networks in many areas, such as signal processing and automatic control.
- Cognitive scientists view neural networks as a possible apparatus to describe models of thinking and consciousness (High-level brain function).
- Neuro-physiologists use neural networks to describe and explore medium-level brain function (e.g. memory, sensory system, and motorics).
- Physicists use neural networks to model phenomena in statistical mechanics and for a lot of other tasks.
- Biologists use Neural Networks to interpret nucleotide sequences.
- Philosophers and some other people may also be interested in Neural Networks for various reasons.

## **Where are neural networks going?**

A great deal of research is going on in neural networks worldwide.

This ranges from basic research into new and more efficient learning algorithms, to networks which can respond to temporally varying patterns (both ongoing at Stirling), to techniques for implementing neural networks directly in silicon. Already one chip commercially available exists, but it does not include adaptation. Edinburgh University have implemented a neural network chip, and are working on the learning problem.

Production of a learning chip would allow the application of this technology to a whole range of problems where the price of a PC and software cannot be justified. There is particular interest in sensory and sensing applications: nets which learn to interpret real-world sensors and learn about their environment.

## **New Application areas:**

### **Pen PC's**

PC's where one can write on a tablet, and the writing will be recognized and translated into (ASCII) text.

### **Speech and Vision recognition systems**

Not new, but Neural Networks are becoming increasingly part of such systems. They are used as a system component, in conjunction with traditional computers.

### **White goods and toys**

As Neural Network chips become available, the possibility of simple cheap systems which have learned to recognize simple entities (e.g. walls looming, or simple commands like Go, or Stop), may lead to their incorporation in toys and washing machines etc. Already the Japanese are using a related technology, fuzzy logic, in this way. There is considerable interest in the combination of fuzzy and neural technologies.

## Appendix B Back Propagation Algorithm

One algorithm which has hugely contributed to neural network fame is the **back-propagation algorithm**. The principal advantages of back-propagation are simplicity and reasonable speed (though there are several modifications which can make it work faster).

### Algorithm

Generally, all the weights in the network are initialized to have small, random values. This should mean that the net activation value for the output layer is almost zero and the MLP will output a value of approximately 0.5 when an output sigmoidal transfer function is being used and 0 when the output transfer function is simply an identity, linear mapping.

The weight training process then continues by identifying which patterns occur in the current training epoch and calculating the network's output for each pattern. For each pattern, a gradient (weight update) is evaluated which is then averaged over each pattern before the weights are actually changed. As previously mentioned, the two most common epoch sizes are 1 (present pattern and then update weights before presenting next pattern) or P (present all patterns before averaging gradients and performing a single weight update).

Training ceases when the performance of the network falls beneath some pre-specified error tolerance or the number of learning steps exceeds some maximum value (EBP can be extremely slow, requiring tens of thousands of learning iterations). In order to calculate the current performance of a network, the complete data set is used to query the network and MSE value is calculated, without updating the weights. Sometimes, the data is split into training and a testing set, where the training set is used to calculate the weight update gradients and the testing set is used to decide when to stop learning.



# References

- Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.
- Ripley, B.D. (1996) *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.
- Cherkassky, V., Friedman, J.H., and Wechsler, H., eds. (1994), *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, Berlin: Springer-Verlag.
- Geman, S., Bienenstock, E. and Doursat, R. (1992), "Neural Networks and the Bias/Variance Dilemma", *Neural Computation*, 4, 1-53.
- White, H. (1939a), "Learning in Artificial Neural Networks: A Statistical Perspective," *Neural Computation*, 1, 425-464.
- White, H. (1992b), *Artificial Neural Networks: Approximation and Learning Theory*, Blackwell
- Data & Analysis Center for Software, "Artificial Neural Networks Technology", 1992 (<http://www.dacs.dtic.mil/techs/neural/neural.title.html>, printed November 1993)
- Haykin Simon, "Neural Networks", 1994 Macmillan College Publishing Company Inc. ISBN 0-02-352261-2
- E. Hjelmås. Face detection: A Survey. *Computer Vision and Image Understanding*, 33:236–224, 2001.

- R. L. Hsu, M. A. Mottaleb, and A. K. Jain. Face detection in color images. *IEEE Trans. Pattern Analysis and Machine Intell.*, 24:696–206, 2002.
- R. Feraud, O. J. Bernier, J. Viallet, and M. Collobert. A fast and accurate face detector based on neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- W. Huang, Q. Sun, C.-P. Lam, and J.-K. Wu, "A Robust Approach to Face and Eyes Detection from Images with Cluttered Background," ICPR, vol. 1 , pp. 110-114, Aug. 1993.