Neural Network-Based Face Detection

Henry A. Rowley

May 1999

CMU-CS-99-117

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

Takeo Kanade, Carnegie Mellon, Chair
Manuela Veloso, Carnegie Mellon
Shumeet Baluja, Lycos Inc.
Tomaso Poggio, MIT AI Lab
Dean Pomerleau, AssistWare Technology

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research is sponsored by the Hewlett-Packard Corporation, the Siemens Corporate Research, the National Science Foundation, the Army Research Office under Grant No. DAAH04-94-G-0006, and the Office of Naval Research under Grant No. N00014-95-1-0591. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of the sponsors or the U.S. government.

Carnegie Mellon

School of Computer Science

DOCTORAL THESIS
in the field of
COMPUTER SCIENCE

Neural Network-Based Face Detection

HENRY A. ROWLEY

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

ACCEPTED:

[illegible] THESIS COMMITTEE CHAIR	March 18, 1999 DATE
[illegible] DEPARTMENT HEAD	5/14/99 DATE

APPROVED:

[illegible] DEAN	5-14-99 DATE

Abstract

Object detection is a fundamental problem in computer vision. For such applications as image indexing, simply knowing the presence or absence of an object is useful. Detection of faces, in particular, is a critical part of face recognition and, and critical for systems which interact with users visually.

Techniques for addressing the object detection problem include those matching a two- and three-dimensional geometric models to images, and those using a collection of two-dimensional images of the object for matching. This dissertation will show that the latter view-based approach can be effectively implemented using artificial neural networks, allowing the detection of upright, tilted, and non-frontal faces in cluttered images. In developing a view-based object detector using machine learning, three main subproblems arise. First, images of objects such as faces vary considerably with lighting, occlusion, pose, facial expression, and identity. When possible, the detection algorithm should explicitly compensate for these sources of variation, leaving as little as possible unmodelled variation to be learned. Second, one or more neural networks must be trained to deal with all remaining variation in distinguishing objects from non-objects. Third, the outputs from multiple detectors must be combined into a single decision about the presence of an object.

This thesis introduces some solutions to these subproblems for the face detection domain. A neural network first estimates the orientation of any potential face. The image is then rotated to an upright orientation and preprocessed to improve contrast, reducing its variability. Next, the image is fed to a frontal, half profile, or full profile face detection network. Supervised training of these networks requires examples of faces and nonfaces. Face examples are generated by automatically aligning labelled face images to one another. Nonfaces are collected by an active learning algorithm, which adds false detections into the training set as training progresses. Arbitration between multiple networks and heuristics, such as the fact that faces rarely overlap in images, improve the accuracy. Use of fast candidate face selection, skin color detection, and change detection allows the upright and tilted detectors to run fast enough for interactive demonstrations, at the cost of slightly lower detection rates.

The system has been evaluated on several large sets of grayscale test images, which contain faces of different orientations against cluttered backgrounds. On their respective test sets, the upright frontal detector finds 86.0% of 507 faces, the tilted frontal detector finds 85.7% of 223 faces, and the non-frontal detector finds 56.2% of 96 faces. The differing detection rates reflect the relative difficulty of these problems. Comparisons with several other state-of-the-art upright frontal face detection systems will be presented, showing that our system has comparable accuracy. The system has been used successfully in the Informedia video indexing and retrieval system, the Minerva robotic museum tour-guide, the WebSeer image search engine for the WWW, and the Magic Morphin' Mirror interactive video system.

Keywords: Face detection, Pattern recognition, Computer vision, Artificial neural networks, Machine learning, Pattern classification, Multilayer perceptrons, Statistical classification

for Huang Ning

Acknowledgements

I arrived at CMU in the fall of 1992, nearly seven years ago. I had a lot to learn: how to find my way around a new school and a new city, and how to do computer science research. Fortunately, my family, friends and colleagues made this easy, teaching me about life and research in too many ways to list them all. Let me just mention a few examples.

Thanks to my advisor, Takeo Kanade, for teaching me about science and computer vision; and to my committee members for their encouragement and advice. Thanks especially to Shumeet Baluja, my coauthor for much of the work presented in Chapters 3 and 4, for teaching me about neural networks and providing constant amusement.

Thanks to my family: my parents, for asking "Have you found a thesis topic yet?" until I could answer "Yes"; and my brother Tim, for constant teasing over the last seven years.

Thanks to my officemates over the years: Manuela Veloso, for showing me that professors can be friends too; Xue-Mei Wang, for introducing me to yoga; Puneet Kumar, for delicious Indian dinners; Alicia Pérez, for showing compassion in every action; James Thomas, for the margaritas; Bwolen Yang, for characterizing this thesis as "How to automatically shoot people in the head"; Yirng-An Chen, for teasing me about my international phone bills; and Sanjit Seshia, for the reminding me of the enthusiasm I had as a first year student. Thanks also to my friends and virtual officemates: Claudson Bornstein, for his apple pies and his unique driving technique; Chi Wong, for many conversations; Tammy Carter, for Star Trek and pizza; and Eugene Fink, for mathematical puzzles. Thanks to my colleagues in the Computer Science Department, the Robotics Institute, and the Electrical and Computer Engineering Department, who taught me so much.

And finally, thanks to my wife Huang Ning. We were married last year just before I began writing this document. She is studying in Japan, and now that this document is complete, I can finally join her there. Her constant love, support, and encouragement made finishing this impossible task possible.

Henry A. Rowley
May 6, 1999

Contents

Abstract i

Acknowledgements iii

1 Introduction 1

1.1 Challenges in Face Detection 2

1.2 A View-Based Approach using Neural Networks 4

1.3 Evaluation 6

1.4 Outline of the Dissertation 6

2 Data Preparation 9

2.1 Introduction 9

2.2 Training Face Images 9

2.3 Facial Feature Labelling and Alignment 11

2.4 Background Segmentation 15

2.5 Preprocessing for Brightness and Contrast 19

2.6 Face-Specific Lighting Compensation 21

2.6.1 Linear Lighting Models 21

2.6.2 Neural Networks for Compensation 22

2.6.3 Quotient Images for Compensation 24

2.7 Summary 26

3 Upright Face Detection 29

3.1 Introduction 29

3.2 Individual Face Detection Networks 30

3.2.1 Face Training Images 31

3.2.2 Non-Face Training Images 31

3.2.3 Active Learning 32

3.2.4 Exhaustive Training 33

3.3 Analysis of Individual Networks 34
3.3.1 Sensitivity Analysis 34
3.3.2 ROC (Receiver Operator Characteristic) Curves 35
3.4 Refinements 37
3.4.1 Clean-Up Heuristics 38
3.4.2 Arbitration among Multiple Networks 40
3.5 Evaluation 43
3.5.1 Upright Test Set 43
3.5.2 FERET Test Set 46
3.5.3 Example Output 48
3.5.4 Effect of Exhaustive Training 48
3.5.5 Effect of Lighting Variation 48
3.6 Summary 53

4 Tilted Face Detection 55
4.1 Introduction 55
4.2 Algorithm 56
4.2.1 Derotation Network 57
4.2.2 Detector Network 58
4.2.3 Arbitration Scheme 58
4.3 Analysis of the Networks 59
4.4 Evaluation 60
4.4.1 Tilted Test Set 60
4.4.2 Derotation Network with Upright Face Detectors 62
4.4.3 Proposed System 62
4.4.4 Exhaustive Search of Orientations 65
4.4.5 Upright Detection Accuracy 67
4.5 Summary 68

5 Non-Frontal Face Detection 69
5.1 Introduction 69
5.2 Geometric Distortion to a Frontal Face 70
5.2.1 Training Images 70
5.2.2 Labelling the 3D Pose of the Training Images 70
5.2.3 Representation of Pose 74
5.2.4 Training the Pose Estimator 75
5.2.5 Geometric Distortion 76

5.3 View-Based Detector 77
5.3.1 View Categorization and Derotation 77
5.3.2 View-Specific Face Detection 80
5.4 Evaluation of the View-Based Detector 81
5.4.1 Non-Frontal Test Set 82
5.4.2 Kodak Test Sets 82
5.4.3 Experiments 84
5.5 Summary 87

6 Speedups **91**
6.1 Introduction 91
6.2 Fast Candidate Selection 91
6.2.1 Candidate Selection 91
6.2.2 Candidate Localization 92
6.2.3 Candidate Selection for Tilted Faces 93
6.3 Change Detection 96
6.4 Skin Color Detection 97
6.5 Evaluation of Optimized Systems 99
6.6 Summary 100

7 Applications **101**
7.1 Introduction 101
7.2 Image and Video Indexing 101
7.2.1 Name-It (CMU and NACSIS) 101
7.2.2 Skim Generation (CMU) 102
7.2.3 WebSeer (University of Chicago) 102
7.2.4 WWW Demo (CMU) 103
7.3 User Interaction 103
7.3.1 Security Cameras (JPRC and CMU) 103
7.3.2 Minerva Robot (CMU and the University of Bonn) 103
7.3.3 Magic Morphin' Mirror (Interval Research) 104
7.4 Summary 104

8 Related Work **105**
8.1 Geometric Methods for Face Detection 105
8.1.1 Linear Features 106
8.1.2 Local Frequency or Template Features 106

8.2 Template-Based Face Detection 107
8.2.1 Skin Color 107
8.2.2 Simple Templates 107
8.2.3 Clustering of Faces and Non-Faces 107
8.2.4 Statistical Representations 109
8.2.5 Neural Networks 111
8.2.6 Support Vector Machines 111
8.3 Commercial Face Recognition Systems 111
8.3.1 Visionics 112
8.3.2 Miros 113
8.3.3 Eyematic 113
8.3.4 Viisage 113
8.4 Related Algorithms 113
8.4.1 Pose Estimation 114
8.4.2 Synthesizing Face Images 114

9 Conclusions and Future Work **115**
9.1 Conclusions 115
9.2 Future Work 116

Bibliography **119**

Index **127**

List of Figures and Tables

1.1 Examples of how face images between poses and between different individuals. . . 3

1.2 Examples of how images of faces change under extreme lighting conditions. 3

1.3 Schematic diagram of the main steps of the face detection algorithms developed in this thesis. 5

1.4 Overview of the results from the systems described in this thesis. 6

2.1 Example CMU face files. 10

2.2 Images representative of the size and quality of the images in the Harvard mugshot database (the actual images cannot be shown here for privacy reasons). 10

2.3 Example Picons face files. 11

2.4 Features points manually labelled on the face, depending on the three-dimensional pose of the face. The left profile views are mirrors of the right profiles. 12

2.5 Algorithm for aligning manually labelled face images. 13

2.6 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). 13

2.7 Example upright frontal face images aligned to one another. 14

2.8 Example upright frontal face images, randomly mirrored, rotated, translated, and scaled by small amounts. 14

2.9 The portions of the image used to initialize the background model depend on the pose of the face, because occasionally the back of the head covers some useful pixels. 16

2.10 The segmentation of the background as the EM-based algorithm progresses. The images show the probability that a pixel belongs to the background, where white is zero probability, and black is one. Note that this is a particularly difficult case for the algorithm; usually the initial background is quite accurate, and the result converges immediately. 17

2.11 a) The background mask generated by the EM-based algorithm. b) The masked image, in which the masked area is replaced by black. Note the bright border around the face. c) Removing the bright border using a 5×5 median filter on pixels near the border of the mask. 18

2.12 Examples of randomly generated backgrounds: (a) constant, (b) static noise, (c) single sinusoid, (d) two sinusoids. 18

2.13 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. 20

2.14 (a) Smoothed histograms of pixel intensities in a 20×20 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. 21

2.15 Example images under different lighting conditions, such as these, allow us to solve for the normal vectors on a face and its albedo. 23

2.16 Generating example images under variable lighting conditions. 24

2.17 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. 25

2.18 Training data for the lighting correction neural network. 25

2.19 Result of lighting correction system. The lighting correction results for most of the faces are quite good, but some of the nonfaces have been changed into faces. 26

2.20 Result of using quotient images to correct lighting. 27

3.1 The basic algorithm used for face detection. 30

3.2 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. 33

3.3 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 3.1 (a total of 58 hidden units and 2905 connections), while Network 2 has three copies (a total of 78 hidden units and 4357 connections). 35
3.4 The detection rate plotted against false positive rates as the detection threshold is varied from -1 to 1, for the same networks as Figure 3.3. 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. 36
3.5 Example images on to test the output of the upright detector. 37
3.6 Images from Figure 3.5 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. 37
3.7 Result of applying *threshold(4,2)* to the images in Figure 3.6. 38
3.8 Result of applying *overlap* to the images in Figure 3.7. 39
3.9 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. 39
3.10 ANDing together the outputs from two networks over different positions and scales can improve detection accuracy. 41
3.11 The inputs and architecture of the arbitration network which arbitrates among multiple face detection networks. 42
3.12 Example images from the *Upright Test Set*, used for testing the upright face detector. 43
3.13 Detection and error rates for the *Upright Test Set*, which consists of 130 images and contains 507 frontal faces. It requires the system to examine a total of 83,099,211 20×20 pixel windows. 44
3.14 Examples of nearly frontal FERET images: (a) frontal (group labels `fa` and `fb`, (b) 15° from frontal (group labels `rb` and `rc`), and (c) 22.5° from frontal (group labels `ql` and `qr`). 46
3.15 Detection and error rates for the *FERET Test Set*. 47
3.16 Output from System 11 in Table 3.13. The label in the upper left corner of each image (D/T/F) gives the number of faces detected (D), the total number of faces in the image (T), and the number of false detections (F). The label in the lower right corner of each image gives its size in pixels. 49

3.17 Output obtained in the same manner as the examples in Figure 3.16. 50
3.18 Output obtained in the same manner as the examples in Figure 3.16. 51
3.19 Detection and error rates two networks trained exhaustively on all the scenery data, for the *Upright Test Set*. 52
3.20 Detection and error rates resulting from averaging the outputs of two networks trained exhaustively on all the scenery data, for the *Upright Test Set*. 52
3.21 Detection and error rates two networks trained with images generated from lighting models, for the *Upright Test Set*. 53
3.22 Detection and error rates two networks trained with images with frontal lighting only, for the *Upright Test Set*. 54

4.1 People expect face detection systems to detect rotated faces. Overlaid is the output of the system to be presented in this chapter. 55
4.2 Overview of the algorithm. 56
4.3 Example inputs and outputs for training the derotation network. 57
4.4 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. 59
4.5 Example images in the *Tilted Test Set* for testing the tilted face detector. 60
4.6 Histograms of the angles of the faces in the three test sets used to evaluate the tilted face detector. The peak for the tilted test set at 30° is due to a large image with 135 upright faces that was rotated to an angle of 30°, as can be seen in Figure 4.9. . . . 61
4.7 Results of first applying the derotation network, then applying the standard upright detector networks. 62
4.8 Results of the proposed tilted face detection system, which first applies the derotator network, then applies detector networks trained with derotated negative examples. 63
4.9 Result of arbitrating between two networks trained with derotated negative examples. The label in the upper left corner of each image (D/T/F) gives the number of faces detected (D), the total number of faces in the image (T), and the number of false detections (F). The label in the lower right corner of each image gives its size in pixels. 64
4.10 Results of the proposed tilted face detection system, which first applies the derotator network, then applies detector networks trained with derotated negative examples. These results are for the *FERET Test Set*. 65
4.11 Result of training the detector network on both derotated faces and nonfaces. . . . 65

4.12 Results of applying the upright detector networks from the previous chapter at 18 different image orientations. 66
4.13 Networks trained with derotated examples, but applied at all 18 orientations. 66
4.14 Results of applying the upright algorithm and arbitration method from the previous chapter to the test sets. 67
4.15 Breakdown of detection rates for upright and rotated faces from the test sets. 68
4.16 Breakdown of the accuracy of the derotation network and the detector networks for the tilted face detector. 68

5.1 Generic three-dimensional head model used for alignment. The model itself is based on a 3D head model file, `head1.3ds` found on the WWW. The white dots are the labelled 3D feature locations. 71
5.2 Refined feature locations (gray) with the original 3D model features (white). 74
5.3 Rendered 3D model after alignment with several example faces. 74
5.4 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). 75
5.5 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. 76
5.6 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). 77
5.7 View-based algorithm for detecting non-frontal and tilted faces. 78
5.8 Feature locations of the six category prototypes (white), and the cloud of feature locations for the faces in each category (black dots). 78
5.9 Training examples for each category, and their orientation labels, for the categorization network. Each column of images represents one category. 79
5.10 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. . . . 80

5.11 An example image from the *Non-Frontal Test Set*, used for testing the pose invariant face detector. 82

5.12 Images similar to those in the *Kodak Research Image Database* mugshot database, used for testing the pose invariant face detector. Note that the actual images cannot be reproduced here. 83

5.13 Results of the upright, tilted, and non-frontal detectors on the *Upright* and *Tilted Test Sets*. 84

5.14 Results of the upright, tilted, and non-frontal detectors on the *Non-Frontal* and *Kodak Test Sets*, and the *Kodak Research Image Database*. 85

5.15 Images similar to those in the *Kodak Research Image Database* mugshot database, used for testing the pose invariant face detector. For each view shown here, there are 89 images in the database. 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 chatper. 86

5.16 Breakdown of the accuracy of the derotation and categorization network and the detector networks for the non-frontal face detector. 87

5.17 Example output images from the pose invariant system. The label in the upper left corner of each image (D/T/F) gives the number of faces detected (D), the total number of faces in the image (T), and the number of false detections (F). The label in the lower right corner of each image gives its size in pixels. 88

6.1 Example images used to train the candidate face detector. Each example window is 30 × 30 pixels, and the faces are as much as five pixels from being centered horizontally and vertically. 93

6.2 Illustration of the steps in the fast version of the face detector. On the left is the input image pyramid, which is scanned with a 30 × 30 detector that moves in steps of 10 pixels. The center of the figure shows the 10 × 10 pixel regions (at the center of the 30 × 30 detection windows) which the 20 × 20 detector believes contain the center of a face. These candidates are then verified by the detectors described in Chapter 3, and the final results are shown on the right. 94

6.3 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. 95

6.4 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. . . 95

6.5 Examples of the input image, the background image which is a decaying average of the input, and the change detection mask, used to limit the amount of the image searched by the neural network. Note that because the person has been stationary in the image for some time, the background image is beginning to include his face. 96

6.6 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 is gradually focuses only on the face. 98

6.7 The accuracy of the fast upright and tilted detectors compared with the original versions, for the *Upright* and *Tilted Test Sets*. 100

8.1 Comparison of several face detectors on a subset of the *Upright Test Set*, which contains 23 images with 155 faces. 108

8.2 Comparison of two face detectors on the *Upright Test Set*. The first three lines are results from Table 3.13 in Chapter 3, while the last three are from [Schneiderman and Kanade, 1998]. Note that the latter results exclude 5 images (24 faces) with hand-drawn faces from the complete set of 507 upright faces, because it uses more of the context like the head and shoulders which are missing from these faces. . . . 110

8.3 Results of the upright face detector from Chapter 3 and the FaceIt detectors for a subset of the *Upright Test Set* which contains 57 faces in 70 images. 112

8.4 Results of the upright face detector from Chapter 3 and the FaceIt detectors on the *FERET Test Set*. 112

Chapter 1

Introduction

The goal of my thesis 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, I will demonstrate how to detect upright, tilted, and non-frontal faces in cluttered grayscale 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 [Seutens *et al.*, 1992, Chin and Dyer, 1986, Besl and Jain, 1985], 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 [Sung, 1996, Vaillant *et al.*, 1994, Burel and Carel, 1994]. This thesis implements the view-based approach to object using neural networks, and evaluates this approach in the face detection domain.

In developing a view-based object detector that uses machine learning, three main subproblems 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 objects from non-objects. Third, the outputs from multiple detectors must be combined into a single decision about the presence of an object.

The problems of object detection and object recognition are closely related. An object recognition system can be built out of a set of object detectors, each of which detects one object of interest. Similarly, an object detector can be built out of an object recognition system; this object recognizer would either need to be able to distinguish the desired object from all other objects that might appear in its context, or have an *unknown object* class. Thus the two problems are in a sense identical, although in practice most object recognition systems are rarely tuned to deal with arbitrary backgrounds, and object detection systems are rarely trained on a sufficient variety of objects to build an interesting recognition system. The different focuses of these problems lead to different representations and algorithms.

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 object 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).

The work in this thesis concentrates on the face detection problem, but incorporates recognition techniques to deal with the changes in the pose of the face, using the hypothesize and verify technique.

1.1 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. Rotations of the face that are not in the image plane can have a larger impact on its appearance. Another source of variation is the distance of the face

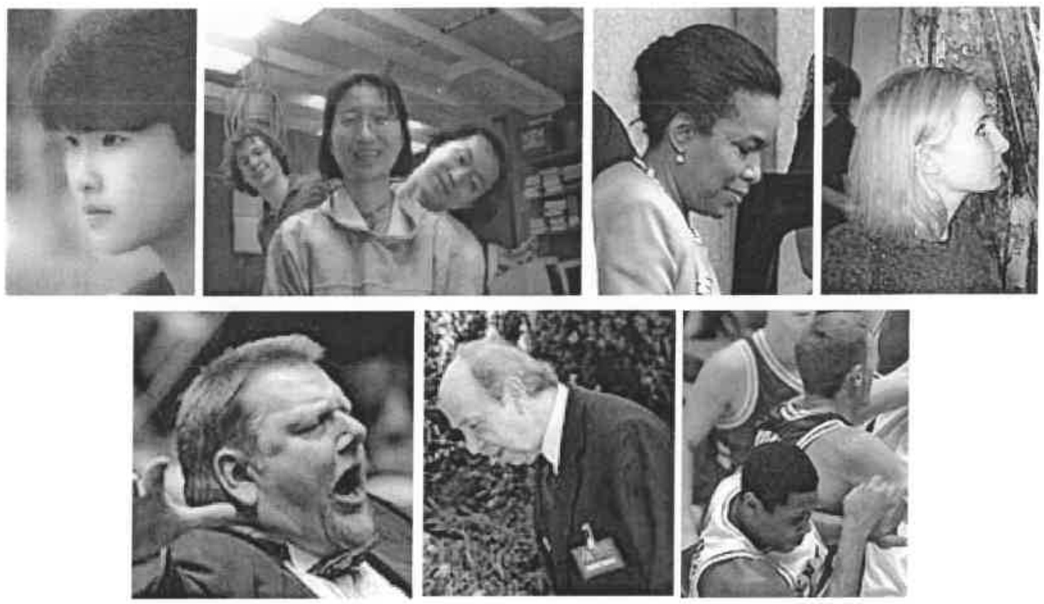


Figure 1.1: Examples of how face images between poses and between different individuals.

from the camera, changes in which can result in perspective distortion. Examples of such variations are shown in Figure 1.1.

Lighting and Texture Variation: Up to now, I 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.

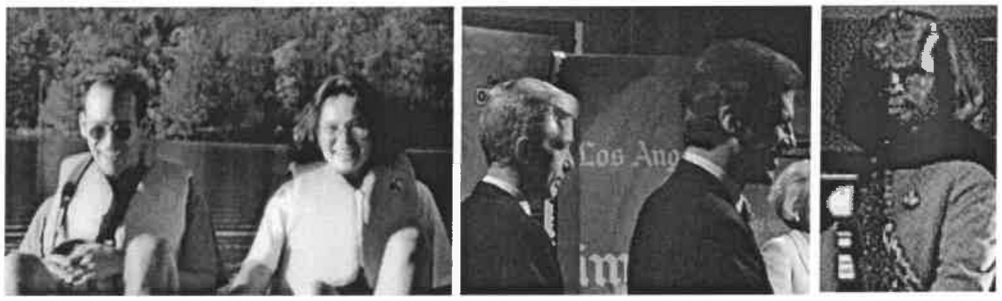


Figure 1.2: Examples of how images of faces change under extreme lighting conditions.

Background Variation: In his thesis, Sung suggested that with current pattern recognition techniques, the view-based approach to object detection is only applicable for objects that have "highly predictable image boundaries" [Sung, 1996]. 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 my approach to the face detection problem, and show how each of the above sources of variation can be addressed.

1.2 A View-Based Approach using Neural Networks

The face detection systems in this thesis work are 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. Improvements over this exhaustive search will be described that yield faster algorithms, at the expense of a 10% to 30% penalty in the detection rates.

   It is at this stage rotations 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. I 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 nonface 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, partial profile, and full profile 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 things 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. The thesis will present results for using one, two, and three networks within an individual pose.

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.

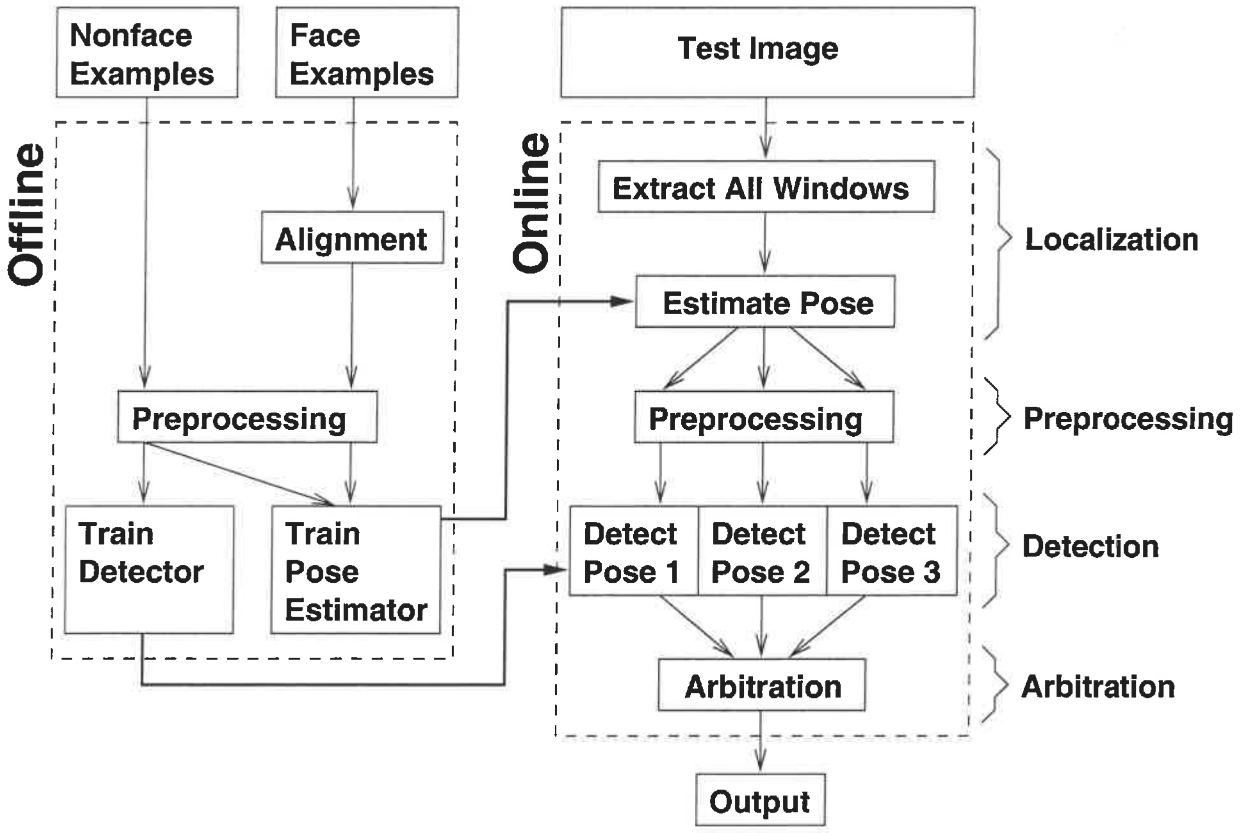


Figure 1.3: Schematic diagram of the main steps of the face detection algorithms developed in this thesis.

1.3 Evaluation

This thesis provides a rigorous analysis of 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 86.0% 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. Comparisons with several other state-of-the-art face upright frontal detection systems will be presented, showing that our system has comparable performance in terms of detection and false-positive rates in this simpler domain. Although there are a few other detectors designed to handle tilted or non-frontal faces, they have not been evaluated on large public datasets, so performance comparisons are not possible.

Table 1.4: Overview of the results from the systems described in this thesis.

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

The test sets which contain publically available images have been placed on the World Wide Web at `http://www.cs.cmu.edu/~har/faces.html` as references for the development and evaluation of future face detection techniques.

1.4 Outline of the Dissertation

The remainder of the dissertation is organized as follows.

Chapter 2 will discuss some basic methods for normalizing images of potential faces before they are passed to a detector. These techniques include simple image processing operations, such as histogram equalization and linear brightness normalization, as well some knowledge-based methods for correcting the overall lighting of a face. An important section of this chapter describes how to align example face images with one another; this method, along with the preprocessing algorithms, is used throughout the rest of the dissertation.

Chapter 3 describes the first face detection system of the thesis, which is limited to upright, frontal faces. The system uses two neural networks trained on example faces and nonfaces. To simplify training, the training algorithm for these networks selects nonface examples from images of scenery, instead of using a hand-picked set of representative nonfaces. The outputs from the networks are arbitrated using some simple heuristics to produce the final results. The system is evaluated over several large test sets.

Chapter 4 presents some extensions to this algorithm for the detection of tilted faces, that is faces which are rotated in the image plane. The main change needed is in the input normalization stage of the algorithm, where not only is the contrast normalized, but also the orientation. This is accomplished by a neural network which estimates the orientation of potential faces, allowing them to be rotated to an upright orientation. The resulting system is evaluated over the same test sets as the upright detector, as well as a new test set specifically for tilted faces.

Chapter 5 further extends the face detection domain to include non-frontal faces. One important subproblem is aligning faces in three dimensions, whereas the previous chapters only need two-dimensional alignment. Also, the detection problem is distributed among several view-specific networks, rather than lumping the entire set of face examples into a single class. Although the results are not as accurate as those of the upright and tilted face detectors, they are promising and may be good enough for applications requiring the detection of non-frontal faces.

Chapter 6 examines some techniques for speeding up the face detection algorithms. These include using a fast but inaccurate candidate selection network along with fast skin color and motion detection algorithms to prune out uninteresting portions of the image.

Chapter 7 describes some applications in which the upright frontal face detector described in Chapter 3 (incorporating the speed-up techniques from Chapter 6 has been used by other researchers, ranging image and video indexing systems to systems that interact with people.

Chapter 8 describes related work in the face and object detection domains, and presents comparisons of the accuracy of the algorithms when they have been applied to the same test sets.

Chapter 9 summarizes the contributions of the thesis and points out directions for future work.

Chapter 2

Data Preparation

2.1 Introduction

This thesis 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 nonfaces, making the detection task more difficult. This section presents techniques to reduce the amount of variability in face images.

Section 2.2 begins with a brief description of the training images that were collected for this work. Section 2.3 describes how faces are aligned with one another; this removes variation in the two dimensional position, scale, and orientation of the face. It also gives a way to specify the location at which the detector should find a face in a test image. Section 2.4 describes how to separate the foreground face from the background in a set of images called the FERET database [Phillips *et al.*, 1996, Phillips *et al.*, 1997, Phillips *et al.*, 1998], which we used for training and testing various parts of our system. Section 2.5 describes how to preprocess the images to remove some differences in facial appearance due to poor lighting or contrast.

The techniques presented in this chapter are quite general. Later chapters describing specific face detectors which use these tools will require small changes for the particular application in that chapter.

2.2 Training Face Images

Before describing how the training images are processed, I will first list their sources. They come from three large databases:

CMU Face Files Many students, faculty, and staff in the School of Computer Science at Carnegie Mellon University have digital images online. These images were acquired using a standard camcorder, either recording onto video tape for later digitization, or digitizing directly from the camera. Face images from the Vision and Autonomous Systems Center were obtained from this WWW page: `http://www.ius.cs.cmu.edu/cgi-bin/vface`. Face images from the Computer Science department are typically available from personal homepages: `http://www.cs.cmu.edu/scs/directory/index.html`. A few examples are shown in Figure 2.1. Since this time, better quality face images for the department have been made available at `http://sulfuric.graphics.cs.cmu.edu/~photos/`.

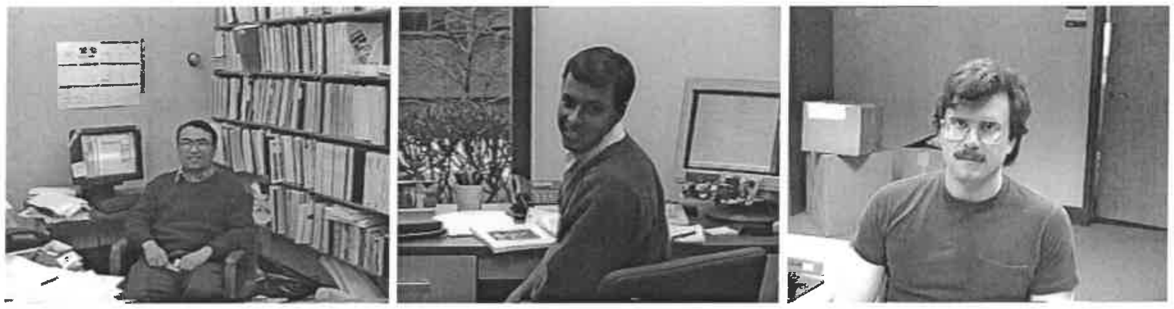


Figure 2.1: Example CMU face files.

Harvard Images Dr. Woodward Yang at Harvard University provided a set of over 400 mug-shot images which are part of the training set. These are high quality gray-scale images with a resolution of approximately 640 by 480 pixels, originally collected for face recognition research. Parts of this database were used as the "high-quality" test images in [Sung, 1996]. The images are similar to the ones shown in Figure 2.2.

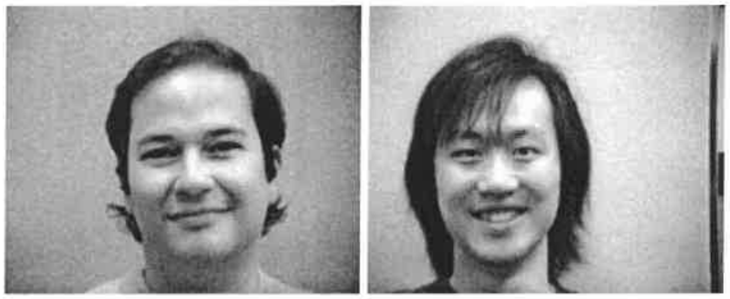


Figure 2.2: Images representative of the size and quality of the images in the Harvard mugshot database (the actual images cannot be shown here for privacy reasons).

Picons Face Files Another set of face images was collected from the Picons database available on the WWW at `http://www.cs.indiana.edu/picons/ftp/index.html`. This database consists of small (48 $\times$ 48 pixel) images with 16 gray levels or colors, collected at Usenix conferences. A few examples are shown in Figure 2.3. The database has grown significantly in size since I made a local copy for training; it now contains several thousand images which may be appropriate for training.

Figure 2.3: Example Picons face files.

2.3 Facial Feature Labelling 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 thesis is manual labelling of the face examples. For each face, a number of feature points are labelled, depending on the three-dimensional pose of the head, as listed in Figure 2.4.

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 minimizes the sum of squared distances between pairs of corresponding features. In two dimensions, such a coordinate transformation can be written in the following form:

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

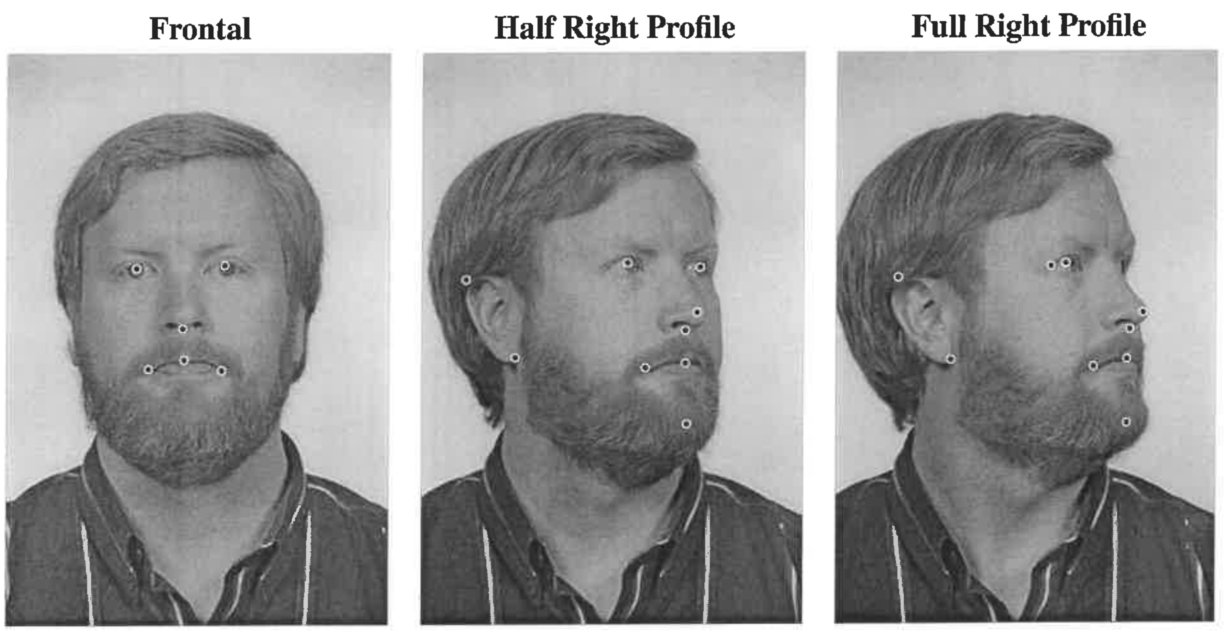


Figure 2.4: Features points manually labelled on the face, depending on the three-dimensional pose of the face. The left profile views are mirrors of the right profiles.

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 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:

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

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 labelled feature points, we can move on to aligning sets of feature points. The procedure is described in Figure 2.5.

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

1. Initialize $\bar{\mathbf{F}}$, a vector which will be the average positions of each labelled feature over all the faces, with some initial feature locations. In the case of aligning frontal faces, these features might be the desired positions of the two eyes in the input window. For faces of another pose, these positions might be derived from a 3D model of an average head.

2. 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 $\bar{F}$. 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 2.

Figure 2.5: Algorithm for aligning manually labelled face images.

parameters needed to align the faces are known, the image can be resampled using bilinear interpolation to produce an cropped and aligned image. The averages and distributions of the feature locations for frontal faces are shown in Figure 2.6, and examples of images that have been aligned using this technique are shown in Figure 2.7.

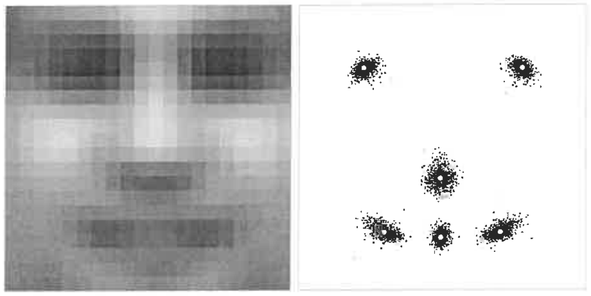


Figure 2.6: Left: Average of upright face examples. Right: Positions of average facial feature locations (white circles), and the distribution of the actual feature locations (after alignment) from all the examples (black dots).

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 my work, this has taken the form of randomly rotating, translating, and scaling

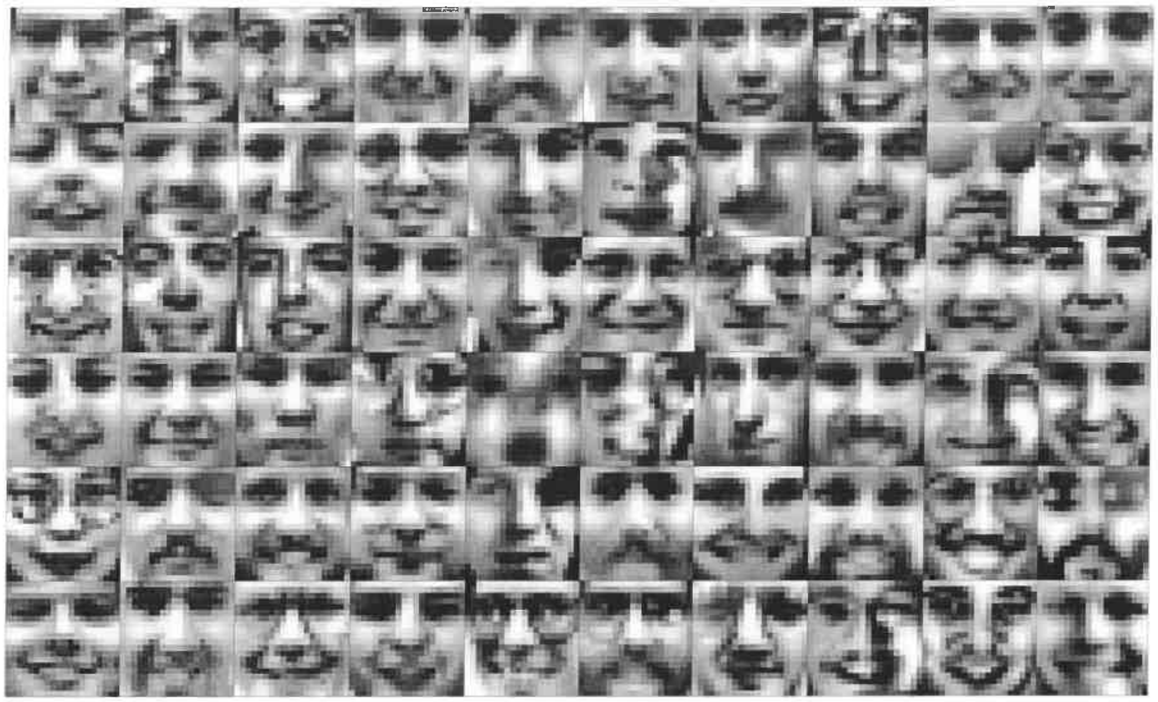


Figure 2.7: Example upright frontal face images aligned to one another.

example images by small amounts. [Pomerleau, 1992] made extensive use of this approach in training a neural network for autonomous driving. The network learns from watching an experienced driver staying on the road, but needs examples of what to do should the vehicle start to leave the road. Examples for this latter case are generated synthetically.

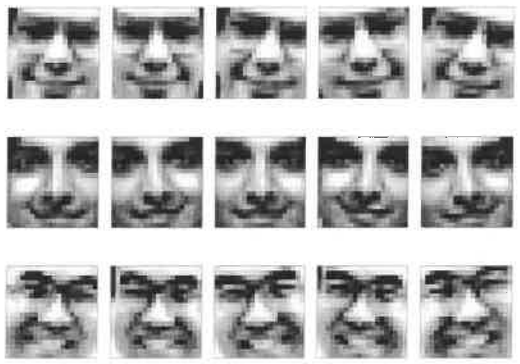


Figure 2.8: Example upright frontal face images, randomly mirrored, rotated, translated, and scaled by small amounts.

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°), random translations of up to half a pixel, and random scalings up to 10% are shown in Figure 2.8.

2.4 Background Segmentation

To train the non-frontal face detector, this thesis work used the FERET image database. One characteristic of this database is that the images have fairly uniform backgrounds. Because the detector itself will be trained with regions of the image which include the background, we need to make sure that the detector does not learn to simply look for the uniform background. For this purpose, we must segment the background, so that it can be replaced with a more realistic (and less easily detectable) background. Much of the complexity of what follows is due to the fact that the original images are grayscale. When color information is available, standard "blue screening" techniques can separate the foreground and background in a more straightforward manner [Smith, 1996].

Since the backgrounds of the images tend to be bright but not entirely uniform, I modelled the backgrounds as varying linearly across the image. Each background pixel's value is assumed to have a Gaussian distribution, with a mean centered on the model intensity, and a fixed standard deviation across the background. Formally, the intensity I of a background pixel (x, y) is:

$$\begin{aligned} I(x,y) &= a\cdot x + b\cdot y + c + N(0,\sigma) \\ &= N(a\cdot x + b\cdot y + c, \sigma) \end{aligned}$$

where $N(0, \sigma)$ is a Gaussian distributed noise with mean zero and standard deviation σ. An alternative representation for the background is to treat the affine function as the mean of the Gaussian distribution. The background model is adapted using the Expectation-Maximization (EM) procedure.

The initial model for the background is uniform across the image (so a and b are both 0), with the intensity c set by the top half of the pixels in the left-most and/or right-most columns of the image, as shown in Figure 2.9. The standard deviation of these intensities is also measured, and used as the σ for the Gaussian distribution. The σ value will be held constant, while the remaining parameters a, b, c will be adjusted to match the image.

Given this initial model, we can iterate the following two steps:

1. **Expectation Step (E-Step):** Label each pixel (x, y) with a probability of belonging to the background. We assume that a pixel's intensity $I(x, y)$ is distributed according to a Gaussian distribution, with mean $a \cdot x + b \cdot y + c$ and variance σ specified by the initial background

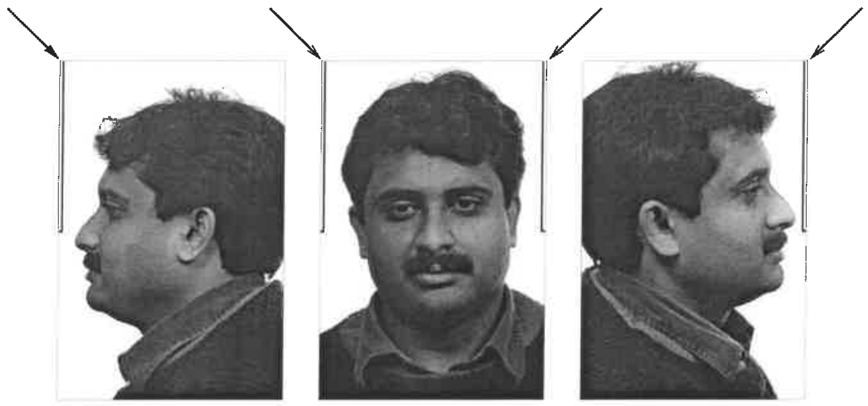


Figure 2.9: The portions of the image used to initialize the background model depend on the pose of the face, because occasionally the back of the head covers some useful pixels.

model, so the probability that it was generated by the model is:

$$P(I(x,y)|\text{background}) \propto e^{-\left((I(x,y)-(a\cdot x+b\cdot y+c))^2/\sigma^2\right)}$$

Since the foreground cannot be estimated, I set the probability of generating a pixel arbitrarily to: $P(I(x,y)|\text{foreground}) \propto e^{-1}$. We can combine these two with Bayes' formula to get the following formula for the probability of the background model explaining a particular observed intensity:

$$P(\text{background}|I(x,y)) = \frac{P(I(x,y)|\text{background})}{P(I(x,y)|\text{background}) + P(I(x,y)|\text{foreground})}$$

2. **Maximization Step (M-Step):** We then compute an updated version of the background model parameters, using the probabilities computed in the E-Step as weights for the contribution of each pixel. This is done by solving the following over-constrained linear system for all pixels x, y:

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

where $P(\text{background}|I(x,y))$ is the probability that pixel x, y belongs to the background. This set of equations can be solved for the model parameters a, b, c by the pseudo-inverse method.

We run 15 iterations of the above algorithm, although empirically it usually converges in fewer iterations. Some examples of the intermediate output for one image are shown in Figure 2.10.

To find the final segmentation, the probability map $P(\text{background}|I(x,y), x, y)$ is thresholded at 0.5. A connected components algorithm is applied, and any background colored components

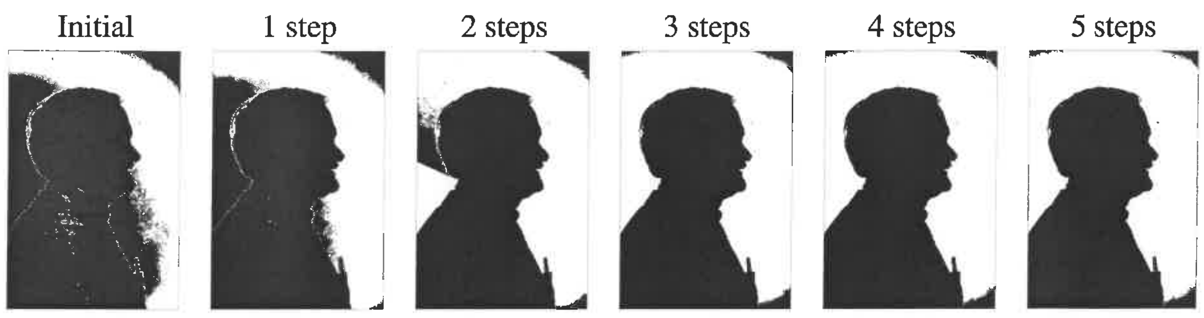


Figure 2.10: The segmentation of the background as the EM-based algorithm progresses. The images show the probability that a pixel belongs to the background, where white is zero probability, and black is one. Note that this is a particularly difficult case for the algorithm; usually the initial background is quite accurate, and the result converges immediately.

which touch the border of the image are merged into a single component. This allows for things like strands of hair which might otherwise separate the background components. Applying the resulting mask (like that in Figure 2.11a) to the original image gives a result like the one shown in Figure 2.11b.

A bright white border is visible around the face. These pixels are actually a mixture of intensities from the foreground and background, and thus their intensities are no longer close enough to the background to be classified as such. One solution to this is to apply a median filter to the inside border of the masked region, using sample intensities from a small neighborhood around the pixel. The result is shown in Figure 2.11c.

Although the E-M segmentation algorithm worked well in the example of Figure 2.11, it sometimes fails when the background is non-uniform, or when the person's skin or clothing is close in intensity to the background. In such cases, the initial model usually gives better results. Since the segmentation is used to produce training data, complete automation is not necessary. Thus rather than trying to make the algorithm work perfectly, I examined the segmentation results for each image using the initial and final background models, and selected the one with the best segmentation.

Once the background mask is computed, we can replace the background with something more realistic. I developed four random background generators:

1. Constant background, with an intensity selected uniformly from the range 0 to 255.

2. Static noise background. Each pixel will have an intensity of either 0 or 255. The probability of pixel being 255 is first picked randomly (from 0% to 100%), and then the intensity of each pixel is chosen randomly according to that probability.

3. Sinusoidal background, with a random mean (between 0 and 255), amplitude (between 0 and

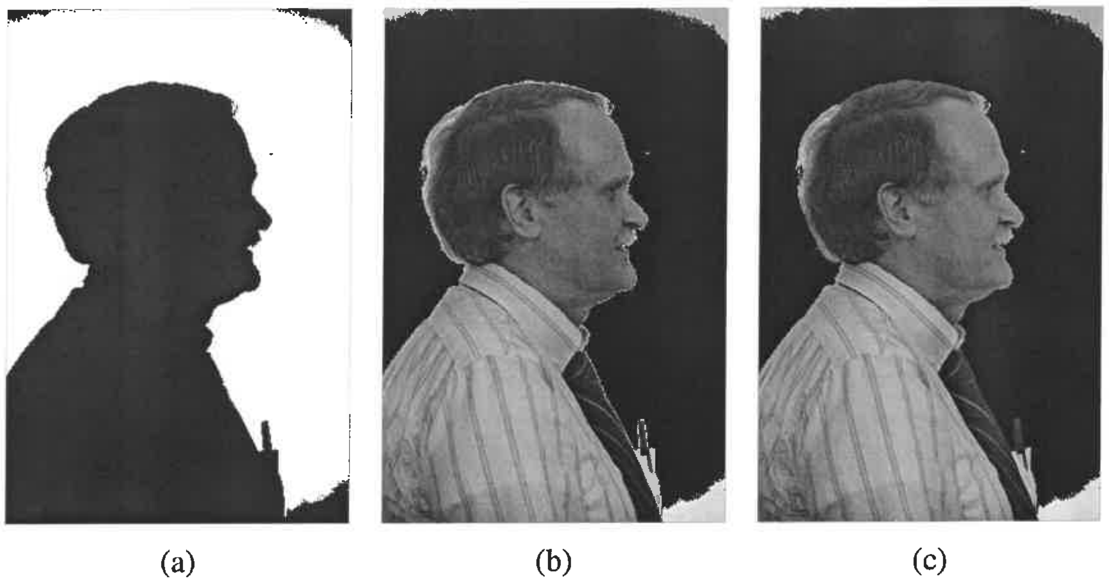


Figure 2.11: a) The background mask generated by the EM-based algorithm. b) The masked image, in which the masked area is replaced by black. Note the bright border around the face. c) Removing the bright border using a 5 × 5 median filter on pixels near the border of the mask.

128), orientation (0° to 180°), initial phase, and wavelength. The wavelength was chosen to be at least one pixel, in the scale of the face image to be generated, usually 20 × 20 or 30 × 30 pixels, and less than the window size. The intensity values were clipped to the range 0 to 255.

4. Two sinusoids added like those described above added together.

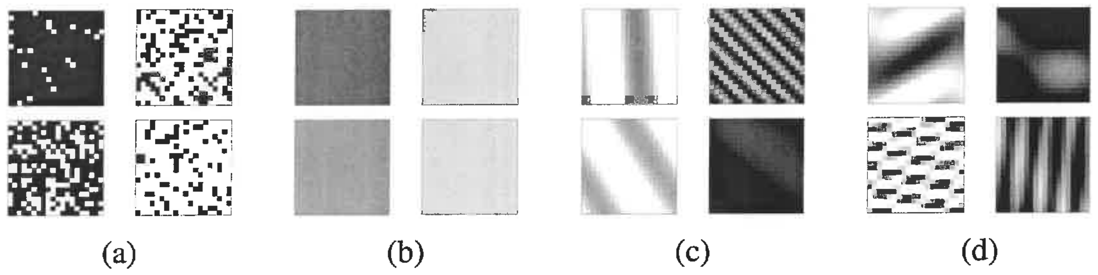


Figure 2.12: Examples of randomly generated backgrounds: (a) constant, (b) static noise, (c) single sinusoid, (d) two sinusoids.

These background generators are illustrated in Figure 2.12. For each face example to be generated, one of these four generators was chosen at random. I do not make any particular claims about the realism of the backgrounds generated by this process, but rather that they are more varied than the uniform images present in the FERET database. I considered using backgrounds extracted from real images which contain no faces, but decided against it for this work. Choosing an appropriate subset of the backgrounds is difficult, because there are so many, and randomly changing these backgrounds at runtime to give a complete sample would be computationally expensive.

2.5 Preprocessing for Brightness and Contrast

After aligning the faces and replacing the background pixels with more realistic values, 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, which was also used in [Sung, 1996]. 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 2.13a). 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, like the one used to model the background. 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 2.13. 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

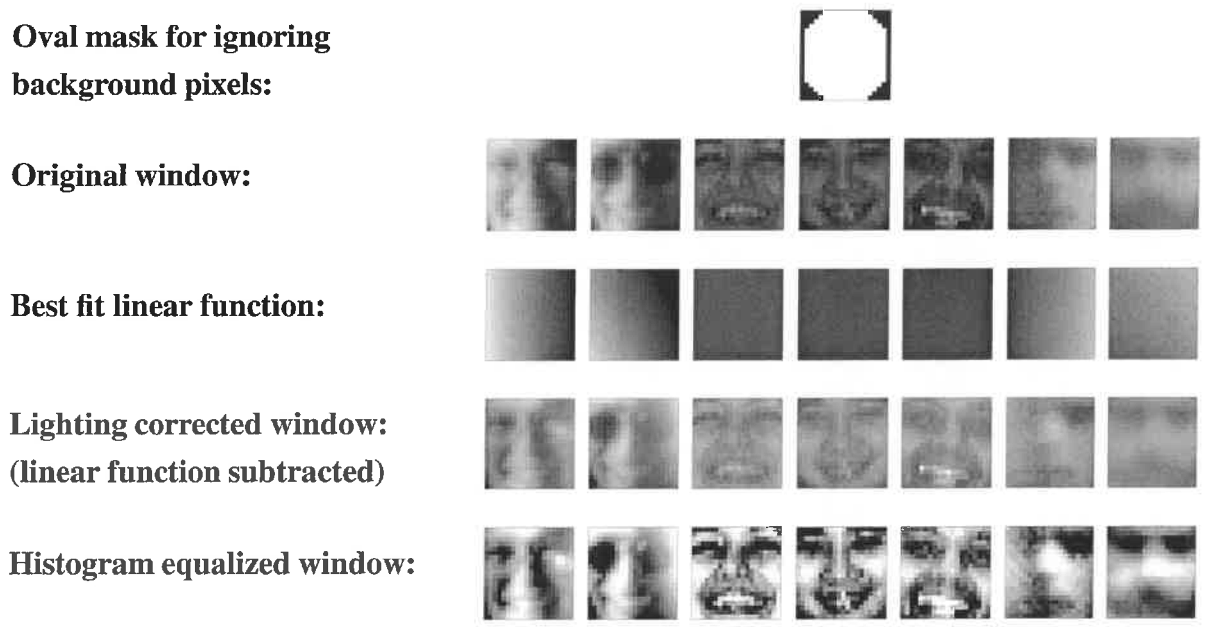

Oval mask for ignoring background pixels:
Original window:
Best fit linear function:
Lighting corrected window: (linear function subtracted)
Histogram equalized window:

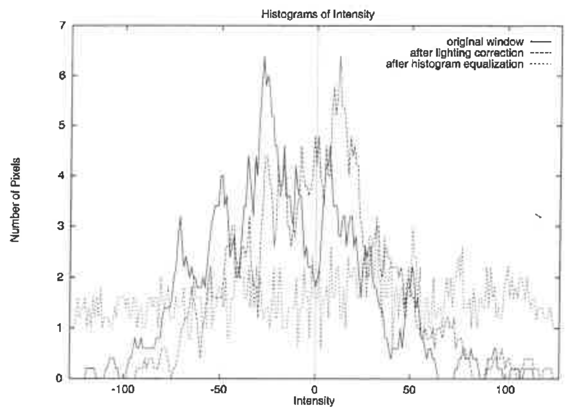
Figure 2.13: The steps in preprocessing a window. First, a linear function is fit to the intensity values in the window, and then subtracted out, correcting for some extreme lighting conditions. Then, histogram equalization is applied, to correct for different camera gains and to improve contrast. For each of these steps, the mapping is computed based on pixels inside the oval mask, and then applied to the entire window.

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.

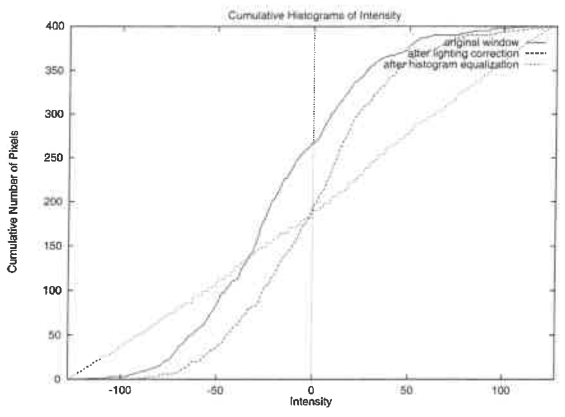
The goal is to produce a flat histogram, that is an image in which each pixel intensity occurs an equal number of times. The cumulative histogram of such an image will have that property that the number of pixels with an intensity less than or equal to a given intensity is proportional to that intensity.

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 2.14.

In some parts of this thesis, 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



(a)



(b)

Figure 2.14: (a) Smoothed histograms of pixel intensities in a 20 $\times$ 20 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.

somewhat, without the background pixels having too much effect on the appearance of the face in the foreground.

2.6 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.

2.6.1 Linear Lighting Models

The ideas in this section are based on the illumination models in the work of [Belhumeur and Kriegman, 1996], in which they explored the range of appearances an object can take on under differently lighting conditions. One assumption they use is that adding light sources to a scene results in an image which is a sum of the images for each individual light source. The authors further use the assumption that the object obeys a Lambertian lighting model for each individual light source, in which light is scattered from the object's surface equally in all directions. 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 2.15.

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 2.16.

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. The next two subsections describe some approaches for this.

2.6.2 Neural Networks for Compensation

Given a new input window to be classified as a face or nonface, 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 nonfaces and vice-versa. The architecture we tried is shown in Figure 2.17.

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

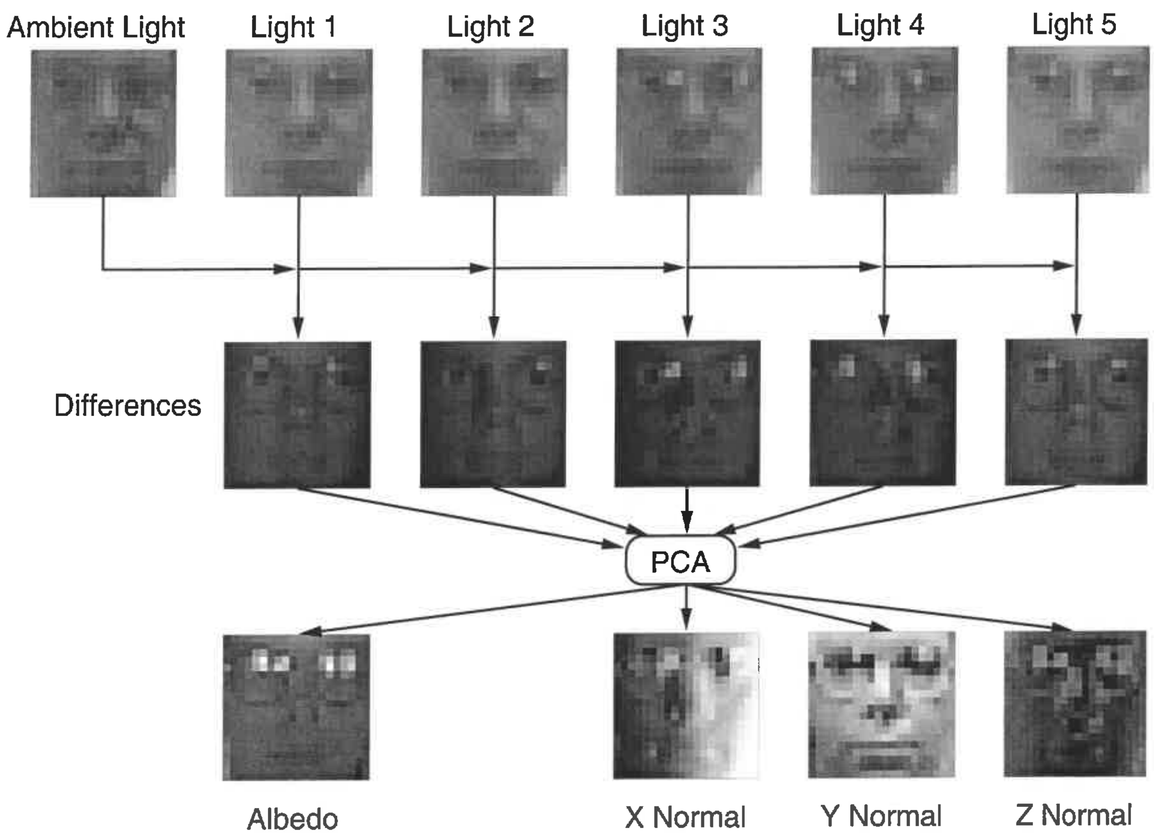


Figure 2.15: Example images under different lighting conditions, such as these, allow us to solve for the normal vectors on a face and its albedo.

that it is coming from the front of the face. Some example training data is shown in Figure 2.18. 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 nonface 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.

Some results from this system for faces and nonfaces are shown in Figure 2.19. 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 nonfaces are troubling. Many of the nonfaces 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 nonfaces as input.

One potential solution to this problem would be to measure how much work the lighting cor-

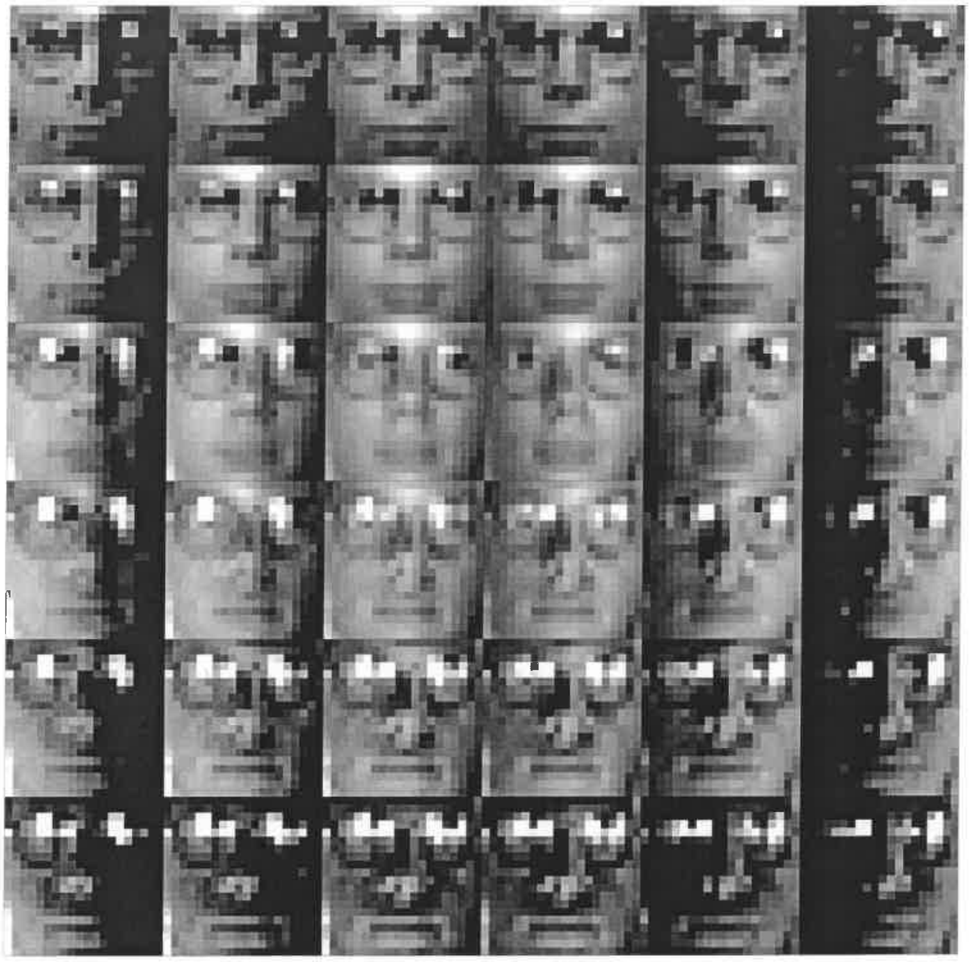


Figure 2.16: Generating example images under variable lighting conditions.

rection 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.

2.6.3 Quotient Images for Compensation

Another approach to intelligently correcting the lighting in an image is presented in [Riklin-Raviv and Shashua, 1998]. The idea in this work is again to use linear lighting models. They present a technique where an input image can be simultaneously projected into the linear lighting spaces of a set of linear models. The simulatenous 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

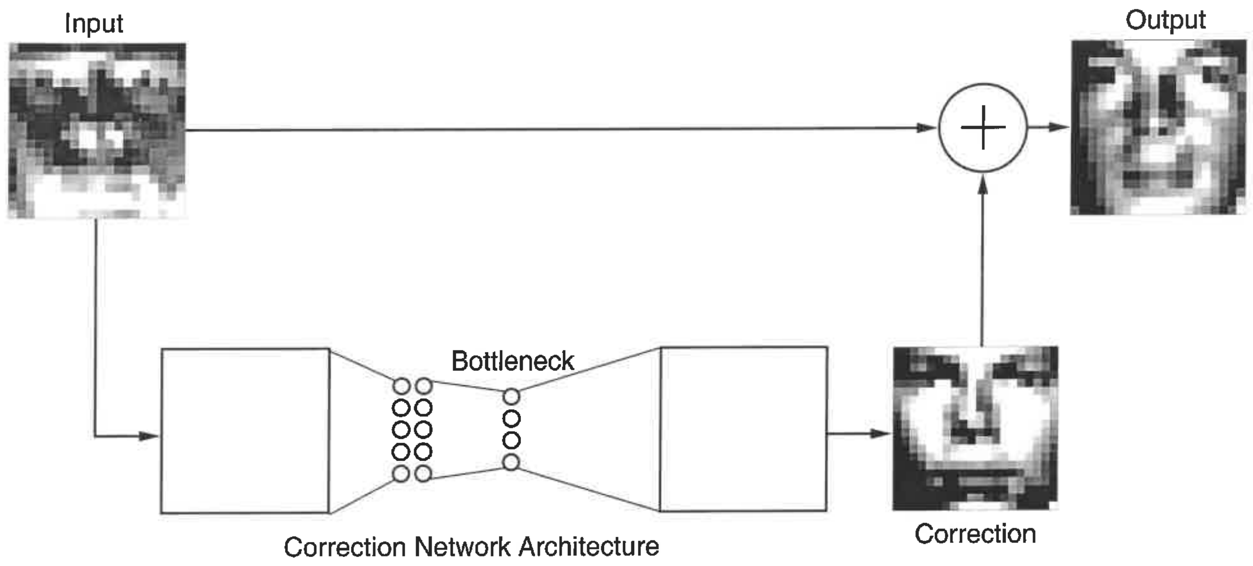


Figure 2.17: Architecture for correcting the lighting of a window. The window is given to a neural network, which has a severe bottleneck before its output. The output is a correction to be added to the original image.

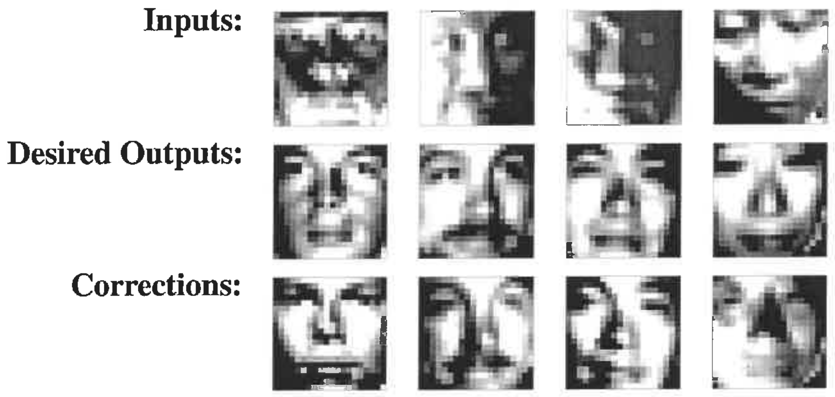


Figure 2.18: Training data for the lighting correction neural network.

result of this optimization is a vector **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:

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

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.

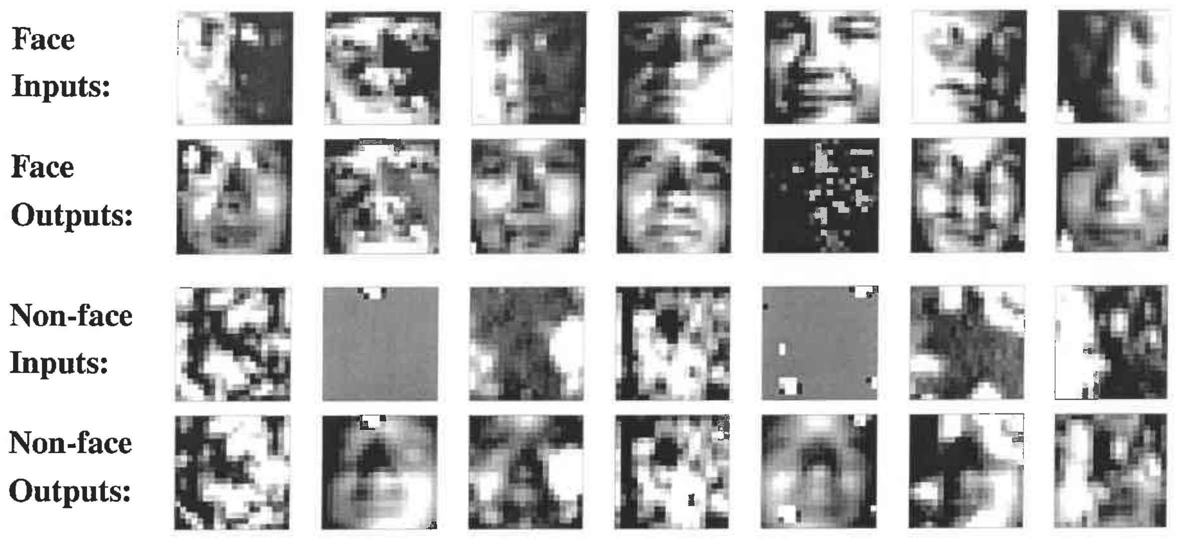


Figure 2.19: Result of lighting correction system. The lighting correction results for most of the faces are quite good, but some of the nonfaces have been changed into faces.

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 [Riklin-Raviv and Shashua, 1998]. 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:

$$I'(x,y) = I(x,y)\frac{\sum_{i=1} nA_i(x,y)(\mathbf{N}_i(x,y)\cdot\mathbf{L})}{\sum_{i=1} nA_i(x,y)(\mathbf{N}_i(x,y)\cdot\begin{pmatrix}1 & 0 & 0\end{pmatrix})}$$

This should ideally give an image of the original face but with frontal lighting. Some examples are shown in Figure 2.20. 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.

2.7 Summary

The first part of this chapter described the training and test databases used throughout this thesis. 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

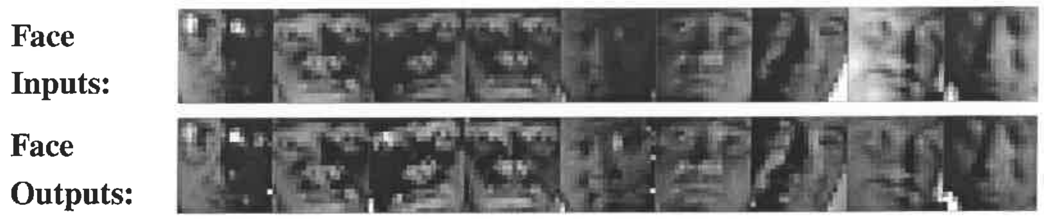


Figure 2.20: Result of using quotient images to correct lighting.

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.

The next chapter will begin the discussion of face detection itself, by examining the problem of detecting upright faces in images.

Chapter 3

Upright Face Detection

3.1 Introduction

In this chapter, I will present a neural network-based algorithm to detect upright, frontal views of faces in gray-scale images. 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 "nonface" 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 nonfaces by selectively adding images to the training set as training progresses [Sung, 1996]. 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 3.2. Section 3.3 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 3.4. The results in Section 3.5 show that the system is able to detect 90.5% of the faces over a test set of 130 complex images, with an acceptable number of false positives.

3.2 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 20×20 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 subsampling), and the detector is applied at each size. This network must have some invariance to position and scale. The amount of invariance determines the number of scales and positions at which it must be applied. For the work presented here, we apply the network at every pixel position in the image, and scale the image down by a factor of 1.2 for each step in the pyramid. This image pyramid is shown at the left of Figure 3.1.

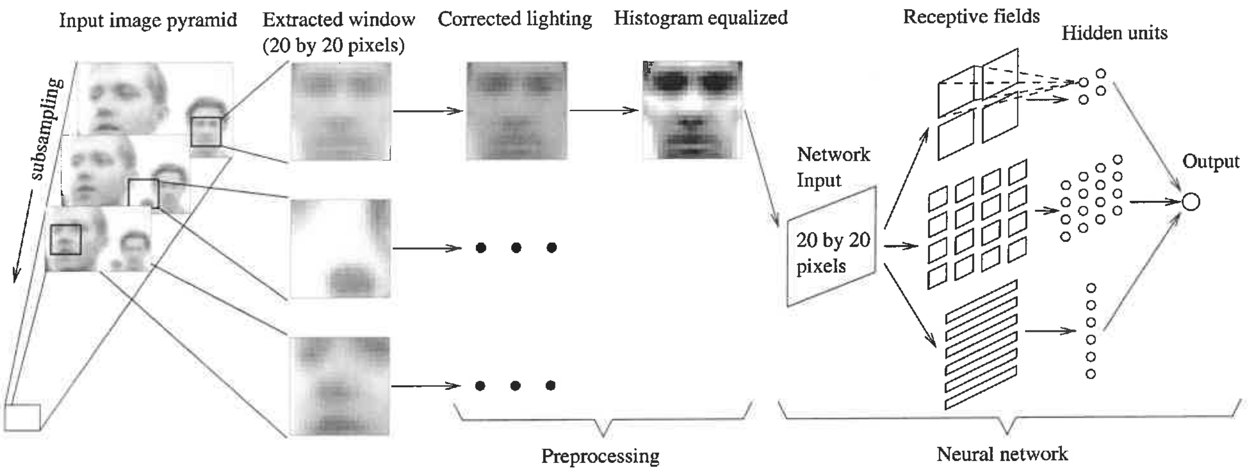


Figure 3.1: The basic algorithm used for face detection.

After a 20×20 pixel window is extracted from a particular location and scale of the input image pyramid, it is preprocessed using the affine lighting correction and histogram equalization steps described in Section 2.5. 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 3.1. The input window is broken down into smaller pieces, of four 10×10 pixel regions, sixteen 5×5 pixel regions, and six overlapping 20×5 pixel regions. Each of these regions will have complete connections to a hidden unit, as shown in the figure. 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. Other experiments have shown that the exact shapes of these regions do not matter; however it is important that the input is broken into smaller pieces instead of using complete connections to the entire input. Similar input connection patterns are commonly used in speech and character recognition tasks [Waibel *et al.*, 1989, Le Cun *et al.*, 1989]. The network has a single, real-valued output, which indicates whether or not the window contains a face.

3.2.1 Face Training Images

In order to use a neural network to classify windows as faces or nonfaces, we need training examples for each set. For positive examples, we use the techniques presented in Section 2.3 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 20×20 pixel window. The images are scaled by a random factor between $1/\sqrt{1.2}$ to $\sqrt{1.2}$, 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.

3.2.2 Non-Face Training Images

We needed a large number of nonface images to train the face detector, because the variety of nonface 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. There is a large collection of images located at `http://wuarchive.wustl.edu/multimedia/images/gif/`. We selected the images with the keyword "Scenery" in their descriptions from the index, and downloaded those images. This, along with a couple other images from other sources, formed our collection of 120 nonface "scenery" images.

Collecting a "representative" set of nonfaces is difficult. Practically any image can serve as a nonface example; the space of nonface 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.

3.2.3 Active Learning

Because of the difficult of training with every possible negative example, we utilized an algorithm described in [Sung, 1996]. 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 nonface 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 nonface 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 as the starting point.

3. 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 backpropogation algorithm with a momentum term [Hertz *et al.*, 1991]. 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 we are not training with all the negative examples, the probabilistic arguments of the previous section do not apply for setting the detection threshold.

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 backpropogation 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 nonfaces in the training sets compared with what the network will see at run time. Although theoretically the wrong thing to do, [Lawrence *et al.*, 1998] observes such techniques often work well in practice.

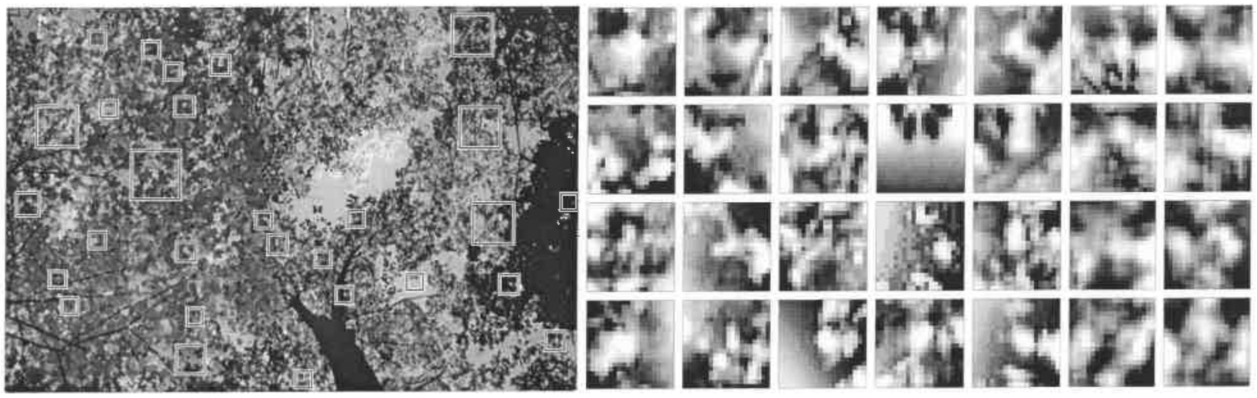


Figure 3.2: During training, the partially-trained system is applied to images of scenery which do not contain faces (like the one on the left). Any regions in the image detected as faces (which are expanded and shown on the right) are errors, which can be added into the set of negative training examples.

Some examples of nonfaces that are collected during training are shown in Figure 3.2. Note that some of the examples resemble faces, although they are not very close to the positive examples shown in Figure 2.7. The presence of these examples forces the neural network to learn the precise boundary between face and nonface images. We used 120 images of scenery for collecting negative examples in the bootstrap manner described above. A typical training run selects approximately 8000 nonface images from the 146,212,178 subimages that are available at all locations and scales in the training scenery images. A similar training algorithm was described in [Drucker *et al.*, 1993], where at each iteration an entirely new network was trained with the examples on which the previous networks had made mistakes.

3.2.4 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 trains 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 nonfaces, 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, though (see also the discussion in [Lawrence *et al.*, 1998]). If we denote $P(\text{face}|\text{window})$ as the probability that a given window is a face, and $P'(\text{face})$ and $P'(\text{nonface})$ as the prior probability of faces and nonfaces 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 $P'(\text{face})$, $P'(\text{nonface})$, and that $P(window|nonface) = 1 - P(window|face)$, this equation simplifies dramatically, giving:

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

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

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

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 3.3.2.

3.3 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*.

3.3.1 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 a sensitivity analysis using the method of [Baluja, 1996]. 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 20×20 pixel input images was divided into 100 2×2 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 3.3. Network 1 uses two sets of the hidden units illustrated in Figure 3.1, while Network 2 uses three sets.

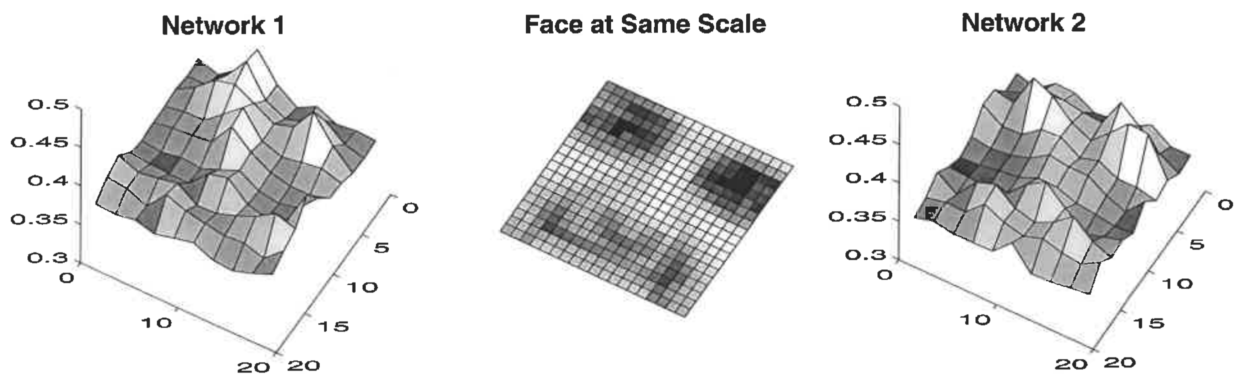


Figure 3.3: Error rates (vertical axis) on a test created by adding noise to various portions of the input image (horizontal plane), for two networks. Network 1 has two copies of the hidden units shown in Figure 3.1 (a total of 58 hidden units and 2905 connections), while Network 2 has three copies (a total of 78 hidden units and 4357 connections).

The networks rely most heavily on the eyes, then on the nose, and then on the mouth (Figure 3.3). Anecdotally, 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.

3.3.2 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.

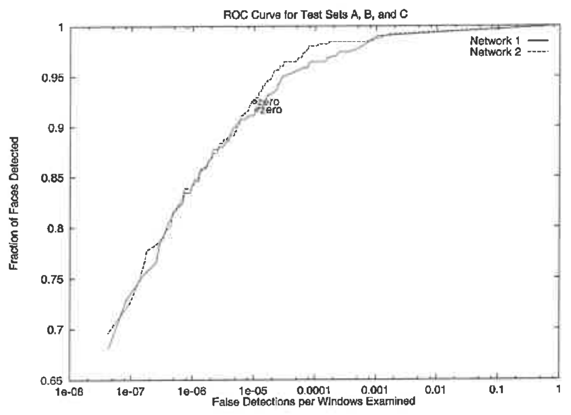


Figure 3.4: The detection rate plotted against false positive rates as the detection threshold is varied from -1 to 1, for the same networks as Figure 3.3. The performance was measured over all images from the *Upright Test Set*. The points labelled "zero" are the zero threshold points which are used for all other experiments.

This tradeoff is presented in Figure 3.4, 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 130 images with 507 faces (plus 4 upside-down faces not considered in this chapter), and requires the networks to process 83,099,211 windows. The false positive rate is in terms of the number of 20×20 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):

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

Since the zero threshold locations are close to the "knees" of the curves, as can be seen from the figure, we used a zero threshold value throughout testing.

To give an intuitive idea about the meaning of the numbers in Figure 3.4 (with a zero threshold), some examples of the output on the two images in Figure 3.5 are shown in Figure 3.6. 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 3.2.4 to determine the threshold for detecting faces in that scheme. Suppose that for a true face, windows one pixel either

Figure 3.5: Example images on to test the output of the upright detector.

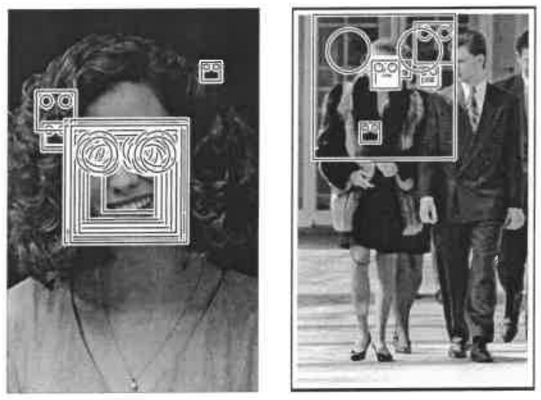


Figure 3.6: Images from Figure 3.5 with all the above threshold detections indicated by boxes. Note that the circles are drawn for illustration only, they do not represent detected eye locations.

side of its location, and windows either side of its scale can be detected, then each face contributes about $3 \cdot 3 \cdot 3 = 27$ face windows. In the training database, there are 1046 faces ($27 \times 1046 = 28242$ face windows) and 592,624,845 20×20 windows, giving a probability of faces equal to $1/20984$. This is the value that will be used later in testing.

3.4 Refinements

The examples in Figure 3.6 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.

3.4.1 Clean-Up Heuristics

Note that in Figure 3.6a, the face is detected at multiple nearby positions or scales, while false detections often occur with less consistency. The same is true of Figure 3.6b, but since the faces are smaller the overlapping detections are not visible. These observation 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 3.6 is shown in Figure 3.7.

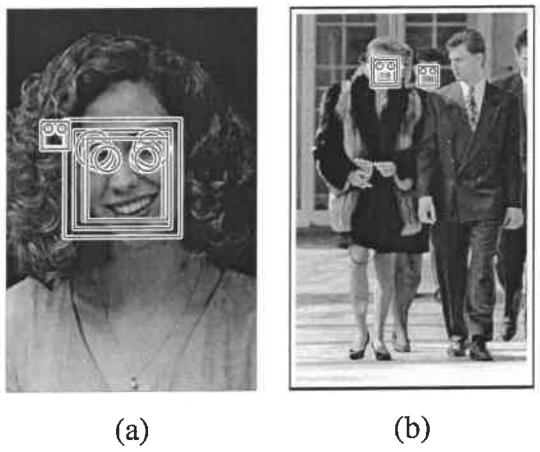


Figure 3.7: Result of applying *threshold(4,2)* to the images in Figure 3.6.

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 3.8b, 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.*, 1998].

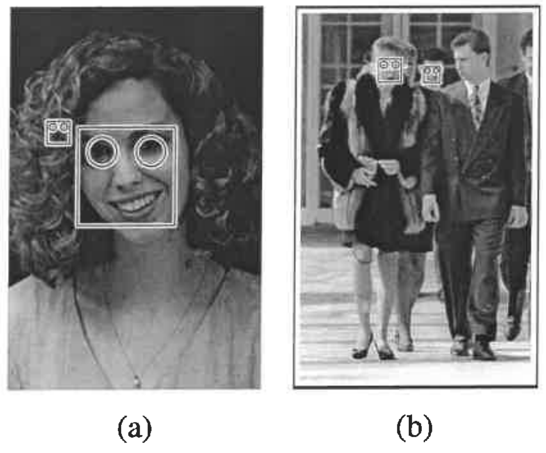


Figure 3.8: Result of applying *overlap* to the images in Figure 3.7.

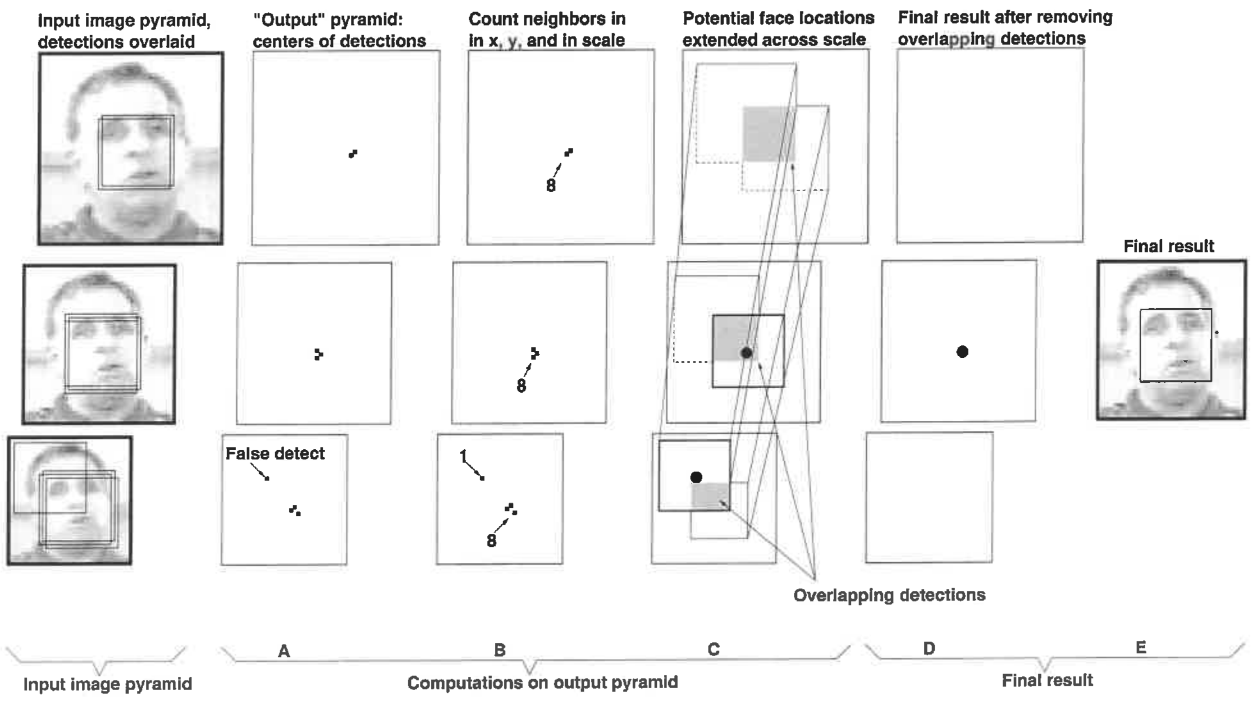

Input image pyramid, detections overlaid
"Output" pyramid: centers of detections
Count neighbors in in x, y, and in scale
Potential face locations extended across scale
Final result after removing overlapping detections
Final result
False detect
8
1
Overlapping detections
A
B
C
D
E
Input image pyramid
Computations on output pyramid
Final result

Figure 3.9: 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 3.9. 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 3.9. 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.

3.4.2 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. [Baker and Nayar, 1996] used the converse idea, that of pattern rejectors, for recognition. Each classifier eliminates a set of potential classifications of an example, until only the example's class is left.

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 nonface 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 3.10. 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.

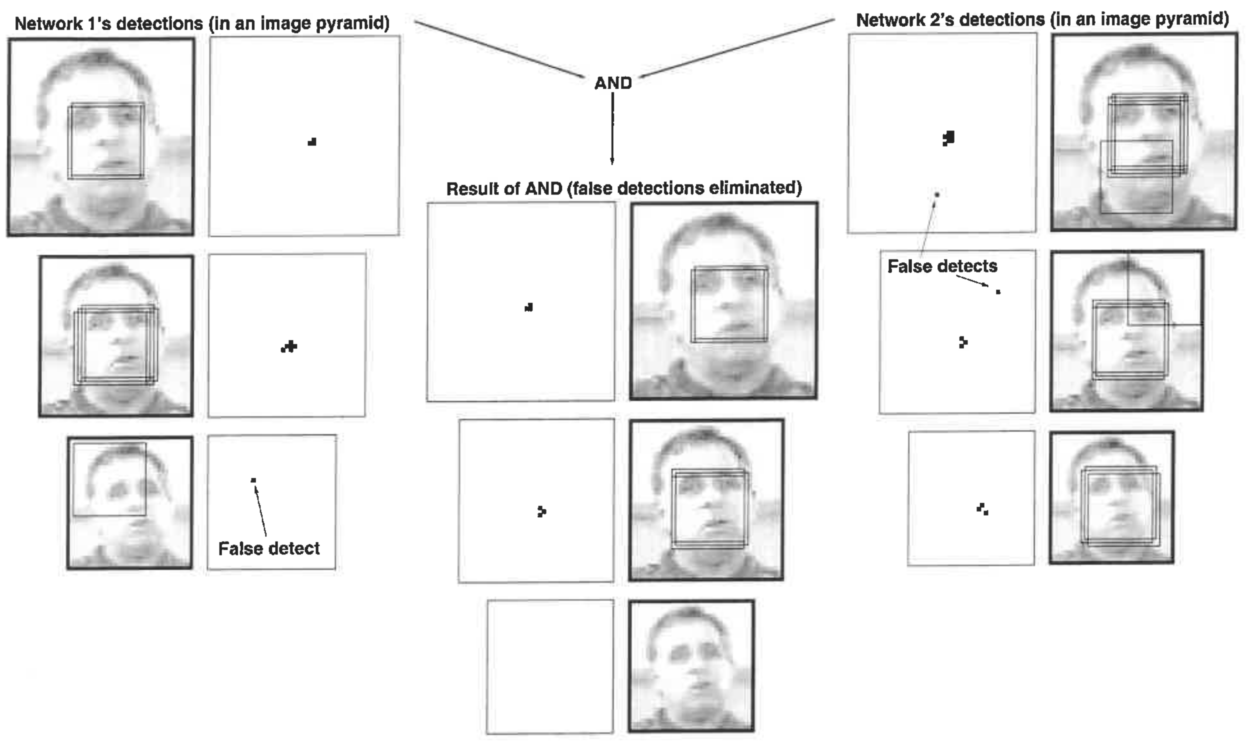

Network 1's detections (in an image pyramid)
AND
Network 2's detections (in an image pyramid)
Result of AND (false detections eliminated)
False detects
False detect

Figure 3.10: 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 3.11. 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 3×3 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.

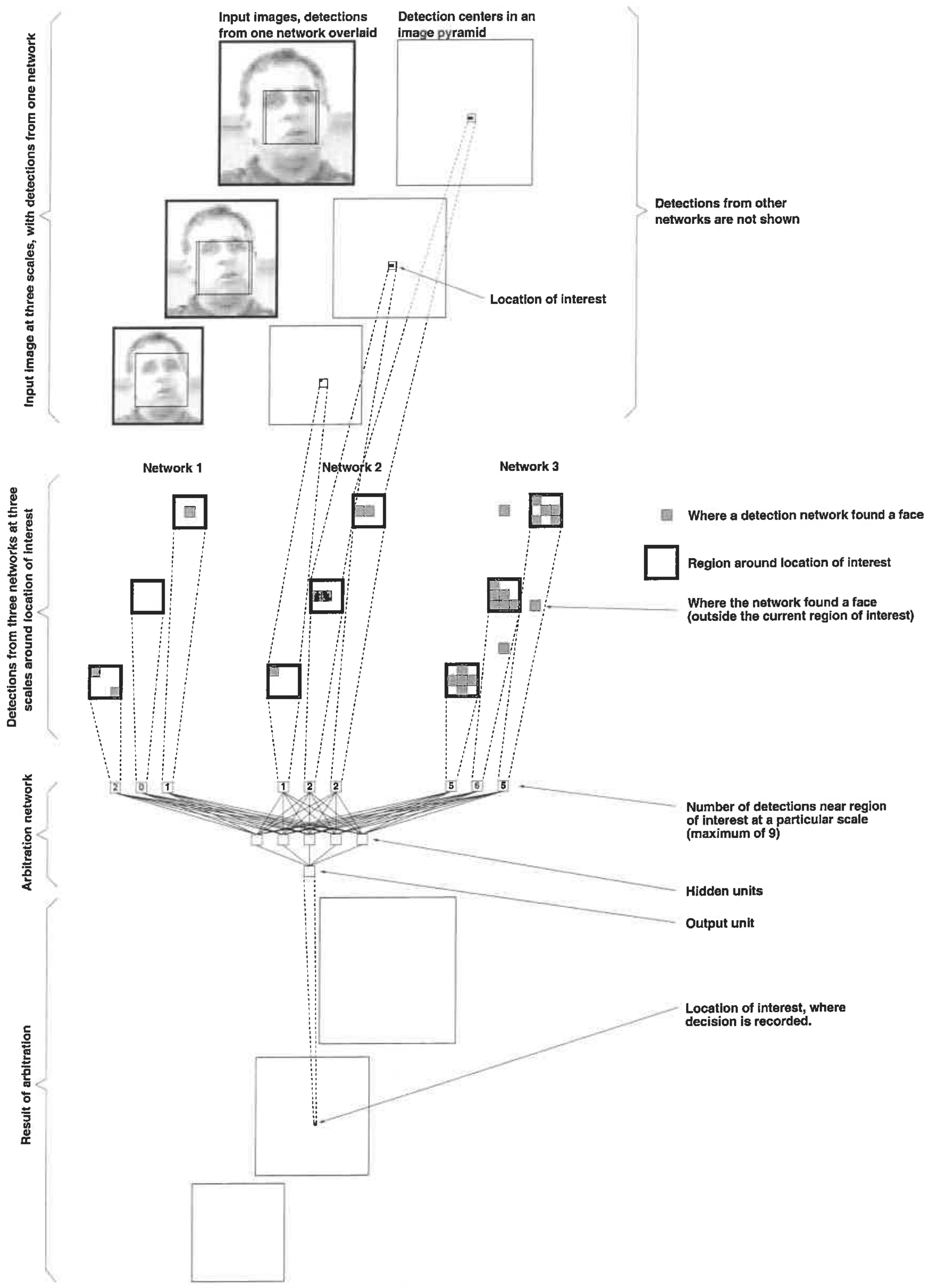

Input images, detections from one network overlaid
Detection centers in an image pyramid
Detections from other networks are not shown
Location of interest
Input image at three scales, with detections from one network
Network 1
Network 2
Network 3
Where a detection network found a face
Region around location of interest
Where the network found a face (outside the current region of interest)
Detections from three networks at three scales around location of interest
2
0
1
1
2
2
5
6
5
Number of detections near region of interest at a particular scale (maximum of 9)
Arbitration network
Hidden units
Output unit
Location of interest, where decision is recorded.
Result of arbitration

Figure 3.11: The inputs and architecture of the arbitration network which arbitrates among multiple face detection networks.

3.5 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.

3.5.1 Upright Test Set

The first set of test images is for testing the capabilities of the upright face detector.

One of the first face detection systems with high accuracy in cluttered images was developed at the MIT Media Lab by Kah-Kay Sung and Tomaso Poggio [Sung, 1996]. To evaluate the accuracy of their system, they collected a test database of 23 images from various sources, which we also use for testing purposes.

In addition to these images, we collected 107 images containing upright faces locally. These images were scanned from newspapers, magazines, and photographs, found on the WWW, or captured with CCD cameras attached to digitizers, or digitized from broadcast television. The latter images were provided by Michael Smith from the Informedia project at CMU.

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 3.12. In the following, this test set will be called the *Upright Test Set*.

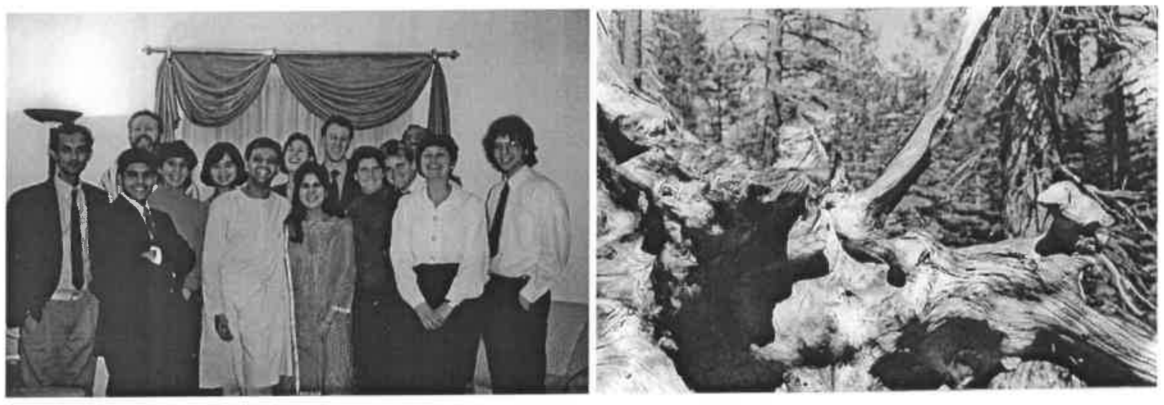


Figure 3.12: Example images from the *Upright Test Set*, used for testing the upright face detector.

Table 3.13 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 507), the detection rate, the total number of false detections, and the false detection rate (compared with the number of 20 × 20 windows examined.

Table 3.13: Detection and error rates for the *Upright Test Set*, which consists of 130 images and contains 507 frontal faces. It requires the system to examine a total of 83,099,211 20×20 pixel windows.

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 → threshold(4,1) → overlap	50	90.1%	516	1/161044
	6) Network 2 → threshold(4,1) → overlap	44	91.3%	453	1/183441
	7) Network 3 → threshold(4,1) → overlap	51	89.9%	422	1/196917
	8) Network 4 → threshold(4,1) → overlap	42	91.7%	452	1/183847
Arbitrating among two networks	9) Networks 1 and 2 → AND(0)	66	87.0%	156	1/532687
	10) Networks 1 and 2 → AND(0) → threshold(4,3) → overlap	92	81.9%	8	1/10387401
	11) Networks 1 and 2 → threshold(4,2) → overlap → AND(4)	71	86.0%	31	1/2680619
	12) Networks 1 and 2 → threshold(4,2) → overlap → OR(4) → threshold(4,1) → overlap	50	90.1%	167	1/497600
Arbitrating among three networks	13) Networks 1, 2, 3 → voting(0) → overlap	55	89.2%	95	1/874728
	14) Networks 1, 2, 3 → network arbitration (5 hidden units) → threshold(4,1) → overlap	85	83.2%	10	1/8309921
	15) Networks 1, 2, 3 → network arbitration (10 hidden units) → threshold(4,1) → overlap	86	83.0%	10	1/8309921
	16) Networks 1, 2, 3 → network arbitration (perceptron) → threshold(4,1) → overlap	89	82.4%	9	1/9233245

The table begins by showing the results for four individual networks. Networks 1 and 2 are the same as those used in Sections 3.3.1 and 3.3.2. Networks 3 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 3. 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 3.11. 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 8 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 $497,600$ windows to down to 1 in $10,387,401$, 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 81.9%. 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 13, which uses voting among three networks, has an accuracy between that of Systems 11 and 12.

3.5.2 FERET Test Set

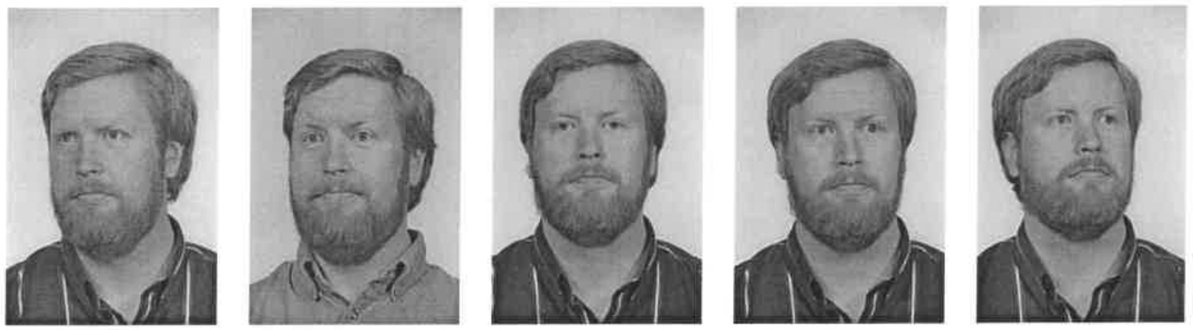


Figure 3.14: Examples of nearly frontal FERET images: (a) frontal (group labels `fa` and `fb`, (b) 15° from frontal (group labels `rb` and `rc`), and (c) 22.5° from frontal (group labels `ql` and `qr`).

The second test set we used was the portion of the FERET database [Phillips *et al.*, 1996, Phillips *et al.*, 1997, Phillips *et al.*, 1998] containing roughly frontal faces. The FERET project was run by the Army Research Lab to perform an uniform comparison of several face recognition algorithms. As part of this work, the researchers collected a large database of face images. For each person, they collected several images in different sessions, from with different angles of the face relative to the camera. The images were taken as photographs, using studio lighting conditions, and digitized later. The backgrounds were typically uniform or uncluttered, as can be seen in Figure 3.14. There are a wide variety of faces in the database, which are taken at a variety of angles. Thus these images are more useful for checking the angular sensitivity of the detector, and less useful for measuring the false alarm rate.

We partitioned the images into three groups, based on the nominal angle of the face with respect to the camera: frontal faces, faces at an angle 15° from the camera, and faces at an angle of 22.5°. The direction of the face varies significantly within these groups. As can be seen from Table 3.15, the detection rate for systems arbitrating two networks ranges between 98.1% and 100.0% for frontal and 15° faces, while for 22.5° faces, the detection rate is between 93.1% and 97.1%. This difference is because the training set contains mostly frontal faces. It is interesting to note that the systems generally have a higher detection rate for faces at an angle of 15° than for frontal faces. The majority of people whose frontal faces are missed are wearing glasses which are reflecting light into the camera. The detector is not trained on such images, and expects the eyes to be darker than the rest of the face. Thus the detection rate for such faces is lower.

Table 3.15: Detection and error rates for the *FERET Test Set*.

Type	System	Frontal Faces		15° Angle		22.5° Angle	
	Number of Images		1001		241		378
	Number of Faces		1001		241		378
	Number of Windows		255129875		61424875		96342750
		# miss	/ Detect rate	# miss	/Detect rate	# miss	/Detect rate
Type	System	False detects	/ Rate	False detects	/Rate	False detects	/Rate
One	1) Net 1 (2 copies of hidden units,	5	99.5%	1	99.6%	8	97.9%
network,	2905 connections)	1743	1/146373	446	1/137723	812	1/118648
no	2) Net 2 (3 copies of hidden units,	5	99.5%	0	100.0%	11	97.1%
heuristics	4357 connections)	1466	1/174031	489	1/125613	614	1/156910
	3) Net 3 (2 copies of hidden units,	4	99.6%	1	99.6%	8	97.9%
	2905 connections)	1209	1/211025	365	1/168287	604	1/159507
	4) Net 4 (3 copies of hidden units,	6	99.4%	0	100.0%	15	96.0%
	4357 connections)	1618	1/157682	471	1/130413	733	1/131436
One	5) Network 1 → threshold(4,1) →	5	99.5%	1	99.6%	11	97.1%
network,	overlap	572	1/446031	127	1/483660	247	1/390051
with	6) Network 2 → threshold(4,1) →	5	99.5%	0	100.0%	12	96.8%
heuristics	overlap	433	1/589214	117	1/524998	131	1/735440
	7) Network 3 → threshold(4,1) →	5	99.5%	1	99.6%	10	97.4%
	overlap	379	1/673165	75	1/818998	135	1/713650
	8) Network 4 → threshold(4,1) →	7	99.3%	0	100.0%	16	95.8%
	overlap	514	1/496361	107	1/574064	193	1/499185
Arbitrating	9) Nets 1 and 2 → AND(0)	13	98.7%	1	99.6%	20	94.7%
among two		290	1/879758	102	1/602204	162	1/594708
networks	10) Nets 1 and 2 → AND(0) →	19	98.1%	1	99.6%	26	93.1%
	threshold(4,3) → overlap	2	1/127564937	1	1/61424875	2	1/48171375
	11) Nets 1 and 2 → threshold(4,2) →	8	99.2%	1	99.6%	20	94.7%
	overlap → AND(2)	9	1/28347763	2	1/30712437	3	1/32114250
	12) Nets1,2→threshold(4,2)→overlap	3	99.7%	0	100.0%	11	97.1%
	→OR(4)→threshold(4,1)→overlap	125	1/2041039	36	1/1706246	55	1/1751686
Arbitrating	13) Nets 1,2,3 → voting(0) →	7	99.3%	2	99.2%	14	96.3%
among	overlap	46	1/5546301	10	1/6142487	20	1/4817137
three	14) Nets 1,2,3 → NN (5 hidden units)	13	98.7%	1	99.6%	20	94.7%
networks	→ threshold(4,1) → overlap	4	1/63782468	2	1/30712437	2	1/48171375
	15) Nets 1,2,3 → NN (10 hidden	16	98.4%	1	99.6%	21	94.4%
	units) → threshold(4,1) → overlap	4	1/63782468	1	1/61424875	2	1/48171375
	16) Nets 1,2,3 → NN (perceptron) →	16	98.4%	1	99.6%	23	93.9%
	threshold(4,1) → overlap	3	1/85043291	1	1/61424875	2	1/48171375

3.5.3 Example Output

Based on the results shown in Tables 3.13 and 3.15, both Systems 11 and 15 make acceptable tradeoffs between the number of false detections and the detection rate. Because System 11 is less complex than System 15 (using only two networks rather than a total of four), it is preferable. System 11 detects on average 86.0% of the faces, with an average of one false detection per $2,680,619$ 20×20 pixel windows examined in the *Upright Test Set*. Figs. 3.16, 3.17, and 3.18 show example output images from System 11 on images from the *Upright Test Set*[1].

3.5.4 Effect of Exhaustive Training

All of the experiments presented so far have used the active training algorithm of Section 3.2.3. In this section, we examine the performance of exhaustively training the algorithm on all available nonface images, as described in Section 3.2.4. As before, I trained two networks, and tested them independently and with arbitration on the *Upright Test Set*. The results are shown in Table 3.19. As can be seen, the results are not as good as those of the active learning algorithm (in Table 3.13. The false alarm rate is significantly lower, but it is unable to detect as many faces. This may be due in part to a poor estimate of the prior probability of faces in images.

An alternative to combining the two outputs using arbitration heuristics is to averages the two probability estimates. Assuming that the two algorithms which produced the estimates are independent and that the two algorithms are unbiased, the average estimator will have a lower variance; in other words it should be more accurate. The result of averaging the two networks of Table 3.19 is shown in Table 3.20. As can be seen from this table, the accuracy is comparable with the arbitration heuristics used earlier.

The results from doing exhaustive training on the scenery data look promising, but are not yet as good as the active learning method. This may be in part due to insufficient training of the networks, caused by the large memory and computational requirements of exhaustive training. Throughout the rest of this thesis, only the active learning scheme will be used.

3.5.5 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

[1]After painstakingly trying to arrange these images compactly by hand, we decided to use a more systematic approach. These images were laid out automatically by the PBIL optimization algorithm [Baluja, 1994]. The objective function tries to pack images as closely as possible, by maximizing the amount of space left over at the bottom of each page.

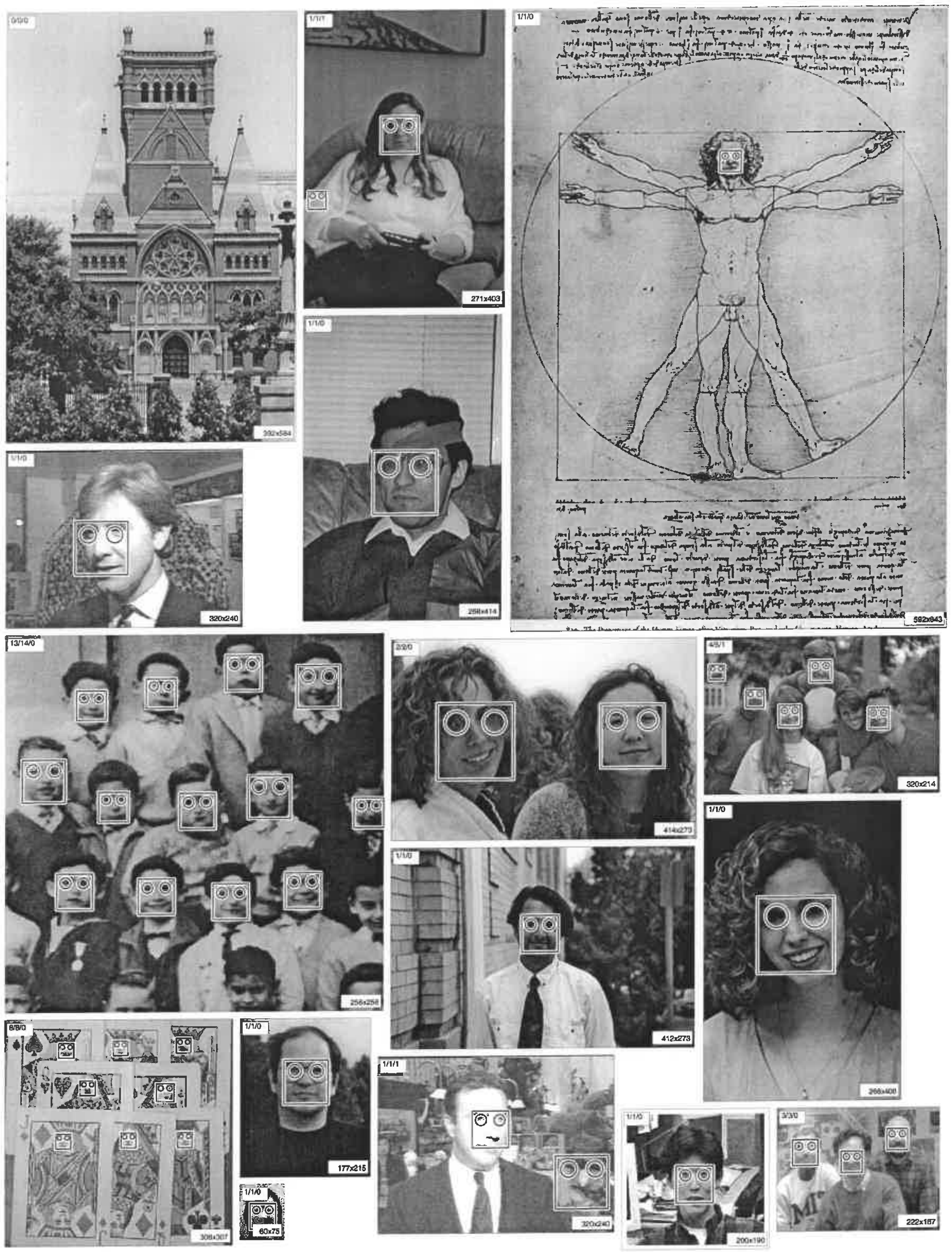


Figure 3.16: Output from System 11 in Table 3.13. The label in the upper left corner of each image (D/T/F) gives the number of faces detected (D), the total number of faces in the image (T), and the number of false detections (F). The label in the lower right corner of each image gives its size in pixels.

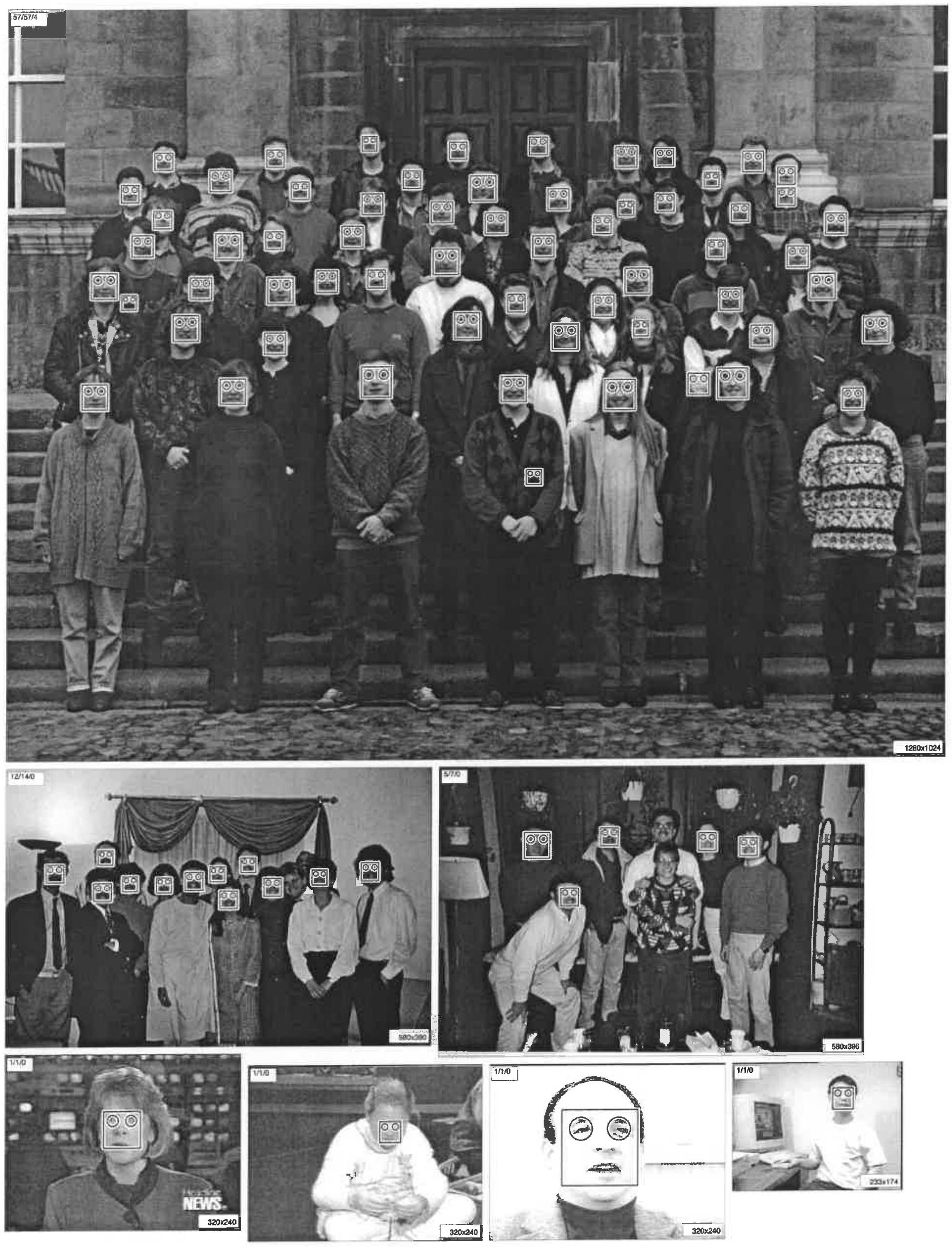


Figure 3.17: Output obtained in the same manner as the examples in Figure 3.16.

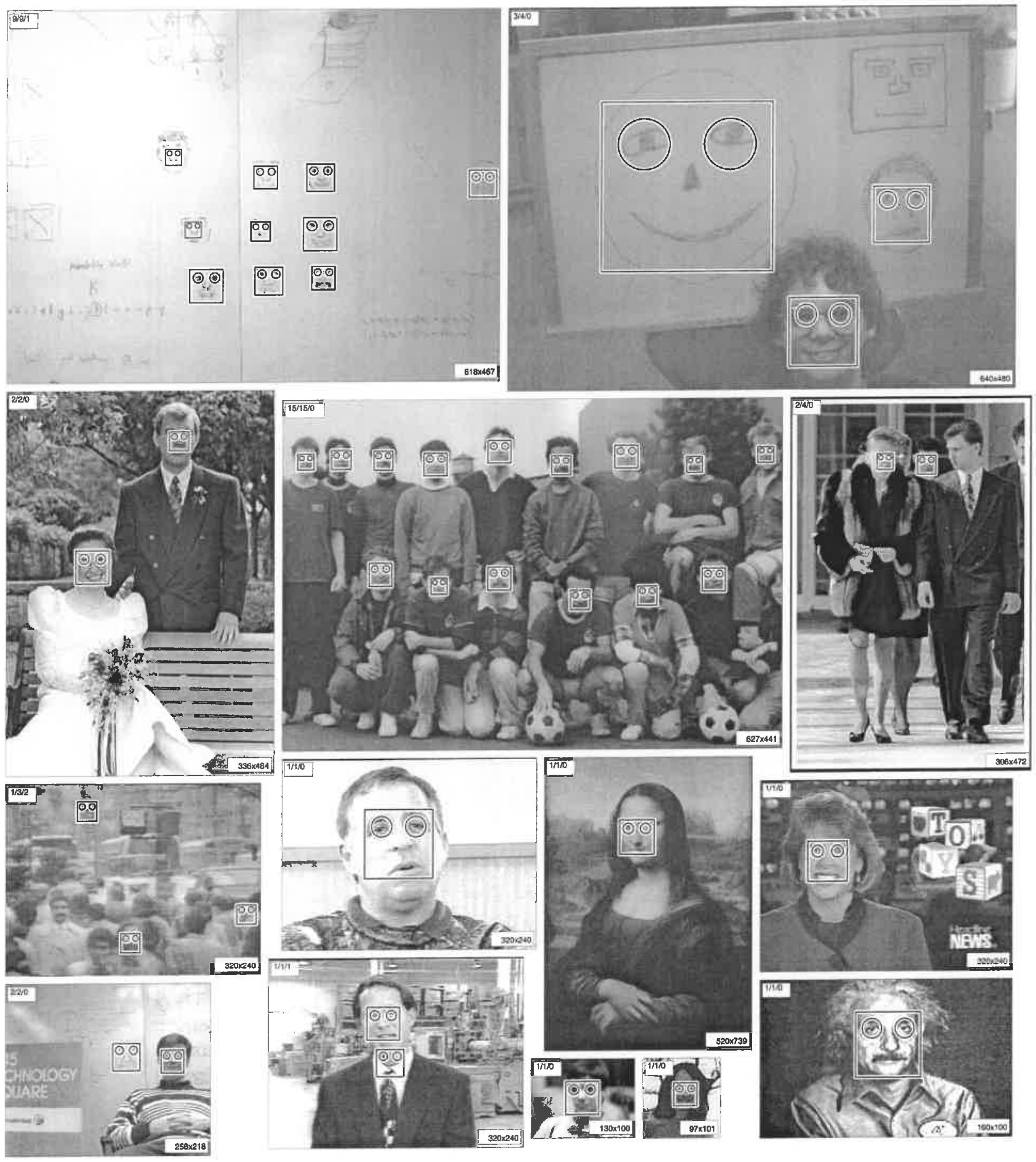


Figure 3.18: Output obtained in the same manner as the examples in Figure 3.16.

Table 3.19: Detection and error rates two networks trained exhaustively on all the scenery data, for the *Upright Test Set*.

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)	86	83.0%	703	1/118206
	2) Network 2 (3 copies of hidden units (78 total), 4357 connections)	97	80.9%	120	1/692493
One network, with heuristics	5) Network 1 → threshold(4,1) → overlap	93	81.7%	312	1/266343
	6) Network 2 → threshold(4,1) → overlap	100	80.3%	68	1/1222047
Arbitrating among two networks	9) Networks 1 and 2 → AND(0)	129	74.6%	80	1/1038740
	10) Networks 1 and 2 → AND(0) → threshold(4,3) → overlap	166	67.3%	4	1/20774802
	11) Networks 1 and 2 → threshold(4,2) → overlap → AND(4)	147	71.0%	5	1/16619842
	12) Networks 1 and 2 → threshold(4,2) → overlap → OR(4) → threshold(4,1) → overlap	99	80.5%	103	1/806788

Table 3.20: Detection and error rates resulting from averaging the outputs of two networks trained exhaustively on all the scenery data, for the *Upright Test Set*.

System	Missed faces	Detect rate	False detects	False detect rate
1) Network 1 (2 copies of hidden units (52 total), 2905 connections)	122	75.9%	52	1/1598061
5) Network 1 → threshold(4,1) → overlap	123	75.7%	25	1/3323968

to generate training data for a face detector, so that the neural network can implicitly learn to handle lighting variation.

Using lighting models of a total of 27 faces collected at CMU, I generated a training database containing 100 examples of each face, with random lighting conditions, in addition to the usual small variations in the scale, angle, and center location of the face. The result of training two networks on these images using the active learning scheme, and testing on the *Upright Test Set*, are shown in Table 3.21. Given the small number of lighting models available, we would expect that the performance would not be comparable with the networks trained on a large number of faces (as in Table 3.13). The fact that this network is able to detect approximately 50% of the faces is quite surprising; it suggests that much of the variation in the appearance of faces can be accounted for by lighting conditions. Note that the *Upright Test Set* was not selected specifically to test the tolerance of lighting variation.

Table 3.21: Detection and error rates two networks trained with images generated from lighting models, for the *Upright Test Set*.

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)	142	72.0%	2656	1/31287
	2) Network 2 (3 copies of hidden units (78 total), 4357 connections)	156	69.2%	1278	1/65022
One network, with heuristics	5) Network 1 → threshold(4,1) → overlap	156	69.2%	1521	1/54634
	6) Network 2 → threshold(4,1) → overlap	165	67.5%	845	1/98342
Arbitrating among two networks	9) Networks 1 and 2 → AND(0)	242	52.3%	116	1/716372
	10) Networks 1 and 2 → AND(0) → threshold(4,3) → overlap	296	41.6%	4	1/20774802
	11) Networks 1 and 2 → threshold(4,2) → overlap → AND(4)	251	50.5%	20	1/4154960
	12) Networks 1 and 2 → threshold(4,2) → overlap → OR(4) → threshold(4,1) → overlap	160	68.4%	374	1/222190

For completeness, I also trained two neural networks on the 27 lighting model faces, but this time only with frontal lighting for each model. Again, 100 variations of each face were generated, with slightly randomized translation, scale, and orientation. The results on the *Upright Test Set* are shown in Table 3.22. As can be seen, the accuracy is much smaller than the networks trained with lighting variation, again suggesting the importance of lighting variation in the face detection problem.

3.6 Summary

The algorithm presented in this chapter can detect between 81.9% and 90.1% of faces in a set of 130 test images with cluttered backgrounds, with an acceptable number of false detections. Depending on the application, the system can be made more or less conservative by varying the arbitration heuristics or thresholds used. The system has been tested on a wide variety of images, with many

Table 3.22: Detection and error rates two networks trained with images with frontal lighting only, for the *Upright Test Set*.

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)	408	19.5%	226	1/367695
	2) Network 2 (3 copies of hidden units (78 total), 4357 connections)	430	15.2%	161	1/516144
One network, with heuristics	5) Network 1 → threshold(4,1) → overlap	408	19.5%	195	1/426149
	6) Network 2 → threshold(4,1) → overlap	433	14.6%	134	1/620143
Arbitrating among two networks	9) Networks 1 and 2 → AND(0)	463	8.7%	10	1/8309921
	10) Networks 1 and 2 → AND(0) → threshold(4,3) → overlap	487	3.9%	1	1/83099211
	11) Networks 1 and 2 → threshold(4,2) → overlap → AND(4)	481	5.1%	2	1/41549605
	12) Networks 1 and 2 → threshold(4,2) → overlap → OR(4) → threshold(4,1) → overlap	439	13.4%	28	1/2967828

faces and unconstrained backgrounds. I have also shown the effects of using an exhaustive training algorithm (for negative examples) and the effect of using a lighting model to generate synthetic positive examples. In the next chapter, this technique is extended to faces which are tilted in the image plane. Chapter 6 will return to the algorithm of this chapter, and present techniques on how to make it run faster.

Chapter 4

Tilted Face Detection

4.1 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 4.1. 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.

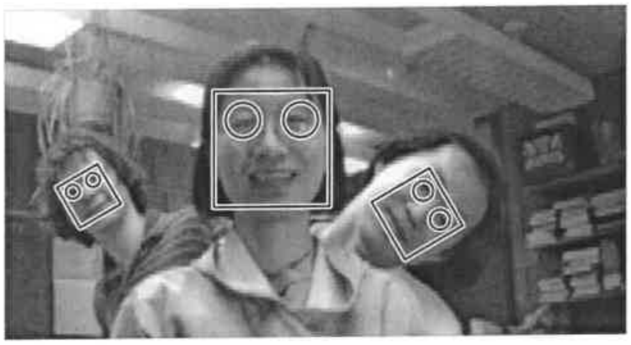


Figure 4.1: People expect face detection systems to detect rotated faces. Overlaid is the output of the system to be presented in this chapter.

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° of tilt from upright (both clockwise and counterclockwise). Therefore, the entire detection procedure would need to be applied *at least* 18 times to each image, with the image rotated in increments of 20°.

An alternate, significantly faster procedure is described in this chapter, extending some early results in [Baluja, 1997]. This 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 nonface is encountered, the derotator will return a meaningless rotation. However, since a rotation of a nonface will yield another nonface, 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 4.2. We then analyze the performance of each part of the system separately in Section 4.3, and test the complete system on three large test sets in Section 4.4. We will see that the system is able to detect 79.6% of the faces over the *Upright Test Set* and *Tilted Test Set*, with a very small number of false positives.

4.2 Algorithm

The overall structure of the algorithm, shown in Figure 4.2, 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. 20×20 pixel windows are extracted from every position and scale in this input pyramid, and passed to a classifier.

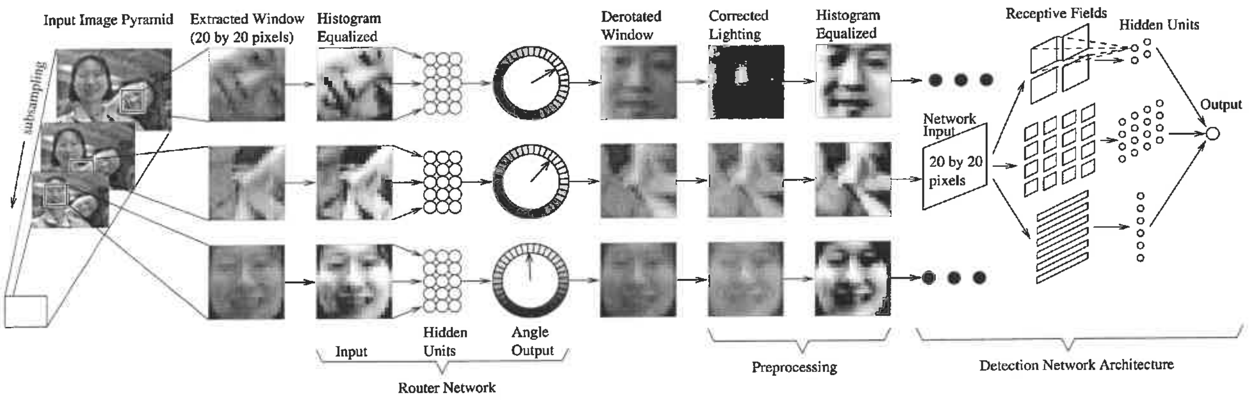


Figure 4.2: Overview of the algorithm.

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.

4.2.1 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 20×20 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 $i * 10°$. To signal that a face is at an angle of θ, each output is trained to have a value of $\cos(\theta - i * 10°)$. This approach is closely related to the Gaussian weighted outputs used in the autonomous driving domain [Pomerleau, 1992]. Examples of the training data are given in Figure 4.3.

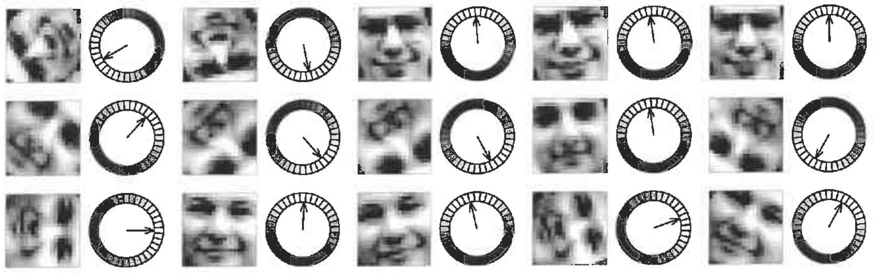


Figure 4.3: Example inputs and outputs for training the derotation network.

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:

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

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 1048 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 400 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 backpropogation algorithm.

4.2.2 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 20×20 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 nonfaces. 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 $-10°$ to $10°$.

Some examples of nonfaces 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 nonface images, rather than only nonface images in their original orientations. In Section 4.4, both possibilities are explored.

4.2.3 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 clean-up 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 4-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°. For each detection, we count the number of detections within 4 units along each dimension (4 pixels, 4 pyramid scales, or 40°). 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 4-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 4.4.

4.3 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 $-30°$ and $30°$ at $1°$ increments. These images were given to the derotation network, and the resulting histogram of angular errors is given in Figure 4.4 (left). As can be seen, 92% of the errors are within $\pm 10°$.

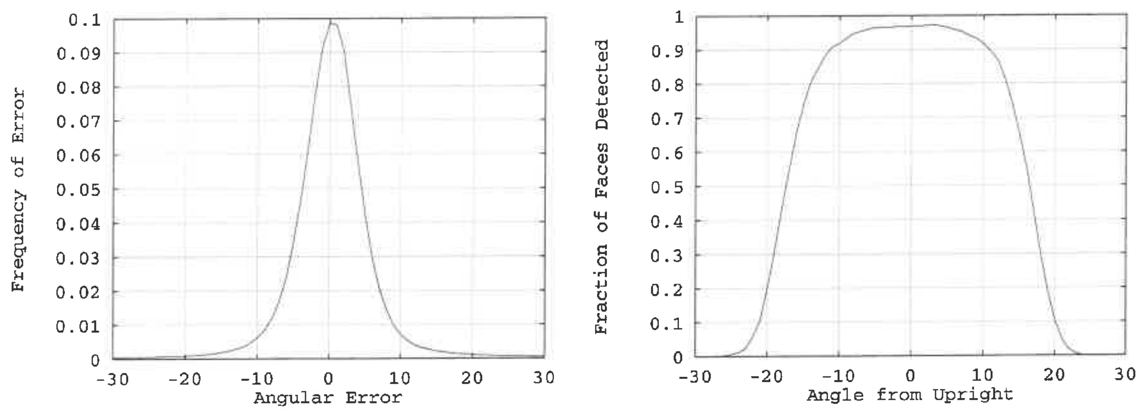


Figure 4.4: Left: Frequency of errors in the derotation network with respect to the angular error (in degrees). Right: Fraction of faces that are detected by a detection network, as a function of the angle of the face from upright.

The detector network was trained with example images having orientations between $-10°$ and $10°$. 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 4.4 (right) shows that the detector detects over 90% of the faces that are within 10° of upright, but the accuracy falls with larger angles.

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

Just as we noted in the previous section that the detector network is applied only to nonfaces 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.

4.4 Evaluation

4.4.1 Tilted Test Set

In this section, we integrate the pieces of the system, and test it on three sets of images. The first set is the *Upright Test Set* used in the previous chapter. It contains many images with faces against complex backgrounds and many images without any faces. There are a total of 130 images, with 511 frontal faces (of which 469 are within 10° of upright), and 83,099,211 windows to be processed. The second test set is the *FERET Test Set*, partitioned into three classes based on how far the face is from frontal.

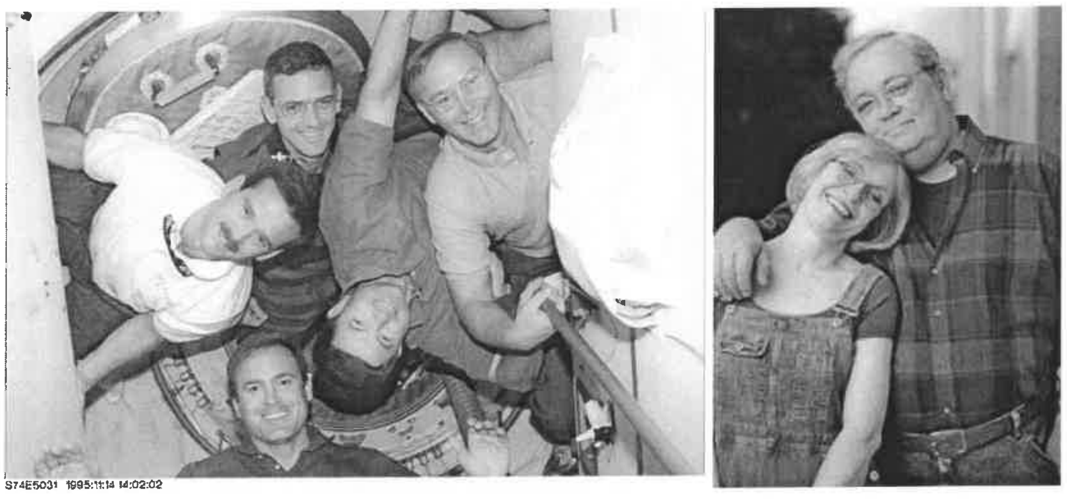


Figure 4.5: Example images in the *Tilted Test Set* for testing the tilted face detector.

To evaluate a version of the system which could detect faces that are tilted in the image, we collected a third set of images to exercise this part of the detector. These were collected from the same variety of sources as the *Upright Test Set*. A few examples are shown in Figure 4.5. The test set contains 50 images, and requires the networks to examine 34,064,635 20×20 pixel windows. Of the 223 faces in this set, 210 are at angles of more than 10° from upright. In the following sections, this test set will be called the *Tilted Test Set*.

The *Upright Test Set* and *FERET Test Set* are used as a baseline for comparison with the previous chapter. They will ensure that the modifications for rotated faces do not hamper the ability to detect upright faces. The *Tilted Test Set* will demonstrate the new capabilities of our system. Figure 4.6 shows the distributions of the angles of faces in each test set; as can be seen, most of the faces in the first two sets are very close to upright, while the last has more tilted faces. The peak for the tilted test set at 30° is due to a large image with 135 upright faces that was rotated to an angle of 30°, as can be seen in Figure 4.9.

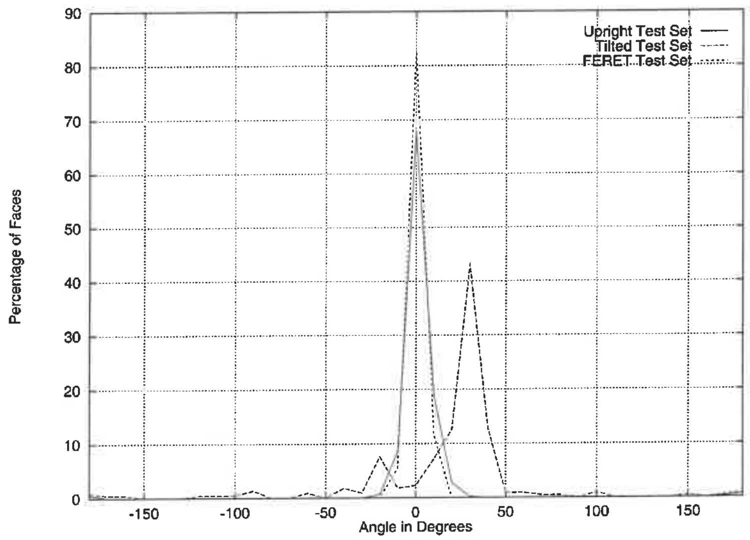


Figure 4.6: Histograms of the angles of the faces in the three test sets used to evaluate the tilted face detector. The peak for the tilted test set at 30° is due to a large image with 135 upright faces that was rotated to an angle of 30°, as can be seen in Figure 4.9.

Knowledge of the distribution of faces in particular applications may allow the detector to be simplified. In particular, faces rotated more than 45° may be quite rare in images on the WWW, so the derotation can be customized to a smaller range of angles, and possibly be more accurate. On the other hand, a digital photograph manager might use face angles to determine whether a photograph was taken with the camera in a horizontal or vertical orientation. For this application, the detector must locate faces at any angle.

4.4.2 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 4.7 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 4.7: Results of first applying the derotation network, then applying the standard upright detector networks.

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

4.4.3 Proposed System

Table 4.7 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 4.8. As can be seen, the use of these new networks reduces the number of false detections by at least a factor of 4. The detect rate has also dropped, because now the detector networks must deal with nonfaces 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 4.9.

Table 4.8: Results of the proposed tilted face detection system, which first applies the derotator network, then applies detector networks trained with derotated negative examples.

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

This system was also applied to the *FERET Test Set*, used to evaluate the upright face detector in the previous chapter. The results are shown in Table 4.10. The general pattern of the results is similar to that of the upright detector in Table 3.15, although the detection rates are slightly lower.

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 4.11. 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 nonfaces.

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

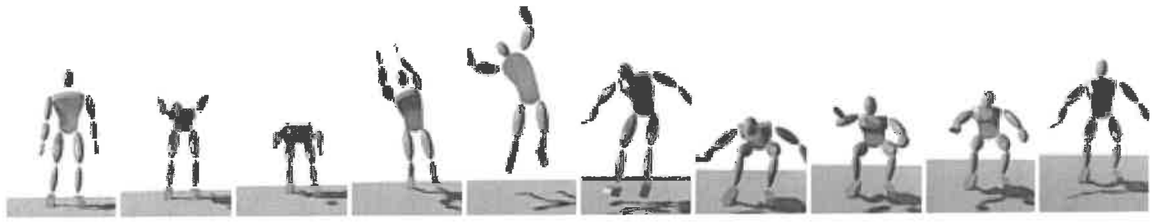


FIGURE 7.12. Frames from the diagonal jump animation.

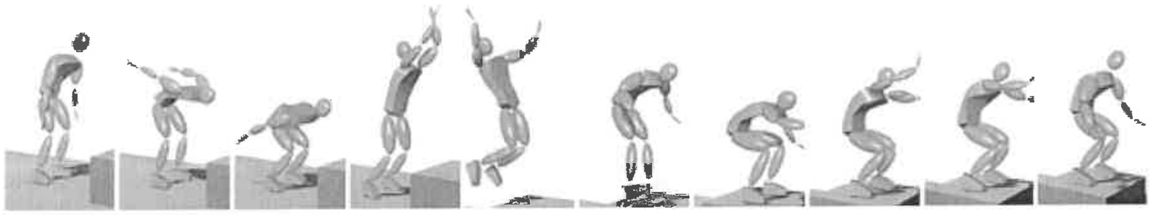


FIGURE 7.13. Frames from the obstacle jump animation.

7.5.5 Obstacle Jump

We raised the mechanical constraint controlling the landing position with intention of generating a jump onto a higher plateau. In addition we introduced the inequality pose constraint which forced the legs to raise higher during flight in order to clear the obstacle. As a result, the character push-off is more vertical, and the legs tuck in during the flight in order to avoid the hurdle. The landing phase also has a different feel which results from a more vertical jump.

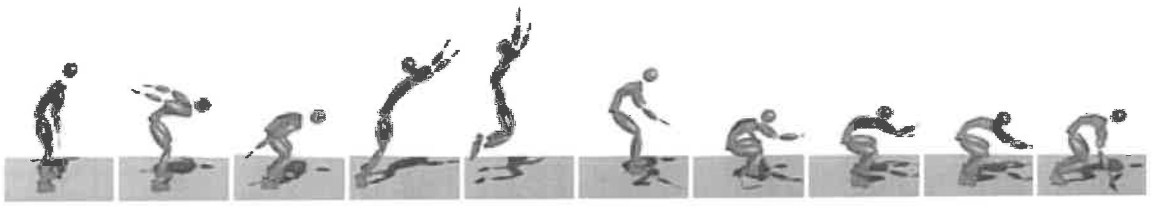


FIGURE 7.14. Frames from the unbalanced jump animation.

7.5.6 Unbalanced Jump

We removed the final pose constraints that impose the upright position. In the resulting sequence, the character never uses its muscles to stand up upon landing since this would add unnecessary non-smoothness of the muscles. Instead of straightening up, the character tumbles forward giving the appearance of poor landing balance.

7.6 Example Summary

Our experiments show that despite the extreme simplification and turning angular joints into linear joints, the hopper spacetime model still encapsulates the realistic properties of the broad jump with surprising accuracy.

We have also demonstrated a wide spectrum of animations produced from a single motion capture sequence. The power of our transformation algorithm stems directly from the underlying spacetime constraints formulation. We also draw attention to the minimal set of intuitive modifications that were performed in order to produce each motion sequence. This relative ease of creating transformed character animations suggests that our methodology can be used for the creation of reusable motion libraries. Such libraries would enable ordinary non-skilled users to create expressive animations.

7.7 Validation of Realism for the Transformed Motions

Under visual inspection the generated transformed motion sequences appear realistic. Unfortunately, there are very few deterministic claims that we can make about the physical correctness of the resulting motion. In fact, since the forces and accelerations are never computed for the full human character, we cannot provide any guarantee the generated animations are realistic.

We can, however, compare the transformed motion sequence with the real-life captured motion sequence that performs the same task. Using this comparison we can measure the relative differences between the synthesized and real-life motion sequences.

Such a comparison sequence should, of course, be realistically possible. Most of the generated animations described in this chapter are not easily replicated in reality. For example, it would be really hard to capture the running sequence on the moon surface, the running with a shortened leg, or a jump where the character exceeds the ability of its real-life counterpart. Although such impossible or hard to replicate motion is one of the main appeals of our method, it is not good for our realism validation purposes.

We used two broad jump motion capture sequences for our comparisons:

- maximum length broad jump
- 45 degree diagonal broad jump of smaller distance

The maximum length broad jump sequence was the original motion for all of the transformed broad jump sequences. The start and the end pose are identical for the two captured motion sequences, with the exception that the character is displaced to the new position in the diagonal jump motion sequence.

When creating the transformed motion sequence, we started with the maximum length broad jump as the original jump sequence. We specified the start and the end character pose constraints, and introduced mechanical constraints for the start and end floor-contact periods. We proceeded to fit the original motion to the spacetime simplified model described in Section 7.5.1. During the spacetime edit stage of our algorithm, we applied the appropriate change to the final pose constraint, and then resolved the spacetime constraints problem. After the reconstruction stage we are left with the transformed motion sequence that accurately matches the diagonal broad jump motion. To demonstrate the accuracy of the match between the generated and real motion, we compare the DOF curves for the three representative joint angles:

lknee Euler angle representing the left knee joint (Figure 7.15).

lshoulderz The z component of the quaternion representing the left shoulder joint (Figure 7.16).

lhipz The z component of the quaternion representing the left hip joint (Figure 7.17).

Each figure contains three curves which depict the DOF function in the original, real-life diagonal, and the transformed diagonal motion sequence.

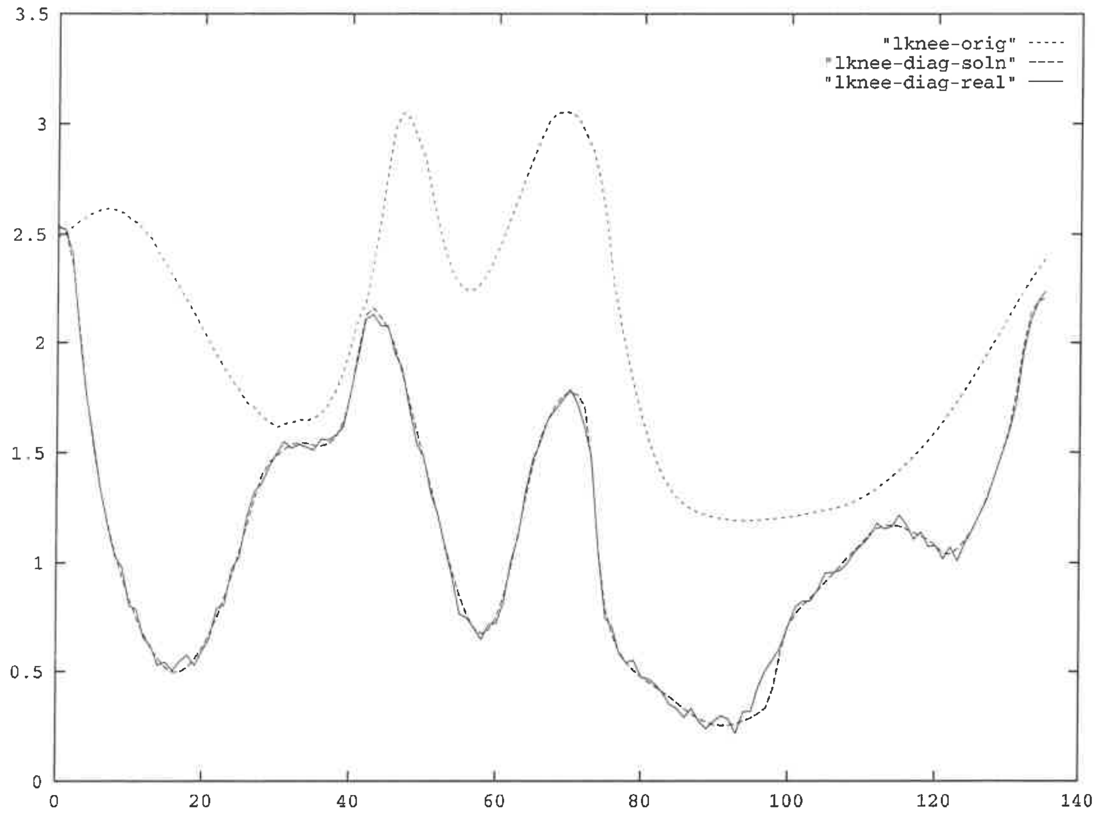


FIGURE 7.15. Left knee DOF functions for the original jump, computed diagonal jump, and the real-life diagonal jump.

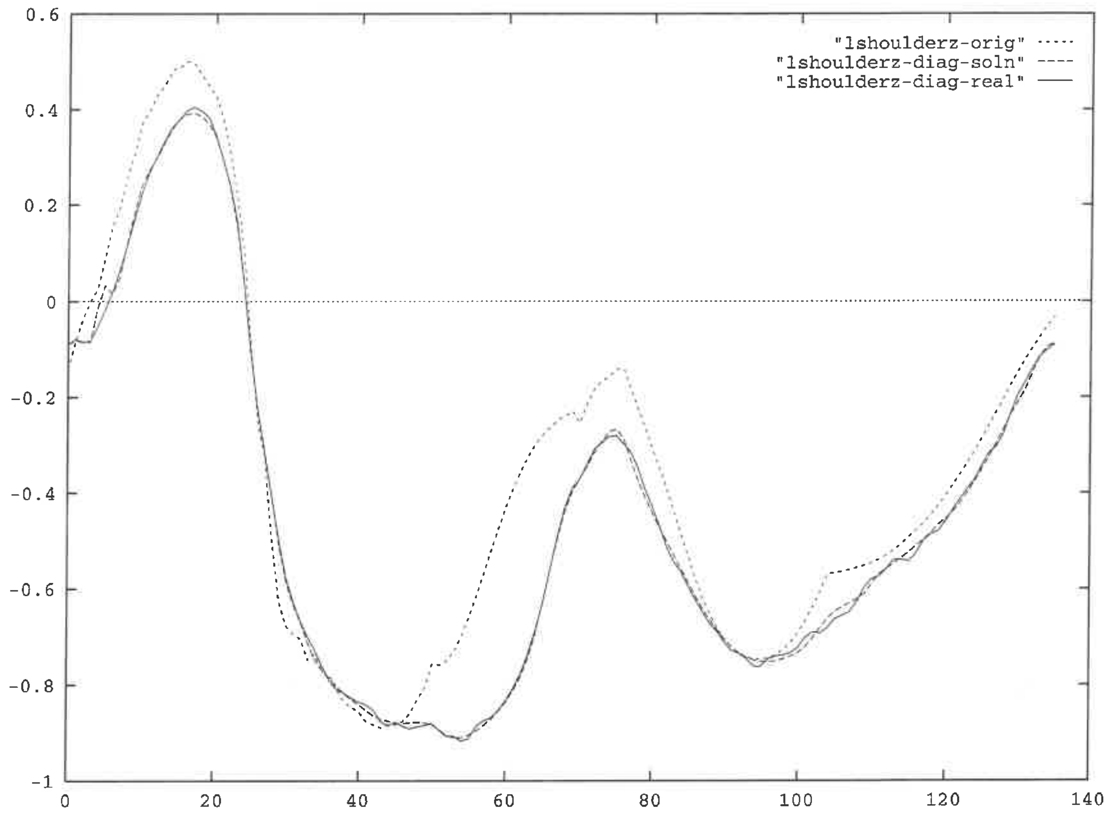


FIGURE 7.16. Left shoulder z quaternion component DOF functions for the original jump, computed diagonal jump, and the real-life diagonal jump.

Both knee and shoulder DOFs show a very good match (Figures 7.15,7.16). Aside from the motion capture noise in the real-life output the real-life and synthetic diagonal jump curves are virtually identical. The only difference appears to be at the extrema points, and even at those points the error is of the same scale as the motion capture noise.

The hip joint appears to show greater deviation between the real-life and the transformed motion (Figure 7.17). Still, the error appears to be less than twice as much as the motion capture noise. We draw attention to the emergent inflection points which are present in both synthetic and real-life curves.

The comparisons show that for this specific example our motion transformation algorithm accurately models motion modifications introduced by changing spacetime constraints. It is quite possible that for other motions the reality match would not be so favorable. There are a number of reasons why such discrepancies might occur:

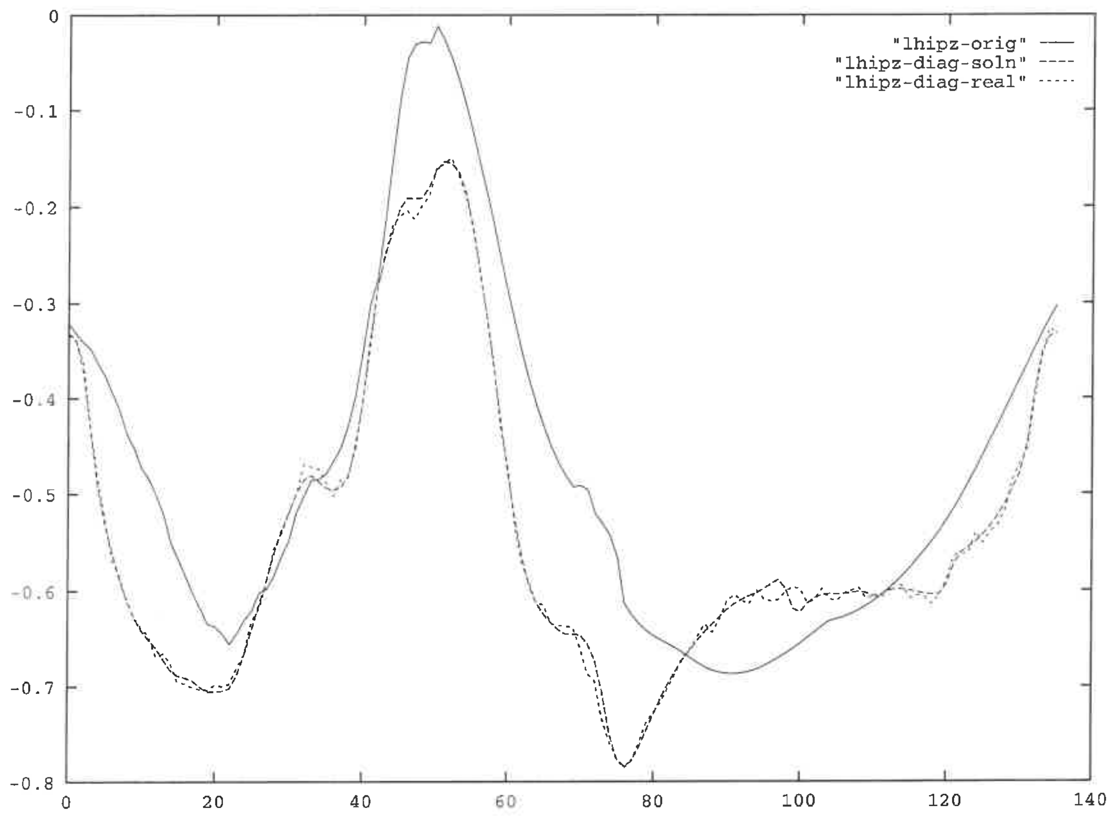


FIGURE 7.17. Left hip z quaternion component DOF functions for the original jump, computed diagonal jump, and the real-life diagonal jump.

- The simplification process might abstract away some aspects of the character which could turn out to be crucial in the transformed motion. For example, if we represent the upper body with the single mass point and use such a simplified character for the ice skating motion, we cannot accurately represent the torque change that occurs when arms are further away from the body.

- The muscle properties are not represented accurately enough. For example, the generalized muscle DOFs could end up producing forces which would not be in accordance with the natural processes in real muscles. In particular, natural muscles have very specific energy buildup patterns which are not modeled in our representation.

- The minimum displaced mass metric measures the static mass displacements and does not take into the account the velocity and acceleration of the body mass when comparing the two body poses.

7.8 Comparisons with Simple Motion Warping

Motion warping [Witkin 95] approaches require not only the modification of the pose constraints, but also additional "pin-down" constraints which essentially restrict the DOF curve deformation to the specific time range. These constraints need to be placed carefully because they greatly affect the motion warping results.

During the motion warping process, we often have to know a priori the realistic pose of the desired motion sequence at some key points in time. Sometimes these poses are easy to determine, but not always. For example, if we wanted to motion warp the broad jump into the broad jump with the 90 degree turn, it would not be sufficient to specify only the endpoint constraints (as in the spacetime formulation). We would also need to know the particular pose of the body while in the air and upon contact with the ground (Figure 7.11). It would be really hard to come up with those realistic poses without having seen the particular twisting body movement which is characteristic for such a turning jump.

The advantages of taking dynamics into the account when transforming motion are apparent in our examples. We demonstrate the power of dynamics constraints in the transformation process by showing the results of excluding dynamics constraints from the algorithm formulation. Although not identical to any of the existing motion warping methods, this non-dynamic algorithm has many of the same properties.

Again, we applied the transformations to the broad jump motion sequence. This time our goal was the obstacle broad jump which has the elevated destination constraint (Figure 7.13). Figure 7.18 shows the global translational DOF in the upward (y) direction for the original, dynamic and non-dynamic motion transformation sequences.

In order to help out the non-dynamic solution we introduced a number of additional constraints during the time when the feet are on the ground before the take-off, and after the landing (take-off and landing occur at the time samples 20 and 44 respectively). These additional constraints essentially made the portion of the curve after the landing identical for both dynamic and non-dynamic transformation. Note that without this a priori knowledge about the exact poses during those intervals we would not be able to pin down these portions of the curve; so, in essence, we are partially including the dynamic solution through these constraints.

The most apparent discrepancy between the dynamic and non-dynamic curve occurs during the flight phase. Since the non-dynamic algorithm knows nothing about gravity the parabolic path of the character is much lower and looks very unrealistic. Also, since the character jumps higher in the air the dynamic transformation has a greater dip during the pre-takeoff phase which allows the character to collect greater upwards momentum before it jumps higher. This dip is completely absent in the non-dynamic version.

This example, shows that even if we try to help out the standard warping method beyond what would be possible under normal circumstances of performing motion transformation, we still endup with grossly unrealistic motion.

7.9 Limitations

Although our algorithm has been proven effective for a wide range of input motion sequences and for an even wider spectrum of transformed motion animation sequences a number of deficiencies and possibilities for improvement remain. In this section we describe a number of limitations of our technique.

It appears that our methodology is best suited for the animation sequences containing

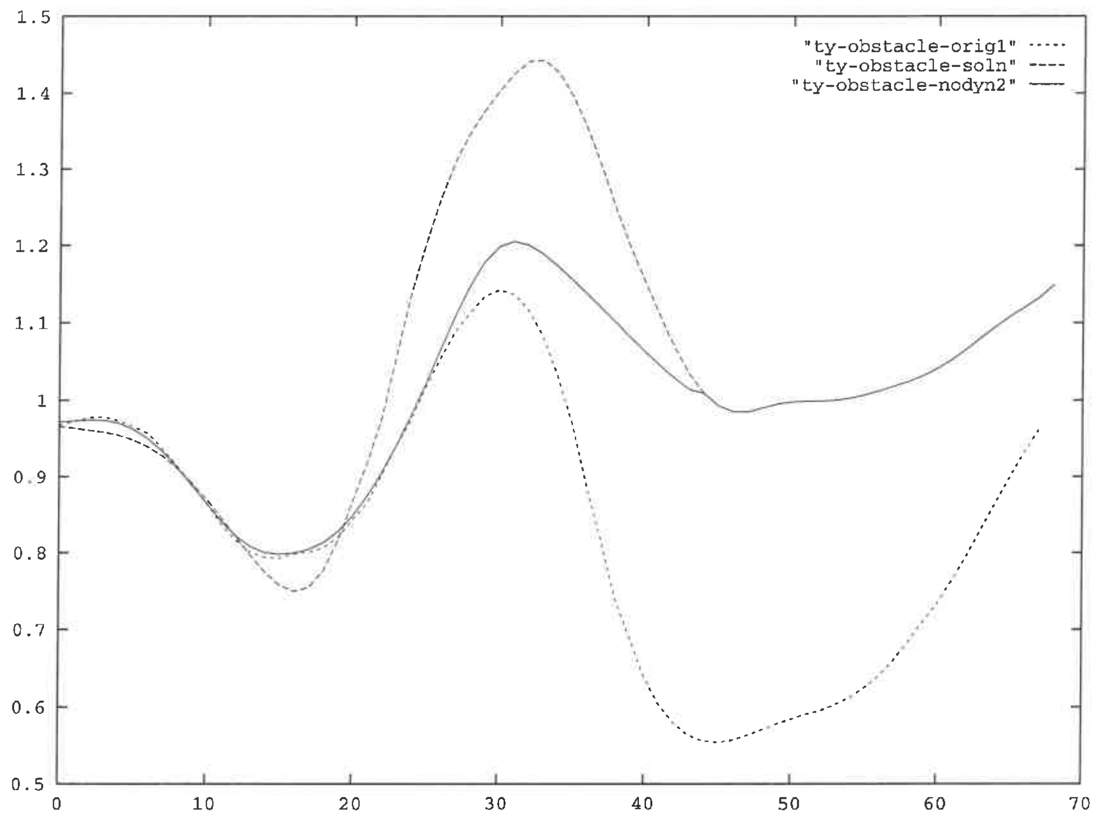


FIGURE 7.18. y component of the global translation shown for the original jump, jump onto the elevated obstacle with and withouth dynamics constraints.

high-energy, very dynamic character movement. Other less animated and more kinematic motion sequences such as picking up an object, climbing a ladder, rowing are not well suited for our dynamics transformation framework. Of course, it is always possible to apply our technique to such animations. However, the benefits of a full-blown dynamics representation would not be large since much simpler methods could be used for transformation without a significant loss of the output motion quality. Most overly constrained motions where both feet and hands are constrained for a large portion of the animation (e.g. climbing a ladder) display very few dynamics properties and are consequently not a good candidate for the input to our algorithm. Other "lethargic" motions such as transferring the weight from one leg to the other, raising a hand while standing, or extremely slow walking would also not be good candidate input motions for our method. In general, the problem of non-realism for such motions is much less of an issue. The more a particular motion contains visible dynamic properties, the more suitable it is for our motion transformation algorithm.

During the development of our method, very little time was devoted to the concerns of speed. Our method is not interactive. Each change in the spacetime formulation produces a new animation in 2-3 minutes. This time lag would prove to be quite tedious for most animators. However, it is our belief that with a few adjustments and code optimizations this delay could be reduced significantly, at least down to the 30 second range. This problem can be further alleviated by interactively displaying the current state of the solution during the optimization. A mistake or an undesirable result often becomes apparent to the animator long before the optimization has converged. With interactive display the user can spot the mistake early and modify the motion again.

Another shortcoming of our approach is that large portions of the motion fitting algorithm stage are performed manually. This stage is performed only once per input motion sequence, so the effort spent by the motion library creator is amortized over the large number of possible transformed animation sequences. In addition, unlike the robot controller based methods for character animation synthesis, the expertise required to perform these manual stages of the motion fitting is small. Many simplification decisions as well as handle selections appear to be quite intuitive. Nevertheless, automating this manual decision making process would enable on-the-fly construction of the physically based spacetime formulation from the input animation.

Furthermore, specific decisions in the motion fitting stage directly affect the types of modifications that can be performed to the motion. For example, if we wanted to add the waving gesture to the human running sequence we cannot simplify the waving arm down to a single rigid object as in the biped model (see Section 7.4.2). Consequently, a modification of the simplification process might be necessary in order to achieve a certain motion transformation which was unforeseen during the motion fitting process. These modifications break down the motion library concept.

Since all dynamics computations are done on the simplified model, there is no guarantee that the reconstruction stage of the algorithm would preserve the dynamics properties. In fact, the final motion sequence is *not* physically realistic in the absolute sense, simply due to the fact that no dynamics computations are done on the full character model. Our algorithm preserves the *essential* physical properties of the motion. This makes our algorithm ill-suited for applications which require that a resulting motion contains all the forces involved in the character locomotion.

Simplifications which reduce the large portions of the body down to a single mass point effectively ignore the inertia moments which are present in the original motion, but are completely ignored in the simplified model representation. As a result, such simplifications cannot be used if moments of inertia are important for the given motion sequence (e.g. any motion involving twisting or rotating with large angular velocities). An alternative simplification which would account for the moments of inertia could include a larger single objects such as a "stick" which could generate the moments of inertia around the relevant axes of rotation.

During the spacetime optimization we ignore self-collision of character's extremities. For example, there is nothing in our dynamics representation that would prevent an arm going through the torso or a leg freely passing through the other leg. These problems can be partially addressed, for example, by introducing the "ankle closeness" penalties in the objective function. Of course, such penalties can also alter the motion appearance by keeping the ankles unnaturally far apart. A better solution would involve collision detection for the various body parts that are likely to collide. For example, we could check for collision between the left and right shin, as well as the hand and forearm intersection with the body torso. Each time the collision occurs, we could introduce the mechanical constraint which would prevent inter-penetration of the body parts.

Chapter 8

CONCLUSION

This thesis describes a novel algorithm for generating realistic animation sequence from a single motion capture sequence. In this chapter we enumerate the main contributions of this work. We proceed to describe what animation problems are best suited for our algorithm. Finally, we suggest a number of interesting and challenging directions for extending our work.

8.1 Contributions

Our algorithm presents the first solution to the problem of editing captured motion taking dynamics into consideration. Dynamic properties of motion are taken into the account by constructing the physically based spacetime constraints formulation of motion. During the editing process various aspects of the dynamic representation are modified. The resulting change in motion is, consequently, mapped back into the space of the input motion in order to produce the final animation. Our algorithm provides a large number of dynamics related motion modifiers to be applied to the original sequence. No current motion editing methods consider preserving the physical properties of motion.

Furthermore, we provide a rich set of motion modification operators which provide an intuitive control interface to all aspects of the motion sequence. Previously, setting constraints at different times was the only tool for motion transformation available to the animators.

The ability to preserve dynamics of the motion, and the existence of a rich set of motion controls enables the creation of motion libraries from a single input motion sequence. Once the original motion is fitted onto the spacetime constraints motion model, our motion transformation model can be presented to the user as tool which can generate the motion that meets the exact specifications of the given animation. So an animator can be presented with a few motion libraries such as human run, human jump, karate leg kick, soccer ball kick, tennis serve. Based on the need, the animator can pull the appropriate library "off the shelf" and edit the motion so that it meets the needs of the animation.

By way of character simplification, we also present the first method which successfully uses spacetime constraints formulation to generate motion sequences for non-trivial characters — characters with complexities comparable to that of a human actor.

We also describe a novel methodology for mapping motion to/from characters with drastically different kinematic structures. We show how a collection of handles and the displaced mass metric can be used to correlate motion between characters of different dimensions, mass properties and even different kinematic structure. These ideas could broaden the applicability of motion capture data to animation of characters that do not exist in nature.

This thesis also demonstrates that complex dynamic systems can be successfully controlled with simpler ones. This realization could potentially have significant impact outside of computer graphics, in particular, in fields which are concerned with real-time control of dynamic systems (e.g. robotics).

8.2 Applications

This method is best suited for the transformation of motion sequences that are highly dynamic:

- jumps, leaps, hops
- runs, skips, trots, high-energy modern dance elements
- karate leg kicks, soccer ball kicks, football punts
- ball throws, tennis serve, shot put, punches and arm kicks
- gymnastic disciplines, board dives

Other less energetic motions such as a slow arm movements in ballet, slow walking, reaching and picking up an object appear to have little use for dynamic analysis. Consequently our method would most likely be an overkill for such input motion. However, other aspects of our method such as retargeting motion to different characters would still be just as useful.

The physically based motion transformation methods have a natural fit with the realistic motion for video games, virtual avatars, telepresence applications and human and animal motion simulations. In such environments, the realism of the motion is important. Furthermore the motion is governed by a certain number of real-time input parameters. These parameter, in turn, specify a number of mechanical and pose constraints. Thus, a number of high-detail captured motion sequences could be transformed to meet the constraints introduced by the real-time input of the user producing a seamless perception of simulated reality that responds to the human input.

8.3 Future Directions

The results described in this thesis open new avenues towards reusable animation that can be utilized not only in the film and video game industry, but more importantly on every home PC. It won't be long before every PC will have sufficient 3D rendering capabilities to enable each one of us to use computer animation as a expressive medium, much like web pages are an ubiquitous form of expression today. I see reusable motion as a crucial concept that will enable most of us, non-skilled animators, to become capable, expressive storytellers. The approach we described lays down the foundation for creating such motion libraries. To this end, additional work needs to be done in order to take our "proof of concept" and turn it into a usable motion editing tool.

Large portions of the motion fitting algorithm stage are performed manually which creates the need for certain amount of human preprocessing for each input sequence. There are a number of possible ways to automate this process by analyzing the structure of the input motion in order to determine which aspects of the motion are more important than the others. In addition, a number of mechanical constraints can be postulated by analyzing the absolute movement of various character body points. While it is unlikely that the entire motion fitting process can be fully automated, the human intervention can be reduced to a few decision approvals suggested by the algorithm.

So far our user interface consists of the animation viewer which contains various UI tools for setting up the optimization problem, for interactive manipulation of $\mathbf{q}(t)$ curves and for visually examining the motion sequences. The character transformation hierarchy, as well as specifics of certain constraints and objectives are specified within C++ files which are dynamically loaded into the system. The motion sequences are stored in files which contain coefficients for each DOF. A number of additional features need to be made accessible to the animator through the UI. Currently, a significant knowledge of C++ character and constraint building blocks is required in order to edit a given captured motion sequence. All of these aspects should be eventually transfered into the UI. Although these issues do not involve creating new technologies, some software engineering effort is required.

Additional efforts need to be made towards the interactive speed of the motion transformation process. The current system was designed as an experimentation tool with little concern towards efficiency. Additional exploitation of sparsity within the optimization problem definition can drastically reduce the computation times. A non-uniform representation of the DOFs can significantly reduce the size of the problems as well.

The methods of physically based spacetime transformation can also be applied to non-realistic motion data. For example, our transformation framework can be modified to allow motion editing of the arbitrary keyframed character animation. In this setup, our motion transformation methods can be thought of as a "beleiveabllity filter" which would improve the physical plausibility of the motion sequence while still preserving the exaggerated non-physical gestures frequently used in character animation.

Further exploration in the use of this method for motion retargeting for the characters with different kinematic and dynamic properties is needed. In particular, significant changes in the kinematic structure and the mass distribution of the character can render the minimum displaced mass objective inapplicable. Certain augmentation of the displaced mass metric should be developed to deal with the significant structural changes of the character.

Earlier we mentioned that this work suggests a methodology for controlling the complex dynamic systems with significantly simpler models. These ideas can be further developed into a real-time optimal robot control planner. Such a planner would internally keep the simplified representation of the robot dynamics. It would, then, predict and plan the outcome of the motion within a small window of time by rapidly solving the spacetime optimization problem for that time period. The model simplification reduces the complexity of the optimizations and enables such real-time computations. Such controllers would give raise to robots moving in very natural ways.

Finally, these methods can be used for motion analysis in biomechanics and help prove the very ideas of common principles of natural motion that motivated this algorithm.

References

R. M. Alexander. *Optimum walking techniques for quadrupeds and bipeds.* J. Zool., London, 192:97–117, 1980. (pp. 29, 71)

R. M. Alexander. *Optimization and gaits in the locomotion of vertebrates.* Physiol. Rev., 69:1199–1227, 1989. (pp. 29, 71)

R. M. Alexander. *Optimum take-off techniques for high and long jumps.* Phil. Trans. R. Soc. Lond., 329:3–10, 1990. (pp. 29, 71)

R. M. Alexander. *Optimum timing of muscle activation for simple models of throwing.* J. Theor. Biol., 150:349–372, 1991. (pp. 29, 71)

K.N. An, B.M. Kwak, E.Y. Chao, and B.F. Morrey. *Determination of muscle and joint forces: A new technique to solve the indeterminate problem.* J. of Biomech. Eng., 106:663–673, November 1984. (pp. 29, 42)

David Baraff. *Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies.* In Computer Graphics (SIGGRAPH 89 Proceedings), volume 23, pages 223–232, July 1989. (p. 24)

David Baraff. *Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulation.* In Computer Graphics (SIGGRAPH 90 Proceedings), volume 24, pages 19–28, August 1990. (p. 24)

David Baraff. *Coping with friction for non-penetrating rigid body simulation.* In Computer Graphics (SIGGRAPH 91 Proceedings), volume 25, pages 31–40, July 1991. (p. 24)

David Baraff and Andrew Witkin. *Large Steps in Cloth Simulation.* In Michael Cohen, editor, *Computer Graphics (SIGGRAPH 98 Proceedings)*, pages 43–54, July 1998. (p. 24)

R. Blickhan and R. J. Full. *Similarity in multilegged locomotion: bouncing like a monopode.* J Comp. Physiol. A, 173:509–517, 1993. (pp. 17, 29, 57)

J.E Bobrow, S. Dubowsky, and Gibson J. S. *Time-optimal control of robotic manipulators along specified paths.* International Journal of Robotics Research, 4(3):3–17, 1985. (p. 26)

Armin Bruderlin and Lance Williams. *Motion Signal Processing.* In Computer Graphics (SIGGRAPH 95 Proceedings), pages 97–104, August 1995. (pp. 17, 27)

Michael F. Cohen. *Interactive spacetime control for animation.* In Computer Graphics (SIGGRAPH 92 Proceedings), volume 26, pages 293–302, July 1992. (p. 25)

Roy D. Crownninshield and Richard A. Brand. *A physiologivcallly based criterion of muscle force prediction in locomotion.* J. Biomechanics, 14(11):793–801, 1981. (pp. 29, 42)

Tony DeRose, Michael Kass, and Tien Truong. *Subdivision Surfaces in Character Animation.* In Michael Cohen, editor, *Computer Graphics (SIGGRAPH 98 Proceedings)*, pages 85–94, July 1998. (p. 24)

B. Donald, P. Xavier, J. Canny, and J. Reif. *Kinodynamic motion planning.* Journal of the ACM, 40(5), 1993. (p. 26)

J. Dul, M.A. Townsend, R. Shiavi, and G.E. Johnson. *Muscular Synergism — I. On criteria for load sharing between synergistic muscles.* J. Biomechanics, 17(9):663–673, 1984. (pp. 29, 42)

Michael Gleicher. *Motion Editing with Spacetime Constraints.* In Michael Cohen and David Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 139–148. ACM SIGGRAPH, April 1997. ISBN 0-89791-884-3. (pp. 17, 27, 28)

Michael Gleicher. *Retargeting Motion to New Characters.* In Computer Graphics (SIGGRAPH 98 Proceedings), pages 33–42, July 1998. (pp. 17, 27)

Michael Gleicher and Peter Litwinowicz. *Constraint-based Motion Adaptation.* The Journal of Visualization and Computer Animation, 9(2):65–94, 1998. (pp. 17, 27, 28)

Jessica K. Hodgins, Paula K. Sweeney, and David G. Lawrence. *Generating natural-looking motion for computer animation.* In Proceedings of Graphics Interface 92, pages 265–272, May 1992. (pp. 26, 43)

J. K. Hodgins and N. S. Pollard. *Adapting simulated behaviours for new characters.* SIGGRAPH 97, pages 153–162, 1997. (p. 26)

Cornelius Lanczos. *The Variational principles of Mechanics.* University of Toronto Press, 1970. (p. 45)

J.C. Latombe. *Robot motion planning.* Kluwer Academic Publisheres, 1991. (p. 26)

Zicheng Liu, Steven J. Gortler, and Michael F. Cohen. *Hierarchical Spacetime Control.* In Computer Graphics (SIGGRAPH 94 Proceedings), July 1994. (p. 25)

Zicheng Liu. *Efficient Animation Techniques Balancing Both User Control and Physical Realism.* PhD thesis, Princeton University, November 1996. (p. 45)

Matthew Moore and Jane Wilhelms. *Collision Detection and Response for Computer Animation.* In Computer Graphics (SIGGRAPH 88 Proceedings), volume 22, pages 289–298, August 1988. (p. 24)

J. Thomas Ngo and Joe Marks. *Spacetime Constraints Revisited.* In Computer Graphics (SIGGRAPH 93 Proceedings), volume 27, pages 343–350, August 1993. (p. 27)

M.G. Pandy, F. E. Zajac, E. Sim, and W. S. Levine. *An optimal control model of maximum-height human jumping.* J. Biomechanics, 23:1185–1198, 1990. (p. 29)

M.G. Pandy and F. E. Zajac. *Optimum timing of muscle activation for simple models of throwing.* J. Biomechanics, 24:1–10, 1991. (p. 29)

M.G. Pandy, F. C. Anderson, and D. G. Hull. *A Parameter Optimization Approach for the Optimal Control of Large-Scale Musculoskeletal Systems.* J. of Biomech. Eng., pages 450–460, November 1992. (pp. 29, 42)

A. Pedotti, V. V. Krishnan, and L. Stark. *Optimization of muscle-force sequencing in human locomotion.* Math. Biosci., 38:57–76, 1978. (pp. 29, 71)

L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *The Mathematical Theory of Optimal Processes.* John Wiley and Sons, New York, N.Y., 1962. (p. 42)

Marc H. Raibert and Jessica K. Hodgins. *Animation of dynamic legged locomotion.* In Computer Graphics (SIGGRAPH 91 Proceedings), volume 25, pages 349–358, July 1991. (pp. 26, 43)

C. Rose, B. Guenter, B. Bodenheimer, and M. Cohen. *Efficient Generation of Motion Transitions using Spacetime Constraints.* In Computer Graphics (SIGGRAPH 96 Proceedings), pages 147–154, 1996. (p. 25)

C. Rose, M. F. Cohen, and B. Bodenheimer. *Verbs and Adverbs: Multidimensional Motion Interpolation.* IEEE Computer Graphics & Applications, 18(5), September – October 1998. (pp. 17, 28)

A. Seireg and R. J. Arvikar. *The prediction of muscular load sharing and joint forces in the lower extremities during walking.* J. Biomechanics, 8:89–102, 1975. (pp. 29, 42)

Z. Shiller and Dubowsky S. *On Computing the Global Time-Optimal Motions of Robotic Manipulators in the Presence of Obstacle.* IEEE Transactions on Robotics and Automation, 7(6), 1991. (p. 26)

K. G. Shin and N. D. McKay. *Minimum-time control of robotic manipulators with geometric path constraints.* IEEE Transactions on Automatic Control, AC-30(6):531–541, 1985. (p. 26)

K. Shoemake. *Fiber bundle twist reduction.* In P. Heckbert, editor, *Graphics Gems IV*, pages 230–236. Academic Press, 1994. (p. 67)

Karl Sims. *Evolving Virtual Creatures.* In Computer Graphics (SIGGRAPH 94 Proceedings), July 1994. (p. 27)

Michiel van de Panne and Eugene Fiume. *Sensor-actuator Networks.* In Computer Graphics (SIGGRAPH 93 Proceedings), volume 27, pages 335–342, August 1993. (pp. 26, 27)

Michiel van de Panne and Eugene Fiume. *Virtual Wind-up Toys.* In Proceedings of Graphics Interface 94, May 1994. (pp. 26, 27)

M. van de Panne. *From Footprints to Animation.* Computer Graphics Forum, 16(4):211–224, 1997. (p. 26)

Andrew Witkin and Michael Kass. *Spacetime Constraints.* In Computer Graphics (SIGGRAPH 88 Proceedings), volume 22, pages 159–168, August 1988. (pp. 15, 25)

Andrew Witkin and Zoran Popović. *Motion Warping.* In Computer Graphics (SIGGRAPH 95 Proceedings), August 1995. (pp. 17, 27, 101)

P. Xavier. *Provably-good approximation algorithms for optimal kinodynamic robot motion plans.* PhD thesis, Cornell University, April 1992. (p. 26)