# Searching Image Databases Using Appearance Models

A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy in the Faculty of Medicine, Dentistry, Nursing and Pharmacy

2004

Ian M. Scott

Division of Imaging Science and Biomedical Engineering

# Contents

**Glossary** **10**

**1 Introduction** **15**

1.1 Motivation . . . . . 16

1.1.1 Object Detection, Why and How? . . . . . 17

1.1.2 Object Detection and Computer Vision . . . . . 18

1.2 Overview . . . . . 19

**2 Literature Review** **21**

2.1 Metadata Based Searching . . . . . 21

2.2 Low-Level Feature-Based Search . . . . . 23

2.2.1 Problems with the Low-Level Feature Approach . . . . . 24

2.2.2 Recent Results with Low-Level Features . . . . . 25

2.3 Object Detection . . . . . 26

2.3.1 Single Image Matching . . . . . 26

2.3.2 Multiple Image Matching . . . . . 27

2.3.3 Statistical Image Matching . . . . . 28

2.3.4 Shape-Based Approaches . . . . . 30

2.3.5 Parts-Based Representations . . . . . 32

2.4 DBMSs and Complete Systems . . . . . 34

2.5 Test Databases and Evaluation . . . . . 35

2.6 Discussion . . . . . 36

**3 Building GMM Network Classifiers** **37**

3.1 Background . . . . . 37

3.2 Implementing GMM Networks . . . . . 40

3.3 Initial Experiments and Results . . . . . 45

3.3.1 Results . . . . . 45

3.3.2 Confirming the Separability of the Data . . . . . 50

3.3.3 Speed . . . . . 51

3.4 Further Experiments . . . . . 52

3.5 Discussion and Conclusions . . . . . 54

**4 Building Support Vector Machine Classifiers** **55**

4.1 Support Vector Machines . . . . . 56

4.1.1 Introduction to SVMs . . . . . 56

4.1.2 Applications and Extensions . . . . . 61

4.1.3 Using Chunking to Train SVMs on Large Datasets . . . . . 62

4.2 Comparison of SVMs with GMM Networks . . . . . 64

4.3 Choosing the Best SVM Training Parameters . . . . . 65

4.3.1 Diameter of the Minimum Enclosing Hypersphere . . . . . 68

4.3.2 Validation of Predicted Test Error . . . . . 71

4.3.3 Automating the Search for RBF Width . . . . . 72

4.3.4 Summary . . . . . 76

4.4 Sequential Minimal Optimisation . . . . . 77

4.4.1 Implementation of the SMO algorithm . . . . . 80

4.4.2 Adding Support for MEH Calculation . . . . . 82

4.4.3 Adding Kernel Value Caching to SMO . . . . . 84

4.4.4 Calculation of Best RBF Width . . . . . 89

4.5 Comparison of Chunking and SMO Methods . . . . . 93

4.6 An SVM-Based Face Detector . . . . . 93

4.7 Discussion and Conclusions . . . . . 96

4.7.1 Comparison with Previously Published Results . . . . . 96
4.7.2 Further Work . . . . . 99
4.7.3 Discussion . . . . . 99

**5 Introduction to Appearance Modelling 101**

5.1 Constructing Active Appearance Models . . . . . 102
5.2 Fitting Active Appearance Models . . . . . 104
5.3 Related Work . . . . . 106
5.4 Summary . . . . . 107

**6 Improving AAM Performance Using Local Image Structure 108**

6.1 Introduction and Motivation . . . . . 108
6.2 Related Work . . . . . 110
6.3 Local Structure Detectors . . . . . 113
6.3.1 Non-linear Transforms . . . . . 113
6.3.2 Gradient Structure . . . . . 113
6.3.3 Corner and Edge Structure . . . . . 114
6.3.4 Developing Measures of Cornerness and Edgeness . . . . . 115
6.3.5 Implementation Issues . . . . . 117
6.4 Experiment with Spinal X-Rays . . . . . 121
6.4.1 Results with Spinal X-Rays . . . . . 123
6.4.2 Simulated Multi-modal Experiments . . . . . 124
6.4.3 Results with Faces . . . . . 126
6.5 Discussion and Conclusions . . . . . 128

**7 Further Experiments with Texture AAMs 129**

7.1 Statistics . . . . . 130
7.1.1 Binomial Statistics . . . . . 130
7.1.2 Linear Modelling of Errors . . . . . 132
7.1.3 Bootstrap Statistics . . . . . 135
7.2 A Better Non-linear Normaliser . . . . . 138

7.2.1 Future Work on Normalisers . . . 141
7.3 More Feature Descriptors . . . 141
7.3.1 Cartesian Differential Invariant Texture AAMs . . . 142
7.3.2 Random Output Texture Descriptors . . . 144
7.4 Discussion and Conclusions . . . 145
7.4.1 Future Work . . . 146

**8 Improving AAM Performance Using Elliptical Limits 148**
8.1 Model Limits for AAM Search . . . 148
8.2 Background and Related Work . . . 150
8.3 Ellipsoidal Limits . . . 151
8.4 Experiments . . . 151
8.5 Discussion and Conclusions . . . 155

**9 Object Detection Using Combined AAMs and SVMs 158**
9.1 Background and Relationship to Existing Work . . . 158
9.1.1 An AAM as Better Normaliser or Feature Detector for a Statistical Classifier . . . 159
9.1.2 An SVM as Better Quality of Fit Measure for an AAM . . . 160
9.1.3 Combining High-level Feature Detectors, and Advanced Statistical Classifiers . . . 161
9.2 AAM-SVM Searching and Training . . . 162
9.3 Building the AAM . . . 162
9.4 Radius of Convergence of AAMs . . . 166
9.4.1 Effect of AAM Improvements on Radius of Convergence . . . 167
9.5 AAM Search . . . 168
9.5.1 Starting Grid . . . 168
9.5.2 Measuring Fitting Error . . . 169
9.5.3 Dealing with Exceptional Events During AAM Search . . . 170
9.6 Building the SVM . . . 172
9.7 Measuring Performance . . . 173

9.7.1 False Positive Rate . . . 173
9.7.2 False Negative Rate . . . 175
9.7.3 Area Above the ROC Curve (AARC) . . . 175
9.7.4 PFPR/TPR—an Overall Statistic for Database Retrieval Performance . . . 176
9.8 What to Use for a Feature Vector? . . . 177
9.8.1 Results with Residuals as the Feature Vector . . . 178
9.8.2 Results with AAM Parameters as the Feature Vector . . . 181
9.9 Conclusions . . . 184

**10 Further Experiments with AAM-SVMs 185**

10.1 Comparison with Existing Methods . . . 186
10.1.1 Comparison with the Patch SVM Detector . . . 186
10.1.2 Comparison of the Original and Improved AAMs in an AAM-SVM . . . 188
10.2 Can Faces be Detected in 100 Pixels? . . . 193
10.2.1 Algorithm Speed . . . 194
10.3 Iterative Refinement with Positive Data . . . 196
10.3.1 Starting with a Small Positive Database . . . 197
10.3.2 Increasing the Size of the Positive Training Set Without a Larger Database . . . 199
10.4 Multistage Approach to Object Detection . . . 202
10.5 Statistical Significance . . . 204
10.6 Discussion and Conclusions . . . 206
10.6.1 Implications of the AAM-SVM Results for Appearance Modelling 206
10.6.2 Future Improvements to the AAM-SVM . . . 207
10.6.3 Better SVM training . . . 209
10.6.4 Variations on the AAM-SVM Method . . . 211
10.6.5 Could the AAM-SVM be Used for Image Retrieval? . . . 212
10.6.6 Conclusions . . . 214

**11 Discussion 215**

11.1 Summary of Original Work and Results . . . . . 215
11.1.1 Building Statistical Classifiers . . . . . 215
11.1.2 Improving the Accuracy of AAMs . . . . . 216
11.1.3 Combined AAM-SVM Object Detection . . . . . 217
11.2 Summary of Further Work . . . . . 217
11.3 Final Conclusions . . . . . 219

**A Contributions to VXL** **221**
A.1 Introduction . . . . . 221
A.2 Background . . . . . 222
A.3 `vsl`—VXL's Binary IO Library . . . . . 222
A.3.1 Dependency Issues and Errors . . . . . 223
A.3.2 IO for Shared Pointers. . . . . 223
A.3.3 Cross Platform Binary IO . . . . . 225
A.4 `vil2`—A Replacement Image Library . . . . . 226
A.5 Discussion . . . . . 227

**B Database Descriptions** **228**

**Bibliography** **233**

# Abstract

This thesis examines the problem of retrieving images from large image databases by detecting objects of interest. The approach adopted involves combining model-based recognition, using Active Appearance Models (AAMs,) with a sophisticated statistical classifier. An experimental evaluation of two published methods for achieving very high detection accuracy leads to the choice of a Support Vector Machine (SVM) classifier with iterated negative-example refinement. The method for training the SVM is further developed, leading to a fully-automated approach to choosing training parameters. Similarly, the standard AAM approach is extended, leading to significant improvements in performance. One contribution is the "Texture AAM," which replaces the grey-level values, ordinarily used in the AAM's shape-normalised patch, with non-linear descriptors of local edge and corner structure. Another contribution is to investigate, more fully than previously, the effects of limiting the AAM parameters during image search—leading to a reappraisal of the optimal approach. Finally, the improved AAM and SVM are combined to create a novel "AAM-SVM" system for image retrieval, that is shown to be significantly more effective than either method alone. Extensive experiments are performed to analyse the behaviour of the system, demonstrating that detection accuracy is superior to that of a simple patch-based SVM approach that is among the state-of-the-art methods. The approach is, however, many times too slow for practical applications. This leads to an initial investigation of a multi-stage approach, which uses an AdaBoost patch classifier as a first stage. The speed of this system approaches that necessary for practical image database search.

# Declaration

No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

# Copyright Statement

# Acknowledgements

I would like to thank the following:

My supervisor, Prof. Chris Taylor OBE, for his guidance and support. His ability to find positive conclusions when I could only see negative results has been a huge encouragement.

Dr. Tim Cootes, who besides providing useful AAM software, has acted like an unofficial second supervisor.

Dr. Carole Twinning and Dr. Iain Buchan for their advice on things mathematical and statistical.

David Cristinacce and Dr. Kevin de Souza for their efforts in the multi-stage face detection and 3D Texture AAM experiments.

The rest of my colleagues at ISBE who have helped throughout my time here, including Judith Adams, Danny Allen, Brian Atherton, Kola Babalola, Frank Bettinger, Christine Beeston, Dave Burke, Louise Butcher, Angela Castledine, Patrick Cherry, Christine Cummings, Dave Ellard, Jermaine Gilmore, Panachit Kittipanya-ngam, Warren Mitto, Dave Nuttal, Vlad Petrovich, Martin Roberts, Roy Schestowitz, Shelagh Stedman, Graham Vincent, Gavin Wheeler, and Andy Yu.

The VXL-maintainers for providing useful software.

Dr. Xianping Ge for putting his SMO code in the public domain at my request.

The Royal Commission for the Exhibition of 1851, and BT, for generous funding.

My deepest gratitude goes to my wife, Jen, for her unstinting support and tolerance. Without her, this thesis would not have been possible.

# Glossary

## Abbreviations

| | |
|---|---|
| **AARC** | Area Above the ROC Curve |
| **AAM** | Active Appearance Model |
| **AAM-SVM** | Combined AAM and SVM |
| **ASM** | Active Shape Model |
| **CBIR** | Content Based Image Retrieval |
| **CDI** | Cartesian Differential Invariants |
| **ceg** | corner-edge-gradient |
| **CMU** | Carnegie-Mellon University |
| **DBMS** | DataBase Management System |
| **DXA** | Dual X-ray Absorptiometry |
| **EM** | Expectation Maximisation |
| **EMD** | Earth Mover's Distance |
| **GiB** | $2^{30}$ Bytes |
| **GMM** | Gaussian Mixture Models |
| **IMS** | Interactive Multimedia Services |
| **IO** | Input/Output |
| **ISBE** | Imaging Science and Biomedical Engineering, U. Manchester |
| **KKT** | Karush Kuhn Tucker |
| **KPCA** | Kernel Principle Component Analysis |
| **LDA** | Linear Discriminant Analysis |
| **MDL** | Minimum Description Length |
| **MEH** | Minimum Enclosing Hypersphere |

| | |
|---|---|
| **MiB** | $2^{20}$ Bytes |
| **MLP** | Multi-Layer Perceptron |
| **MRI** | Magnetic Resonance Imaging |
| **PCA** | Principal Component Analysis |
| **PDF** | Probability Density Function |
| **PDM** | Point Distribution Model |
| **PFPR/TPR** | *Perfect* False Positive Rate to True Positive Rate ratio |
| **PRFR** | Pairwise Reinforcement of Feature Responses |
| **QBIC** | Query By Image Content |
| **QP** | Quadratic Programming |
| **RBF** | Radial Basis Function |
| **RHUL** | Royal Holloway, University of London |
| **RMS** | Root Mean Square |
| **ROC** | Receiver Operating Characteristic |
| **SMO** | Sequential Minimum Optimisation |
| **SVM** | Support Vector Machine |
| **VC** | Vapnik Chervonenkis |
| **VXL** | Vision-X-Libraries |

# Definitions

**specificity** or **true negative rate** $= \frac{\#\text{true negatives}}{\#\text{true negatives}+\#\text{false positives}}$

Ability of binary classifier to reject negative examples

**sensitivity** or **true positive rate** or **recall ratio** $= \frac{\#\text{true positives}}{\#\text{true positives}+\#\text{false negatives}}$

Ability of binary classifier to accept all positive examples

**false positive rate** $= \frac{\#\text{false positives}}{\#\text{true negatives}+\#\text{false positives}}$

$1 -$ specificity

**false negative rate** $= \frac{\#\text{false negatives}}{\#\text{true positives}+\#\text{false negatives}}$

$1 -$ sensitivity

**precision** $= \frac{\#\text{true positives}}{\#\text{true positives}+\#\text{false positives}}$

Version of specificity relevant in large database retrieval context

**accuracy** $= \frac{\#\text{true positives}+\#\text{true negatives}}{\#\text{all examples}}$

Ability of any classifier not to make mistakes

# Image Databases

Faces

| | |
|---|---|
| `CMU` | Published by Rowley *et al.*[122, 145] |
| `XM2VTS` | Published by Messer *et al.*[93] |
| `BioID` | Published by Jesorsky *et al.*[73] |
| `expression` | Collected by ISBE |
| `all22r` | `BioID` + `expression` + other ISBE databases |
| `webimagesB` | Collected by the author |
| `webimagesC` | Collected by the author |

Other

| | |
|---|---|
| `UWash` | Published by Shapiro[137] |
| `DXAspinesA` | Described by Smyth *et al.*[141] |

See appendix B for examples from the databases.

# Widely Used Symbols

| | |
|---|---|
| $C$ | SVM soft margin error weighting |
| $D$ | MEH diameter |
| $e$ | Edgeness texture descriptor |
| $E$ | Total AAM texture error |
| $E$ | Local image energy for cornerness calculation |
| $E_i$ | $i^{\text{th}}$ KKT error for SVM training |
| $f(\cdot)$ | Classification score—$f(\cdot) > 0$ is a hit |
| $f(\cdot)$ | AAM texture vector normalisation function |
| $\boldsymbol{g}$ | Gradient texture descriptor |
| $I(u, v)$ | Image pixel intensities |
| $K(\cdot,\cdot)$ | SVM kernel function |
| $\boldsymbol{M}$ | Matrix of image gradient products for edge and corner calculation |
| $M$ | SVM margin width |
| $n_{\text{select}}$ | Number of false positives found per refinement iteration |
| $\boldsymbol{p}$ | AAM parameters |
| $\boldsymbol{r}$ | AAM texture residual |
| $\boldsymbol{r}_0$ | Texture residual for highest-resolution level of multi-res. AAM |
| $r$ | Cornerness texture descriptor |
| $R$ | Classifier's expected error |
| $R_{\text{struct}}^{\text{upper bound}}$ | Upper bound on SVM's expected test error |
| $R_{\text{RMS}}$ | Scale of AAM control points |
| $\boldsymbol{x}_i$ | $i^{\text{th}}$ example for classifier training |
| $y_i$ | $i^{\text{th}}$ example's label for classifier training |
| $\alpha_i$ | $i^{\text{th}}$ Lagrange multiplier for SVM training |
| $\sigma$ | RBF radius |
| $\Phi$ | High-dimensional projection implied by kernel function $K(\cdot,\cdot)$ |
| $\Psi$ | Lagrangian for SVM training |
| $\#(\cdot)$ | Number of $\cdot$ |

# Chapter 1

# Introduction

The work described in this thesis attempts to answer the following question: Can appearance modelling be used for image database retrieval, and if so how?

In recent times, the prevalence of digital multimedia databases has expanded enormously, outstripping our ability to access and manage them efficiently. Recent developments in modelling the appearance of objects in images, have provided a promising framework through which computers can automatically understand the contents of images.

This thesis develops the Active Appearance Model (AAM), one of the more advanced appearance modelling methods, in conjunction with modern statistical classification techniques, to detect objects in image databases. This is motivated by the observation that a large proportion of users' retrieval requests to image databases can be satisfied by the accurate detection of relevant objects in the images.

## 1.1 Motivation

This section explores three problems faced by the Interactive Multimedia Services (IMS) industry. This industry sector is large and growing all over the developed world. It supplies video, speech, music, text and images to users under their own detailed control. Currently IMS includes such applications as public information kiosks, catalogues, encyclopaedias on CD-ROM, and WWW sites with embedded sound, video, etc. Expected in the future are applications like Video on Demand, which connect huge on-line video rental libraries direct to the home.

**1. Managing a large library** at low cost is key to successfully capturing the Video-on-Demand market. There have been several trials of Video-on-Demand by companies around the world, including a 2000 customer commercial trial by BT[77]. One of the predominant findings[77, 164][1] from these trials is the large expense in managing video libraries. Media arrives from distribution companies with, at best, very little labelling, and, at worst, misleading labels. This uncertainty requires a librarian to physically watch each item to verify its content and quality. Managing customer trials, which provide only about 500 hours of video content, was found to cost many times more than expected. In a real service situation, libraries of 10,000 or more hours of content will be required. Hence, it is imperative that costs are reduced, otherwise the service will be unprofitable. This requires transferring the tedious verification and management effort to a computer, thereby making the librarian more productive.

**2. The WWW** is another application area. In modern online text libraries, the ability to search for specific words has proved invaluable in reducing costs and in transferring search effort from librarians to users. In multimedia libraries the equivalent search is for specific objects or actions in images and video. Such a multimedia

[1] also via personal communications from BT project managers

search engine should allow the user to specify video content in a convenient manner. The vast and unorganised library that is the WWW, is a case in point. Here text-based search engines such as AltaVista[5] have given users direct access without having to pay for the services of a librarian. In an increasingly multimedia-enabled WWW, there is a need for a video and image search engine.

**3. Liability** for the classification of content falls on the supplier. Several years ago France Telecom, contracted to supply a television feed to Saudi Arabia, made a *technical error* at their satellite ground station. An adult channel for European viewers replaced the intended programming. The company had its contract cancelled without refund[19]. The loss was not catastrophically expensive for France Telecom, but the risk is clear—viewers must be correctly supplied with the item they request.

### 1.1.1 Object Detection, Why and How?

Whilst there are many approaches to attaching semantic labels to images, object detection is one of the most direct. The existence of an object in an image, is something that we as humans can be unambiguous about. We can also be unambiguous about why we think something seen in an image is or is not the object we are interested in. In particular an object can often be identified without reference to the context. An ordinary photograph of dog, cut out and overlaid on an radiograph, is still identifiable as a dog even though it is out of place. This ease of reasoning is attractive.

In the time leading up to the initiation of this work, the low-level feature based approach provided the vast majority of papers in this problem area. Simple functions of pixel grey-levels, edge positions, etc. were assembled into large feature vectors for classification. As argued in section 2.2.1, this approach appeared to be a dead-end, trapped by the lack of semantic relevance of the low-level features employed.

Generative models of object appearance were appearing in the literature, with im-

pressive claimed abilities [35, 37, 52, 76, 113, 134]. For example, they were supposed to be able to represent all valid appearances of an object, and only valid appearances of that object. Nobody at the time was investigating the use of generative appearance models for searching image databases. BT had sponsored some of the early work on one of these approaches—Active Appearance Models (AAMs), and supported this author to investigate using AAMs for image database retrieval.

### 1.1.2 Object Detection and Computer Vision

The field of object detection in images has evolved with computer vision, and remains mostly unsolved. Equally, the general computer vision problem of automatically providing semantic descriptions of scenes remains unsolved. Marr's early proposal [88] for a solution to the general computer vision problem included a layer to recognise objects from their outlines and internal edge structure. Since then, published attempts at object detection have not been explicitly embedded in a wider attempt to solve the more general scene description problem. It now appears more likely that solutions to the object detection problem will precede a solution to the more general problem.

It is still valuable to consider the object detection problem with respect to general scene description. It allows us to disassociate detection from the closely related problem of segmentation/location. In the segmentation problem, we want to know the location of a known object in an image that is *assumed to contain that object.* In the detection problem, the object may not exist in some images, and may occur in multiple locations in others. However, in order to provide a semantic description of the scene, we would really like to both detect objects, and identify their positions.

## 1.2 Overview

The basic approach taken in this work is to use AAMs as a complex feature detectors, to find potential objects. Then a Support Vector Machine (SVM) is used to make a final decision about whether the object is real or a hallucination.

The outline of the thesis is as follows:

**Chapter 2** reviews previously described approaches to content-based image retrieval and object detection.

The first experimental part of this thesis compares two state-of-the-art classifiers and their application to object detection, for use both as a baseline level of performance and as a suitable classifier in later chapters.

**Chapter 3** reviews existing classification/detection techniques, and investigates Sung and Poggio's GMM network in detail.

**Chapter 4** reviews the SVM classification method, and shows its superiority to GMM networks. An extension to the SVM method allows optimal parameter selection with no manual intervention.

The next part of this thesis describes several improvements to AAMs, which are useful for object detection, but also more generally.

**Chapter 5** describes the standard AAM method.

**Chapter 6** introduces the Texture AAM which replaces the standard intensity texture model, with a description of local structure, such as edge and corner features. This is shown to give greatly improved search performance.

**Chapter 7** compares several additional texture descriptions. Several methods for determining the statistical significance of AAM improvements are compared, leading to the development of a powerful bootstrap approach.

**Chapter 8** examines the use of limiters in AAM search, and shows that the widely used box limits on the model parameters of an AAM are much inferior to elliptically shaped limits.

The final experimental part of this thesis describes using appearance models to search image databases.

**Chapter 9** introduces the combined AAM-SVM and examines its behaviour. The use of the AAM's residuals and model parameters, are compared as feature vectors for the SVM.

**Chapter 10** compares existing object detection techniques with the AAM-SVM showing its superior performance. An analysis of several variations on the AAM-SVM leads to a multi-stage method with very large improvements in speed.

After a discussion of the work in this thesis, some significant contributions to a publicly available computer vision software package are detailed in an appendix.

**Chapter 11** contains a general discussion of the work presented in this thesis.

# Chapter 2

# Literature Review

This review seeks to place object detection as one of three broad approaches to searching image databases. Two of the approaches, based on metadata and on low-level features are examined first. A range of object detection methods is then examined in detail.

## 2.1 Metadata Based Searching

It is perfectly possible to search an image database without looking at any pixels, but instead using external data about the images, i.e. metadata. Metadata-based searching can involve hand-labelling the images with useful information. Searching is then performed using standard database and text-based search techniques of the metadata. This is the approach taken by commercial picture libraries (e.g. Corbis[43].) It works because the re-use value of their stock is high, so the human effort in labelling a picture (e.g. 30 minutes per image) can be commercially justified.

There are a few variations on this method. One of the most relevant is that of searching for images in large multimedia documents such as web pages by performing a search on the text parts of the documents which would appear near, or refer to, the

image. AltaVista[4] were the first (in 1999) to provide a WWW-scale search engine based on this method. This and the human-labelled metadata techniques have the advantage that they generalise readily to search for other types of media such as sound fragments. Where these techniques fail is with large databases where each item is of low value, and where there is poor or no labelling. It is not hard to see that any labelling scheme will fail to cover every possible interpretation that a searcher might place on it.

Although it is a strictly content-based approach, Sivic and Zisserman's Video-google technique is relevant here because it uses many of the techniques developed for free-text database search. By treating vector quantised local region descriptions as words, they were able to apply techniques such as common word suppression, dot product distance, relevancy and saliency weighting, and inverted file indexing.

Another variation used in the Four-Eyes system[94], starts off with a poorly labelled database. A "Society of [texture] Models" is used to produce many overlapping indices into the various textures within the database images. As users make their queries and retrieve their images, they are asked to describe what they are searching for and indicate which pictures do or do not satisfy their query. The system continually learns which photos, textures, and models satisfy which type of queries from users during use of the system. However, in unpublished observations, Minka has suggested that the system learns to ignore most of the texture-model driven labelling in the system, whilst building a direct mapping from query to image.

There are also techniques which combine metadata- and content-based techniques, where some (e.g. MPEG-7[89]) or all of the metadata is generated automatically. However, this is mainly done for speed of searching. For the purposes of this thesis, there is not a big difference between searching automatically-generated metadata, and searching the image content itself. Rather, the divide is whether a human was involved in explicitly adding semantic labels to all of the items in the database.

## 2.2 Low-Level Feature-Based Search

Along with object detection, low-level feature based search is often referred to as Content Based Image Retrieval (CBIR). The extreme end of the content-based approach is searching by example. Here the user presents the search engine with an example picture, and wants images similar to it. This approach requires some sort of signal processing to decide what constitutes a similar picture. Another example of a content-based approach is when the user gives a text and logic based query which is then used to control a signal-processing based search.

The first attempts at CBIR produced low-level image feature based systems. These pick some easy-to-calculate low-level feature, such as a colour histogram, and define a distance metric for the feature, e.g. Euclidean distance. The feature is calculated for every image in the database, and the images with the closest feature to the query are returned to the user. Swain and Ballard's seminal paper[146] on colour histograms was the first to pose the problem in terms of image retrieval.

Such systems can get quite complex, involving several features, and positional constraints. The best known example of these systems is IBM's Query By Image Content (QBIC) system[104] which retrieves images that match a user drawn sketch. The system can be impressively demonstrated by drawing a pink circle on a green background, upon which the system will retrieve pictures of roses. Given other semantic classes, however, it is hard to devise a suitable sketch.

One problem with the low-level image feature approach is that the meaning of an object is often independent of its location in the image. So systems which tie a particular low-level feature response to a particular image location will never be very successful. One way to avoid this problem is to ignore positional information altogether. Recognition of objects from a database solely by their colour histogram characteristics can have very high tolerance of viewing angle[146], and with modification, a high tolerance to varying lighting conditions[59].

There are a whole range of features that can be used. Adini *et al.*[1] describe a range of grey-scale filters for use in the face recognition domain. Viola[160] compares a set of Gabor-filter like features over various object recognition tasks. Rui *et al.*[124] and SmeuldersPAMI2000 *et al.*[139] provide broad reviews of the low-level feature approach to CBIR.

### 2.2.1 Problems with the Low-Level Feature Approach

Before 1999 papers using the low-level feature approach suffered from a number of problems. Many papers (e.g. [104, 128, 131, 140]) do not attempt to measure performance quantitatively, instead just showing some example results. Where qualitative results were given, a limited size of test database was often used, with little evidence of unbiased selection. For example, many papers (e.g. [123, 150]) used parts of the Corel Stock Photo Collection. From looking at the image collection, it is unclear whether these author's methods recognised the objects of interest, or rather the background colour or texture which was common to many of the subject classifications chosen by Corel. For example, Rubner and Tomasi[123] used Earth Mover's Distance (EMD) to match Gabor-like filter responses between unsegmented images of animals. They used a selection of 500 images, all containing animals, from the Corel Collection, and achieved high precision when trying to retrieve images of zebras and big cats. Their paper does not mention any of the other animals they used, so this author shall unfairly pick two that make the point. If their collection also contained polar bears, all which are on a flat white background, or deer, which usually include a flat blue sky background, then just distinguishing between these flat textures and the cat's textured savannah background would be easy. Thus any claimed precision (with respect to the animal class) is suspect.

There is a fundamental argument against this whole approach. By definition, low-level features are not semantically meaningful, and no simple distance metric is likely to add semantic meaning. If more complicated distance metrics such as statistical

classifiers are used, there is no reason to expect the classification regions of feature space to be compact enough for it to be feasible to train the classifier.

In order to use these approaches for truly recognising objects within an image, rather than the whole image, it would be first necessary to segment the objects from the scene. Automatic segmentation methods are far from perfect[2]—indeed, using the same argument as above, it seems that automatic segmentation is only likely to work well when it is based on a knowledge of the properties of the objects in the scene. This implies that object detection needs to precede feature extraction—which is circular.

### 2.2.2 Recent Results with Low-Level Features

Whilst there was some confidence that these problems would be solved (notably amongst some of the MPEG-7 participants[89],) prior to 1999 there was no practical result, or theoretical argument to support this view. More recently however, there have been several papers on the low-level approach which show promising levels of performance.

Chapelle *et al.*[24] used a Support Vector Machine (SVM) to classify 4096-bin colour histograms. They classified 2670 images into airplanes, birds, boats, buildings, fish, people, and vehicles, with accuracy rates of about 85%. Tieu and Viola[149] used AdaBoost[130] to classify 46,875 dimensional feature vectors of simple wavelet-like filter responses, showing reasonable recall and precision rates when asked to distinguish between five types of outdoor scenery. Vailaya *et al.*[155] used Bayesian classification with Gaussian mixture models to classify features with hundreds of dimensions. The features consisted of edge direction histograms, and first and second colour moments in each of a hundred image sub-blocks. For decisions such as city v. landscape, or forest v. mountain, they obtained approximately 96% accuracy. Koubaroulis *et al.*[80] took sparse features of local colour variation, and by carefully modelling the conditional distributions of those features with discriminative power, were able to classify random shots from video coverage of the Olympics into 5 categories with

85% accuracy.

MPEG-7 chose to include quite a few low-level feature descriptors[89], including colour histograms, colour moments, Gabor filter responses, and edge histograms. To further widen the user's choice, the features descriptors can be refined, for example, by measuring them over a rectangular grid, or by describing their time-series variation for videos.

## 2.3 Object Detection

High-level feature detection attempts to directly identify semantically relevant parts of the image. This thesis treats an object detector as the prototypical, high-level, semantic, feature detector.

In a small study[55] on the requests made to a commercial photo library, Enser found that 48% were about identifiable objects without any reference to context—the domain of object detection.

There are many ways of detecting objects in images, and this overview groups most of them according to whether the object model is based on a single training image, multiple images, or a statistical distribution. Special distinction is also made of shape- and parts-based approaches.

### 2.3.1 Single Image Matching

In this approach, a single training image is used to represent a class. During use, the matching algorithm compares each new image to the (possibly processed) training image.

In Lades *et al.*[81] a grid was laid over the training image, and a Gabor filter jet was measured at each grid point. During use, the grid was then placed over each

new image, and the algorithm attempted to minimise a distance metric based on the elastic distortion of the grid, and a difference in the Gabor jets. When tested on an object recognition task with a small database (∼100 images,) it could perfectly reject all negative test images, while correctly identifying 93% of positive test images.

Sali and Ullman[125] calculated a correspondence between the training image and the new test image, as in stereo vision. Their matching algorithm was then able to decide if such a correspondence made geometric sense. This method assumes that the object is rigid, and that it is possible to segment out the image patch of interest in order to calculate the correspondence.

A much older attempt by Bajcsy and Kovacic[8] at corresponding two images, rather than objects, is notable because it introduced the idea of multi-resolution matching. In this approach a rough elastic match from model to image is made at a very low resolution. The model parameters are then transferred to a higher resolution model, which then refines the match. This process is repeated until the model has reached the resolution of the image. This vastly increases the speed compared to searching only at the higher resolution. It also reduces the number of false minima to which the higher resolution models are prey.

There are also many examples in the literature which are effectively variations on the theme of single image matching where there are multiple training views of a given object[102, 10, 131, 132, 103] but with these attempts as before, the system is trying to recognise identical (or at least very similar) instances of the object under different lighting, or viewing conditions. These approaches make no attempt to deal with recognising a class of objects, as opposed to a particular instance.

### 2.3.2 Multiple Image Matching

With multiple image matching, each class of objects is represented by one data structure, that combines several training images. Wiskott *et al.*[169] extended the single

object work of Lades *et al.*[81] by combining the Gabor jets from all the training images for a class, into a list of jets at each grid point. Their matching process picked the best jet at each grid point as it attempted to minimise the distance.

Gavrilla and Philomin[60] used a hierarchical scheme to search car-mounted camera images for a small number of objects. In this method, each plausible combination of edge detections is examined by the root node in the hierarchy. At each level in the hierarchy at least half the remaining plausible models are discarded. Once a leaf node is reached, a final match/non-match decision is taken.

### 2.3.3 Statistical Image Matching

Statistical image matching builds a statistical model for each class of objects. Although a refinement of the multiple image method above, the explicit modelling of a potentially infinite population from a finite training set is very powerful. Here the archetypal object is the human face, and face detection papers probably outnumber all other statistical object detection papers. The basic approach is simple—the pixel intensities from a compact image patch are arranged into a feature vector which is then modelled probabilistically. For any tested patch, a decision is taken about whether the image patch fits the model. Often the feature vector is first normalised (e.g. plane and histogram normalisation[108]) to partially remove the effects of lighting variation.

The seminal paper in the field by Sirovich and Kirby[138], showed how to compactly represent the statistical distribution of face images, by using Principal Component Analysis (PCA)—a method that came to be known as Eigenfaces. Variations on this theme include using Linear Discriminant Analysis (LDA) to optimise the linear subspace for discrimination rather than reconstruction[12], non-linear principal component analysis[96, 119], and modelling the face region with piecewise PCA[99].

One problem with PCA is the indistinct nature of some of the principal components,

where a set of sharp edges in the training set is represented by the superposition of a set of blurred components. This requires a large amount of information to be stored in the model, a bad indication in itself[1], and does not provide the best reconstruction of unseen images. Lanitis *et al.*[82] solved this problem by warping the images so that corresponding edges coincided. Using their approach, the representation of the image texture does not need to represent the variation in edge position. Their method merges the parameterisation of the warping and of the image texture by using PCA to identify and model the correlations between the two. A non-probabilistic alternative involves using linear combinations of the training images, rather than of the principal components. Jones *et al.*[159, 76] produced a similar linear model of warping and texture representation, but used linear combinations of the training images, rather than of the principal components, and a dense warp field induced by an optical flow analysis.

Rather than use pixel intensities, from an image patch, it is possible to use a simple texture filter to provide better feature vectors, more amenable to classification. Rikert *et al.*[113] used multi-scale wavelet filters to produce an over-complete feature vector, which they then classified using a Gaussian mixture model and Bayesian classification. They reported good results on both face and car finding.

The recent introduction of powerful statistical classification techniques has revolutionised face detection. These classifiers require a training set of positive and negative examples, i.e. real faces, and image patches that do not contain faces, respectively. To get good accuracy, these methods require a very large collection of negative examples to help refine the classification boundary. Most recent works acquire these negative samples using an iterative refinement process introduced by Sung and Poggio[145]. They trained an initial classifier with a small set of randomly chosen negative examples, and then scanned a set of face-less images to find negative examples that are necessary to refine the boundary.

[1]By appealing to Occam's Razor, a simpler model should be preferable to a more complex one if they both model the data equally well. Kolmogorov complexity[84] and Minimum Description Length (MDL)[115] offer more rigorous treatments of this idea.

Neural networks were the first of the modern general-purpose classifiers. Recently, heavily specialised and tuned neural network methods have given excellent results. Rowley and Kanade[122] used multiple voting Multi-Layer Perceptrons (MLPs) to achieve excellent results. Sung and Poggio[145] used two Gaussian Mixture Models (GMM)s to model the positive, and the confounding negative distributions, and used a small neural network to arbitrate between them. These two papers also introduced the `CMU` face detection test database which is widely used in face detection experiments.

Osuna *et al.*[108] were the first to use a Support Vector Machine (SVM) as the classifier, gaining significant advantages over neural network users in terms of much reduced tuning requirements, and consequently easier training. The computational complexity of using an SVM classifier was hugely cut by Romdhani *et al.*[121] through use of a cascade of reduced-set SVM classifiers, to reject non-faces early. They also avoided the cost of intensity normalisation by repeating examples from the positive training set with random changes in contrast and lighting direction.

Viola and Jones[161] introduced what is widely regarded as the best face detection method currently available, in terms of its detection/speed performance tradeoff. By first calculating the running sum of pixel intensities across the image, their method can very cheaply calculate the responses to small rectangular filters similar to Haar wavelets. It then trains a classifier using a cascaded version of AdaBoost, which rejects most non-faces very quickly. Whilst other methods may be slightly more accurate, Viola and Jones' method is many times faster.

### 2.3.4 Shape-Based Approaches

One approach to dealing with objects in images, whilst reducing computational complexity, is to use only their outlines' edge information.

There are some image recognition problems where outlines are readily available, for

instance when comparing a test image to a large database of trademark images[49]. As in the case of low-level feature based searching (section 2.2) there are many schemes described in the literature, perhaps because the approach is computationally accessible. Histogram methods stand out because of their theoretically justifiable $\chi^2$-based distance metric. Thacker *et al.*'s pairwise geometric histograms[148] measured the distribution of distances and angles between line segments on a shape, in order to find multiple shapes in an image, despite overlapping and some clutter. Mori *et al.*[101] built log-polar histograms between randomly sampled points on the shapes, and showed that there was enough information in the histograms such that when trying to match a new shape to one of a large database of objects, most non-matches could be discarded early after checking the histograms of only a few points on the shape.

MPEG-7 performed extensive testing on many shape descriptors, before choosing two 2D descriptions[17] of shape for their standard. The Angular Radial Transform decomposes shapes into a set of Fourier-like basis-functions on a polar co-ordinate system, and then truncates and quantises the responses to produce a 140 bit descriptor. This approach can be thought of as a rotation-invariant lossy compression of binary images, and is robust to topology changes and speckle noise. For simple closed contours, MPEG-7 uses a scale-space approach—as the contour is successively smoothed, the position and scale of the disappearing concave sections is recorded. The resulting convex contour's eccentricity and circularity are added, and the whole representation quantised to produce a descriptor of typically 112 bits. The description, along with a suitable distance metric, provides shape matches similar to human expectations. However, in an approach common to international technical standards, MPEG-7 does not define how the contours are extracted or compared.

It should be noted that the shape-based approach is often used in conjunction with other approaches described earlier. There are several statistical approaches to shape, such as Cootes *et al.*'s Active Shape Models (ASMs)[36], and the related Active Contours of Blake and Isard[15]. Both of these methods learn a subspace model

(known as a Point Distribution Model (PDM)[31]) of the variation of the position of control points of a spline that represents the shape outline. They can automatically fit to an object in a image, by iteratively searching for strong edges along profiles perpendicular to the shape outline, finding the best fit of the PDM to these edges, and moving the spline to the best fit.

There remains a fundamental problem with shape in that it ignores the majority of the information in each image. For this reason, no purely shape-based method has demonstrated good results on a detection task on textured objects like faces.

### 2.3.5 Parts-Based Representations

Representing whole objects by parts has a strong appeal. The idea is to have an object detector respond when some or all of its constituent part detectors fire[2]. A significant problem with the parts based approach can be seen in two extremes. At one end, there is no modelling of the relationship between the parts. Here the detector is very prone to hallucinate an object, because its component detectors are triggered on many false positives. At the other extreme, a computational combinatorial explosion occurs when every possible configuration (over all the desired invariants) is explicitly listed.

Sali and Ullman[126] attempted to tackle this problem explicitly. Their detectors consisted of image patches taken from the training set (in the multiple-image recognition fashion above,) and a distance metric that compared intensity ranks and gradients. The patches, which were manually chosen with varying sizes, overlapped each other. Enforcing this overlapping between the smaller and larger patch detectors, and the use of spatial voting (feature-based Hough Transform,) helped to avoid the combinatorial explosion.

Mel and Fiser[92] suggested that the patches (or at least the distribution of patch

[2]Such an approach is supported by neurophysiological evidence[162].

sizes) could be chosen optimally, such that a small number of large detectors serve to bind together a much larger number of loosely coupled small detectors. However, they only demonstrated this method in the text search domain.

Non-negative matrix factorisation[83] is another approach to parts based representation. This method decomposes a set of training images into a basis set composed of parts of the object. This author's attempts to reproduce these results, however, failed to find that basis set was actually localised.

One means of improving the parts-based approach is to discover feature extractors that are invariant to lighting, distortion, orientation, and scale. Lowe[86] took several feature extractors that do not have such invariances, and either enforced invariance (in the case of lighting and orientation) by normalisation, or sparsified the feature map to record only those locations where the feature detectors were stable (in the case of scale.) Ohba and Ikeuchi[106] used a PCA analysis of optimally selected windows from the training set, to provide a dictionary against which windows from a test image could be matched. Both papers suggest using spatial voting to combine the detector hits and decide if an object is present. Obdržálek *et al.*[105] developed a detector that provides a local affine frame for invariant comparison of raw pixel intensities, demonstrating excellent performance in an object recognition task.

A more sophisticated version of this approach, and possibly the most interesting of all the methods considered, is due to Weber *et al.*[167] Their "constellation" model is a sparse, generative, model of appearance. A set of positive training images is scanned using an interest point detector, the small neighbourhoods of which are approximated using vector quantisation. Those features which are found throughout the training set become templates. In this way they discover objects without having to do any marking up. As in an earlier work[22], the probabilities of each template—matching a positive training image, failing to match a positive training image (i.e. being occluded,) or matching a false positive—are estimated. The distribution of relative positions of each template in the training set is modelled using a PDM.

During a normal search these templates are used as non-maximally suppressed feature detectors. The PDM's probability estimate of the shape of each combination of template responses, the probability of occlusion for any missing templates, and the probability of a template's false-positive response, are integrated over all the possible matches to give a single probability of the image containing the object. Although this removes the ability to estimate the number of objects in an image, it improves the quality of the decision about whether an image contains at least one object.

In a development by Fergus *et al.*[58], the use of fixed templates was replaced by eigenpatches and a joint probabilistic appearance model over texture and shape. By also non-maximally suppressing the initial interest detectors in the scale direction, they were able to model the scale, and make the detectors invariant to it. In a further development by Fei-Fei *et al.*[57] they note that humans can learn a new object class from only a few examples. They suggest that this is because, having learnt models for thousands of other object classes, we have a prior on the parameters of any object-class model, and that we use this prior to constrain the estimate of the new object-class model. Their technique attempts to replicate this feat by first building a model of several object classes with a large number of examples, using mixture-models to represent the (now) multi-modal distributions. The covariance of the parameters of these mixture-models is also estimated. Up to this point, the process takes several hours. Once this previously learnt mixture model is established as a prior, a constellation model for a new object class can be learnt from a small (1-5) number of training images, in less than a minute.

## 2.4 DBMSs and Complete Systems

In addition to the content analysis algorithms, CBIR also requires a DataBase Management System (DBMS). On the DBMS side there seems to be little agreement on suitable database structures. Almost every academic paper on the subject proposes a new scheme[64, 3]. One concerted effort to come to agreement, was the international

standardisation process MPEG-7[89]. However, the MPEG-7 standard only defines interface protocols necessary for inter-operability and thus avoids specifying almost anything about the DBMS itself.

Complete, publicly-available CBIR systems, are fairly rare. One well-known academic system is Photobook[97] from the MIT Media Lab. Complete systems are more evident in the commercial sector. QBIC[104] from IBM was the first system to achieve any serious commercial recognition. Other companies include Virage[6], and MATE[90] who once claimed to be able to detect and classify many high-level objects and events, e.g. faces, explosions, and sky, but have moved out of the IMS market since 1999.

## 2.5 Test Databases and Evaluation

Papers in the computer vision field are prone to providing less than rigourous evaluation of proposed techniques. This is due to many factors, including the application specificity of any possible test. Another common factor is the expense of generating a properly labelled and representative database. In this field, copyright can also become a difficulty. Creating a representative database means copying images that are found on real websites, magazines, television channels, etc. All of these pictures are implicitly and often explicitly copyrighted in a manner that includes (possibly unintentionally) banning their use in test databases.

The most comprehensive evaluations have been carried out by organisations wanting to make an objective decision about purchasing or investing in a technology. For example the FERET test[109] evaluated several competing face recognition systems, for the American military in order that they could decide whether to purchase a Face Recognition system for physical access control. More recently, as governments have been looking to use image-based biometrics, they have been funding academic comparative work, e.g. BANCA[7] and U-FACE[33].

A list of freely available databases is maintained by Carnegie-Mellon University (CMU) at the Computer Vision Homepage[71]. The typical database is either constrained to pictures taken for a specific application, or on a limited range of environments. For example, FERET has mugshots of faces with plain black backgrounds, and reasonably consistent scaling, lighting conditions, and other camera and environmental factors. The `XM2VTS`[93], collected by Messer *et al.* is public database with high-quality images of "clean" faces—see figure B.1 in the appendix for some examples. Whilst not ideal for use in evaluating face detection performance, ISBE does have high-quality labelling and markup for `XM2VTS` which makes it useful for testing. One exception to the cleanliness of many existing test databases is the `CMU` face database collected by Rowley *et al.*[122, 145] This contains images from a wide variety of sources, with faces of varying cleanliness and other attributes, and highly textured and variable backgrounds. Some examples are shown in figure B.2 in the appendix.

## 2.6 Discussion

The above review separated previous work into two broad categories. But, is there any merit in the distinction between the advanced classification of low-level features (e.g. Tieu and Viola's AdaBoost on thousands of primitive image features [149]) and the relatively simple classification of complex features (e.g. Weber *et al.*'s constellation model of sparse appearance of objects[167]?) The boundary between feature extraction and classification is somewhat arbitrary, and both build models that claim to have semantic meaning. In the past, there has been more of an identification of the concept of *object* in the approaches using complex features. In future however, it may be more fruitful to see a convergence of these two approaches, as the known limitations of each are addressed. The work in this thesis can be seen in that light—using a high-level feature detector and modern statistical classification together, to achieve superior performance.

# Chapter 3

# Building GMM Network Classifiers

At the start of the work of this thesis, Sung and Poggio's GMM-based neural network face detector was the state of the art. It was implemented to use as a baseline against which to assess new developments, and as a classifier for use in combination with appearance-model based feature extraction.

## 3.1 Background

In order for any method to detect an object in an image, there must be a classifier to decide whether or not the image (or part of the image) contains the object. In contrast to most classification tasks, reliably detecting objects in images over a wide variety of positions and scales requires extremely high performance. False positive rates (the fraction of negative test examples that are incorrectly classified) in the region of one in a million are required in order to reject the vast majority of image patches that do not contain the desired object. Fortunately, *true* positive rates of nine out of ten are acceptable. Recently, classification methods have been developed which are capable of achieving this sort of performance. The computer vision field, and in particular the face detection problem, has been a strong contributor to the

development of these techniques.

Whilst later chapters will use more complicated appearance models, this chapter will represent the appearance of an object as a simple image patch. The rigid shape of the patch is chosen to fit the object, e.g. face images have a vaguely oval shape. Within the patch, the vector of grey level values sampled on a fixed pixel grid *is* the model of appearance. The position of the patch can be moved around to sample the appearance anywhere in the image. It can be rotated, expanded and shrunk. The patch elements are sampled from the underlying image, with appropriate interpolation when the image's and the patch's pixel grids do not align. In order to detect faces, the patch is sampled everywhere on a dense grid of positions and scales, and the vector of pixel values is passed to the detector's classifier.

Early work by Pentland *et al.*[154] implicitly (and later explictly[98]) classified face from non-face patches by using a multivariate Gaussian Probability Density Function (PDF). If the probability density of a particular patch was above a learnt threshold, then the patch was a face. The patch was sampled at all positions and all scales. However, the hyper-elliptical classification boundary implied by this scheme is not very accurate. Several variations[69, 38, 99] have been suggested on a theme of using mixture models to represent the PDF of an appearance model. The advantage of this approach is that it is possible to remain explicitly in the probability domain and thus be able to reason convincingly about the behaviour of these systems. The main disadvantage of the approach is that it does not learn anything about the PDF of the appearance model on non-face patches. The face/non-face threshold is set at some fixed probability density, incorrectly assuming a flat PDF for the appearance of background.

Neural networks, on the other hand, can explicitly model a decision boundary. Rowley *et al.*[122] achieved very good face detection performance using a neural network with a complicated structure. Unfortunately Perceptron-style neural networks have many problems, making it hard to treat them as anything other than black boxes. In

particular it is difficult to distinguish any particular problem's cause such as training issues, incorrectly structured networks, or poor separation of the data. Selection of model parameters is also difficult. For example with back-propagation training of Multi-Layer Perceptrons (MLPs), the learning rate and number of iterations can greatly affect the performance of the final classifier[114, p.155].

In Sung and Poggio's[145] work using GMM-based neural networks (henceforth "GMM networks") to perform the face detection task, their model is structurally much simpler than the Perceptron-style neural network developed by Rowley *et al.* The GMM network had the highest reported performance in the face detection task at the start of the work of this thesis (jointly with Rowley *et al.*'s neural network.) For these reasons, GMM networks were selected for initial investigation, which are described later in this chapter.

The GMM network uses one GMM to model the distribution of the appearance of face (or "positive") patches. An initial GMM PDF of the appearance of background is built from a small selection of non-face (or "negative") patches. The GMMs are trained using classification Expectation Maximisation (EM)[23], which assigns each training example as a whole to the nearest cluster, rather than the weighted assignment used by Dempster's original EM[48] method. Also, rather than using Bayes's rule, the final face/non-face decision is made by a small MLP. After initial training, a series of "refinement iterations" take place. In each iteration, the detector is run over a database of images known to be devoid of faces. Any hits are added to the training set, and the negative GMM and the MLP are retrained.

The iterative refinement process is very powerful, and is applicable anywhere it is easy to create a database containing just negative examples, and where the prior probability of a positive example in the test set is very low. First introduced into this field by Sung and Poggio with their work on GMM networks, it has been taken up by other designers of face detectors[108, 121]. It is also used heavily in this thesis.

During the period of the work described in this thesis, Vapnik's Support Vector

Machine (SVM)[156] has become extremely popular, replacing the MLP as the *black box* classification technique of choice for many problems. In the simple linear case, the SVM puts the hyperplane boundary equidistant between the closest negative and positive training examples. Unlike neural networks, the SVM's training algorithm has no false minima, and minimises the expected error on unseen test data rather than the training set. Osuna *et al.*[108] were the first to use SVMs for face detection. Further details of the SVM can be found in the next chapter, where they are shown to be superior in several ways to GMM networks.

More recently, boosted methods that optimally combine many weak classifiers into a single very high performance one, have been shown to be useful. The cascaded AdaBoost classifier developed by Viola and Jones[161] can accurately reject a large fraction of the background patches extremely quickly using a very simple classifier. The remainder of the background image patches are rejected by ever more complicated but precise classifiers, until only real faces remain.

## 3.2 Implementing GMM Networks

Sung and Poggio's algorithm for training a GMM network is shown in detail in figure 3.1. Since we were unable to acquire Sung and Poggio's software, the algorithm was implemented from scratch. In doing so, several significant implementation issues had to be addressed.

**Collecting False Positive Examples** It is difficult to handle and store the variable, and sometimes huge, number of false positives, found during each of the refinement iterations. Even after rewriting the software so that the vectors were stored on disk, there were some occasions when the available 3GB of disk space was not enough. At every iteration, only a random subset (size $n_{\text{select}}$) of all the false positives needs to be added to the training dataset. Storing all the false positives, only to select a

- Build a sampler that can acquire 283 pixel values spaced on a compact grid, from an image. The grid can be placed over the image with any desired transform. In particular, if the grid is zoomed so that the grid points are spaced more widely than the pixels in the image, the image is appropriately smoothed.
- Given the set of training images, and a list of the positions of every face in them, sample in the marked positions. Also sample with slight rotations from the marked positions, and with mirroring, to increase the size of the training set.
- Planar normalise each sample patch. i.e. find the best fit of the equation $I_{\text{planar}}(x, y) = \alpha(x) + \beta(y) + \gamma$ to the sample patch $I_{\text{sample}}(x, y)$, and then remove this planar component, $I_{\text{sample}}(x, y) := I_{\text{sample}}(x, y) - I_{\text{planar}}(x, y)$.
- Histogram normalise each sample patch, and convert them into vectors. This list of vectors is referred to as the "positive (training) set."
- Find a small set of non-face image patches and convert them as above into a list of vectors, called the "negative (training) set."
- Repeat the following iterative refinement steps until a suitably performing classifier is obtained:
  - Train two GMMs, one each on the positive and negative training sets.
    - Perform K-Means with K=6, on the training set.
    - Use the six clusters to create six Gaussians. Perform PCA on the each Gaussian, and rebuild them with only 75 principal components. Turn the six Gaussians into a mixture model.
    - Repeat:
      - Perform `expectations-step` for each training vector. i.e. Find the Mahalanobis distance from each Gaussian centre to each data point
      - Perform `classification-step`. i.e. Assign each point to the cluster with the lowest Mahalanobis distance
      - Perform `mean-maximisation-step`. i.e. Update Gaussian means with cluster means
      - If there have been no changes, or if there have been five consecutive `mean-maximisation-steps`, then perform `full-maximisation-step`, i.e. Update Gaussian means and covariance with cluster statistics. Perform PCA on each Gaussian as before.
    - Until no more changes with Full-Maximisation Step.
  - For each data point in the positive and negative training sets, fill a 24-element distance vector with the Mahalanobis distance from each Gaussian centre, and with the Euclidean distance in each Gaussian's Complementary Space from the centres.
  - Train a 24 input, 24 hidden units, 1 output, 3 layer, MLP using simple back-propagation.
  - Using the sampler, and GMM-network classifier, scan a database of non-face images extracting every possible image patch at a range of scales (but only one orientation.) Store those false-positives returned by the classifier, and add them to the negative training set.
- Done.

**Figure 3.1:** Summary of Sung and Poggio's algorithm[145] for training a GMM-based object detector

few is very expensive. Instead, a `stochastic-array` module was developed which stores just a random selection of all the examples presented. Initially every example is stored until the array contains $n_{\text{select}}$ examples. Any further examples are then accepted with probability

$$p(\text{selecting next example}) = \frac{n_{\text{select}}}{i} \cdot \frac{n_{\text{select}}}{n_{\text{select}} + 1}$$

where $i$ is the number of samples presented so far. Selected examples overwrite one of the existing $n_{\text{select}}$ samples current stored in the array. At any time `stochastic-array` can be shown to have an equal probability of containing any of the examples presented so far. Initially $n_{\text{select}}$ was set to 5000.

**Selecting the Number of Principal Components** Sung and Poggio described a means of picking the number of principal components for each mixture model thus:

> 'We arrived at our choice of 75 "significant" eigenvectors per cluster using the following criterion: For each cluster, eliminate as many trailing eigenvectors as possible, so that the sum of all the eliminated eigenvectors is still smaller than the cluster's largest eigenvalue. This procedure leaves us with approximately 75 eigenvectors for each cluster, which we standardise at 75 for simplicity.' [145]

None of the reasoning behind this method was given, and when performing the same test, only 25 components were found.

In a mistaken attempt to improve the performance of the algorithm, the above rule was used during each `full-maximisation-step` to dynamically choose the number of principle components. This modified algorithm either broke when one of the Gaussians was starved of any examples, or failed to converge. This failure can be explained by Dempster *et al.*'s proof of convergence[48] of the expectation maximisation algorithm. The proof requires that the change in log likelihood during the maximisation step be continuous with respect to the model parameters. Since the modification

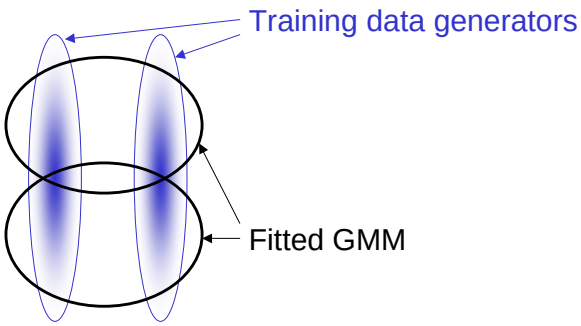
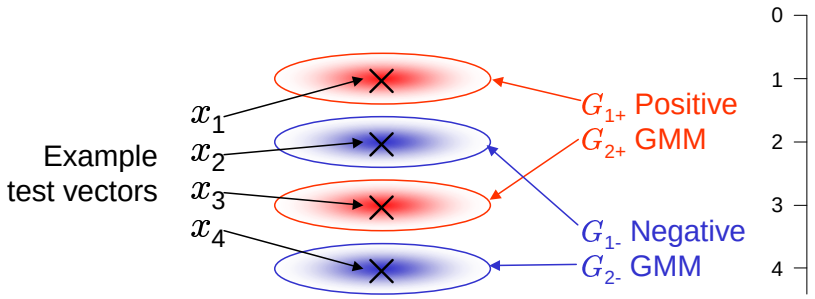



**Figure 3.2:** Non-global minimum consistently found by the GMM training algorithm, when the actual distribution of training data has a strong variance in a different direction to the cluster separations.

to the algorithm makes a change to the number of principle components, the log likelihood change will necessarily be discontinuous at that point.

**Termination of Classifier Refinement** Sung and Poggio suggest terminating the refinement iterations after having found about ten times as many negative examples as positive ones. During the two initial experiments described below, the algorithm ran out of false positives from the training set before reaching this limit. In order to confirm that this was not just a problem with too small a negative database, some of these experiments were restarted with a completely new negative database. Invariably, the experiments ran for at most one more iteration before running out of false positives again.

**GMM Initialisation** While testing the GMM training algorithm on an artificial 4000 vector, 20-D dataset, the EM method was found under some circumstances to consistently fall into a local minimum—see figure 3.2. This was despite random initialisation of the cluster centres, for the first K-Means clustering step. The solution was to temporarily whiten the data during the K-Means clustering step. This allowed the K-Means step to ignore directions that had large variances, when those variances were not caused by cluster separations. It could then correctly identify the cluster centres, which left the EM algorithm in the region of the best minimum.




| Test vector | Distances, $d$, from $G_{1+}$ | $G_{1-}$ | $G_{2+}$ | $G_{2-}$ | $f_{\text{linear}} = d_{G_{1+}}$ $+d_{G_{2+}} - d_{G_{1-}} - d_{G_{2-}}$ | $f_{\text{reciprocal}} = d_{G_{1+}}^{-1}$ $+d_{G_{2+}}^{-1} - d_{G_{1-}}^{-1} - d_{G_{2-}}^{-1}$ |
|---|---|---|---|---|---|---|
| $x_1$ | 0 | 1 | 4 | 9 | -6 $\not>$ 0—Wrong | $+\infty > 0$—Correct |
| $x_2$ | 1 | 0 | 1 | 4 | -2 $<$ 0—Correct | $-\infty < 0$—Correct |
| $x_3$ | 4 | 1 | 0 | 1 | $+2 > 0$—Correct | $+\infty > 0$—Correct |
| $x_4$ | 9 | 4 | 1 | 0 | $+6 \not< 0$—Wrong | $-\infty < 0$—Correct |

**Figure 3.3:** A linear function (e.g. $f_{\text{linear}}$) of Mahalanobis distances will not produce a working classifier. The sum of the reciprocals ($f_{\text{reciprocal}}$) of the Mahalanobis distances will work.

**Output to Final MLP** Sung and Poggio state that they fill the 24-element vector (that represents the output of the GMM for any example) with Mahalanobis distances. Whilst testing the scheme on some artificial datasets, it became clear that the reciprocal of the Mahalanobis distance was superior. Figure 3.3 shows that simply summing the Mahalanobis distances from each cluster will not produce a classifier capable of correctly identifying examples at each cluster's centre, whereas summing the reciprocal of the Mahalanobis distances will produce a viable classifier. Any monotonically decreasing function can be used instead of the reciprocal. The log likelihood of the distance is a natural function to use in this case, and it avoids the infinities created by the reciprocal.

## 3.3 Initial Experiments and Results

A GMM network classifier was trained using 4398 positive examples taken from the 507 individual faces in the images of the `CMU`[122, 145] database (see figure B.2 in the appendix) and 226 faces from the rotated face database published with the `CMU` database. The database of non-face (negative) images was assembled from a random selection of images taken from the WWW, and consisted of 20 images totalling $\sim 6.5 \times 10^6$ patches. The GMMs in the classifier were built with 25 principal components. As the classifier was built and refined, it was tested on the 144-face `webimagesB` dataset (see figure B.5) and 28 non-face images, all obtained randomly from the WWW. The test set's characteristics were indistinguishable from the training set. Testing on the positive database was performed at the just 27 patch positions surrounding the closest fit of the sampling grid to the labelled eyes. If any of the 27 patches was classified positive, that was considered a hit. To save time, testing on the negative database was restricted to approximately 6% of the patches, selected randomly from the 28 non-face images. This amounted to $\sim 8.9 \times 10^5$ patches.

### 3.3.1 Results

The classification results are summarised in figure 3.4. Figure 3.5 shows the Receiver Operating Characteristic (ROC) curves for the initial classifier before refinement, the final classifier, and the classifier after 7 refinement iterations (which performed better than the final one.)

The same experiments were then repeated using GMMs built with 75 principal components as suggested by Sung and Poggio. The results are shown in figures 3.6 and 3.7.

In neither experiment does the performance approach that reported by Sung and Poggio[145]. They claim 80% of faces detected, and 22 false positives, or a false

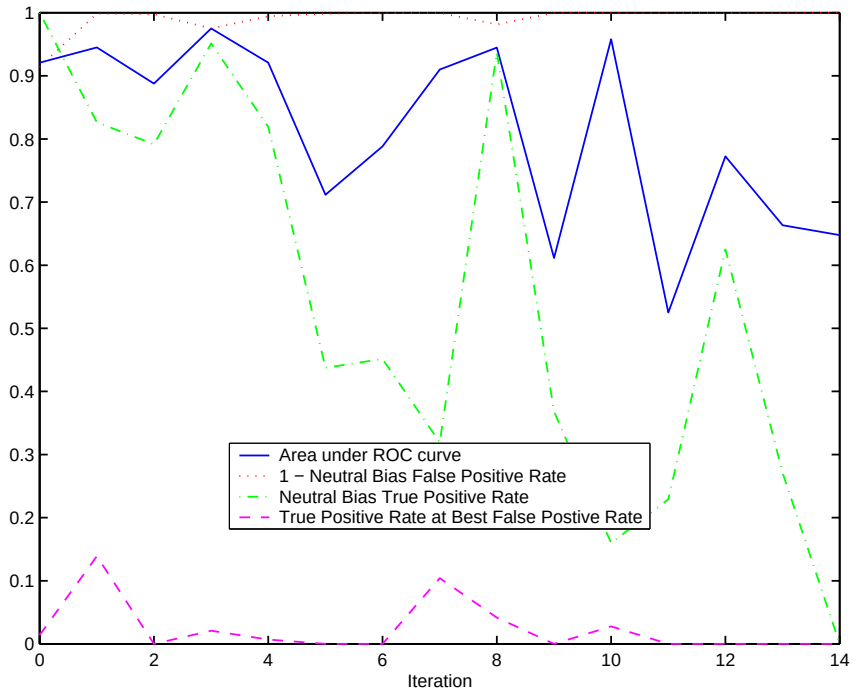



**Figure 3.4:** Various performance measures for the 25-principal-component GMM network during the refinement process.

positive rate of 1 in 2,000,000 patches, on the `CMU` database. The closest these experiments came were a false positive rate of 1 in 833 patches for an 80% true positive rate, or with a different bias, 0 in 890,000 false positives for a true positive rate of 14%.

Why is the algorithm is not working as expected? Looking at the GMM cluster centres (figure 3.8) they appeared similar to ones Sung and Poggio obtained (figure 3.9,) especially in the 25-principle-component model. This suggested that the GMM building process was working well. The rectangular nature of the ROC curve suggested that the classifier score was quantised to relatively small set of probability values. Further investigation showed that the GMM distance values for each point were not obviously quantised in this way. The outputs of the MLP's hidden layer showed some signs of quantisation, but not to the same extent as the final output. This suggested that there was a problem with the training of the neural network.

The neural network module was based on software[87] published by John Manslow.

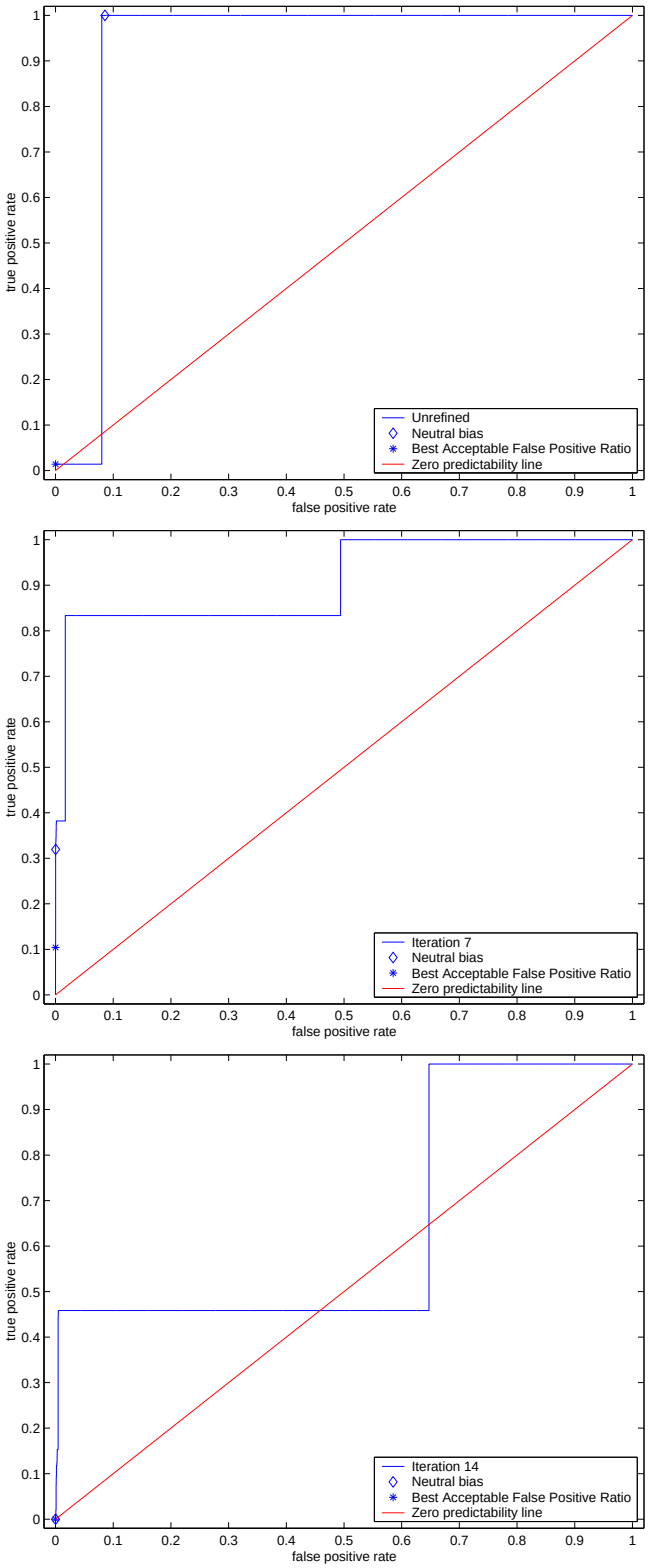



**Figure 3.5:** ROC curve for the 25-principal-component GMM network before refinement (top,) after 7 refinement iterations (middle,) and after training stopped.

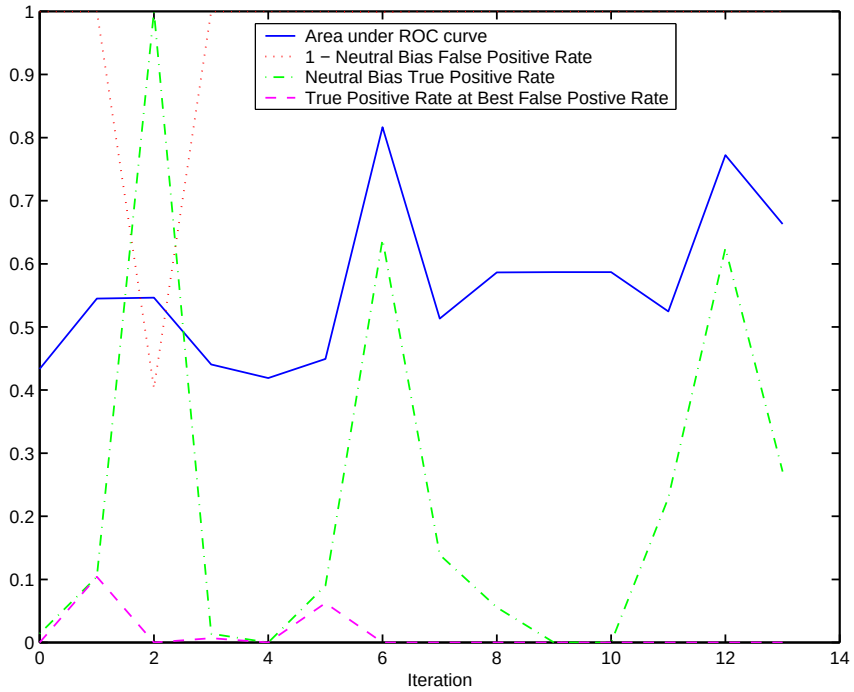



**Figure 3.6:** Various performance measures for the 75-principal-component GMM network during the refinement process

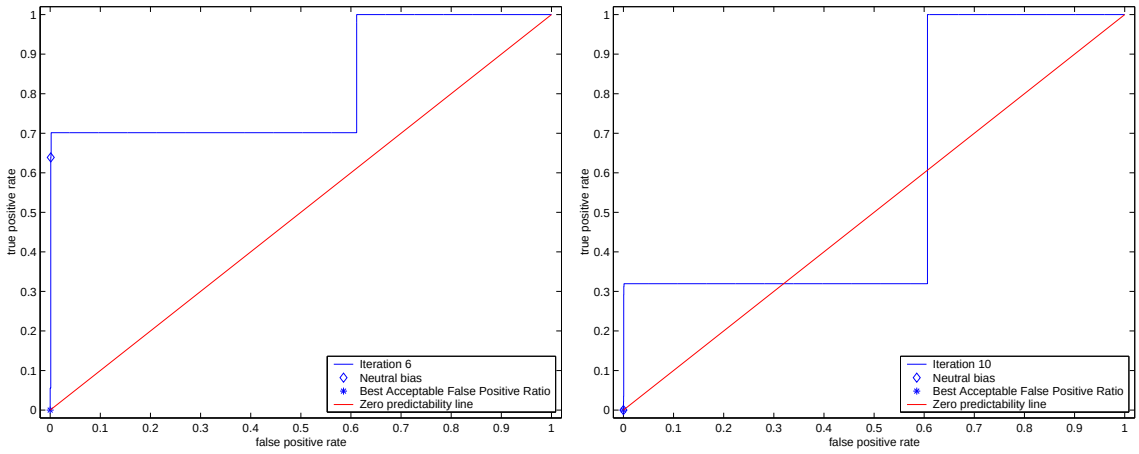



**Figure 3.7:** ROC curve for the 75-principal-component GMM network after 6 refinement iterations (left,) and after training stopped (right.)

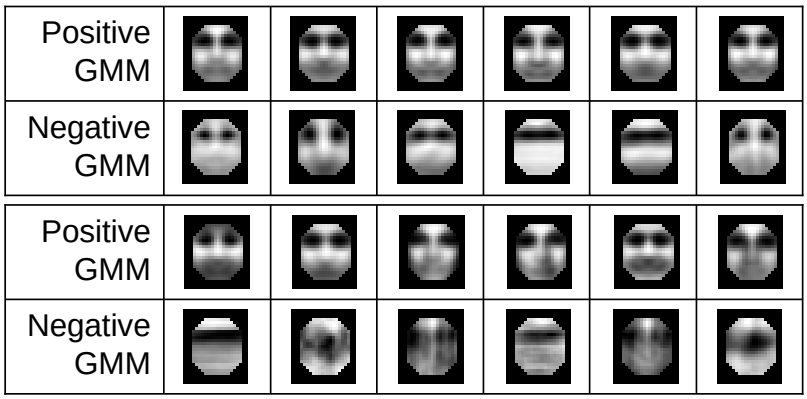



**Figure 3.8:** The centres of the 25 (top) and 75 (bottom) principal component Gaussian Mixture models after refinement.

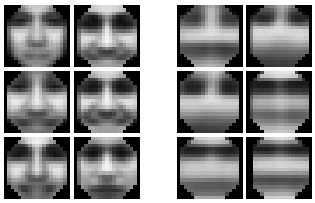


**Figure 3.9:** The centres of the 75 principal component Gaussian Mixture Models after refinement, as found by Sung and Poggio[145].

It used a simple downhill optimiser with momentum and weight decay. Multiple restarts with random weight initialisation were added to the code as it was ported to work on top of the Vision-X-Libraries (VXL) framework. It had been tested on some simple 2D and 3D datasets and appeared to work well. The performance might have been improved by using cross-validation to decide when to stop training, but Sung and Poggio never mentioned using this (or any other specific detail about MLP training.) The performance of the classifiers on the training set was no better than on the test set, suggesting that cross-validation would not help. In fact, the performance on the negative training set was much worse, because it consisted of images that the classifier had been unable to correctly classify at some point.

### 3.3.2 Confirming the Separability of the Data

The failure to achieve the performance results approaching those of Sung and Poggio, prompted a further search for implementation errors. The test framework in which the code was developed suggested that every part of the system was behaving as intended and, other than problems in the final stage of the MLP, no specific problems could be found with the GMM network. Indeed, the whole GMM network appeared to work reasonably well on artificial datasets, and was doing better than random on the face detection task. Perhaps there was something wrong with the feature vector extraction, despite a test framework that sampled from an image, and then reconstructed the image from the feature vector, checking they were consistent. Alternatively, perhaps there was a problem with the data.

An attempt was made to see if a flaw somewhere caused the feature vectors to be non-separable before they were passed to the GMM network for either training or classification. Sammon[127] describes an algorithm to perform dimensionality reduction "such that the inherent data 'structure' is approximately preserved." This algorithm calculates the distances between all the sample points in an original high-dimensional space and, starting with a random arrangement of the points in a low-dimensional space, iteratively modifies the arrangement to maximise the similarity of point-to-point spacing in high and low dimensional spaces.

Sammon mapping was applied to a combined dataset of 500 positive and 500 negative face samples, selected randomly[1] from the final training set of the first experiment described in section 3.3 . The results (figure 3.10) show that the data appears to be mostly separated. In addition it also suggests that the positive region is not fully connected, but instead is divided into a small number of regions surrounded by an invalid face view region. It is ill-advised to read much more into the results.

[1]The algorithm's storage requirements are $O(n^2)$ in the number of samples. The implementation[158] that was used appeared to be somewhat profligate in the use of local memory, thereby placing a relatively low limit on the number of samples that can mapped.

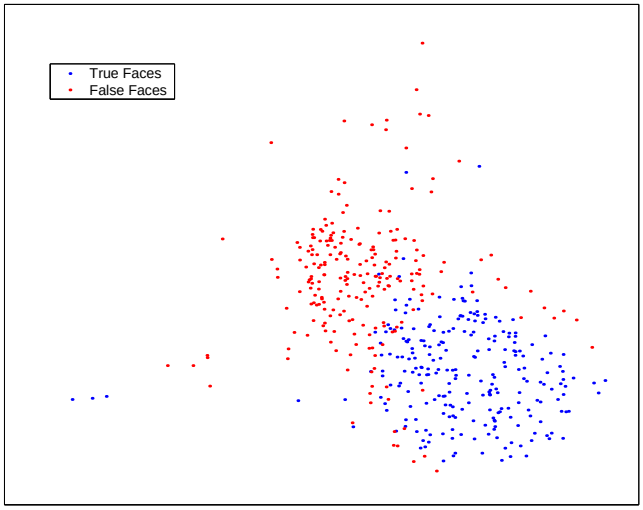



**Figure 3.10:** 2D Sammon map of positive and negative examples of faces appearance vectors.

### 3.3.3 Speed

The classification was very slow. Scanning a 400x400 pixel image over the full range of scales (separated by a scaling factor of $1.2 \times 1.2 = 1.44$) took 433,305 individual patch classifications. This took a 600MHz Pentium3 processor approximately 1 hour with a classifier built from GMMs with 25 principal components and 3 hours with 75 principal components. This gives a patch testing rate of $\sim$50 per second for the 75-principal-component case. This speed was doubled by first using a 20-principal-component single-mode Gaussian classifier to quickly reject about half the samples, at the expense of also excluding 0.5% of the true faces.

In relation to the longer term aims of this project, this speed may not be a problem. It is possible that more complicated appearance models can find less than 100 possible model view matches per second, then testing the validity of each of those matches would take another second. Nevertheless, an algorithm which classifies at a speed that is orders of magnitude faster would be very useful.

The slow classification speed is compounded by slow training. During the one refinement iteration, building the negative GMM and training the neural network can take

many hours. In the case of a 75 principal component GMM and MLP, with ∼20,000 negative and ∼4400 positive examples, for one iteration the GMM training took ∼15 hours, and the MLP training (with 10 random initialisations) took ∼5 hours.

## 3.4 Further Experiments

Since the (supposedly simple) MLP in the classifier appeared to be the cause of the poor performance, it was replaced by a linear classifier. Sung and Poggio describe experiments where they used a single layer Perceptron—the linear classifier used here was expected to be roughly equivalent. Since training a linear classifier uses a simple least squares fitting method, no problems were expected with local minima and convergence failure.

In an attempt to speed up the refinement process, the selection of false positives was changed from random, to taking the false positives with the highest classification score. The idea was that by focussing on the *worst* false positive examples, the classifier would more quickly find the true boundary between faces and non-faces. Alternatively, this would also allow the number of false positives added to the negative training set at each step to be reduced, so that training would not take so long. So $n_{\text{select}}$ was set to 500. However, this experiment failed completely. After the second refinement iteration, the worst false-positives were ones that had been added to the training set in the previous iterations. Since repeated examples in the training set should make the classifier more sensitive to those values, and eventually be able to classify them correctly, the experiment was allowed to continue for another eight iterations. But each iteration added the same examples, which remained the worst despite being weighted eight to ten times more important during training than the others.

As explained previously, finding all the false positives in a test set at each iteration is expensive in terms of space. It is also expensive in terms of time, especially since

many of the false positives will be discarded. Instead, it is faster to search for the first $n_{\text{select}}$ false positives. These might not be very representative of the whole negative training set, so the performance improvement per refinement iteration would not be as good. But, the whole iteration is completed much faster, and so the performance improvement per unit of computer time is much faster. This *find the next $n_{select}$ false positives* strategy was employed in all subsequent experiments, based on GMM networks and later with other classifiers.

A larger face database, called `webimagesC`, was acquired for testing—see figure B.6 in the appendix for some examples. Searching the WWW using terms like "team pictures" and "celebrity photos" made it very easy to acquire a large number of varying quality face images, leaving more time for the harder task of marking up the eyes. The negative test database was extended to 44 images containing $1.22\times10^7$ patches. To save time, testing on the negative database was restricted to a randomly selected $2.03\times10^6$ patches.

A final experiment using all these changes was run for eight iterations looking for $n_{\text{select}} = 2500$ new false positives, before the computer crashed (for unrelated reasons.) After eight iterations, the negative training set had 25,000 items. The positive set still had the 4398 examples from the `CMU` database. The performance on the test database measured on the last iteration gave a sensitivity of 54% for a false positive rate of one per ∼32,000 patches. As can be seen from the ROC curve (figure 3.11,) the performance has increased significantly over the previous experiments (figures 3.5 and 3.7.) This is almost certainly due to the replacement of the broken MLP, with a working linear classifier. Despite the large improvements over the previous experiments, the results still do not compare well with those reported by Sung and Poggio.

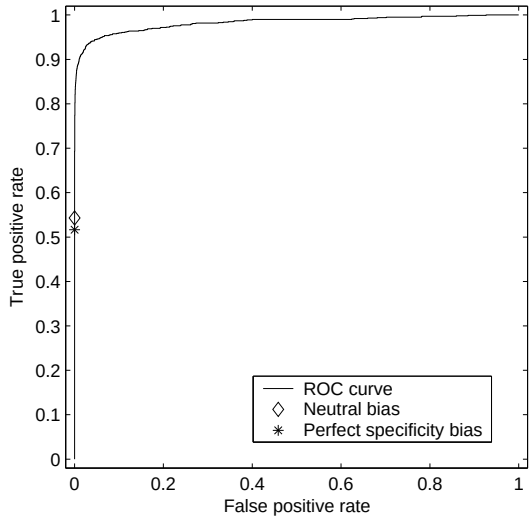



**Figure 3.11:** ROC curve of final GMM Network experiment, using 2×6 Gaussians with 35 principal components, and a linear classifier for the final stage.

## 3.5 Discussion and Conclusions

The work of this (and the next) chapter shows considerable evolution in experimental design. Since large amounts of data are involved, the training and testing of a classifier can take weeks. It was simply infeasible to run experiments multiple times merely to refine the experimental design, or re-run old experiments after the design was subsequently improved. The lessons learnt at each experiment were applied to the next ones, and enough lessons were learnt to enable the experimental designs to stay consistent in later chapters. However, even if results from different experiments could not be directly matched, the designs are similar enough to reach conclusions.

Although the idea of incremental classifier refinement seems a powerful idea, and Sung and Poggio reported very encouraging results with their GMM network, my results suggested that performance was very sensitive to the choice of training data and parameter settings, making it difficult, in practice to achieve similar results consistently. This lead to an investigation of more robust methods of classification.

# Chapter 4

# Building Support Vector Machine Classifiers

This chapter examines the Support Vector Machine (SVM) as an alternative classifier to the GMM network, and experimental work shows that SVMs are more robust and more accurate. A novel improvement to the training of SVMs is described, which automates the selection of a key SVM training parameter—the Radial Basis Function (RBF) width. Automatic selection of the SVM training parameters is valuable because running many experiments with different feature vectors would make manual intervention very expensive.

Two algorithms for training SVMs, Chunking-QP and Sequential Minimum Optimisation (SMO), are compared, after the SMO algorithm was heavily improved to increase its speed, numerical accuracy, and to calculate values needed for the automatic selection of the SVM training parameter. Experimental results show that Chunking-QP was numerically unstable, and unsuitable for use in situations where little manual intervention is required. However, the improved SMO algorithm was reliable and usually faster than Chunking-QP, making it most suitable as a classifier for later use with appearance modelling.

## 4.1 Support Vector Machines

Support Vector Machines (SVMs) were first described by Vapnik, in a Russian journal in 1979, but his first English book[156][1] is the standard reference. Whilst an SVM can also be used for non-linear regression, only its use as a classifier is examined here.

### 4.1.1 Introduction to SVMs

In order to understand the methods developed in this and subsequent sections, a short summary of the SVM derivation is given.

The expected error (or structural risk, $R_{\text{struct}}$) for a classifier applied to an unseen test set was shown by Vapnik[156] to be bounded above by the sum of the empirical error on the training set, $R_{\text{train}}$, and a function of a positive integer $h$ called the Vapnik Chervonenkis (VC) dimension and a variable $\eta = [0, 1]$:

$$R_{\text{struct}} \leq R_{\text{train}} + \sqrt{\frac{h(\ln \frac{2l}{h} + 1) - \ln \frac{\eta}{4}}{l}} \tag{4.1}$$

where $R_{\text{train}}$ is the error on the $l$-example training set, and the inequality holds with probability $1-\eta$. The VC dimension, $h$, is a measure of the capacity of the classifier to learn to fit to any arbitrary dataset, and is entirely independent of the data. $h$ is the largest number of vectors, for which it is possible to train the classifier correctly for every possible labelling of the vectors. For example, given three distinct $\mathbb{R}^2$ vectors, it is easy to place a line so that the three vectors are all on one side of the line, all on the other side, or any arbitrary combination of sides. This is not possible with four vectors (the cause of the infamous XOR problem for simple Perceptrons,) and so the VC dimension of the 2D linear classifier is three.

Equation 4.1 provides an upper bound on the expected test error of any classifier, and behaves as we might expect—the test error is never smaller than the training

[1] Vapnik's work is very dense and theoretically-oriented. Burges published a useful tutorial[21], and Gunn published a very clear introduction[65] with an easy to understand Matlab toolbox.

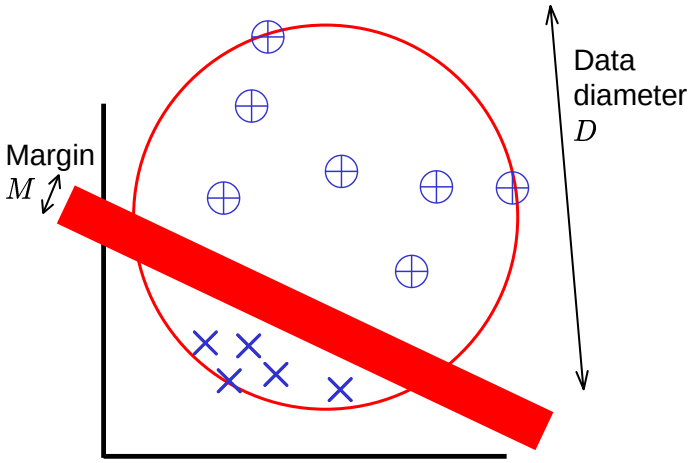



**Figure 4.1:** A thick hyperplane classifier separating two classes.

error. The second term, the risk of poor generalisation, increases with the capacity of the classifier to arbitrarily fit larger datasets, and decreases with the number of training samples.

Given a separating hyperplane of thickness (or margin) $M$, Vapnik argues that there is an upper bound on the classifier's VC dimension:

$$h \leq \frac{D^2}{M^2} \tag{4.2}$$

where $D$ is the diameter of a hypersphere enclosing the training data (figure 4.1.) To keep the upper bound on the expected test error as low as possible, we therefore need to maximise the margin, $M$, of the hyperplane.

A (thin) hyperplane classifier operates according to the following rule:

$$\text{sign}\, f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} - b) \tag{4.3}$$

where $w$ is the hyperplane's normal, and $b$ is the bias (figure 4.2a.) A thick hyperplane classifier is created by adjusting $\boldsymbol{w}$ and $b$ so that the edges of the margin are at $f(\boldsymbol{x}) = \pm 1$. Maximising the thickness, while modifying the hyperplane position to keep the training data out of the margin, gives a maximum-margin classifier (figure 4.2b.)

To find the values of $\boldsymbol{w}$ and $b$, for a training set of $n$ vectors $\boldsymbol{x}_i$ and labels $y_i$, we want

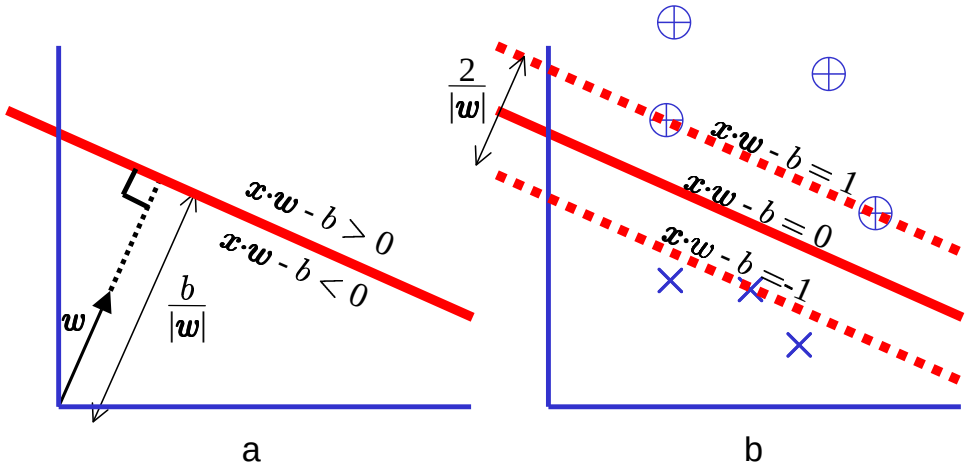



**Figure 4.2:** (a) A thin hyperplane classifier. (b) A maximum margin hyperplane classifier

to maximise $\frac{M^2}{2} = \frac{2}{|w|^2}$, or minimise its inverse. i.e.

$$\min_{\boldsymbol{w}} \frac{1}{2}|\boldsymbol{w}|^2 \tag{4.4}$$

such that the training vectors $\boldsymbol{x}_i$ are all on the correct side of the hyperplane and outside the margin:

$$\forall i : E_i = y_i(\boldsymbol{w} \cdot \boldsymbol{x} - b) - 1 \geq 0$$

To make this constrained optimisation easier to perform, the Lagrange multiplier technique[2] is used. We form a Lagrangian, $\Psi$, by introducing a positive Lagrange multiplier $\alpha_i$ for each constraint $E_i \geq 0$, and subtracting the products from the cost function.

$$\min_{\boldsymbol{w},b} \left[ \Psi = \frac{1}{2}|\boldsymbol{w}|^2 - \sum_{i=1}^{n} \alpha_i (y_i(\boldsymbol{w} \cdot \boldsymbol{x} - b) - 1) \right]$$

According to the theory of Lagrangians we will get the same parameters as equation 4.4 if we minimise $\Psi$, subject to the derivatives of the Lagrangian with respect to each multiplier going to zero ($\frac{\partial \Psi}{\partial \alpha_i} = 0$,) and to the multipliers being non-negative. Another result from Lagrangian theory tells us that if the Lagrangian is a convex function and the feasible set is convex (both of which are true here,) then there exists

[2]See any good book on optimisation e.g. Beale[11] for more about Lagrange multipliers.

a dual formulation. In this dual formulation we maximise $\Psi$, subject to the derivatives of the Lagrangian with respect to the original parameters ($\boldsymbol{w}$ and $b$) going to zero and the Lagrange multipliers being non-negative. i.e.

$$\max_{\alpha_1,\ldots,\alpha_n} \left[ \Psi = \frac{1}{2}|\boldsymbol{w}|^2 - \sum_{i=1}^{n} \alpha_i y_i (\boldsymbol{w} \cdot \boldsymbol{x} - b) + \sum_{i=1}^{n} \alpha_i \right]$$

subject to $\frac{\partial \Psi}{\partial \boldsymbol{w}} = 0$, $\frac{\partial \Psi}{\partial b} = 0$ and $\forall i : \alpha_i > 0$. We can use the constraints of this dual as follows:

$$\begin{aligned} \frac{\partial \Psi}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_i \alpha_i y_i \, \boldsymbol{x}_i &= 0 \\ \boldsymbol{w} &= \sum_i \alpha_i y_i \, \boldsymbol{x}_i \end{aligned}$$

$$\frac{\partial \Psi}{\partial b} = \sum_i \alpha_i y_i = 0$$

Inserting these results back into the Lagrangian gives

$$\max_{\alpha_1,\ldots,\alpha_n} \left[ \Psi = \sum_i \alpha_i y_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \, \boldsymbol{x}_i \cdot \boldsymbol{x}_j \right] \tag{4.5}$$

subject to $\sum_i \alpha_i y_i = 0$ and $\forall i : \alpha_i \geq 0$. This quadratic cost function with linear constraints is known as a Quadratic Programming (QP) problem, for which there are standard algorithms.

One important generalisation allows the method to be used with non-separable training sets (e.g. figure 4.3a,) by allowing the vectors to encroach into the margin, with relative cost $C$. After extending the derivation to cope with this we get:

$$\max_{\alpha_1,\ldots,\alpha_n} \left[ \Psi = \sum_i \alpha_i y_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \, \boldsymbol{x}_i \cdot \boldsymbol{x}_j \right]$$

subject to $\sum_i \alpha_i y_i = 0$ and $\forall i : 0 \leq \alpha_i \leq C$. The only change is an additional set of simple upper bounds on the Lagrange multipliers. Again this is a straightforward QP problem. Figure 4.3b shows a example of the optimisation problem with just two training vectors. In such a case, the two training points lie on the margin.

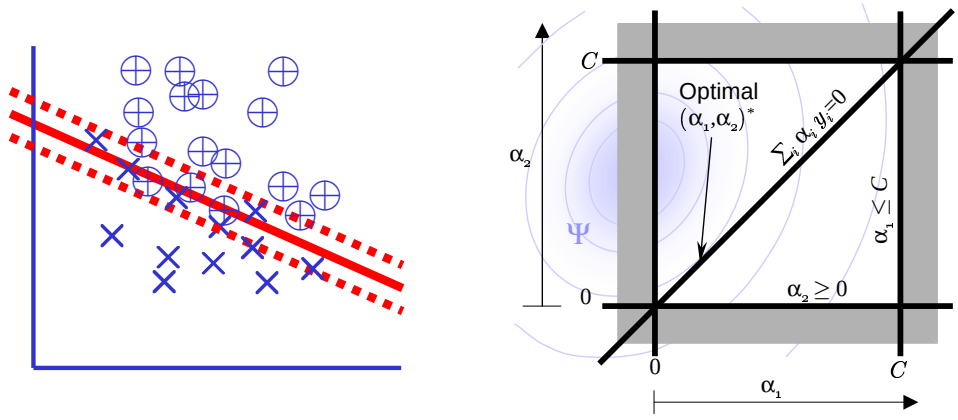



**Figure 4.3:** (a) A hyperplane classifier with non-separable data. Some data points are inside the margin. (b) The QP problem for a simple two-example training set.

The next stage is to turn this hyperplane classifier into a non-linear classifier. By mapping the data non-linearly into a higher dimensional space, $x \rightarrow \Phi(\boldsymbol{x})$, it becomes easier for the classifier to find a separating hyperplane. The Lagrangian thus becomes

$$\max_{\alpha_1,\ldots,\alpha_n} \left[ \Psi = \sum_i \alpha_i y_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_j) \right]$$

Note that all the references to the input data, e.g. in the Lagrangian (equation 4.5) or the decision rule (equation 4.3,) take the form of dot products between two input vectors. We can therefore roll the dot product and non-linear mapping into a single kernel function, $K$.

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_j)$$

The kernel function $K()$ is chosen so that it is cheap to calculate—certainly cheaper than explicitly mapping the data into a higher dimensional space, and calculating the dot product. In some cases, it may not even be possible to actually compute the mapping. For example, with the Gaussian RBF kernel:

$$K_{\text{RBF}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi_{\text{RBF}}(\boldsymbol{x}_i) \cdot \Phi_{\text{RBF}}(\boldsymbol{x}_j) = \exp - \frac{|\, \boldsymbol{x}_i - \boldsymbol{x}_j \,|^2}{2\sigma^2}$$

The codomain of the mapping is the unit-radius $\infty$-dimensional hypersphere. Two input vectors that are close in the input space (i.e. closer than the RBF width, $\sigma$)

are also close on this hypersphere. Two vectors that are far away in the input space, are up to 90° apart on the hypersphere.

Replacing the dot product with the kernel function, $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, gives

$$\max_{\alpha_1,\ldots,\alpha_n} \left[ \Psi = \sum_i \alpha_i y_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \right] \tag{4.6}$$

subject to $\sum_i \alpha_i y_i = 0$ and $\forall i : 0 \leq \alpha_i \leq C$, which can be solved as before.

It will be useful later in this chapter to be able to calculate the margin width, $M$, from the optimal Lagrange multipliers:

$$M^2 = \frac{4}{|\boldsymbol{w}|^2} \tag{4.7}$$

$$= \frac{4}{\sum_{i,j} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)} \tag{4.8}$$

### 4.1.2 Applications and Extensions

Osuna *et al.*[108] were the first to use an SVM for the object (face) detection problem. Their feature extractor, and normalisation was identical to Sung and Poggio's[145]. The main differences compared to the work in this chapter, were the use of a single refinement iteration, and the manual choice of a second order polynomial kernel, $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and soft-margin cost, $C$, of 200.

Burges[20] showed that the classification boundary of an SVM can be estimated reasonably accurately by a similarly structured classifier but with many less support vectors. His Reduced Set method optimises the values of the new classifier's support vectors to fit the old decision boundary. Romdhani *et al.*[121] extended this, turning the decision into a series of reduced set SVMs. For tasks like object detection, where the vast majority of test examples are negative, a large fraction of the negative examples can be rejected by a classifier with a single support vector. A two vector classifier then goes on to reject a fraction of the remainder, and so on. The biases are set so that positive test examples will make it through to a 100-vector classifier.

All examples that make it to that stage are then tested by the full SVM to maintain specificity. Romdhani *et al.*, like Osuna *et al.*, only used a single refinement stage, and manually chose their kernel parameter (RBF width.)

Lin *et al.*[85] showed how different costs can be associated with misclassification of positive and negative data (similar to the Bayes-risk in biased Bayesian classifiers) leading to different values of soft-margin cost $C$ for positive and negative training data.

There have been several alternative functions proposed for estimating the test error on unseen datasets as the kernel function and parameters (e.g. RBF width) are varied. Joachims[75] proposed a very efficient estimate of the leave-one-out cross-validation error. Vapnik and Chapelle[157] proposed a tighter upper bound than equation 4.2 on $R_{\text{struct}}$, but were unclear about how to calculate it. Later they proposed[25] an extension of the method used in this thesis (section 4.3.3) to choose a different RBF width for each element of the input vector. In this work they showed how to use an optimiser to find the best kernel parameters, but appear to have used prior knowledge of the data to manually choose the initial value passed to the optimiser.

### 4.1.3 Using Chunking to Train SVMs on Large Datasets

Standard quadratic programming implementations, such as the `quadprog` function in Matlab's Optimisation toolbox can be used to solve the SVM training maximisation—indeed `quadprog` is the basis of Gunn's SVM toolbox[65][3]. The objective function is passed to such routines in vector-matrix form, i.e. $\Phi = \boldsymbol{g}^T\boldsymbol{\alpha} + \boldsymbol{\alpha}^T\boldsymbol{H}\boldsymbol{\alpha}$. The constraints are similarly assembled into vectors and matrices. This standard approach has a problem: When the number of training vectors $n$ is large, it is not possible to

[3]Gunn's SVM toolbox for Matlab is a very useful for exploring SVM methods, and performing simple experiments. This is largely because it is easy to express the SVM optimisation in terms of vector and matrix products, and because Matlab provides a quadratic programming function. Unfortunately it is very slow, and does not scale to large datasets.

fit the entire Hessian $\boldsymbol{H}$ in memory.

$$\boldsymbol{H} = \begin{pmatrix} y_1 y_1 K_{11} & y_1 y_1 K_{12} & \dots & y_1 y_n K_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ y_n y_1 K_{n1} & y_n y_1 K_{n2} & \dots & y_n y_n K_{nn} \end{pmatrix}$$

This problem can be dealt with by realising that only a small proportion of the training vectors, the *support vectors*, will be touching the margin. All the other training vectors will have their associated Lagrange multipliers set to zero. This means that the contribution of the associated elements of the Hessian will also be zero, and so can be left out. The "Chunking" method introduced by Osuna *et al.*[108] uses this observation to reduce the full $n^2$-sized QP problem to a series of $n_{\text{sub}}^2$-sized QP sub-problems, according to the algorithm in figure 4.4. The terminating condition for this (or any) convex-constrained optimisation problem is given by the Karush Kuhn Tucker (KKT) conditions. They can be interpreted as requiring that the Lagrangian be stationary at the optimal point. For the SVM optimisation this means[4]:

$$\begin{array}{lllll} \forall i, & \alpha_i = 0 & \Rightarrow & E_i >= 0 \\ & 0 < \alpha_i < C & \Rightarrow & E_i = 0 \\ & C = \alpha_i & \Rightarrow & E_i <= 0 \end{array} \tag{4.9}$$

where the KKT errors, $E_i$, are defined as the amount by which the conditions are violated:

$$E_i = y_i(f(x_i) - y_i) \tag{4.10}$$

A concrete implementation of the Chunking algorithm was published by the Saunders *et al.*[129](Royal Holloway, University of London—RHUL) and this code was used for the SVM experiments in this and the next section (4.3.) The RHUL implementation uses a constrained version of the conjugate-gradient optimisation method to solve

[4]Strictly, the KKT conditions[11] also include the constraints already imposed within and upon the Lagrangian (equation 4.6.)

- Initialise all Lagrange multipliers to zero
- Randomly choose $n_{\text{sub}}$ Lagrange multipliers, and solve QP sub-problem
- Calculate current bias, and test KKT conditions
- While not all KKT conditions hold
  - Add any multipliers to the QP sub-problem, for which the associated KKT conditions fail
  - Remove similar number of zero-valued multipliers from the QP sub-problem
  - Solve QP sub-problem again
  - Calculate current bias, and test KKT conditions

**Figure 4.4:** Summary of the Chunking algorithm.

the QP problem. The inequality constraints are added to the standard conjugate-gradient algorithm[11], by projecting the conjugate-gradient vector into the subspace defined by the equality constraints and any active inequality constraints.

SVM training has a single globally optimal solution, which the algorithm is guaranteed to find (at least within the limits imposed by numerical rounding.) This is in distinct contrast to other classifier methods, including Sung and Poggio's GMM network classifier, where the optimisation can get stuck in local minima. The SVM's theoretical underpinnings are also reassuring, and can be used to track down problems, or construct extensions to the methods.

## 4.2 Comparison of SVMs with GMM Networks

Before significant work on SVM classification was undertaken, an initial comparison was made between SVMs and the full Sung and Poggio GMM network. The training data from the 25-principal-component GMM network with MLP was extracted and randomly sampled to produce 250 positive and 250 negative training examples. Using this data, and the RHUL software, a Gaussian RBF SVM was trained with $C = \infty$ and the RBF width of 0.71. The SVM was then tested on the same examples as the 25-principal-component GMM network (see section 3.3.) The SVM's bias was

**Figure 4.5:** Comparing SVM and GMM networks on face detection

| | GMM Network | | | SVM | | |
|---|---|---|---|---|---|---|
| Positive training samples | 4398 | | | 250 | | |
| Negative training samples | 39000 | | | 250 | | |
| Error rate on positive samples | 3.5% | 16.7% | 61.8% | 3.5% | 5.2% | 26.8% |
| Error rate on known difficult negative samples | 49.4% | 2.2% | 0.9% | 8.5% | 2.2% | 0.9% |

adjusted (by manually adjusting $b$ in equation 4.3) to three different values to show the performance over a range of trade-offs between specificity and sensitivity. The results are shown in figure 4.5 along with three equivalent results for the GMM network. The SVM achieved much superior performance, with much less training data. The SVM's excellent performance *out of the box* compares very favourably with the amount of effort needed to get Sung and Poggio's algorithm to work at all. Further experiments with SVMs confirmed this initial promise, so further work on Sung and Poggio's GMM network was abandoned.

## 4.3 Choosing the Best SVM Training Parameters

One advantage of SVMs over other classification methods is the very small number of control parameters. This is valuable, because classifier performance can be quite sensitive to the choice of parameters. For example, the problems with Sung and Poggio's GMM network (section 3.3) illustrated this. Similarly, with back-propagation training of MLPs, the learning rate and number of iterations can greatly affect the performance of the final classifier[114, p.155]. For a standard non-linear SVM classifier, the only parameters are the value of the upper bound, $C$, on the Lagrange multipliers, the choice of kernel function $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and the parameters of the kernel function.

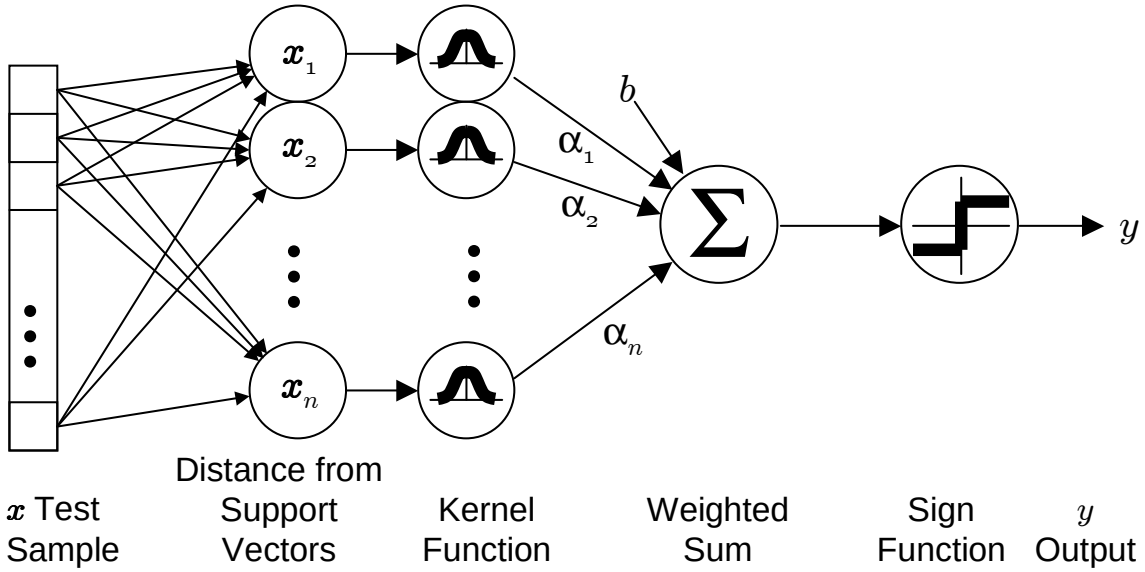



**Figure 4.6:** After training, the RBF SVM is structurally identical to the simple RBF neural network. They both have the classification score: $y = \sum_{i=1}^{n} \exp(\frac{|\boldsymbol{x}-\boldsymbol{x}_i|^2}{2\sigma^2}) - b$

By assuming that the positive and negative classes are separable, we can set $C = \infty$. This assumption is usually valid with high dimensional data sets—more dimensions mean more room to find a separating hyperplane. The choice of kernel function is restricted to those which can be expressed as dot product of some mapping, $\Phi$, of the data. $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_j)$. In the literature, it appears that the choice of actual kernel function is often irrelevant. As in the choice of function to use in a neural network, a range of non-linear functions seem to perform roughly equivalently. However, in the absence of a good reason to choose any other, the Gaussian RBF $K_{\text{RBF}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp -\frac{|\boldsymbol{x}_i - \boldsymbol{x}_j|^2}{2\sigma^2}$, and the polynomial $K_{\text{poly}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = [(\boldsymbol{x}_i \cdot \boldsymbol{x}_j) + 1]^p$ appear to be most popular in much of the literature (e.g. [44, p.149].) One advantage of the Gaussian RBF is its familiarity—the final SVM looks just like the well-known[14, pp.167-169] single-output-node RBF neural network (figure 4.6.) RBF kernel SVMs also appear to behave more intuitively in regions with no data, with the classification boundary being close to that of a K-Nearest Neighbour classifier. As can be seen in the classification boundaries calculated by Gunn[65], this is not so often true of polynomial kernel SVM classifiers. In the absence of any evidence that the RBF is a poor choice of kernel, the work of this thesis examined no others.

This leaves the RBF kernel width as the only remaining parameter choice. The standard method of automatically adjusting such parameters in any trained statistical model is to try several different values of the parameter, and pick the one with the best performance. The performance of the algorithm is assessed using cross-validation, by splitting the training set, and using one part for training, and the other part for testing (or cross-validation.) Splitting the training set equally, unfortunately gives the best estimate of the parameter for a training set half the size of the whole set. This problem can be dealt with by using leave-one-out cross-validation. Here the dataset is split into a large training set and a singleton test set. After training and performance testing on the one example, the process is repeated with examples selected in turn for testing. The average of all the single example performance values gives an almost unbiased estimate of performance given the full size training set. Leave-one-out cross validation is a well-understood, accurate method and is recommended in several introductions to SVMs[65, 44, p.149] but is very computationally expensive. If it takes 1 hour to train an SVM on a 2000 vector training set, it will take about 2000 hours to provide an estimate of the performance of one parameter value.

An alternative approach was suggested briefly by Vapnik[156], using the structural risk minimisation principle that underpins SVMs. Using equations 4.1 and 4.2, it is possible to estimate (an upper bound on) the expected test error for an SVM with any given value of RBF width, $\sigma$. By finding the RBF width, $\sigma$, that gives the lowest value of (the upper bound on) expected test error, $R_{\text{struct}}$, we might expect that this would also give the lowest actual test error. It is important to recognise the caveats in the preceding sentence—all that can be minimised is a function that $100.(1-\eta)\%$ of the time will be no less than the desired cost function. However, a simple experiment on digit recognition by Burges[21] showed that the minimum in the actual test error occurred for similar values of the RBF width as that predicted by the upper bound. Vapnik[156, pp.147-149] found similar behaviour when picking the exponent for a polynomial kernel SVM on the same database. Early experiments (see section 4.3.2) with faces tentatively confirmed this result.

### 4.3.1 Diameter of the Minimum Enclosing Hypersphere

Equations 4.1 and 4.2 show that to calculate (and so minimise) the upper bound on the expected test error, $R_{\text{struct}}$, it is necessary to calculate the diameter of the training data's Minimum Enclosing Hypersphere (MEH).

It is known in the SVM literature (e.g. [21]) that the MEH can be found using an optimisation similar to the SVM's training algorithm, and the derivation is briefly described here. Given a set of $l$-dimensional vectors $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$, we want the smallest hypersphere:

$$\min_{\boldsymbol{a},D} D^2 \tag{4.11}$$

such that the distances to each data point from the centre $\boldsymbol{a}$ of the hypersphere, is no larger than the radius $D/2$.

$$\forall i : E_i = D^2 - 4|\,\boldsymbol{x}_i - \boldsymbol{a}|^2 \geq 0 \tag{4.12}$$

We choose $D^2$ for the objective function rather than $D$ to make the maths easier later. We can form a Lagrangian $\Psi_{MEH}$ by introducing a positive Lagrange multiplier $\alpha_i$ for each constraint $E_i \geq 0$, and subtracting the products from the objective function.

$$\min_{\boldsymbol{a},D} \left[ \Psi_{\text{MEH}} = D^2 - \sum_i \alpha_i (D^2 - 4|\,\boldsymbol{x}_i - \boldsymbol{a}|^2) \right]$$

According to the theory of Lagrangians we will get the same parameters as equation 4.11 if we minimise $\Psi_{\text{MEH}}$ subject to the derivatives of the Lagrangian (with respect to each multiplier) going to zero, and the multipliers being non-negative. As before, a dual formulation exists where we maximise $\Psi_{\text{MEH}}$ subject to the derivatives of the Lagrangian with respect to the parameters ($D$ and $\boldsymbol{a}$) going to zero, and the Lagrange multipliers being non-negative. i.e.

$$\max_{\alpha_1,\ldots,\alpha_n} \left[ \Psi_{\text{MEH}} = D^2 - \sum_i \alpha_i (D^2 - 4|\,\boldsymbol{x}_i - \boldsymbol{a}|^2) \right]$$

subject to $\frac{\partial \Psi_{\text{MEH}}}{\partial D} = 0$ and $\frac{\partial \Psi_{\text{MEH}}}{\partial \boldsymbol{a}} = \mathbf{0}$

We can use the constraints of this dual as follows

$$\frac{\partial \Psi_{\text{MEH}}}{\partial D} = 2D - 2D \sum_i \alpha_i = 0$$

$$\sum_i \alpha_i = 1$$

$$\frac{\partial \Psi_{\text{MEH}}}{\partial \boldsymbol{a}} = 4 \sum_i \alpha_i (2\,\boldsymbol{x}_i - 2\boldsymbol{a}) = 0$$

$$\boldsymbol{a} = \sum_i \alpha_i\, \boldsymbol{x}_i$$

Inserting these results back into the Lagrangian gives:

$$\max_{\alpha_1,\ldots,\alpha_n} \left[ \Psi_{\text{MEH}} = \sum_i \alpha_i\, \boldsymbol{x}_i \cdot \boldsymbol{x}_i - \sum_{i,j} \alpha_i \alpha_j\, \boldsymbol{x}_i \cdot \boldsymbol{x}_j \right]$$

subject to $\sum_i \alpha_i = 1$ and $\forall i : \alpha_i \geq 0$.

The dual formulation has $n$ parameters, 1 linear equality constraint, and $n$ very simple linear inequality constraints. This compares favourably with the primal formulation which had $l + 1$ parameters ($\boldsymbol{a}$ has $l$ dimensions,) and $n$ quadratic constraints. The other advantage of the dual formulation is that the problem is expressed in terms of dot products of the training vectors. This allows us to apply the same kernel trick as is used in SVM training to perform non-linear mapping into the same high-dimensional in which the maximum margin hyperplane is determined.

$$\max_{\alpha_1,\ldots,\alpha_n} \left[ \Psi_{\text{MEH}} = \sum_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}_i) - \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \right] \tag{4.13}$$

subject to $\sum_i \alpha_i = 1$ and $\forall i : \alpha_i \geq 0$.

To find the actual diameter, $D$, one can measure the distance from the hypersphere's centre $\boldsymbol{a}$ to any of the support vectors (which will be on the boundary.) To make the

result robust, it is better to use the Root Mean Square (RMS) of all the distances. To make the maths neater, the mean is biased by the Lagrange multipliers $\alpha_i$ rather than $1/\#$(support vectors). This is valid due to the constraint $\sum_i \alpha_i = 1$.

$$\begin{aligned} D^2 &= 4 \sum_i \alpha_i |\, \boldsymbol{x}_i - \boldsymbol{a}|^2 \\ &= 4 \sum_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}_i) - 4 \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &= 4 \max_{\alpha_1, \ldots, \alpha_n} \Psi_{\text{MEH}} \end{aligned}$$

subject to $\sum_i \alpha_i = 1$ and $\forall i : \alpha_i \geq 0$.

Comparing the MEH Lagrangian (equation 4.13) to the SVM Lagrangian (equation 4.6) shows that the two equations are quite similar, and that it should be relatively straightforward to modify almost any SVM training algorithm to perform the MEH diameter calculation. In particular, just setting $C = \infty$, and $\forall i : y_i = 1$ does much of the work. For an intuitive justification of why the same equations can represent both a hypersphere and a hyperplane, think of the quadratic Lagrangian as some kind of generalised hyper-cone. When intersected by the linear equality constraint this gives a generalised conic section. For linear constraints through the origin (point of the hyper-cone,) we get a hyperplane. For linear constraints perpendicular to the axis of the cone, we get a hypersphere.

In RHUL's Chunking code, dealing with—the extra $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ factors on the linear part of the cost function, the different constant factor on the quadratic part, and the slightly different linear constraint—can be devolved to the gradient calculation performed during the conjugate gradient routine (section 4.1.3.) One can no longer initialise all the multipliers to 0, since that would be outside the feasible region due to the constraint $\sum_i \alpha_i = 1$. So instead, the first multiplier is initialised to 1.

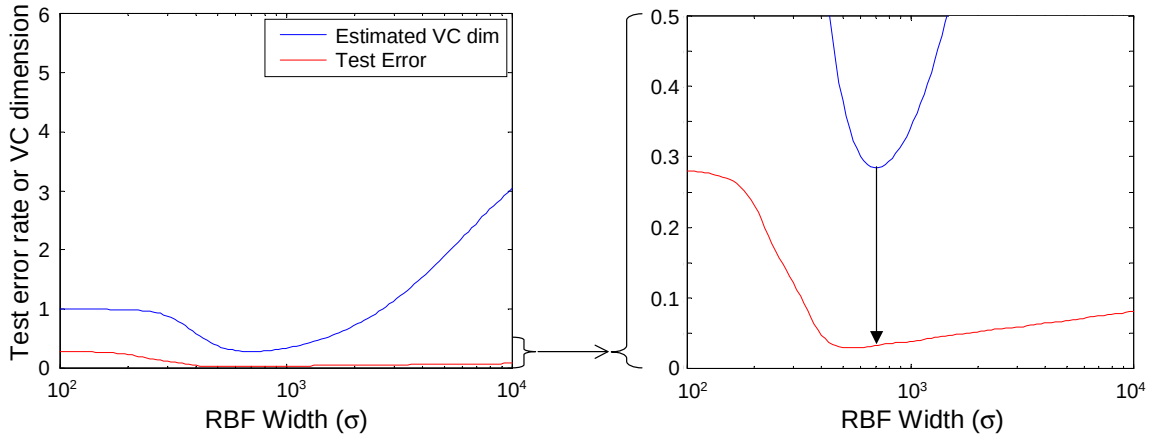



**Figure 4.7:** The upper bound on the VC dimension predicts the minimum test error of an SVM. The right hand graph is identical to the left hand one, but has the $y$-axis magnified 30 times.

### 4.3.2 Validation of Predicted Test Error

To test the validity of choosing the RBF width on the basis of predicted test error, the SVM training experiment in section 4.2 was repeated with 120 different RBF widths geometrically spaced between 100 and 10,000. In each case, the SVM could perfectly separate the training set. This meant that the upper bound on the expect test error is controlled solely by the VC dimension. For each trained SVM, the margin width was calculated using equation 4.7, and the MEH-diameter calculated using the MEH-modified code. The upper bound on the VC dimension was calculated using equation 4.2.

A test set of 250 positive examples and 500 known difficult negative samples was randomly selected from the 25-principal-component GMM network training sets in section 3.3. (The test set did not include any examples from the training set.) The results (figure 4.7) show that the minimum in the (upper bound of the) VC dimension predicts the best value of RBF width (of about 530) reasonably accurately. The prediction gives 24 test errors instead of the optimal 22. This is much better than the 210 test errors that picking an RBF width of 100 would give.

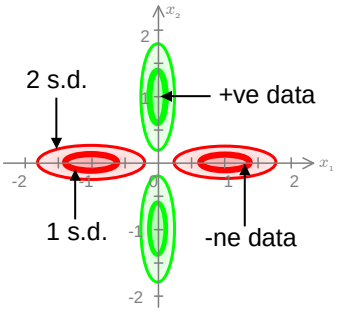



**Figure 4.8:** Artificial $n$-dimensional test-set generator for SVM tests. Positive data is sampled randomly from the green GMM, and negative data from the red GMM. The principal variance for each Gaussian is always ten times larger than the non-principal variance, although the size of all the variances can be scaled up to make the classification harder. All the Gaussians have zero mean and the same non-principle variance in directions $x_3$–$x_n$.

### 4.3.3 Automating the Search for RBF Width

Since it was intended to perform many experiments using classifier training, having the RBF width picked completely automatically would be very useful. Using the artificial data generator described in figure 4.8, $200 \times 2D$ vector training sets, and 1000 vector test sets, were generated with varying levels of difficulty. The easiest test set had the principle variance set to 0.15. This principle variance was then increased in factors of 1.5 to push the training data closer and closer together until they were overlapping significantly. For each dataset, the SVM optimisation was run at different values of $C$, the multiplier upper bound. Finally, SVM optimisation was repeated for different values of RBF width $\sigma$. Two notable issues arose whilst attempting to perform these experiments:

1. For large values of RBF width, some of the training vectors would fail to be classified correctly by the SVM. Despite this, the minimum training error would sometimes also be found at large RBF widths. To deal with this, rather than just use the upper bound on the VC dimension as previously, the full upper bound on $R_{\text{struct}}$ was used (equation 4.1.) However, for values of VC dimension greater than twice the number of training vectors, the bound starts

to fall—and will become complex eventually. This is presumably outside the range of equation 4.1's derivation, but no reference to this could be found in the literature. Since an RBF width finding algorithm might venture into this territory, equation 4.1 was replaced with a function that is monotonic in VC dimension:

$$R_{\text{struct}} \leq R_{\text{struct}}^{\text{upper bound}} = R_{\text{train}} + \begin{cases} \sqrt{\frac{h(\ln\frac{2l}{h}+1)-1}{l}} & : \quad h \leq 2l \\ \sqrt{2}\frac{\ln h}{\ln 2l} & : \quad h > 2l \end{cases}$$

2. Many of the SVM training runs failed to optimise with the QP code declaring that the optimisation suffered from "Numerical instability," meaning that it had spent many iterations getting nowhere. In order to get a sufficient fraction of the experiments to return useful results, a small ($\iota = 1{\times}10^{-9}$) value was added to the diagonal of the Hessian. This regularisation of the optimisation problem, slightly increased the slope of the Lagrangian away from the minimum, at the expense of shifting it slightly.

Using the above solutions, the experiments were then repeated, many of which either completed correctly, or did some useful work before stalling. Only a few of the MEH calculations failed to get anywhere, leaving a diameter estimate of zero. All the results are shown in figure 4.9, apart from the gaps in $R_{\text{struct}}^{\text{upper bound}}$ due to failures of the MEH algorithm. The graphs also contain a first guess at a bracket on the minimum in $R_{\text{struct}}^{\text{upper bound}}$. These were calculated by ranking all of the Euclidean distances between training points. For large datasets, a random subset of training vector pairs was used to save time. The lowest decile was used for the lower bracket, and the maximum separation used as the upper bracket.

Several further experiments with other artificial datasets showed similar results. For easy to separate datasets (i.e. where the data generator's variances are low,) the left hand minimum in $R_{\text{struct}}^{\text{upper bound}}$ is a good predictor of the lowest test error. For difficult to separate datasets, the left hand minimum in $R_{\text{struct}}^{\text{upper bound}}$ can disappear. In these harder to separate datasets, the maximum data vector separation (the right

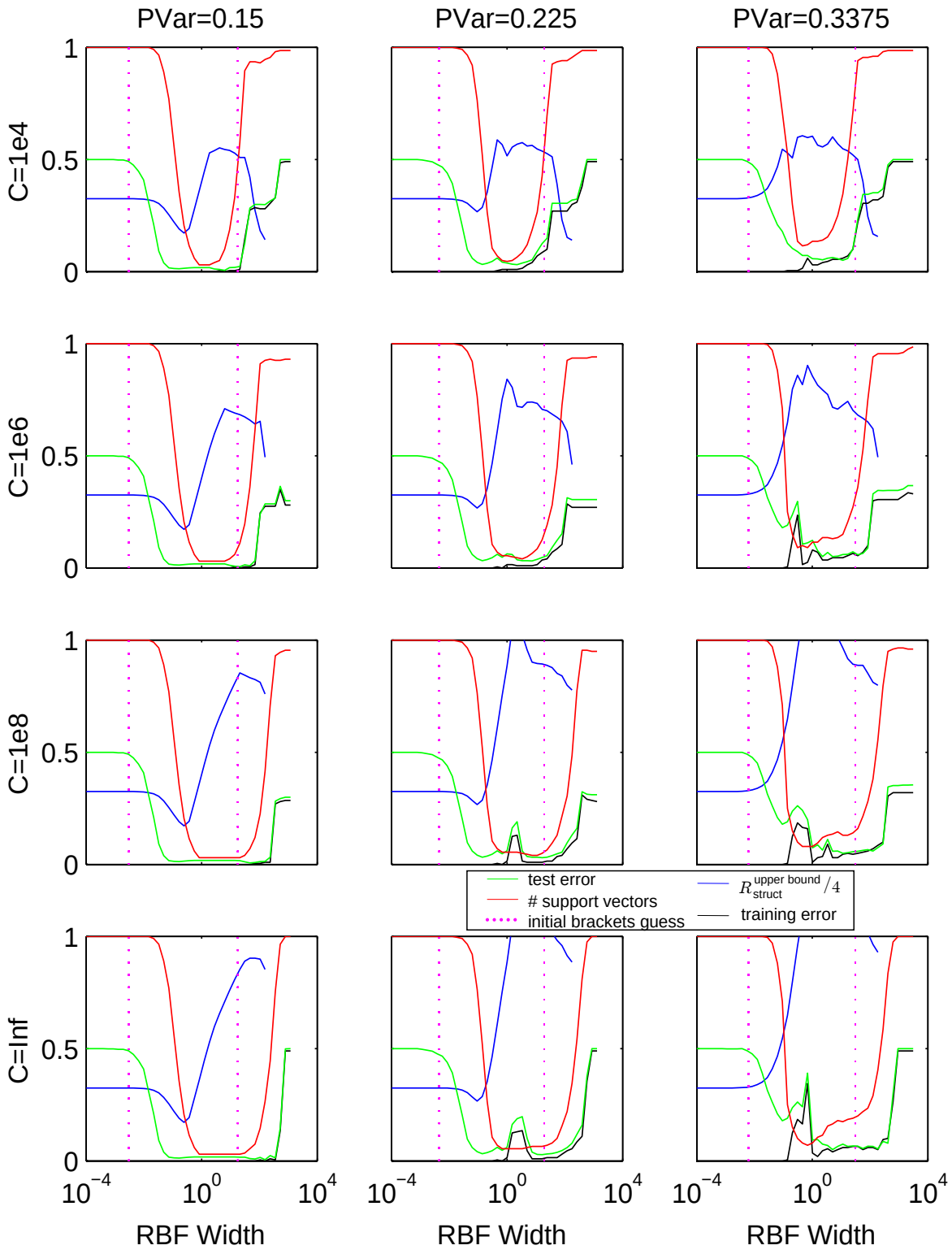
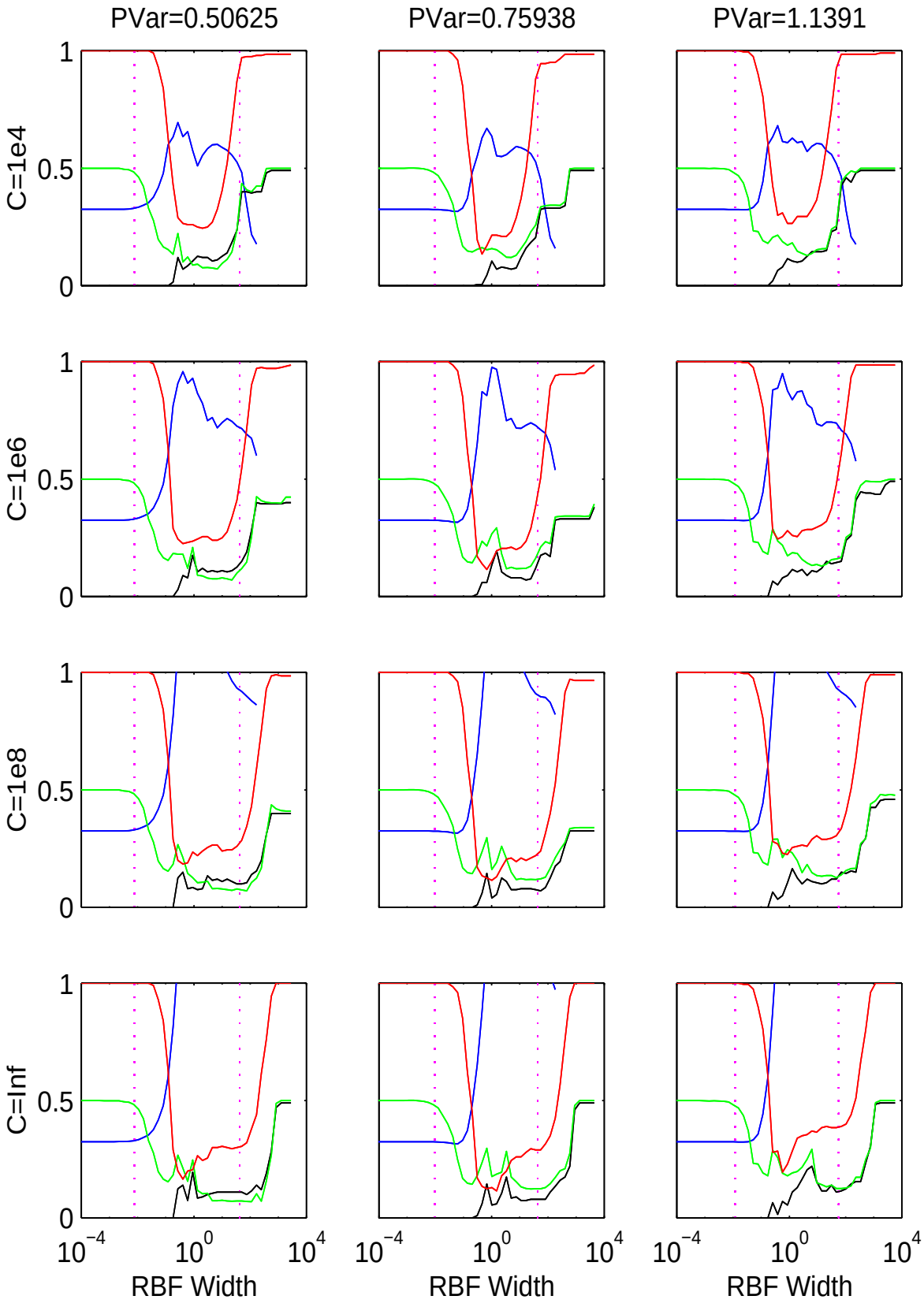



**Figure 4.9:** Test errors, Training errors, $R_{\text{struct}}^{\text{upper bound}}$, for different difficulties of test set (PVar—Principle variance of each mode of data generator—see figure 4.8—harder left-to-right,) different weights on training vectors encroaching into margin ($C$—less encroachment allowed top-to-bottom,) and different RBF widths ($x$-axis of each graph.)




Series continued from previous page.

hand dashed line) appears to often provide a better estimate of test error, than the minimum in $R_{\text{struct}}^{\text{upper bound}}$. This suggests an algorithm that starts searching around the lower guessed bracket for a minimum in $R_{\text{struct}}$ and if it fails to find a minimum, uses the upper guessed bracket.

The fact that $R_{\text{struct}}^{\text{upper bound}}$ appears to be useless at higher values of RBF width, appears to be connected with the emergence of non-zero training errors. Theoretically at least, the RBF SVM should be able to perfectly classify any training set, whatever the RBF width. Further investigation of this problem showed that the KKT conditions were not being accurately satisfied for these cases. This was obviously true for the SVM's which had failed to converge, but also in some cases where the RHUL implementation had completed normally.

For large values of RBF width, when $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \simeq 1$, all the data is close together in the kernel's mapping space. This makes it hard for the numerical conjugate gradient algorithm to find a true uphill direction, and calculations involving the direction are swamped by rounding error. Another problem which was found with larger datasets is that the Chunking algorithm keeps on growing the sub-problem until it contains every non-zero Lagrange multiplier, or runs out of memory. Whilst an algorithm to find the correct minimum in $R_{\text{struct}}^{\text{upper bound}}$ was developed, and which worked on some datasets, the numerical problems with the RHUL software led to a search for a more robust SVM training algorithm.

### 4.3.4 Summary

In summary, this section has shown that an upper bound on the test error can be predicted directly from the training data, and that there is typically a local minimum in this value, as a function of the RBF width, close to the true minimum in test error. Experiments showed that it is generally possible to find this local minimum automatically, allowing an optimal value of RBF width to be selected without manual intervention. This promised the ability to run large numbers of SVM exper-

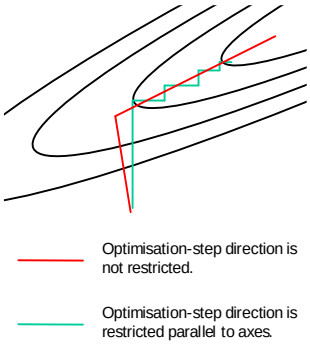



**Figure 4.10:** Restrictions on the step direction during optimisation are generally a disadvantage.

iments without having to choose any arbitrary parameters, but problems with the Chunking/QP algorithm for difficult-to-separate data sets, suggested the need for a more robust approach to training.

## 4.4 Sequential Minimal Optimisation

Sequential Minimum Optimisation (SMO) was developed by Platt[110] to solve the optimisation problem required when training an SVM. Until then, all proposed solutions to this optimisation required a numerical QP implementation, which can be difficult to obtain and use. Platt discovered that instead of optimising hundreds or thousands of Lagrange multipliers at once, it was possible and efficient to perform the optimisation by sequentially optimising the smallest number of multipliers necessary to maintain the constraints. In the case of the SVM optimisation, with one equality constraint, this minimum number of multipliers is just two.

One might expect a high cost due to this heavy restriction in the allowed direction at each step, such that the optimisation would be very slow. An appropriate two dimensional analogy would be optimisation of a function with a strong diagonal principal direction when restricting optimisation step direction to be parallel to one of the axes (figure 4.10.) With SMO this disadvantage appears to be irrelevant for two reasons. First, the heuristics, which choose the pair of parameters to optimise,

rarely result in the optimisation getting stuck in troughs whose principal direction is at a strong angle to all the allowed step directions. Second, since the optimisation is only performed on two parameters at each step, it is possible to analytically and very accurately calculate the optimal point along each step. This is in comparison to numerical QP, where numerical accuracy problems often lead to very inaccurate estimates of the optimal point at each step, which hinders fast convergence, and can prevent the minimum being found at all.

The original SMO algorithm is shown in figure 4.11—there are several features that warrant additional comment.

1. A KKT violation (see equations 4.9 and 4.10,) and the magnitude and sign of the relevant KKT error, $E_i$, allows the SMO heuristics to decide which pair of multipliers to optimise next.

2. For efficiency, the values of $E_i$ are cached for all unbound multipliers, i.e. $\forall i : 0 < \alpha_i < C$. They are then cheaply updated every time a pair of multipliers is changed. There is no cheap update of $E_i$ for bound multipliers, they need to be completely recalculated each time multipliers change.

3. The optimal value with respect to a pair of Lagrange multipliers, is derived from the full Lagrangian (equation 4.6.) First, the terms in $\alpha_i$ and $\alpha_j$ $(i \neq j,)$ are pulled out while keeping the rest constant:

$$\begin{aligned}\Psi(\alpha_i, \alpha_j) &= \alpha_i + \alpha_j - \frac{1}{2}K_{ii}\alpha_i^2 - \frac{1}{2}K_{jj}\alpha_j^2 - sK_{ij}\alpha_i\alpha_j \\ &\quad -y_i\alpha_i v_i - y_j\alpha_j v_j + \Psi_{\text{const}} \end{aligned} \tag{4.14}$$

where

$$\begin{aligned} K_{ij} &= K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ v_k &= \sum_{q \neq i,j} y_q \alpha_q K_{kq} \\ &= f^{\text{old}}(\boldsymbol{x}_k) + b^{\text{old}} - y_i\alpha_i^{\text{old}}K_{ik} - y_j\alpha_j^{\text{old}}K_{jk} \\ s &= y_i y_j \in -1, 1 \end{aligned}$$

Main Function:

Repeatedly choose first of pair of Lagrange multipliers to jointly optimise.

- For every Lagrange multiplier $i$:
  - Call examine-example($i$)
- Do:
  - Do:
    - For every non-bound Lagrange multiplier $i$:
      - Call examine-example($i$)
  - Until no more changes
  - For every Lagrange multiplier $i$:
    - Call examine-example($i$)
- Until all Lagrange multipliers satisfy KKT conditions, equation 4.9

Function: examine-example $i$:

Repeatedly choose second of pair of Lagrange multipliers to jointly optimise.

- If KKT condition error $E_i$ does not exceed tolerance: return no change
- Find the multiplier, $j$, whose KKT condition is maximally broken in the opposite direction to the $i^{\text{th}}$ multiplier. i.e. $\text{argmax}_j |E_i - E_j|$
- Call take-step($i, j$)
- If successful: return changed
- Else:
  - For every other Lagrange multiplier $j$:
    - Call take-step($i, j$)
  - If successful: return changed
- Return no change

Function: take-step $i$ $j$:

Attempt to analytically find jointly optimal position of pair of Lagrange multipliers $(\alpha_i, \alpha_j)$.

- Since there is a joint equality constraint on $\alpha_i$ and $\alpha_j$, calculate everything in terms of $\alpha_j$
- Find limits of $\alpha_j$ due to inequality constraints—see figure 4.12a
- If limits allow no movement: return failed
- Find optimal value of $\alpha_j$ w.r.t. objective function, $\Psi$
- Apply limits to $\alpha_j$
- If $\alpha_j$ has not moved much: return failed
- Calculate $\alpha_i$ from $\alpha_j$
- Update bias $b$, and cached KKT condition values $E_i$

**Figure 4.11:** Summary of Platt's[110] original SMO algorithm.

From the equality constraint, $\alpha_i + s\alpha_j = \alpha_i^{\text{old}} + s\alpha_j^{\text{old}} = \text{constant}$, equation 4.14 can be rewritten in terms of just $\alpha_j$. Setting the derivative, $\frac{d\Psi(\alpha_j)}{d\alpha_j} = 0$, finds the stationary point, giving the optimal new position of $\alpha_j$ :

$$\alpha_j = \alpha_j^{\text{old}} - \frac{y_j(E_i - E_j)}{2K_{ij} - K_{ii} - K_{jj}} \tag{4.15}$$

where $E_k = f^{\text{old}}(\boldsymbol{x}_k) - y_k$ are the cached or calculated KKT errors.

$\alpha_j$ and implicit value of $\alpha_i$ are then checked against the inequality constraints (see figure 4.12a) to give the optimal result.

4. For further details, such as the update of the bias $b$, and the cached error values $E_i$, see Platt's chapter[110] in Kearns *et al.*

### 4.4.1 Implementation of the SMO algorithm

An implementation of the SMO algorithm was published by Xianping Ge[61] and he kindly put the code in the public domain, at this author's request. As well as the whole algorithm being easier to understand, Ge's implementation was much simpler than the RHUL code—simple enough to easily port the code to VXL, rather than just writing a wrapper API as was done for the RHUL code. As well as porting the code, a series of technical improvements were made as follows:

**Code Efficiency** The code was rearranged to avoid some unnecessary calculations.

**Shrinking the Active Set** A shrinking heuristic proposed by Joachims[74], and tested by Platt[111], was added to the code. After each full run through every Lagrange multiplier, any multipliers whose equivalent KKT condition are satisfied by a margin, $E_i$, greater than that by which the worst multiplier violates its KKT condition, are moved out of the active set. The inactive set is then ignored until the very end, when at least one iteration though every multiplier takes place.

**Improved Treatment of Numerical Precision** The optimisation can often involve intermediate multiplier values very close to, but not at the bounds. Subsequent steps involving such a multiplier, that might otherwise move the value exactly to the bound, are rejected because the movement is very small. Now the code allows very small movements if they result in the multiplier reaching the bound.

In addition, the ratio of a pair of optimised multipliers can be greater than precision of the floating point representation. In this case, the smaller number is numerically indistinguishable from zero and can profitably be set to zero. The profit comes both during training where bound multipliers need to be considered less often, and during classification, where zero-valued multipliers are no longer support vectors and can be ignored completely.

**Recalculating the Error Cache** As mentioned above, the values of $E_i$ are cached for all unbound multipliers. For multipliers which have not been directly modified recently, the related cached error values can accumulate small numerical rounding errors. This drift is enough to give false results for the optimal Lagrange multipliers. Worse, since the accumulated rounding errors depend on the path through the optimisation problem, the final erroneous position varies depending on the initial values of the multipliers. To deal with this problem, the cached errors are explicitly calculated afresh once per outermost iteration of the SMO algorithm.

**Margin Encroachment Weights** The upper bound, $C$, on the multipliers comes from the weighting given to the encroachment of data examples into the SVM's separating margin. There is no reason why this has to be the same for every example. In order to experiment with biasing an SVM such that positive examples could encroach upon the margin, but negative ones could not, it is necessary to have different upper bounds for positive and negative training data. In terms of programming, it is as easy to generalise this further and allow a different upper bound on each multiplier. In order to do this, the four cases Platt considered when deriving the bounds on

the multiplier $\alpha_j$ during joint optimisation (with respect to $\alpha_i$ and $\alpha_j$) need to be extended to eight—see figure 4.12.

**Adding Support for MEH Calculation** See section 4.4.2 below.

**Caching Kernel Values** See section 4.4.3 below.

### 4.4.2 Adding Support for MEH Calculation

Due to SMO's use of cached KKT errors, and its use of an analytical inner optimising step, it appears at first to be more complicated to modify SMO than the RHUL code to perform the MEH calculation. Following the derivation in equations 4.14 and 4.15 the optimal value of one of a pair of varying multipliers in the MEH calculations is:

$$\alpha_j = \alpha_j^{\text{old}} - \frac{K_{jj} - K_{ii} + 2(E_i - E_j)}{4K_{ij} - 2K_{ii} - 2K_{jj}}$$

where $E_k = f^{\text{old}}(\boldsymbol{x}_k)$ are the cached or calculated MEH KKT errors. For the Gaussian RBF kernel only, $K_{ii} = K_{jj} = 1$, so this becomes identical to the SVM case (equation 4.15.) Again for the RBF kernel only, the update step for the cached errors, turns out to be identical to that already implemented for the SVM optimisation. For the unbound multipliers, the KKT errors are calculated afresh after each step, from equation 4.12. In terms of the Lagrange multipliers, this becomes

$$E_i = 8 \sum_j \alpha_j K_{ij} - 4K_{kk} + D^2 - 4|\boldsymbol{a}|^2$$

The value of $D^2 - 4|\boldsymbol{a}|^2$ is analogous to the bias term, $b$, in the original SVM SMO algorithm. It is updated so that the KKT conditions are satisfied for the recently optimised pair of multipliers. Thus, very little effort is required to convert the original SVM SMO into the new MEH SMO algorithm. The detailed derivation of these results is straightforward but tedious, and has thus been omitted.

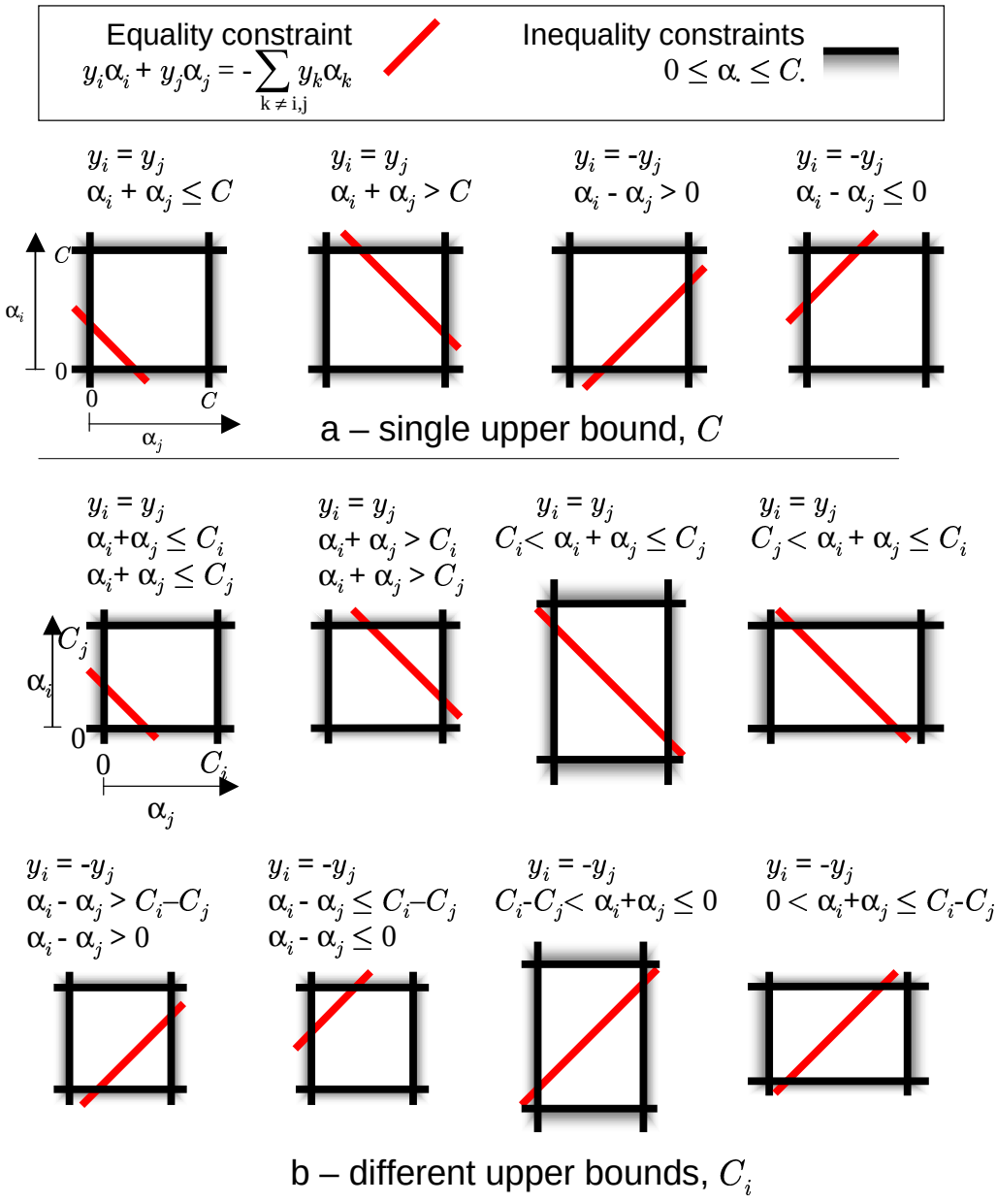



**Figure 4.12:** Calculating the bounds on $\alpha_j$. (a) the four original cases when there is a single upper bounds. (b) the eight cases when each Lagrange multiplier can have a different bound.

Since this derivation is only valid for RBF kernels (or others where $K(\boldsymbol{x}, \boldsymbol{x}) = K_{\text{const}},$) testing the algorithm's correctness was difficult. It was not possible, for example, to use a linear kernel, and test the method on a constructed dataset with known radius. Instead, several easy datasets were tested on a modified version of the RHUL code, and the results were compared with the SMO implementation, and with a simple Matlab implementation based on Gunn's toolbox[65]. The results were found to be similar. For extra confirmation, three known vectors were constructed in the infinite dimensional mapped space (see section 4.1.) By setting the RBF width to a very small number, the mapped vectors should all be mutually separated by 90° on the $\infty$-dimensional unit-radius hypersphere, thus forming an equilateral triangle of known size.

$$\forall\, \boldsymbol{x}_i, \boldsymbol{x}_j : \boldsymbol{x}_i \neq \boldsymbol{x}_j : (\Phi_{\text{RBF}}(\boldsymbol{x}_i) \cdot \Phi_{\text{RBF}}(\boldsymbol{x}_j)) = 0$$

All three implementations agreed with the analytically calculated value of $\sqrt{6}/3$.

Several numerical rounding problems became evident while testing the MEH SMO algorithm, resulting in some of the above numerical precision improvements.

### 4.4.3 Adding Kernel Value Caching to SMO

The majority of the computation time in the SMO algorithm is spent calculating kernel function values. The proportion of time increases with the dimensionality of the training vectors. Since these kernel values are functions of the training vectors only, and do not change during the calculation, they are a perfect candidate for caching. Unfortunately there are very many of them: $\frac{1}{2}n(n-1)$. A computer with 128MiB of available memory can only store the kernel values for a training set of just under 5800 vectors.

### Cache Structure and Algorithm

If the training set is so large that only a proportion of all the kernel values can be stored at any one time, then it is necessary to decide which values to cache, and how to structure the storage. An attempt was made to analyse the kernel value access patterns, but apart from showing that some values are used more than others, it was difficult to infer any useful pattern.

A most recently used cache (or least-recently used expiration) was considered. This is a common caching algorithm that gives fairly good average case performance under the assumption that a more recently used piece of information is more likely to be needed next, than a less recently used piece of data. To store the kernel values $K_{ij}$ for fast lookup, requires storing the index pair $(i, j)$, and the overhead of a binary tree. To store the most recently used order, requires a linked list of index pairs, pointers in each element of the binary tree back into the linked list, and the overhead of a linked list. Assuming 64-bit storage of the kernel values themselves, and 32-bit wide pointers and indices, the total overhead is five times the actual memory devoted to kernel values. With careful use of other data structures such as heaps and hash tables, it might be possible to reduce the overhead to 300%. But this would still require a very good cache hit rate to make it worthwhile, and so was not attempted.

Rather than make the decision of what to cache or discard at every access of a kernel value, it might be more efficient to just attempt to discard items from the cache after every outer-most iteration of the SMO algorithm. Since it is not efficient to store any useful information about individual kernel values, it makes more sense to base a caching strategy on the rows or columns of the kernel matrix, $\boldsymbol{K}$. This is the matrix of kernel function values:

$$\boldsymbol{K} = \begin{pmatrix} K_{11} & K_{12} & \dots & K_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ K_{n1} & K_{n2} & \dots & K_{nn} \end{pmatrix}$$

A new column brought into the matrix would at first contain null-value markers, and

slowly be filled with values as they were next required. A further possible refinement comes from the realisation that if column $i$ is not needed in the cache, then likely neither is row $i$. The cache therefore stores the same rows as it does columns. (Of course, it only needs to store one triangle from this symmetric matrix.) This approach was adopted, but a means of predicting which kernel values, and hence columns of the kernel matrix, are likely to be in demand, was still required.

Each multiplier's KKT condition error, $E_i$, was considered as a predictor of the demand for related kernel values. In order to test this, the sum of the KKT errors, $E_i$ biased by the number of relevant kernel lookups, was compared with the same sum, but of unbiased $E_i$. i.e. compare the biased sum:

$$\sum_{i \in \text{active set}} E_i(\text{at start of iteration}) \cdot \#(\text{Lookups of } K_{\cdot,i})(\text{during iteration})$$

with the unbiased sum:

$$\sum_{i \in \text{active set}} E_i(\text{at start of iteration}) \cdot \overline{\#(\text{Lookups of } K_{\cdot,i})(\text{during iteration})}$$

Over all the iterations these two sums were -4341.114 and -4341.211 respectively. The lack of a substantial difference, shows that multipliers that strongly violate their KKT conditions are no more likely to have their associated kernel values looked up than others.

At each outer-most iteration, the use of the shrinking heuristic to separate the data into the active and inactive sets is an excellent predictor of kernel lookups—if a multiplier is not in the active set, then no associated kernels are looked up. This predictor was therefore chosen for the implementation. The cache was set to discard any kernel column that has been added to the inactive set. If the active set contained more kernel columns than could be stored in the cache, then the uncached columns were selected at random to enter the cache.

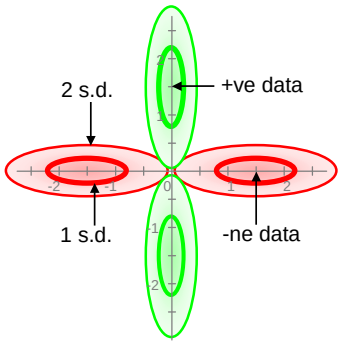



**Figure 4.13:** Artificial test-set generator for various SVM optimisation tests and experiments. Positive data is sampled randomly from the green GMM, and negative data from the red GMM. To produce data with more than two dimensions, the distributions were extended with zero means, and variances of 0.05 (the same as the minor variances in the first two dimensions,) in the third and subsequent dimensions.

### Caching Both SVM and MEH Kernel Values

When the RBF width estimation method is used, each call to the SMO routine will be with a different RBF width, $\sigma$. This means that the kernel values $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp -\frac{|\boldsymbol{x}_i - \boldsymbol{x}_j|^2}{2\sigma^2}$ are different each time. If there has been an small change in $\sigma$, there is no reason to expect that the pattern of kernel values needed will change a large amount. For anything but the lowest dimensional training vectors this kernel calculation is dominated by the calculation of Euclidean distances between the two vectors. Therefore, the cache stores the Euclidean distances and calculates the full kernel value on the fly from the current value of the kernel width. The speed effects of this were tested on 1000 200-D vectors from the artificial dataset illustrated in figure 4.13. The RBF width estimating, SMO optimisation, algorithm was run with the cache preserved between calls to the SMO routines, and with the cache cleared between calls. The effects of re-using the previous iteration's best Lagrange multiplier estimates was also tested. The times for each iteration of the RBF width estimator's call to the SMO routine are shown in figure 4.14. They show that keeping the cache values has a small but consistent effect in reducing the computational times, adding up to a overall improvement in speed of about 15%. Keeping the best Lagrange multiplier estimates has an even bigger effect, resulting in an overall improvement of about 30%.

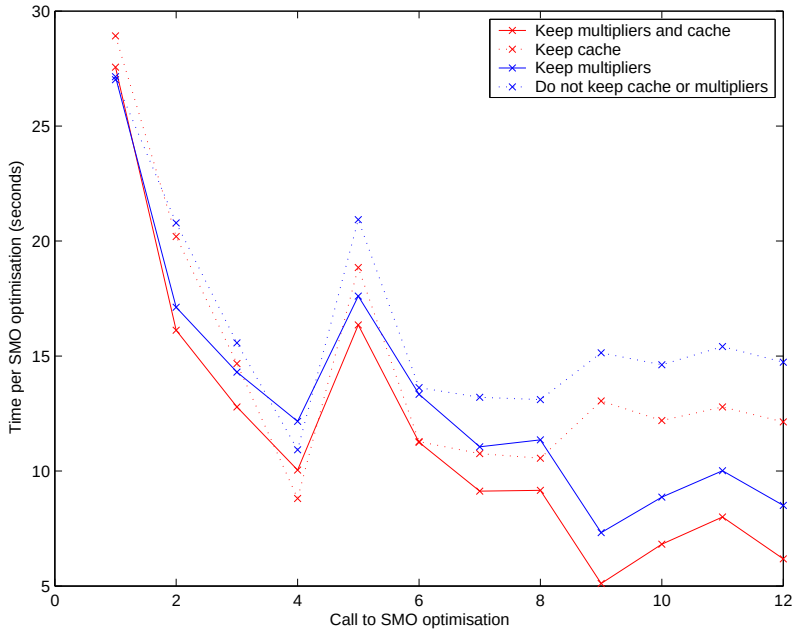



**Figure 4.14:** Time for each iteration of the RBF width estimating algorithm, comparing the effects of preserving the cache, and the Lagrange multipliers between iterations.

### Future Improvements to Caching

Now, with a working and reasonably fast SVM optimiser, no further work was attempted to improve the speed of the SMO method. There are still, however, two obvious improvements that could be made, however.

The first would be to test most recently used caching on the rows and columns of the cached kernel matrix. i.e. discard a whole row/column when a new kernel value is needed. The cache's data structure overhead would not be significant when applied to $k$ row/columns rather than $\frac{1}{2}k(k-1)$ kernel entries.

The second is to use SMO as the quadratic programming module in another SVM method. Joachims's SVMlight[74] method chooses a fixed number of Lagrange multipliers to be optimised using a general-purpose quadratic programming module, with the other Lagrange multipliers staying fixed. The multipliers are chosen according to an approximate estimation of the steepest ascent direction. Normally the fixed size of problems passed to the quadratic programming module is kept small (e.g.

10) to avoid numerical accuracy problems. Since SMO is not as prone to numerical accuracy problems, the number of multipliers jointly optimised can be much larger. This number would be set to use the available memory for storing the entire kernel matrix for the subproblem. In this way the SVMlight algorithm would be improved by being able to jointly optimise more parameters at a time, and the SMO algorithm could effectively defer caching decisions to SVMlight's heuristics.

### 4.4.4 Calculation of Best RBF Width

Several artificial datasets were created for examining SVM training behaviour. In contrast to the RHUL software, wide RBF kernels did not lead SMO to produce non-optimal solutions. Instead the speed of the optimisation slows down, sometimes considerably. The calculation became intractable for very large values of RBF width, and so the experiments were carried out without testing those larger values. For comparison with the RHUL code, a subset of the experiments that provided the data for figure 4.9, were repeated, with only the SVM optimiser being different. The SMO results are shown in figure 4.15.

In general, SMO was able to build good classifiers without the use of a regularisation term added to the Hessian's diagonal. The curves of $R_{\text{struct}}^{\text{upper bound}}$ were less noisy than with the RHUL experiments. Since the aim of these experiments was to find an algorithm that would find the minimum in $R_{\text{struct}}^{\text{upper bound}}$, two extra graphs that represent (along with figure 4.15) the range of its behaviour, are shown in figures 4.16 and 4.17. The training database for the former consisted of a pair of elongated four-dimensional Gaussians arranged in an L shape. Their principle variances were 1, and other variances 0.05. The negative dataset was generated from a spherical Gaussian that enclosed the positive Gaussians. Negative data items were then rejected if they had too high a probability with respect to the positive distribution. The rejection threshold could be altered to make the dataset easier or harder. The training set consisted of 1000 samples, approximately equally split between positive and negative.

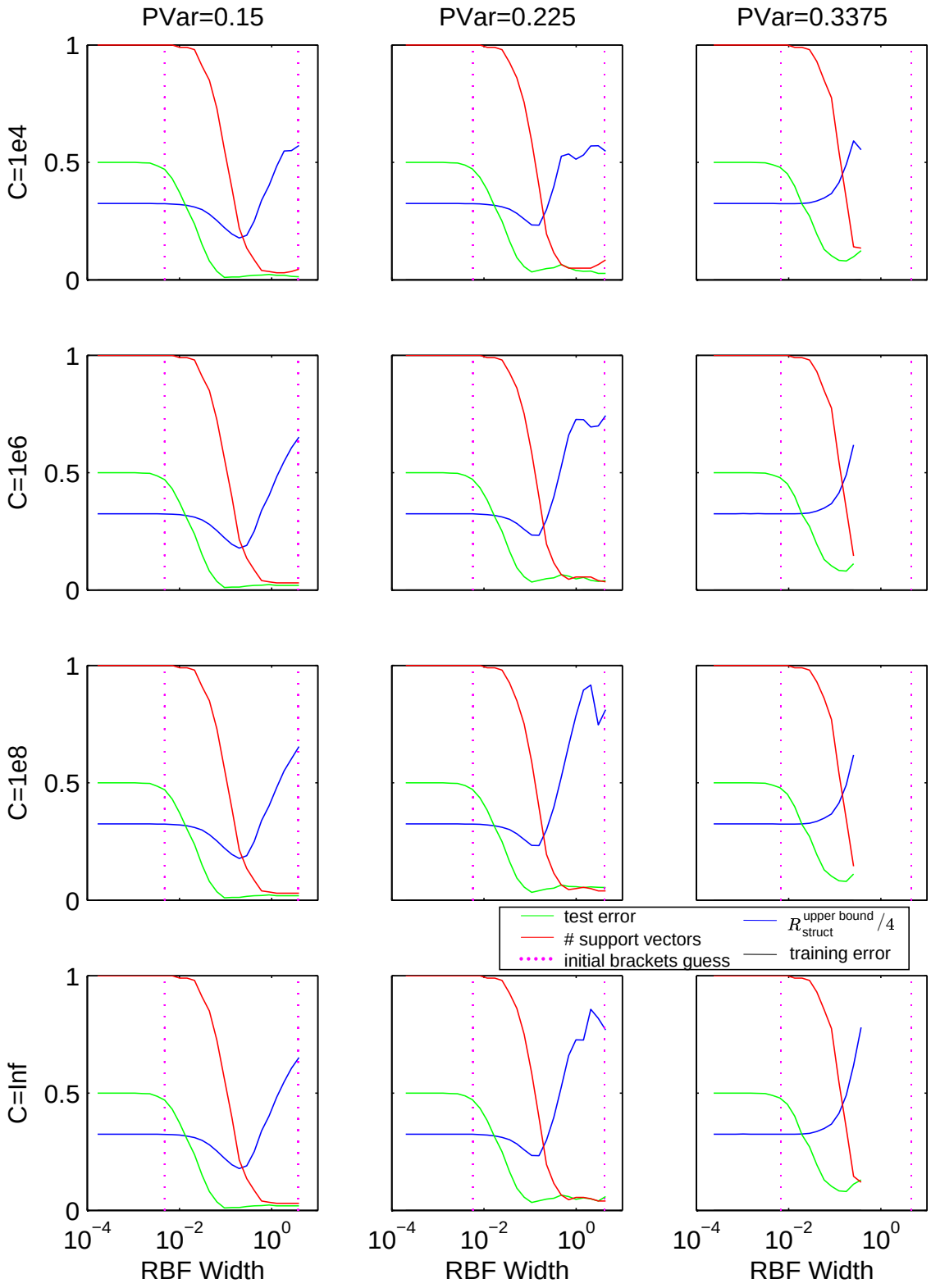



**Figure 4.15:** Test errors, Training errors, $R_{\text{struct}}^{\text{upper bound}}$, for different difficulties of test set (PVar—Principle variance of each mode of data generator—see figure 4.8—harder left-to-right,) different weights on training vectors encroaching into margin ($C$—less encroachment allowed top-to-bottom,) and different RBF widths ($x$-axis of each graph.) Same experiment as first half of figure 4.9 but trained using SMO.

The test set had 5000 samples. The graph shows the results for no upper bound on the Lagrange multipliers (i.e. $C = \infty$,) and negative samples rejected if they were closer to the positive Gaussians than 7.9% of the positive distribution. The graph in figure 4.17 was created using 500 ten-dimensional training samples generated by ten positive Gaussians distributed on different axes close to the origin. A single negative Gaussian enclosed all the positive distributions, and negative samples were again rejected if they were too close to the positive distribution. In this case they were rejected if they were closer than 50% of the positive distribution.

When all these graphs are considered, it can be seen that the $R_{\text{struct}}^{\text{upper bound}}$ minimum is usually reasonably close to the lower bracket value, which is the 10%-ile of the Euclidean distance between all pairs of the training samples. The upper bracket is the maximum Euclidean distance between any pair of training examples. (N.B. The minimum distance would be inadvisable to use as a lower bracket, since two training vectors could be arbitrarily close.) Several experiments found (e.g. figure 4.16) minima outside the initial guessed brackets. However, these initial guessed brackets do give a starting point from which to find a true bracket, and a step size for moving around the RBF width scale.

Standard minimum bracketing algorithms work by moving down the gradient in exponentially increasing step size until the gradient changes direction. That approach will not work here for several reasons. $R_{\text{struct}}^{\text{upper bound}}$ is flat for low values of RBF width, so determining a direction to move initially would be difficult. Choosing large step sizes while looking for the right hand side of the minimum in $R_{\text{struct}}^{\text{upper bound}}$ here would risk trying too large a value of RBF width—since SMO slows down considerably, as the RBF width is increased much past optimal value, a lot of time could easily be wasted. Finally, sometimes the minimum is barely detectable, or even completely missing. In that case, the best that can be done is to bracket the right-hand end of the flat region of $R_{\text{struct}}^{\text{upper bound}}$. An algorithm for finding a true bracket on the minimum of $R_{\text{struct}}^{\text{upper bound}}$ is given in figure 4.18. This bracket is then passed to a standard Brent optimiser, which finds the minimum to within a thousandth of the width of the

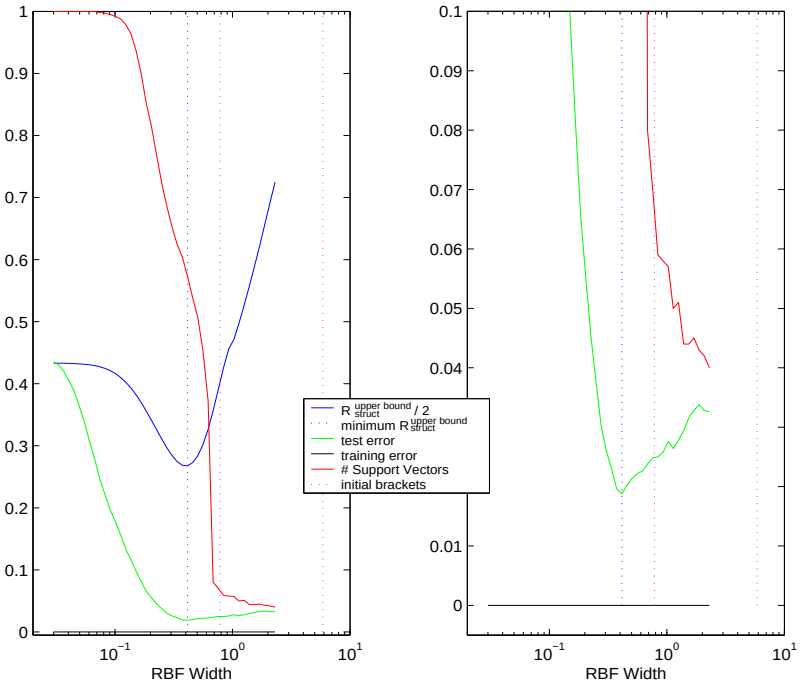



**Figure 4.16:** The test error, $R_{\text{struct}}^{\text{upper bound}}$, training error, and number of support vectors for an SVM as the RBF kernel width is varied, using for an artificial 4-D dataset. The right-hand graph is a magnified version of the left-hand one.

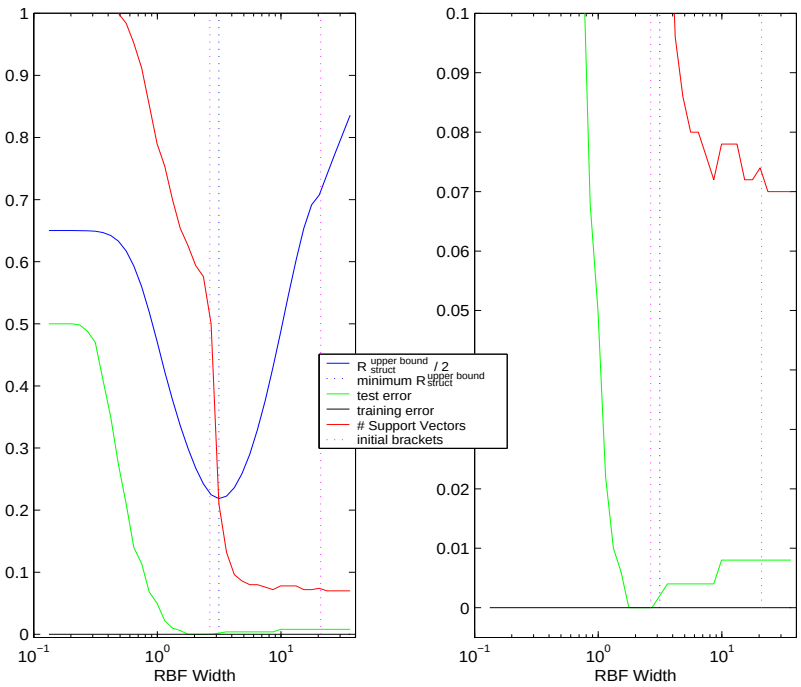



**Figure 4.17:** The test error, $R_{\text{struct}}^{\text{upper bound}}$, training error, and number of support vectors for an SVM as the RBF kernel width is varied, using for an artificial 10-D dataset. The right-hand graph is a magnified version of the left-hand one.

initial guess at the bracket. The implementation keeps track of already calculated values of $R_{\text{struct}}^{\text{upper bound}}(\sigma)$ to avoid recalculating the same value again. The algorithm was successfully tested on several of the datasets described above, where it was always able to find the expected minimum of $R_{\text{struct}}^{\text{upper bound}}$.

## 4.5 Comparison of Chunking and SMO Methods

To compare the speed of the RHUL and SMO implementations, several artificial datasets were generated from the distributions described figure 4.13. The RHUL and SMO implementations were timed both for a preset value for RBF width, and using the RBF width finding algorithm. The results (figure 4.19) show that SMO was usually faster than Chunking. Chunking was occasionally faster in a fixed RBF width experiment, however with large datasets, the Chunking experiments were aborted after over 1500 times as long as the SMO algorithm took to converge. In a much smaller dataset experiment, the RBF width finder failed to converge using Chunking.

These speed results were the final reason to abandon use of RHUL's Chunking method. The very restrictive license conditions, and severe code complexity, make it hard to use from an engineering point of view. The numerical stability of the SVM optimisation, and, in particular, the quality of the estimates of VC dimension, are greatly improved by using the SMO code.

## 4.6 An SVM-Based Face Detector

As a final validation of the SVM methods described above, a face patch classifier was trained using a very large positive and negative training set.

The standard patch feature extractor with plane and histogram normalisation (described in figure 3.1) was used to extract 3563 faces, including some from the pub-

Get initial guess to minimum, and useful step size:

- If less than 200 training vectors:
  - Find distances between all pairs of training vectors
- If more than 200 training vectors:
  - Find distances between a random selection of 10000 pairs of training vectors
- lower guess = 10%-ile of distances
- upper guess = maximum distance
- Step size $s = \sqrt[3]{\text{upper guess/lower guess}}$
- Bracket $[a, b, c] = [\text{lower guess}/s^2, \text{lower guess}/s, \text{lowerguess}]$

Find left hand side of minimum, or the flat region to the left of that:

- Define $R_{\text{struct}}^{\text{upper bound}}(\sigma)$ to be the value of $R_{\text{struct}}^{\text{upper bound}}$ after training an SVM with RBF width $\sigma$
- While gradient is positive (i.e. $R_{\text{struct}}^{\text{upper bound}}(a) < R_{\text{struct}}^{\text{upper bound}}(b)$):
  - Move whole bracket left one step. $[a, b, c] = [a/s, a, b]$
  - If we have gone too far left: Give up and set RBF width to lower guess

Find right hand side of minimum:

- Reduce step size, $s = \sqrt[10]{\text{upper guess/lower guess}}$
- While gradient is non-positive (i.e. $R_{\text{struct}}^{\text{upper bound}}(b) >= R_{\text{struct}}^{\text{upper bound}}(c)$):
  - Move whole bracket right one step. $[a, b, c] = [b, c, c \times s]$

Bracket now surrounds minimum if it exists:

- Call Brent minimiser with bracket $[a, b, c]$

**Figure 4.18:** The RBF Width Finding Algorithm.

**Figure 4.19:** Comparing SVM optimisation times using SMO and Chunking method.

| Dataset size | Data dimensionality | RBF width | Time to optimise (seconds) | |
|---|---|---|---|---|
| | | | SMO | Chunking |
| 200 | 2 | Fixed | 0.08 | 0.23 |
| 200 | 2 | Optimised | 1.45 | 2.68 |
| 200 | 200 | Fixed | 1.32 | 0.84 |
| 200 | 200 | Optimised | 14.25 | Failed after 16.17 † |
| 1000 | 2 | Fixed | 1.32 | 58.94 |
| 1000 | 2 | Optimised | 24.45 | 1283.94 |
| 2000 | 200 | Fixed | 552.33 | 102.74 |
| 2000 | 200 | Optimised | 5579.62 | 5644.10 |
| 10000 | 200 | Fixed | 3591.40 | Aborted after ∼223,000 ‡ |

† The RBF width finding method failed to bracket the minimum.

‡ Experiment was manually aborted.

licly available `XM2VTS` and `BioID` databases. Also included were faces from ISBE's `expression`, `webimageB` and `webimagesC` databases—see appendix B for some examples. Although some of the databases included full face markup, only the eye centres were used. The `CMU` database was not used for training so that it could be used for testing, and allowing comparison with other published results. The training patches were rotated and mirrored as before to create 21,378 examples for the SVM's positive training set.

The negative training set was initialised with 10,000 examples selected at random from the `UWash` database used for non-face examples. After the training of the initial SVM, the `UWash` database was searched sequentially until $n_{\text{select}} = 200$ new false positives were found. These were added to the negative database, and the SVM retrained. This was repeated for 16 refinement iterations with an SVM kernel cache of ∼1.3GiB, which is only enough to store just over half the kernel values. The behaviour of during training is shown in figure 4.20.

The final face detector was then tested on the positive CMU database. As before, searching for true positives was restricted to small regions around the labelled faces for speed. Testing for false positives was done using the full $1.22 \times 10^7$ patches of the same negative database used in the final experiment of section 3.4. The experiment found 63.9% of the faces with 98 false positives (1 in ∼124,000—see the neutral bias point on the ROC curve in figure 4.22.) Some of the false positives are shown in figure 4.21.

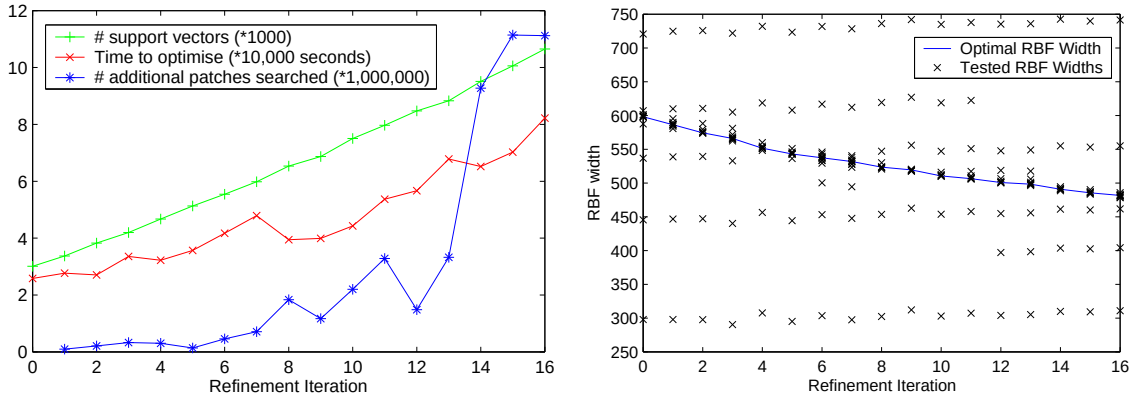
## 4.7 Discussion and Conclusions

### 4.7.1 Comparison with Previously Published Results

The best results of this and the previous chapter are tabulated in figure 4.23, together with those reported by Sung and Poggio[145] with their GMM network, and by Osuna *et al.*[108] with an SVM. It is clear that, although a reasonably high level of performance has been, the experimental results in this thesis are not as good as those reported previously. There are several possible explanations for this:

1. There may still be a problem with the GMM network implementation used here. However, extensive testing suggests each component is working as intended.

2. Neither Sung and Poggio nor Osuna *et al.* describe their training set beyond stating its size. It could be that they had a particularly good training set that included a great deal of normal variation. They used a positive training set of 4150 examples, generated from 1067 real face images. (They did not apply rotations and mirroring to all of their database.) The experiments on GMM networks here only used 733 original faces images. Later, Gong reported[62] that to achieve the published level of performance using the approach of Sung and Poggio, a positive training set of about 3000 original faces was required.




a b

**Figure 4.20:** Training behaviour over each refinement iteration during the training of a patch-based SVM face detector on very large face database. (a) The number of additional negative patches searched to acquire 200 false positives, the number of support vectors, and the time to do full SVM optimisation with RBF width finding. (b) The optimal RBF width, and the values of RBF width tested to get there.

**Figure 4.21:** Some of the false positives found by the patch-based SVM detector.

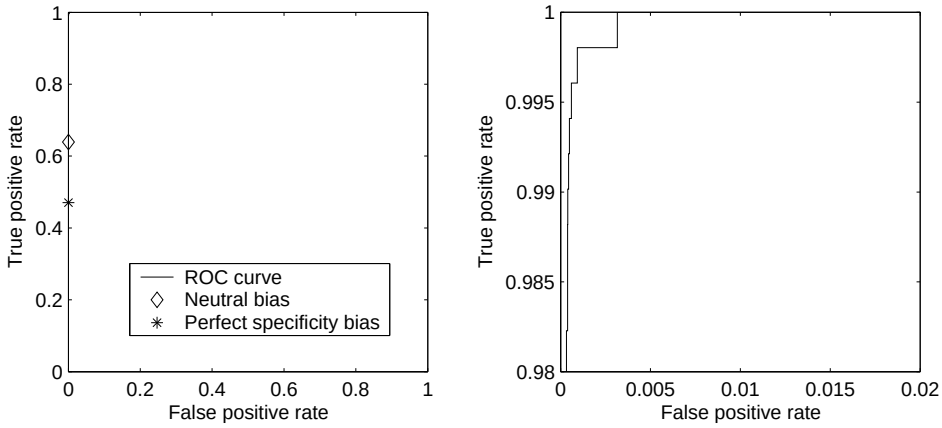



**Figure 4.22:** ROC curve of the patch-based SVM detector after 16 refinement iterations. The right hand graph is a magnified version of the left hand one.

**Figure 4.23:** The best face detection results and comparison with published figures.

| Classifer | Implementation | True positive rate | False positive rate |
|---|---|---|---|
| GMM | Section 4.6 | 54.3% | 1 in ∼32,000 |
| | Sung and Poggio[145] | 98.3% | 1 in ∼1,257,000 |
| SVM | Section 3.4 | 63.9% | 1 in ∼124,000 |
| | Osuna *et al.*[108] | 89.3% | 1 in ∼419,000 |

The experiments on SVMs here used 3563 faces, but most of them were high-quality posed portraits. Getting a better, more variable database would improve the detector performance, particularly the sensitivity.

3. The `CMU` positive test set used in these experiments does not coincide completely with that of Sung and Poggio and Osuna *et al.* `CMU` contains the harder 155-face test set B used by Sung and Poggio and Osuna *et al.*, plus another 342 similar quality faces used by Rowley *et al.*[122] On their test set B alone using a GMM network with a single layer Perceptron, Sung and Poggio reported a true positive rate of 84.6% for a false positive rate of 1 in ∼414,000, and Osuna *et al.* reported a true positive rate of 74.2% for a false positive rate of 1 in ∼269,000.

4. Sung and Poggio acquired ∼43,200 negative training vectors compared to the 25,000 that were used in section 4.6. Osuna *et al.* acquired 50,000 negative training vectors compared to the ∼13,200 that were used in section 4.6. Running the experiments for longer, and acquiring more examples would probably improve the specificity.

5. One false positive (or true positive) patch is likely to have positively classified neighbours. Neither Sung and Poggio, nor Osuna *et al.* describe a method for reducing multiple hits to a single one. (Rowley *et al.* described a module for their neural network based face detector, which suppresses overlapping hits so that only the best candidate is detected.) However, both Sung and Poggio's

and Osuna *et al.*'s systems indicated hits by marking the detected face with a rectangle. They then manually checked the images for hits and false positives. It could be that they only reported distinct false positives. In view of this, the results of the final SVM experiment (section 4.6) were examined. Of the 98 false positives, 52 could easily have been considered to be marking the same face. If potentially overlapping faces were suppressed as per Rowley *et al.*[122], then there would have been 35 false positives. This would bring the false positive rate to 1 in ∼348,000, which compares very favourably with that reported by Osuna *et al.* Assuming that the clustering of the GMM network's false positives is similar to that of the SVM's, this would also account for some of the difference between the results of section 3.4 and those of Sung and Poggio.

### 4.7.2 Further Work

The optimal value of RBF width behaves very predictably, from one refinement iteration to the next. Judging by figure 4.20b, up to half of the 12 RBF width values that were typically tested could be ignored after the first few refinement iterations.

The field of statistical classifiers is still evolving rapidly. Some other techniques such as cascaded-SVM[121] and cascaded-AdaBoost[161] have been shown to be much faster than the full SVM on the face detection problem. This author's initial attempts to reproduce Romdhani *et al.*'s cascaded-SVM suggested that picking the bias term for each level of the cascade is harder than their paper might lead one to believe. However the promised speed improvements would be valuable.

### 4.7.3 Discussion

The main purpose of the work described in this chapter was to provide a baseline for object detection performance, and develop a classifier that could be used in a completely automated way in later experiments with more sophisticated appearance

models. SMO-trained SVMs with optimal RBF width finding were chosen for this role due to their classification performance, reliability and ease of use.

This author has been unable to find any other published descriptions of the difficulties in choosing the RBF width automatically. It could be that other authors have not found it necessary to avoid manually investigating the behaviour of their SVMs with each new training set. Indeed it is generally considered a good idea to experiment with any new dataset in order to check for problems such as missing data. It may also be the case that the difficulties found here are mainly due to some pathological aspect of the artificial datasets used in producing figures 4.15, etc. However, experience with real datasets confirmed that choosing too large an RBF width led to numerically inaccurate results with the RHUL code, or very slow optimisation with the SMO code.

The initial assumption that the boundary between valid and invalid appearances of a face is necessarily complicated, bears closer scrutiny. That assumption is the motivation for the various mixture model PDF methods, including the GMM network investigated in the previous chapter. However, the optimal SVM has a large RBF width of 481.5. Since the pixel samples in the vector are histogram normalised to $[0, 256)$ this suggests that the boundary is not particularly lumpy, and that if the manifold of pixel values was sliced perpendicular to the axes, the intersection would normally be a circle. However, the mean of the distances between all the positive data is $\sim$1274, with a standard deviation of $\sim$246. The RBF width is thus small compared to the total variation, and so the boundary may not be spherical on a large scale.

# Chapter 5

# Introduction to Appearance Modelling

This short chapter reviews Cootes's *et al.*[32, 52] work on constructing statistical appearance models and matching them to new images using the Active Appearance Model (AAM) search algorithm. The AAM can represent both the shape and texture variability seen in a training set. Despite there being no negative training set, the AAM attempts to learn a model of an object class that is both specific and general. That is, it can only model the appearances of objects from the class, and it can model all possible appearances of objects from the class. The AAM search algorithm is a fast local optimisation for accurately adjusting the appearance model's shape and texture to fit a previously unseen image.

This review serves as a background to subsequent chapters, which describe significant improvements to the AAM method. Together with these improvements, the AAM's ability to locate objects accurately and extract a precise description of the object is key to the performance of the combined AAM-SVM method described later in this thesis.

## 5.1 Constructing Active Appearance Models

The AAM's training set consists of labelled images, where key landmark points are marked on each example object. The training set is usually labelled manually, though automatic methods are being developed (e.g. by Baker *et al.*[9]) AAMs can be thought of as generalisations of Eigen-patches or Eigen-faces[99, 153] in which, rather than representing a rigid region, the region's shape is allowed to deform according to a model.

The numbered label points from each training example constitute an example shape, and they correspond between examples. A shape model is built from these shape examples. First, the mean shape, $\overline{\boldsymbol{s}}$, and best pose transform from each training image to the mean, are iteratively re-estimated until convergence. The pose transforms usually allow for rigid rotation and translation, and isotropic scaling, and the parameters of the $i^{\text{th}}$ example's pose are represented in a pose vector, $\boldsymbol{v}_i$. The example shapes are then transformed by the best fit pose estimates, and their point positions $(s_{\rightarrow}, s_{\uparrow})_j$ are arranged into a shape vector: $\boldsymbol{s} = (s_{\rightarrow 1}, s_{\uparrow 1}, s_{\rightarrow 2}, \dots)$ Next, PCA is applied to the shape vectors. Typically, enough principle components are kept to represent 99% of the variation in the training set. This produces a linear shape model:

$$\boldsymbol{s} = \overline{\boldsymbol{s}} + \boldsymbol{Q}_s \boldsymbol{b}_s + \boldsymbol{r}_s$$

where $\boldsymbol{r}_s$ is the shape residual, and $\boldsymbol{b}_s$ are the shape parameters. $\boldsymbol{Q}_s$ is a matrix whose columns are the unit-length orthogonal eigenvectors of the PCA analysis. The points, reconstituted from $\boldsymbol{s}$, and now called the control points.

The shape parameters $\boldsymbol{b}_s$ can be constrained to values near to the mean. A "limiter" applies these constraints during AAM search, but not during AAM training. The limiter records the size of the eigenvalues for each component during the PCA calculation, and typically restricts each parameter to $\pm 3$ standard deviations. Chapter 8 describes these limiters in more detail.

The shape model (plus pose transform) is fitted to the label points from each training

image. The shape residual is explicitly fixed at zero, constraining the fit to the learnt subspace. A triangulation mesh between the points of the shape model allows the object on each image to be divided up into triangles. Each triangular image segment is then affine warped, so that the three control points defining the triangle are in their mean positions. The process of warping all the segments, warps the whole image patch to the mean shape.

The pixels from each training image, now warped to the mean shape, are in correspondence with the pixels in the other training images. The pixel values are arranged into a texture vector $\boldsymbol{t}_i$. Optionally, the grey-level pose (otherwise known as brightness and contrast) can be taken out in a similar manner to the shape pose, above.

PCA is used again to produce a linear model of the texture variation:

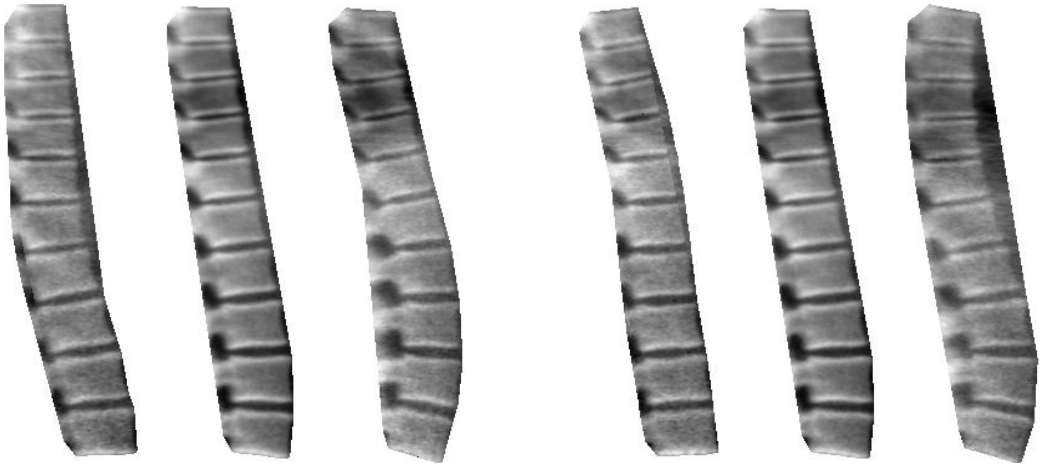
$$\boldsymbol{t} = \overline{\boldsymbol{t}} + \boldsymbol{Q}_t \boldsymbol{b}_t + \boldsymbol{r}$$

where $\boldsymbol{b}_t$ are the texture model parameters, etc. The texture residual, $\boldsymbol{r}$, is often called the AAM residual. Again, the PCA is stopped when the model can explain 99% of variation in the training set. Another limiter is also created for the texture model.

To build a combined model of shape and texture, or appearance model, the best fit shape and texture parameters for each example are concatenated into a combined vector, $\boldsymbol{b}_i = (\omega b_{t,1}, \omega b_{t,2}, \ldots \omega b_{t,k_t}, b_{s,1}, \ldots)$. The weighting, $\omega$, on the shape parameters is to account for the shape and texture being non-commensurate. Typically, $\omega$ is set so that the shape and texture variation make equal contribution to the total variation in the combined vectors. In order to account for correlation between the shape and texture variation (e.g. due to self-shading), another PCA is applied to the combined vectors, to produce the final combined appearance model:

$$\boldsymbol{b} = \overline{\boldsymbol{b}} + \boldsymbol{Q}_b \boldsymbol{p} + \boldsymbol{r}_b$$

where, $\boldsymbol{p}$, are the appearance parameters, etc. The PCA stops after 99%, and a final limiter is created.



$p_1 = -3$ $p_1 = 0$ $p_1 = +3$ $p_2 = -3$ $p_2 = 0$ $p_2 = +3$

**Figure 5.1:** Effect of varying first two parameters ($p_1$ and $p_2$) of a spinal X-ray AAM, by $\pm 3$ standard deviations from the mean. Each image is a reconstruction with all but the specified parameters set to zero. The AAM was built from side views of the human spine, (see figure 6.7a for an original.)

This model can be applied forward to find the parameters, $\boldsymbol{p}$, and pose, $\boldsymbol{v}$, for given label points and image. The model can also be applied backwards, synthesising an image for a given $\boldsymbol{p}$, by generating a texture image from the vector $\boldsymbol{t}$ and warping it using the control points described by $\boldsymbol{s}$, and the transform described by $\boldsymbol{v}$. See figure 5.1 for examples of reconstructed images from an AAM of X-ray images of human spines.

## 5.2 Fitting Active Appearance Models

In order to fit an image, when there are no label points, another algorithm is needed—AAM search. Treating the problem as a standard function fitting, our aim is to get the model reconstruction looking as much as possible like the object in the test image. Thus the texture residual's squared magnitude, $E(\boldsymbol{p} + \delta\boldsymbol{p}) = |\boldsymbol{r}(\boldsymbol{p} + \delta\boldsymbol{p})|^2$, should be pushed towards zero, by manipulating the model parameters $\boldsymbol{p} + \delta\boldsymbol{p}$, and pose (which we ignore here for clarity.) Using Taylor expansion about the current state of the model, $\boldsymbol{p}$, we explicitly set the next measurement of $E$ to be zero.

$$
\begin{aligned}
E(\boldsymbol{p}+\delta\boldsymbol{p}) &= E(\boldsymbol{p}) + \delta\boldsymbol{p}\frac{\partial E}{\partial \boldsymbol{p}} + O(|\delta\boldsymbol{p}|^2) \\
0 &= E(\boldsymbol{p}) + \delta\boldsymbol{p}\frac{\partial E}{\partial \boldsymbol{p}} + O(|\delta\boldsymbol{p}|^2) \\
&= \boldsymbol{r}^T(\boldsymbol{p})\,\boldsymbol{r}(\boldsymbol{p}) + 2\,\delta\boldsymbol{p}\,\boldsymbol{r}(\boldsymbol{p})\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}} + O(|\delta\boldsymbol{p}|^2) \\
\\
\delta\boldsymbol{p} &= -\frac{1}{2}\boldsymbol{r}^T\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}^T\left(\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}^T\right)^{-1} + O(|\delta\boldsymbol{p}|^2)
\end{aligned}
$$

Setting the error to zero gives us the estimate:

$$
\widehat{\delta\boldsymbol{p}} = -\frac{1}{2}\boldsymbol{r}^T\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}^T\left(\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}^T\right)^{-1} \tag{5.1}
$$

Thus, given the current model, we need to adjust the parameters by $\widehat{\delta\boldsymbol{p}}$ to set the error, $E$, to zero. Due to the second-order errors (from the expansion truncation, and the pseudo-inverse of $\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}$) a single step is unlikely to be enough. So the model is updated, the residual re-estimated, and $\widehat{\delta\boldsymbol{p}}$ recalculated. This process is iterated until convergence, which usually takes less than 8 steps when starting from the mean position $\boldsymbol{p} = \boldsymbol{0}$.

In normal optimisation situations, the down-hill direction, $-\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}$, needs to be re-estimated each iteration. The key insight that led to the AAM algorithm, was that each AAM search is solving largely the same optimisation problem. Therefore $\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}$ (or preferably its pre-calculated pseudo-inverse) can be learnt.

The AAM method uses a linear model for the optimisation's update step. Given a combined appearance model, fitted to some labelled training images, we can perturb the parameters by small amounts, $\delta\boldsymbol{p}$ and $\delta\boldsymbol{v}$. The control points of the shape model, are therefore moved slightly. We then warp the training image to the mean shape, using these modified control points, and sample the pixels into an texture vector. The difference $\delta\boldsymbol{t}$ between this sampled texture vector and the texture predicted by the slightly modified model, is the texture residual, $\boldsymbol{r}$, for this perturbed case.

Using regression, we can finally learn a linear relationship, $\boldsymbol{R}$, between lots of model perturbations, and the resulting texture differences.

$$\delta \boldsymbol{p} = \boldsymbol{R}\, \delta \boldsymbol{t}$$

where $\boldsymbol{R}^T = -\frac{1}{2}\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}^T \left(\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}^T\right)^{-1}$ from equation 5.1 above. We can now simply calculate the AAM-search update step as simple matrix-vector multiplication:

$$\widehat{\delta \boldsymbol{p}} = \boldsymbol{R}\, \boldsymbol{r}(\boldsymbol{p})$$

The search and regression can be trivially extended to adjust the pose parameters at the same time. One important detail deals with the case where the estimated change, $\widehat{\delta \boldsymbol{p}}$, leads to increasing error, $E$. This can happen because the linear relationship, $R$, breaks down, with an increasing likelihood of failure as $|\widehat{\delta \boldsymbol{p}}|$ increases. So, when the error increases, the step $\alpha\, \widehat{\delta \boldsymbol{p}}$ is tried instead, with $\alpha = \frac{1}{2}$. If the error is still worse, $\alpha$ is halved again. This line search stops after a few iterations, at which point the AAM can be assumed to have converged.

It is straight-forward to build different resolution AAMs from the same training data. A multi-resolution AAM search can then start at a coarse scale first, where it quickly and robustly finds an optimal if imprecise fit. A few iterations at each of the higher resolution layers (usually separated by a scale factor of two) can then refine the fit.

## 5.3 Related Work

The AAM approach is similar to the Active Blobs of Sclaroff and Isidoro[134]. Built from a single image, the Active Blob's shape model is based on elastic vibration modes, and its texture model on planar lighting changes. However, the projection into a fixed shape space and using the residual to drive the model parameters was performed similarly to AAMs.

One other closely related work is Jones *et al.*'s Morphable Models[159, 76]. Their

shape model is a dense warp field, trained using correspondences estimated from an optical flow analysis. The texture model is a rectangular image based on either linear combinations of training examples, or a PCA model of the training images. The multi-resolution update algorithm uses a general stochastic gradient descent algorithm to minimise the residual norm (in the image frame) by modifying the shape and texture parameters.

Appearance models and AAMs have been shown to be powerful tools for medical image interpretation[95, 18] and face image interpretation[32].

## 5.4 Summary

The AAM is a statistical model of the appearance of objects in a training set. It represents the appearance variation as the non-linear composition of linear models of shape, texture, and of the correlations between the shape and texture. AAM search fits a model to the object in an unseen image using a small number of iterations of fast, pre-computed, update step.

Further details of the AAM method can be found in Cootes's technical report[30].

# Chapter 6

# Improving AAM Performance Using Local Image Structure

This chapter builds on the AAM method described in the previous chapter, introducing the "Texture AAM." Rather than just recording the intensities at each pixel, a local structure tuple is recorded. A new representation of texture, based on the Harris corner detector[68], is included in the local structure tuple, leading to a significant improvement in the accuracy and robustness of AAM search. Two different objects are used for experimentation in this chapter, images of faces, and X-ray images of the spine.

The work of this chapter was published at IPMI 2003[135] and at MIUA 2003[136].

## 6.1 Introduction and Motivation

It is known[33] that in AAM face recognition tasks, poor AAM search convergence is the single most significant source of poor overall recognition performance. Similarly, it is reasonable to expect poorly converged AAM searches to undermine any AAM-based object detection scheme.

Previous work by Cootes and Taylor[41] suggested that there was a significant improvement to be made in AAM search performance, by going beyond the modelling of texture as a vector of pixel intensities, and using non-linear gradients instead. This and the next chapter examine several image-feature based models of texture, including corners, edges, and Cartesian Differential Invariants (CDI). In particular, features that can be sparsely identified using non-maximal suppression were interesting, in the initial hope that sparse feature detection might lead to fast hypothesis generation and rejection of negatives.

When building models of the appearance of objects it is advantageous to choose a representation of the image structure which can capture the features of interest in a way that allows a reliable comparison between model and image, and is invariant to the sorts of global transformation that may occur. For instance, when building statistical appearance models[32, 153] it is common to represent the image texture by a vector of intensity values sampled from the image, normalised by a linear transform so as to be invariant to global changes in brightness and contrast. By sampling across the whole region, all image structures are represented and all pixels treated equally (though the statistical analysis will then typically favour pixels in some regions over others, as dictated by the data.) However, models based on raw intensity tend to be sensitive to changes in conditions such as imaging parameters or lighting. Thus, models built on one database may not perform well on images taken under different conditions. Also, intensity models do not explicitly distinguish between areas of noisy flat texture and real structure, and thus may not lead to accurate fitting in AAM search.

Edge based representations tend to be less sensitive to imaging conditions than raw intensity measures. Therefore, an obvious alternative to modelling the intensity values directly is to record the local image gradient in each direction at each pixel. Although this yields more information at each pixel, and at first glance might seem to favour edge regions over flatter regions, it is only a linear transformation of the original intensity data. Since AAM building involves applying a linear PCA to the

samples, the resulting model will be almost identical to one built from raw intensities, apart from some effects around the border where computing the gradients includes some background information into the model.

It is useful to think about the rest of this chapter as using *texture preprocessors* which take an input image, and produce an image of tuples representing various aspects of local structure. This local structure tuple can include such things as edge orientation, corner strength, etc. When sampling the image to produce a texture vector for a model, instead of sampling $n$ image intensity values from the original image, this approach samples all the values from each $m$-tuple at $n$ sample locations, to produce a texture vector of length $nm$. This evolution from intensity AAM through single descriptor local structure AAM to multi-descriptor local-structure (or "Texture") AAM is illustrated in figure 6.1.

## 6.2 Related Work

Active Shape Models and Active Appearance Models were developed in this lab and have been widely used. However, almost all of that work has been on normalised intensity or (for ASM profile models) linearly normalised gradient models.

Eigen-faces[153] model the statistics of the intensities in a region of an image, and have been widely used for object recognition. Eigen-faces attempt to model shape variation as a change in texture, usually resulting in a smearing of destinctive texture. Moghaddam and Pentland preprocessed images to build Eigen-faces of smoothed Canny edges[99]. In the image registration community, edge maps are widely used[152]. However, they tend to use either linearly normalised gradients, squared gradients or non-maximally suppressed edges (all pixels other than those thought to be exactly on the edge are set to zero.)

Hond and Spacek[70] used edge orientation histogram images for face localisation and

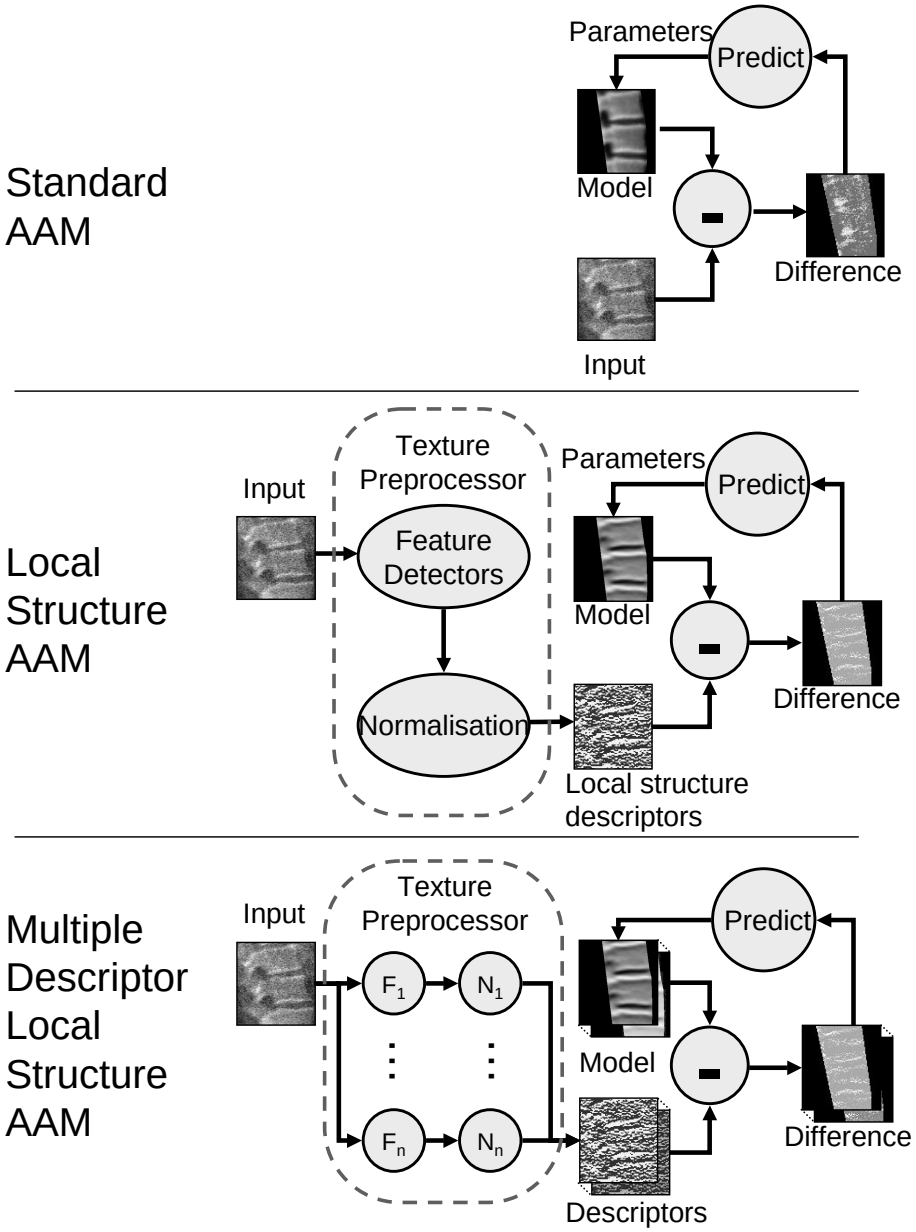



**Figure 6.1:** The evolution of intensity to texture AAMs as used in this chapter.

recognition. One of the structure descriptors described in this chapter can be thought of as a weighted version of edge orientation, in which strong edges are given more weight than weak edges. Rather than use a histogram of large blocks of the image as in Hond and Spacek's work, here the edge structure is modelled at every pixel.

Bosch *et al.*[18] used non-linear image normalisation to deal with the strongly non-symmetric pixel-intensity distribution of ultrasound images of the heart. Converting the intensities from an exponentially-decaying distribution to a Gaussian gave significantly improved AAM matching results.

Several authors have attached feature detectors to points on a PDM. This PDM can been automatically generated created using elastic variation of a single image[81]. A manually trained, but statistically learnt PDM can be used with profiles[36], and Gabor jets[91]. In all these approaches there is no dense model of texture. More importantly, the feature detector locations, and effect on the shape model, has been set by humans rather than learnt.

Kittler *et al.*[79] demonstrated that the choice of pixel-value normalisation could have a significant effect on a face verification task. Overall, histogram normalisation tended to perform well.

Stegmann and Larsen[142], developed a multiple-descriptor AAM (using the adjective "multi-band" instead.) Starting with RGB images, they reduced these to hue and value descriptors. In this limited sense, multi-descriptor AAMs were first reported with the some of the original AAM experiments[51] which used RGB models. Stegmann and Larsen, also added a grey-level gradient-magnitude descriptor, but did not attempt to normalise the relative scales of the three descriptors. Overall they report an improvement over normal grey-scale and RGB AAMs.

## 6.3 Local Structure Detectors

### 6.3.1 Non-linear Transforms

As noted earlier, the texture preprocessor needs to be non-linear to make a significant difference to a linear PCA-based model. If one restricts the choice of preprocessor to those whose magnitude reflects the strength of response of a local feature detector, then it would be useful to transform this magnitude $m$ into a reliability measure. Initially a sigmoid function is used as this non-linear transform:

$$f(m) = \frac{m}{m + \overline{m}} \tag{6.1}$$

where $\overline{m}$ is the mean of the feature response magnitudes $m$ over all samples. This function has the effect of limiting very large responses, preventing them from dominating the image. Any response significantly above the mean gives similar output. Also, any response significantly below the mean gives approximately zero output. This output behaves like the probability of there being a real local structure feature at that location.

### 6.3.2 Gradient Structure

The first local structure descriptor to be experimented on is gradient orientation. Early work on non-linear gradient orientation was described by Cootes and Taylor[41]. They used an image gradient $\boldsymbol{g} = (g_x \quad g_y)^T$, calculated at each point, giving a 2-tuple texture image for 2D input images. The magnitude $|\boldsymbol{g}|$ can be transformed using equation 6.1, while preserving the direction. This is followed by the non-linear normalisation step to give

$$\boldsymbol{g}_n = \frac{(g_x \quad g_y)^T}{|\boldsymbol{g}| + \overline{|\boldsymbol{g}|}} \tag{6.2}$$

### 6.3.3 Corner and Edge Structure

This author had observed that corners of the eyes and mouth were sometimes badly matched by gradient and intensity AAMs. Corners are well known as reliable features for corresponding multiple images[152] and, in applications such as medical morphometry[107], accurate corner location is important in diagnosis.

Harris and Stephens[68] describe how to build a corner detector, extending work by Moravec[100]. They construct a local structure descriptor by calculating the sum of square differences between an image patch and itself as one is scanned over the other. This local image energy $E$ is zero at the patch origin, and rises faster for stronger textures.

$$E(x,y) = \sum_{u,v} [I(u+x, v+y) - I(u,v)]^2$$

To enforce locality and the consideration of only small shifts, they added a Gaussian window $w(u,v)$, and then made a first order approximation:

$$E(x,y) = \sum_{u,v} w(u,v) \left[ x\frac{\partial I}{\partial u}(x,y) + y\frac{\partial I}{\partial v}(x,y) + O(x^2, y^2) \right]^2$$

Expanding the square-term gives

$$E(x,y) \quad = \quad Ax^2 + 2Cxy + By^2 \quad = \quad (x \;\; y)\boldsymbol{M}(x \;\; y)^T \tag{6.3}$$

where $\boldsymbol{M} = \left(\begin{smallmatrix} A & C \\ C & B \end{smallmatrix}\right)$, $A(x,y) = \left[\frac{\partial I}{\partial u}\right]^2 \otimes w$, $B(x,y) = \left[\frac{\partial I}{\partial v}\right]^2 \otimes w$, $C(x,y) = \frac{\partial I}{\partial u}\frac{\partial I}{\partial v} \otimes w$, and $w(u,v) = \exp -(u^2 - v^2)/2\sigma^2$.

The eigenvalues $\alpha$,$\beta$ of $\boldsymbol{M}$ characterise the rate of change of the sum of squared differences function as its moves from the origin. It is also possible to think about $\alpha$ and $\beta$ as the principal curvatures of the local image autocorrelation function—see figure 6.2. Since $\alpha$ and $\beta$ are the principal rates of change, they are invariant to rotation. Without loss of generality, the eigenvalues can be rearranged so that $\alpha >= \beta$. The local structure at each point in the image can be described by these two values. As shown in figure 6.3, low values of $\alpha$ and $\beta$ imply a flat image region.

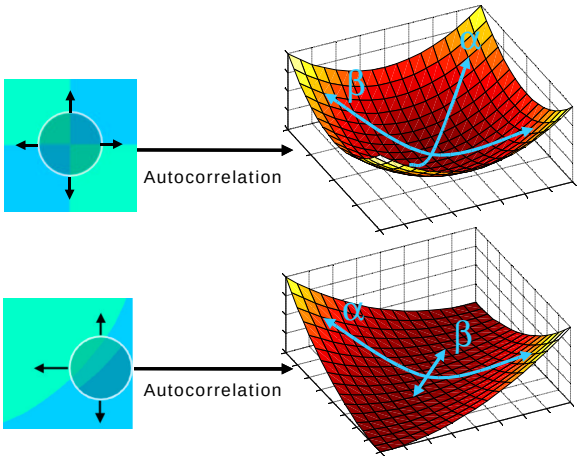



**Figure 6.2:** $\alpha$ and $\beta$ measures the strength of the principal local variations in texture.

A high value of $\alpha$ and low value of $\beta$ imply an image region flat in one direction, but changing in another, i.e. an edge. High values of both $\alpha$ and $\beta$ imply a region that is not flat in any direction, i.e. a corner. (Perhaps it is more precise to axiomatically denote areas with high $\alpha$ and $\beta$ to be a "corner," as this avoids the semantic problem that image corners may not correspond to actual corners in the subject but, for example, to points of high intensity in a low intensity background such as specular reflections.)

At this point Harris and Stephens identified individual points of interest by looking for local maxima in $\det \boldsymbol{M} - k[\mathrm{tr}\, \boldsymbol{M}]^2$. We leave their approach here, except to note that useful measures derived from $\alpha$ and $\beta$ can be found without actually performing an eigenvector decomposition.

$$\mathrm{tr}(\boldsymbol{M}) = A + B \qquad \det(\boldsymbol{M}) = AB - C^2$$

### 6.3.4 Developing Measures of Cornerness and Edgeness

It would be useful to have independent descriptors of edgeness and cornerness. To force $\alpha$ and $\beta$ into an independent form, one can take the vector $(\alpha \;\; \beta)^T$ and double the angle from the $\alpha$ axis, as in figure 6.4.

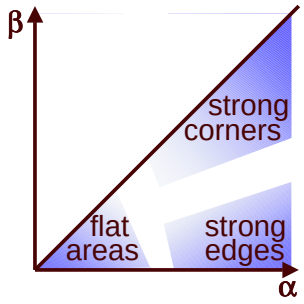



**Figure 6.3:** How $\alpha$ and $\beta$ relate to cornerness and edgeness.

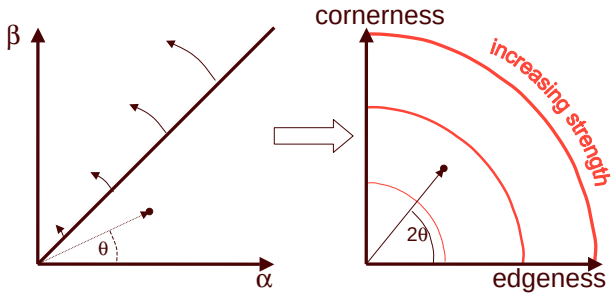



**Figure 6.4:** Making cornerness independent of edgeness by doubling angle from axis.

It is possible to calculate the cornerness, $r$, and edgeness, $e$, defined this way, without explicitly having to calculate an eigenvector decomposition:

$$\tan\theta = \frac{\beta}{\alpha} \quad \Rightarrow \quad \sin\theta = \frac{\beta}{\sqrt{\alpha^2+\beta^2}} \quad \text{and} \quad \cos\theta = \frac{\alpha}{\sqrt{\alpha^2+\beta^2}}$$

$$\begin{aligned} \sin 2\theta &= 2\sin\theta\cos\theta \\ &= \frac{2\alpha\beta}{\alpha^2+\beta^2} \\ &= \frac{2\det \boldsymbol{M}}{\operatorname{tr}^2 \boldsymbol{M} - 2\det \boldsymbol{M}} \end{aligned} \qquad \begin{aligned} \cos 2\theta &= 2\cos^2\theta - 1 \\ &= \frac{2\alpha^2}{\alpha^2+\beta^2} - 1 \\ &= \frac{\sqrt{(\operatorname{tr}^2 \boldsymbol{M} - 4\det \boldsymbol{M})\operatorname{tr}^2 \boldsymbol{M}}}{\operatorname{tr}^2 \boldsymbol{M} - 2\det \boldsymbol{M}} \end{aligned}$$

$$\begin{aligned} r &= (\alpha^2+\beta^2)\sin 2\theta \\ &= 2\det \boldsymbol{M} \\ &= 2AB - 2C^2 \end{aligned} \quad (6.4) \qquad \begin{aligned} e &= (\alpha^2+\beta^2)\cos 2\theta \\ &= \operatorname{tr} \boldsymbol{M}\sqrt{\operatorname{tr}^2 \boldsymbol{M} - 4\det \boldsymbol{M}} \\ &= (A+B)\sqrt{(A-B)^2+4C^2} \end{aligned} \quad (6.5)$$

Figure 6.5 shows some spinal images after being preprocessing to produce a sigmoidally-normalised corner image, edge image, and $x$- and $y$-gradient images. Note that this edgeness measure is different from the gradient measure, by being independent of edge direction. Nor is edgeness $e$ merely the magnitude of some directional edge vector. It is not in general possible to give a direction to the edge data. For example, there is a strong edgeness on a straight white line on a black background, but of what direction? The image gradient on the line is 0 in all directions.

As with the gradient structure descriptors, the edgeness and cornerness descriptors

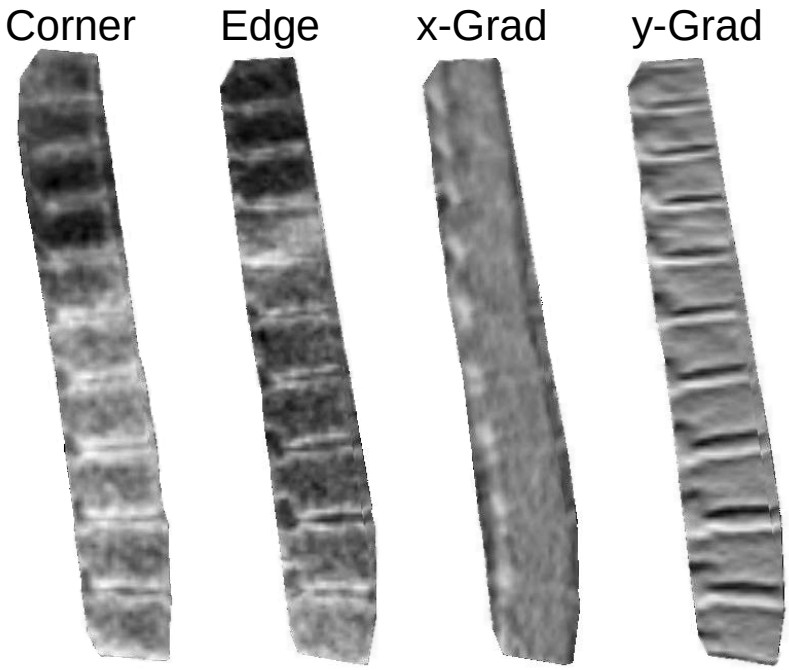



**Figure 6.5:** Images of the spines after processing by the feature detectors. (See figure 6.7a for an original spine image.)

are sigmoidally normalised before being placed in the AAM texture vector.

### 6.3.5 Implementation Issues

This description of this work as texture preprocessors for AAMs is slightly oversimplified. Ideally, the texture processing has to happen after an input image has been warped to the mean shape (figure 6.6.a) so that the effects of any pixel on the resulting texture vector are consistent. AAMs currently use the triangulation between the points of a PDM to warp faces to the shape-free space. This triangulation is not defined outside the boundary of the PDM and so none of the background can be warped into the shape-free space. Therefore, it is necessary instead to perform the inverse warp on the sampling grid, and sample in the original image.

Cootes and Taylor's first implementation of gradient AAMs explicitly warped the gradient sampler from the shape free space into the original image. Their gradient sampler was unsmoothed, and each direction of each sample consisted of a central

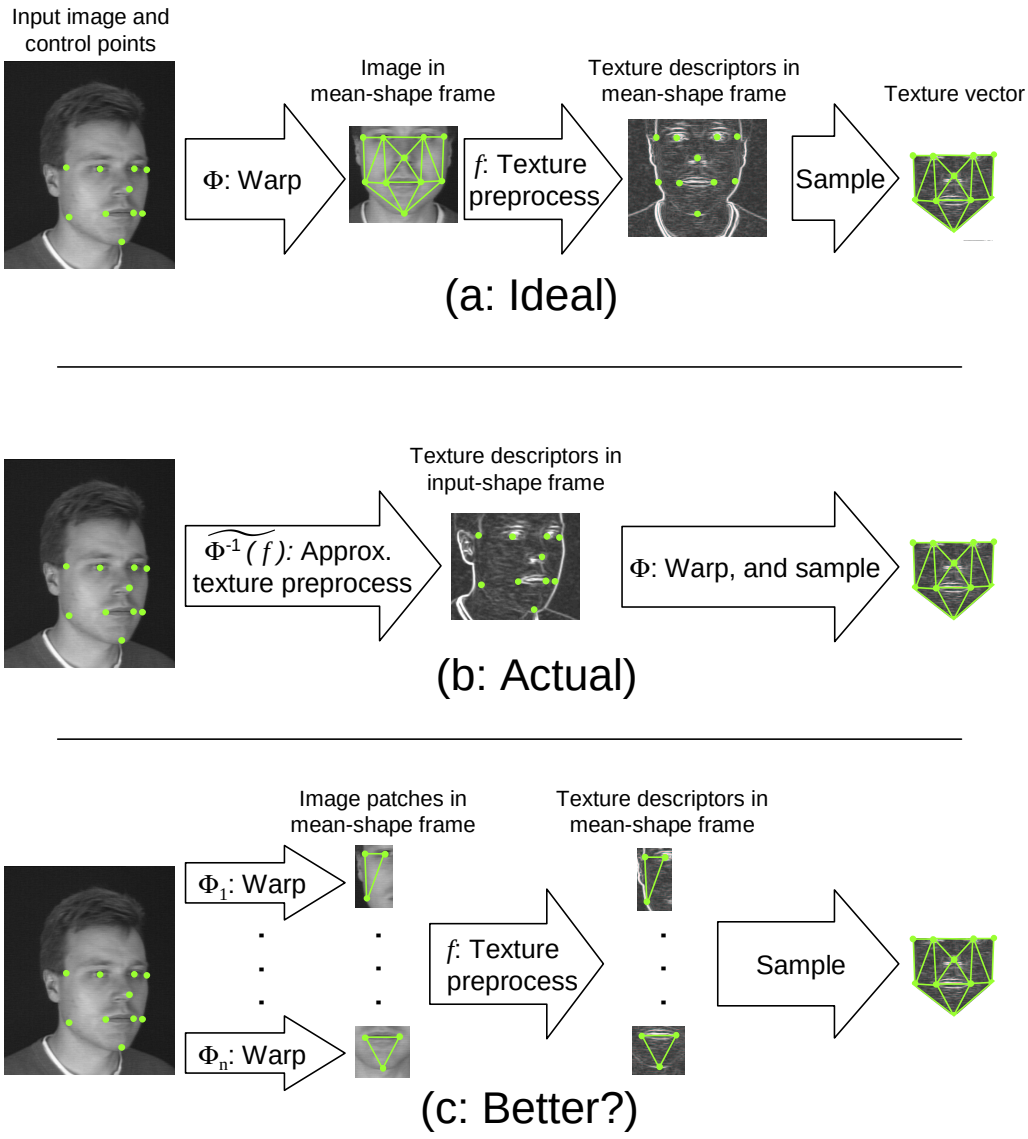



**Figure 6.6:** How the texture preprocessing and warping interact in the (a) ideal case, (b) actual experiments, and (c) a possibly better design.

difference estimate between two points. These two points were warped into the original image where sampling occurs, accordingly. The warping of the gradient sampler points makes this scheme very close to the ideal category of figure 6.6a, but for the sampling of the gradient just outside the triangulation. This approach is appropriate for raw gradient samples, but does not scale when using any more advanced image processing. When a separable convolution operation (e.g. Gaussian smoothing) is involved, warping the sample point for each pixel in a sample's support region is very wasteful, since the temporary values found after applying the first pass of the convolution cannot be reused in the normal way for other samples.

The solution used here is to do all the image processing in the original image space, making use of the efficiencies provided by regular spacing, separability, etc. of a rectangular image. To do this correctly, the parameters of the image processing functions must be adjusted to compensate for difference between the shape-free frame and original-image frame (figure 6.6.b)

The corner and edge estimator starts with a derivative calculation, which is performed using central differences. Standard central differences performed on the input image will use the existing pixel widths. Since one needs to emulate the derivatives being calculated in the shape-free image space, the central differences are approximated in this space. This can be done by multiplying the central differences result by the scale factor between the input image and the shape-free image. The loss of higher order effects are not important because the warping uses bi-linear interpolation anyway.

Another problem is also due to this scale factor. The Gaussian smoothing should take place in the shape-free image. So the width of the Gaussian window has to be multiplied by the scale factor. Both of these simplifications assume that the scale factor is isotropic and constant over the image, which may be unrealistic in some places. So the warp-compensated image processing, is really only an approximation of the ideal case.

Rotation of model patches can be dealt with by simply rotating the feature detec-

tor. This is computationally cheap in the case of gradient measurements, where the two sample points of the central difference are rotated appropriately. In the case of more complicated feature detectors, this rotation would be computationally expensive. Instead, only complex feature detectors that are rotationally invariant have been used.

### Other Possible Approaches

One approach is to use some warping scheme other than triangulation, one that is defined outside the control points. Unfortunately a better warping scheme is the subject of as yet unresolved research.

Sticking with triangulation-based warps, there is an alternative (see figure 6.6.c) to the above approach. Each individual facet in the triangulation defines an affine warp from original image into the shape-free space. Each affine warp can be extended outside its triangular facet, to include a margin for the smoothing and gradient operations. This alternative approach would involve affine warping the unprocessed raw image into a box in the shape-free space. The sampling would be based on the affine field defined for the single triangle contained in the box. Image processing operations could then work efficiently on the box, followed by simple pixel-sampling of the results inside the triangle. This would avoid the need for any fudging of the image processing parameters, and would remove most restrictions on the type of image processing (only finite support restrictions would remain.) There would be a cost overhead, over the existing scheme, to account for the overlapping bounding boxes. But this would be limited to a factor two or three, for most normal triangulations. There is also a theoretical problem that the sample pixel in the original image would contribute inconsistently to samples from neighbouring facets, so reducing correlations that could be picked up by the AAM. This is no worse (and could well be much better) than the current implementation's complete inability to deal with shear in the warp field.

## 6.4 Experiment with Spinal X-Rays

For the spinal X-ray experiments, the `DXAspineA` database (described by Smyth *et al.*[141]) of low-dose Dual X-ray Absorptiometry (DXA) lateral scans of the spines of 47 normal women was used. The vertebrae from Thoracic 7 (T7) to Lumbar 4 (L4) were marked up under the supervision of an experienced radiologist. Figure 6.7 shows an example of an original image (a), and with markup (b). The images are 8-bit greyscale and roughly 140×400 pixels in size. Each vertebra is about 20-25 pixels tall.

Since the database was not large, leave-one-out experiments were performed, by repeatedly training an AAM on 46 of the images and testing it on the remaining image. For each test image 9 AAM searches were performed, starting with the mean shape learnt during training, displaced by all combinations of $[-10, 0, +10]$ pixels in x and y. After the AAM search had converged, the error was measured from each control point on the AAM to the nearest point on the curve through the equivalent marked-up points, called the *point-to-curve error* (see figure 6.13.) Because of the even spacing of control points around each vertebra, this mean error per search will be approximately proportional to the total pixel overlap error, commonly used in segmentation performance experiments.

This whole experiment was run for each of the following texture preprocessors:

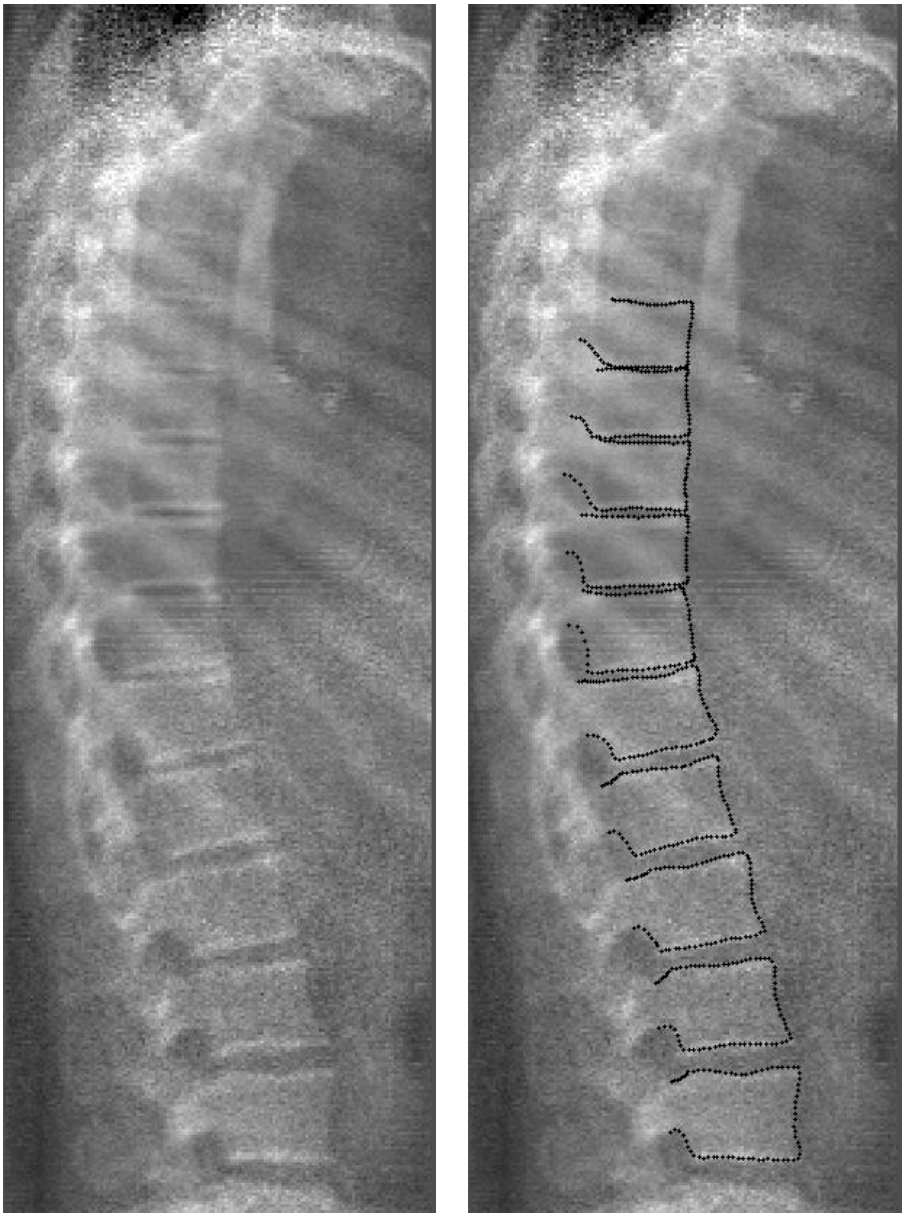
**Intensity:** Original AAM.

**Sigmoidal gradient:** 2-tuple output $\boldsymbol{g}_n$ of sigmoidally normalised directed gradient (equation 6.2.)

**Sigmoidal edge:** Sigmoidally normalised undirected edgeness $e$ (equation 6.5.)

**Sigmoidal corner:** Sigmoidally normalised cornerness $r$ (equation 6.4.)

**Sigmoidal corner and edge:** 2-tuple of the sigmoidally normalised cornerness and edgeness $(r, e)$. (equations 6.4 and 6.5.)



a b

**Figure 6.7:** (a) A spinal DXA image. (b) The same image with markup.

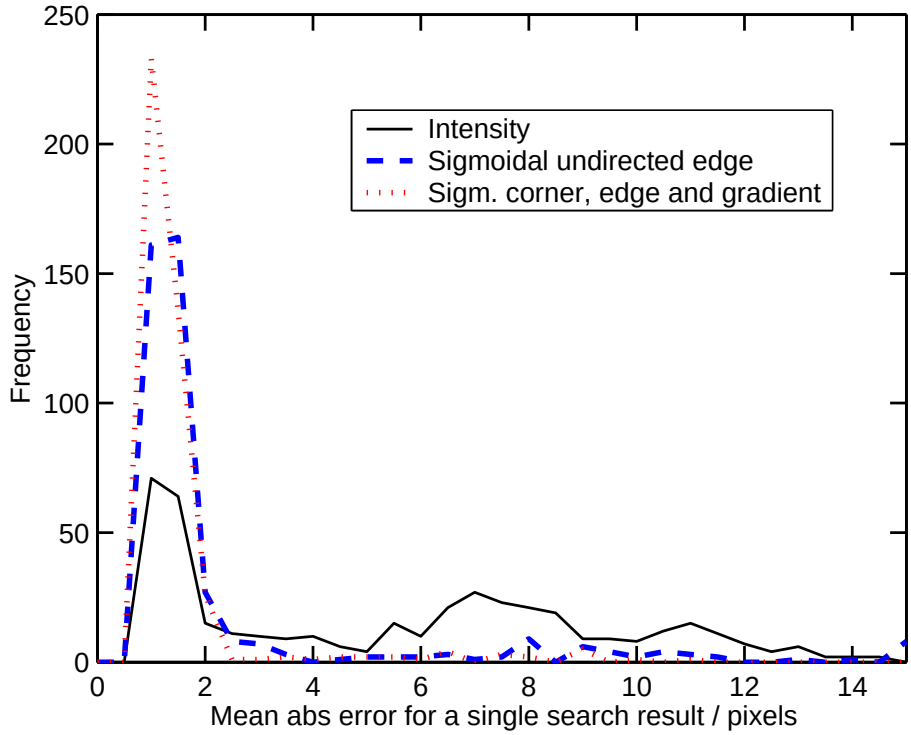



**Figure 6.8:** Error spread between spinal AAM control points and the marked-up curves.

**Sigmoidal corner and gradient:** 3-tuple of the sigmoidally normalised cornerness and directed gradient $(r, g_{nx}, g_{ny})$.

**Sigmoidal edge and gradient:** 3-tuple of the sigmoidally normalised edgeness and directed gradient $(e, g_{nx}, g_{ny})$.

**Sigmoidal corner, edge, and gradient:** 4-tuple of the sigmoidally normalised versions of $(r, e, g_x, g_y)$.

### 6.4.1 Results with Spinal X-Rays

Each search was considered a success if the mean point-to-curve error for that search was less than 2 pixels. (The estimated repeatability of expert annotation is 1 to 1.5 pixels on this data.) Figure 6.9 summarises the results for all of the preprocessors. Texture AAMs with more descriptors show a clear trend towards better performance.

The overall mean error is a mixture of the accuracy of the good matches, and the reliability of the search. The distribution of mean errors for the $47 \times 9 = 423$ searches of the normal database for three of the preprocessors is shown in figure 6.8. The error distributions of the two Texture AAMs show a large shift of results to the left, compared to the intensity AAM. The peak on the left could be thought to represent "successful" fits, and so the Texture AAMs have a large reduction in "failed" fits. This shift is responsible for the majority of the improvement in the overall mean error. If we consider the mean errors for the 423 searches, the $90^{\text{th}}$ percentile is useful to indicate how much of the overall mean error is due to accuracy and how much due to reliability. However, it is valuable to improve both the accuracy and to reduce the number of failed searches, so the overall mean error will remain the primary measure of performance.

**Figure 6.9:** Comparing point-to-curve errors (in pixels) for different spinal AAM texture preprocessors.

| Texture preprocessor | Searches | Point to curve error | | |
|---|---|---|---|---|
| | <2 pixels | mean | std | 90%-ile |
| Intensity | 35% | 5.4 | 3.8 | 11.0 |
| Sigmoidal gradient | 40% | 5.1 | 4.0 | 10.8 |
| Sigmoidal edge | 82% | 2.4 | 3.1 | 6.5 |
| Sigmoidal corner | 75% | 2.6 | 2.7 | 7.5 |
| Sigmoidal corner and edge | 81% | 2.2 | 2.6 | 4.8 |
| Sigmoidal corner and gradient | 80% | 2.1 | 2.2 | 1.9 |
| Sigmoidal edge and gradient | 85% | 1.9 | 2.1 | 4.6 |
| Sigmoidal corner, edge, and gradient | 92% | 1.5 | 1.4 | 1.8 |

### 6.4.2 Simulated Multi-modal Experiments

The medical image community often has to deal with images, taken under different imaging conditions. Sometimes the images come from completely different modalities,

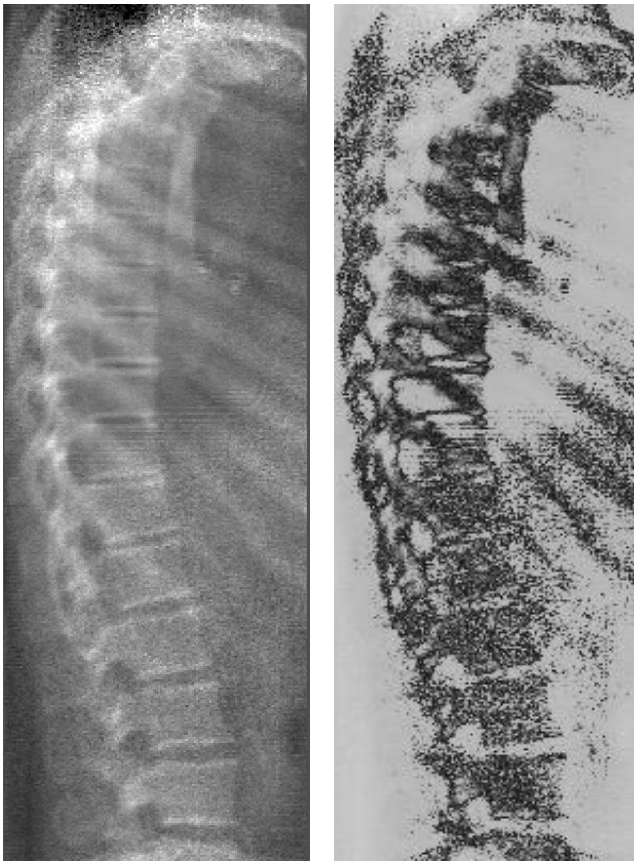


**Figure 6.10:** A spinal DXA image (left,) and after multi-modal simulation (right.)

e.g. X-ray and Magnetic Resonance Imaging (MRI). One might expect texture AAMs to be less sensitive than intensity AAMs to the changes in contrast, etc. that characterise different modalities. So, an experiment was devised to demonstrate this improvement. Roughly half of the set of spinal images were transformed by applying a bitonic pixel-value transfer function (figure 6.11.) This function was chosen by a colleague to produce an image (figure 6.10) that appeared similar to an MRI slice. The two groups were then merged, to give a set of 47 images. A leave-one-out experiment, similar to the previously described one, was then performed on this simulated multi-modal database.

The results for the original "Intensity" and the "Sigmoidal corner, edge and gradient" AAMs are summarised in figure 6.12. The results show that the Texture AAM deals with the *corruption* of multi-modality much better that the intensity AAM.

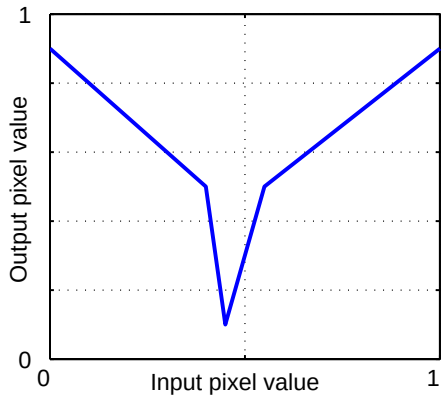



**Figure 6.11:** Pixel value transfer funtion applied to spinal DXA images to produce simulated other modality of multi-modal database.

**Figure 6.12:** Comparing point-to-curve errors (in pixels) for simulated multi-modal spinal images

| Texture preprocessor | Searches | Point-to-curve error | | |
|---|---|---|---|---|
| | <2 pixels | mean | std | 90%-ile |
| Intensity | 7% | 9.5 | 6.1 | 16.0 |
| Sigmoidal corner, edge, and gradient | 60% | 3.4 | 3.8 | 9.3 |

### 6.4.3 Results with Faces

The experiments were repeated with a much larger image database of faces. A face AAM was built using 100 images from ISBE's `expression` database (figure B.4,) and then tested on the independently collected `XM2VTS`[93] database (figure B.1.) This test set consists of 1817 good images of 295 distinct subjects from this database. A few remaining images in the database suffer from extreme motion blur and interlace artifacts, and so cannot be marked up.

The raw "intensity", and "sigmoidal gradient" are repetitions respectively, of the NI and ES cases described by Cootes and Taylor's paper[41]. That paper only used a 100-image subset of `XM2VTS` for testing, and did no statistical analysis of the results.

The results are tabulated in figure 6.14. The $90^{\text{th}}$ percentile of the mean error per search is given for all experiments. A search was considered successful when the

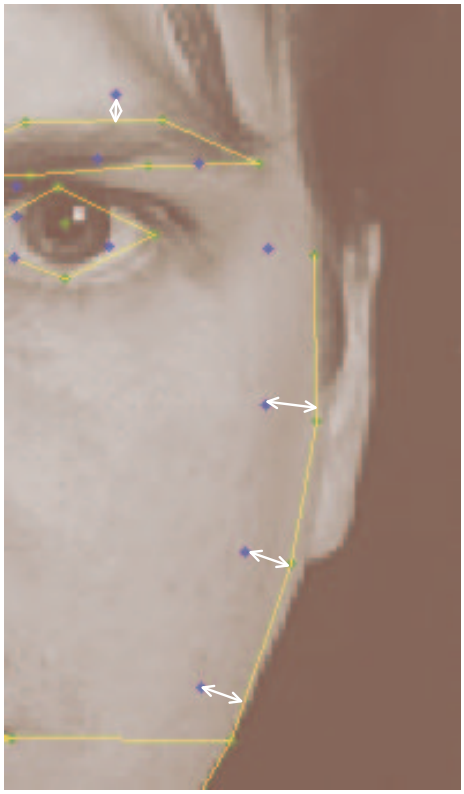


**Figure 6.13:** A marked-up face and model points, showing the point-to-curve error (white arrows) between converged model points (blue) and the marked up curve (yellow.) The error measurement ignores polarity with respect to the curve.

**Figure 6.14:** Comparing point-to-curve errors for different facial AAM texture preprocessors.

| Texture Preprocessor | Searches | Point to curve error | | |
|---|---|---|---|---|
| | $<5$ pixels | mean | std | 90%-ile |
| Intensity | 55.8% | 5.4 | 2.9 | 9.0 |
| Sigmoidal gradient | 72.5% | 4.5 | 2.0 | 7.2 |
| Sigmoidal edge | 68.8% | 4.8 | 2.5 | 8.1 |
| Sigmoidal corner | 68.0% | 4.8 | 2.3 | 7.8 |
| Sigmoidal corner and edge | 73.9% | 4.5 | 2.3 | 7.6 |
| Sigmoidal corner and gradient | 83.6% | 3.9 | 1.4 | 5.7 |
| Sigmoidal edge and gradient | 80.3% | 4.1 | 1.7 | 6.2 |
| Sigmoidal corner, edge, and gradient | 83.7% | 3.9 | 1.7 | 5.8 |

mean absolute point to curve error fell below 5 pixels. This is about 5% of the mean inter-ocular distance in the XM2VTS database. All the Texture AAMs show a strong improvement in mean error and fraction of successful searches. As with the spinal results there is a trend that Texture AAMs with more descriptors give better performance.

## 6.5 Discussion and Conclusions

The local structure descriptors are less dependent than ordinary pixel intensities on the global or sub-global contrast effects caused by differing imaging parameters. The simulated multi-modal spinal image experiment shows that the intensity AAM needs to devote so much variance to its texture model to cope, that it fails to learn any useful information about the images. Comparing the results for the sigmoidal corner, edge and gradient preprocessor in figures 6.9 and 6.12 shows that the severe image corruption has a relatively small effect on a Texture AAM.

This chapter has described the Texture AAM, a novel extension of the intensity-based AAM. The use of descriptions of local structure for the texture model of an AAM significantly improves the performance of the AAM search, and this can be expected to improve object recognition performance.

# Chapter 7

# Further Experiments with Texture AAMs

This chapter further develops and analyses the Texture AAM. First, the statistical significance of the previous chapter's results is examined, leading to the development of a powerful hierarchical bootstrap technique that can cope with the lack of independence and normality in the test data. The role of the sigmoidal normaliser is examined, and compared to the more theoretically-justified histogram normaliser. Experiments with additional texture features show that extra descriptors can, in some cases, further improve performance. Also, analysis of the behaviour of the worst possible descriptors, places a tight bound on any increased error when picking additional descriptors.

The binomial-statistics based analysis of the previous chapter's results in section 7.1.1 7.1.1 was published in papers at IPMI2003[135] and MIUA2003[136].

## 7.1 Statistics

It has been a common failing of many papers comparing various AAM methods (and often in the wider computer vision literature) of not verifying the statistical significance of an improvement. Often this was due to a perceived lack of need, but also due to a lack of tools capable of dealing with the data. This section details several statistical methods which were applied to the data, with the joint aim of proving the value of the texture preprocessors, and providing a statistical method for future work on AAMs.

### 7.1.1 Binomial Statistics

It is not possible to show that the improvement is significant by simply comparing the means and standard deviations in figure 6.9, because the data is not normally distributed. Instead, the percentage of successful results was used. If the results are classified as successes or failures according to the above test (section 6.4,) and the number of successes counted, one should expect the result to be a binomially distributed random variable. When comparing two experiments, one needs to show that any improvement in the percentage of successful results is statistically significant. To do this one must assume that there is an underlying distribution based on a probability of a single success of $\theta$. After performing an experiment with $n$ trials there are $ny$ successes, and so we estimate a probability of success $y$. Performing another experiment of $m$ trials and gives a probability of success $x$. We are interested in the probability of $x$ being from the same distribution as $y$, having already measured $y$.

$$p(x|y) = \frac{p(x \cap y)}{p(y)}$$

Each of these probabilities depends on the parameter of the underlying binomial distribution $p(x|\theta)$, so we must marginalise $\theta$ away.

$$p(x|y) = \frac{\int_0^1 p(x \cap y|\theta)d\theta}{\int_0^1 p(y|\theta)d\theta} = \frac{\int_0^1 p(x|\theta)p(y|\theta)d\theta}{\int_0^1 p(y|\theta)d\theta}$$

**Figure 7.1:** Binomial-statistics based log probabilities ($p$-values) that an experiment could be a random result of a worse performing spinal experiment. Any individual result with a confidence lower than 98% ($\log_{10} p < -1.7$) is marked fail.

| Texture Preprocessor | Result | $-\log_{10} p$-value given base result | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 35% | 40% | 75% | 80% | 81% | 82% | 85% |
| Intensity | 35% | | | | | | | |
| Sigmoidal gradient | 40% | 0.5 | | | | | | |
| Sigmoidal corner | 75% | 4.7 | 3.9 | | | | | |
| Sigmoidal corner and gradient | 80% | 5.6 | 4.8 | 0.6 | | | | |
| Sigmoidal corner and edge † | 81% | 6.1 | 5.3 | 0.8 | 0.5 | | | |
| Sigmoidal edge † | 82% | 6.1 | 5.3 | 0.8 | 0.5 | 0.4 | | |
| Sigmoidal edge and gradient | 85% | 6.7 | 5.8 | 0.9 | 0.6 | 0.5 | 0.5 | |
| Sigmoidal corner, edge, and gradient | 92% | 9.5 | 8.5 | 2.2 | 1.7 | 1.4 | 1.4 | 1.2 |

† Note that the fraction of successful results is rounded down to the next lowest multiple of $1/n$ for $p$-value calculation, causing two rows with slightly dissimilar success rates to have identical $p$-values.

where the binomial distribution is

$$p(x|\theta) = \begin{pmatrix} n \\ x \end{pmatrix} \theta^x \left(1-\theta\right)^{n-x}$$

It does not appear to be possible to find an analytic solution to these integrals. However, one can use numeric integration. Figure 7.1 gives the $p$-values for each spinal AAM result, given a null hypothesis that a poorer performing experiment could have produced that result. It should be noted that because the 9 search tests per image can not be considered independent of each other, the significance calculation was based on a conservative value of $n = 47$.

We can see that the large improvements between the intensity AAM and the various texture preprocessor AAMs are certainly significant. With the exception of the

sigmoidal gradient preprocessor, the differences between the various texture preprocessors are mostly not significant at the $\alpha = 0.02$ level. In the simulated multi-modal experiment, the improvement of the sigmoidal corner, edge, and gradient preprocessor over the intensity AAM, is significant with $p = 5 \times 10^{-7}$.

The same analysis can be performed on the facial data. As described before, the measurements cannot all be used because they are not independent. This is especially unfortunate with the faces experiments because there is a lot of data for which the tests should be largely if not completely independent. In particular, AAM searches of different images of the same person should be mostly independent.

- There are 1062945 individual measurements
- which consist of 16353 search results with 65 measurements each
- which consist of 1817 images each searched 9 times
- which consist of $\sim 950$ sessions mostly containing 2 images
- which consist of 295 people who sat up to 4 sessions each.

However, $n = 295$ was chosen as the number of strictly independent measurements.

### 7.1.2 Linear Modelling of Errors

In order to use the binomial statistical analysis, it was necessary to choose the method to measure success. The thresholds were chosen by asking domain experts, but these values could flatter the results. Avoiding chosen thresholds would be preferable.

The single biggest problem of the binomial approach is that most of the data must be thrown away. This is done for good reasons, to ignore non-normal data distributions, and to deal with the lack of independence. However, by throwing away this data, the experiments are deprived of statistical power to detect improvements that actually

**Figure 7.2:** Binomial-statistics based log probabilities ($p$-values) that an experiment could be a random result of a worse performing facial experiment. Any individual result with a confidence lower than 98% ($\log_{10} p < -1.7$) is marked fail.

| Texture Preprocessor | Result | $-\log_{10} p$-value given base % rate | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 55.8 | 68.0 | 68.8 | 72.5 | 73.9 | 80.3 | 83.6 |
| Intensity | 55.8% | | | | | | | |
| Sigmoidal corner | 68.0% | 3.0 | | | | | | |
| Sigmoidal edge | 68.8% | 3.2 | 0.4 | | | | | |
| Sigmoidal gradient | 72.5% | 5.0 | 1.0 | 0.8 | | | | |
| Sigmoidal corner and edge | 73.9% | 5.7 | 1.2 | 1.1 | 0.5 | | | |
| Sigmoidal edge and gradient | 80.3% | 10.3 | 3.6 | 3.3 | 2.0 | 1.6 | | |
| Sigmoidal corner and gradient† | 83.6% | 13.5 | 5.5 | 5.1 | 3.4 | 2.9 | 0.9 | |
| Sigmoidal corner, edge, and gradient† | 83.7% | 13.5 | 5.5 | 5.1 | 3.4 | 2.9 | 0.9 | 0.3 |

† See note in figure 7.1.

exist. Another, related issue is that the datasets are matched, and not taking account of this further reduces the experiments' potential power.

The non-normality problems can be minimised by transforming the absolute point-to-curve errors $\boldsymbol{x}_{trans} = \log \boldsymbol{y}$. Two preprocessors are compared by subtracting the results in one set of results (e.g. $\boldsymbol{y}_{\text{intensity}}$) from the other, i.e.

$$\boldsymbol{y}_{\text{diff}} = \log \boldsymbol{y}_{\text{intensity}} - \log \boldsymbol{y}_{\text{corner}}$$

When tested on some real error measurements, the results looked approximately normal, but had too high a kurtosis and failed a Jarque-Bera normality test.

In order to try and use more of the data, the lack of independence can be overcome using linear modelling. In the spine data we have 695 measurements made at each search. Each of those measurements is conditionally independent of each other, conditioned on the search result. Each search was performed 9 times from different starting positions. Each search was conditionally independent of the other 8, but

dependent on the patient. Turning this conditionality structure into a linear model, an individual measurement on the $i^{\text{th}}$ patient, $j^{\text{th}}$ search start, and $k^{\text{th}}$ measurement position has contributions from the mean difference $\mu$, any unmodelled error $p(\xi) = N(0, \theta^2)$, and from the 3 controlling factors: the patient $\alpha_i$, the search start $s_j$, and the individual measurement error $m_k$.

$$y = \mu + \alpha_i + s_j + m_k + \xi$$

The equation is rearranged to make calculation easier, by using dummy variables $x$, which switches each individual contribution on or off. All unknown factor contribution variables are placed in a vector $\boldsymbol{\beta}$, with $\beta_0$ the linear intercept, and equivalent to the mean in the previous formulation. ($x_0$ always equals 1.) $\beta_1$–$\beta_{46}$ are the unknown patient factor contributions, $\beta_{46}$–$\beta_{53}$ the search start contributions, etc. The equations for each measurement are then stacked, to give the matrix equation

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} \tag{7.1}$$

The usual practice at this point is to feed the $\boldsymbol{y}$ and $\boldsymbol{X}$ values into a standard statistics package e.g. R[112], which then estimates $\boldsymbol{\beta}$, and various statistics for hypothesis testing. Unfortunately the size of the full matrix $\boldsymbol{X}$ ($\sim$ 210 million elements) was beyond the computer's memory. However, there is no reason to calculate the full matrix, since most of its elements are zero. Hence it was necessary to reimplement the algorithms using Matlab's sparse matrices. The algorithm [56] finds the least squares estimate $\widehat{\boldsymbol{\beta}}$, and calculates several statistics measuring confidence in the estimate.

In order to perform hypothesis testing, it is necessary to create an estimate $\widehat{\boldsymbol{\beta}}_{\text{null}}$ that represents the null hypothesis. This is the least-squares fit to the data, constrained so that the mean is zero. This turns out to be identical to the original estimate, with $\beta_0$ reduced by $\bar{\boldsymbol{y}}$.

The ANOVA method using the $F$-ratio is appropriate to compare the two models $\widehat{\boldsymbol{\beta}}_{\text{null}}$ and $\widehat{\boldsymbol{\beta}}$. This method tests whether two group means are equivalent given knowledge of the inter- and intra-group variances. Full details can be found in Faraway's tutorial[56].

Comparing the intensity AAM and the corner-edge-gradient AAM gives an $F$-ratio of $1.58 \times 10^6$ (for 1 and 293236 d.o.f.) The $p$-value for this is smaller than $10^{-308}$ and underflows on double precision. It appears possible to reject the null hypothesis with confidence. A little further investigation, however, reveals a problem. The $R$-squared statistic for the regression estimate is 0.00217. The linear sum of error factor contributions (eqtn. 7.1) models less than 0.2% of all the variance in the measurement. The ultra-high confidence that the two linear model means are different fails to translate to confidence that the AAM performances are significantly different because the linear models themselves are worthless.

The possible reasons for the poor performance of the linear model are many, but are all likely to involve the assumptions made during the model building.

- The data has already been shown to be non-normal. Perhaps further transformations of the data could deal with this.
- The model assumes that the residuals $\xi$ are independent, identically-normally distributed. It would be possible to deal with this problem using the techniques of generalised linear modelling.
- The most likely source of error is the linear assumption. This implies that, for example, the contribution of first start position is consistent, and is independent of the contribution of the patient. Attacking this problem would require making other less general assumptions about the data.

### 7.1.3 Bootstrap Statistics

The bootstrap is a method for estimating the properties of a statistic directly from the data, while making very few assumptions. A statistic $\theta = s(x)$ is estimated from a sample $X$ of the population. Bootstrap samples $X^B$ are drawn from the sample with replacement, and bootstrap statistic estimate found $\widehat{\theta}^B = s(X^B)$. Repeating the

bootstrap sampling and bootstrap statistic estimates gives an empirical distribution $\{\widehat{\theta}_i^B\},\ i = 1..m$ which was shown to be be an estimate of the statistic's true sampling distribution $F_\theta$ by Efron [54, 53]. The statistic's confidence intervals and standard error can be read directly from the statistic's bootstrap distribution. In cases such as the mean, where we might expect a statistic's bootstrap distribution to be Gaussian, then the $p$-value of the null hypothesis can be also estimated.

The sampling approach described above requires that the original data be independent and identically distributed, which is not true of the AAM comparison data. The conditionally-independent hierarchical structure of the data implies that a hierarchically structured bootstrap is required. So, for example in the face data case, the first factor, the person id, is sampled with replacement 295 times. Then for each sampled person, the sitting id is sampled between 1 and 4 times, depending on for how many sittings that person sat. Then for each sampled sitting, the image id is sampled once or twice, depending on how many images were clean enough to markup. Finally the 65 values per AAM search result, are sampled with replacement 65 times.

The aim of these statistics is to test whether one AAM preprocessor produces better results than another. As with the linear modelling approach above, the data is matched. However, the bootstrap does not require the data to be normally distributed. This allows a simple test; is the mean matched difference significantly different from zero? So the values sampled in the last stage of the hierarchical bootstrap are the difference values, and it is the mean statistic that is applied to each bootstrap population of $295 \times \{1..4\} \times \{1..2\} \times 9 \times 65 \approx 1.38 \times 10^6$ samples.

A two factor version of this hierarchical approach is discussed by Davison and Hinkley [47, p.100], but they admit later that little has been written about the hierarchical bootstrap. Standard bootstrap confidence intervals are known to suffer from a set of problems due to potential bias, and acceleration (which is the change of the standard error of $\widehat{\theta}$ with respect to the true value of $\theta$.) These can be accounted for using methods such as the $\mathrm{BC_a}$, and ABC [53, Ch. 14]. The bias can be trivially estimated

as the difference between the mean of the bootstrap statistics and the mean of the samples $\overline{\widehat{\theta^B}} - \overline{X}$. Calculating the acceleration however is more difficult, and it does not appear to be easy to extend the theoretical underpinnings (or the empirical methods) to a hierarchical bootstrap.

It is useful here to understand the purpose of these bootstrap corrections. ABC and $\mathrm{BC_a}$ have been shown to be second-order accurate—errors in estimating the confidence interval go to zero at a rate $O(1/n)$, where n is the original data sample size. The confidence estimates of the basic bootstrap method are only first-order accurate, with errors going to zero at a rate $O(1/\sqrt{n})$. Since one purpose of the bootstrap here is to use many more data samples that would otherwise be available, one may instead assume that these extra samples compensates for the loss in asymptotic performance.

### Simulation of Hierarchical Bootstrap Method

To supplement the above justification of the hierarchical bootstrap method, it is useful to provide a simulation-style verification. One thousand datasets of $8 \times 8 \times 8 = 512$ items $y_{i,j,k}$ were generated using the following model

$$y_{i,j,k} = 3 + 4a_i + 3b_{i,j} + 2c_{i,j,k}, \quad i,j,k = 1..8$$

where $a$, $b$ and $c$ are independent samples from the normal distribution. 100 bootstrap estimates of each dataset's mean were then calculated. Each set of 100 bootstrap means was examined to give the non-parametric 80% confidence intervals. The distribution of these confidence limits is shown in figure 7.3. In this simulation we should expect the lower confidence limits of about 100 of the 1000 datasets to be on the wrong side of 3.0. The actual number was 77, and for the upper limit, the number was 80, again instead of an expected 100. So the hierarchical bootstrap non-parametric confidence limit method appears to generally correct if slightly conservative.

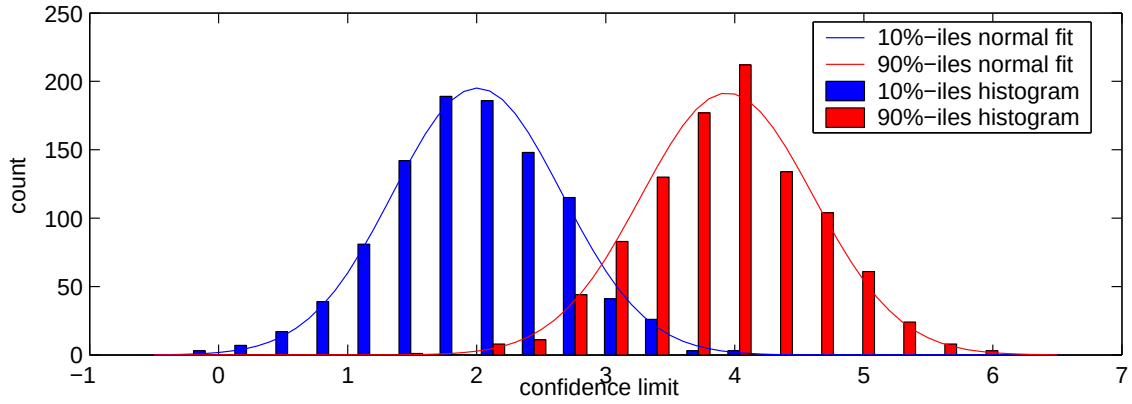



**Figure 7.3:** Simulated error distribution of hierarchical bootstrap confidence limits on the mean. The true mean is 3.0

### Bootstrap Statistics for Facial and Spinal AAM Experiments

Using this bootstrap approach the results from the spinal and facial experiments were tested. The results can be found in figures 7.4 and 7.5. It is clear from the reduction in red ink compared to figures 7.1 and 7.2 that the bootstrap statistics are more statistically powerful. The new statistics confirm the general trend that AAMs with more individual texture descriptors have better performance. The AAM with the most descriptors has either the best performance, or a performance not significantly different from the best.

As well as comparing matched data, the same bootstrap method can be used to put reliable confidence bounds on the mean error for a single database. See the left-most column of the two graphs in figure 7.6 for an example of the mean error with confidence bounds. (The 90%-ile of the mean error per search is also shown.)

## 7.2 A Better Non-linear Normaliser

The non-linear normaliser is a core part of the texture preprocessor. Its original purpose was to break the linear chain in the first linear preprocessors. In this role it can be thought of as similar to the hidden transfer function in an SVM, transforming the

**Figure 7.4:** Bootstrap hypothesis test (confidence limits pass/fail and estimated $p$-values) that an experiment could not be a random result of a worse performing spinal experiment. The experiments have been listed in order of decreasing mean error.

| Texture Preprocessor | Mean Error | $-\log_{10} p$-value, and Pass or Fail | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | I | G | C | E | CE | CG | EG |
| Intensity (I) | 5.4 | | | | | | | |
| Sigmoidal gradient (G) | 5.1 | 1.4 | | | | | | |
| Sigmoidal corner (C) | 2.6 | 24 | 21 | | | | | |
| Sigmoidal edge (E) | 2.4 | 30 | 23 | 0.1 | | | | |
| Sigmoidal corner and edge (CE) | 2.2 | 43 | 30 | 1.3 | 0.6 | | | |
| Sigmoidal corner and gradient (CG) | 2.1 | 48 | 44 | 1.6 | 0.8 | 0.4 | | |
| Sigmoidal edge, and gradient (EG) | 1.9 | 49 | 50 | 2.6 | 1.6 | 1.2 | 0.8 | |
| Sigmoidal corner, edge, and gradient | 1.5 | 62 | 48 | 7.0 | 4.2 | 4.8 | 3.3 | 2.2 |

**Figure 7.5:** Bootstrap hypothesis test (confidence limits pass/fail and estimated $p$-values) that an experiment could not be a random result of a worse performing facial experiment. The experiments have been listed in order of decreasing mean error.

| Texture Preprocessor | Mean Error | $-\log_{10} p$-value, and Pass or Fail | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | I | E | C | CE | G | EG | CEG |
| Intensity (I) | 5.4 | | | | | | | |
| Sigmoidal edge (E) | 4.8 | 8.3 | | | | | | |
| Sigmoidal corner (C) | 4.8 | 8.8 | 0.1 | | | | | |
| Sigmoidal corner and edge (CE) | 4.5 | 17 | 19 | 6.5 | | | | |
| Sigmoidal gradient (G) | 4.5 | 27 | 4 | 3.5 | 0.2 | | | |
| Sigmoidal edge and gradient (EG) | 4.1 | 48 | 30 | 28 | 14 | 20 | | |
| Sigmoidal corner, edge, and gradient (CEG) | 3.9 | 60 | 58 | 51 | 35 | 28 | 19 | |
| Sigmoidal corner and gradient | 3.9 | 60 | 40 | 46 | 23 | 42 | 11 | 0.7 |

inputs for which there is no linear correlation, into a space in which the data fits a good linear model. When merging multiple texture descriptors the non-linear normaliser also serves another purpose, that of making the descriptor values compatible. At the very least, compatibility requires having similar ranges so that larger-valued descriptors do not swamp smaller ones. Taking compatibility to its logical conclusion would suggest that two descriptors should have similar statistical distributions.

The non-linear normaliser used so far has been the sigmoid $n_{\text{sig}}(x) = \frac{x}{x+\overline{x}}$. Whilst this will do a fine job of fixing the range of any input data, it will not force different distributions into the same shape. There are several target distributions one could choose, however if we want a range between zero and one, then the box distribution is an obvious and simple choice. Forcing data to fit the box distribution is better known as histogram equalisation. It does not have a simple transfer function, but can also be thought of as replacing $x$ by its rank.

The corner, edge and gradient AAM is used to compare sigmoid and histogram normalisation[1] with the results shown in the second column of the graphs in figure 7.6. Although the face experiment with a sigmoidally-normalised corner, edge and gradient AAM shows significantly worse performance than the equivalent histogram normalised AAM, later experiments will show both the opposite and no significant difference. We must therefore accept that the optimal choice of normaliser is dependent on the actual data and the details of the AAM. Nevertheless, one might consider the histogram normaliser to be simpler and using Occam's razor, recommend its use when building an AAM on new data.

[1] After the completion of all the experimental and analysis work for this and the previous chapter, a small mistake was found whereby all the non-linear texture normalisation was erroneously followed by a further step of linear texture normalisation. Given that both the non-linear transforms produce a texture vector with a fixed range $[0, 1]$, there should not have been any significant effect. Rerunning all the experiments would have taken a significant amount of setup and computing time. Instead, a representative set of six AAM build and test experiments (out of $\sim 50$,) were rerun with the mistake corrected. The mean difference in point-to-curve error was compared between the original mistaken and correct results. The mean differences in error ranged ranged from 0.001 to 0.05 pixels. These were much smaller that the confidence intervals on those differences. And importantly, they were an order of magnitude smaller than the differences upon which any conclusions relied.

### 7.2.1 Future Work on Normalisers

There are several possible directions to improve the theoretical validity of the probabilistic interpretation of the normaliser, which might lead to improved performance. As discussed previously, the output of the normaliser can be thought of as the probability of there being real structure in the object, rather than noise. Currently the noise distribution is estimated as the sampled values from the image, which is in contradiction to the AAM's core assumption that the sampled pixel values represent some real underlying object appearance. It may be worth attempting to estimate the noise distribution from the whole image, or from just outside the currently sampled region. Alternatively, the noise distribution could be explicitly learnt from the training set.

There is a further problem in attaching probabilistic semantics to output of the normaliser. When the AAM compares the measured descriptor to the values predicted by the model, this comparison takes the form of a difference operator between the measurements and the model. But what is the difference between two probabilities? The update equation for the AAM search predicts a change in the model parameters given the descriptor difference, so maybe it would be preferable to have a probability of difference between the model and the descriptor measurements. There is no obvious relationship between the difference of two probabilities and the probability of a difference. One obvious way forward would be to perform another normalisation on the difference, but then the whole approach starts to look like a series of hacks. It would be beneficial to solve this problem, if only so that an AAM could be treated as a Bayesian graphical model, and benefit from the knowledge in that field.

## 7.3 More Feature Descriptors

One hypothesis raised by the results so far is that it is possible to keep adding ever more descriptors without loss in performance. The mechanism for performance loss

is that additional descriptors will code for information that has already been learnt by the model, but that these additional descriptors might code for that information non-linearly. Therefore the model would be forced to expend more variance coping with similar information that it is unable to model linearly. The mechanism for overcoming this performance loss, is that the learning abilities of the PCA will ignore excess correlations that are of no use, and treat them as noise.

To examine this hypothesis, a large set of additional descriptors is needed. A set of parameterised filters, or filter bank is the most obvious way to find large numbers of descriptors. Potentially interesting families include the Cartesian Differential Invariants (CDI) (which have been shown[163] to have high saliency in shape modelling applications) and complex wavelets[78]. Due to the implementation issues described earlier (section 6.3.5,) it is easier to write a texture preprocessor when the descriptors are rotationally invariant, so the CDI were chosen.

### 7.3.1 Cartesian Differential Invariant Texture AAMs

The CDI[147] are a set of image operators that are simple functions of Gaussian derivative filter ($G_{x^n y^m}$) responses to the image intensities. In two dimensions, there is one first-order invariant, three second-order invariants, and four third-order invariants. As an example, one of the second-order invariants is

$$I_3 = G_{xx}G_yG_y - 2G_{xy}G_xG_y + G_{yy}G_xG_x, \quad \text{where} \quad G_x = \frac{d}{dx}\frac{1}{2\pi\sigma^2}\exp\frac{-I^2}{2\sigma^2}, \quad \text{etc.}$$

Including all the first, second and third order invariants gives eight individual descriptors. The width of the Gaussian was set to one pixel. An AAM was built using these eight descriptors with sigmoidal normalisation, and another built using these CDI descriptors and the corner, edge, and gradient descriptors from before, again with sigmoidal normalisation. These AAMs were tested on the spine and face databases as before. The experiments were then repeated using histogram normalisers.

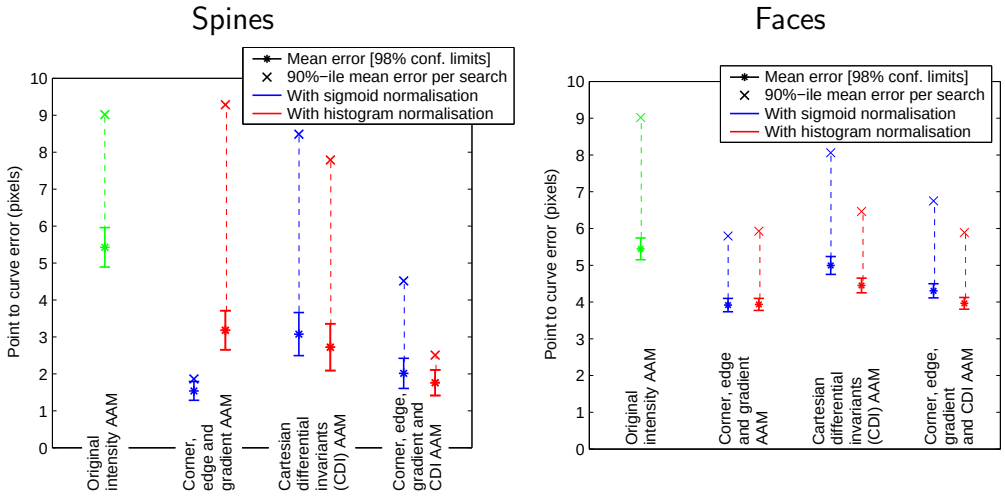



**Figure 7.6:** Comparison of AAM performance with and without Cartesian differential invariants in the descriptor mix. The original intensity AAM and the previously best performing corner, edge and gradient AAM are shown for comparison. The performance of the Texture AAMs with both sigmoidal and histogram normalisation is also compared.

The results (figure 7.6) show that the 8 Cartesian differential invariants perform worse than the corner, edge and gradient descriptors. However, even using matched differences, it was not possible to find a significant difference between the best corner, edge, and gradient AAM (sigmoid normalised) and the best corner,edge, gradient and CDI AAM (histogram normalised.) These results are consistent with the hypothesis than it is always worth adding new texture descriptors to the mix.

Walker *et al.*[163] used CDI at multiple scales to get more information about image texture, and a similar approach could work with AAMs. Starting with the histogram-normalised 8-descriptor CDI AAMs, another 8 CDI descriptors with Gaussian widths of 2 pixels were added to give a 16-descriptor AAM. The face and spine experiments were run as before, and then repeated after adding another 8 descriptors with Gaussian widths of 4 pixels, and then again with 8 Gaussian widths of 8 pixels. The results (figure 7.7) show that the extra information helped in the case of the spines. In particular the number of outliers is heavily reduced. Against the face data, the extra information caused harm, with the point-to-curve fitting error getting slowly, but eventually significantly, worse.

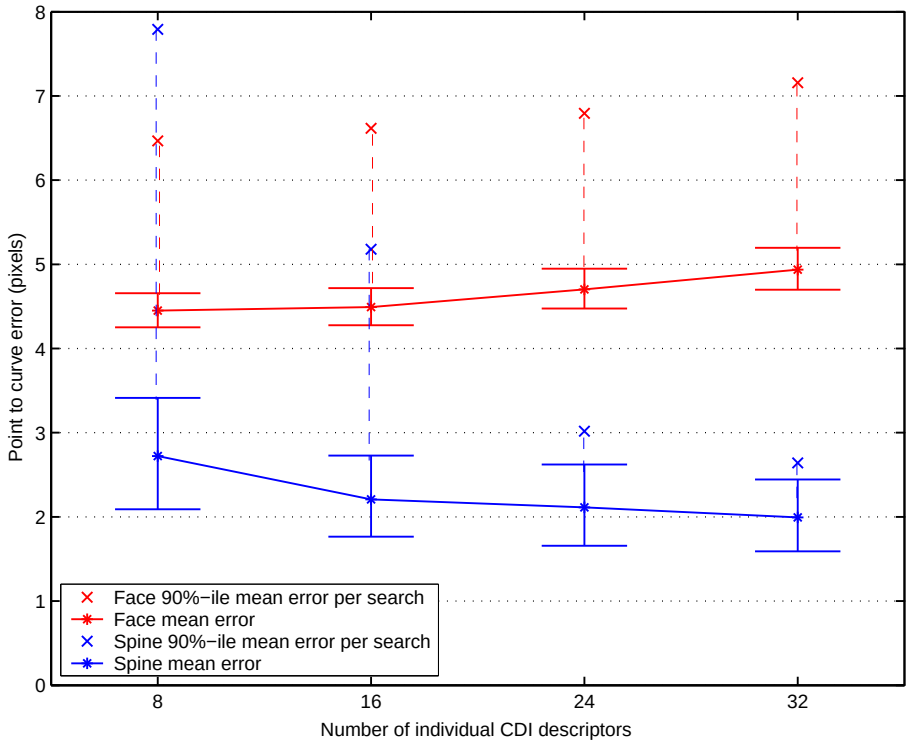



**Figure 7.7:** Performance of CDI AAMs with one, two, three and four different scales.

### 7.3.2 Random Output Texture Descriptors

The results with CDI show that one cannot keep adding more descriptors to Texture AAMs and expect performance to improve. For any situation, there exists a structure description which is optimal in terms of AAM performance. Since this optimum may be difficult to find, it is worth examining the cost of not finding the optimum.

It is possible to put a worst case bound on the ability of truly bad local structure descriptors to damage the performance of a multi-descriptor AAM. The worst possible descriptor is one that produces completely random data. In a simple experiment, the preprocessor of the histogram normalised, corner edge and gradient face AAM was burdened with more and more random samples. The random texture preprocessor consisted of a set of numbers evenly distributed between zero and one, which were randomly shuffled between every sampling of the texture. Figure 7.8 shows that the error rises surprisingly slowly. The performance of the modified AAM only falls to

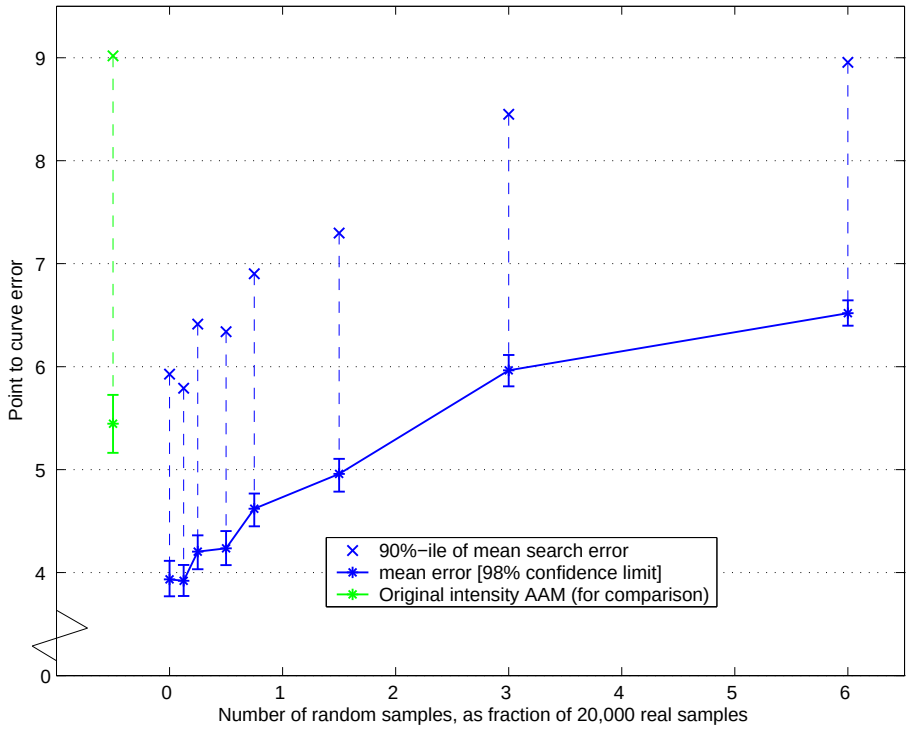



**Figure 7.8:** How adding random numbers as texture descriptors slowly reduces the performance of a facial corner, edge and gradient AAM.

that of the original intensity AAM, when there is more than twice as much random as real data.

## 7.4 Discussion and Conclusions

This chapter has shown that the improvements found in the pervious chapter due to the use of the Texture AAM over the intensity AAM are statistically significant. The hierarchical bootstrap method introduced in this chapter can use the large number of individual error measurements to find statistical significance in small mean differences, despite the measurements lack of normality or independence. These improvements in fitting error can be expected to lead to improved object detection performance.

"Texture AAMs" are now being used by other researchers in ISBE because of their

improved accuracy. Some have also found the hierarchical bootstrap technique to be useful when confirming the significance of small improvements.

The best AAM found for both the spine and face data was the sigmoidally-normalised corner, edge and gradient AAM. CDI were examined as feature descriptors, and found to be less suitable, especially given the extra expense in computation time and space. It is not clear that the sigmoidally-normalised corner, edge and gradient AAM would be best for other situations, and the strictly small loss in performance in any specific circumstance may well be worth the increased generality of having some extra descriptors in the mix. It might be worth examining other descriptors, e.g. the complex wavelets[78] or even other simple corner-, or edge-like descriptors. It is clear from the slow increase in error in figure 7.8 that the AAM's learning algorithm will mostly ignore one or two poor descriptors, in favour of useful ones.

### 7.4.1 Future Work

The work of this and the previous chapter has revealed several promising extensions to Texture AAMs.

Current work in this lab, is looking at separating the peaks in the feature response from the troughs into two image planes, and then smoothing the two planes separately[2]. The aim is to widen the effect of features in the image, so that the AAM can *feel* the feature from further away, and so widen the radius of convergence. Initial results show a small but statistically significant improvement.

One idea from the start of this work, was that it should be possible to look at the influence of each feature detector response for each pixel of the input images. By stripping out all but the most influential, leaving a very sparse AAM, it should be possible to vastly speed up AAM search. Definitely failed searches could be abandoned immediately, with equivocal cases subjected to a less sparse version of the

[2]P. Kittipanya-ngam, private communication

AAM. If some of the feature detectors of the very sparse AAM had locally maximal response in the training images, it may even be possible to directly guess potential locations of AAM fits, in the manner of Weber *et al.*[167]

In theory, this work should extend straightforwardly to 3D images, with the joint calculation of the corner-edge pair in 2D extending to an analogous calculation of a plane-edge-corner triplet in 3D. However, an initial attempt to derive these descriptors found that the some of the tricks used in 2D cannot be applied in 3D. In particular, explicit calculation of the eigenvalues of the matrix of gradient products ($\boldsymbol{M}$—equation 6.3) appears to be unavoidable in 3D. The cost of this calculation, combined with the sheer size of 3D AAMs, has hindered the author's collaborative efforts with Dr. Kevin de Souza to complete a fully-fledged 3D Texture AAM experiment. To reduce computational cost it would be possible to use some cheaper 3D texture descriptors—see Rohr [117] for various options. One very early result on a 27-image hippocampal MRI database[3] shows a reduction in the point-to-surface error of 40% over the standard intensity AAM when using the 3D plane-edge-corner triplet. This is encouraging, but requires further investigation.

[3]The database was kindly provided by G. Gerig *et al.* from the University of North Carolina, Chapel Hill. It was based on a schizophrenia study supported by the Stanley Foundation, and is described further in Styner *et al.*[143]

# Chapter 8

# Improving AAM Performance Using Elliptical Limits

During AAM search, constraints are applied to the model parameters to improve the optimisation's robustness. Although previously assumed to be benign, this chapter shows that choice of the shape and size of these constraints has a significant effect on the AAM fitting error. After examining the previously used box-shaped limits, the more theoretically justifiable ellipsoidally-shaped limits are developed, and the two are compared experimentally, showing that ellipsoidally-shaped limits can produce significantly better fitting error. Finally, this chapter discusses a previously unknown systematic error in the estimation of the PDF of an AAM's parameters.

## 8.1 Model Limits for AAM Search

During the AAM search process, which is in effect an optimisation process, there are several constraints applied to (or freedoms for) the texture, shape, and combined models:

1. Complete freedom to vary in a highly restricted subspace that is equivalent to pose in the shape model, and normalisation in the texture model. This subspace is explicitly designed rather than learnt.

2. Restriction to a learnt principal-component subspace. This is achieved by projecting each model into this principal-component space, setting the values of secondary components to zero.

3. Restriction to regions of the principal-component subspace, that have been learnt to be likely.

It is the latter constraint against implausible regions of the prior-PDF on the models that is being investigated in this chapter. The valid region within the principal-component subspace can be thought of as being defined by a "limiter." During each step of the AAM search, the limiter takes the parameters of the shape model, and if they are outside the valid region, moves them to the nearest point on the edge of the valid region. Another two limiters do the same to the texture and combined models. The purpose of these constraints is to prevent the AAM search process being dominated by a poor current interpretation of the data.

Previous descriptions of the AAM have used a box limiter to enforce these plausibility constraints within the principal-component subspace. This clips each parameter in the model to within some range of the mean. The natural measure of the size of these limits (for implementation purposes anyway) is the number of standard deviations. Previous work from this lab has used $\pm 3$ standard deviations. So, for example, the first principal component of the model can move as far as three standard deviations from the mean (as measured on a learnt prior.) The value of the first component has no effect on the limiting value of the second or any other component. These box limits are cheap and easy to implement, but mean that the model can achieve very low prior probabilities in some directions (on the hyper-diagonals,) but not on others (on the principal axes.) If an AAM-based classifier were to use the parameter values to decide on valid or invalid fits, this arbitrary bias to the diagonals could have a strong

negative effect on the classifier's performance. This work of this chapter attempts to remove that bias, by using ellipsoidal limits as an alternative to the box limits.

AAMs were originally designed using these box limits, and understanding of their performance is based on them. Before switching to another limiter for the reasons above, it is necessary to show that the new limiter does not adversely affect AAM performance. In particular, is there a quantitative change in AAM search performance when changing from a box-shaped to an ellipsoidally-shaped limiter? The effects of different width limiters are also investigated.

## 8.2 Background and Related Work

There is a large literature on AAMs. But (to the author's knowledge) no-one has reported large performance differences due to the shape of the constraint.

Cootes and Taylor[40] have previously compared soft limits, in the form of a Gaussian prior in the AAM's objective function, with hard box limits. They found a general improvement in point-to-curve error performance, but nevertheless remain dubious about the prior overriding the data. In particular, it may reduce the number of completely failed matches at the expense of an increase in the error of successful matches. Ellipsoidal limits have also been used by Cootes *et al.*[34] in ASMs.

Charters[26] compared both ellipsoidal and box limits for a Point Distribution Model (PDM) of chromosome shape in microscopy. He found that box and ellipsoidal limits gave similar performance in terms of rejecting incorrect hypothecated fits of the PDM to the images. The intersection of box and ellipsoidal limits gave better performance.

## 8.3 Ellipsoidal Limits

If the purpose of limits on the model parameters is to prevent the model entering an implausible state, then the conceptually simplest constraint is a constant probability density surface. The usual PDF to be associated with the model parameters is a Gaussian, with separate variances for each of the principal components. The constant density surface for a Gaussian is an ellipsoid. The natural measure of the size of this probability-based constraint is the acceptance probability. This is the integral of the learnt PDF inside the ellipsoidal constraints.

Model positions that exceed the limits, are moved to the *nearest* point on the constraint boundary. Finding the shortest Euclidean distance to an ellipsoid is non-trivial. However, the model space has a Gaussian PDF defined on it, and hence also a Mahalanobis distance. The Mahalanobis *nearest* point on an ellipsoid is simply the intersection of the ellipsoid and a line between the current point and the centre of the ellipsoid. This calculation is only about twice as expensive as applying box limits, which is an insignificant part of the AAM search's computational cost.

## 8.4 Experiments

The performance of AAM search using different sized and shaped limits was compared. The `XM2VTS` database of marked up faces was split in half (with no people shared over the two sets.) An AAM was trained on the first half, and tested on the second. For each image, the model was initialised to the default pose, and placed at each of 9 grid points spaced at 10 pixels in a three by three grid. The grid was centred on the image's labelled points. From each starting position, the model ran to completion.

The performance was measured using the distance from the model control points of the completed search to the nearest point on the labelled curve. The mean error

is reported, with confidence limits on the mean. Confidence limits on each mean were calculated using the bootstrap method (see section 7.1.3) taking account of the structure of the data (repeated subjects, multiple measurements per image, etc.)

Different sized limits were tested with each shape. The results are in figure 8.1. Note that the $x$-axes of the plots for the two different limiters are both measured in standard deviations, but with different offsets. As stated above, the natural control of the size of the ellipsoidal limits, is the included fraction of the distribution. For the face AAM experiments, this value was varied from a fraction as low as 0.2% to fraction as high as 99.999% of the distribution. On the highest resolution level of the AAM, which had a 225 parameter combined model, varying the included fraction caused the ellipsoidal limit radius to vary from 13.1 to 18.2. In order to fit both these results, and the box limiter results on the same graph, the two curves are given different arbitrary $x$-axes. The results show that the AAM with ellipsoidal limits never has a significantly lower mean fitting error than the AAM with the box limits, whatever the size of the valid region.

The purpose of the limiters is to prevent the AAM search diverging due to a poor initial estimate of the fit. In order to judge the effect of the choice of limiter when the start of the AAM search is further from the correct position, the experiment was repeated with the grid spacing raised to 25 pixels (figure 8.2.) Again, the ellipsoidal limiter has a lower error over most choices of valid region size.

For further verification, box and ellipsoidal limiters were also compared using a leave-one-out experiment on the spinal data described previously, with a complex Texture AAM (histogram-normalised with CDI, corner, edge and gradient preprocessors.) Again different size limits were tested, and the AAM matching performance compared—see figure 8.3. With this Texture AAM there is much less difference caused by the choice of limiters. However, the ellipsoidal limiter is still less sensitive to the choice of valid region size.

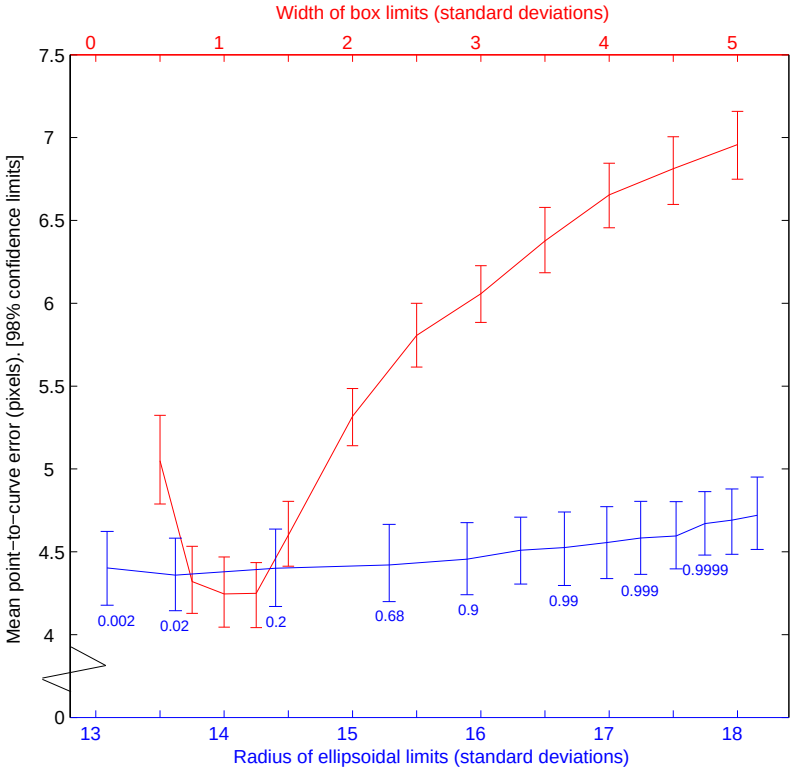



**Figure 8.1:** Comparing AAM search errors between box and ellipsoidal limits, using an intensity AAM tested on half the XM2VTS database. Starting search from 10 pixels displacement. Both curves' $x$-axes are in standard deviations, but on arbitrarily offset positions. The acceptance probability of selected ellipsoidal limits is also given beneath the data points. 98% confidence intervals are shown on the data points.

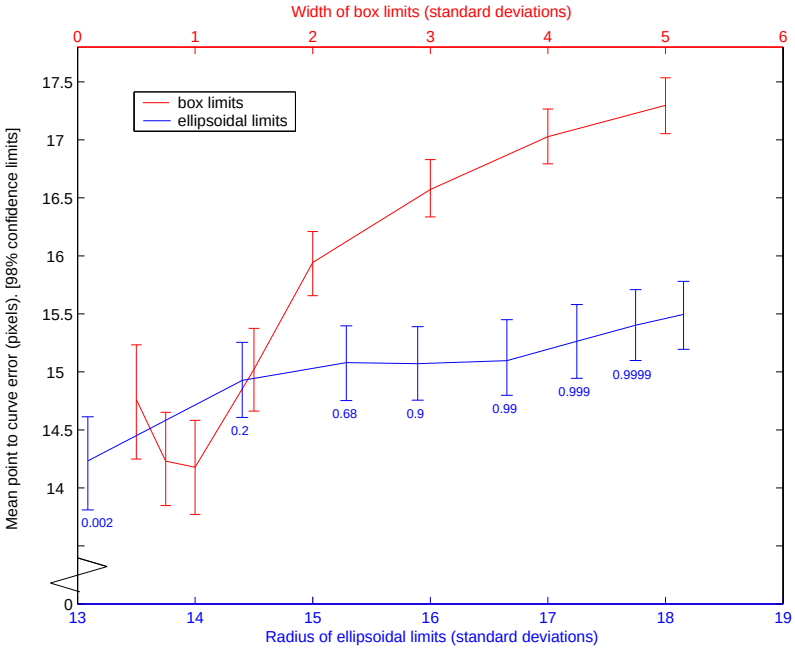



**Figure 8.2:** As figure 8.1, but starting search from 25 pixels displacement.

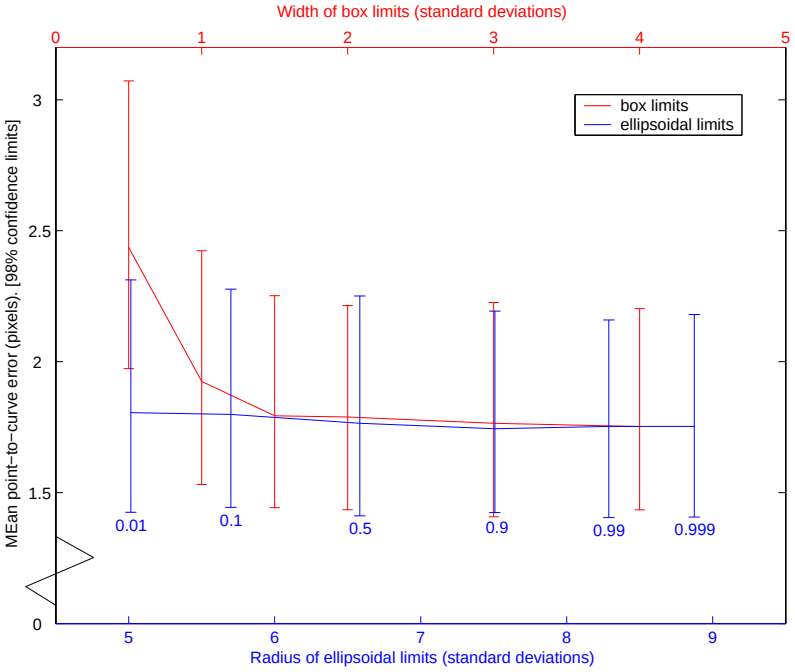



**Figure 8.3:** Comparing search errors between box and ellipsoidal limits, using a corner, edge, gradient and CDI spinal AAM. Starting search from 10 pixels displacement. Both curves' $x$-axes are in standard deviations, but on arbitrarily offset positions. The acceptance probability of the ellipsoidal limits is also given beneath the data points. 98% confidence intervals are shown on the data points.

## 8.5 Discussion and Conclusions

These results suggest that AAMs should use ellipsoidal limits by default. However, they also raise several questions. It has been observed that if a high resolution AAM model is moved from the mean along the axis of the first mode of variation, the model stops being a valid face somewhere after 3 standard deviations. From discussions with Cootes and Taylor, this observation reinforced the 1-D distribution assumption (i.e. 3 s.d. includes about 99% of all normal variation) that appears to have been the origin of the 3 s.d. box limits in the original formulations of AAMs. However, in a 200-D space (which is normal for a 5000-pixel face AAM,) the probability of being less than three standard deviations from the mean is $2.6 \times 10^{-95}$.

Explicitly changing the parameters of a high resolution face AAM (trained on half the `XM2VTS` database,) gives the images shown in figure 8.4. We can see that when varying the first parameter, the model does indeed stop representing a real face at about 3 s.d. However for the higher-numbered modes we can go much further from the mean before the model obviously becomes invalid. In particular we can move to the sort of radius that includes a significant proportion of probability distribution, e.g. 15 s.d. radius includes 89% of a 200-D Gaussian.

It is clear that this aspect of AAM behaviour is not well understood, and deserves further study. One possible explanation is that, for large AAMs, the documented 3 s.d. limit is an artefact of sampling error. Figure 8.5 shows what happens when 1000 samples are taken from a 200-D unit Gaussian, and then used to re-estimate the shape of the distribution. The first principal variances are overestimated, and the last components underestimated. If this is happening with AAMs, then correcting for it might enable the first principal parameter of the model to move to a radius (e.g. 15 s.d.) which includes a more sensible proportion of the probability distribution.

One possible approach to estimating unbiased variances would be to take the initial PCA estimate of the variances, generate many examples with those statistics, and

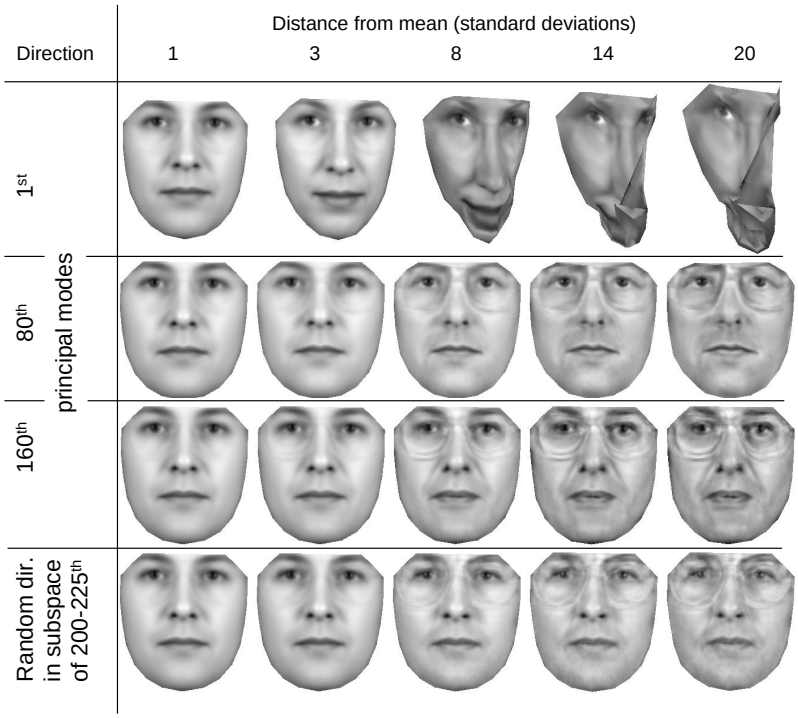



**Figure 8.4:** An AAM model at several distances from the mean, in various directions.

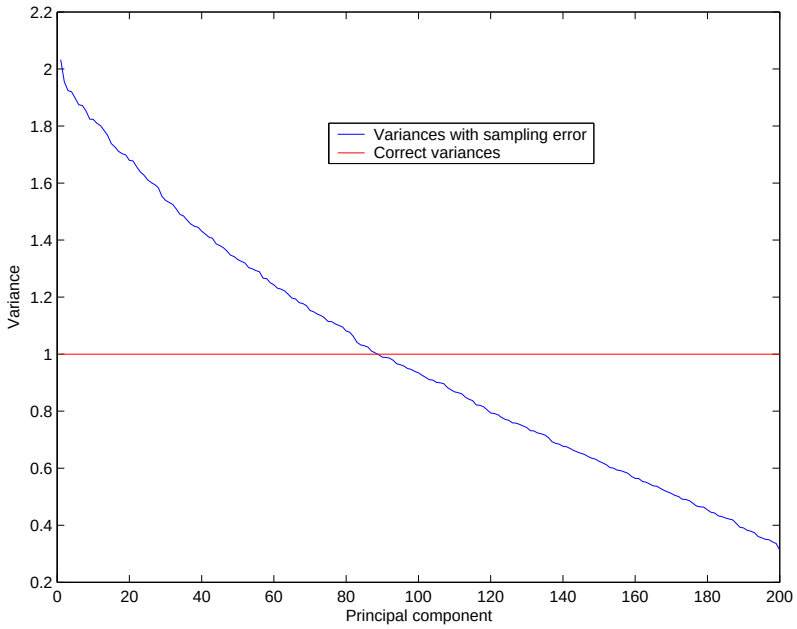



**Figure 8.5:** The effect of sampling error on PCA estimates of variance.

then re-estimate the variances using PCA on the generated examples. Then, adjust the initial estimate of the variances in proportion to the discrepancy between the initial and second estimate, and repeat until the re-estimate of variances gives values close enough to the very original estimate. An alternative approach would be to do PCA analysis on part of the training set to find the principle subspace. Then, use the remainder of the training set to independently estimate the variances within the subspace. To make more use of the training set, this could be repeated with multiple permutations of the training set, and the results combined using eigenspace arithmetic[66].

Whether or not the PCA correctly estimates the principle variances, the Gaussian assumption could be wrong. A multi-variate rectangular PDF with smoothed sides would be a plausible alternative. The iso-probability surface of the this PDF could be similar to the intersection of a hyper-box with a hyper-ellipsoid, which might explain the superior experimental performance of this shape in Charters work[26]. Further work studying alternative PDFs on the AAM parameters may result in improved AAM search performance.

Finally, the experiments described in this chapter confirm the hypothesis that switching from box to ellipsoidal limits does not reduce AAM search performance. The final AAM fitting error is never significantly worse with ellipsoidal limits, than with box-limits. Further, with an ellipsoidal limiter, the choice of limiter size has much less effect on the AAM fitting error. This is very useful, because there are few theoretical guidelines upon which to base a choice of the correct size.

# Chapter 9

# Object Detection Using Combined AAMs and SVMs

This chapter introduces the combined AAM-SVM method, for searching image databases. The discrimination abilities of the SVM are shown to improve the object detection capabilities of the AAM. The converse hypothesis is also confirmed: The AAM improves the object detection capabilities of the SVM. In order to create a combined AAM-SVM, there must be a feature vector extracted from the fitted AAM and passed to the SVM for classification. The obvious feature vector is the texture residual, $\boldsymbol{r}$, of the fitted AAM. The parameters, $\boldsymbol{p}$, of the fitted AAM are also examined as a feature vector. Unsurprisingly the residual gives better performance. Several variations on the residual and parameters are examined, in particular the spatial resolution of the residual is varied.

## 9.1 Background and Relationship to Existing Work

There are three broad mental models for the advances embodied in the combined AAM-SVM:

- Using an AAM as a better feature detector for a modern statistical classification technique
- Using a statistical classifier as a better quality of fit measure for an AAM
- Using both a high-level feature detector and an advanced statistical classifier

This section examines the AAM-SVM's relationship to existing work under these classifications.

### 9.1.1 An AAM as Better Normaliser or Feature Detector for a Statistical Classifier

The AAM can be thought of as just another feature extractor for passing to a statistical classifiers. Whilst a pure-classifier based method can in theory learn every possible variation of an object within an image, it is hard to assemble a representative training set that can cover the entire classification boundary. Instead, all classifier-based object-detection methods view the image through a window, with the classifier being independent of the position of the window. Grey-scale normalisation is also applied during feature extraction. Some object detection methods (e.g. Sung and Poggio's[145] and Osuna *et al.*'s[108]) explicitly remove planar variation in grey level, in order to normalise away strong directional lighting. Without these normalisations, the classifiers would need to learn which variations are due to these invariances, and which are not. In this sense, it is possible to think of the AAM as an extremely good normaliser. The AAM implicitly learns and models changes due to lighting, pose, intra-class variation, object deformation, etc. Thus AAMs should reduce the complexity of the classification boundary to be learnt. A less complex boundary should require a much smaller training set, for a given performance.

To generalise the concept of AAM as normaliser, it is possible to view the work of this chapter through the classical approach to pattern recognition. In this approach,

a vector is created by a feature extractor, and that extractor is then fed to the statistical classifier. Under this view, the AAM is a feature extractor that is superior to a vector of simple intensity-normalised pixels. Unlike the patch, the AAM can (by means of its search algorithm) pass only the locally most likely feature vectors to the classifier, reducing the computational and learning burden on the classifier. The AAM can extract those features with full sub-pixel accuracy, whereas the patch extractor is limited to unit-pixel accuracy. Use of the AAM as a feature extractor will therefore prevent the classifier from having to learn some of the sub-pixel shift invariances that a patch-based feature extractor would need to learn.

### 9.1.2 An SVM as Better Quality of Fit Measure for an AAM

Romdhani *et al.* have used Kernel Principle Component Analysis (KPCA) for the shape and texture models of Active Shape Model (ASM)s[119] and AAMs[120]. This was mostly to deal with very large facial pose changes, without having to resort to a collection of normal AAMs as used by Cootes *et al.*[42] Later, Romdhani *et al.*[118] replaced the KPCA models, with linear PCA models of 3D shape and texture variation, and physics-based models of 3D to 2D projection, lighting, occlusion, etc.

Blanz *et al.*[16] have used an SVM classifier to produce a quality of fit measure. The 3D morphable model above[118] was fitted to a database of faces. The fits were manually categorised into successes and failures, and used to train an SVM classifier. The feature vector consisted of the residual magnitude, and the parameters of the shape and texture parts of the fitted model. They did not test their quality of fit measure, but instead used the classifier score to show that there is a correlation between whether the fit was good (the SVM score) and performance of a face recogniser. The face recogniser itself did not use an SVM, but a modified Mahalanobis distance from the fitted model parameters to the database of known identities.

Edwards, Cootes and Taylor[51] used the AAM to locate a face within a single image. In their approach, a set of AAM hypotheses are started on a grid on the image. All

the AAMs are put through one search iteration. Then, hypotheses with a residual magnitude $E = |\boldsymbol{r}|^2$ of greater than a learnt threshold are discarded. This is repeated, until only hypotheses with $E$ less than another learnt threshold are left, and accepted as being matches. They were able to reject about half of the hypotheses at each iteration. Unfortunately, later unpublished results (by Cootes) strongly suggested that the published performance did not generalise to other test sets, even those known to contain one face with bounded size and position. The residual magnitude $E$ was not a reliable enough indicator of good matches even after convergence, let alone one iteration. Cootes found that using various simple cost functions of the residual and parameter values, rather than $E$, did not make any significant difference. The work in this chapter can be seen as a better estimate of AAM quality of fit, than the residual magnitude $E$.

### 9.1.3 Combining High-level Feature Detectors, and Advanced Statistical Classifiers

The general approach preferred by this author is to use both a high-level feature detector, and a modern statistical classifier, in order to detect objects. A similar approach was adopted by Hamouz *et al.*[67] They used a Gabor-filter map of the image and a GMM to find the most likely candidates for three features from the face. These three features then induced an affine shape model of the pose variation of the face, with implausible poses being discarded. For each hypothesised face, they transformed the original image, using the affine shape model, into a shape free image. A box of pixels around the face was treated as the feature vector, and tested for face/non-face by an SVM. Since they were trying to find single faces in an image for a face verification task, they did not have to make a final decision about each hypothesis, but were able to choose the one with the highest classification score.

## 9.2 AAM-SVM Searching and Training

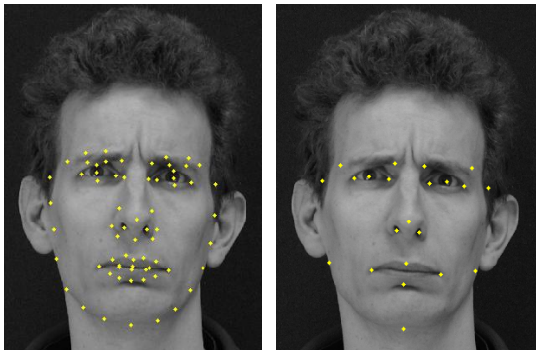
The basic algorithm used in this chapter is a straightforward extension of the patch-based statistical classifier. An AAM is initialised in the top left hand-corner of an image. AAM search is used to find a face in the immediate vicinity. The parameters or residuals of the converged AAM are then passed to an SVM classifier. The classifier decides if the fitted AAM is indeed a real match. The AAM is then moved a few pixels to the right, and the process is repeated.

Training the AAM-SVM is again an extension of the patch-based approach. First an AAM is trained on a set of marked up example images. An AAM is explicitly fitted to each marked object in the positive training set, and the feature extracted. Here, the AAM itself was trained on only half the training set, so as to allow the SVM to learn about the AAM's performance on unseen images.

A set of negative examples is generated by randomly selecting some starting positions for an AAM in a training set known to contain no examples of the object. AAM search is performed from each starting point, and the feature vectors extracted after convergence. These positive and negative examples are used to train an SVM. The AAM-SVM object detection algorithm is then run over the training set to acquire new negative examples, which are added to the training set, and the SVM re-trained. This refinement iterations are repeated several times, to produce a fully trained AAM-SVM.

## 9.3 Building the AAM

This and following sections (9.3–9.5) cover the details of training an AAM specifically for detecting objects. The object to be examined in this chapter is the face. There are more marked up databases for faces that any other object. It is also useful to use the face so that it can be compared to other methods.



a b

**Figure 9.1:** Markup according to (a) the 68-point scheme, and (b) the 22-point scheme.

The standard face AAM markup used in ISBE has 68 points (see figure 9.1a.) This is used to produce reasonably high resolution models for face recognition, reconstruction, etc. For the purposes of face detection, there is little point in having an AAM with a higher resolution that the faces we want to detect. The `CMU` [122, 145] face detection database has faces ranging in interocular distance from 7 to 229 pixels. 95% of the faces are larger than 10 pixels interocular distance. Accurately marking up such small faces, either for testing or training purposes, with the full 68-points would be very difficult. It would also leave only a few pixels per triangle in the shape-model, which would be wasteful in terms of the amount of computational effort needed to warp each pixel. So, a 22-point markup scheme was chosen instead—see figure 9.1b. These points were chosen because either they were easy to identify, or they were useful in expanding the convex-hull within which the AAM is defined.

To train the AAM-SVM, several ISBE databases of marked-up faces were used. These included the `expression` database, and others devoted to identity variation and pose variation, and were already marked up to at least the 68 point scheme. The relevant 22 points were extracted to produce the required markup. Unfortunately these were all collected under very controlled circumstances, fixed backgrounds, etc.—see figure B.7 in the appendix. To augment this database, the publicly available `BioID` dataset[73] was also used. The `BioID` consists of a limited range of people taken under an office computer identity-verification scenario. 20-point markup was available for the

BioID database, of which 18 points were shared with the 22-point scheme needed. Using the constrained AAM search method[1] the known 18 points were used to fit an AAM to both the existing markup and the image data. The four new points found by the AAM were added to the existing 18 points, and were then manually checked. Anecdotally, when using the original intensity AAM trained on other databases above, about 70% of the faces needed some adjustments to at least one point. Using the Texture AAM reduced the number of images that needed any adjustments to about 50%, and also reduced the distance that the points needed to be moved. This latter improvement does have quite a noticeable effect on the required manual effort, and the two improvements together lead to more than a halving in required markup time.

Together, these databases produced a training set of 3970 faces, varying in size with interocular distances from 34 to 420 pixels. These were randomly shuffled to produce a database called all22r. The first half (half-all22r) was used to train the AAM, and whole set (all22r) was used to train the SVM.

Regarding test sets, the XM2VTS database is really quite easy, given its flat backgrounds. The CMU database, is very demanding—the variation in faces here is much greater than in the training set. For example, some of the faces are quite dishevelled. For the purposes of detecting good hits of the face detector, 22-point markup for every face is needed. ISBE already has markup for the XM2VTS set. 6-point markup is provided with the CMU database, of which five are common with the 22-point scheme. Using the constrained AAM method above, the other 17 points were found, and manually checked. The original markup and the manually checked additions were not very accurate (e.g. eye centres were often out by up to 20% of the interocular separation.) However, since this markup will only be used to confirm an AAM's hit or miss, the lack of accuracy is not important.

[1]Developed by Cootes and Taylor[40], the constrained AAM search method modifies standard AAM search, constraining some of the control-points to preset values. This is done by expanding the error term to include the difference between the control-points and their desired positions. The various error terms are made commensurate by reformulating the error into a probabilistic value. The variance of the texture residual is learnt during training, and the variances of the known control points are explicitly set to be tight, with the unknown control-point variances set to infinity.

Choosing the size of the AAM is not straightforward. It is believed within ISBE that 100 pixels is the smallest size of an AAM layer required for the update regression to work reliably. A 283-pixel patch introduced by Sung and Poggio has been successfully used by others[108, 121]. This suggests an AAM larger than 100 pixels would be useful. In order to get the optimal speed and efficiency out of the AAM, two-layer multi-resolution AAM search was used here. Standard multi-resolution AAM design has a two-fold increase in linear resolution at each layer, leading to a four-fold increase in the number of pixels in the texture model of each layer. Therefore, the AAM was designed with 400 pixels in the highest resolution layer (called layer 0.) (Due to approximations in the triangulation code, this layer actually had 399 pixels.) AAM search would be started in layer 1, which had 97 pixels. AAM building was extended into layer 2 with 22 pixels, and layer 3 with 6 pixels, but these were not used for search.

During training, the Principal Component Analysis (PCA) was set to chop off the secondary modes after 99% of the variation in the training set could be modelled. This lead in the highest resolution layer, to 18, 203, and 88[†] modes in the shape, texture, and combined appearance models respectively. The results described in chapters 6–8 imply that the AAM should be built with the corner-edge-gradient texture preprocessors, histogram texture normalisation and ellipsoidal valid-region limiters. The valid region was set to 99% of the acceptance probability, despite the results of chapter 8 suggesting that a lower value may improve performance (see figures 8.1 and 8.2.) The lack of a convincing explanation of those results, means that those results should not trump the straightforward goal of being able to match 99% of all positive faces.

[†]It is reasonable for the combined model to have many less modes than either of the shape or texture model. The final PCA for the combined appearance model only keeps 99% of the 99% of the variation that was not thrown away previously. 98% of the original texture variation can be described in 81 modes.

## 9.4 Radius of Convergence of AAMs

Unlike pure feature-classification based object detection systems, the AAM feature detector does not need to be evaluated at every possible position in the image. An AAM can converge onto the correct position from some distance away. One would expect this radius of convergence to be approximately the radius of the smoothing kernel applied during texture preprocessing of the lowest resolution level of the multi-level AAM. However, the intrinsic width of the features in the image and noise levels also affect the radius. Rather than estimate the ideal value, it is better to measure it.

For faces, the previously described `half-all22r` AAM was tested against the other half of the `all22r` database. 100 times for each test image, the mean model was randomly displaced, search performed, and the final Root Mean Square (RMS) point-to-curve error measured. The random displacements ($x$-translation, $y$-translation, and isotropic scale) were chosen so that the overall length of the displacement was approximately evenly distributed. Displacements of the isotropic scale parameter were made commensurate with the pixel measurements of translation, by multiplying the scaling displacement by the RMS radius, $R_{\mathrm{RMS}}$, of the control points. Initial displacements off the edge of the image were excluded. Since the database contained images of greatly varying sizes, (ranging from 3500 to 550,000 pixels) the initial displacement and final error were normalised by $R_{\mathrm{RMS}}$. (For the 22-point face model, $R_{\mathrm{RMS}}$ is 10-20% less than the interocular distance, or about 2.5 times smaller than the face width.) Figure 9.2 shows the results. We can see that error is approximately flat up to relative displacement of 0.2. That is the displacement that will be used later as the radius of convergence for the AAM.

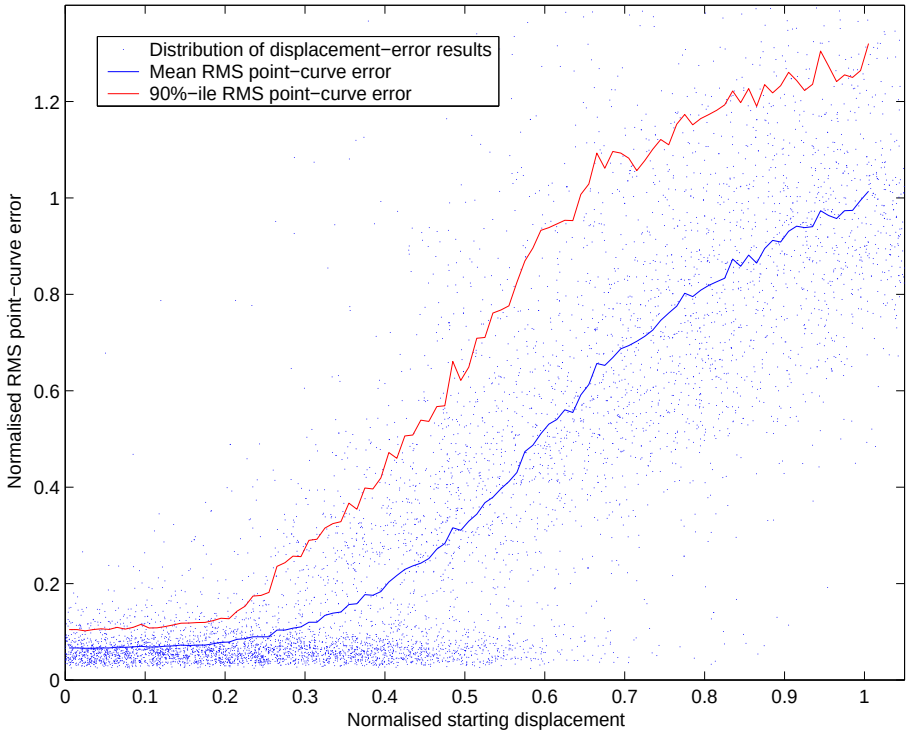



**Figure 9.2:** Search error of the `half-all22r` AAM as a function of initial displacement. Only a random selection of 10,000 result points is shown to avoid saturation. All the results were placed into 100 evenly spaced bins according to initial displacement and the mean and 90%-ile RMS error plotted for each bin.

### 9.4.1 Effect of AAM Improvements on Radius of Convergence

In previous chapters, the radius of convergence was never directly measured, and was not an objective of the improvements made to the AAMs. Whilst it is reasonable to expect the measured reduction in mean error to have included an increased radius of convergence, an experimental validation is preferable. Figure 9.3 shows the same experiment performed on the original intensity AAM with box limits, compared to the fully improved AAM used in this chapter. The AAMs were trained on half of the `XM2VTS` database[93], and tested on the other half, with 50 displacements and searches for each of the 908 test images.

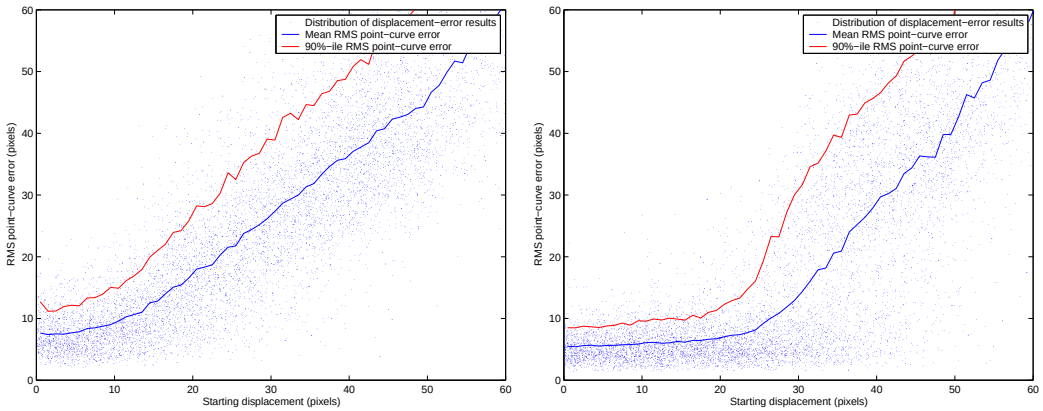



**Figure 9.3:** Search error of the original intensity (left) and ellipsoidally-limited, corner-edge-gradient (right) AAM as a function of initial displacement. Only a random selection of 10,000 result points is shown to avoid saturation. All the results were placed into 100 evenly spaced bins according to initial displacement and the mean and 90%-ile RMS error plotted for each bin.

## 9.5 AAM Search

This section describes how the AAM search process is configured for object detection.

### 9.5.1 Starting Grid

In order to give the AAM at least one chance to find a face within its radius of convergence, AAM search is started from a square grid of initial positions at 2-pixels spacing. The grid extends up to a small margin around the edge. In the scale direction, the grid is spaced at a ratio of 1.2. The pixel separation and margin width are increased with the increase in scale. The initial scale is selected so that the resolution of layer 0 of the AAM matches the resolution of the image. This design gives the AAM the opportunity to match the smallest faces in the `CMU` database.

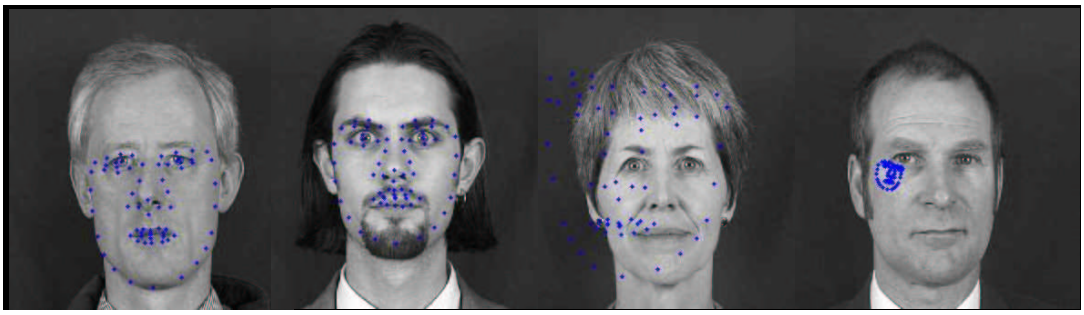
### 9.5.2 Measuring Fitting Error

It will be useful later to have a means of distinguishing a few qualities of fit, namely a definitely failed fit, and a definitely successful fit. The RMS point-to-point error between the AAM control points and test database label points is used as the basic measure of closeness of fit. The point-to-point was chosen over the point-to-curve error used extensively in previous chapters because the point-to-point error is much cheaper to calculate. The RMS error is used over the mean absolute error, because it penalises partial fits where only a few of the control points are far from the labels.

In the radius of convergence experiment (section 9.4 above,) the point-to-point error was normalised by the RMS radius, $R_{\text{RMS}}$, of the control points. For symmetry reasons, it was decided, this time, to normalise the error using the geometric mean of the $R_{\text{RMS}}$ of the control points and the $R_{\text{RMS}}$ of the label points.

In order to decide what might constitute a successful or failed hit, the radius of convergence experiment on the `XM2VTS` database described in section 9.4.1 was repeated on just the first one hundred faces in the database, with just one displacement and search per face. The fits were ranked according to the normalised point-to-point error, and displayed. The maximum error of 0.1 for "definitely successful" fits was manually chosen, so that intuitively good fits were inside it. A "definite fail" minimum error of 0.8 was chosen so that intuitively completely failed fits were outside it. Some images from the experiments along with their normalised point-to-point errors are shown in figure 9.4.

Whilst there was not time to investigate an automatic approach, the choosing of these decision boundaries could be done without manual intervention. By running a radius of convergence experiment, the horizontal section of the results distribution that defines the radius of convergence (e.g. in figure 9.2,) could be identified. The definite fail boundary could be marked so that a suitably small proportion of fits were outside the boundary, and the definite fit boundary so that some proportion of the fits were inside.



Normalised error:

| | | | |
|---|---|---|---|
| 0.083 | 0.119 | 0.662 | 2.09 |

**Figure 9.4:** Several images from the XM2VTS training set, after AAM searches of varying degrees of success. The control points of the fitted AAMs are shown, but the label points for the images are hidden. The normalised RMS point-to-point error is shown below each fit.

### 9.5.3 Dealing with Exceptional Events During AAM Search

Whilst AAMs have been used to search whole images before, the previous implementation used in ISBE did not explicitly deal with certain exceptional events. The AAM can wander completely off the edge of the image, or become too small or large such that the resolution of the AAM and image are too mismatched. Ordinary intensity AAMs can deal with these problems by virtually extending the edge of images while sampling outside, and just ignoring the resolution mismatches. With Texture AAMs these exceptions become difficult to handle. For example, the Gaussian smoothing code cannot handle the empty intersection of the image and current model position. Another example is the inability of the histogram normalisation code to deal with a vector full of zeros, resulting from sampling the gradient off the corner of the image, where all the virtual pixels are identical. Rather than alter substantial volumes of code to deal implicitly with these situations, it was easier to explicitly detect them, and terminate the AAM search.

This requires a decision of what to do about these obviously failed searches (previously they would have been outliers in any error distribution.) While using AAMs for object detection, the ideal solution is to handle search failure explicitly. Unfortunately, the software written for the patch classifier experiments of chapters 3 and 4, and which

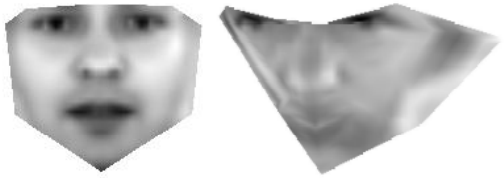


**Figure 9.5:** A normal intensity AAM with model parameters set to the mean value (left,) and to the specific value indicating a failed search (right.)

was being re-used to build the AAM-SVM for this chapter, could not handle the concept of search failure. The ideal solution would have been to modify the framework to handle search failure, but this would have required too much programming effort.

Instead of modifying the entire framework, the following method was implemented. The failed search is intercepted, and a specific "failed feature' vector is emitted in place of the normal AAM feature vector. One immediately appealing choice for a failed feature vector, where all the elements are set to a very large number, is not suitable. Such a vector would grossly affect initial guesses of the SVM's RBF kernel width, massively increasing the number of iterations required to find the optimal result. It would also have an effect on diameter of the data's Minimum Enclosing Hypersphere (MEH) skewing the final optimal value of the RBF kernel width. Instead we want to pick a value that is not too far from all the other data, but is far enough away not to be confused too easily with real faces. Using the known variances of the parameters or residuals, each component of the failed feature is set to one standard deviation. This vector is then scaled, so that it is less probable than 99.99% of the learnt PDF. If this process is applied to an intensity AAM, on the model parameters, the reconstruction (figure 9.5) looks a bit like a face, but certainly not like the optimal fit on any real face. This failed feature is explicitly inserted into the negative training set with the randomly chosen negative examples.

A different strategy of dealing with failed searches is used for local AAM search performance testing. Here, the control points of failed AAMs are set to the search's initial positions. (N.B. This rule was not used in previous chapters because the problem did not occur. The rule was needed for the experiments in section 9.4.)

## 9.6 Building the SVM

Following the work of chapters 3 and 4, the SVM training method will here use an iterative training method. The initial SVM training set consists of all the positive data, and a small negative training set. The positive data is obtained by fitting[3] the trained AAM directly to the known control points of the `all22r` database. No search is performed. The `all22r` database includes both the images seen by the AAM during training and the other, as yet unseen, images.

A negative `UWash` database of natural and city scenes, was originally taken from the publicly available U.Washington image database[137]—see figure B.8 in the appendix. There is no point burdening the SVM learning algorithm with a lot of obviously false examples, that will mostly be superseded by more useful negative examples later in the training process. Therefore 2000 negative examples, (about half the size of the positive training set) are initially selected at random from the `UWash` database. In obtaining negative examples, full AAM search from the starting grid is used.

After initial training of the SVM on this dataset, the iterative refinement process begins. The negative image database is searched exhaustively for faces. Any AAM searches that the SVM deems to be hits are added to the negative training set. The search stops after a fixed number, $n_{\text{select}}$, of new training examples have been found, and the SVM. Stopping after a small number of examples is useful in order to observe the changing behaviour of the classifier during training. However, too few, and the whole process will be dominated by the time to retrain the SVM. $n_{\text{select}} = 200$ was chosen as a useful number of new negative examples to retrain after. In early tests, it was enough to have a significant effect on classification performance.

[3]The fitting process consists of finding a good fit of the AAM's shape model to the label points. Then the texture is sampled, using the fitted control points, and the best fit of texture model found. The best fit of the combined model is then found from the parameters of the shape and texture model, followed by checking of the constraints imposed by the AAM limiter. Strictly speaking, this process should be iterated in an Expectation Maximisation (EM) manner, since the combined model projection will have affected the position of the control points of the shape model. The effect of any further iterations is thought to be insignificant and so they are not performed.

## 9.7 Measuring Performance

Rather than find a single final data-point for the final classification performance, more insight into the experiment could be achieved by following the performance of the AAM-SVM as it was trained. For every AAM-SVM experiment, five performance measurements were plotted for each refinement iteration. See figure 9.6 for an artificial example of the five graphs, each showing a particular performance measure for two different AAM-SVMs as a function of the number of negative training examples scanned. The rest of this section describes the five measures. All of the graphs have the same $x$-axis showing, on a log scale, the number of negative training examples considered thus far during the refinement iterations. The computational time increases approximately proportionally with this value. Each graph has been arranged so that detection performance improves down the $y$-axis, and so that the axes' scales can remain consistent from one figure to the next.

### 9.7.1 False Positive Rate

During each iteration's search for new false positives, the classifier was effectively being tested on unseen data. This measurement may not be perfectly consistent between iterations or across experiments, but it nevertheless provides a cheap unbiased estimate of the current false positive rate. To have consistently measured the false positive rate on a separate test database would have required many times the computational effort. For example, a single experimental run might have taken 4 months, not 3 weeks, on the computers available. The false positive rate is shown in figure 9.6a.

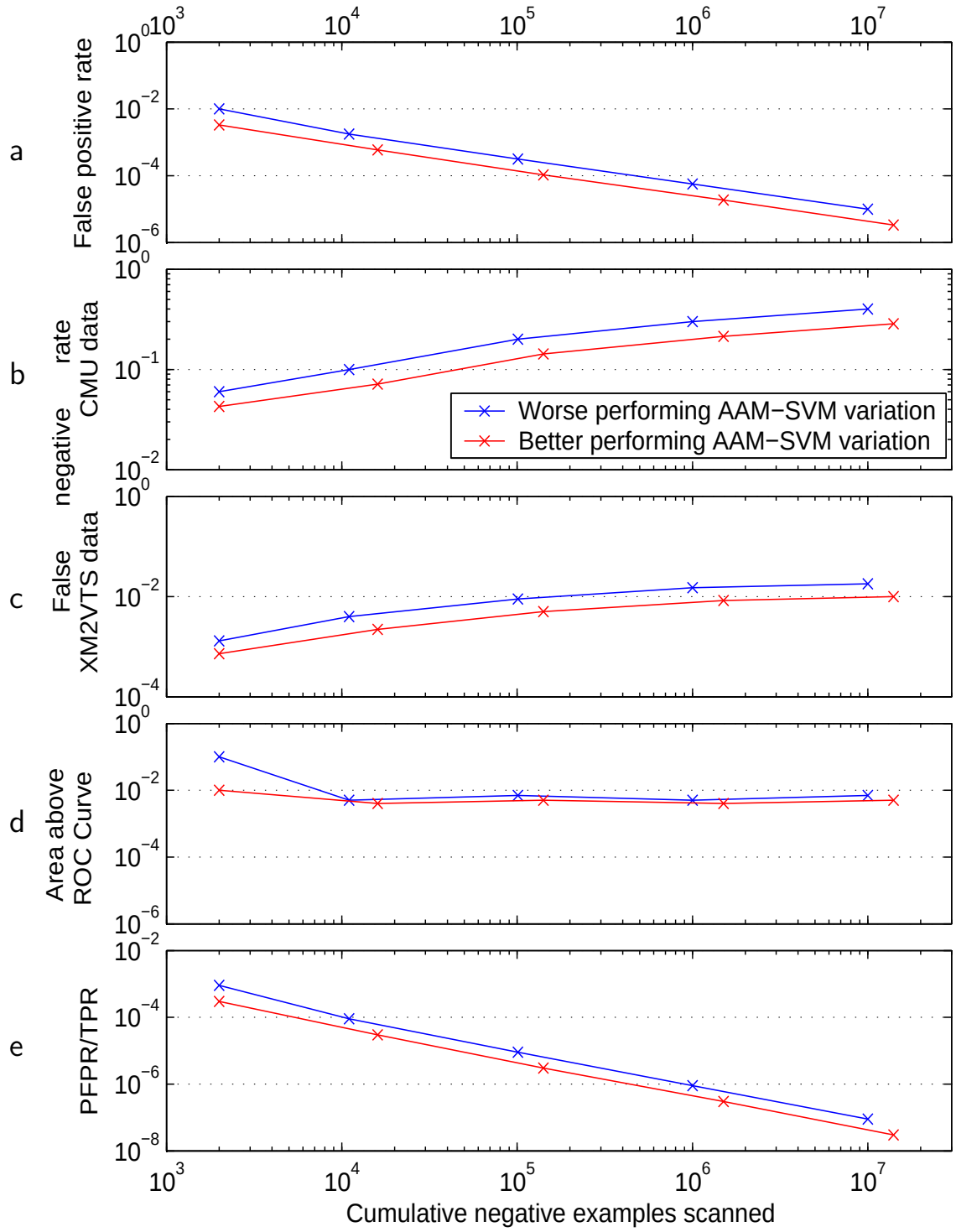



**Figure 9.6:** An example graph comparing the performance of 2 variations on the AAM-SVM method during training. The red AAM-SVM performs better that the blue one in all the graphs. The axes have been arranged so that closer to 0 (down) is better. The top three graphs describe performance with the neutral bias chosen by the SVM training algorithm. The bottom two graphs describe the performance of the classifier, independent of the bias. See section 9.7 for an detailed explanation of each graph.

### 9.7.2 False Negative Rate

After each training iteration, the AAM-SVM is tested on the `CMU` and `XM2VTS` positive test databases to obtain the false negative rate ($1 - \text{sensitivity}$). The two results are shown on figures 9.6b and 9.6c. For speed of testing, the search grid was restricted to starting positions within 2 grid separations of the known face. The starting points remain aligned with the overall image, to avoid bias. For example, if the restricted section of grid's central point was centred on the test data's label points, one might find unrealistically good fitting.

In order to avoid lucky false negatives in this test, the final control points were compared against the label points in the database. If the normalised point-to-point distance was greater than 0.8, the match was automatically failed, ignoring the SVM score.

### 9.7.3 Area Above the ROC Curve (AARC)

The bias chosen by the SVM training may not be the ideal value. However, there are an infinite number of values it could take. Commonly in classification experiments, the area under the ROC curve is used as a single measure of the classification performance, that is independent of the bias. Despite the popularity of area under the ROC curve as a measure of classifier performance, it has a number of problems when describing very good classifiers. It is not possible to use logarithmic scale graphs to help accurately convey the relative performance of different classifiers with areas under the ROC curve of, for example, 0.99, 0.999, and 0.9999. These example values, suggests that Area Above the ROC Curve (AARC) (but within the $[0, 1]^2$ box) is a preferable statistic. The AARC graph is shown in figure 9.6d.

### 9.7.4 PFPR/TPR—an Overall Statistic for Database Retrieval Performance

The biggest problem with the area above (or under) the ROC curve is that for this application area, only a small section of the curve is interesting—the part of the curve near the $y$-axis. The performance away from the perfect-specificity end of the curve, is simply not targeted by the vast quantity of negative training data. So whilst better AARC results are welcome and have been calculated, they should not be used to discriminate between variations in the AAM-SVM method. Another commonly-used scalar measure of classifier performance, is the equal error rate. This is the point on the curve where specificity and sensitivity are equal. Again, this is not particularly illuminating in this application. If the detector misses 10 out of 1000 real faces, but returns $10^7$ false positives out of $10^9$ tested image locations, it will not be of much use.

A more relevant statistic from the ROC curve is the best sensitivity value, for a perfect specificity. This is the highest point where the curve still touches the $y$-axis. Unfortunately, it is very sensitive to the size of the negative test set. The statistic is calculated as the fraction of positive test examples with a higher classification score than the single negative example with the highest (i.e. most wrong) score. Increasing the number of negative examples increases the likelihood of finding a new highest scoring negative example.

A statistic, named "*Perfect* False Positive Rate to True Positive Rate ratio" (PFPR/TPR), was created to account for this effect. The perfect false positive rate is (by definition) zero. However that is strictly a biased estimate of the true, underlying, false positive rate. If there were twice as much negative test data, we might expect to find one worse example. So the '*perfect*' false positive rate is defined to be the false positive rate one would get if there twice as much negative data and one false positive. The PFPR/TPR is thus the ratio of the *perfect* false positive rate

to the true positive rate:

$$\text{PFPR/TPR} \;=\; \frac{1}{\text{TPR}_{\text{at perfect FPR}}} \cdot \frac{1}{2\ \#(\text{negative examples})}$$

The PFPR/TPR measure has a useful interpretation in an image retrieval context. Assuming a very large database, and unlimited computing power, this statistic measures the fraction of all the negative examples in the database that are falsely returned for every real positive hit that is returned. Often used in the general database retrieval field, the *precision* is the number of true positive results returned by a search, as a fraction of the number of all true and false positives returned. The PFPR/TPR is the inverse of the product of the precision and density of positive examples in the test set. Figure 9.6e shows an artificial example of a PFPR/TPR graph.

## 9.8 What to Use for a Feature Vector?

The most interesting option in the design of AAM-SVMs is the choice of feature vector to use for the SVM. There are two obvious sources of information regarding the quality of fit of an AAM—the residual and parameters. The residual (the difference between the best fit of the AAM and the test image) by definition contains a substantial amount of information about the success or otherwise of the fit. However, it is quite large, with four elements for every pixel (the corner, edge and two gradient components.) In the highest resolution layer of the multi-resolution AAM the residual vector $\boldsymbol{r}_0$ has 1596 dimensions. Even with very good statistical classification methods, having unnecessary dimensions that just contain noise on top of no extra discriminatory information, will lead to lower classification performance, as well as lower speed. In order to obtain a smaller vector, a lower resolution layer of the multi-resolution AAM can be fitted directly to the control points of the best-fit solution from the full multi-layer search. Lower resolution residuals can then be obtained from layer 1, $\boldsymbol{r}_1$ with 388 dimensions, or layer 2, $\boldsymbol{r}_2$ with 88 dimensions.

The AAM parameters at the end of AAM search are another candidate for the feature

vector. Although they contain less information, there is no reason to expect the ellipsoidal constraints imposed by the AAM limiter to be a true classification boundary between valid and invalid faces. So the parameter values (of the final, high-resolution layer) should contain some information regarding the validity of the face it has fit to. This should be especially true when the constraints are relatively loose.

### 9.8.1 Results with Residuals as the Feature Vector

As described before, the AAM residuals are the most obvious features. Figure 9.7 shows the performance of the AAM-SVMs for each of the three different resolutions of residual. The experiments were halted manually, invariably during the search for new negative examples. Sometimes they were halted when it became clear that it was going to be many weeks before the end of that iteration, and sometimes after enough data had been obtained to make an inference.

The $\boldsymbol{r}_2$ area above the ROC curve has been marked with 98% confidence limits. The confidence limits were calculated using the bootstrap resampling method, as described in section 7.1.3. As before, no second-order adjustments were used. But this time, no complicated hierarchical sampling was needed, just simple resampling from the positive and negative test results. Calculating these limits was computationally expensive, and so it was only performed once to give an indication of the reliability of that statistic. One would expect the bounds to be tighter on the better performing AAM-SVMs.

The first thing to say about the results is that the AAM-SVM training is working. As more negative training examples are added, the false positive rate (figure 9.7a) falls much faster than the false negative rate (figures 9.7b–c) rises.

Comparing the three AAM-SVMs shows that the lowest resolution residual $\boldsymbol{r}_2$ was worst performing on all measures (`r2` on figure 9.7.) There is simply not enough information in the 22 pixels ($\times 4 = 88$ dimensions) to reliably discriminate faces. The

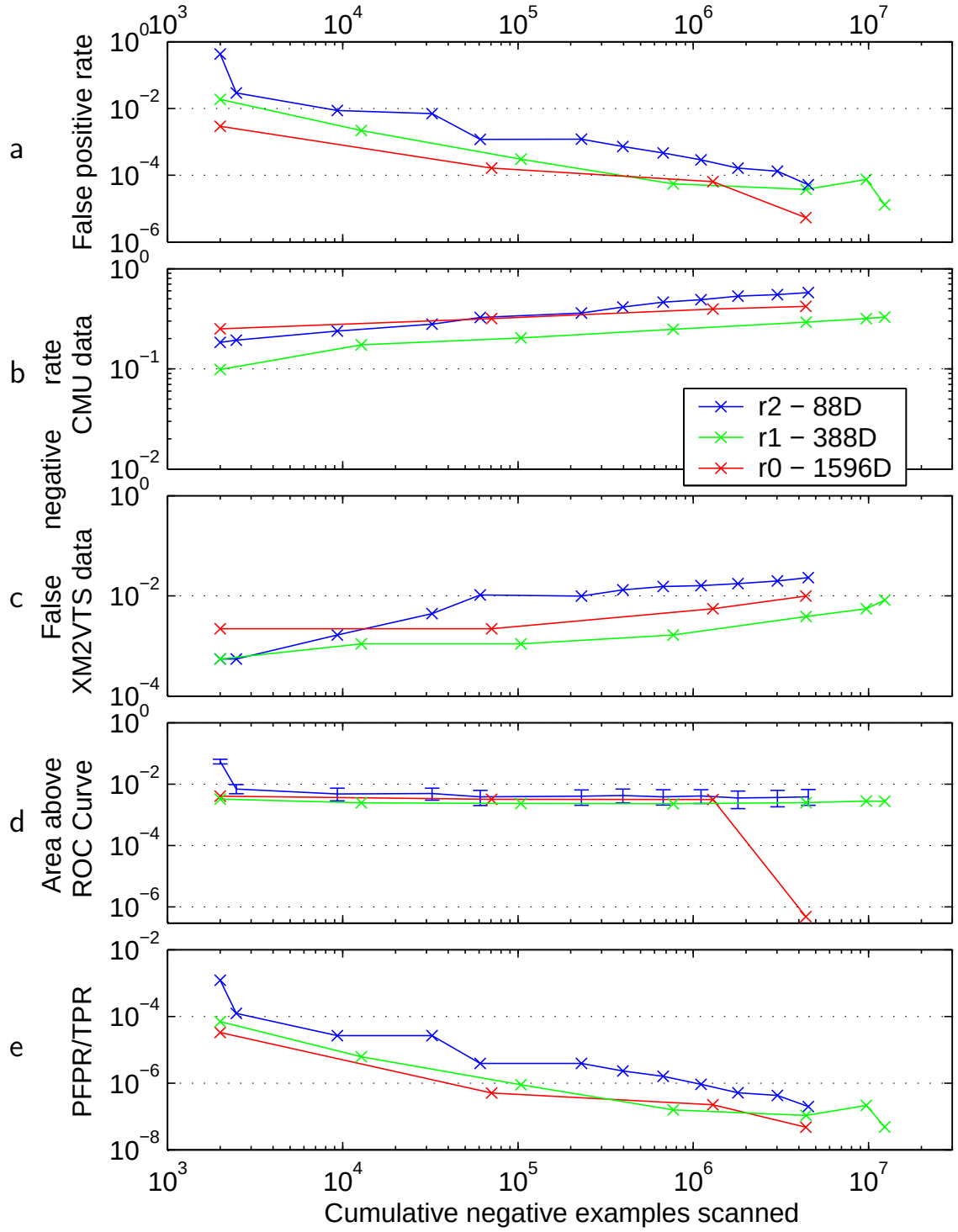



**Figure 9.7:** Comparing the performance of three different residual resolution AAM-SVMs during training. $\boldsymbol{r}_0$ (r0 on the key) is the residual from the highest-resolution layer of the multi-resolution AAM with 1596 dimensions. $\boldsymbol{r}_1$ from the next lower-resolution layer with 388 dimensions, and $\boldsymbol{r}_2$ from the lowest resolution layer with 88 dimensions. The $\boldsymbol{r}_1$ AARC statistic has been marked with 98% confidence limits. (See figure 9.6 and section 9.7 for a description of the layout.)

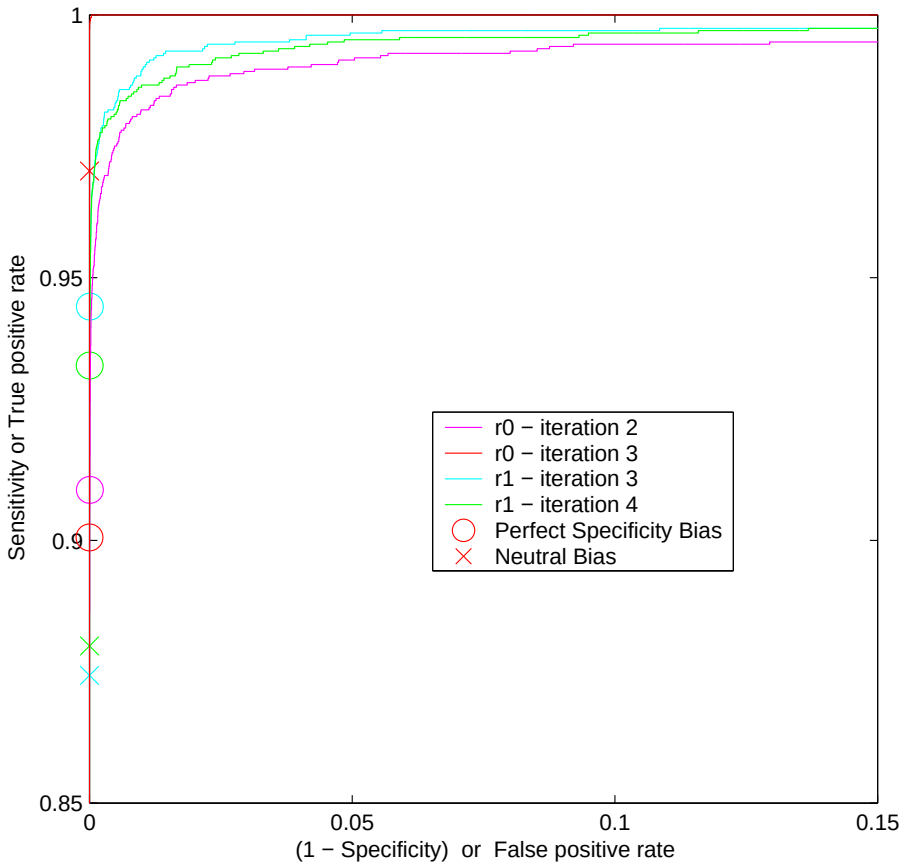



**Figure 9.8:** The upper-left corners of ROC Curves for the final two iterations of the $\boldsymbol{r}_0$ AAM-SVM. ROC curves of iterations 3 and 4 of the $\boldsymbol{r}_1$ AAM-SVM are shown for comparison at a similar stage of training. The points on each curve for the neutral bias, and best sensitivity for perfect specificity are also marked.

performance of the two higher resolution residuals is more interesting. Whilst it is clear that the highest resolution residual, $\boldsymbol{r}_0$ (`r0` on the graphs,) is performing slightly better overall, there appears to be enough information in the 97 pixels $\times$ 4 = 388D of the $\boldsymbol{r}_1$ residual to make almost as good a detector.

It is unclear why the last SVM training iteration for the $\boldsymbol{r}_0$ feature AAM-SVM gives such a small area above the ROC curve (figures 9.7d.) As has been stated before, the training regime is not intended to produce significantly better performance of this statistic. As can been seen from figure 9.8 the false negative rate of that detector at neutral bias is not very high. No obvious explanation can be found for this very high performance except a lucky training set.

Due to the lack of explanation of the performance of the final iteration of the $\boldsymbol{r}_0$ AAM-SVM, and the relatively good performance of the $\boldsymbol{r}_1$ AAM-SVM, the $\boldsymbol{r}_1$ was used instead as baseline for further experiments, and is sometimes labelled `ceg-AAM-SVM-r1`.

### 9.8.2 Results with AAM Parameters as the Feature Vector

As an alternative to the residual, the fitted AAM parameters are obvious candidate features. Whilst the limiter (as discussed in chapter 8) provides a crude boundary on the valid parameter values, there is no reason to believe that an elliptical limit with an arbitrarily chosen radius is a true classification boundary between valid and invalid faces. Providing the parameter values to the SVM will allow this true boundary to be determined. The existing experimental setup was changed to use the 88 parameters of the fitted high-resolution layer of the AAM. This was designated the $\boldsymbol{p}$ AAM-SVM.

Between them, the parameters and residual contain all the information the AAM has about the underlying image patch. However, they do not contain the same information. Therefore, it might be be valuable to have both the residual and parameters values available to the classifier. The experiment was rerun with this $\boldsymbol{pr}_1$ feature vector.

One potential problem with using the parameter values is the interaction with the limiters. After all, the size of the limiters is set (in theory at at least) to produce plausible faces, and the SVM is being used to find invalid faces. The AAM could be run with no limits, so that invalid faces are well modelled by the AAM, but then rejected by the SVM. However, there is a reason for the use of limiters in AAMs—search on real faces usually fails without them. The initial state of the AAM needs to be close to the optimal value not to diverge. This suggests a solution. First, run the AAM with the (now standard) ellipsoidal limiters, to fit the image patch, and then remove the limits, and continue the search to get a better fit to invalid faces.

The vast majority of the work of AAM fitting is done by the lower resolution layer.

The higher resolution layer mostly adjusts the control points within a pixel or two (in the higher resolution.) So, one alternative is to run the multi-resolution search with the lower-resolution layer controlled by the normal limiters, and the higher-resolution layer unconstrained. This approach was easier to fit into existing implementation, and so was chosen instead of the one described in the previous paragraph. One implementation consideration was ceded—because the multi-resolution AAM implementation shares the shape model between layers, it was left constrained as normal. Only the texture and combined models were unconstrained. This detector was named $\boldsymbol{p}_{\text{loose}}$.

The three regimes $\boldsymbol{p}$, $\boldsymbol{pr}_1$, and $\boldsymbol{p}_{\text{loose}}$ are compared against the previous $\boldsymbol{r}_1$ result in figure 9.9. We can see that it is possible to use fitted parameters $\boldsymbol{p}$ to discriminate faces from non-faces, but with worse performance that when using the residual $\boldsymbol{r}_1$. The broadly similar performance of the residual $\boldsymbol{r}_1$ AAM-SVM and the combined residual/parameters, $\boldsymbol{pr}_1$, AAM-SVM suggests that the parameters contain no additional information to the residual, to help distinguish faces from non-faces. The use of an AAM with no limiter in the high-resolution level appears to be an improvement upon the normal AAM when using the parameters as a feature vector, but only at the expense of substantially worse false negative rate. This is almost certainly due to the loose AAM's poorer ability to accurately converge on the real faces.

When the AAM parameters are used for classification, the best sensitivity for perfect specificity is not very good. The best value for the $\boldsymbol{p}$ AAM-SVM was 45%, and this fell to 29% at the end of the refinement iterations. This compares to typical values for the residual-based classifiers of 85%–95%. The value for the $\boldsymbol{p}_{\text{loose}}$ and $\boldsymbol{pr}_1$ variants of the parameter-based AAM-SVM were 68% and 53% respective at the end of the refinement iterations.

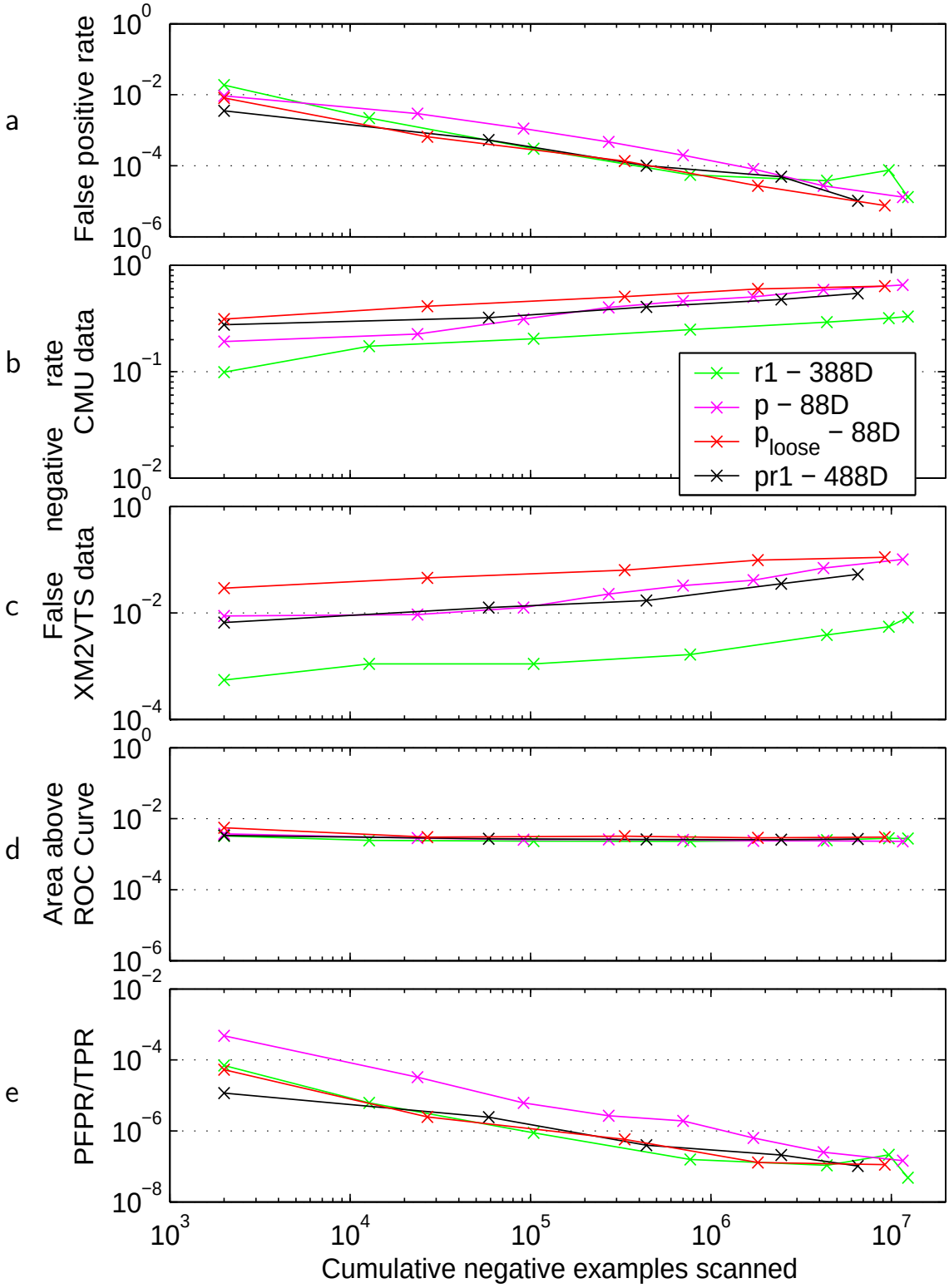



**Figure 9.9:** AAM-SVM performance using various versions of the AAM parameters for the SVM's feature vector; the normal AAM parameters, $\boldsymbol{p}$; the concatenated AAM parameters and medium resolution residual, $\boldsymbol{pr}_1$; and the parameters, $\boldsymbol{p}_{\text{loose}}$, of an AAM with no limiter in the final high resolution level. The $\boldsymbol{r}_1$ results from the previous experiment are included here for comparison. (See figure 9.6 and section 9.7 for a description of the layout.)

## 9.9 Conclusions

This chapter has described the AAM-SVM, a novel method of object detection, which uses an AAM as a high-level feature detector and an SVM to accurately discriminate faces from non-faces.

The AAM residual has been shown to be useful as a feature vector. The residual from the medium-resolution layer of the AAM gives almost as good detection performance as the high-resolution residual. Either way, the residual is more useful, as a feature vector, than the AAM parameters. Nevertheless, there is enough information in the AAM parameters to discriminate faces from non-faces.

A key limitation of the AAM-SVM at this stage is its slow speed, due to its exhaustive search for potential hits.

# Chapter 10

# Further Experiments with AAM-SVMs

Having introduced the AAM-SVM previously, this chapter investigates several variations, and compares its performance to related object detection techniques. It is shown that the AAM-SVM is more accurate than the Osuna *et al.*-style patch-SVM face detector, given the same training set. The improvements to the AAM described in chapters 6–8 are shown to have a large positive effect on the AAM-SVM's detection rate. Further investigation of the $\boldsymbol{r}_1$ residual as a feature vector leads to conclusion that a face can be reliably detected at a much lower resolution than others have reported. The use of lower resolution leads to a large increase in speed. In order to achieve a very large increase in speed, a multistage approach to object detection is developed. This uses a fast patch-based detector to find potential hits, and an AAM to refine those guesses. Finally, there is a statistical and qualitative analysis of the AAM-SVM method.

A paper on the multistage face-detection work described in section 10.4 has been accepted for publication at BMVC 2004[45].

## 10.1 Comparison with Existing Methods

This section shows that the AAM is a superior feature extractor/normaliser for an SVM-based detector, over the patch-based feature extractor used by Osuna *et al.*[108] It also confirms the complementary hypothesis that the SVM is a superior classifier between valid and invalid AAM fits than the natural error of the AAM.

### 10.1.1 Comparison with the Patch SVM Detector

In order to compare the AAM-SVM to alternative approaches, the SVM-based image patch detector described in section 4.6 (similar to other published SVM face detectors[108, 121]) was tested on the same database. The full `all22r` database was used to train the classifier. With the reflections, and small rotations applied to the positive examples, this creates an initial positive training set of 23,820 examples. The negative sample training regime was similar to that used above for the AAM-SVM experiments. 2000 negative samples were selected initially at random from the `UWash` database, and the SVM trained to produce an initial detector. The SVM used the optimal-width RBF kernel (described in section 4.6) rather than the fixed-degree quadratic kernel used by Osuna *et al.*[108] (An RBF kernel was also used by Romdhani *et al.*'s[121] SVM-based face detector but, like Osuna *et al.*, the kernel parameter was fixed.) This detector was then run over the `UWash` database to find false positives for the negative training set. The SVM was retrained after each new set of $n_{\text{select}} = 200$ false positives had been found. The false negative rate, etc., were measured after each iteration as before. The results are shown (labelled `patch-SVM`) in figure 10.1.

Because of the need to test the detector on a 1×1 pixel grid in this experiment, the `patch` SVM detector needs to scan four times as many patches per image as in the AAM-SVM experiments. This is approximately $1.2 \times 10^6$ patches for each of the 756×504-pixel images in the `UWash` database. So, to compare the progress of the

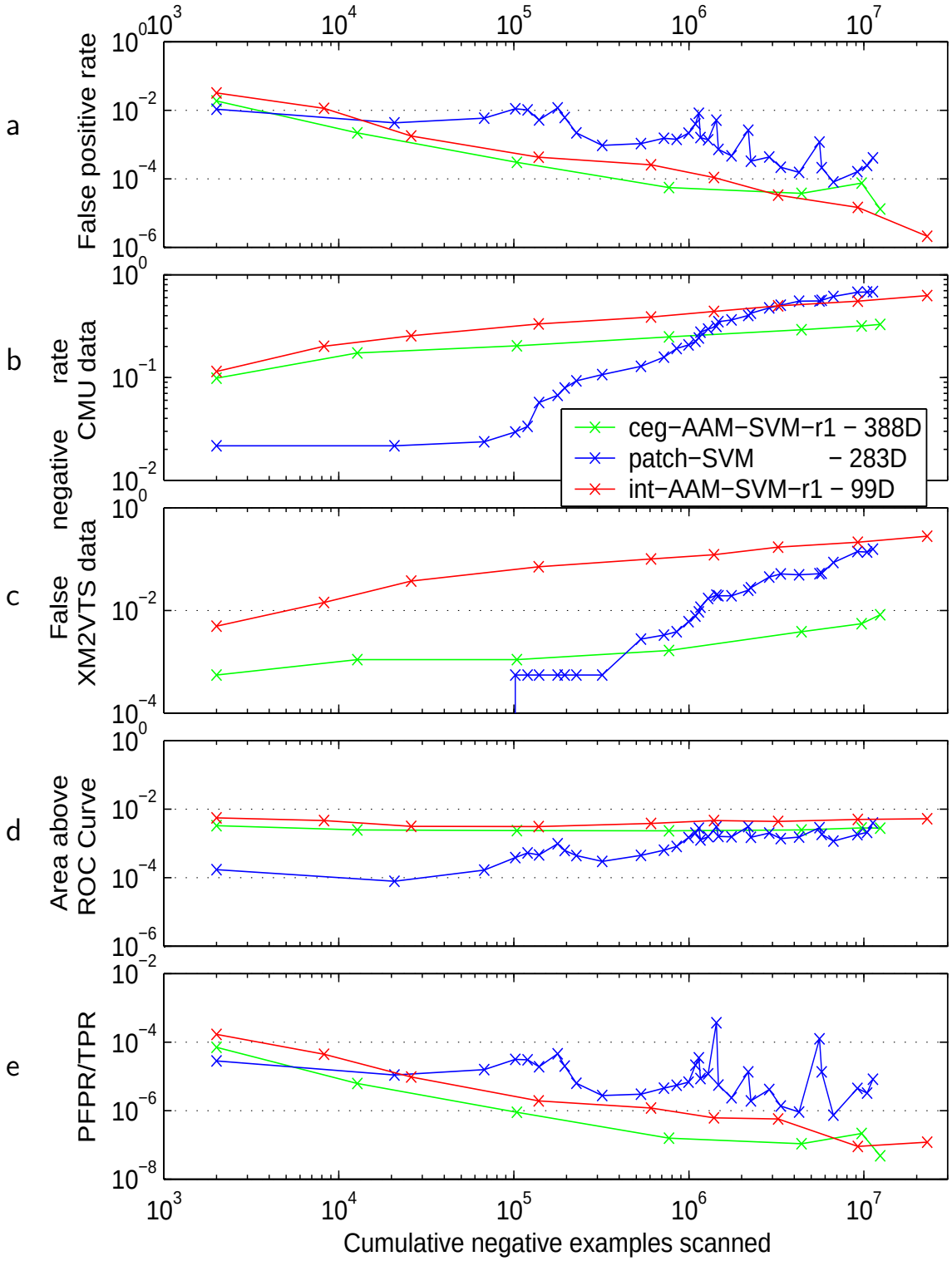



**Figure 10.1:** Comparing the performance of: the medium-resolution residual, corner-edge-gradient AAM-SVMs (marked `ceg-AAM-SVM-r1` here, it is the $\boldsymbol{r}_1$ AAM-SVM elsewhere); the Osuna *et al.*-style patch feature vector SVM detector (`patch-SVM`); the medium-resolution residual, original intensity AAM-SVM (`int-AAM-SVM-r1`.) The first iterations of the `patch` SVM had a perfect false negative rate on the `XM2VTS` database, and so are off the bottom of the graph. (See figure 9.6 for a description of the layout.)

`patch` SVM with the AAM-SVMs in terms of number of negative training images used, the blue `patch` SVM curves should be shifted right by a factor of four.

The spikes in the false positive curve (figure 10.1) for the `patch` SVM were investigated. The spike at $1.14 \times 10^6$ patches corresponded to the end of the first image (figure B.8a in the appendix) in the `UWash` database. The next two spikes (at $1.43 \times 10^6$ and $2.18 \times 10^6$ patches) occur part way through the second image (figure B.8b.) The next spike at $5.57 \times 10^6$ patches was in the forth image (figure B.8d.) The most likely explanation is the that these spikes represent new texture areas that the classifier training has not encountered before. A similar explanation can be made for the single spike near the end of the $\boldsymbol{r}_1$ AAM-SVM experiment.

Osuna *et al.*[108] reported their sensitivity on a subset of the `CMU` database to be 89.5%, with a false positive rate of $2.39 \times 10^{-6}$ on the same `CMU` database. This experiment has not reached that absolute performance, almost certainly due to a lack of representative variability in the `all22r` training set. This experiment has compared the AAM-SVM with the Osuna *et al.*-style classifier on identical training and test databases and the $\boldsymbol{r}_1$ AAM-SVM considerably outperforms the `patch` SVM.

### 10.1.2 Comparison of the Original and Improved AAMs in an AAM-SVM

The original intensity AAM only has a single residual per pixel. Since most classification algorithms (including the SVM) benefit from a more compact representation of the same information, it is important to confirm that this effect does not override the other benefits of using the improved corner-edge-gradient Texture AAM with ellipsoidal limiters. The standard $\boldsymbol{r}_1$ AAM-SVM experiment was repeated with the original intensity AAM. No other details were changed, e.g. the starting grid separation was not re-estimated. The results are shown in figure 10.1, with the corner-edge-gradient AAM-SVM marked `ceg-AAM-SVM-r1`, and the original intensity AAM-SVM

marked `int-AAM-SVM-r1`.

The performance of the original intensity $\boldsymbol{r}_1$ AAM-SVM is slightly, worse than the now standard, improved, corner-edge-gradient AAM-SVM. The improvement is particularly clear in the false negative rate measured on the `XM2VTS` set (figure 10.1c.) The much better false negative rate of the improved AAM in the `XM2VTS` database is most likely due to its better fitting ability. This is not reflected in the much more difficult `CMU` database, suggesting that the AAM may need to be further improved to work well on this database.

The hypothesis that using an AAM to extract feature vectors improves the performance of SVMs has a complement—that using an SVM improves the discrimination performance on AAMs. To examine relative behaviour of the AAM and AAM-SVM, the `ceg-AAM-SVM-r1` detector (built just before the training had examined $1\times10^7$ negative patches) was retested on $2\times10^6$ unseen patches (from patch numbers $1\times10^7$ to $1.2\times10^7$.) The residual magnitude $E$ of the final high-resolution layer of the AAM was recorded during this search for false positives. The detector was also retested on the `CMU` and `XM2VTS` databases, and $E$ recorded. This whole experiment was further repeated with the `int-AAM-SVM-r1` detector.

The performance of the AAM residual-magnitude based detectors (marked `ceg-AAM-E` and `int-AAM-E`) is compared to the performance of the full AAM-SVM-based detectors in figures 10.2 and 10.3. Of particular interest are the best sensitivity for perfect specificity points on the curves. As can be seen, the SVM makes a very large contribution to the overall performance of either AAM. As could have been expected from the results of previous chapters, the corner-edge-gradient AAM residual magnitude was much better at discriminating faces from non-faces than the intensity AAM residual magnitude.

The patch SVM built by the final training iteration was tested on $2\times10^6$ patches spread evenly over the same unseen images as the examples tested in the previous experiment. The results are also shown in figures 10.2 and 10.3. As could be expected

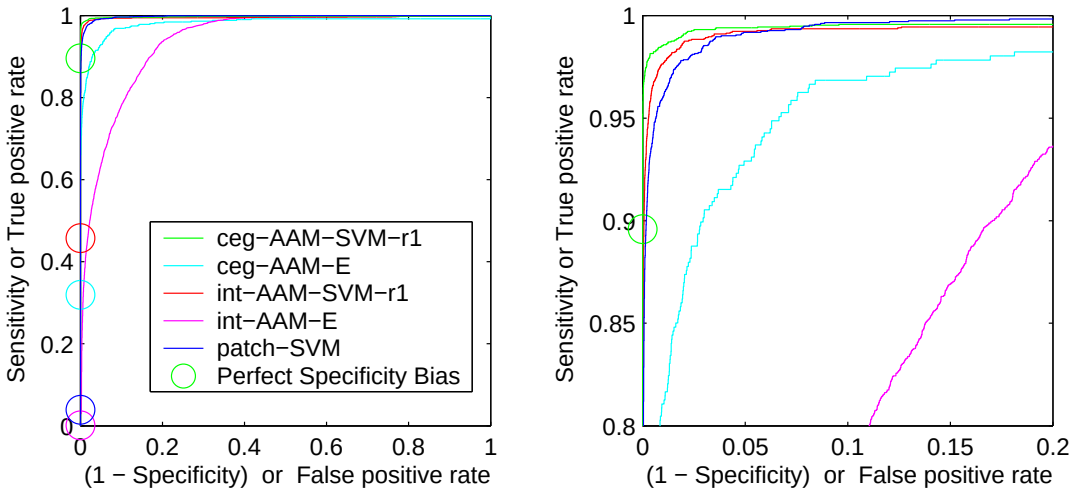



**Figure 10.2:** The ROC curves of the corner-edge-gradient $\boldsymbol{r}_1$ AAM-SVM (`ceg-AAM-SVM-r1`,) the intensity $\boldsymbol{r}_1$ AAM-SVM (`int-AAM-SVM-r1`,) and of the residual magnitudes alone (`ceg-AAM-E` and `int-AAM-SVM-E` respectively.) The performance of the Osuna *et al.*-style patch classifier (`patch-SVM`) is also included. The biases giving perfect specificity for each experiment are marked. The right-hand graph is a magnified version of the left-hand graph.

**Figure 10.3:** Specific values from the ROC curves in figure 10.2.

| Detector method | Area Above ROC Curve | Equal Error Rate | Best Sensitivity for perfect specificity |
|---|---|---|---|
| `ceg-AAM-SVM-r1` | 0.0028 | 0.013 | 0.90 |
| `ceg-AAM-E` | 0.019 | 0.059 | 0.32 |
| `int-AAM-SVM-r1` | 0.0049 | 0.017 | 0.46 |
| `int-AAM-E` | 0.061 | 0.14 | 0.0013 |
| `patch-SVM` | 0.0025 | 0.0215 | 0.039 |

the patch SVM generally outperforms the AAM residual-magnitude based classifiers. The patch SVM's low sensitivity at perfect specificity is surprisingly bad. Whilst the patch SVM saw as many negative training examples as the AAM-SVMs, that is only about a quarter the number of images, and may be the cause of this low result. However, that makes the `ceg-AAM-E` detector's sensitivity at perfect specificity very impressive, since the AAM training involves no negative examples.

It is interesting how much improvement in performance the SVM adds to the ordinary intensity AAM. Presumably the SVM is learning to distinguish faces, even when the intensity AAM has not fitted particularly well to the face. By switching from the intensity AAM with box limits to the corner-edge-gradient AAM with ellipsoidal limits, the experiments of previous chapters strongly suggest that the AAM should fit better. To confirm that this is actually happening on the database, the lowest point-to-point error for each face in the `XM2VTS` and `CMU` databases was recorded for both AAMs. Figure 10.4 shows that the `ceg-AAM-SVM-r1` detector had the benefit of significantly lower AAM fitting errors over the `int-AAM-SVM-r1`. The mean of the normalised error distribution has fallen by 21%, and the fraction of results with a normalised error worse than 0.1 has fallen by 63%. These are of the same order of magnitude as the improvement between the `int-AAM-SVM-r1` and the `ceg-AAM-SVM-r1` detector.

Finally, figure 10.5 shows the direct correlation between the AAM's best fit point-to-point error for each face in the `XM2VTS` and `CMU` databases, and the final SVM classification score. (These results are from the same classifiers as above, each having examined slightly less than $1 \times 10^7$ patches during training.) The scatter plot clearly shows that a lower fitting error leads to higher classification score. A simple linear relationship against the point-to-point error explains slightly more 30% of the total variation in classification score for each AAM. It is not possible to strictly separate the effects of a particular face's innate *difficulty* on the ability of the AAM to accurately fit it, from the ability of the SVM to accurately classify the fit. (A gold standard measure of a face's difficulty would be required for that.) However, the general trend in figure 10.5, and the overall improvement from the `int-AAM-SVM-r1` to the

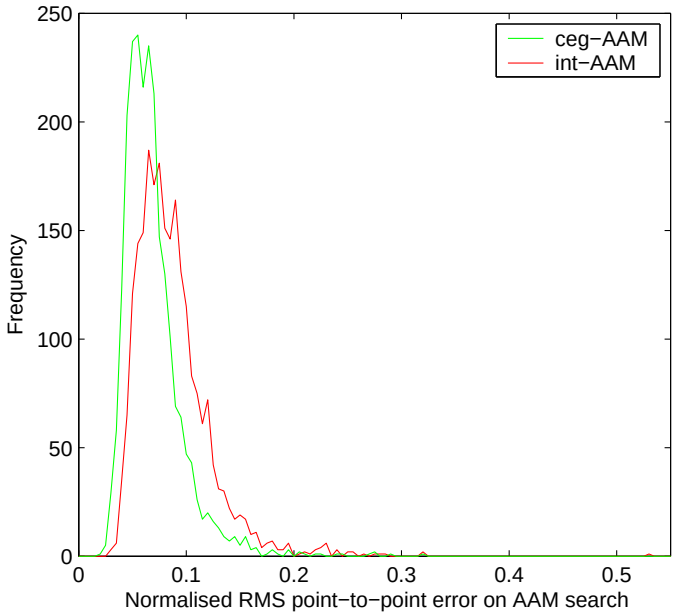



**Figure 10.4:** Distribution of normalised fitting errors on the 2123 known faces of the XM2VTS and CMU databases. (For scale: the interocular distance is about 1.2 normalised units; the pixel width at level 1 of the multi-resolution AAM is about 0.25 units.)

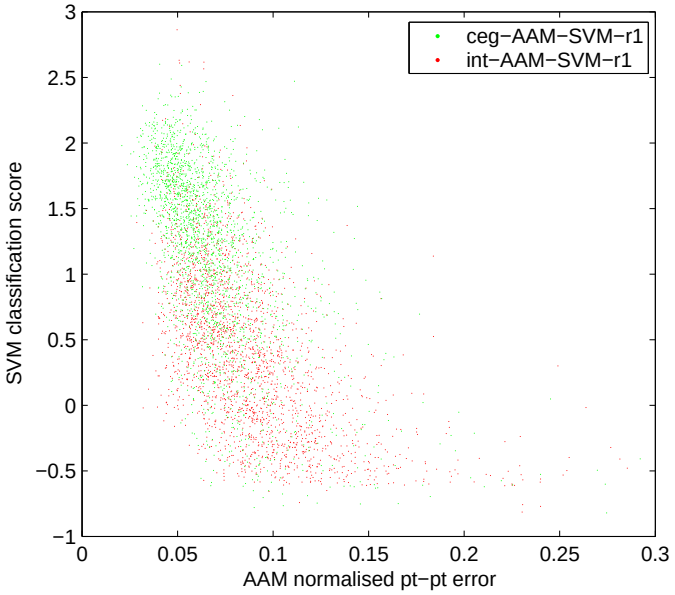



**Figure 10.5:** Correlation of SVM classification score (at neutral bias) against AAM fitting error on the 2123 known faces of the XM2VTS and CMU databases.

**Figure 10.6:** Two of the four faces from the CMU databases that both the original and the improved AAM failed to fit accurately. The two best attempts by the improved AAM are shown.

`ceg-AAM-SVM-r1` detector, lead to the conclusion that improvement of the AAM fit causes improvement in the overall AAM-SVM detector performance.

The SVM score improves even as the point-to-point error falls well below a single pixel. This sub-pixel accuracy is therefore one of the reasons why AAM-SVMs perform better than simple patch SVMs.

There are some faces (e.g. figure 10.6) that the AAM is currently incapable of fitting at all accurately . In these cases, if the fitting performance is improved such that the faces can be fit accurately, then overall detector performance will definitely improve.

## 10.2 Can Faces be Detected in 100 Pixels?

There is strong evidence from figure 9.7 that $\boldsymbol{r}_1$, the residual from the 100-pixel layer of the AAM, is adequate to test for the existence of the face. Therefore it is interesting to consider if a single-level AAM of 100 pixels would be sufficient overall. The lowest resolution layer of AAM search does most of the work to move the control points to their optimal position, with each subsequent layer only providing further accuracy. This further accuracy should be mostly less than a pixel as measured in the initial layer. Since the actual pixel values are sampled using linear interpolation, there

should not be a substantial amount of extra information provided by the roughly half-pixel improvement in accuracy. The speed advantage in not having to do any AAM calculations at the higher resolution would be significant.

To test this idea, the previous experiment was repeated but the AAM search took place only in the 100-pixel layer of the AAM. The results of this experiment (`ceg-l1-AAM-SVM-r1`) are compared with the standard (`ceg-AAM-SVM-r1`) experiment in figure 10.7. We can see that, even though the sensitivity (figures 10.7b and 10.7c) is a little lower, there is very little difference in overall performance.

Whilst the `ceg-l1-AAM-SVM-r1` detector is only sampling the texture maps at 100 pixels, those texture maps are generated using a 5×5-tap filter. So really, $\sim$ 180 pixels are being used to detect the face[1]. This is still smaller than others have reported. The widely used mask introduced by Sung and Poggio[145] is 283 pixels. The cascading AdaBoost classifier of Viola and Jones[161] has a base resolution of 24×24 pixels, of which the face takes up about 400. The ability to detect a face in such a small number of pixels, surely depends on the AAM-SVM's ability to choose the right pixels.

### 10.2.1 Algorithm Speed

One of the main benefits of using the `ceg-l1-AAM-SVM-r1` over the `ceg-AAM-SVM-r1` detector is speed. If the AAM does not have to go into the high-resolution layer 0, or fit the control-points found in layer 0 back into layer 1, this is all time saved. With this in mind, the speed of several AAM-SVMs was measured. The results are compared in figure 10.8. The speeds of the AAM-SVMs did not vary much during the course of training, simply because the size of the training set, and number of support vectors did not have the opportunity to grow significantly. The speed of the patch-based SVMs slowed by a factor of four, as the number of support vectors increased from 3847 before refinement to 12,193 after 31 refinement iterations.

[1]The outermost pixels have less than 10% of the total influence on any sample, so the system could probably get away with only $\sim$ 140 pixels.

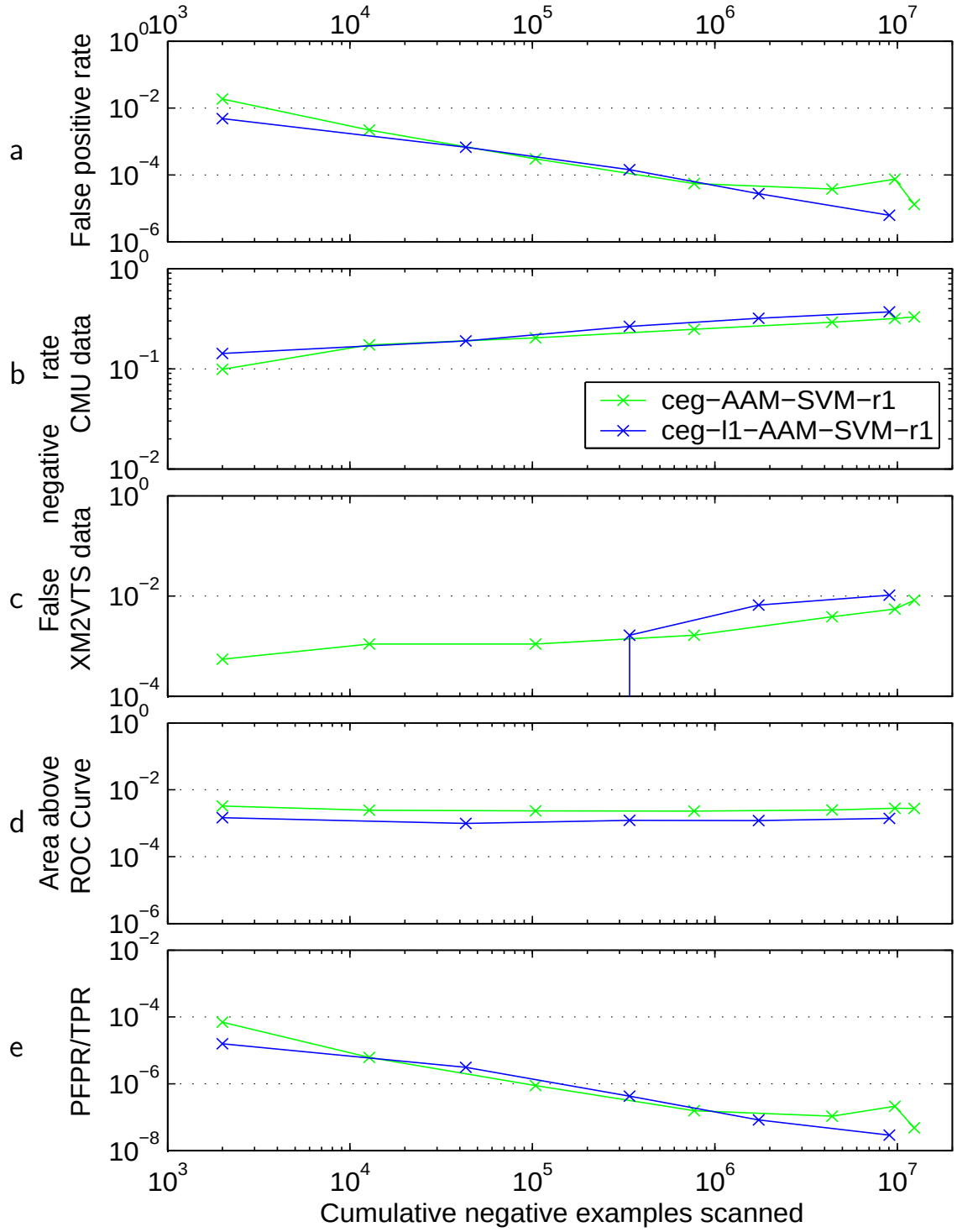



**Figure 10.7:** Comparing the AAM-SVM face detection performance between the normal two-level multi-resolution AAM search (`ceg-AAM-SVM-r1`) and just using the 100-pixel single-level during AAM search (`ceg-l1-AAM-SVM-r1`.) The $\boldsymbol{r}_1$ residual is used as the SVM's feature vector in both cases, but in the former (original) case, the control points' position is determined by the optimal fit for the AAM with respect to the 400-pixel residual $\boldsymbol{r}_0$. (See figure 9.6 and section 9.7 for a description of the layout.)

**Figure 10.8:** The speed of various detectors, and the time taken to train the SVM each refinement iteration. The speeds were measured on 1.7GHz Pentium4 after each classifier was at a similar stage in training—at $\sim 10^7$ negative samples searched.

| Detector | Detector speed (patches per minute) | SVM training time (minutes) |
|---|---|---|
| ceg-AAM-SVM-r1 | ∼650 | ∼55 |
| ceg-AAM-SVM-r0 | ∼480 | ∼35 |
| ceg-AAM-E | ∼770 | |
| int-AAM-SVM-r1 | ∼5000 | ∼55 |
| patch-SVM† | ∼2200 | ∼1650 |
| ceg-l1-AAM-SVM-r1 | ∼2300 | ∼45 |

† The patch-based SVM has to test four times as many patches per image as the AAM-SVMs so the equivalent speed is really about 550 patches per minute. Since the patch-based SVM's training could not fit all the kernel values into the cache, the training time would be very dependent on the cache size. It was set to 512MiB for these experiments.

One unexpected result is that the extra support vectors needed by the patch SVM pushed its computational costs well above the computational costs of the AAM-SVM. The availability of much faster cascade version of SVM (and AdaBoost,) unfortunately undermines the value of this result. Given the much greater time needed to run the SVM training for the patch-based SVM classifier—it would more efficient in future work to search for more than $n_{\text{select}} = 200$ false-positives per refinement iteration.

## 10.3 Iterative Refinement with Positive Data

The idea of iterative refinement of the classifier, introduced to this field by Sung and Poggio[145], has so far only been used on the negative examples. This section examines two ways of using it with positive examples.

### 10.3.1 Starting with a Small Positive Database

If a good enough detector existed, which could find some true positives in the search of a database, then these could be added to the training set and the whole detector rebuilt. This should lead, in turn, to a better classifier which could find more true positives. In order to examine the viability of this approach, the $\boldsymbol{r}_1$ AAM-SVM was rebuilt using two much smaller databases. One, named `10th-22r`, contained a $10^{\text{th}}$ of the original `all22r` database. The other, named `100th-22r`, contained a $100^{\text{th}}$. As in previous experiments, half of each training database was used to train the AAM so that the SVM could learn about the AAM's response to unseen images.

With the described changes, the standard AAM-SVM experiment was repeated. The results are shown in figure 10.9. The smallest training database `100th-all22r` had only 40 faces in it, and this does not appear to be enough to allow the SVM to accurately set the SVM's bias. The false positive rate was so high (figure 10.9a,) that the AAM-SVM found no false positives in the first four million tests of the negative training database.

Looking at the point-to-point error distribution of the AAMs (in figure 10.10) the 20 examples of half the `100th-22r` database does appear to be enough to produce a good AAM for face location purposes. This means that it should be possible, in future work with a larger database, to increase the proportion of SVM training examples (currently 50%) that are unseen during AAM training, which should improve the detection performance.

The inability to use a small database to build an AAM-SVM classifier capable of refinement need not be a problem. They are capable of detecting some faces. The `100th-all22r` AAM-SVM built from 40 faces was able to find another 56 faces even in the difficult `CMU` database. The very high specificity, a problem for refinement training, is now an advantage. When run over a large database, the detector will be able to find some faces, without returning any false positives. A human could, at this stage correctly label the results, and add the real faces to the positive training

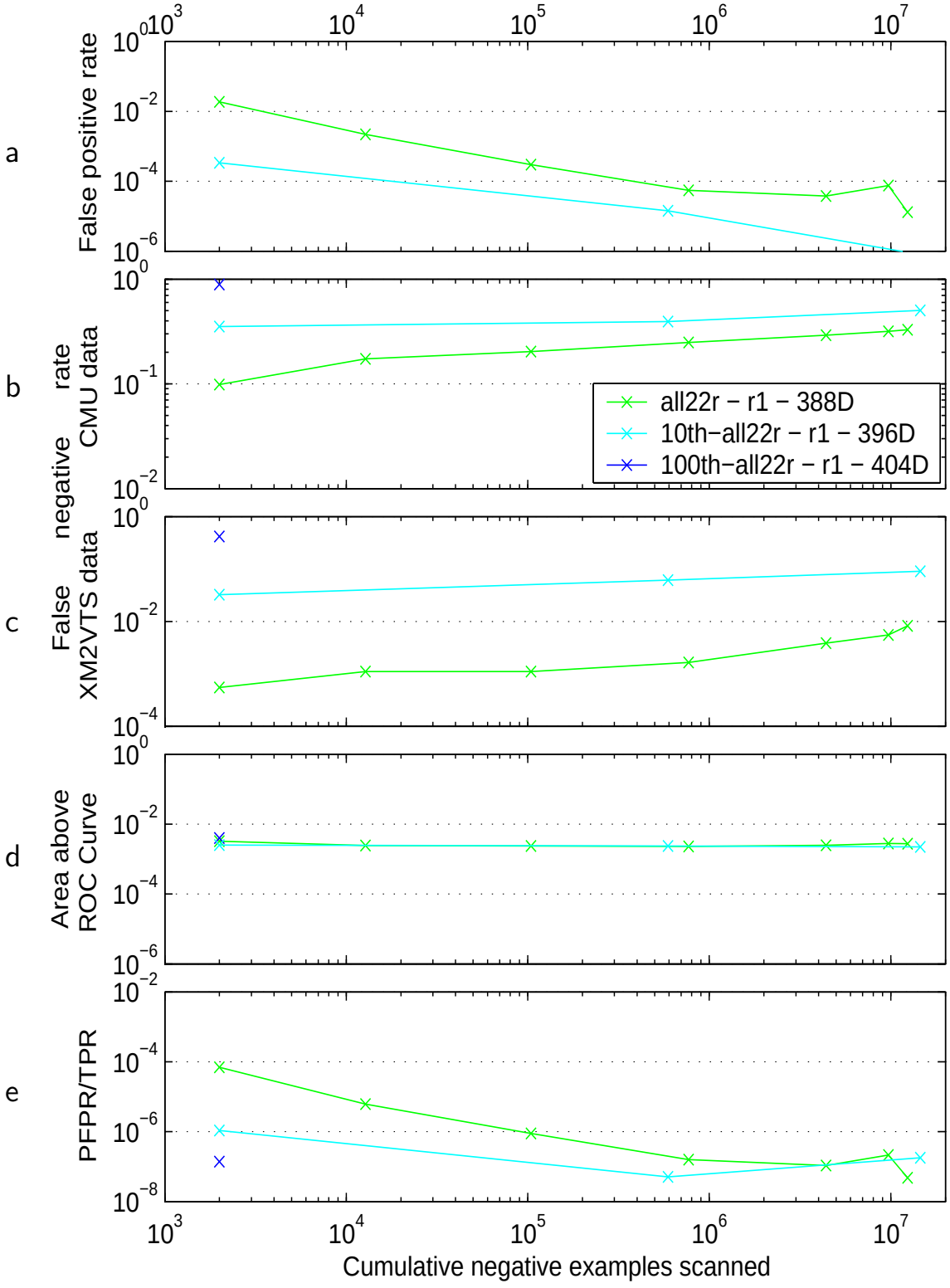



**Figure 10.9:** The performance of the standard `ceg-AAM-SVM-r1` during training on the original `all22r` database, and on two smaller subsets—`10th-all22r` and `100th-all22r`. (See figure 9.6 and section 9.7 for a description of the layout.)

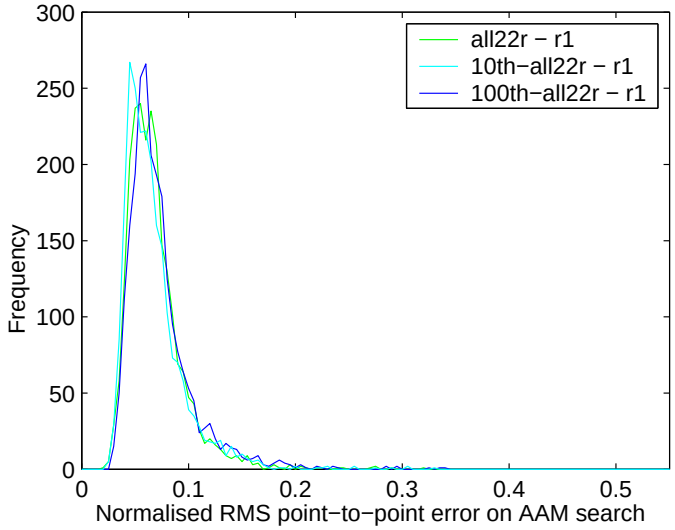



**Figure 10.10:** Distribution of normalised fitting errors on the 2123 known faces of the `XM2VTS` and `CMU` databases. (For scale: the interocular distance is about 1.2 normalised units; the pixel width at level 1 of the multi-resolution AAM is about 0.25 units.)

database, and the false positives to the negative database. This would improve the AAM-SVM's performance, and allow it to find more true examples in the database.

More work would be needed to understand when this iterative process would converge. It may well converge very quickly without having found a lot of the object examples that do not look enough like the original training database. The results from the 398-example `10th-all22r` database, where after 2 refinement iterations, the AAM-SVM could find about half of the faces in the `CMU` database are also encouraging.

### 10.3.2 Increasing the Size of the Positive Training Set Without a Larger Database

In any of the AAM-SVM experiments, the initial positive training set passed to the SVM consists of examples obtained by fitting the AAM to the known points of its labels. This is not strictly relevant training data since, during test, the SVM will only see examples that the AAM has (locally) optimally fitted. Even accounting for this, figure 9.2 shows that AAM search does find local minima away from the

labelled points. By finding these local minima on the training database, and adding them to the training set for the SVM, the overall classification performance should be improved.

The minima can be found by starting the AAM from different initial positions near to the labelled faces, and running AAM search. The pose displacements were selected randomly from a box distribution 50% larger than the search grid, and also included a $\pm 20°$ rotation. After AAM search, the point-to-point error can be measured. If the error is close enough to represent a possible fit, it is added to the SVM training set. Close enough is defined (as per section 9.5.2) to be a point-to-point error of less than 0.1. The reason not to use fits that were classified as "not definitely unsuccessful" (i.e. error of less than 0.8,) is because it is unclear whether fits with point-to-point errors between 0.1 and 0.8 should be considered good fits. Some of them look intuitively to be reasonable fits, but some do not. It is important to avoid poisoning the SVM's positive training set with bad examples, so only fits with very low point-to-point error are added.

To test this idea, the final `ceg-l1-AAM-SVM-r1` detector from section 10.2 was put though several more refinement iterations. In each of these nine new iterations, the positive training database was searched 10 times per face. On average, 40% of the searches had too high a point-to-point error. The remainder were tested by the SVM, and any false negatives added to the positive training set. In the first additional refinement iteration, 725 false negatives were found, out of the ∼24,000 successful searches. This fell to 12 new false negatives in the next iteration. The remaining iterations found a total of 17 new false negatives.

During each refinement iteration, the negative database was also searched, in order to estimate the effect of the false negatives on the false positive rate. For all but the last iteration, the search for false positives was stopped after $n_{\text{select}} = 10$ had been found. For the final iteration, the search was not stopped until the more usual $n_{\text{select}} = 200$ false positive examples had been found.

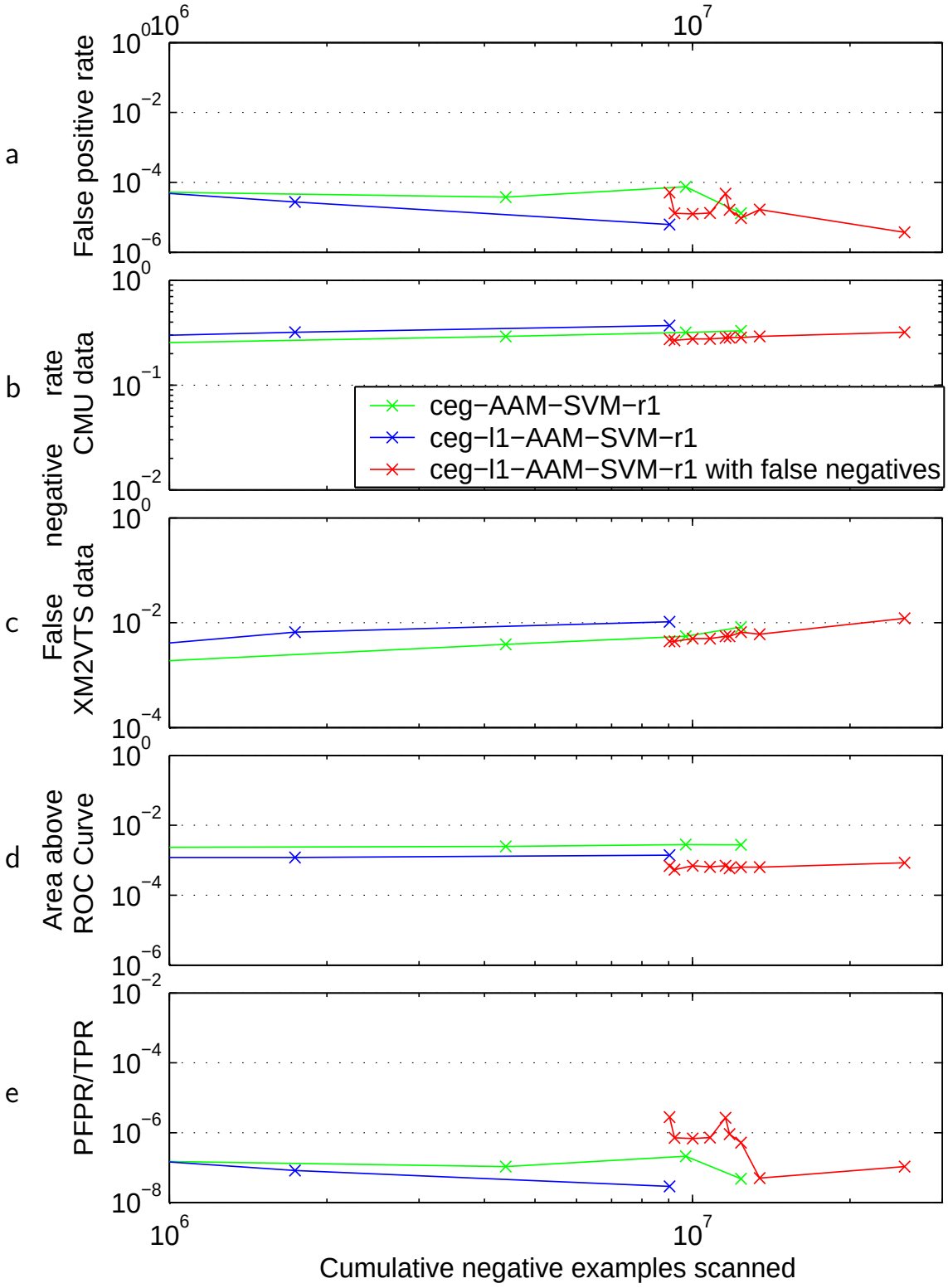



**Figure 10.11:** The standard `ceg-l1-AAM-SVM-r1` detector trained with additional false negatives from the positive training set. The original `ceg-l1-AAM-SVM-r1` and `ceg-AAM-SVM-r1` results are shown for comparison. (See figure 9.6 and section 9.7 for a description of the layout.)

The results (along with the previous `ceg-l1-AAM-SVM-r1` and the original `ceg-AAM-SVM-r1` results) are shown in figure 10.11. The PFPR/TPR measurements (figure 10.11e) are not easily comparable here, due to the reduction in $n_{\text{select}}$ and hence the reduction the potential number of negatives that could be successfully classified with the bias set for a perfect false positive rate. Nevertheless, it appears that the addition of the false negatives has not improved the detection performance. This is perhaps due to the false negatives not being added to the training set early enough in the refinement process to be useful. Given more time, this experiment should be repeated but with the extra false negatives added every iteration.

## 10.4 Multistage Approach to Object Detection

One obvious method of speeding up the AAM-SVM as a face detector is to use one of the fast face detection methods as an initial guess. In some experiments performed mainly by David Cristinacce (a fellow Ph.D. student in ISBE,) but with my collaboration, a 400-pixel improved corner-edge-gradient AAM was used as the final fitter in a face localisation context. Initially, the cascaded-AdaBoost face detector[161] was used to find the most likely face position in an image. Next, a set of pre-learnt regions around the face were searched with AdaBoost detectors trained on small patches around each labelled point of the training set, and the most likely three candidate positions located. To find the maximum likelihood locations of the label points, histogram models of the distributions of each label point given each candidate location were combined over all the candidate locations. This combination of feature detectors and pairwise histograms is called a "Pairwise Reinforcement of Feature Responses" (PRFR) model. Finally a corner-edge-gradient AAM with ellipsoidal limiters was initialised on the optimal candidate locations, and normal AAM search was run.

The face detector, PRFR model and AAM were trained on a 1055 face `webcam2` database collected within the lab, and marked-up with 17 points. It was tested on the `BioID` database—see figure B.3. Figure 10.12 shows the cumulative distribution of

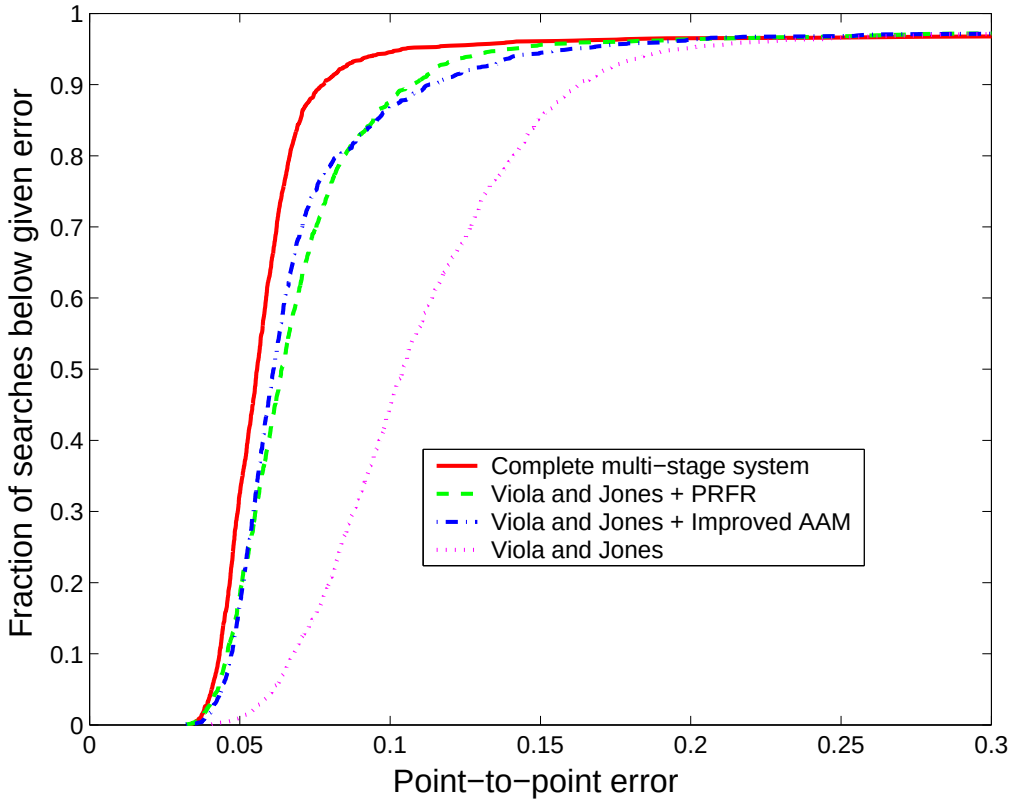



**Figure 10.12:** The cumulative distribution of point-to-point errors for several combinations of stages of the multi-stage face localisation method.

point-to-point errors found by the complete system. The errors have been normalised by dividing through by the interocular distance. The complete system clearly gives the most accurate face location.

Figure 10.12 also shows that the average control point locations for the best face candidate, found by the Viola and Jones-style face detector, are not very accurate. The best face candidate is also completely wrong for 3% of the test set. Since the rest of the system currently depends on the single best face candidate, the whole system has at least this amount of error. Future work will investigate using several of the top face candidates, and the AAM-SVM to make the final decision.

On a 500MHz Pentium2 computer, the global search takes $\sim$ 300ms, PRFR takes $\sim$ 800ms, and AAM search takes another $\sim$ 300ms, for a single image. Taking account

of the differences in resolution of the faces being searched for, this is about 700 times faster than the `ceg-l1-AAM-SVM-r1` detector could search the same image.

## 10.5 Statistical Significance

It is useful to quantify the statistical significance of the differences between the various AAM-SVM methods. Without over-generalising, one can see that the *Perfect* False Positive Rate to True Positive Rate ratio (PFPR/TPR) graphs of all the different AAM-SVMs are approximately straight (figures 9.7e, 9.9e, 10.1e, 10.7e, 10.9e and 10.11e.) That is, the graphs could be represented by a straight line, plus some short-term deviations from the long-term trend. Note that there is not enough evidence (i.e. long enough experimental runs) to make this statement confidently, but it is a very simple model that the results do not contradict. The model has a simple interpretation. A particular AAM-SVM has a given performance for any fixed number of scanned negative training examples, which is related to the intercept of the line. Increasing the number of negative samples scanned $n$-fold, reduces the PFPR/TPR, $\xi n$-fold, where $(-\xi)$ can be thought of as a marginal efficiency, and is equal to the negative slope of the line.

$$log\,\text{PFPR/TPR} = \xi log\,\text{NN} + C_0$$

$$\text{PFPR/TPR} = C_1 \text{NN}^{\xi}$$

where $C_0$ and $C_1$ are constants, and NN is the number of negative examples scanned. The (negative) $y$-axis intercept, $-C_0$, could be seen as an theoretical initial performance when given no negative training examples, and short of orders of magnitude difference, it is probably unwise to read too much into it.

Using simple linear regression (the single variable case of the methods described in section 7.1.2,) straight lines were fitted to the PFPR/TPR graphs. The estimates and 95% confidence intervals for the slope and intercept of the fits are shown in

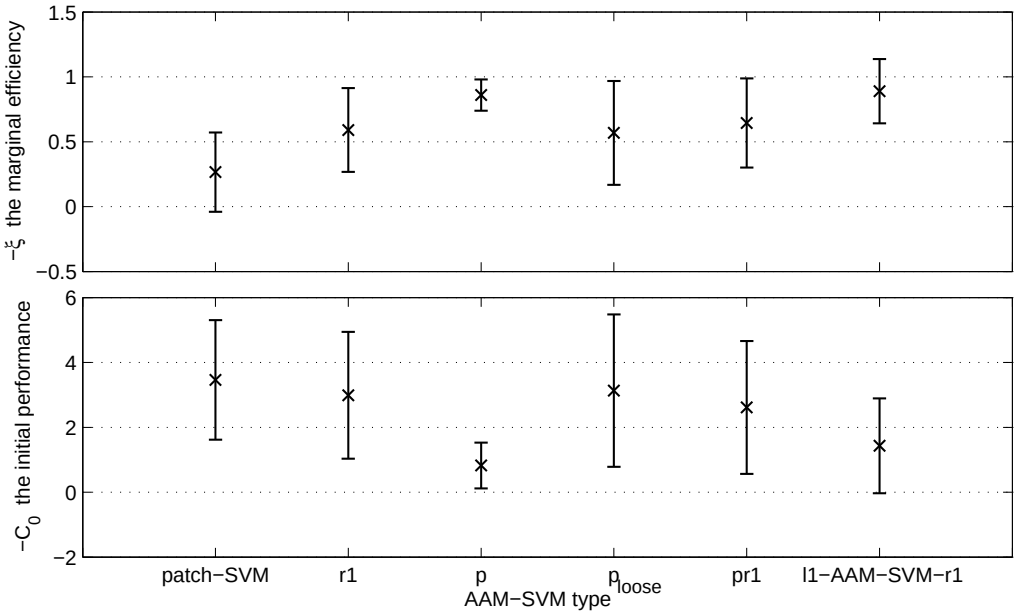



**Figure 10.13:** The initial performance, and marginal efficiency of face detectors, including the `patch-SVM` and various AAM-SVMs. The confidence intervals are at 95%.

figure 10.13. It shows that in the long run, there is not be any statistically significantly differences in asymptotic performance between the various AAM-SVMs. Although, if we consider the AAM-SVMs as a whole, there may be a significant difference between the AAM-SVM's and the patch-SVM's asymptotic efficiency. The $R^2$ values for all of the AAM-SVM linear fits were between 0.83 and 0.98. The Osuna *et al.*-style patch SVM had a much less confident straight line fit, with an $R^2$ value of 0.10, putting some doubt on the reliability of any inferences.

The wide confidence interval on the slope and intercept of the $\boldsymbol{r}_0$ AAM-SVM is due largely to the very few number of data points. However, due to the $\boldsymbol{r}_0$ AAM-SVM's superior performance, many more samples were tested to get each of those data points than in other AAM-SVMs, and so the error on those data points should be lower. No consideration was taken of the intrinsic error on those data-points in the statistics reported above. Measuring and using these errors with generalised linear modelling is likely to weight the fit to the data points on the right of the five graphs (in each of figures 9.7, 9.9, 10.1, 10.7, 10.9 and 10.11,) where more samples per data point will have reduced the error. Such an improvement in the statistics may have a large effect

on the slope estimates as well as the confidence widths, but this will be the subject of future work.

## 10.6 Discussion and Conclusions

This section discusses whether AAM-SVMs could be used for image database search, as well as the broader implications of the results of this and the previous chapter. Potential variations on the AAM-SVM and possibilities for future work are also examined.

### 10.6.1 Implications of the AAM-SVM Results for Appearance Modelling

The fact that the small training database AAMs work so well (figure 10.10) may be related to the evidence in section 8.5 that the PCA is poorly estimating the true distribution. There is evidence[39] that restricting the number of modes to well below the number necessary to explain 99% of the total variation, can improve model matching performance. 20 training examples effectively restricts the number of learnt model parameters to no more than 20. Either 20 modes really is enough for good AAM search, or there is good use to be made of more than 20 training examples. Either way the PCA's estimate of the principle variances, and of the dimensionality of the principle manifold, is suspect.

Further evidence that too many modes are being included in the principle component space comes from the surprisingly accurate ability of the $\boldsymbol{p}$ AAM-SVM's (section 9.8.2) to distinguish faces from non-faces. One explanation for this is that some of the modes are modelling noise as well as the variation of valid faces. The SVM could thus be learning to discriminate on the basis of these noise modes.

The number of modes for optimal search can be picked using cross-validation. However, this author is always reluctant to use cross-validation during training. Doing 1-out-of-n cross-validation is very slow, but using any coarser grain will often result in a biased estimate of optimal model parameters. For example, figure 4.20 shows that decreasing the SVM training set size, would significantly inflate the estimate of the optimal RBF width. More importantly, the use of cross-validation during training implies a lack of knowledge of the behaviour of that system, and this author would prefer to obtain that knowledge. Of course, cross-validation's ability to work with no assumptions makes it invaluable—particularly for testing, when as few assumptions as possible should be made. However, the best way to deal with the poor estimates of the number of model parameters (and their variance) will come from investigating their cause.

Similar to the results in section 4.6, the RBF width of the optimal SVM was larger than the range of each element from the feature vector. The typical RBF width was $\sim 2.5$ compared to texture samples that were normalised to $[0, 1]$. This suggests that a distribution, like the Gaussian, that has a smooth iso-probability surface is still the correct choice for the appearance model's PDF.

### 10.6.2 Future Improvements to the AAM-SVM

There are several aspects of the design of the AAM-SVM configuration that could probably be improved in future.

#### AAM Starting Grid

Before explicitly thinking about the size (in pixels) of the AAM that would be required for the work in this and the previous chapter, the author had implicitly assumed that the radius of convergence of the AAM would be ten or a hundred times bigger than a single pixel. The extra cost of performing AAM search over patch-based methods

would be largely offset, by the need to search much less frequently than once per pixel. Unfortunately, the measured radius is two pixels, leading to only a four fold decrease in searches. Rolling the scale and displacement into one commensurate measure of radius of convergence was crude, and separately measuring the scale radius might allow for a larger step between AAM starting scales. Much more than another two-fold improvement is unlikely however. The lack of significant numbers of outliers in the face fitting error of the original intensity AAM (figure 10.4,) implies that the starting grid was closely enough spaced for the intensity AAM. Therefore the starting grid may be more dense than necessary for the improved corner-edge-gradient (ceg) AAM. Increasing the grid spacing to 3×3 pixels would more than double the speed of the AAM.

### Reducing the AAM or Feature Vector Size

The belief within ISBE that the minimum workable size of an AAM is about 100 pixels, might not be true of the improved AAM. It is also likely to be based on more complicated shape-models that the 22-point versions used here.

Anecdotal results showed that a 22-pixel improved AAM trained on the `half-all22r` data was able to converge on the majority of the faces in the `CMU` database. (Unsurprisingly the 6-pixel layer was completely unable to converge on even easy faces, such as are in the `XM2VTS` database.) Every pixel removed from the model will improve the speed of the method.

The most likely reason why the higher resolution feature is not always unambiguously better (see figure 9.7,) is that the $\boldsymbol{r}_0$ feature vector is too long. Whilst SVM classifiers can cope with very long feature vectors, if the feature vector could be reduced whilst still keeping the same information accessible, the performance might be increased. As with the suggestion to reduce the AAM's size, stripping out the unnecessary elements from the SVM's feature vector should have a significant effect on the speed of the system.

**Better AAM Search**

The results from section 10.4 imply that the AAM search could be getting stuck in non-optimal solutions. By initialising the AAM with control points *better* than their mean position, AAM search finds a closer fit. This has implications for the ordinary AAM search algorithm, suggesting that it could still be significantly improved.

Further work is needed to find out if the closer fit has a lower AAM residual error, or is really just a local minimum. If the latter, then investigation of better cost functions would be warranted. If this closer fit really does have a lower residual error, then better texture descriptors, may smooth out the cost function surface, and allow AAM search to find the optimal minimum. Alternatively, every use of AAM search could be preceded by PRFR point location to get a more accurate fit.

### 10.6.3 Better SVM training

There is a connection between the iterative refinement used for classifier training and the SMO decomposition for SVM training. In an ideal world every single AAM search result from the negative image database would be added to the SVM's training set, and the optimisation would make use of every vector to find the ideal classification boundary. Unfortunately, in the real world it is not possible to store this entire training set in a computer's memory (let alone the Hessian required by simple SVM training algorithms.) The refinement iterations are an approximation to the SMO's heuristics for choosing the next pair of training vectors to examine. The approximation is that some training vectors may not be considered at all.

Can the iterative refinement method be modified to improve the approximation? Currently, false positive examples (and the false negatives in section 10.3.2) are added to the SVM's training set if they are on the wrong side of the classification boundary. In a perfect SVM algorithm, they would be considered if they were inside the margin, i.e. if there classification score was on the wrong side of $\pm 1$.

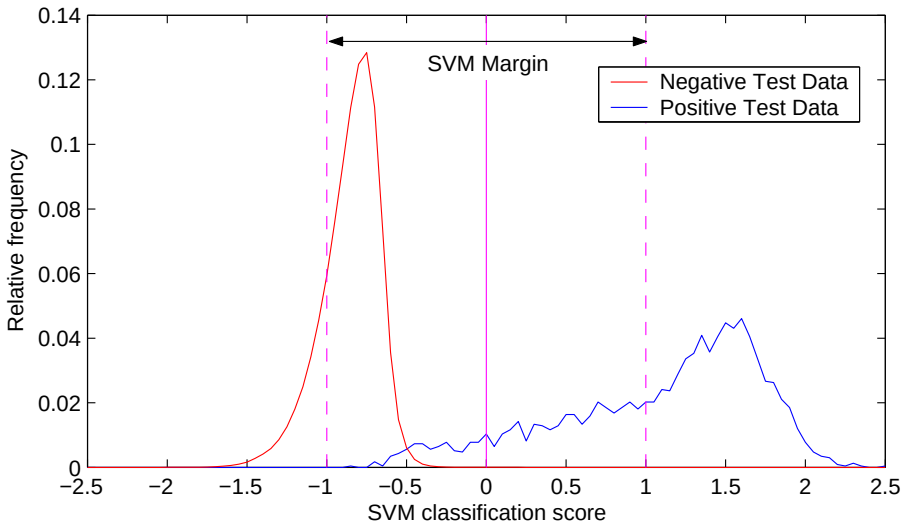



**Figure 10.14:** The distribution of SVM classification scores on unseen data from the final iteration of the `ceg-l1-AAM-SVM-r1` detector.

Figure 10.14 shows the distribution of SVM classification scores on unseen data from the final iteration of the `ceg-l1-AAM-SVM-r1` detector extended with false negatives from the positive database (section 10.3.2.) Most of the negative test data, and a significant fraction of the positive test data is inside the margin. Training might thus be expected to proceed more slowly, simply because it is much easier to find examples from the negative database inside the margin, rather than on the wrong side of the classification boundary. Further investigation would be necessary to see if this reduction in speed was compensated for by improved classification performance per iteration.

The SVM training method currently assumes that the face and non-face classes are strictly separable, and so the relative cost of margin encroachment, $C$, is set to infinity. This produces a classification boundary at the so-called neutral bias. During the analysis of the experimental results in this and the previous chapter, the biases are then adjusted, to calculate the AARC and PFPR/TPR statistics. However, Lin *et al.*[85] point out that this is not the ideal way to adjust the bias, because the adjusted margin position takes no account of any training vectors it may encounter as it moves. Instead, the relative cost of margin encroachment (which becomes the upper bound,

$C$, on the Lagrange multipliers in the SVM optimisation) should be split into $C_{\text{neg}}$ and $C_{\text{pos}}$. As $C_{\text{pos}}$ is reduced, the classifier bias would move against false positives. Further investigation would be needed to determine how much improvement in biased classifier performance would be obtained using this method, rather than simply adjusting the bias term in the final classification rule.

### 10.6.4 Variations on the AAM-SVM Method

This section discusses two, structurally very different, variations on the AAM-SVM.

#### Virtual Support-Vector Method

When treating the AAM as a very good form of patch normaliser, before normal statistical classification, it is possible to see an alternative arrangement. Schölkopf *et al.*[133] introduced the method of virtual support vectors for SVM classifiers. In this method, SVM is trained on an un-normalised training set. Then the normaliser is inverted, and applied to the learnt support vectors, to generate a set of examples to which the classifier should be invariant. These virtual support vectors are then added to the training set, and the SVM retrained. This method was employed by Romdhani *et al.*[121] to avoid having to normalise for lighting variation.

Instead of using an AAM to normalise every input patch, one could build a normal patch SVM detector from a limited, un-normalised, training set. After building an AAM from the same data, take each support vector, fit an AAM, and then randomly perturb the AAM parameters. In order for this to work with negative support vectors, one would need to record the texture residuals during fitting, and add them to the perturbed model to give the virtual negative support vector. This approach would then acquire the rest of the advantages and disadvantages of the patch SVM method, but with a significantly reduced training-set size requirement.

#### Cascade Methods

The cascade method of combining multiple classifiers in sequence from cheapest/lowest performance to slowest/highest quality, rejecting definite failures along the way, is well known as a method of speeding up algorithms. Indeed this is the basis for Viola and Jones[161], and Romdhani *et al.*[121]

Section 10.4 covers one obvious multi-stage approach. Although more work is needed to add a final classifier onto the end of the process, section 10.4 does give an insight into the expected performance of a future experiment. If the algorithm were working as a face detector, and checked at most 10 potential fits per image, compared to the 100,000 that the existing AAM-SVM detector has to check, then there is the potential for more than a thousand-fold increase in speed.

Given the large improvement (figure 10.2) in the discriminatory power of the residual magnitude $E$ when using improved AAMs, it should improve the performance of the hierarchical search method of Edwards *et al.*[51] (see section 9.1.2.) Further, a full SVM classification could be applied after the convergence of each level of the AAM.

A more powerful approach would be to use interest point detectors to suggest a small set of initial hypotheses, which could then be refined by AAM search. Indeed, this was one of the motivations for using the corner detector described in chapter 6. This might lead to a scheme similar to first using Weber *et al.*'s constellation model[165, 58] to quickly locate some hypothesis, followed by full fitting and selection using the AAM-SVM method.

### 10.6.5 Could the AAM-SVM be Used for Image Retrieval?

The classification performance found by the best AAM-SVM used here (the `ceg-ll-AAM-SVM-r1` detector—see section 10.2) is not yet good enough to be used for image database retrieval. Assuming approximately $10^5$ starting positions tested per

image, and that a given object is found on 500 images within a million image database, a false positive rate of $4.5 \times 10^{-9}$ at a reasonable true positive rate of 0.9, would be needed to find the objects while still returning a 50% precision to the user. The best result from the `ceg-l1-AAM-SVM-r1` detector has a false positive rate of $6.2 \times 10^{-6}$ for a true positive rate of 0.9. That is not good enough by more than three orders of magnitude.

Can these results be extrapolated, to the point where the classification performance would be good enough for retrieval purposes? One reason for doubting the validity of any extrapolation can be found in figure 4.20, where we can see the number of support vectors growing faster than the number of examples added to the SVM's training set. These trends cannot go on for ever.

Even assuming that the straight line fits to performance curves will extend, there is still a problem. It is unlikely that any of the true straight-line fits has slope, $\xi$, lower than -1.0. This implies that the asymptotic efficiency of the detectors are all worse than unity, or that to get a ten-fold improvement in classification performance will need ever more training examples as the performance improves. In other words—any detection method with $n$ training examples that gives a excellent PFPR/TPR such as $10^{-6}n$ on a small scale experiment, may still need a training set bigger than the test set for very large databases.

Obviously, the AAM-SVM is not currently fast enough to be used as a image database retrieval method, by several orders of magnitude. The single-level `ceg-l1-AAM-SVM-r1` detector could maybe search 5000 patches per minute on a modern CPU. To index a new million-image database would take about 40 years. Using the speed of the multi-stage face locator as an indicator, this time could be reduced to only a few weeks by a method that does not need to exhaustively search every possible face location.

### 10.6.6 Conclusions

It has been shown that the use of the AAM as an advanced feature detector improves the performance of a statistical-classifier based object detector. The use of the statistical classifier massively improves upon the AAM's easily-available measures as means of deciding whether the current fit really is an object, or merely a hallucination.

The success of the single-level 100-pixel AAM-SVM (`cegl1-AAM-SVM-r1`, section 10.2) shows that the resolution of detectable faces is significantly lower that previously demonstrated by others.

The AAM-SVM does not yet have a high enough specificity to be used as an object detector for image database search. A statistical analysis suggests that the AAM-SVM, and potentially other object-detection methods, may need negative training databases at least as large as the intended test database in order to achieve the necessary precision or specificity.

# Chapter 11

# Discussion

This chapter summarises the work described in this thesis, and highlights important directions for future research, to reach the goal of detecting objects for Content Based Image Retrieval.

## 11.1 Summary of Original Work and Results

### 11.1.1 Building Statistical Classifiers

Whilst harder to understand than the GMM network, the SVM was found to be a generally better performing classifier. It was also much easier to train (no local minima) and had fewer tunable training parameters.

No-one has previously reported adapting SMO to calculate the diameter on the MEH. Whilst the maths involved in this modification was simple, there are several numerical precision issues which needed to be dealt with for successful convergence. Several authors have suggested selecting the best RBF width (or other classifier parameters) on the basis of the VC dimension of the data, or other simple cost functions. However, these selections have all been described as manual processes. Whilst the final search for the minimum is straightforward, bracketing the correct minimum

successfully, efficiently and completely automatically is nontrivial and has not been described elsewhere. This eliminates the effort and potential errors introduced by manual intervention.

### 11.1.2 Improving the Accuracy of AAMs

AAMs (which were developed in this lab) have been widely studied and used. Whilst it was expected that there existed room for improvement, it was assumed that that this would be as a series of small steps. The addition of multiple texture descriptors to create the Texture AAM, and the use of more theoretically justifiable limiters, as described in this thesis, resulted in large reductions in search error, with improvements in mean accuracy of 30%–70%. Others have reported large improvements for image databases on which the original AAM does not work well, and where modifications to the AAM have been aimed at the requirements of a particular modality. The use of multiple texture descriptors, and their combination through probabilistic methods, means that the AAM can learn the best combination of the descriptors given the data. So, as well as improving search performance on normal face images, the method improves AAM performance on completely different modalities, such as medical X-rays.

The examination of experimental results for statistical significance is not very popular in the computer vision field. However, it is essential when deciding whether a particular theoretical improvement actually reduces error. One of the reasons (or excuses) is that we know that our databases and results lack qualities that are assumed by standard statistical methods, e.g. independence and normality. This thesis introduces a bootstrap method for analysing the statistical significance of any improvements in AAM search error. Since any further improvements to the AAM are expected to result in smaller reductions in search error, the superior statistical power of the proposed bootstrap method over standard statistical methods will be valuable for engineering further improvements in AAM performance.

Whilst the use of limiters to constrain a statistical shape or appearance model is well known, with both box and ellipsoidal limiters having been used in the past, this thesis presents evidence for the first time that the choice has a significant effect on AAM performance. All AAM work in this lab now uses ellipsoidal limiters by default.

### 11.1.3 Combined AAM-SVM Object Detection

This is the first time complicated appearance models like AAMs have been used with statistical classifiers for object detection. A detailed study of the choice of feature vector was undertaken. The residual outperformed the AAM parameters, and the 100-pixel residual was the best choice of resolution on the tasks examined. This means that the AAM-SVM is detecting faces with about half the number of pixels needed by other described methods.

The classification performance of the AAM-SVM detector was shown to improve upon the patch SVM detector at the expense of a considerably lower speed than is achieved by more recent patch-based detection methods. Initial work to bring multi-stage techniques to the AAM-SVM suggests that this difference in speed can be reduced without significant loss of detection accuracy.

## 11.2 Summary of Further Work

Directions for future work have been discussed at several points within this thesis. The most salient of these are drawn together and discussed below.

This thesis introduced the Texture AAM. Further experiments on 3D and multi-modal appearance modelling are in progress to further validate the Texture AAM, and possibly refine the current set of texture descriptors.

The problem of the grossly incorrect estimates of the principal variances of the PDF

of AAM model parameters (section 8.5,) are in obvious need of further investigation. It might be possible to use the decision boundary of the SVM to estimate the best Gaussian fit to the valid face manifold. This would improve the theoretical validity of the AAM and should lead to better performance.

There are several additional fundamental improvements to AAMs that would be valuable to an AAM-based image search engine. Currently it requires many hours of human effort to manually mark up the AAM training images with the label points. Possible methods of acquiring the first guess at the markup include the discriminating eigenpatches method used by Perona *et al.*[166, 58], and the salient points method of Walker *et al.*[163] Further optimisation of the label points, to create a good AAM, may be possible with an appearance-model version of the automatic optimisation of shape models due to Davies *et al.*[46]

In order to acquire enough false positives to train an AAM-SVM to the performance levels required for database retrieval, the system will need to get much faster. The obvious way of doing this, and a sensible target for future work, would be a means of quickly guessing the most likely object locations in a given image. By only having to check a few locations in each image, rather than a few hundred thousand, the system could be fast enough to be potentially useful. Section 10.4 considered one approach using multiple stages to quickly search for potential locations, rejecting implausible hypotheses early, and refining the accuracy of plausible hypotheses. Another intended approach that partially motivated the AAM texture preprocessor work in chapter 6 was to match a sparse non-maximally suppressed version of the multiple-texture descriptor AAM, to interest point detectors run over the current image.

Further work is necessary to show that the AAM-SVM combination works for object classes other than faces. Appearance models have been built and tested on other classes such as cars, hands, eyes, and rats[29]. It is reasonable to expect the AAM-SVM method to work on these. It had originally been hoped to test the AAM-SVM on more object classes, e.g. horses, cars. However, after the poor initial results in

section 10.3 it did not appear worth attempting the work on the small databases that could be easily acquired. More rigorous experimentation should be attempted with a more comprehensive database, or after acquiring evidence to suggest that the AAM-SVM can learn from small databases.

The standard AAM method does not fit very well to partially occluded objects, and so the AAM-SVM could not be expected to perform well in the presence of occlusion. Several methods have been proposed to make appearance model search robust to occlusion. Rogers[116] added structure variables to the AAM alongside the existing shape and appearance parameters. When these structure variables were close to zero, a section of the texture vector was not sampled from the image, but implied by the texture PDF instead. Since the structure variables were correlated with the shape and appearance in the combined model, the AAM learnt when to ignore parts that were self-occluded. Gross *et al.*[63] have shown how to use a robust kernel during AAM search to deal with occlusion. Spatial consistency of the occlusion is enforced by using the same robust kernel scale over each triangle in the shape mesh. With either of these methods, it is possible to mark as missing the residuals for occluded pixels, hopefully allowing the SVM to learn to ignore the occlusion. Alternatively the SVM classifier could be split into sections, and only the sections for non-occluded pixels used to decide the current patch.

Stegmann and Larsen[142] have shown that using colour information improves AAM search. Edwards *et al.*[50] showed better estimation of identity when tracking a face through a video sequence, compared to just using a single frame. It would be straightforward to extend the AAM-SVM to benefit from colour, and with some effort, also from video.

## 11.3 Final Conclusions

The initial question posed in this thesis:

> Can Appearance Modelling be used for image database retrieval, and if so how?

can be answered by stating that with further work to speed up the process, the combined AAM-SVM could be used for image database retrieval.

# Appendix A

# Contributions to VXL

## A.1 Introduction

Almost all of the software written for this thesis, was written in C++ using VXL (Vision-X-Libraries)[28] to provide numeric and image primitives and standard processing functions. As well as using VXL, significant contributions were made to these public libraries, including code, documentation, and maintenance. Two major contributions to the public core VXL libraries are described in this appendix.

Also of note, is VXL's coding environment[28]. Whilst the testing, documentation methods, cross-platform coding and maintenance methods of VXL can be ignored by users, they are nevertheless extremely valuable to adopt. Adding code level comments which are automatically extracted to produce comprehensive API documentation, makes it easier to remember and understand one's old code. Writing software at the same time as, or even after, writing the testing framework, gives a degree of confidence not often warranted in research software. And by using software repositories and automatic test systems, new bugs are caught, and cross-platform issues are exposed, before they can affect experimental results.

## A.2 Background

> "VXL (the Vision-something-Libraries) is a collection of C++ libraries designed for computer vision research and implementation. It was created from TargetJr and the Image Understanding Environment (IUE) with the aim of making a lighter, faster and more consistent system. VXL is written in ANSI/ISO C++ and is designed to be portable over many platforms." [28]

The research group I was involved within ISBE decided to adopt VXL soon after I started working here. We contributed two large libraries to the core of VXL as part of this adoption. I was joint designer and implementer of both of these contributions, as well as being responsible for their successful proposal to the VXL consortium. Most design decisions were taken in consultation with my co-designer Dr. Tim Cootes, and often with the VXL consortium. However the design decisions described here were primarily my responsibility.

## A.3 `vsl`—VXL's Binary IO Library

The one missing piece of functionality that prevented ISBE from using being able to use VXL initially, was fast binary Input/Output (IO). ISBE's existing libraries expected to be able to read and write themselves quickly to and from a file stream. They also expected to be able to read and write all C++ and base library primitives (e.g. numbers, numeric vectors, geometric points, and arrays.) The IO file format in ISBE's existing libraries was supported both Intel and Sun 32-bit platforms.

There were several constraints imposed by the VXL consortium that prevented a straight port of ISBE's existing IO code. The most interesting three are:

- The various libraries of primitives (e.g. `vnl` which includes numeric vectors, and

`vgl` which includes geometric points) could not share any common code beyond what was provided by the standard C++ library. The modular design of VXL also forbade the inclusion of the IO code for numeric vector in the same file or even the same library as the rest of the code for the vector.

- There was a need to support a wider range of platforms including ones with 64 bit words.
- In ISBE's original libraries ownership of data on the heap was made explicit—each heap object always had exactly one other object responsible for its deletion, and IO. VXL used a lot of shared-ownership smart pointers, and this causes problems during IO.

### A.3.1 Dependency Issues and Errors

In our experience, IO is prone to subtle errors, and separating the IO from the rest of the code for a class is especially problematic. (This is mostly due to one source file being modified without the other.) However there was no other way to meet VXL's dependecy requirements. Instead, errors have been avoided by adopting VXL's self-testing mechanisms, and rigourously testing each class's IO behaviour. The IO system was split into a library that could deal with C++ standard types and classes, and an add-on IO library for each core VXL library.

### A.3.2 IO for Shared Pointers.

Shared pointers are objects that behave like ordinary pointers or references. Several of them can point to the same object, and they are said to *share ownership*. This means that you cannot make one pointer responsible for the objects deletion or IO. The problem of deleting a shared object too early, is dealt with using reference counting.

With IO, the problem is more subtle. If two objects A and B each have a shared

pointer to the same object C, then normal IO will not work correctly. During the output, both A and B will be saved along with two copies of C (one each for A and B) in the file. There will therefore be two copies of C when the file is loaded back into memory. Since we are using IO to save the state of a data structure, this is not satisfactory.

The standard solution to this problem is called *serialisation*[168]. This gives each object a serial number when it is written to disk. If the object already has a serial number, it is assumed to have already been output and instead a record of that serial number is written to disk. During input a record of each object's location and serial number are stored, and whenever a record is found indicating that something has been stored twice, the loaded value is set to point to the existing object.

I looked at several implementations: Microsoft Foundation Classes[13], Qt[151], the Image Understand Environment DEX[72] , and Java[144]. The first two implementations suffered from needing to use macros in the class declaration to add a variable to the class which would record its serial number. Java also has special requirements of the class definition to perform serialisation. This general approach could not be used by VXL, which needed an entirely external record connecting serial numbers to objects. These implementations also suffer from requiring serial numbers for every object undergoing IO whether they are shared or not—a significant space overhead. The IUE's DEX protocol used the explicit instantiation of external helper objects by the programmer, with one helper object per object to be serialised. This required a large coding overhead for the programmer.

Discarding these previous approaches, a better solution was to use a mapping from an object's memory address to serial number. The extra space overhead of map management would be exceeded by wasted space of the implementations above, for most applications. Whilst not every object has a unique memory address (if A is the first member of B both will have identical addresses) every object that might need a serial number does have a unique address. The map is filled with entries

each time an object that needs a serial number is written to disk. During input, the map is from serial number to memory address. By storing a unique map with each stream, this mechanism requires no coding overhead for programmers, and also allows simultaneous IO to multiple streams without any threading conflict.

### A.3.3 Cross Platform Binary IO

To be acceptable to the VXL consortium is was necessary to use an IO format that would be compatible on the widest possible range of platforms that might be used in computer vision research. One possible solution was to use text rather than binary IO. However, testing showed this to be many times slower than our existing binary IO solution.

Almost all platforms use IEEE standard 754 floating point representations, so byte swapping to a fixed endian format works for these. The most interesting problem is in saving integers. Since it is not possible to take a integer variable over $2^{32}$ saved from a 64 bit platform, and load it in meaningfully on a 32-bit platform, a useful set of constraints are as follows:

1. A platform must be able to save and load correctly every integer it is capable of representing.

2. A platform must be capable of loading any number that it can represent, even if that number was saved by a platform with a larger machine word.

Two solutions are

1. Record the size of the various types in the file header during saving, and then output in native format. This has the advantage of being very quick.

2. Save integers in a single fixed format that can cope with any word length. This has the advantage of having the number of possible file/machine configura-

tions increase linearly with the number of platforms as opposed to a quadratic increase as required by the first solution.

I implemented an instance of the second solution. Each integer was saved as an arbitrary length set of bytes. The first seven bits represented the next most significant bits of the number. The eighth bit was used to mark the last byte. Timing experiments showed that the conversion times were much smaller than text conversion or the disk IO time.

## A.4 `vil2`—A Replacement Image Library

VXL's original imaging library `vil1`, was not capable of providing many of ISBE's requirements. We needed support for arithmetic indexing, planes, an easy-to-use default image class, and compatibility with 3d. A VXL-compatible Manchester Image Library was written which had these features, and used `vil1` for image loading and saving. Over the next 18 months it became clear that we were duplicating large amounts of image processing code, and did not have access to some of the more speculative work published by other members of the VXL consortium. We decided to try and merge the two imaging libraries to everyone's satisfaction.

`vil1` had (and its users required) good support for very large images. It did this by having a single image class hierarchy for users. At the base was a general representation of an image somewhere (on disk or in memory.) More specialised classes provided abilities to read and write particular image file formats, or in the case of in-memory images, manipulate the pixels. Unfortunately, the forcing of in-memory images into the same tree without significant loss of efficiency, resulted in a great deal of confusion for users, without making it that easy to manipulate very large images on disk. The only common access to pixel values across all image types was by getting and putting *sections*—packed buffers of raw image data.

The rewrite solved this complexity by separating the generalised-image-somewhere concept from the in-memory image. The in-memory image concept was copied from the Manchester imaging library after extensive simplification.

I found through several code writing experiments that the generalised image concept could be made similar to the existing generalised image type in `vnl`, only much easier to use, if the common API allowed users to get and put an in-memory image object. This separates the design of algorithms which run efficiently over large images into an efficient in-memory implementations followed by the stitching of sections of in-memory results together.

## A.5 Discussion

Whilst the work in this appendix was not research, it is published and used by the VXL community. It would be very difficult to tell if the designs are optimal in any useful sense. However, it is worth noting that a section on IO added to the C++ FAQ[27] made very similar recommendations to, but two years later than, the above design of VXL's IO library.

# Appendix B

# Database Descriptions

**Figure B.1:** Examples from Messer *et al.*'s[93] `XM2VTS` database. There are 1816 images of 720×576 pixels. They were labelled by ISBE with 68 points.

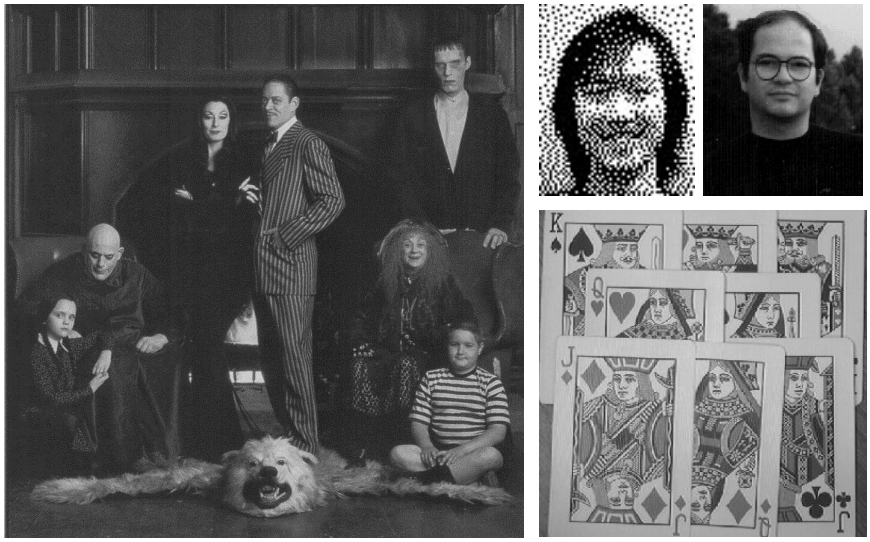


**Figure B.2:** Examples from Rowley *et al.*'s[122, 145] `CMU` database—the heads on the playing cards are included, the bear is not. There are 507 faces in 117 images of varying resolutions. The non-upright part of the published database was ignored. They are published at `http://vasc.ri.cmu.edu//idb/html/face/frontal_images/index.html` with rough 6 point markup, which was extended to 22 points by this author.

**Figure B.3:** Examples from Jesorsky *et al.*'s[73] `BioID` database, available from `http://www.humanscan.de/support/downloads/facedb.php`. There are 1518 images of 384×286 pixels. They are published with 20-point markup, which was extended to 22 points by this author.

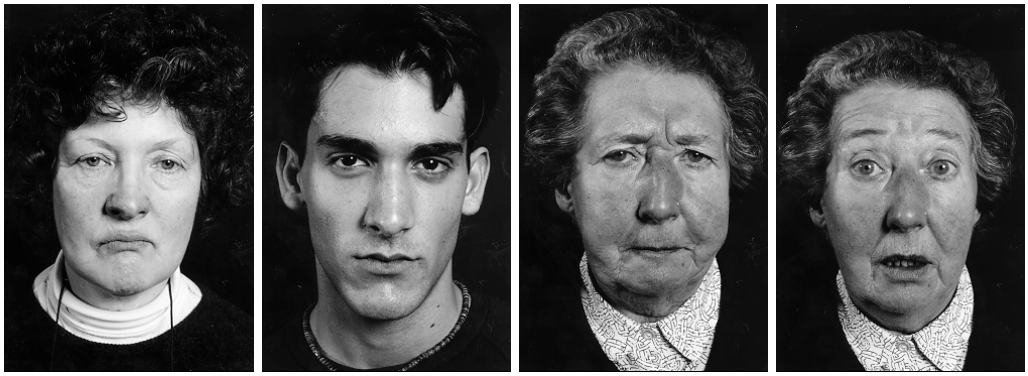


**Figure B.4:** Examples from ISBE's `expression` database. There are 399 images of 247×365 pixels, and they are labelled with 68 points.

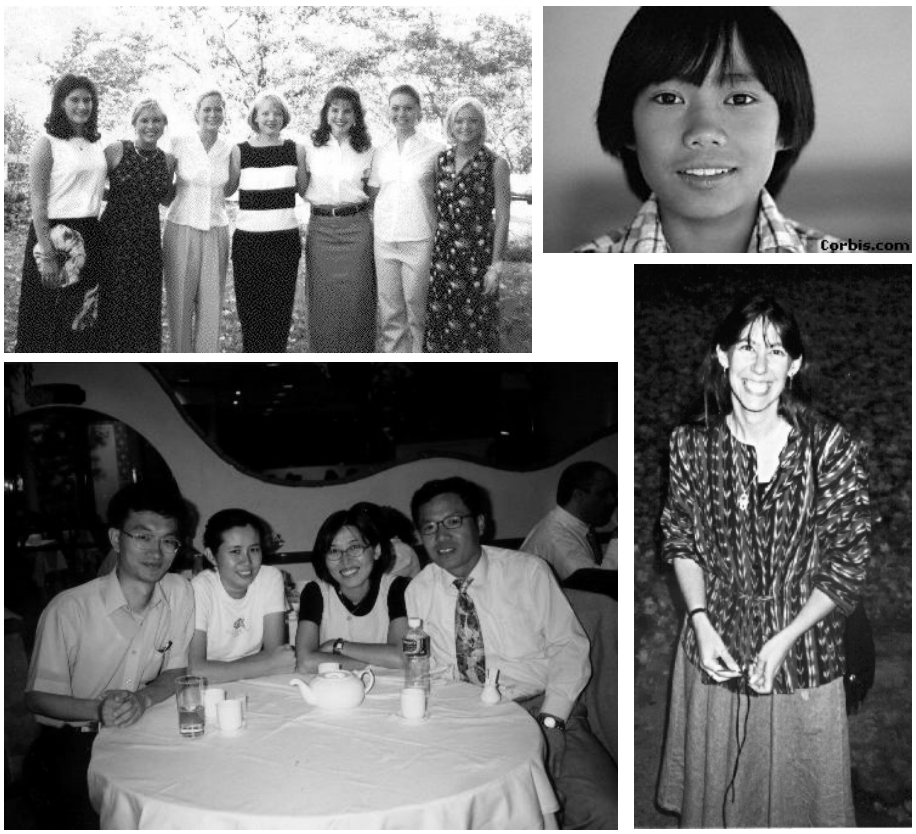



**Figure B.5:** Examples from the author's `webimagesB` database. There are 144 faces in 43 images of varying resolutions. The two eye centres of each face are labelled.

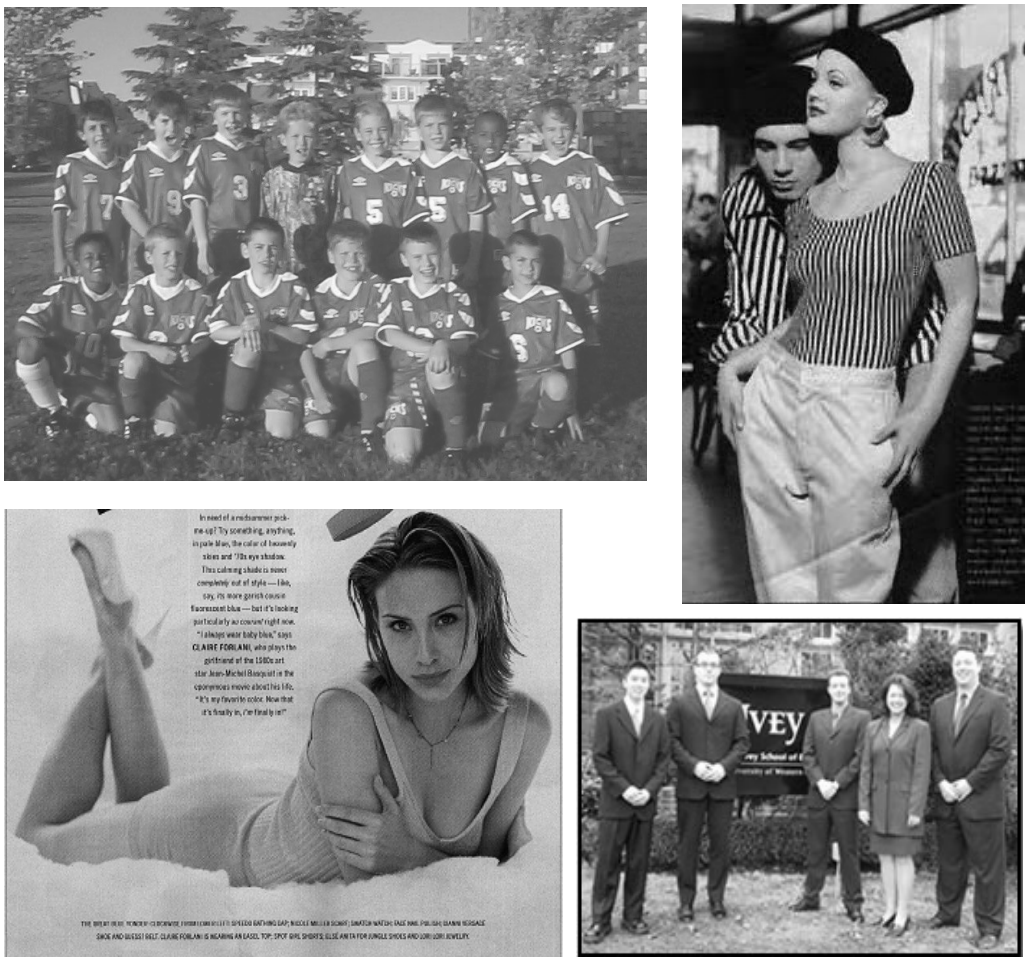


**Figure B.6:** Examples from the author's `webimagesC` database. There are 991 faces in 623 images, with the faces varying in resolution from 4.5 to 373 pixels interocular separation. The two eye centres of each face are labelled. Faces where the two eyes cannot be seen are ignored.

**Figure B.7:** Examples from some of ISBE's additional databases which were used in the `all22r` database. The images are labelled with at least 68 points.

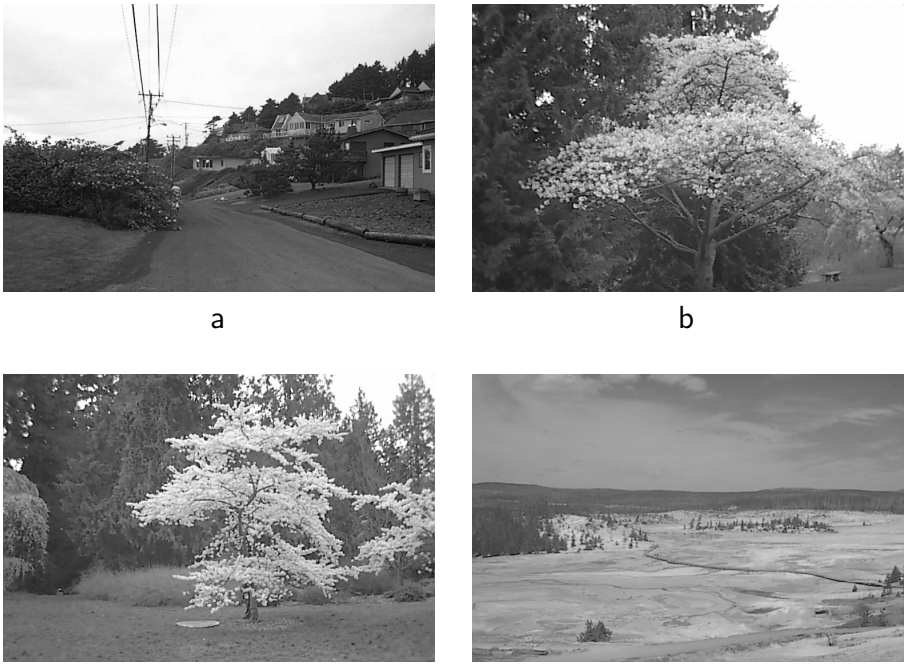



**Figure B.8:** The first four images from Shapiro's [137] `UWash` database. It consists of 282 images containing no faces, downloaded from `http://www.cs.washington.edu/research/imagedatabase/groundtruth/`. The images are all 756×504 pixels.

# Bibliography


[1] Y. Adini, Y. Moses, and S. Ullman. Face recognition: The problem of compensating for changes in illumination direction. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 19(7):721–32, 1997.

[2] A. A. Alatan, L. Onural, M. Wollborn, R. Mech, E. Tuncel, and T. Sikora. Image sequence analysis for emerging interactive multimedia services—the European COST 211 framework. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(7):802–13, 1998.

[3] C. F. Alaya, B. Cramariuc, C. Reynaud, Q. Meng, A. B. Dragos, B. Hnich, M. Gabbouj, P. Kerminen, T. Makinen, and H. Jaakkola. MUVIS: A system for content-based indexing and retrieval in large image databases. In *Storage and Retrieval for Image and Video Databases*, volume 3656, pages 98–106. Proceedings SPIE, 1999.

[4] AltaVista. Image search. `http://www.altavista.com/cgi-bin/query?pg=q&stype=simage`, 1999.

[5] AltaVista. WWW search engine. `http://www.altavista.com`, 1999.

[6] J. R. Bach, C. Fuller, A. Gupta, A. Hampapur, B. Horowitz, R. Humphrey, R. Jain, and C.-F. Shu. The Virage image search engine: An open framework for image management. *Proceedings-of-the-SPIE*, 2670:76–87, 1996.

[7] E. Bailly-Bailliére, S. Bengio, F. Bimbot, M. Hamouz, J. Kittler, J. Mariéthoz, J. Matas, K. Messer, V. Popovici, F. Poirée, B. Ruiz, and J. Thiran. The BANCA database and evaluation protocol. In *Audio- and Video-based Biometric Person Authentification*, pages 625–38, 2003.

[8] R. Bajcsy and S. Kovacic. Multiresolution elastic matching. *Computer Vision, Graphics, and Image Processing*, 46:1–21, 1989.

[9] S. Baker, I. Matthews, and J. Schneider. Automatic construction of active appearance models as an image coding problem. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 26(10), 2004.

[10] R. Basri. Viewer-centred representations in object recognition: A computational approach. In C. H. Chen, L. F. Pau, and P. S. P. Wang, editors, *Handbook of pattern recognition and computer vision*, pages 863–82. World Scientific Publishing Company, 1991.

[11] E. M. L. Beale. *Introduction to optimisation*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley and Sons, 1988.

[12] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman. Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 18(7):711–20, 1997.

[13] J. Beveridge. Inside MFC serialization. *Dr. Dobb's Journal*, 20(10):62, 4, 6–7, 122–3, 1995.

[14] C. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995.

[15] A. Blake and M. Isard. *Active contours*. Springer, 1998.

[16] B. Blanz, S. Romdhani, and T. Vetter. Face identification across different poses and illuminations with a 3D morphable model. In *International Conference on Automatic Face and Gesture Recognition*, 2002.

[17] M. Bober. MPEG-7 visual shape descriptors. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):716–9, 2001.

[18] H. G. Bosch, S.C.Mitchell, P.F.Boudewijn, P.F.Leieveldt, F.Nijland, O. Kamp, and M. Sonka. Active appearance-motion models for endocardial contour detection in time sequences of echocardiograms. In *SPIE Medical Imaging*, pages 257–68, 2001.

[19] N. Brown. Satellite transmission snafu leads to diplomatic incident. In P. G. Neumann, editor, *The risks digest*, volume 19.26. ACM Committee on Computers and Public Policy, (`http://catless.ncl.ac.uk/Risks/19.26.html#subj1l`), 1997.

[20] C. Burges. Simplified support vector decision rules. In *13th Intl. Conf. on Machine Learning*, pages 71–7, 1996.

[21] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Journal of Data Mining and Knowledge Discovery*, 2(2):121–67, 1998.

[22] M. C. Burl, M. Weber, and P. Perona. A probabilistic approach to object recognition using local photometry and global geometry. In *European Conference on Computer Vision*, volume 2, pages 628–41, 1998.

[23] G. Celeux and G.Govaert. A classification EM algorithm for clustering and two stochastic versions. *Computational Statistics and Data Analysis*, 14:315–32, 1992.

[24] O. Chapelle, P. Haffner, and V. N. Vapnik. Support vector machines for histogram based image classification. *IEEE Transactions on Neural Networks*, 10(5):1055–64, 1999.

[25] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1):131–59, 2000.

[26] G. C. Charters. *Segmentation and classification in automated chromosome analysis using trainable models*. PhD thesis, University of Manchester, 1994.

[27] M. Cline. The C++ FAQ lite. `http://www.parashift.com/c++-faq-lite`, 2003.

[28] VXL Consortium. VXL homepage. `http://vxl.sourceforge.net`, 2001.

[29] T. F. Cootes. Personal communication, 1999.

[30] T. F. Cootes. Statistical models of appearance for computer vision. Technical Report `www.isbe.man.ac.uk/%7ebim/Models/app_models.pdf`, ISBE, University of Manchester, 2004.

[31] T. F. Cootes, D. Cooper, C. J. Taylor, and J. Graham. Training models of shape from sets of examples. *British Machine Vision Conference*, pages 9–18, 1992.

[32] T. F. Cootes, G. J. Edwards, and C.J.Taylor. Active appearance models. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 23(6):681–885, 2001.

[33] T. F. Cootes, K. H., and P. V. EU project U-face final report. Technical report, ISBE, University of Manchester, 2003.

[34] T. F. Cootes, A. Hill, C. J. Taylor, and J. Haslam. The use of active shape models for locating structures in medical images. *Image and Vision Computing*, 12(6):355–66, 1994.

[35] T. F. Cootes, G. Page, C. Jackson, and C. J. Taylor. Statistical grey-level models for object location and identification. *Image and Vision Computing*, 14(8):533–40, 1996.

[36] T. F. Cootes and C. J. Taylor. Active shape models—'smart snakes'. In *British Machine Vision Conference*, pages 266–75, 1992.

[37] T. F. Cootes and C. J. Taylor. Locating faces using statistical feature detectors. In *International Conference on Automatic Face and Gesture Recognition*, pages 204–9, Killington, VT, USA., 1996. IEEE Computer Society Press.

[38] T. F. Cootes and C. J. Taylor. A mixture model for representing shape variation. *Image and Vision Computing*, 17(8):567–73, 1999.

[39] T. F. Cootes and C. J. Taylor. Combining elastic and statistical models of appearance variation. In *European Conference on Computer Vision*, volume 1, pages 149–63, 2000.

[40] T. F. Cootes and C. J. Taylor. Constrained active appearance models. In *International Conference on Computer Vision*, volume 1, pages 748–54, 2001.

[41] T. F. Cootes and C. J. Taylor. On representing edge structure for model matching. In *CVPR*, volume 1, pages 1114–9, 2001.

[42] T. F. Cootes, K. N. Walker, and C. J. Taylor. View-based active appearance models. In *International Conference on Automatic Face and Gesture Recognition*, pages 271–6, 2000.

[43] Corbis. Image library. `http://www.corbis.com`, 2000.

[44] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines, and other kernel-based learning methods.* Cambridge University Press, 2000.

[45] D. Cristinacce, T. F. Cootes, and I. M. Scott. A multi-stage approach to facial feature detection. In *British Machine Vision Conference*, page to appear, 2004.

[46] R. H. Davies, C. J. Twining, T. F. Cootes, J. C. Waterton, and C. J. Taylor. A minimum description length approach to statistical shape modeling. *IEEE Transactions on Medical Imaging*, 21(5):525–37, 2002.

[47] A. C. Davison and D. V. Hinkley. *Bootstrap methods and their application.* Cambridge University Press, 1997.

[48] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, B*, 39(1):1–38, 1977.

[49] J. P. Eakins, M. E. Graham, and J. M. Boardman. Trademark image retrieval by shape similarity. *IEEE Multimedia*, 5(2):53–63, 1998.

[50] G. Edwards, T. F. Cootes, and C. J. Taylor. Face recognition using active appearance models. In *European Conference on Computer Vision*, pages 581–95. Springer-Verlag, Berlin, 1998.

[51] G. Edwards, T. F. Cootes, and C. J. Taylor. Advances in active appearance models. In *International Conference on Computer Vision*, pages 137–42, 1999.

[52] G. Edwards, C. J. Taylor, and T. F. Cootes. Interpreting face images using active appearance models. In *International Conference on Automatic Face and Gesture Recognition*, pages 300–5, 1998.

[53] Efron and Tibshirani. *An introduction to the bootstrap.* Monographs on statistics and applied probability. CRC, Boca Raton, 1998.

[54] B. Efron. Bootstrap methods: Another look at the jackknife. *Ann. of Statist.*, 7:1–26, 1979.

[55] P. G. B. Enser. Query analysis in a visual information context. *Journal Document and Text Management*, 1(1):25–52, 1993.

[56] J. J. Faraway. Practical regression and ANOVA using R. `http://www.stat.lsa.umich.edu/~faraway/book/`, July 2002 2002.

[57] L. Fei-Fei, R. Fergus, and P. Perona. A Bayesian approach to unsupervised one-shot learning of object categories. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1134–41, 2003.

[58] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–71, 2003.

[59] G. B. Finlayson, B. Schiele, and J. L. Crowley. Comprehensive colour image normalization. In *European Conference on Computer Vision*, pages 475–90, 1998.

[60] D. M. Gavrila and V. Philomin. Real-time object detection for "smart" vehicles. In *International Conference on Computer Vision*, pages 87–93, 1999.

[61] X. Ge. C++ code: SMO training of SVM. `http://www.datalab.uci.edu/people/xge/svm/index.html`, 2001.

[62] S. Gong. Personal communication, 2001.

[63] R. Gross, I. Matthews, and S. Baker. Constructing and fitting active appearance models with occlusion. In *IEEE Workshop on Face Processing in Video*, 2004.

[64] V. N. Gudivada and V. V. Raghavan. Modeling and retrieving images by content. *Information Processing and Management*, 33(4):427–52, 1997.

[65] S. Gunn. Support vector machines for classification and regression. Manual, ISIS, University of Southampton, 1998.

[66] P. Hall, D. Marshall, and R. Martin. Adding and subtracting eigenspaces. In *British Machine Vision Conference*, volume 2, pages 453–62, 1999.

[67] M. Hamouz, J. Kittler, J. K. Kamarainen, and H. Kälviäinen. Hypothesis-driven affine invariant localisation of faces in verification systems. In *Audio- and Video-based Biometric Person Authentification*, pages 274–84, 2003.

[68] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, pages 147–51, 1988.

[69] T. Heap and D. Hogg. Improving specificity in PDMs using a hierarchical approach. In *British Machine Vision Conference*, pages 80–9, 1997.

[70] D. Hond and L. Spacek. Distinctive descriptions for face processing. In *British Machine Vision Conference*, Colchester, 1997.

[71] D. Huber. Computer vision home page. `http://www.cs.cmu.edu/afs/cs/project/cil/ftp/html/vision.html`, 1999.

[72] Amerinex Applied Imaging. Image understand environment. `http://www.aai.com/AAI/IUE/IUE.html`, 1999.

[73] O. Jesorsky, K. Kirchberg, and R. Frischholz. Robust face detection using the hausdorff distance. In *Audio- and Video-based Biometric Person Authentification*, pages 90–5, 2001.

[74] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in kernel methods—support vector learning*. MIT Press, 1998.

[75] T. Joachims. Estimating the generalization performance of an SVM efficiently. In *International Conference on Machine Learning*, pages 361–70, 1999.

[76] M. J. Jones and T. Poggio. Multidimensional morphable models. In *International Conference on Computer Vision*, pages 683–8, 1998.

[77] G. W. Kerr. A review of BTs interactive TV trials. In *IEE Colloquium on Interactive Television*, London, 1996.

[78] N. Kingsbury. Image processing with complex wavelets. *Philosophical Transactions of the Royal Society, A*, 357(1760):2543–60, 1999.

[79] J. Kittler, Y. P. Li, and J. Matas. On matching scores for LDA-based face verification. In *British Machine Vision Conference*, volume 1, pages 42–51. BMVA Press, 2000.

[80] D. Koubaroulis, J. Matas, and J. Kittler. Colour-based object recognition for video annotation. In *International Conference on Pattern Recognition*, volume 2, pages 1069–72, 2002.

[81] M. Lades, J. C. Vorbruggen, J. Buhmann, J. Lange, C. von der Malsburg, R. P. Wurtz, and W. Konen. Distortion invariant object recognition in the dynamic link architecture. *IEEE Transactions On Computers*, 42(3):300–11, 1993.

[82] A. Lanitis, C. J. Taylor, and T. F. Cootes. A unified approach to coding and interpreting face images. In *International Conference on Computer Vision*, pages 368–73, 1995.

[83] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–91, 1999.

[84] M. Li and P. Vitanyi. *An introduction to Kolmogorov complexity and its applications*. Springer, 1997.

[85] Y. Lin, Y. Lee, and G. Wahba. Support vector machines for classification in nonstandard situations. Technical Report 1016, Dept. of Statistics, University of Wisconsin, 2000.

[86] D. G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, pages 1150–7, 1999.

[87] J. Manslow. MLP builder software. `http://www.isis.soton.ac.uk/~jfm96r`, 2000.

[88] D. Marr. *Vision. A computational investigation into the human representation and processing of visual information*. W.H. Freeman & Co., 1982.

[89] J. M. Martnez. MPEG-7 overview. Standard ISO/IEC JTC1/SC29/WG11 N5525, International Organisation For Standardisation, 2003.

[90] MATE. Media access technologies. `http://www.mate.co.il`, 1999.

[91] S. J. McKenna, S. Gong, R. P. Wrtz, J. Tanner, and D. Banin. Tracking facial feature points with gabor wavelets and shape models. In *International Conference on Audio-Video Based Biometric Person Authentication*, pages 35–43, 1997.

[92] B. W. Mel and J. Fiser. Minimizing binding errors using learned conjunctive features. *Neural Computation*, 12:247–78, 2000.

[93] K. Messer, J. Matas, J. Kittler, J. Luettin, and G. Maitre. XM2VTSdb: The extended M2VTS database. In *2nd Conf. on Audio and Video-based Biometric Personal Verification*. Springer Verlag, 1999.

[94] T. P. Minka and R. W. Picard. Interactive learning with a "Society of models". *Pattern Recognition*, 30(4):565–81, 1997.

[95] S. C. Mitchell, P. F. Boudewijn, P. F. Lelievedt, R. J. v.d.Geest, H. G. Bosch, J. H. Reiber, and M. Sonka. Time continuous segmentation of cardiac MR image sequences using active appearance motion models. In *SPIE Medical Imaging*, 2001.

[96] B. Moghaddam. Principal manifolds and Bayesian subspaces for visual recognition. In *International Conference on Computer Vision*, pages 1131–6, 1999.

[97] B. Moghaddam and A. Pentland. Face recognition using view-based and modular eigenspaces. *Proceedings of the SPIE*, 2277:12–21, 1994.

[98] B. Moghaddam and A. Pentland. A subspace method for maximum likelihood target detection. In *International Conference on Image Processing*, pages 512–15, 1995.

[99] B. Moghaddam and A. Pentland. Probabilistic visual learning for object representation. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 19(7):696–710, 1997.

[100] H. Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical Report CMU-RI-TR-80-03, Robotics Inst., CMU, 1980.

[101] G. Mori, S. Belongie, and J. Malik. Shape contexts enable efficient retrieval of similar shapes. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 723–30, 2001.

[102] H. Murase and S. K. Nayar. Visual learning and recognition of 3-D objects from appearance. *International Journal of Computer Vision*, 14(1):524, 1995.

[103] R. C. Nelson and A. Selinger. Large-scale tests of a keyed, appearance-based 3-D object recognition system. *Vision Research*, 38(15):2469–88, 1998.

[104] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin. The QBIC project: Querying images by content using color, texture, and shape. In *Storage and Retrieval for Image and Video Databases. Feb. 1993.*, volume 1908, pages 173–87, San Jose, CA, USA., 1993. The International Society for Optical Engineering.

[105] S. Obdržálek and J. Matas. Local affine frames for image retrieval. In *International Conference on Image and Video Retrieval*, pages 318–27, 2002.

[106] K. Ohba and K. Ikeuchi. Detectability, uniqueness, and reliability of eigen windows for stable verification of partially occluded objects. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 19(9):1043–8, 1997.

[107] National Osteoporosis Foundation Working Group on Vertebral Fractures. Assessing vertebral fractures. *Journal of Bone and Mineral Research*, 10(4):518–23, 1995.

[108] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 130–6, 1997.

[109] P. J. Phillips, H. Wechsler, J. Huang, and P. J. Rauss. The FERET database and evaluation procedure for face-recognition. *Image and Vision Computing*, 16(5):295–306, 1998.

[110] J. C. Platt. Fast training of support vector machines using sequential minimal optimisation. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in kernel methods—support vector learning*, pages 185–208. MIT Press, 1998.

[111] J. C. Platt. Using sparseness and analytic QP to speed training of support vector machines. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems*. MIT Press, 1998.

[112] The R Project. R. `http://www.r-project.org/`, 2003.

[113] T. D. Rikert, M. J. Jones, and P. Viola. A cluster-based statistical model for object detection. In *International Conference on Computer Vision*, pages 1046–53, 1999.

[114] B. D. Ripley. *Pattern recognition and neural networks*. Cambridge University Press, 1996.

[115] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–71, 1978.

[116] M. Rogers. *Exploiting weak constraints on object structure and appearance for segmentation of 2-D images*. PhD thesis, University of Manchester, 2001.

[117] K. Rohr. On 3D differential operators for detecting point landmarks. *Image and Vision Computing*, 15:219–33, 1997.

[118] S. Romdhani, B. Blanz, and T. Vetter. Face identification by fitting a 3D morphable model using linear shape and texture error functions. In *European Conference on Computer Vision*, volume 4, pages 3–19, 2002.

[119] S. Romdhani, S. Gong, and A. Psarrou. A multi-view nonlinear active shape model using kernel PCA. In *British Machine Vision Conference*, 1999.

[120] S. Romdhani, A. Psarrou, and S. Gong. On utilising template and feature-based correspondence in multi-view appearance models. In *European Conference on Computer Vision*, volume 1, pages 299–813, 2000.

[121] S. Romdhani, P. Torr, B. Scholkopf, and A. Blake. Computationally efficient face detection. In *International Conference on Computer Vision*, volume 2, pages 695–700, Vancouver, 2001.

[122] H. A. Rowley, S. Baluja, and T. Kanade. Rotation invariant neural network-based face detection. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 20(1):23–38, 1998.

[123] Y. Rubner and C. Tomasi. Texture-based image retrieval without segmentation. In *International Conference on Computer Vision*, pages 1018–24, 1999.

[124] Y. Rui, T. S. Huang, and S. F. Chang. Image retrieval: Current techniques, promising directions and open issues. *Journal of Visual Communication and Image Representation*, 10:39–62, 1999.

[125] E. Sali and S. Ullman. Recognizing novel 3-D objects under new illumination and viewing position using a small number of example views or even a single view. In *International Conference on Computer Vision*, pages 153–61. IEEE, 1998.

[126] E. Sali and S. Ullman. Combining class specific fragments for object classification. *British Machine Vision Conference*, pages 203–13, 1999.

[127] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 18(5):401–9, 1969.

[128] S. Santini and R. Jain. Similarity queries in image databases. In *IEEIEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 646–51, 1996.

[129] C. Saunders, M. O. Stitson, J. Weston, L. Bottou, B. Schölkopf, and A. Smola. Support vector machine—reference manual. Technical Report CSD-TR-98-03, Computer Science, Royal Holloway, University of London, 1998.

[130] R. E. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[131] B. Schiele and J. L. Crowley. Object recognition using multidimensional receptive field histograms. In *European Conference on Computer Vision*, page 6109, 1996.

[132] B. Schiele and A. Pentland. Probabilistic object recognition and localization. In *International Conference on Computer Vision*, pages 177–82, 1999.

[133] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In *Int. Conf. Artificial Neural Networks*, pages 47–52. Springer, 1996.

[134] S. Sclaroff and J. Isidoro. Active blobs. In *International Conference on Computer Vision*, pages 1146–53, 1998.

[135] I. M. Scott, T. F. Cootes, and C. J. Taylor. Improving appearance model matching using local image structure. In *Information Processing in Medical Imaging*, pages 258–69, 2003.

[136] I. M. Scott, T. F. Cootes, and C. J. Taylor. Improving appearance model matching using local structure. In *Medical Image Understanding and Analysis*, pages 5–9, 2003.

[137] L. Shapiro. University of Washington image database. `http://www.cs.washington.edu/research/imagedatabase/groundtruth/`.

[138] L. Sirovich and M. Kirby. Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America A*, 4(3):519–24, 1987.

[139] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 22(12):1349–80, 2000.

[140] J. R. Smith and S.-F. Chang. Visually searching the web for content. *IEEE Multimedia*, 4(3):12–20, 1997.

[141] P. P. Smyth, C. J. Taylor, and J. E. Adams. Vertebral shape: Automatic measurement with active shape models. *Radiology*, 211:571–8, 1999.

[142] M. B. Stegmann and R. Larsen. Multi-band modelling of appearance. *Image and Vision Computing*, 21(1):61–7, 2003.

[143] M. Styner, J. A. Lieberman, and G. Gerig. Boundary and medial shape analysis of the hippocampus in schizophrenia. *Medical Image Analysis*, page To appear, 2004.

[144] Sun. Java documentation. `http://java.sun.com/`.

[145] K. K. Sung and T. Poggio. Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 20(1):39–51, 1998.

[146] M. J. Swain and D. H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[147] B. M. t. H. Romeny, L. M. J. Florak, A. H. Salden, and M. A. Viergever. Higher order differential structure of images. In *Information Processsing in Medical Imaging*, LNCS, pages 77–93, 1993.

[148] N. Thacker, P. Riocreux, and R. Yates. Assessing the completeness properties of pairwise geometric histograms. *Image and Vision Computing*, 13(5):423–9, 1995.

[149] K. Tieu and P. Viola. Boosting image retrieval. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 228–35, 2000.

[150] A. B. Torralba and A. Oliva. Semantic organization of scenes using discriminant structural templates. In *International Conference on Computer Vision*, pages 1068–75, 1999.

[151] Trolltech. QT documentation. `http://doc.trolltech.com/2.3/index.html`, 2001.

[152] E. Trucco and A. Verri. *Introductory techniques for 3-D computer vision.* Prentice-Hall, 1998.

[153] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[154] M. Turk and A. Pentland. Face recognition using eigenfaces. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–91, 1991.

[155] A. Vailaya, M. A. T. Figueiredo, A. K. Jain, and H.-J. Zhang. Image classification for content-based indexing. *IEEE Transactions on Image Processing*, 10(1):117–30, 2001.

[156] V. Vapnik. *The nature of statistical learning theory.* Springer-Verlag, New York, 1995.

[157] V. Vapnik and O. Chapelle. Bounds on error expectation for support vector machines. *Neural Computation*, 12(9), 2000.

[158] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas. SOM toolbox for Matlab 5. Manual A57, Helsinki University of Technology, Neural Networks Research Centre, 2000.

[159] T. Vetter, M. J. Jones, and T. Poggio. A bootstrapping algorithm for learning linear models of object classes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 40–6, 1997.

[160] P. Viola. Complex feature recognition: A Bayesian approach for learning to recognize objects. Technical report, AI Lab., MIT, 1996.

[161] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 511–18, 2001.

[162] E. Wachsmuth, M. W. Oram, and D. I. Perrett. Recognition of objects and their component parts—responses of single units in the temporal cortex of the macaque. *Cerebral Cortex*, 4(5):509–22, 1994.

[163] K. Walker, T. F. Cootes, and C. J. Taylor. Locating salient object features. In *British Machine Vision Conference*, pages 463–72, 1998.

[164] R. Walker, P. Foster, and S. Banthorpe. Content production and delivery for interactive multimedia services—a new approach. *BT Technology Journal*, 15(2):74–82, 1997.

[165] M. Weber, W. Einhuser, M. Welling, and P. Perona. Viewpoint-invariant learning and detection of human heads. In *International Conference on Automatic Face and Gesture Recognition*, pages 20–7. IEEE Computer Soc., 2000.

[166] M. Weber, M. Welling, and P. Perona. Towards automatic discovery of object categories. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 101–8, 2000.

[167] M. Weber, M. Welling, and P. Perona. Unsupervised learning of models for recognition. In *European Conference on Computer Vision*, pages 18–32, 2000.

[168] D. Wiebe. A distributed repository for immutable persistent objects. *SIGPLAN Notices*, 21(11):453–65, 1986.

[169] L. Wiskott, J. M. Fellous, N. Kruger, and C. Malsburg. Face recognition by elastic bunch graph matching. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 19(7):775–9, 1997.