MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY
and
CENTER FOR BIOLOGICAL AND COMPUTATIONAL LEARNING
DEPARTMENT OF BRAIN AND COGNITIVE SCIENCES

A.I.T.R. No. 1572 January, 1996

Learning and Example Selection for Object and Pattern Detection

Kah-Kay Sung

This publication can be retrieved by anonymous ftp to publications.ai.mit.edu.

Abstract

This thesis presents a learning based approach for detecting classes of objects and patterns with variable image appearance but highly predictable image boundaries. It consists of two parts. In part one, we introduce our object and pattern detection approach using a concrete human face detection example. The approach first builds a distribution-based model of the target pattern class in an appropriate feature space to describe the target's variable image appearance. It then learns from examples a similarity measure for matching new patterns against the distribution-based target model. The approach makes few assumptions about the target pattern class and should therefore be fairly general, as long as the target class has predictable image boundaries.
Because our object and pattern detection approach is very much learning-based, how well a system eventually performs depends heavily on the quality of training examples it receives. The second part of this thesis looks at how one can select high quality examples for function approximation learning tasks. We propose an *active learning* formulation for function approximation, and show for three specific approximation function classes, that the active example selection strategy learns its target with fewer data samples than random sampling. We then simplify the original active learning formulation, and show how it leads to a tractable example selection paradigm, suitable for use in many object and pattern detection problems.

Copyright © Massachusetts Institute of Technology, 1995

This report describes research done at the Artificial Intelligence Laboratory and within the Center for Biological and Computational Learning. This research is sponsored by grants from the Office of Naval Research under contracts N00014-91-J-1270 and N00014-92-J-1879; by a grant from the National Science Foundation under contract ASC-9217041. Support for the A.I. Laboratory's artificial intelligence research is provided by ONR contract N00014-91-J-4038.

Learning and Example Selection for Object and Pattern Detection

by

Kah Kay Sung

Submitted to the Department of Electrical Engineering and Computer Science
on December 15, 1995, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

Object and pattern detection is a classical computer vision problem with many potential applications, ranging from automatic target recognition to image-based industrial inspection tasks in assembly lines. While there have been some successful object and pattern detection systems in the past, most such systems handle only specific rigid objects or patterns that can be accurately described by fixed geometric models or pictorial templates.

This thesis presents a learning based approach for detecting classes of objects and patterns with variable image appearance but highly predictable image boundaries. Some examples of such object and pattern classes include human faces, aerial views of structured terrain features like volcanoes, localized material defect signatures in industrial parts, certain tissue anomalies in medical images, and instances of a given digit or character, which may be written or printed in many different styles.

The thesis consists of two parts. In part one, we introduce our object and pattern detection approach using a concrete human face detection example. The approach first builds a distribution-based model of the target pattern class in an appropriate feature space to describe the target's variable image appearance. It then learns from examples a similarity measure for matching new patterns against the distribution-based target model. We also discuss some pertinent learning issues, including ideas on *virtual example* generation and example selection. The approach makes few assumptions about the target pattern class and should therefore be fairly general, as long as the target class has predictable image boundaries. We show that this is indeed the case by demonstrating the technique on two other pattern detection/recognition problems.

Because our object and pattern detection approach is very much learning-based, how well a system eventually performs depends heavily on the quality of training examples it receives. The second part of this thesis looks at how one can select high quality examples for function approximation learning tasks. *Active learning* is an area of research that investigates how a learner can intelligently select future training examples to get better approximation results with less data. We propose an active learning formulation for function approximation, and show for three specific approximation function classes, that the active example selection strategy learns its target with fewer data samples than random sampling. Finally, we simplify the original active learning formulation, and show how it leads to a tractable example selection paradigm, suitable for use in many object and pattern detection problems.

Thesis Supervisor: Tomaso A. Poggio
Title: Uncas and Helen Whitaker Professor

Acknowledgments

Many people have enriched my life during my umpteen years here at MIT as a graduate student. I wish to thank them now for all that they have been to me.

First and foremost, I must thank Tomaso Poggio who has been my thesis supervisor since I started graduate school, and whom I have also come to look upon as a mentor and friend. I have had a truly memorable learning experience with him. I have benefited much working with him and with other researchers in his group. Eric Grimson and Tomás Lozano-Pérez served on my thesis committee. I would like to thank them for their time, and for all the invaluable feedback and suggestions they have given me about my work. I am also grateful to Eric Grimson for having sparked my interest in computer vision while I was here as an undergraduate, and for having been an excellent source of academic and administrative advice during my graduate school years.

Among my friends at the lab, many have made my stay a lot more enjoyable. Tao Alter has been my officemate and close "buddy" ever since we started graduate school together. I have always had his company in the office, be it late at night or during the weekends. My only regret is that I should have graduated a term earlier with him. I worked jointly with Partha Niyogi on the *Active Learning* material presented in Chapter 4. Partha has been a wonderful source of new ideas and an interesting traveling companion on conferences. We have had many thought provoking discussions, both on topics related to our work and on other things too numerous to mention. I have also greatly enjoyed interacting with David Beymer, Mark Bonchek, Tony Ezzat, Federico Girosi, Mike Jones, Tina Kapur, Lily Lee, Steve Lines, Liana Lorigo, Kimberly Lowe, Marina Meila, Jose Robles, Raquel Romano, Amnon Shashua, Pawan Sinha, Gideon Stein, Brian Subirana, Tanveer Syeda, Paul Viola and Sandy Wells.

I cherish the years I have spent with my long-time house-mates: Boon Seong Ang and Velusamy Subramaniam; my long-time friends: Chock Hing Gan, Choon Phong Goh, Hui Lin Lai, Tze Yun Leong, Beng Hong Lim, Su-Lin Low, Fiona Tan, Yang Meng Tan, Siang Chun The, Kathy Wang, Hsi-Jung Wu, Shih Jih Yao and Ming Huam Yuk; and my *virtual* [73] pets: Red-Herring *(Clupea harengus redus)*, Joyce *(Carcharodon carcharias joycias)*, Yuk *(Cervus canadensis yukis)*, Orange *(Ursus orangus)* and Mumbo Hippo *(Hippopotamus mumblus)*, whose images appear in my aquarium screensaver.

Most of all, I would like to thank my parents and sister who have loved and encouraged me through all these years. I love you all.

Contents

Chapter 1

Introduction

Vision is perhaps the most powerful perceptual sense that a human being or a machine can have. As humans, we are able to gather a remarkable amount of detailed information about things in our environment through our sense of vision, all without direct physical contact. Our visual ability helps us perform many ordinary but essential daily routines, such as recognizing people at work, walking through hallways without colliding into obstacles, reading newspapers and magazines, etcetera. In the world of machines, one can expect systems with visual sensors to be more versatile and robust than similar systems without visual sensors. The former has access to visual data, which it can use as an additional rich source of information for understanding its environment and interacting with the world more intelligently.

Machine vision can be described as a process that converts a digitized image of sensor values into a symbolic description of patterns and objects in the scene, suitable for subsequent use in a machine dependent task. One of the foremost goals in artificial intelligence has been to develop algorithms and equip machines with the ability to process and "understand" visual data in some meaningful fashion. Most current "image understanding" tasks fall under one or more of the following categories:

1. **Pattern Classification.** This category describes perhaps the widest range of machine vision tasks that researchers have been able to formulate algorithmically. A pattern classification problem involves assigning identities or labels to patterns in an image, or sometimes to an entire image. Many classical computer vision problems like image annotation, object recognition and object detection can be posed, at least in

part, as a pattern classification problem.

2. **Registration.** A registration problem involves establishing a match between an input image and a reference image or model. An image registration system "understands" the input image by explaining its appearance in terms of a transformed reference image or model.

3. **Reconstruction.** Reconstruction problems "understand" an input image by creating a computer representation of surfaces and objects in the scene. An appropriate representation scheme could be a depth map or a CAD model. In some reconstruction problems, the goal may be to determine the imaging conditions under which the image is taken, such as the distribution and color of light sources.

There are many potential areas for machine vision applications in the world today, ranging from surveillance and census systems to process control and human-computer interfaces. In industrial robotics, one can introduce vision sensors to help modern robots find and manipulate objects in their workspace. Vision systems can also be used to take over certain mundane tasks currently performed by human workers in various settings. A computerized camera that recognizes people can be used as a security system for access control. In medical image analysis, one can reduce operating costs by having vision systems take over certain tedious image screening and annotation tasks, like automatically segmenting MRI scans into regions of different anatomical parts for image-guided surgery — a technology that is just beginning to emerge. In manufacturing, there is an expected need for more sophisticated vision-based inspection systems in the twenty-first century to make processes less labor intensive.

Although much work has been put into the field of machine vision over the past twenty to thirty years, existing computer vision systems still fall far short of human abilities. While there has been some successful computer vision systems in the past, especially object recognition and localization systems based on pictorial templates and geometric models, most such systems perform very specific tasks at best, and operate only under very heavily constrained conditions. For example, in some systems, one can only deal with a very limited library of objects and patterns made up of only rigid parts. In other cases, lighting must be controlled and the imaging geometry must be fixed. Computer vision systems today are still too inflexible for any widespread use beyond the specific tasks that they have been designed to perform.

Why is computer vision a difficult problem? If one views vision as interpreting image patterns in terms of objects and surfaces present in a scene, then perhaps the main reason is that most real objects and surfaces can have very unpredictable image appearances. These unpredictable image appearances can be extremely difficult to encode, and hence interpret with computer programs. The image appearance of objects and surfaces can depend on many interacting factors, including pose, lighting conditions, surface reflectance properties, occlusion and sensor characteristics. Because there are many tightly coupled interacting factors that affect image formation, even a simple perfectly rigid object with uniform surface reflectance properties, can still give rise to a very large and complex set of possible image patterns. In general, a vision system may not even have prior information about some of the key imaging factors needed to predict object appearances, which makes the image pattern interpretation task even more ill-posed. So far, computer vision researchers have not been able to derive a sufficiently comprehensive scheme that can reliably recover and account for all the interacting factors contributing to the appearance of objects in an image.

When interpreting images with non-rigid objects, or when dealing with the notion of object classes (for example the class of human faces), the vision problem becomes even more complicated, because the system has to account for additional sources of pattern variation. These pattern variations can be due to non-rigid transformations in the shape of an object, or structural differences between individual objects from the same class. As an example of the former, a snake when viewed in a curled posture can give rise to a very different set of image readings than when viewed in an extended posture. Ideally however, a vision system should still be able to identify both sets of image readings as images of the same snake. In the latter case, the faces of two different people can appear very different even under identical imaging conditions, but ideally, a vision system should still identify both image patterns as faces. At a more abstract level, there are classes of objects, like chairs, that are identified primarily by a common functional property, and to a much lesser extent by a particular physical structure. To identify new objects as chairs, a vision system must not only be able to recover physical structure from an image; it has to extract semantic information from structure as well. As humans, our ability to understand images so effortlessly often leads us to underestimate the difficulty of designing computer algorithms for similar tasks.

Example-based learning is an area of research that has captivated the interest of many scientists, mathematicians and technologists over the last decade. The field deals with techniques of implementing and empirically discovering complex mappings between input patterns and output values. The *learning* approach to building systems is fundimentally different from the traditional approach of writing programs to encode knowledge. In *learning*, the "programmer" trains a system using a set of representative input-output examples, and the system adapts itself by modifying its state appropriately to perform the desired information processing task. Knowledge about the desired task is encoded in the set of examples provided by the "programmer". Existing example-based learning techniques have been successfully used in a wide range of multi-variate problems characterized by complex relationships between input and output variables, such as stock market prediction and process control. We believe that the same learning techniques that have worked well in many areas, can be similarly applied to a wide range of computer vision problems with comparable success.

Because *learning* is essentially "programming" a system from examples, how well a system eventually performs in its task depends heavily on the quality of examples it receives during training. The problem of obtaining high quality examples that capture sufficient information about a task, is therefore a very critical issue that should always be carefully addressed in any non-trivial example-based learning application.

1.1 Problem Definition

This thesis looks at a classical computer vision problem on detecting objects and patterns in images. There has been some successful work on restricted versions of this problem in the past. One set of simplifications allows for only specific rigid objects and patterns that can be described by fixed geometric models or templates. An example problem in this class is to recognize and locate (i.e. detect) telephones of a particular design in an office scene. The family of *interpretation-tree* based object recognition and localization algorithms [40] [42] is a popular and well tested approach for solving problems in this class.

In general, the scope of object and pattern detection covers a wider range of object and pattern types, including *non-rigid* bodies and *classes* of patterns. By non-rigid bodies, we mean specific objects that can undergo non-rigid shape transformations, and hence cannot be adequately described by a fixed geometric model. A specific snake or a particular person's face are examples of non-rigid bodies. Object and pattern *classes* refers to collections of individual objects and patterns that share some common identity. One example is the class of human faces, which includes faces of all people. Another example is the digit class "2", which comprises many different instances of machine printed styles and hand written patterns of the digit "2". While a few different approaches exist for dealing with the more general case of detecting non-rigid objects and pattern classes under varied imaging conditions, there has been limited success in this area to date.

1.1.1 Detecting Spatially Well-Defined Pattern Classes

Ideally, an object and pattern detection approach should be able to deal with the full spectrum of non-rigid, highly articulate and arbitrarily shaped objects, as well as highly varied *classes* of objects and patterns in different contextual settings. Unfortunately, we believe that such a goal is still beyond our reach with current computer vision technology. From a pattern interpretation standpoint, an approach that recognizes highly articulate and arbitrarily shaped objects or pattern classes, such as humanoid forms, must not only be able to correctly identify isolated image patterns as instances of some known object or class. It must also be able to first extract patterns from their image backgrounds. In general, pattern segmentation is still an unsolved computer vision problem, especially when there are few constraints and little prior knowledge to spatially bound the patterns. An alternative strategy that avoids explicitly segmenting complex patterns from images, is to define simpler sub-pattern classes that can be easily isolated and identified in images. At some later stage, the detection algorithm must analyze the global arrangement and composition of these sub-patterns in the image to identify and locate the full target pattern. Unfortunately, this approach is also unreliable at best because vision researchers today are still unclear about the key processes behind integrating local and global image information for interpreting scenes.

We shall instead explore a reduced version of the general object and pattern detection problem, that deals only with image patterns whose spatial boundaries can be well estimated a-priori. Because we have good prior boundary estimates for these image patterns, one can simply extract them from the image using one of several fixed shape masks, without having to actually perform the difficult general image segmentation task. With these simplifications, one can cast object and pattern detection as a *classification* problem on image patches, where the classifier's task is to determine the identity of each patch using only spatially local image measurements from within the patch. For tractability reasons, we shall also impose that each target pattern class can have only a small number of possible boundary shapes and sizes, so one can search for these target patterns using only a few masks and classification routines.

A spatially local object and pattern detection framework has the following two desirable characteristics:

1. Because the framework performs only local and spatially invariant image operations, one can efficiently implement applications within this framework on massively parallel hardware and SIMD machine architectures for real-time performance.

2. Because all image operations within this framework are local in nature, one can expect to gain a lot more insight from a successful application by analyzing and understanding the problem in terms of only local image measurements.

Applications wise, the local framework is well suited for detecting classes of patterns with minor but nevertheless significant shape and texture variations. We shall henceforth refer to pattern classes with these properties as *spatially well-defined* pattern classes. The possible image appearances of a specific non-rigid object that can only undergo minor shape deformations, such as a particular person's face, can be treated as a *spatially well-defined* pattern class. Henceforth, we shall also refer to specific objects of this sort as *semi-rigid* bodies. Other examples of *spatially well-defined* pattern classes include localized material defect signatures in industrial parts, aerial images of structured terrain features like volcanoes, anomalous tissue appearances in medical images and the set of all isolated human face views.

Clearly, the proposed framework is not well suited for detecting arbitrarily shaped pattern classes or extremely non-rigid and highly articulate objects, because one cannot obtain good prior boundaries estimates to isolate these patterns for classification. Nevertheless, we argue that our work here still represents a notable improvement over current non-rigid object and pattern detection approaches in terms of its generality and the good empirical results it has produced.

1.1.2 Formulation

We formulate our local object and pattern detection problem as one of learning to identify target image patterns from examples. To deal with the unpredictable pattern variations within the local pattern boundaries, we statistically model the distribution of target patterns in an appropriate feature space. We also define a set of distribution dependent distance feature measurements which we use as a "difference" notion for matching new patterns with our distribution-based model. Finally, we train a classifier to separate target patterns from distractor patterns using the distribution dependent distance feature measurements as input.

A key issue in our learning-based approach is to maintain a comprehensive but tractable training database of target and distractor patterns. Because *learning* is essentially "programming" a system from examples, how well a pattern detection system eventually performs in our approach depends heavily on the quality of examples it receives during training. We introduce two newly developed techniques for managing training databases. The first is the idea of exploiting prior domain dependent knowledge to generate *virtual training examples* [73] from existing ones. We use these virtual examples to artificially enlarge our training data set for a more comprehensive sampling of input patterns. Because the idea of generating *virtual examples* has already been carefully addressed in several recent papers [73] [10] [11], we shall not dwell too deeply on this topic other than to briefly mention how we have applied this idea in our sample pattern detection applications.

The second is the idea of selecting only useful patterns from among many redundant ones for training, in order to keep the number of training patterns reasonably small and the learning problem computationally tractable. We propose a "boot-strap" [89] [90] example selection strategy that incrementally finds highly informative new patterns between training runs. We shall devote an entire chapter of this thesis to discuss the example selection issue in depth, within the context of a related class of learning problems, called *active learning.* We believe that good example selection strategies can make a huge difference in helping one deal with very large learning problems that could otherwise be hopelessly unmanageable.

1.1.3 Goals

Our work is an attempt to formalize and demonstrate a general technique for taking on a restricted but fairly wide class of spatially well-defined object and pattern detection problems. The approach consists of: (1) a proposed system architecture to implement object and pattern detection tasks, and (2) a set of operating guidelines and procedures for developing applications using the given system architecture.

The individual components within the proposed system architecture are not new. In fact, the subsequent chapters will show how individual components of our system architecture relate to existing techniques in classical fields like statistics, linear algebra, regularization theory and pattern classification. To the best of our knowledge however, our work is the first attempt of integrating and understanding these separate components as parts of an overall framework for modeling and detecting semi-rigid objects and patterns.

Much of this thesis focusus on interpreting the functions performed by the individual system components, their underlying assumptions and limitations. We believe that an intelligent understanding of the proposed technique, its components and rationale behind the operating guidelines, will all be critical for successfully applying this framework to real problems.

1.2 Pattern Detection, Recognition and Classification

In this section, we shall look at *pattern detection* and a very closely related problem called *pattern recognition* in greater detail. While the task descriptions of the two problems may differ considerably in certain domains, we shall argue that both problems are in fact instances of a wider class of computer vision problems, called *pattern classification*. We shall also show that in some other problem domains, the distinction between a *detection* task and a *recognition* task is somewhat artificial, and often merely a matter of terminology. The issues influencing *pattern detection* and *pattern recognition* problems are therefore very similar, and their solutions have a lot in common.

1.2.1 Pattern Detection and Recognition

The goal of *pattern detection* is to automatically locate, in an image, instances of objects that a computer has been trained or programmed to identify. Given prior knowledge of some known objects or classes of objects and a digitized image of a scene to analyze, the pattern detection task is to return an encoding of the location, spatial extent and possibly pose of each known object in the scene. An example of a pattern detection task is human face detection. The task involves analyzing a possibly heavily cluttered image with zero or more human faces, and identifying portions of the image that correspond to faces. A very closely related problem to pattern detection is *pattern recognition*, whose goal is to compare a given image pattern against models in a library of known patterns, and to report the best matching model if a match is found. Face recognition is a popular example of a pattern recognition problem. Here, the task is to establish the identity of (i.e. recognize) a person from an input image of an isolated human face. Most face recognition systems identify faces by matching the input image against face models in a library of known people.

Many real world computer vision applications perform tasks that are both pattern detection and pattern recognition in nature. In some applications, there can be a clear division of work into a detection stage and a separate recognition stage. Returning to the domain of human faces, let us consider an automatic surveillance problem of identifying people in an arbitrary scene. One plausible approach divides the problem into a *detection* stage for locating human faces in an input image, and a separate *recognition* stage for establishing individual identities from the isolated faces.

In other computer vision applications, there may not be a clear division of work into a distinct detection stage and a separate recognition stage. Nevertheless, these applications still perform tasks that involve both locating objects and identifying them in images. The following is an example from classical model-based "object recognition". Consider a typical "object recognition" problem of finding telephones of a particular design in cluttered office scenes, among other common desk top objects like books, pens and staplers. One popular approach uses image features to first generate likely hypotheses about the locations and poses of telephones in the image. Each hypothesis is then tested by matching an appropriately transformed telephone model to the image region containing the telephone. Notice that in this approach, the final matching step fulfills the role of both an object recognizer and detector, because each successful match identifies an image object as a telephone and also locates the object in the scene. Notice also that although computer vision literature commonly refers to problems like the above as "object recognition" problems, the actual tasks performed in these problems are clearly both object *recognition* and *detection* in nature.

Algorithmically, a *pattern detection* problem can often be re-expressed as a *pattern recognition* task within an appropriate driver framework. We provide one such example to illustrate what we mean, and to argue that pattern *detection* and *recognition* are in fact very similar problems in spirit. We shall consider again the example of face detection. Computational efficiency aside, one possible approach of finding faces is to test all local image windows over a range of sizes for "face-like" pattern properties, and to report the location and scale of all successful matches. Notice that in this framework, the embedded test procedure performs essentially a *pattern recognition* task. It identifies face window patterns from among all natural occuring background window patterns — i.e. it *recognizes* the class of face patterns. Recall that the *face recognition* problem is to identify a person from an input face image. The task can be organized as many single-person face recognizers working in parallel, where each single-person face recognizer identifies all instances of a given person's face from an input domain of all isolated face images — i.e., each single-person face recognizer recognizes the class of images corresponding to a given person's face, from among the class of all face patterns. The face recognition and face detection problems are thus fundementally very similar in spirit. They are perceived as being different problems only because they operate on different input domains and assign different output class labels.

1.2.2 Pattern Detection and Recognition as Classification Problems

Both *pattern detection* and *pattern recognition* fall under a wider class of vision problems, called *pattern classification.* Let $\mathcal{X}$ be the set of all input patterns and $\mathcal{W} = \{w_1, w_2, \ldots, w_N\}$ be the set of all possible output classes or labels. For each input pattern $x \in \mathcal{X}$, let $z_x \in \mathcal{W}$ be the true class label for x. The goal of *pattern classification* is to construct a functional mapping, $\mathcal{F} : \mathcal{X} \mapsto \mathcal{W}$, such that $\mathcal{F}(x) = z_x$ for all input patterns $x \in \mathcal{X}$. The function $\mathcal{F}$ is commonly known as a *classifier.* When $|\mathcal{W}|$ =N= 2, we have a special case called a 2-class pattern classification problem. Many real world N-class pattern classification applications are implemented as N 2-class pattern classifiers operating in parallel, with a special arbitration stage to resolve output conflicts.

As an example of how pattern detection and recognition are related to pattern classification, we shall show how both face detection and face recognition can be cast as pattern classification problems. For face detection, the task is to identify the class of face window patterns from an input domain, $\mathcal{X}$, of all natural occuring background patterns. The set of output class labels is $\mathcal{W} = \{\texttt{Face}, \texttt{Non}-\texttt{Face}\}$, and the face detector is simply a classifier that performs the following mapping: $\mathcal{F}(x) = \texttt{Face}$ if $x \in \mathcal{X}$ is a face window pattern, and $\mathcal{F}(x) = \texttt{Non}-\texttt{Face}$ otherwise.

For face recognition, we consider first the broader case of a database with N people. The input class $\mathcal{X}$ is the set of all appropriately segmented human face images, and the set of output class labels is $\mathcal{W} = \{\texttt{Person}_\texttt{1}, \ldots, \texttt{Person}_\texttt{N}, \texttt{Unknown}\}$. The face recognizer performs the following mapping: for all $x \in \mathcal{X}$, $\mathcal{F}(x) = \texttt{Person}_\texttt{i}$ if x is a face image of the $i^{\texttt{th}}$ person in the database, and $\mathcal{F}(x) = \texttt{Unknown}$ otherwise.

A N-person face recognizer can be implemented as N single-person face recognizers operating in parallel, with a special arbitration stage to resolve class label conflicts. Each single-person face recognizer identifies all face images of a given person, from an input domain of all appropriately segmented human face images. It can therefore be cast as a 2-class pattern classification problem, whose input class $\mathcal{X}$ is the set of all appropriately segmented face images, and whose output class labels are $\mathcal{W} = \{\texttt{KnownPerson}, \texttt{UnknownPerson}\}$. The classifier performs the following mapping: for all $x \in \mathcal{X}$, $\mathcal{F}(x) = \texttt{KnownPerson}$ if x is a face image of the given person in the database, and $\mathcal{F}(x) = \texttt{UnknownPerson}$ otherwise.

1.2.3 Difficulty

As mentioned earlier, most successful object detection and pattern recognition systems today still operate under heavily constrained imaging conditions, and handle only very restricted classes of target objects (usually only specific rigid objects or specific rigid objects with moving parts). The underlying techniques in these systems tend to perform poorly for detecting non-rigid and semi-rigid objects, as well as classes of patterns under more general imaging conditions. What makes object detection and pattern recognition difficult vision problems? Clearly, both object detection and pattern recognition inherit many of the difficulties common to computer vision problems in general. In object and pattern detection, perhaps the biggest problem is that non-rigid and semi-rigid 3D objects can have very unpredictable appearances, when viewed under different pose and imaging conditions. That is, the same non-rigid or semi-rigid object can take on one of many very different sets of image values, when projected onto a 2D image plane. For classes of objects, one also has to deal with physical variations between individual members of an object class that can be difficult to quantify. Often, these image differences are significant enough so that traditional pictorial template-based matching techniques and geometric model-based object recognition methods cannot fully capture all the permissible pattern variations.

A second and closely related problem is scene clutter. Scene clutter increases the variety of image patterns that an object detector has to deal with, which in turn makes correct classification in an object detection task much harder. If the input scene were simple with little or no clutter, then an object detection task can still be fairly straight forward, even when dealing with highly complex objects or pattern classes. This is because even though it may be difficult to fully model all possible image appearances of a specific highly complex object or pattern class, one can still rely on less comprehensive models and coarse image measurements to correctly identify instances of the target from trivial background distractor patterns. Unfortunately, most real world object detection applications deal with highly cluttered scenes containing many objects in complex background texture. Even without occlusion, the object detector has to correctly identify instances of the target from among many other very similar looking image patterns. To do this, we need very precise modeling schemes that correctly account for all possible image appearances of a target object or pattern class, while correctly rejecting even very similar looking background patterns. This brings us back to the first difficult problem we were trying to address, which is the need for better object and pattern representation schemes.

1.3 Previous Work in Recognizing and Detecting Spatially Well-Defined Patterns

As previously discussed, the key issue and difficulty in detecting non-rigid objects and large pattern classes is to correctly account for the wide range of possible pattern variations that these bodies can exhibit in images. There have been four main approaches for dealing with variations in large pattern classes, namely the use of: (1) view-based correlation templates, (2) sub-space methods, (3) deformable templates, and (4) image feature invariants. Most of these techniques have been used for detecting either specific semi-rigid objects or classes of spatially well-defined objects and patterns in images.

1.3.1 View-based Correlation Templates

Fixed correlation templates are like matched filters. In object detection, they compute a difference measurement between a fixed reference pattern and candidate image locations, and the output is thresholded for matches. While most semi-rigid objects and large object classes are too complex to have all possible views modeled by a single fixed template, there are some object detection techniques that use a bank of several correlation templates to account for large image appearance variations. One simple and direct scheme is the *view-based* approach. Here, variations in object appearance and structural differences between individual members of a class are represented by simply storing many example 2D views of the target. The views may be from a variety of poses, lighting conditions, shape deformations and any other sources of variability that one wishes to handle in the detection problem. When analyzing a new image location, the detector simply tries to match the input pattern against stored patterns that are sufficiently close in appearance.

Since pose, lighting and shape deformation are all multi-dimensional parameter spaces, populating this space densely enough with sample views can require an intractably large number of stored patterns. One key issue in view-based approaches is to determine a reasonably small set of views necessary for acceptable detection performance.

The view sampling problem has been studied extensively for variations in pose space of a specific object. If the viewing distance to the target object is fixed, then the problem becomes one of sampling views from a spherical surface centered at the object (a.k.a. *viewing sphere*), and the key issue is to determine an appropriate number and distribution of image samples that would adequately cover all possible views of the object.

When dealing with simple objects or small variations in pose, one can possibly get by with a tractable number of regularly spaced views. In the view-based system of Breuel [15], two airplane toy models are represented by sampling only the upper half of the viewing sphere. Only 32 views are used for one plane and 21 views for the other. In a pose independent face recognition task, Beymer [12] showed that by making some fine image alignment adjustments during run-time, one can very reliably represent human faces over a fairly wide range of poses with only 15 views for identification purposes.

To handle more complex objects over a wider range of pose, a number of researchers have used *aspect graphs* [52] to formally analyze the view sampling problem. An *aspect graph* is roughly defined as an object view whose features are stable with respect to pose perturbations. Each aspect carves out a patch of qualitatively similar views from the viewing sphere. The aspect graphs are used to derive sets of similar views from models of 3D objects [23] [53] [48]. These techniques have been successfully applied to polyhedral objects [87] [37] and curved objects constructed with parametric surfaces [74].

A closely related extension of *view-based* correlation templates is the *linear combination* approach for modeling object appearances [97]. This technique is best known in its original formulation for dealing with image changes due to variations in pose of a specific object. It can also be directly used to account for image appearance changes due to shape variations or physical differences between objects in a target class. The *linear combination* approach works as follows: Instead of storing a dense set of 2D views as isolated patterns for modeling object appearances, it preserves only a small number of sample views as examples. To generate intermediate reference "templates" for pattern matching, the approach interpolates from the stored example views. Ullman and Basri have shown that any intermediate 2D view of a specific object can be written as a linear combination of example 2D views under orthographic projection and in the absence of occlusion. The approach has one major drawback. It assumes that one has already established accurate point or contour feature correspondances between the example views and the new pattern being identified. This is a difficult problem even for moderately complex objects and pattern classes, which makes the *linear combination* technique highly impractical for modeling pattern variations in object detection tasks.

Overall, the *view-based* template techniques show that a sufficiently dense set of 2D views is equivalent to having the 3D structure of a specific object. The underlying idea is similar to binocular stereo and structure-from-motion algorithms, where multiple 2D views of an object are used to compute 3D structure.

1.3.2 Sub-Space Methods

Another closely related approach to correlation templates is that of *view-based eigenspaces* [85] [51] [69]. As in the *view-based* template and *linear combination* approaches, the technique collects a large set of views for the target object or pattern class, sampled under all the varying conditions that one wishes to account for. To generalize about the target object's appearance from the set of collected views, the approach assumes that the set of all possible object views occupies a small and easily parameterizable linear sub-space of the high dimensional view space. So instead of storing all the sample views, the technique constructs a compressed representation of the sub-space of object views and uses this sub-space representation as a reference “template” for detecting objects.

Typically, one recovers the sub-space of object views by performing *principal components analysis* (PCA) [31] [35] on a large set of sample views, and preserving only the largest few principal components. If one assumes that the principal components, or eigenvectors, capture the different causes of variation in object appearance, then the largest principal components correspond to the key sources of variation. By using only the most significant principal components to describe the sub-space of object views, one can argue that the technique is effectively preserving only the semantically meaningful sources of variation in object appearance, without factoring into the description meaningless pattern variations due to noise.

During object detection, the approach computes and thresholds a difference measure which is the Euclidean distance between an input pattern and the sub-space of object views. Because the approach assumes that the linear sub-space represents all possible target object views, one can treat the Euclidean distance so computed as a difference indicator between the input pattern and the target object class. It should therefore be reasonable to identify image patterns as target objects based on such a distance metric.

Murase and Nayar [64] use a similar parameterizable sub-space approach for modeling the 2D appearance of objects and pattern classes. Although they have only demonstrated their technique on recognizing isolated specific objects under varying pose, one can apply the same idea directly for detecting specific objects under semi-rigid shape deformation, or for classes of patterns in images. As in the *eigenspace* approach, sample views are represented by projecting them onto a small number of principal components, forming an “eigenspace” representation. Murase and Nayar take the representaion one step further by fitting a parameterized surface to the sample projections in the eigenspace. Given a new pattern to identify, the technique first projects the new pattern onto the eigenspace, and compares the projected location with the hypersurface of sample views.

So far, the sub-space approaches described above have only been used for identifying specific isolated target objects, or for detecting object classes like human faces in images with little scene clutter.

1.3.3 Deformable Templates

Deformable templates work like classical correlation templates, except that the former has some built-in non-rigidity component. It is this non-rigidity component that makes them better suited for describing and detecting semi-rigid objects and classes of patterns in images. In pattern detection, the overall paradigm for finding objects with deformable templates is very similar to that of classical correlation templates. One simply fits a deformable template to candidate image locations and thresholds the output for matches.

One common application of deformable templates has been in modeling the image appearance of large, spatially well-defined pattern classes like human faces. In one implementation, Yuille, Hallinan and Cohen [104] use hand constructed parameterized curves and surfaces to model the non-rigid components of faces and facial sub-features, such as the eyes, nose and lips. The parameterized curves and surfaces are fixed elasitcally to a global template frame to allow for minor variations in position between facial features. To locate faces in an image, one uses a matching process that aligns the template with one or more pre-processed versions of the image, such as the peak, valley and edge maps. An energy functional constrains the alignment process by attracting the parameterized curves and surfaces to corresponding image features, while penalizing "deformation stress" in the template. The best fit configuration is found by minimizing the energy functional, and the minimum value also serves as a closeness value for the match.

A related deformable template approach uses a global head model defined by tens of individual feature locations [7] [27] [25] to represent the appearance of human faces. During a match, the model is aligned to the image by varying individual feature locations. In another related approach, Terzopoulos and Waters [92] have used an active contour model of snakes to represent human facial features and track them across image sequences.

1.3.4 Image Feature Invariants

In the invariants approach, the key is to find a class representation that holds true even as the target varies in shape, or as the target is viewed under different pose and lighting conditions. If such a representation exists, then the matching process in object detection is quite simple. We compute the invariant representation at all candidate image locations and report a match wherever the local image pattern satisfies the invariance condition. Clearly, this approach is only possible if one can find object features that are truly independent of pose, shape deformation and imaging conditions. When such features do exist howeever, this technique is often preferable to the *view-based* and *deformable template-based* approaches, because an invariance-based representation tends to be much simpler than a view-based or a deformable template-based model.

One example of pose-invariant features, useful for detecting only specific rigid objects, is based on a relative distance idea called *cross ratios*. Consider four collinear points A, B, C and D, let AB denote the planar distance between points A and B in the image, and so on. It can be shown that the cross ratio $\frac{AB}{AC} \div \frac{AD}{BD}$ is invariant with respect to pose. Forsyth et. al. [34] have generalized the above observation to develop geometric invariants of a similar flavor for 3D planar objects using contour and point features. The same idea, however, does not extend well to more complicated and semi-rigid 3D objects. For instance, Clemens and Jacobs (cite) have shown that for non-planar 3D objects defined by an arbitrary set of feature points, there are no geometric invariants.

A closely related idea to geometric invariants on coutour and point features is that of *spatial invariants* on image regions. One such scheme is based on image intensity invariants between different parts of a specific object or objects in a target class. The underlying assumption is that while illumination, shape deformation and other variations can significantly alter image brightness values at different parts of an object in a target class, the local ordinal structure of brightness distribution remains largely unchanged. Sinha [84] has applied this idea to the problem of detecting human faces under varying lighting conditions. He showed empirically that pairs of regions exist on a human face, where the average brightness of one region is consistantly brighter or darker than the other. For example, the eye regions of a face are almost always darker than the cheeks and the forehead, except possibly under some very unlikely lighting conditions. Similarly, the bridge of the nose is always brighter than the two flanking eye regions. To exploit these image intensity invariants for finding faces, Sinha encodes these obseerved brightness regularities as a *ratio template* which he uses to pattern match an input image for faces. The ratio template is a coarse spatial template of a face with a few appropriately chosen sub-regions that roughly correspond to key facial features. The brightness constraints are captured by a set of pairwise brighter-darker relationships between the corresponding sub-regions. An image pattern matches the template if it satisfies all the pairwise brighter-darker constraints.

The color distribution of a specific object can also be used as an invariant feature. Swain and Ballard [91] use a histogram of image colors to perform indexing into a library of objects. They use this stage to reduce the number of possible target object hypotheses prior to more detailed matching.

1.4 Example-based Learning for Object and Pattern Detection

Example-based learning is an area of research that has captivated the interest of many technologists, scientists and mathematicians over the last decade. The field deals with models of memory retrieval and techniques for empirically discovering complex relationships in sparse data. The *learning* approach to "software development" is fundimentally different from the traditional *programmed computing* approach. In *example-based learning*, a "programmer" trains a system to perform an information processing task by providing the system with input *features measurements* and corresponding output *values* of the task. The system "learns" the task from the input-output examples the programmer provides by adaptively modifying its state to implement an appropriate mapping. Example-based learning techniques have been used in many areas ranging from stock market prediction and signal processing to motor control in robots. It is this idea of *training* machines instead of having to actually program them that makes learning especially appealing in areas where general algorithms for dealing with problems are still relatively unavailable.

In this thesis, we formulate the detection problem for *spatially well-defined* objects and pattern classes as learning to recognize instances of a target pattern class from example image feature measurements. Here, we are using learning methods to complement human knowledge for capturing complex variations in the image appearance of objects and patterns. The learning task is performed in an appropriate feature space of image measurements. The exact choice of image features depends largely on the particular problem at hand. For some object classes, it may be most convenient to use view-based image features. For other pattern classes, the learning problem may be much simpler with feature measurements derived from a transformed image domain, such as a local 2D power spectrum. We shall look at some reasonable heuristics for choosing appropriate feature measurements later in Chapter 3.

The key idea behind our learning-based object and pattern detection approach is as follows: Rather than trying to manually parameterize all the complex image variations of a target pattern class, we simply collect a sufficiently large number of sample views for the pattern class we wish to detect, covering all possible sources of image variation we wish to handle. We then choose an appropriate feature space to represent the pattern class as a distribution of all its permissible image appearances. Finally, we train a decision procedure to correctly identify instances of the target pattern class from background image patterns, based on a set of distance measurements between the input pattern and the distribution-based class representation in the chosen feature space. During training, the decision procedure learns from example target and background patterns a set of operating thresholds and parameters that performs the desired classification task.

Our learning-based approach has the following advantages over existing object and pattern detection techniques: First, the distribution-based modeling scheme does not rely much on domain specific knowledge or special hand-craft techniques to accurately parameterize the patterns we wish to describe. This immediately eliminates one potential source of modeling error — that due to incomplete or incorrect knowledge. Unlike *deformable template* approaches or *image invariance* techniques whose object models are based heavily on prior knowledge and assumptions, our modeling scheme essentially builds models that describe an empirical distribution of sample patterns. So as long as we provide our scheme with a sufficiently comprehensive sample of training views, we can expect our distribution-based models to be more accurate and more descriptive than the manually synthesized representations examined earlier.

Second, unlike most non learning-based object detection approaches that typically obtain their operating parameters and thresholds manually from a few trial cases, our scheme derives its classifier parameters and thresholds automatically from a large number of input-output training examples. This makes our scheme potentially superior in two ways: (1) Given any decision procedure with free thresholds and parameters to learn, we can expect our scheme to arrive at a statistically more reliable set of operating values because it is able to process a wider sample of training data automatically. (2) Because our scheme automatically learns thresholds and parameters, it can be easily made, if necessary, to learn high-dimensional and non-linear relationships on feature measurements for identifying the target class from background patterns. These relationships, even if they do exist, may be too complex for human observers to discover manually.

Third, our approach is better suited for building increasingly robust and arbitrarily complex pattern detection systems. Because our approach uses an example-based representation scheme and training methodology, one can, in principle, make existing detection systems more robust in our approach by simply increasing the number and variety of training examples. Both *false positive* and *false negative* detection errors can be easily corrected by further training an existing system with the patterns it wrongly classifies. The same may not be true for systems based on other classical object detection techniques, whose performance is often limited by finite human knowledge and concepts that can be conveniently expressed as computer algorithms. Functionality wise, systems based on our approach can also be highly extensible. To detect objects and patterns over a wider range of conditions, one can, in principle, re-train a system with a larger example database that covers all the new sources of image variation one wishes to handle.

We now review some key issues and ideas from example-based learning, which we shall utilize in our local object and pattern detection approach.

1.4.1 Example-based Learning and Function Approximation

Learning from examples is a common supervised-learning paradigm that hypothesizes a target concept given a stream of input-output examples that describes the concept. In the domain of real numbers, the task of learning an input-output "concept" from a set of examples is essentially equivalent to approximating a multivariate function that (1) maps input examples onto their respective output values, and (2) reasonably interpolates between output values at regions of the input space where no examples are available [71].

The learning problem can thus be more formally stated as follows: Let $\mathcal{D} = \{(\vec{x_i}, y_i) \in \Re^d \times \Re | i = 1, \ldots, n\}$ be a set of n data points sampled from an unknown multivariate function $f(\vec{x})$, possibly in the presence of noise. The task is to recover the function $f(\vec{x})$, or at least a reasonable estimate of it, by means of an approximation function from a function class $F(\vec{w}, \vec{x})$, parameterized by the vector $\vec{w}$. For a fixed function class F, the problem is then to find the set of parameters $\vec{w}$ that best approximates $f(\vec{x})$ based on information from the set of "examples" $\mathcal{D}$.

Clearly, how well one can approximate an unknown target function $f(\vec{x})$ depends heavily on the following two factors:

1. **The function class $F(\vec{w}, \vec{x})$.** Needless to say, it is very important to choose an approximation function class F that can represent the unknown target function f sufficiently well. There would be little point in trying to recover $f(\vec{x})$, if the chosen approximation function class $F(\vec{w}, \vec{x})$ only gives a very poor representation of $f(\vec{x})$ even with optimal parameter values. The choice of which function class, F, to use is known as a *representation* problem. Some popular network-based function classes with universal approximation properties include *multilayer perceptron* nets [67] [77] and *radial basis function* nets [62] [71]. A closely related issue is the *complexity* of the approximation function class, which is often measured by the number of free parameters in $\vec{w}$. A more complex function class usually has a better chance of approximating an unknown target function well. However, it also requires a larger number of data samples to arrive at a reasonable approximation for predicting unseen data (see [66] for the case of *radial basis function* nets).

2. **The data sample $\mathcal{D}$.** The accuracy of an approximation also depends on the quality of data in $\mathcal{D}$, i.e. the quality of available information about the unknown target function f. A larger data sample charts the output value of f at more input locations, and hence conveys more information about f. It is well known in learning theory that as the complexity of the approximation function class F increases, one must also increase the number of data samples in $\mathcal{D}$ to avoid large approximation errors due to overfitting. The distribution of data samples is another critical aspect of $\mathcal{D}$ that has often been overlooked in example-based learning. Ideally, we want a well distributed data sample that provides a balanced representation of f. If the learning objective is to closely approximate the unknown target function f at all input locations, then there is little point in collecting a lot of data at one input location while ignoring other input locations. Similarly, there is no point in collecting a lot of data at input locations where the chosen function class F is slowly changing.

We shall pay very close attention to the problem of obtaining high quality data samples in our learning-based object and pattern detection approach.

1.4.2 Pattern Classification as a Function Approximation Problem

We have argued that an *object detection* task can be formulated as a *pattern classification* problem. One can approach *pattern classification* as learning an approximation function for predicting the class identities of input patterns. Consider a 2-way classification task that operates on input domain $\mathcal{X}$ with output classes $\mathcal{W} = \{w_0, w_1\}$. For each input pattern $x \in \mathcal{X}$, let $z_x \in \mathcal{W}$ be the true class label for x. Recall that the goal of pattern classification is to construct a functional mapping, $\mathcal{F} : \mathcal{X} \mapsto \mathcal{W}$, such that $\mathcal{F}(x) = z_x$ for all input patterns $x \in \mathcal{X}$.

The classifier $\mathcal{F}$ can be implemented as a real-value target function f that computes the following conditional probability density: $f(x) = P(z_x = w_1|x)$. We have:

$$\mathcal{F}(x) = \begin{cases} w_1 & \text{if } f(x) = P(z_x = w_1|x) > 0.5 \\ w_0 & \text{otherwise} \end{cases}$$

To construct $\mathcal{F}$, we simply learn the conditional probability density function f from examples. In practice, it may not even be necessary to learn the full conditional probability distribution function $f(x) = P(z_x = w_1|x)$. Any regression function that interpolates reasonably between the available data samples should suffice. The training data set $\mathcal{D}$ consists of (x, y) pairs in $\mathcal{X} \times [0.0, 1.0]$. Each $x \in \mathcal{X}$ is an example input pattern in the original problem domain. The corresponding output y value is 1.0 if $z_x = w_1$, and 0.0 if $z_x = w_0$.

1.4.3 Exploiting Prior Knowledge in Learning

A common pitfall in example-based object detection approaches, and more generally example-based learning problems, is that there may not enough training examples available to learn the problem well. Recall that in order to accurately represent an unknown target function $f(\vec{x})$, one must use an approximation function class $\mathcal{F}(\vec{w}, \vec{x})$ of sufficient complexity. In doing so however, one also needs a sufficiently large number of data samples to avoid large approximation errors due to overfitting. With too few training examples, an object detection scheme may not generalize properly, and hence may not be able to identify variations of the target object that appear too different from the training views. To overcome this potential pitfall, we explore ways of using domain dependent prior knowledge to artificially expand the training data set by generating *virtual examples* [73] of the target object. Here, we are assuming that with the addition of virtual training examples, we have a denser sample of training data which helps an object detector generalize better.

Some recent pieces of work have used the idea of *virtual examples* to generate novel image views of objects and patterns. In an early demonstration, Poggio and Vetter [73] showed that for bilaterally symmetric objects, one can use knowledge of symmetry to generate a virtual view of an object from a single real view. The virtual view is simply the mirror image of the real view. Since then, Beymer and Poggio [11] have further developed the virtual views idea in a variable-pose face recognition approach that builds models using only one example view of each person. They show that one can learn mappings that transform faces from a standard pose to one of many virtual poses. One can think of these learned mappings as prior information about the structure of faces. By using these mappings, they are able to generate enough example views of a face from a single view to perform pose-independent face recognition. In hand-printed digit recognition, Simard et. al. [83] showed that one can greatly improve the correct classification rate of digit recognizers by training with additional virtual examples of hand-printed digits. To generate virtual examples of digit patterns, they use a set of simple affine transformations and morphological operations, based on prior knowledge that the identity of digits do not change under these transformations.

The idea of *virtual examples* is closely related to *hints*, a framework proposed by Abu-Mostafa [2] [3] for exploiting prior knowledge in example-based learning. In example-based learning, the goal is to represent an unknown target function f by means of an approximation function from the class $\mathcal{F}$. A *hint* is any prior information that helps reduce the size of $\mathcal{F}$, thus making the learning task easier with fewer functions to consider.

One type of hint which best describes the idea of *virtual examples*, assumes that the function f is invariant over known partitions of the input space. That is, if $\vec{x}$ and $\vec{x'}$ are two input patterns belonging to the same partition, then $f(\vec{x}) = f(\vec{x'})$. Abu-Mostafa showed that the *invariance* hint can be easily incorporated into a standard backpropagation multilayer perceptron net training algorithm as follows: For each input-output pair $(\vec{x}, f(\vec{x}))$, backpropagation computes the current network output $\mathcal{F}(\vec{w}, \vec{x})$ and feeds back the error $(f(\vec{x}) - \mathcal{F}(\vec{w}, \vec{x}))^2$ through the network to modify weights $\vec{w}$. Suppose $\vec{x'}$ belongs to the same invariance partition as $\vec{x}$, then the modified algorithm uses the error $(\mathcal{F}(\vec{w}, \vec{x}) - \mathcal{F}(\vec{w}, \vec{x'}))^2$ to update weights as well. In generating virtual examples, one is essentially treating the operations being used as a set of transformations that define an invariance partition.

Most *learning* researchers today would agree that the availability of training examples is perhaps the most critical constraint in an example-based learning problem. Because a learner essentially extracts information about its task from training examples, a task may not be learnable at all if there are too few examples available, and hence too little information to be extracted. In certain real world scenarios, it may not be practical to collect a sufficiently large number of training examples due to sampling costs and other constraints. Sometimes, there may not even be any new examples available at all. Virtual example generation is thus a very powerful idea, which in extreme scenarios, could even make an otherwise impractical problem learnable. Because the idea has already been carefully developed in several recent papers [73] [10] [11], this thesis will only describe briefly how we have used the idea in our sample pattern detection applications.

1.4.4 Selecting Useful Examples

In example-based learning, how well a learner eventually turns out depends heavily on the quality of training examples it receives. Ideally, one would like to have available as large a set of training examples as possible, in order to guarantee a dense and comprehensive sampling of the input space. Unfortunately, there are real world constraints that can seriously limit the size of training databases.

One obvious constraint is that all real computer systems have only a finite amount of storage space for data. This means that even for moderately sized learning problems with an input feature space of a few tens of dimensions, it quickly becomes impossible to densely sample the input space for training examples. Another major limitation is the finite amount of time and computation resource available for training real systems. Most learning algorithms, including standard backpropagation and other gradient descent-based optimization schemes, converge in an amount of time that grows at least linearly in proportion to the number of training examples used. This quickly limits the number of training examples one can have in a real learning problem. To keep training databases tractably small while maintaining a representative sample of the input space, we look at ways of explicitly selecting only useful examples with high information value for training. Here, we are assuming that one can still generalize comparably well using only a reasonable amount of computation time and memory resource, by replacing a dense example database with a smaller but well distributed sample of training patterns.

Active learning is an area of research that has dealt with the question of how one can intelligently select useful examples for a learning problem. In active learning, a learner is allowed to pose *queries* to a teacher about how the target function behaves in specific regions of the input space. With the additional power of generating intelligent queries, one can expect active learning techniques to have faster learning rates and better approximation results than traditional example-based learning algorithms. The active learning idea extends beyond function approximation learning and has appeared in various forms throughout *knowledge engineering* and *machine learning* literature. In computational learning literature, the field also looks at the different types of queries that can be defined (see for example [6]), and how their learning convergence rates compare with traditional random sampling data selection techniques in a *probably approximately correct* (PAC) framework [98].

Our main interest is in connectionist and function approximation approaches toward active learning. Here, the focus has been on developing principled ways of sampling the input space for different classes of networks and approximation functions. Some existing ideas include: selective attention heuristics for training networks [4], boundary hunting schemes that generate queries near classification boundaries [47] [65] and Bayesean methods cast in an optimal experiment design framework [58] [24] [88] [86].

In our learning-based object detection approach, we shall look at how one can adapt these function approximation based active learning techniques, to sieve through extremely large training databases for useful examples relavant to the learning problem. We shall devote an entire chapter of this thesis to formulate and discuss the active example selection idea in depth. Active example selection affects the tractability of learning problems. Although most *learning* researchers may agree that *tractability* is usually less of a concern than *learnability* in many situations, we shall describe, in this thesis, a real learning scenario on human face detection, whereby the training data set can grow hopelessly large without reasonable example selection schemes. In fact, it should be clear from our particular human face detection scenario, that the example selection issue affects learning based approaches to other similar object and pattern detection tasks as well. Example selection is thus a very general and pertinent problem, if part of one's goal is to train working systems in reasonable amounts of time with limited memory and computation resource.

1.5 Thesis Outline and Contributions

The rest of this thesis describes and analyzes our learning-based technique for detecting spatially well-defined objects and pattern classes. We begin in Chapter 2 by presenting a human face detection system we have developed using this approach. Human faces make up a natural and challenging class of spatially well-defined 3D objects, and the problem of finding them is further complicated if one has to deal with varied lighting conditions. Here, our goal is twofold. We introduce the key elements of our object detection technique using a concrete example from a specific pattern detection application. We also quickly demonstrate the power of our approach by showing how well it performs on a reasonably complex real world problem.

Chapter 3 generalizes from our human face detection system and presents the underlying framework as a scheme for detecting spatially well-defined objects and pattern classes. We attempt to understand the overall approach in terms of its individual components, their functionality, limitations and underlying assumptions. Much of our analysis here will be based an effort to identify the critical components of our face detection system, which includes an empirical study on how the system's performance varies as one changes the architecture of the various components.

We also show three new applications of our pattern detection approach to demonstrate its generality and estensibility. The first is a direct extension of our human face detection system to handle a wider range of poses. We present a new but identically structured system, trained with additional examples covering a wider range of views. The second application detects a different class of spatially well-defined objects — human eyes, to show that the approach works well for more than just human faces. Although there is less pattern variability between human eyes than between human faces, the problem is still challenging because there can be a much wider range of isolated background patterns that resemble human eyes than human faces. The third application is somewhat different in spirit from the *detection* problems considered so far. We look at the problem of recognizing isolated hand-printed digits using our underlying object and pattern class identification approach. There has been a lot of work in hand-printed digit recognition over the past twenty to thirty years, with current state-of-art systems achieving recognition rates comparable to humans. Our goal in this third application is not an attempt to better the state-of-art performance in isolated hand-printed digit recognition systems. Rather, we wish to demonstrate that our underlying approach is truly general enough to model and capture localized pattern variations, even in a totally different problem domain, and on a task that is essentially pattern *recognition* and not *detection* in nature.

In Chapter 4, we take a formal look at one very critical aspect of our learning-based object and pattern class detection approach — the problem of selecting high utility examples for training a system. We argue that the example selection task is essentially an *active learning* problem, and we propose a function approximation based active learning formulation to show that one can indeed select useful training data in a principled and "optimal" fashion. While the formulation we propose is computationally intractable in its original form for a wide range of approximation function classes, we see it as a possible benchmark for evaluating other active example selection schemes. We then consider a reduced version of the original active learning formulation that essentially hunts for new data where approximation "error bars" are high. Furthermore, we show how such a scheme, with minor modifications, can lead to a practical "boot-strap" example selection strategy that we have adopted in our object detection training methodology. Although the "boot-strap" strategy loses some of the original active learning flavor, and may thus be "sub-optimal" in its choice of new examples, it is nevertheless a very effective means of sieving through unmanageably large sets of potential training data to make learning problems tractable.

Finally, in Chapter 5, we discuss two extensions to our object and pattern detection technique. The first looks at how one can combine the output results of several pattern detectors to achieve better detection rates with fewer false alarms. Recently, Rowley et. al. [76] have applied some simple arbitration techniques to a few face detection networks trained with our example selection methods, and have reported very impressive face detection results. We shall discuss about a more powerful arbitration scheme, called *network boosting* [30], that can potentially lead to systems with arbitrarily high correct classification rates. The second extension is about building hierarchical architectures for dealing with occlusion, and for detecting pattern classes with less well-defined boundaries. Recall from an earlier discussion that one can detect pattern classes with moderately variable boundaries, by defining simpler sub-pattern classes that can be easily isolated and identified in an image. One main difficulty in this approach is to reliably identify and locate full target patterns from the spatial distribution and composition of these sub-patterns in an image.

We shall look at some possible techniques for performing such a task.

The contributions of this thesis are as follows:

1. A new framework for detecting spatially well-defined objects and pattern classes with image variations that are difficult to parameterize. While most of the individual components within the proposed system architecture are not new, our work is the first attempt at integrating and understanding these separate components as parts of an overall framework for detecting spatially well-defined objects and image patterns.

2. An implementation of a very robust human face detection system, based on our proposed pattern detection scheme. At its time of conception, our system was probably the state of art implementation for correctly finding human faces with extremely few false alarm errors, even in highly cluttered images. We currently know of two later systems [61] [76] that are based on ideas developed in our face detection approach. Both systems have have also reported very impressive classification results.

3. A "boot-strap" paradigm for selecting useful training examples and for incrementally training object and pattern detection systems to arbitrary levels of robustness. Our object detection approach uses the "boot-strap" paradigm as part of its recommended example selection and training procedure. We have successfully demonstrated the paradigm in building our human face detection system, and have shown that the paradigm is vital for making an otherwise unmanageably complex learning problem tractable. The "boot-strap" idea is very general and is suitable for training most highly complex learning architectures and approximation function classes.

4. A highly robust 2-Value distance metric for measuring directionally dependent distances to a Gaussian mixture sample distribution model. The individual components of our 2-Value metric correspond to different classification measures that have been used recently. As far as we know, our work is the first attempt to combine the two measures for pattern classification. We also show how the 2-Value distance metric relates to classical measures like the Mahalanobis distance and probabilistic models in a Bayesean framework.

5. The idea of explicitly modeling the distribution of highly informative negative examples to create additional features for classification. We show empirically that a

well chosen negative example distribution of a learning problem gives rise to a very discriminative set of additional classification features for pattern detection. We also propose two possible interpretations of a negative example distribution in the context of building distribution-based models for representing large pattern classes.

6. An active learning formulation for example-based function approximation learning. We propose an optimality criterion for measuring the marginal utility of new data samples in a function approximation learning problem. We also derive a principled strategy for sampling new data in an "optimal" fashion based on our proposed utility measure. Finally, we show how simplifying the original formulation leads to practical example selection strategies like the "boot-strap" paradigm used by our object and pattern detection training approach.

Chapter 2

Learning an Object Detection Task — Human Face Detection

This chapter introduces our distribution-based modeling cum example-based learning scheme for detecting spatially well-defined objects and pattern classes in images. To help make our account concrete, we shall focus on a specific problem of finding human faces in cluttered scenes. We describe the construction of a generic human face detection system based on our proposed object and pattern detection technique. The system detects vertically oriented and unoccluded frontal views of human faces in grey-level images. It handles faces over a wide range of scales and works under different lighting conditions, even with moderately strong shadows. We stress again, however, that our goal here is to present a general approach for taking on spatially well-defined object and pattern detection tasks in multiple domains, including hand-printed character recognition, industrial inspection, medical image analysis and terrain classification. In Chapter 3, we complete the presentation by generalizing from this specific face detection example to highlight again the key components that make up our object and pattern detection scheme.

2.1 The Face Detection Problem

We begin by briefly defining the human face detection problem. Given as input an arbitrary image, which could be a digitized video signal or a scanned photograph, determine whether or not there are any human faces in the image, and if there are, return an encoding of the location and spatial extent of each human face in the image. An example encoding might be to fit each face in the image with a bounding box, and return the image co-ordinates of the box corners.

A closely related problem to face detection is *face recognition*. Given an input image of a face, compare the input face against models in a library of known faces and report if a match is found. In recent years, the face recognition problem has attracted much attention because of its many possible applications in automatic access control systems and human-computer interfaces.

Another very similar problem to face detection is *face localization*. In this problem, the input image contains exactly one human face, and the task is to determine the location and scale of the face, and sometimes also its pose. There are some existing face "detection" systems that have only been demonstrated on face *localization* tasks according to our definitions of detection and localization (see for example [61] and [55]). In many *localization* approaches, the key idea is simply to return the best matching location and scale between a face model and the input image. These approaches assume that in any image containing a face, the face region should match the face model best. We believe that face *detection* can be a much harder problem than face *localization*, because in the former, one must not only have a reasonable face model to describe faces, one also needs an absolute set of match thresholds to determine whether or not a given image pattern "resembles" the model enough to be labeled a face.

2.1.1 Motivation

Why is automatic face *detection* an interesting problem? Applications wise, face detection has direct relevance to the face recognition problem, because the first important step of a fully automatic human face recognizer is usually one of identifying and locating faces in an unknown image. So far, the focus of face recognition research has been mainly on distinguishing individual faces from others in a database. The task of finding faces in an arbitrary background is usually avoided by either hand segmenting the input image, or by capturing faces against a known uniform background.

Face detection also has potential applications in human-computer interfaces and surveillance systems. For example, a face finder can make workstations with cameras more user friendly by turning monitors on and keeping them active whenever there is someone in front of the terminal. In some security and census systems, one could determine the number of people in a scene by counting the number of visible human faces.

From the standpoint of this thesis, we are interested in face detection because faces make up a natural and challenging class of spatially well-defined image patterns for demonstrating and testing our object detection methodology. There are many other object classes and phenomena in the real world that share similar characteristics, for example different machine printed and handwritten styles of the character 'A', tumor anomalies in MRI scans and structural defects in manufactured parts. A successful methodology for finding faces should generalize well for other spatially well-defined pattern and feature detection problems.

2.1.2 Difficulty

Like most object detection problems, face detection is difficult because there can be significant face pattern variations that are hard to parameterize analytically. We have identified three common sources of pattern variations in face images:

1. **Differences in Facial Appearance and Expression.** Although most faces are similarly structured with the same facial features arranged in roughly the same spatial configuration, there can be significant shape and textural differences among faces. For the most part, these elements of variability are due to the basic differences in "facial appearance" between individuals — person 'A' has a larger nose than person 'B', person 'C' has eyes that are farther apart than person 'D', while person 'E' has a darker skin complexion than person 'F'. Even between images of the same person's face, there can still be significant geometrical or textural differences due to changes in expression and the presence or absence of facial makeup.

2. **Presence or Absence of Common Structural Features.** Face detection is also made difficult because certain common but significant features, such as glasses or a moustache, can either be present or totally absent from a face. Furthermore, these features, when present, can cloud out other basic facial features (eg. the glare in one's glasses may de-emphasize the darkness of one's eyes) and have a variable appearance themselves (eg. glasses come in many different designs). All this adds more variability to the range of permissible face patterns that a comprehensive face detection system must handle.

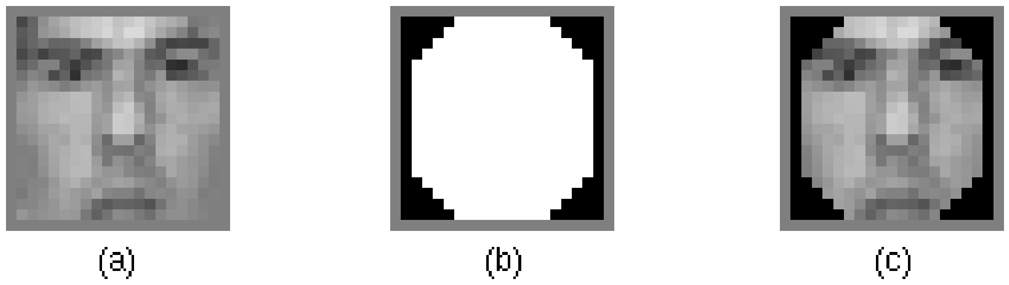


Figure 2-1: **(a):** A "canonical" face pattern. **(b):** A 19×19 mask for eliminating near-boundary pixels of canonical face patterns. **(c):** The resulting "canonical" face pattern after applying the mask.

3. **External Imaging Factors.** Face detection can be further complicated by unpredictable imaging conditions in an unconstrained environment. Because faces are essentially 3-dimensional structures, a change in light source distribution, for instance, can cast or remove significant shadows from a particular face, hence bringing about even more variability to 2D images of face patterns.

Clearly, one of the most critical issues in face detection is to devise a reliable scheme that can accurately account for the wide range of permissible variations in face patterns.

2.2 Approach and System Overview

In this section, we outline our approach for detecting faces in images. Faces are a highly structured class of image patterns that can be detected by examining only local image information from within a spatially well-defined boundary. We present an overview of a face detection system, based on a technique that represents and detects image patterns using only local image measurements.

The detection paradigm works by testing candidate image locations for local patterns that appear like faces. At the heart of the paradigm is a classification procedure that determines whether or not a given local image pattern is a face. Our approach formulates the classification problem as learning to identify faces from annotated examples of face and non-face patterns. Here, we use learning methods to help capture complex face pattern variations that may otherwise be difficult to parameterize by classical programming techniques.

2.2.1 Detecting Faces by Local Pattern Matching

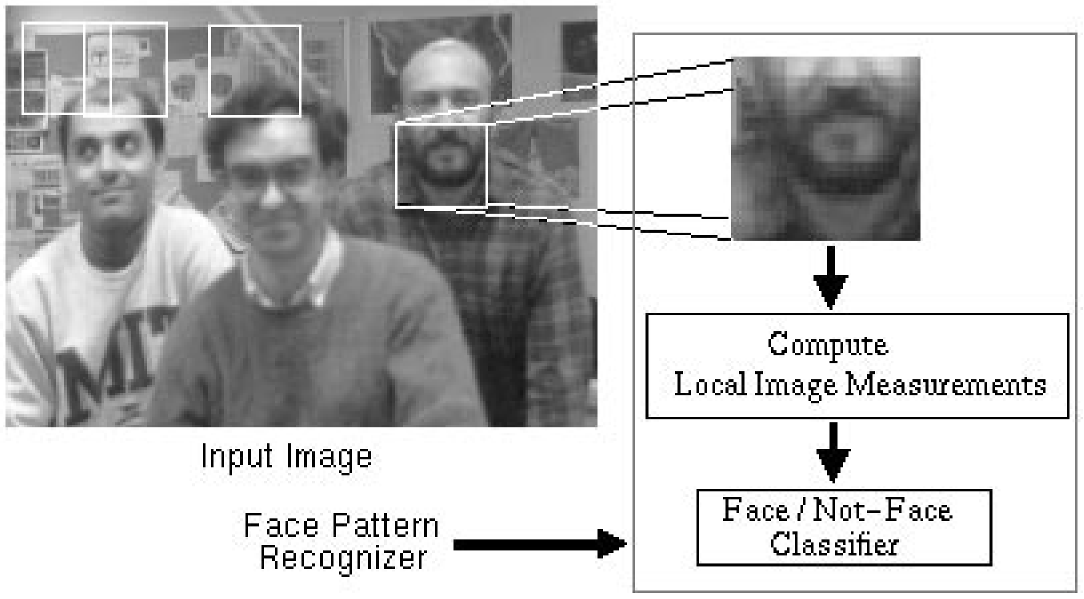


Figure 2-2: The system's task at each scale. The image is divided into many possibly overlapping windows. Each window pattern gets classified as either "a face" or "not a face", based on a set of local image measurements.

Human faces are a highly structured class of objects, with the same key features geometrically arranged in roughly the same fashion. One can treat human faces as a target class of spatially well-defined patterns with very stable boundaries in the image domain. To detect faces, one can therefore define a fixed shaped semantically stable "canonical" face notion in the image domain, and use a "template-like" matching paradigm to search for similar face-like patterns in an image. Figure 2-1(a) shows the canonical face structure used by our approach. It corresponds to a square portion of the human face whose upper boundary lies just above the eyes and whose lower boundary falls just below the mouth. The face detection task thus becomes one of appropriately representing the class of all such "face-like" image patches, and finding instances of these patterns in a scene.

The overall search paradigm for faces works as follows: We exhaustively scan an image for these "face-like" window patterns at all image locations over a range of scales. Figure 2-2 depicts the system's task at one fixed scale. The image is divided into multiple, possibly overlapping sub-images of the current window size. At each window, the system attempts to classify the enclosed image pattern as being either "a face" or "not a face". Each time a "matching" window pattern is found, the system reports a face at the window location, and also returns the scale as given by the current window size. We handle multiple scales

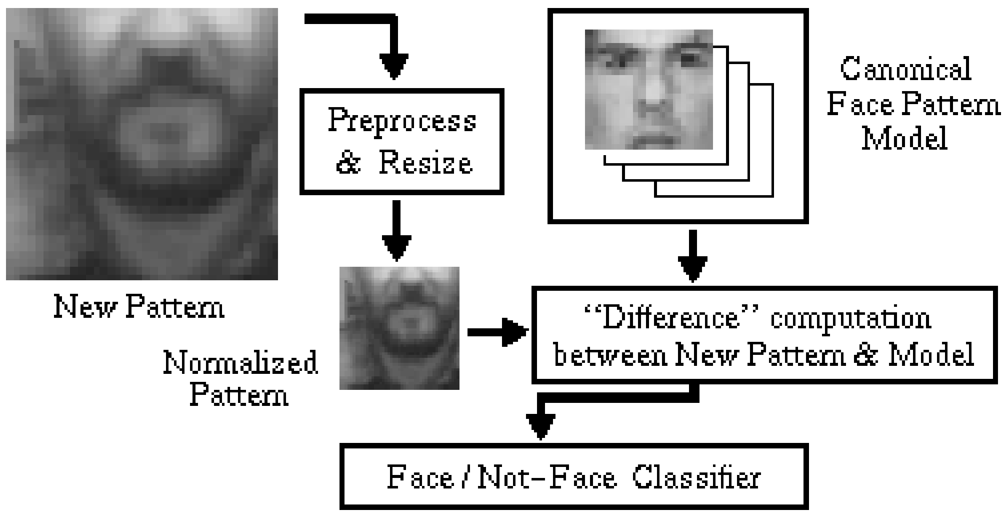

New Pattern
Preprocess & Resize
Normalized Pattern
Canonical Face Pattern Model
"Difference" computation between New Pattern & Model
Face / Not-Face Classifier

Figure 2-3: The key components of the *face pattern identification* procedure in greater detail. The face pattern identification procedure classifies new patterns as "faces" or "non-faces". The algorithm uses a distribution-based model to represent the space of all possible canonical face patterns. For each new pattern to be classified, it computes a set of "difference" measurements between the new pattern and the canonical face model. A trained classifier identifies the new pattern as being either "a face" or "not a face", based on the set of "difference" measurements.

by testing window patterns of different sizes for these "face-like" properties. The actual search procedure works by matching a *pyramidal* representation of the image with a fixed size "template". To detect faces at a larger scale than the "template" size, our system first resizes the input image by sub-sampling, so that the desired scale corresponds to the fixed "template" dimensions before searching through the image for matches.

2.2.2 The Face Classification Procedure

Clearly, the most critical and difficult part of our approach is the algorithm for identifying window patterns as "faces" or "non-faces". A good identification procedure must not only correctly label all valid face patterns as faces. It must also reject all background window patterns as non-faces. The task becomes especially complex if one has to deal with both a wide variety of faces and background patterns.

The rest of this chapter focuses on the identification procedure which makes up the crux of our detection scheme for spatially well-defined objects and pattern classes. Figure 2-3 shows the key components of the procedure. Basically, the approach is one of appropriately modeling the distribution of canonical face patterns in a reasonably chosen image feature space, and learning a functional mapping of input feature measurements to output classes from a representative set of "face" and "non-face" window patterns. More specifically, our approach works as follows:

1. **Choosing an appropriate feature space for representing and detecting faces.** Because face patterns are relatively stable in the image domain, we choose to represent and detect faces directly in a normalized and appropriately masked 19×19 pixel image feature space (see Figure 2-1). All window patterns of different dimensions must first be re-scaled to this size and masked before further processing. Matching with a fixed sized window simplifies our algorithm because it allows us to use the same classification procedure for all scales. The masking operation applies some prior domain knowledge to the matching problem by ignoring certain near-boundary pixels that may fall outside the spatial boundaries of a face.

2. **Building a distribution-based model to represent faces.** Using a database of canonical "face" window patterns and a similar database of "non-face" window patterns, we construct a distribution-based model of canonical face patterns in the masked 19×19 dimensional normalized image vector space. Our modeling scheme uses a few "face" pattern prototypes to piece-wise approximate the distribution manifold of canonical face patterns in the masked 19×19 dimensional image feature space. It also uses a few "non-face" pattern prototypes to help capture the concavities in the distribution manifold more accurately. Together, these "face" and "non-face" pattern prototypes serve as a distribution-based model for the class of canonical face views. The "face" pattern prototypes are synthesized offline by performing clustering on the example database of "face" window patterns, and the "non-face" prototypes are similarly synthesized from the database of "non-face" window patterns. The prototypes are hard-wired into the face detection system at compile time. Each prototype is a multi-dimensional Gaussian cluster with a centroid location and a covariance matrix that describes the local data distribution around the centroid.

3. **Defining a set of measurements for comparing new patterns with the distribution-based face model.** For each new window pattern to be classified, we compute a set of image measurements as input to a "face" or "non-face" decision procedure. Each set of image measurements is a vector of distances from the new window pattern to the window pattern prototypes in the masked 19×19 pixel

image feature space. To compute our vector of distances, we define and use a new "Mahalanobis-like" distance metric for measuring the distance between the input window pattern and each prototype center. The distance metric takes into account the shape of each prototype cluster, in order to penalize distances orthogonal to the local data distribution. The resulting vector of distances coarsely encodes the new window pattern's location relative to the overall distribution of canonical face patterns in the image feature space. It serves as a notion of "difference" between the new window pattern and the class of all canonical face patterns.

4. **Learning a set of classification thresholds for identifying face patterns.** We train a *multi-layer perceptron* (MLP) net to identify new window patterns as "faces" or "non-faces" based on their vector of distance measurements to the window pattern prototypes. When trained, the multi-layer perceptron net takes as input a vector of distance measurements and outputs a '1' if the vector arises from a face pattern, and a '0' if otherwise.

The remaining sections in this chapter will look at the window pattern identification approach in greater detail.

2.2.3 More Efficient Search Strategies

A final remark about our overall face detection approach: We see a "template" based matching approach like ours as having two highly important but nevertheless very independent parts. The first is an algorithm for identifying window patterns as "faces" or "non-faces", which is the main focus of this thesis and our proposed object and pattern detection approach. This component is critical for generating correct face detection results. The second is a search strategy that uses the window pattern identification scheme to look for faces in images. This component is important primarily for efficiency reasons.

For the second component, our current implementation uses a very naive search paradigm that exhaustively scans an image for faces over all possible locations and scales. Computational efficiency aside, our exhaustive search paradigm is an excellent test framework for the window identification procedure. Basically, the results we get from exhaustive testing show the outcome of applying the identification test at all locations and scales in an input image. To obtain good detection results, the test procedure must not only correctly label face patterns; it must also correctly reject a very wide variety of non-face background patterns.

A more efficient search paradigm will be vital for building real applications with limited computation resource. For the purpose of this thesis however, we shall not dwell any deeper into the issue of smarter search strategies, other than to highlight some existing ideas used in similar object and pattern detection problems. Suffice to say, our work will only benefit from smarter search paradigms in terms of producing the same correct results with less computation, as long as these "smarter" paradigms only reduce the number of irrelevant background patterns to consider, without wrongly discarding any face patterns.

We now look at two main approaches for implementing smarter object detection search strategies. Both approaches act as *focus of attention* mechanisms that quickly highlight image locations likely to contain the target object, hence reducing the total amount of irrelevant search. Clearly, these techniques would only be useful if they are also computationally inexpensive. In particular, the amount of overhead they incur should be negligible when compared to the amount of computation they save by not performing exhaustive search.

1. **Data driven techniques.** In this class of approaches, one extracts simple structural or textural features from an input image, and uses their distribution or spatial arrangement to hypothesize likely image locations containing the target object. Some examples of work in this area can be found in classical geometric model-based *object recognition and localization* literature, such as the *interpretation tree* based algorithms by Grimson and Lozano-Pérez [39] [38] [41] [42]. To detect objects, these algorithms use edge-based features to quickly generate a small number of likely image locations containing the object, before applying more expensive test procedures to verify their presence. Reisfeld et. al. [75] have used cheap local symmetry measurements to focus on image regions likely to contain faces. Yang and Huang [103] have a multi-staged face finding approach in which the first stage quickly isolates smooth and compact image regions as candidate locations for further testing. Where color information is available, one can also limit search to image regions whose colors are consistent with faces.

2. **Gradient descent techniques.** These techniques sample an image on a sparse grid and look for better matches by performing *gradient descent* in the local search space if

the initial match is reasonable. Betke and Makris [9] have demonstrated a stochastic version of this approach on locating traffic signs in noisy images. Gradient descent techniques are useful only if their match procedures have large *capture zones* in the search space.

When dealing with image sequences in a stationary background, one can take advantage of additional simplifying assumptions to reduce search, such as using motion cues to infer the location and scale of faces in an image.

2.3 Defining a Stable Feature Space

We now examine the key components of our window pattern identification procedure. The first step of desigining a window pattern identification procedure is to define an appropriate feature space for representing faces and performing matches. Ideally, we want a feature space in which the target pattern class (i.e. the class of "canonical" face patterns) has a continuous and smoothly varying distribution. As we shall see later, our entire pattern detection approach assumes strongly that we are indeed working in such a feature space.

Because faces are highly structured in the image domain, and face patterns with minor spatial or grey-level variations still look like valid face patterns, we choose a *view-based* feature space to model and detect faces. Our view-based feature space is a space of appropriately *masked* and *normalized* 19×19 pixel image patterns. Masking eliminates a fixed set of pixels from each 19×19 pattern. Each unmasked pixel can thus be viewed as an independent vector dimension, and the dimensionality of the full feature space equals the number of unmasked pixels. Normalization compensates for certain sources of image variation that do not affect the identity of window patterns. It reduces that range of possible window patterns the subsequent stages have to consider, thus making the window pattern identification task easier.

The following sequence of image operations maps an arbitrarily sized square window pattern into our view-based feature space:

1. **Window Resizing.** We use a fixed sized 19×19 view-based feature space to represent and identify face patterns. This operation re-scales square window patterns of different sizes to 19×19 pixels. We choose a 19×19 standard window size to keep the

dimensionality of the resulting feature space manageably small, but also large enough to preserve structural details that distinguish face patterns from non-face patterns.

2. **Masking.** We use the 19×19 binary pixel mask in Figure 2-1(b) to eliminate some near-boundary pixels of each window pattern. For "face" patterns, these masked pixels usually correspond to background pixels irrelevant to the description of a face (see Figure 2-1(c)). Eliminating these pixels ensures that our subsequent modeling scheme does not wrongly encode any unwanted background structure in our canonical face representation. It also reduces the dimensionality of our image feature space, which helps to make the modeling and face pattern identification tasks a little more tractable.

3. **Illumination gradient correction:** This is a normalization operation that subtract a best-fit brightness plane from the unmasked window pixels. For face patterns, it does a fair job at reducing the strength of heavy shadows caused by extreme lighting angles.

4. **Histogram equalization:** This is another normalization operation that adjusts for several geometry independent sources of window pattern variation. Some effects it accounts for include changes in illumination brightness and differences in camera response curves.

2.4 A Distribution-based Face Model

Our window classification algorithm identifies faces by transforming new window patterns into our view-based feature space and comparing the transformed window patterns with a canonical face model. A window pattern is declared a "face" if the match is good. This section describes our canonical face model. We use a distribution-based modeling scheme that tries to represent frontal faces as the set of all masked and normalized 19×19 pixel patterns that are canonical face views. More specifically, our distribution-based modeling scheme works as follows: Recall that we are operating in a view-based feature space of masked and normalized 19×19 pixel patterns. Suppose we treat each 19×19 image pattern as a point in our view-based feature space, then the set of all 19×19 pixel canonical face patterns maps to a fixed region in this space. So, in theory, one can model the class of all canonical face views by identifying the portion of this multi-dimensional feature space that corresponds to canonical face patterns, and representing the region in some tractable fashion.

2.4.1 Identifying the Canonical Face Manifold

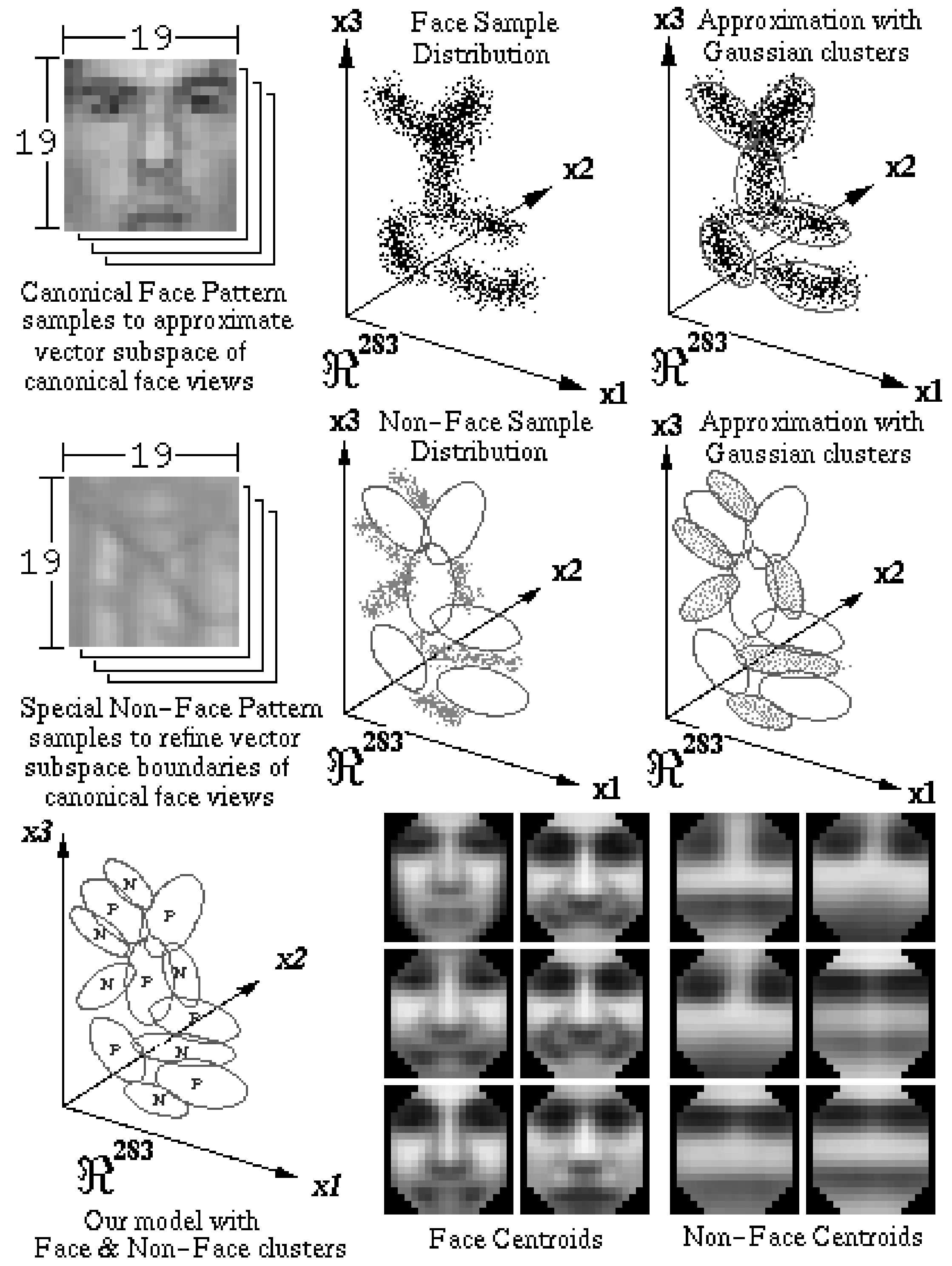
In practice, one does not have the set of all 19×19 pixel canonical face patterns to recover the sub-space of canonical face views exactly. Figure 2-4 explains how we localize and represent the region of canonical face patterns with limited data. Basically, we use a reasonably large example database of canonical face patterns and a carefully chosen database of non-face patterns to infer the spatial extent of canonical face views in the multi-dimensional view-based feature space. We assume that our "face" database contains a sufficiently representative sample of canonical face patterns that adequately populates the actual vector sub-space of canonical face views. So by building a model of the "face" sample distribution, one can still obtain a coarse but fairly reliable representation of the actual canonical face manifold.

Our "non-face" data samples are specially selected patterns that lie near the boundaries of the canonical face manifold. We use the "non-face" patterns to help localize and refine the boundaries of the canonical face manifold by explicitly carving out regions around the "face" sample distribution that do not correspond to canonical face views. We shall explain how we synthesize our special database of "non-face" patterns in Section 2.6.2.

2.4.2 Representing the Face Distribution

A Single Linear Sub-Space Representation — A Poor Model

One can model the "face" pattern distribution by fitting the face data sample with a *single* multi-dimensional Gaussian cluster, consisting of a centroid location and a full covariance matrix. The *view-based eigen-space* approach to face detection by Pentland et. al. [69] is a special case of this modeling technique. The *eigen-space* approach assumes that all face patterns occupy a low dimensional *linear* sub-space in the 19×19 pixel view-based feature space, and penalizes test patterns according to their Euclidean distance to the linear sub-space. This linear sub-space description is equivalent to a single multi-dimensional Gaussian cluster approximation with infinitely large eigenvalues in a few eigenvector di-



19
19
Canonical Face Pattern samples to approximate vector subspace of canonical face views
x3
Face Sample Distribution
x2
x1
$\Re^{283}$
Approximation with Gaussian clusters
Special Non-Face Pattern samples to refine vector subspace boundaries of canonical face views
Non-Face Sample Distribution
Our model with Face & Non-Face clusters
P
N
Face Centroids
Non-Face Centroids

Figure 2-4: Our distribution-based canonical face model. **Top Row:** We use a representative sample of canonical face patterns to approximate the volume of canonical face views in a *masked* 19×19 pixel image vector space. We model the "face" sample distribution with 6 multi-dimensional Gaussian clusters. **Center Row:** We use a selection of non-face patterns to help refine the boundaries of our Gaussian mixture approximation. We model the "non-face" sample distribution with 6 Gaussian clusters. **Bottom Row:** Our final model consists of 6 "face" clusters and 6 "non-face" clusters. Each cluster is defined by a centroid and a covariance matrix. The 12 centroids are shown on the right. **Note:** All the distribution plots are fictitious and are shown only to help with our explanation. The 12 centroids are real.

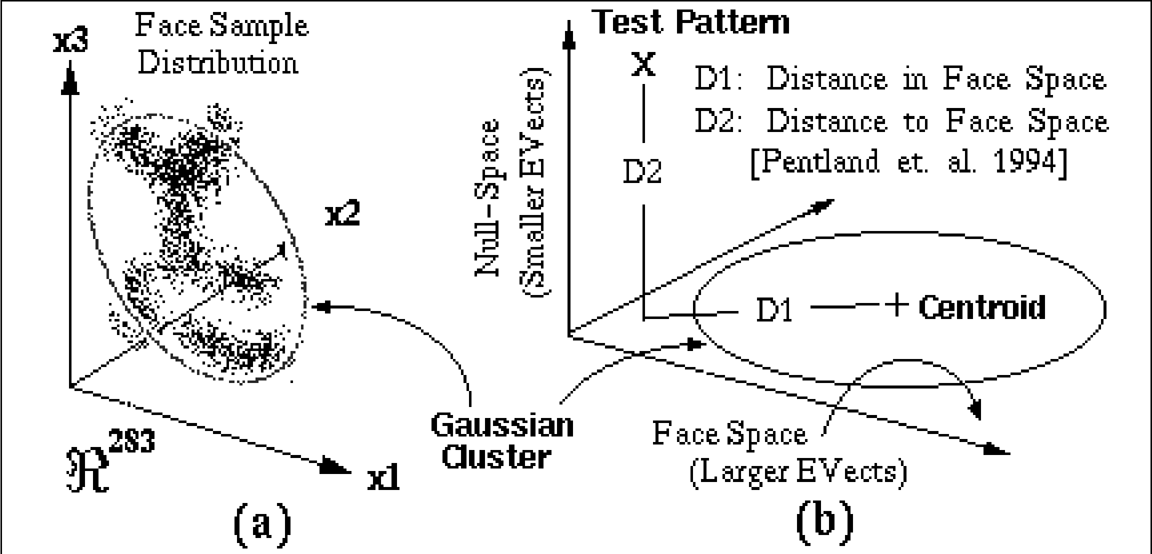


Figure 2-5: **(a):** An illustration to show that a *single* Gaussian cluster can be a very poor representation for an arbitrarily shaped "face" pattern distribution. **(b):** The two distance components we use in our scatter plots. The first component $\mathcal{D}_1$ is a distribution dependent distance between a test pattern and the multi-dimensional Gaussian centroid in a subspace of the cluster's larger eigenvectors. The second component $\mathcal{D}_2$ is the Euclidean distance between the test pattern and the subspace of larger eigenvectors.

rections (those spanning the linear sub-space of faces) and equal finite eigenvalues in the remaining eigenvector directions.

As illusrated by our hypothetical example in Figure 2-5(a), a *single* Gaussian cluster can be a very poor representation for the "face" pattern distribution if the actual distribution is not unimodal. We conducted the following experiment to show that this is indeed the case — i.e., a *single* Gaussian distribution poorly describes the space of canonical face views. Using a face sample of 4150 patterns, we modeled the "face" distribution as a single multi-dimensional Gaussian cluster. We then chose a small number of the cluster's largest eigenvectors as basis vectors for spanning a "face space", similar in spirit to the *eigen-space* approach for representing faces by Pentland et. al. [69]. For each face pattern in the sample, we resolved its displacement vector from the cluster centroid into two complementary components (see Figure 2-5(b)). The first component is a *Distance within Face Space* measure, described in [69]. This component is computed by projecting the face pattern onto the subspace of larger eigenvectors (i.e. the "face space"), and taking a distribution dependent distance between its projection and the cluster centroid. The second component is a *Distance from Face Space* measure, also descibed in [69]. We use a *Euclidean* distance

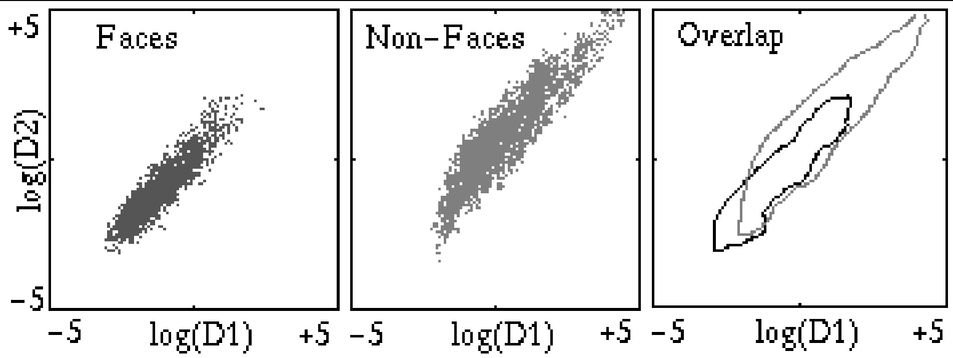


Figure 2-6: Scatter plots to show that a single Gaussian cluster approximation poorly describes the space of canonical face views.

between the face pattern and the subspace of larger eigenvectors. We also collected some non-face patterns and similarly resolved their displacement vectors from the "face" cluster centroid into the same two distance components.

Finally, we plotted the face and non-face sample distributions in a feature space of the two distance components. Figure 2-6 shows that there is a significant amount of overlap between the face and non-face distributions in this two distance feature space. The huge overlapping region suggests that a single Gaussian cluster poorly represents the face distribution, because one cannot separate the face and non-face pattern classes well using simple feature measurements derived from the representation scheme. We repeated the experiment by varying the number of eigenvectors spanning the "face space". The distribution scatter plots were qualitatively very similar for all cases.

A Piecewise Smooth Gaussian Mixture Representation

Our approach approximates the "face" pattern distribution in a piecewise smooth fashion using a few multi-dimensional Gaussian clusters (6 in our case). This model is reasonable as long as the actual face pattern distribution is *locally* linear, even though its *global* shape may be arbitrarily complex. Qualitatively, one can view the six "face" clusters as a coarse distribution-based representation of the canonical face manifold. We also use six "non-face" clusters to help define boundaries in and around the manifold by carving out nearby regions in the vector space that do not correspond to face patterns. Each prototype cluster is a multi-dimensional Gaussian with a centroid location and a covariance matrix that describes the local data distribution.

We believe our piecewise smooth modeling scheme is reasonable because the actual face pattern manifold appears continuous and smoothly varying in our multi-dimensional image feature space. More often than not, a face pattern with minor spatial and/or grey-level perturbations still looks like another valid face pattern. Similarly, a non-face pattern with minor variations would most likely still appear as a non-face pattern. The piecewise smooth modeling scheme serves two important functions. First, it performs generalization by applying a prior smoothness assumption to the observed data sample distribution. This results in a stored data distribution function that is well defined even in regions of the image vector space where no data samples have been observed. Second, it serves as a tractable scheme for representing an arbitrary data distribution by means of a few Gaussian basis functions.

The bottom right image of Figure 2-4 shows the 12 cluster centroids in our canonical face model. The six "face" prototypes are synthesized by clustering the database of canonical face patterns, while the six "non-face" prototypes are similarly derived from the database of non-face patterns.

2.4.3 Modeling the Distribution of "Face" Patterns — Clustering for Positive Prototypes

We use a database of 4150 normalized canonical "face" patterns to synthesize 6 "face" pattern prototypes in our multi-dimensional image vector space. The database consists of 1067 real face patterns, obtained from several different image sources. We artificially enlarge the original database by adding to it slightly rotated versions of the original face patterns and their mirror images. This helps to ensure that our final database contains a reasonably dense and representative sample of canonical face patterns.

Each pattern prototype is a multi-dimensional Gaussian cluster with a centroid location and a covariance matrix that describes the local data distribution around the centroid. Our modeling scheme approximates the observed "face" sample distribution with only a small number of prototype clusters because the sample size is still very small compared to the image vector space dimensionality (4150 data samples in a 283 dimensional *masked* image feature space). Using a large number of prototype clusters could lead to overfitting the observed data distribution with poor generalization results. Our choice of 6 pattern prototypes is somewhat arbitrary. The system's performance, i.e. its face detection rate versus the number of false alarms, does not change much with slightly fewer or more pattern prototypes.

We use a modified version of the *k-means* clustering algorithm to compute 6 face pattern centroids and their cluster covariance matrices from the enlarged database of 4150 face patterns. Our clustering algorithm differs from the traditional *k-means* algorithm in that it fits directionally elongated (i.e. elliptical) Gaussian distributions to the data sample instead of isotropic Gaussian distributions. We adopt a non-isotropic mixture model because we believe the actual face data distribution may in fact be locally more elongated along certain vector space directions than others.

To implement our *elliptical k-means* clustering algorithm, we use an adaptively changing *normalized Mahalanobis* distance metric instead of a standard *Euclidean* distance metric to partition the data sample into clusters. The normalized Mahalanobis distance metric returns a directionally dependent distance value between a new pattern $\vec{x}$ (in column vector form) and a Gaussian cluster centroid $\vec{\mu}$ (also as a column vector). It has the form:

$$\mathcal{M}_n(\vec{x}, \vec{\mu}) = \frac{1}{2}(d \ln 2\pi + \ln |\Sigma| + (\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu})),$$

where Σ is the covariance matrix that encodes the cluster's shape and directions of elongation. The normalized Mahalanobis distance differs from the Euclidean distance in that it reduces the penalty of pattern differences along a cluster's major directions of data distribution. This penalty adjustment feature allows the clustering algorithm to form elliptical clusters instead of spherical clusters where there is a non-isotropic local data distribution. Section 2.5.2 explains the *normalized Mahalanobis* distance metric in greater detail.

The following is a crude outline of our *elliptical k-means* clustering algorithm:

1. Obtain k (6 in our case) initial pattern centers by performing vector quantization with *euclidean* distances on the enlarged face database. Divide the data set into k partitions (clusters) by assigning each data sample to the nearest pattern center in euclidean space.

2. Initialize the covariance matrices of all k clusters to be the identity matrix.

3. Re-compute pattern centers to be the centroids of the current data partitions.

4. Using the current set of k pattern centers and their cluster covariance matrices, re-compute data partitions by re-assigning each data sample to the nearest pattern center in *normalized Mahalanobis* distance space. If the data partitions remain unchanged or if the maximum number of *inner-loop* (i.e. Steps 3 and 4) iterations has been exceeded, proceed to **Step 5**. Otherwise, return to **Step 3**.

5. Re-compute the covariance matrices of all k clusters from their respective data partitions.

6. Using the current set of k pattern centers and their cluster covariance matrices, re-compute data partitions by re-assigning each data sample to the nearest pattern center in *normalized Mahalanobis* distance space. If the data partitions remain unchanged or if the maximum number of *outer-loop* (i.e. Steps 3 to 6) iterations has been exceeded, proceed to **Step 7**. Otherwise, return to **Step 3**.

7. Return the current set of k pattern centers and their cluster covariance matrices.

The inner loop (i.e. Steps 3 and 4) is analogous to the traditional *k-means* algorithm. Given a fixed distance metric, it finds a set of k pattern prototypes that stably partitions the sample data set. Our algorithm differs from the traditional *k-means* algorithm because of Steps 5 and 6 in the outer loop, where we try to iteratively refine and recover the cluster shapes as well.

2.4.4 Modeling the Distribution of "Non-Face" Patterns — Clustering for Negative Prototypes

There are many naturally occuring "non-face" patterns in the real world that look like faces when viewed in isolation. Figure 2-7 shows one such example. In our multi-dimensional image window feature space, these "face-like" patterns lie along the boundaries of the canonical face manifold. Because we are coarsely representing the canonical face manifold with 6 Gaussian clusters, some of these face-like patterns may even be located nearer the "face" cluster centroids than some real "face" patterns. This may give rise to misclassification problems, because in general, we expect the opposite to be true, i.e. face patterns should lie nearer the "face" cluster centroids than non-face patterns.

In order to avoid possible misclassification, we try to obtain a comprehensive sample of these face-like patterns and explicitly model their distribution using 6 "non-face" prototype

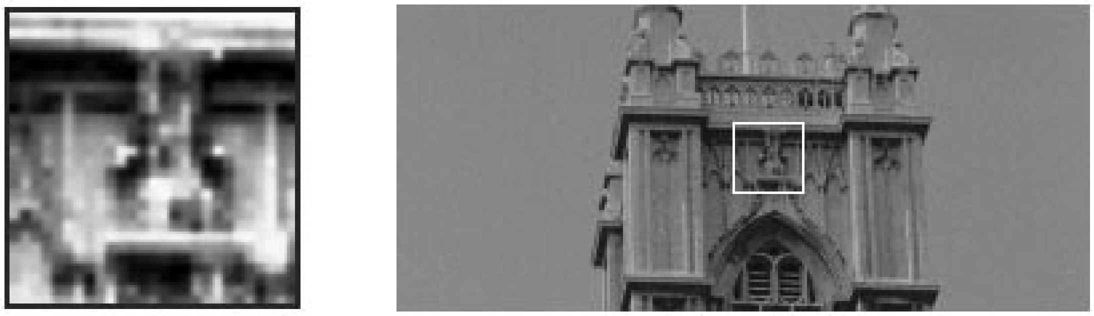


Figure 2-7: An example of a naturally occuring "non-face" pattern that resembles a face. **Left:** The pattern viewed in isolation. **Right:** The pattern viewed in the context of its environment.

clusters. These "non-face" clusters carve out negative regions around the "face" clusters that do not correspond to face patterns. Each time a new window pattern lies too close to a "non-face" prototype, we favor a "non-face" hypothesis even if the pattern also lies near a "face" prototype. Our choice of using exactly 6 pattern prototypes to model the "non-face" distribution is also somewhat arbitrary, as in the case of modeling the "face" pattern distribution. The system's face detection rate versus false alarm ratio does not change much with slightly fewer or more "non-face" pattern prototypes.

We use our *elliptical k-means* clustering algorithm to obtain 6 "non-face" prototypes and their cluster covariance matrices from a database of 6189 face-like patterns. The database was incrementally generated in a "boot-strap" fashion by first building a reduced version of our face detection system with only "face" prototypes, and collecting all the *false positive* patterns it detects over a large set of random images. Section 2.6.2 elaborates further on our "boot-strap" data generation technique.

2.4.5 Summary and Remarks

One of the key difficulties in face detection is to account for the wide range of permissible face pattern variations that cannot be adequately captured by geometric models, correlation templates and other classical parametric modeling techniques. Our approach addresses this problem by modeling the distribution of frontal face patterns directly in a fixed 19×19 pixel image vector space. We infer the actual distribution of face patterns in the image vector space from a sufficiently representative set of face views, and a carefully chosen set of non-face patterns. The distribution-based modeling scheme statistically encodes observable face pattern variations that cannot otherwise be easily parameterized by classical modeling

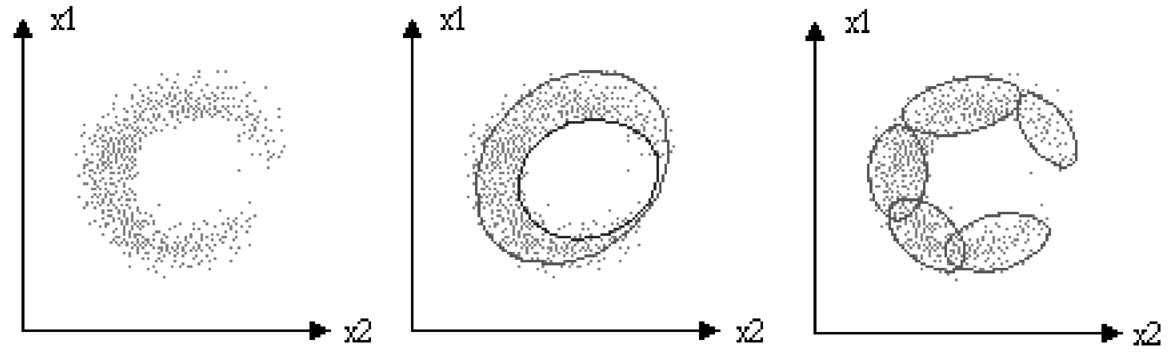


Figure 2-8: An illustration on how "negative" prototypes can help model concavities in a distribution with fewer Gaussian basis functions. **Left:** A hypothetical data distribution in 2 dimensions with a concave surface. **Center:** The distribution can be reasonably approximated with one *positive* Gaussian prototype encircling the distribution and one *negative* prototype carving out the concave region. **Right:** To model the same distribution with only positive prototypes, one has to use more Gaussian basis functions. The difference in number of basis functions can be a lot larger in a higher dimensional space.

techniques. To generalize from the slow varying nature of the face pattern distribution in the image vector space, and to construct a tractable model for the face pattern distribution, we interpolate and represent the observed distribution in a piecewise-smooth fashion using a few multi-dimensional Gaussian clusters.

Our final distribution-based model consists of 6 "face" clusters for coarsely approximating the canonical face pattern manifold in the image vector space, and 6 "non-face" clusters for representing the distribution of a specially selected "non-face" data sample. The non-face patterns come from non-face regions in the image vector space near the canonical face manifold. We propose two possibly related ways of reasoning about the 6 "non-face" pattern clusters:

1. The 6 "non-face" clusters explicitly model the distribution of face-like patterns that are not actual faces — i.e. "near miss" patterns that the face detection system should classify as "non-faces". They define and represent a separate "near-miss" pattern class in the multi-dimensional vector space, just like how the 6 "face" clusters encode a "canonical face" pattern class. Test patterns that fall under this "near-miss" class are classified as non-faces.

2. The 6 "non-face" clusters carve out regions in and around the "face" clusters that should not correspond to face patterns. They help shape the boundaries in our Gaussian mixture model of the canonical face manifold. More interestingly, they can also

help one model concavities in the canonical face manifold with fewer Gaussian basis functions (see Figure 2-8). This advantage of using "non-face" clusters becomes especially critical when one has too few face data samples to model a concave distribution region with a large number of Gaussian "face" clusters.

2.5 Matching Patterns with the Model

To detect faces, our system first matches each candidate window pattern in an input image against our distribution-based canonical face model. Each match returns a set of feature measurements that captures a notion of "similarity" or "difference" between the test pattern and the face model. At a later stage, a trained classifier determines, based on the set of feature measurements, whether or not the test pattern is a canonical frontal face view.

This section describes the set of feature measurements we compute for each new window pattern. Each set of measurements is a vector of 12 distances between the test window pattern and the model's 12 cluster centroids in our multi-dimensional image feature space (see Figure 2-9(a)). One way of interpreting our vector of distances is to treat each distance measurement as the test pattern's actual distance from some key reference location on or near the canonical face pattern manifold. The set of all 12 distances can then be viewed as a crude "difference" notion between the test pattern and the entire "canonical face" pattern class.

2.5.1 A 2-Value Distance Metric

We now define a new metric for measuring distance between a test pattern and a prototype cluster centroid. Ideally, we want a metric that returns small distances between face patterns and the "face" prototypes, and either large distances between non-face patterns and the "face" prototypes, or small distances between non-face patterns and the "non-face" centers. We propose one such measure that normalizes distances by taking into account both the local data distribution around a prototype centroid, and the reliability of the local distribution estimate. The measure scales up distances orthogonal to a cluster's major axes of elongation, and scales down distances along the cluster's major elongation directions. We argue that such a distance measure should meet our ideal goals reasonably well, because we expect most face patterns on the face manifold to lie along the major axes of one or more "face" clusters, and hence register small distances with one or more "face" prototypes. Similarly, we expect most non-face patterns to lie either far away from all the "face" clusters or along the major axes of one or more "non-face" clusters, and hence either register large distances with all the "face" prototypes or small distances with some "non-face" prototypes.

Our proposed distance measure consists of two output components (see Figure 2-9(b)). The first value is a *Mahalanobis-like* distance between the test pattern and the prototype centroid, defined within a lower-dimensional sub-space of our masked 19×19 pixel image vector space. The sub-space is spanned by the 75 largest eigenvectors of the prototype cluster. This distance component is directionally weighted to reflect the test pattern's location relative to the major elongation directions in the local data distribution around the prototype center. The second value is a normalized Euclidean distance between the test pattern and its projection in the lower-dimensional sub-space. This is a uniformly weighted distance component that accounts for pattern differences not included in the first component due to possible modeling inaccuracies. We elaborate further on the two components below.

2.5.2 The Normalized Mahalanobis Distance

We begin with a brief review of the *Mahalanobis* distance. Let $\vec{x}$ be a column vector test pattern, $\vec{\mu}$ be a column vector prototype centroid, and Σ be the covariance matrix describing the local data distribution near the centroid. The *Mahalanobis distance* between the test pattern and the prototype centroid is given by:

$$\mathcal{M}(\vec{x}, \vec{\mu}) = (\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu}).$$

Geometrically, the *Mahalanobis distance* can be interpreted as follows. If one models the prototype cluster with a best-fit multi-dimensional Gaussian distribution, then one gets a best-fit Gaussian distribution centered at $\vec{\mu}$ with covariance matrix Σ. All points at a given *Mahalanobis distance* from $\vec{\mu}$ occupy a constant density surface in this multi-dimensional vector space. This interpretation of the *Mahalanobis distance* leads to a closely related distance measure, which we call the *normalized Mahalanobis distance*, given by:

$$\mathcal{M}_n(\vec{x}, \vec{\mu}) = \frac{1}{2}(d \ln 2\pi + \ln |\Sigma| + (\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu})).$$

Here, d is the vector space dimensionality and $|\Sigma|$ means the determinant of Σ.

The *normalized Mahalanobis* distance is simply the negative natural logarithm of the best-fit Gaussian distribution described above. It is normalized in the sense that it originates directly from a probability density distribution that integrates to unity. Our modified *k-means* clustering algorithm in Section 2.4.3 uses the *normalized Mahalanobis* distance metric instead of the standard form because of stability reasons. Notice that for a given spatial displacement between a test pattern and a prototype centroid, the standard *Mahalanobis* distance tends to be smaller for long clusters with large covariance matrices than for small clusters. So, if one uses a standard *Mahalanobis* distance metric to perform clustering, the larger clusters will constantly increase in size and eventually overwhelm all the smaller clusters.

As a distance metric for establishing the class identity of test patterns, the *Mahalanobis distance* and its normalized form are both intuitively pleasing for the following reason: They both measure pattern differences in a distribution dependent manner, unlike the Euclidean distance which measures pattern differences in an absolute sense. More specifically, the distances they measure are indicative of how the test pattern ranks relative to the overall location of other known patterns in the pattern class. In this sense, they capture very well the notion of "similarity" or "dis-similarity" between a test pattern and a pattern class.

2.5.3 The First Distance Component — Distance within a Normalized Low-Dimensional Mahalanobis Subspace

As mentioned earlier, our distance metric consists of 2 output values as shown in Figure 2-9(b). The first value, $\mathcal{D}_1$, is a *Mahalanobis-like* distance between the test pattern and the prototype center. This distance is defined within a 75-dimensional sub-space of our masked 19×19 pixel image vector space, spanned by the 75 largest eigenvectors of the current prototype cluster. Geometrically, we can interpret the first distance value as follows: The test pattern is first projected onto the 75 dimensional vector sub-space. We then compute the normalized Mahalanobis distance between the test pattern's projection and the prototype center as the first output value. Like the standard Mahalanobis distance, this first value locates and characterizes the test pattern relative to the cluster's major directions of data distribution.

Mathematically, the first value is computed as follows: Let $\vec{x}$ be the column vector test

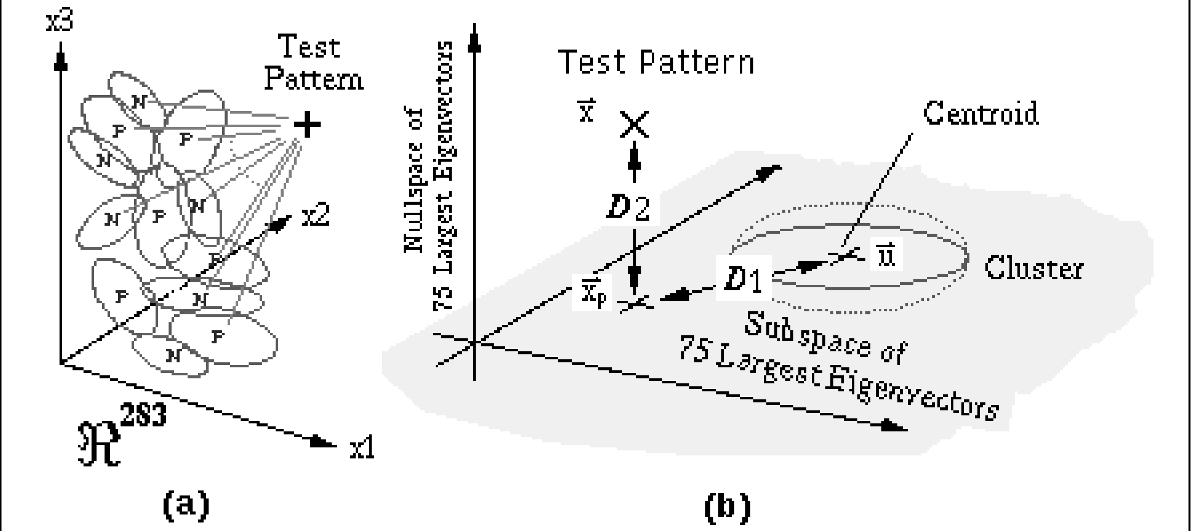


Figure 2-9: The measurements returned from matching a test pattern against our distribution-based model. **(a):** Each set of measurements is a vector of 12 distances between the test pattern and the model's 12 cluster centroids. **(b):** Each distance measurement between the test pattern and a cluster centroid is a 2-value distance metric. The first component is a Mahalanobis distance between the test pattern's projection and the cluster centroid in a subspace spanned by the cluster's 75 largest eigenvectors. The second component is the Euclidean distance between the test pattern and its projection in the subspace. So, the entire set of 12 distance measurements is a vector of 24 values.

pattern, $\vec{\mu}$ be the prototype pattern, E_{75} be a matrix with 75 columns, where column i is a unit vector in the direction of the cluster's i^{th} largest eigenvector, and W_{75} be a diagonal matrix of the corresponding 75 largest eigenvalues. The covariance matrix for the cluster's data distribution in the 75 dimensional subspace is given by $\Sigma_{75} = (E_{75}W_{75}E_{75}^T)$, and the first distance value is:

$$\mathcal{D}_1(\vec{x}, \vec{\mu}) = \frac{1}{2}(75 \ln 2\pi + \ln |\Sigma_{75}| + (\vec{x} - \vec{\mu})^T \Sigma_{75}^{-1} (\vec{x} - \vec{\mu})).$$

We emphasize again that this first value is not a "complete" distance measure in our masked 19×19 pixel image vector space, but rather a "partial" distance measure in a subspace of the current cluster's 75 most significant eigenvectors. The measure is not "complete" in the sense that it does not account for pattern differences in certain image vector space directions, namely differences orthogonal in direction to the cluster's 75 most significant eigenvectors. We omit the smaller eigenvectors in this directionally dependent distance measure because we believe that their corresponding eigenvalues may be significantly inaccurate due to the small number of data samples available to approximate each cluster. For instance, we have, on the average, fewer than 700 data points to approximate each "face"

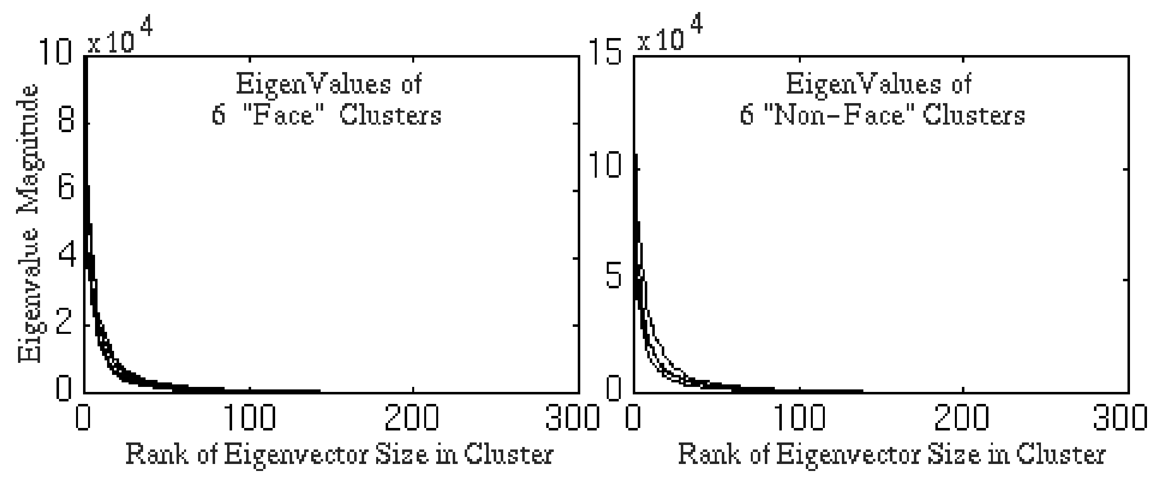


Figure 2-10: Graphs of eigenvalues in decreasing order of magnitude. **Left:** Decreasing eigenvalues for 6 "Face" clusters. Right : Decreasing eigenvalues for 6 "Non-Face" clusters.

cluster in a 283 dimensional masked image vector space. Using the smaller eigenvectors and eigenvalues to compute a distribution dependent distance can therefore easily lead to meaningless results.

Figure 2-10 plots the eigenvalues of our 12 prototype clusters in decreasing order of magnitude. Notice that for all 12 curves, only a small number of eigenvectors are significantly larger than the rest. We use the following criterion to arrive at 75 "significant" eigenvectors for each cluster: We eliminate from each cluster the maximum possible number of trailing eigenvectors, such that the sum of all eliminated eigenvalues is still smaller than the largest eigenvalue in the cluster. This criterion leaves us with approximately 75 eigenvectors for each cluster, which we standardize at 75 for the sake of simplicity.

2.5.4 The Second Distance Component — Distance from the Low-Dimensional Mahalanobis Subspace

The second output value of our distance metric is a standard Euclidean distance between the test pattern and its projection in the 75 dimensional sub-space. This distance component accounts for pattern differences not captured by the first component, namely pattern differences in the smaller eigenvector directions. Because we may not have a reasonable estimate of the smaller eigenvalues, we simply assume an isotropic Gaussian sample data distribution in the smaller eigenvector directions, and hence a Euclidean distance measure.

Using the notation from the previous sub-section, we can show that the second component is simply the L_2 norm of the displacement vector between $\vec{x}$ and its projection $\vec{x_p}$:

$$\mathcal{D}_2(\vec{x}, \vec{\mu}) = ||(\vec{x} - \vec{x_p})|| = ||(I - E_{75}E_{75}^T)(\vec{x} - \vec{\mu})||.$$

Notice that on its own, the second distance value is also not a "complete" distance measure in our masked 19×19 pixel image vector space. Specifically, it does not account for pattern differences in the subspace of the cluster's 75 most significant eigenvectors. It complements the first output value to form a "complete" distance measure that captures pattern differences in all directions of our masked 19×19 pixel image vector space.

2.5.5 Relationship between our 2-Value Distance and the Mahalanobis Distance

There is an interesting relationship between our 2-Value distance metric and the "complete" normalized Mahalanobis distance involving all 283 eigenvectors of a prototype cluster [1]. Recall from Section 2.5.2 that the Mahalanobis distance and its normalized form arise from fitting a multi-dimensional full-covariance Gaussian to a sample data distribution. For high dimensional vector spaces, modeling a distribution with a full-covariance Gaussian is often not feasible because one usually has too few data samples to recover the covariance matrix accurately. In a d dimensional vector space, each full-covariance Gaussian requires d parameters to define its centroid and another $d(d+1)/2$ more parameters to define its covariance matrix. For $d = 283$, this amounts to 40469 parameters!

One way of reducing the number of model parameters is to use a restricted Gaussian model with a diagonal covariance matrix, i.e., a multi-dimensional Gaussian distribution whose elongation axes are aligned to the vector space axes. In our domain of 19×19 pixel images, this corresponds to a model whose individual pixel values may have different variances, but whose pair-wise pixel values are all uncorrelated. Clearly, this is a very poor model for face patterns which are highly structured with groups of pixels having very highly correlated intensities.

An alternative way of simplifying the Gaussian model is to preserve only a small number of "significant" eigenvectors in the covariance matrix. One can show that a d dimensional

[1]See [45] for a similar interpretation and presentation. Our explanation here is adapted from theirs.

Gaussian with h principal components has a covariance matrix of only $h(2d - h + 1)/2$ free parameters. This can easily be a tractable number of model parameters if h is small. Geometrically, the operation corresponds to projecting the original d dimensional Gaussian cluster onto an h dimensional subspace, spanned by the cluster's h most "significant" elongation directions. The h dimensional subspace projection preserves the most prominent correlations between pixels in the target pattern class. To exploit these pixel-wise correlations for pattern classification, one computes a directionally weighted Mahalanobis distance between the test pattern's projection and the Gaussian centroid in this h dimensional subspace — $\mathcal{D}_1$ of our 2-Value distance metric.

The orthogonal subspace is spanned by the $d - h$ remaining eigenvectors of the original Gaussian cluster. Because this subspace encodes pixel correlations that are less prominent and possibly less reliable due to limited training data, we simply assume an isotropic Gaussian distribution of data samples in this subspace, i.e. a diagonal covariance matrix Gaussian with equal variances along the diagonal. The isotropic Gaussian distribution requires only 1 free parameter to describe its variance. To measure distances in this subspace of isotropic data distribution, we use a directionally independent Euclidean distance — $\mathcal{D}_2$ of our 2-Value distance metric.

We can thus view our 2-Value distance metric as a robust approximate Mahalanobis distance that one uses when there is insufficient training data to accurately recover all the eigenvectors and eigenvalues of a Gaussian model. The approximation degrades gracefully by using its limited degrees of freedom to capture the most prominent pixel correlations in the data distribution. Notice that as the data sample size increases, one can preserve a larger number of principal components in the Gaussian model. This results in $\mathcal{D}_1$ becoming increasingly like the "complete" Mahalanobis distance with $\mathcal{D}_2$ vanishing in size and importance.

2.5.6 Relationship to Probabilistic Models

Let λ_i be the i^{th} largest eigenvalue of an unsimplified d dimensional Gaussian model for a local data distribution, and let h be the number of *significant* eigenvectors preserved in the simplified model. The two components of our resulting 2-Value distance metric, $\mathcal{D}_1$ and $\mathcal{D}_2$, can be combined into a single "robust" Mahalanobis distance measure, $\mathcal{M}_h$, by taking their weighted sum as follows:

$$\mathcal{M}_h = \mathcal{D}_1 + \frac{d-h}{\sum_{i=h+1}^{d} \lambda_i}\mathcal{D}_2. \qquad (2.1)$$

The weighting factor, $(d-h)/(\sum_{i=h+1}^{d} \lambda_i)$, can be derived by assuming an isotropic Gaussian distribution in the orthogonal subspace, spanned by the remaining $d-h$ less significant eigenvectors. This isotropic Gaussian distribution has a "regularized " variance whose magnitude is simply the mean of the corresponding $d-h$ smaller eigenvalues. One can also argue, from an information theoretic point of view, that the weighting factor given above is an optimal choice that minimizes a Kullback-Leibler divergence cost function between the unsymplified Gaussian model and its robust estimate [61].

The "robust" Mahalanobis distance, $\mathcal{M}_h$, gives rise to a convenient probabilistic interpretation for our 2-Value distance metric. Recall from Section 2.5.2 that the *standard Mahalanobis* distance is simply the negative natural logarithm of the best-fit full covariance Gaussian density approximation for a sample data distribution. The robust Mahalanobis distance is thus the negative natural logarithm of a best-fit simplified Gaussian approximation for the same data distribution.

Let $\mathcal{A}$ be the object sub-class that generates the local data distribution modeled by our best-fit Gaussian approximation. Given a new window pattern $\vec{x}$, the best-fit Gaussian distribution function computes the likelihood of pattern $\vec{x}$ arising from sub-class $calA$, i.e. $P(\vec{x}|\mathcal{A})$. Our 2-Value distance metric, when expressed as a single value *robust Mahalanobis* distance, can thus be viewed as a negative logarithmic likelihood measure of the test pattern arising from the local pattern class represented by the underlying Gaussian model.

In a very recent demonstration of a probabilistic learning-based face *localization* technique, Moghaddam and Pentland [61] discusses about an optimal estimate for high dimensional Gaussian densities, whose form is the product of two lower-dimensional and independent Gaussian densities. This estimate is exactly a re-expression of Equation 2.1 as a probabilistic likelihood measure.

2.5.7 Relationship between our 2-Value Distance and some PCA-based Classical Distance Measures

Our 2-Value distance metric is also closely related to the classical distance measures used by *principal components analysis* (PCA, also called Karhunen-Loeve expansion) based clas-

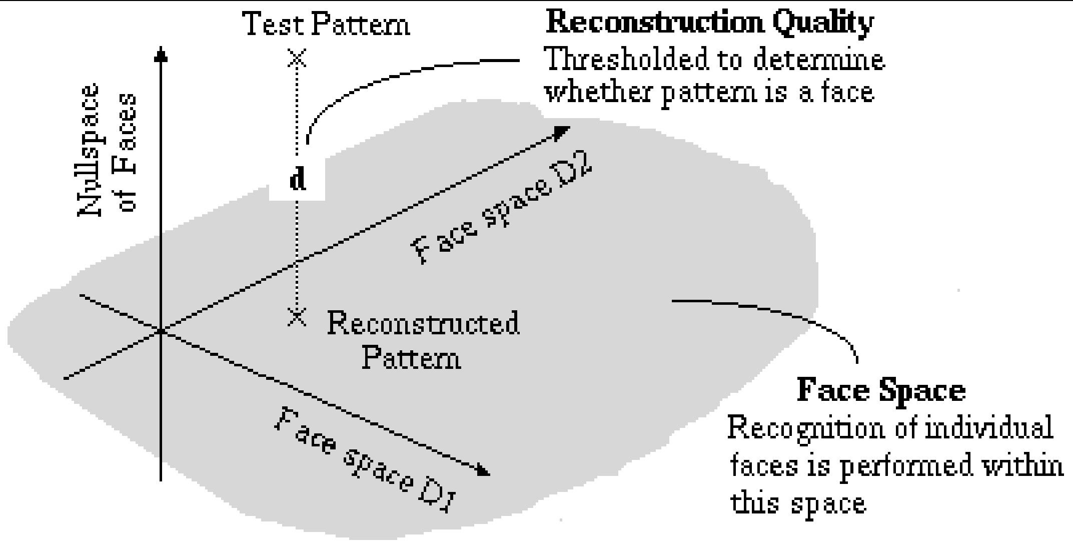

Test Pattern
Reconstruction Quality
Thresholded to determine whether pattern is a face
Nullspace of Faces
d
Face space D2
Reconstructed Pattern
Face space D1
Face Space
Recognition of individual faces is performed within this space

Figure 2-11: The eigen-image model of faces by Turk and Pentland [96] [69]. The approach represents the space of all face views (i.e. the "Face Space") as a linear combination of several orthonormal eigen-images. The eigen-images are computed by performing PCA on an example database of face patterns. The modeling scheme assumes that all face views lie within the "face space", and each face view corresponds to a set of coefficients that describes its appearance as a linear combination of the eigen-images. One can use the coefficients as features to recognize faces of different individuals. One can also determine whether or not a test pattern is a face by computing and thresholding d, a measure of how badly the eigen-images reconstruct the test pattern.

sification schemes [32] [35] [93]. We examine two such distance measures below.

A standard way of modeling a 3D object for pattern recognition is to represent its space of 2D views using a few basis images, obtained by performing *principal components analysis* on a comprehensive set of 2D views. The modeling approach assumes that all possible 2D views of the target object lie within a low dimensional sub-space of the original high dimensional image space. The low dimensional sub-space is linearly spanned by the principal components. Each new view of the target object can be represented by a set of coefficients that describes the view as a linearly weighted superposition of the principal components.

Kirby and Sirovich [85] [51] have used principal components analysis methods to build low-dimensional models for characterizing the space of human faces. As a representative recent example, let us consider Turk and Pentland's application of PCA techniques to face recognition [96] and facial feature detection [69]. To optimally represent their "face space", Turk and Pentland compute an orthonormal set of eigen-images from an example database of face patterns to span the space. New face views are represented by projecting their image patterns onto the set of eigen-images to obtain their linear combination coefficients. Because the technique assumes that all face patterns actually lie within the "face space", one can use the set of linear combination coefficients as features to parameterize and recognize faces of different individuals (see Figure 2-11). Similarly, one can determine whether or not a given pattern is a face by measuring how well or poorly the eigen-images reconstruct the pattern — i.e., the pattern's distance (d in Figure 2-11) from the "face space" [69].

In another recent application of classical techniques, Burl et. al. [22] have used a different PCA based approach for a terrain classification task on finding volcanos in SAR images of Venus. Their system builds a low dimensional view-based volcano model, parameterized by a small number of the largest principal component vectors from an example set of volcano images (see Figure 2-12). Within this low-dimensional model space, Burl et. al. further observe that not all reconstructible patterns are actual volcano views; i.e., not all linear combinations of principal component images are volcano patterns. Thus, to detect volcanos, they use the set of linear combination coefficients as classification features, and learn from examples the range of coefficient values, i.e. distances within the volcano model space, that actually correspond to volcano patterns.

One can relate our 2-Value distance metric to the above PCA based classification techniques as follows: Suppose we treat each Gaussian cluster in our distribution-based model as locally representing a sub-class of "face" or "non-face" patterns, then the cluster's 75 largest eigenvectors can be viewed as a set of basis images that linearly span the sub-class. Each new pattern in the sub-class corresponds to a set of 75 coefficients in the model space.

Our distance component $\mathcal{D}_2$ is the Euclidean distance between a test pattern and the cluster's 75 largest eigenvector subspace. Assuming that the 75 eigenvectors are the "basis views" that span our target sub-class, this distance component measures how poorly the basis views reconstruct the test pattern, and hence how "different" the test pattern is relative to the entire target sub-class. This distance is analogous to Turk and Pentland's *distance to "face space"* metric (i.e. d in Figure 2-11), which quantifies the "difference" between a test pattern and the class of face views.

The distance component $\mathcal{D}_1$ is a Mahalanobis distance between the cluster centroid and a test pattern's projection in the 75 largest eigenvector subspace. Here, we implicitly assume, like Burl et. al., that not all reconstructible patterns in our 75 eigenvector model space are views of the target sub-class, and that all views of the target sub-class are located within a certain Mahalanobis distance from the cluster centroid. This constrains the range

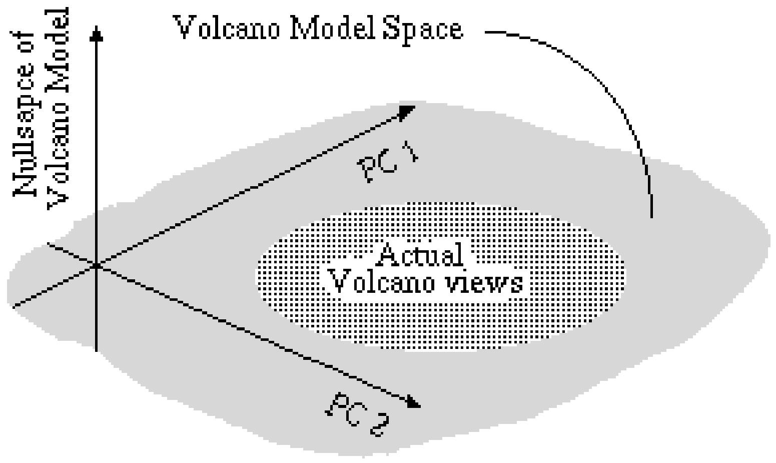


Figure 2-12: The view-based model of volcanos by Burl et. al. [22]. The approach assumes that the set of all volcano patterns lies within a low dimensional model space, spanned by a small number of the largest principal component vectors from a volcano image database. Each pattern in the model space corresponds to a set of coefficients that describes the pattern's appearance as a linear combination of the principal component images. Not all linear combinations of principal component images are volcano patterns, so one can detect volcanos by learning from examples the range of coefficient values that corresponds to volcanos.

of model coefficients that correspond to actual target patterns, just like how Burl et. al. constrain by learning their range of model parameters that correspond to actual volcano views.

Thus, our $\mathcal{D}_1$ and $\mathcal{D}_2$ distance measurements correspond to different classical classification criteria used also recently. As far as we know, our work, as reported in an earlier paper [89], presents the first attempt at combining these two measures for pattern classification.

2.6 The Classifier

The classifier's task is to identify "face" test patterns from "non-face" patterns based on their match feature vectors of 12 distance measurements. Our approach treats the classification stage as one of learning a functional mapping from input feature distance vectors to output classes using a representative set of training examples.

2.6.1 A Multi-Layer Perceptron Classifier

We use a *Multi-Layer Perceptron* (MLP) net to perform the desired classification task. The net has 12 pairs of input terminals, one output unit and 24 hidden units. Each hidden and output unit computes a weighted sum of its input links and performs sigmoidal thresholding on its output. During classification, the net is given a vector of the current test pattern's

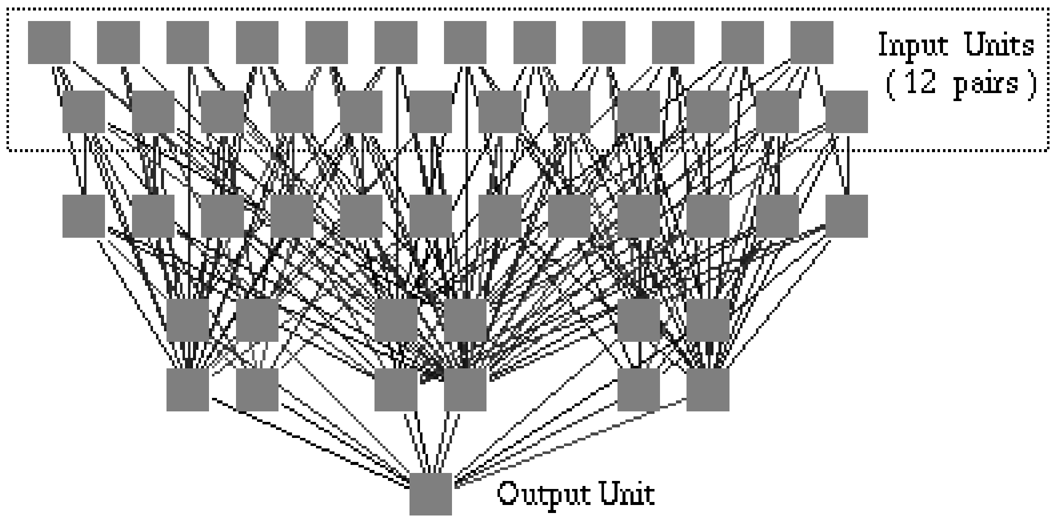
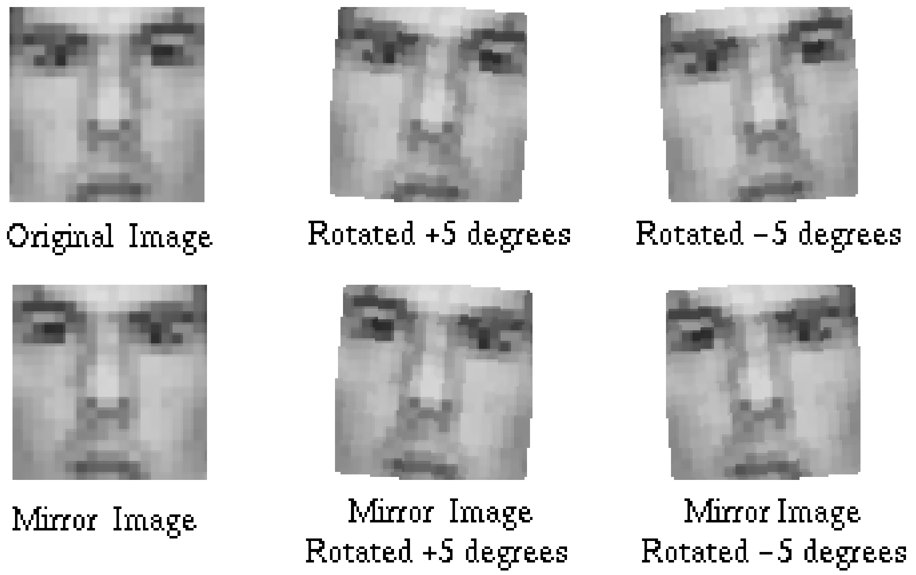


Figure 2-13: One of the multi-layer perceptron (MLP) net architectures we trained to identify face patterns from our vector of distance measurements. The net has 12 pairs of input units to accept the 12 pairs of distance values from our matching stage. When trained, the output unit returns a 1 for face patterns and a 0 otherwise. Notice that this net has one special layer of hidden units that combines each 2-Value distance pair into a single distance value. Our experiments show that the detailed net architecture is really not crucial in this application.

distance measurements to the 12 prototype centers. Each input terminal pair receives the distance values for a designated prototype pattern. The output unit returns a '1' if the input distance vector arises from a "face" pattern, and a '0' otherwise.

In our current system, the hidden units are partially connected to the input terminals and output unit as shown in Figure 2-13. The connections exploit some prior knowledge of the problem domain. Notice in particular that there is one layer of hidden units that combines each 2-Value distance pair into a single distance value. Our experiments in the next chapter will show that the number of hidden units and network connectivity structure do not significantly affect the classifier's performance.

We train our multi-layer perceptron classifier on feature distance vectors from a database of 47316 window patterns. There are 4150 positive examples of "face" patterns in the database and the rest are "non-face" patterns. The net is trained with a standard back-propagation learning algorithm [77] until the output error stabilizes at a very small value. For the particular multi-layer perceptron net architecture in Figure 2-13, we actually get a training output error of 0. Because we have many more positive and negative training examples than free parameters in our multi-layer perceptron net, we believe that this low training error rate is not a result of overfitting the training data. Rather, we believe it shows



Original Image
Rotated +5 degrees
Rotated −5 degrees
Mirror Image
Mirror Image Rotated +5 degrees
Mirror Image Rotated −5 degrees

Figure 2-14: Artificially generating *virtual examples* of face patterns. For each original face pattern in our training database, we can generate a few new face patterns using some simple image transformations. We artificially enlarge our face database in this fashion to obtain a more comprehensive sample of face patterns.

that our "difference" notion of 2-Value distances is a very discriminative set of classification features for separating face and non-face image patterns.

2.6.2 Generating and Selecting Training Examples

In many example-based learning applications, how well a learner eventually performs on its task depends heavily on the quality of examples it receives during training. An ideal learning scenario would be to give the learner as large a set of training examples as possible, in order to attain a comprehensive sampling of the input space. Unfortunately, there are some real-world considerations that could seriously limit the size of training databases, such as shortage of free disk space and computation resource constraints.

How do we build a comprehensive but tractable database of "face" and "non-face"" patterns? For "face" patterns, the task at hand seems rather straight forward. We simply collect all the frontal views of faces we can find in mugshot databases and other image sources. Because we do not have access to many mugshot databases and the size of our mugshot databases are all fairly small, we do not encounter the problem of having to deal with an unmanageably large number of "face" patterns. In fact, to make our set of "face" patterns more comprehensive, we even artificially enlarged our data set by adding *virtual examples* [73] of faces to the "face" database. These virtual examples are mirror images

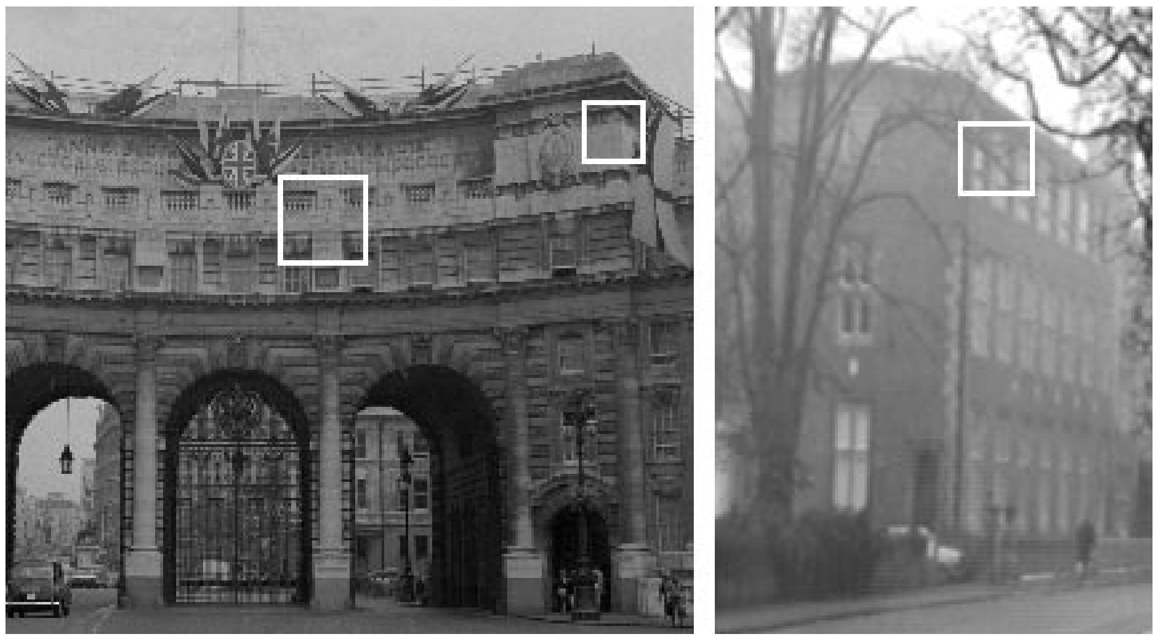


Figure 2-15: Some false detects by an earlier version of our face detection system, marked by solid squares on the two images above. We use these false detects as new negative examples to re-train our system in a “boot-strap” fashion.

and slightly rotated versions of the original face patterns, as shown in Figure 2-14.

For “non-face” patterns, the task we have seems more tricky. In essence, every square non-face window pattern of any size in any image is a valid training example. Collecting a “representative” set of non-face patterns is difficult because there are simply too many possibilities to consider. Even by choosing only non-face patterns from a few source images, our set of “non-face” examples can still grow intractably large if we are to include all valid “non-face” patterns in our training database.

To constrain the number of “non-face” examples in our database, we use a “boot-strap” strategy that incrementally selects only those “non-face” patterns with high information value. This “boot-strap” strategy reduces the number of “non-face” patterns needed to train a highly robust face detector. The idea works as follows:

1. Start with a small and possibly highly non-representative set of “non-face” examples in the training database.

2. Train the multi-layer perceptron classifier with the current database of examples.

3. Run the face detector on a sequence of images with no faces. Collect all the “non-face” patterns that the current system wrongly classifies as “faces” (see Figure 2-15). Add

these “non-face” patterns to the training database as new negative examples.

4. Return to Step 2.

At the end of each iteration, the “boot-strap” strategy enlarges the current set of “non-face” patterns with new “non-face” patterns that the current system classifies wrongly. We argue that this strategy of collecting wrongly classified patterns as new training examples is reasonable, because we expect these new examples to improve the classifier’s performance by steering it away from the mistakes it currently commits. Notice that if necessary, we can use the same “boot-strap” technique to enlarge the set of positive “face” patterns in our training database. Also, notice that at the end of each iteration, we can re-cluster our “face” and “non-face” databases to generate new prototype patterns that might model the distribution of face patterns more accurately.

In Chapter 4, we shall discuss the example selection issue in greater depth. *Active learning* is an area of research that investigates how one can select new training data in a principled and “optimal” fashion. We propose one such formulation for function approximation-based learning, and show how such a formulation, with some minor simplifications, can lead to a “boot-strap” example selection strategy like ours. We also present some empirical results suggesting that a face detection system actually generalizes better when trained with examples selected by “boot-strapping”.

2.7 Experimental Results

We implemented and tested our face detection system on a wide variety of images. Figures 2-16 to 2-23 show some sample results. The system detects faces over a fairly large range of scales, beginning with a window size of 19×19 pixels and ending with a window size of 100×100 pixels. Between successive scales, the window width is enlarged by a factor of 1.2.

The system writes its face detection results to an output image. Each time a “face” window pattern is found in the input, an appropriately sized dotted box is drawn at the corresponding window location in the output image. Notice that many of the face patterns in Figure 2-16 are enclosed by multiple dotted boxes. This is because the system has detected those face patterns either at a few different scales or at a few slightly offset window positions or both.

The upper and middle left images of Figure 2-16 show that our system works reliably without making many false positive errors (none in this case), even for fairly complex scenes. Notice that the current system does not detect Geordi's face. This is because Geordi's face differs significantly from the notion of a "typical" face pattern in our training database of faces — his eyes are totally occluded by an opaque metallic visor. The upper and middle right images demonstrate that system finds faces successfully across different scales. Finally, the bottom two images show the system detecting real faces and face drawings.

To quantitatively measure our system's performance, we ran our system on two test databases and counted the number of correct detections versus false alarms. All the face patterns in both test databases are new patterns not found in the training data set. The first test database consists of 301 frontal and near-frontal face mugshots of 71 different people. All the images are high quality digitized images taken by a CCD camera in a laboratory environment. There is a fair amount of lighting variation among images in this database, with about 10 images having very strong lighting shadows. We use this database to obtain a "best case" detection rate for our system on high quality input patterns. Notice that even though this first database contains mostly face images with simple background patterns, we believe that it is still a good test for gauging our system's ability to successfully recognize face patterns, because this operation does not depend on the image background appearance. On the other hand, the database is a poor test set for measuring our system's ability to correctly reject non-face patterns, because it clearly does not contain a representative sample of background patterns in real images.

The second database contains 23 images with a total of 149 face patterns. There is a wide variation in quality among the 23 images, ranging from high quality CCD camera pictures to low quality newspaper scans. Most of these images have complex background patterns with faces taking up only a very small percentage of the total image area. This second database serves two purposes. First, because the images vary greatly in quality, we use this database to obtain an "average case" successful face detection measure for our system. Second, because the images are of complex scenes, we also view this database as a more reliable test for how well our system correctly rejects non-face background patterns.

For the first database, our system correctly finds 96.3% of all the face patterns and makes only 3 *false detects*. All the face patterns that it misses have either strong illumination shadows or fairly large off-plane rotation components or both. Even though this high

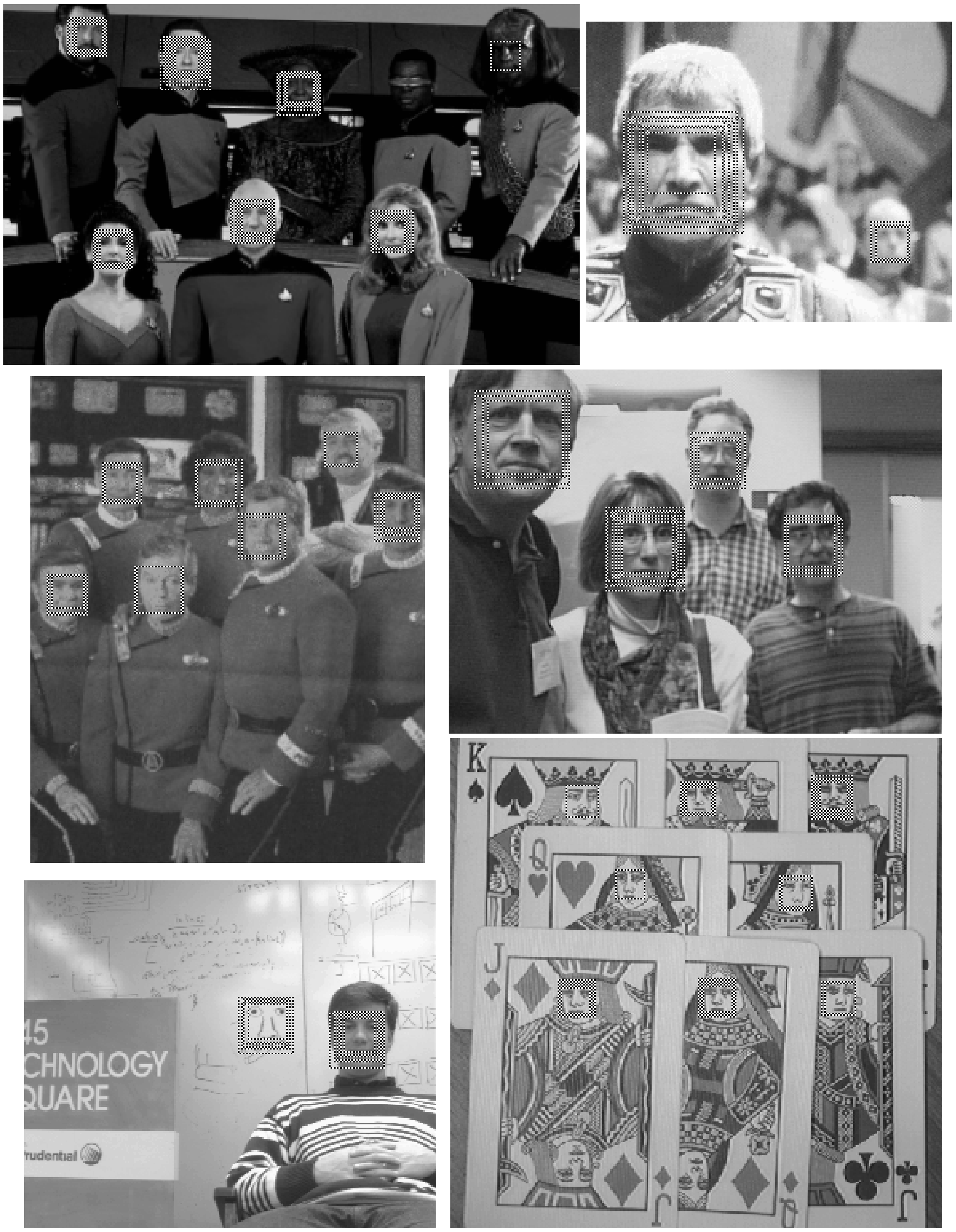


Figure 2-16: Some face detection results by our system. The upper and middle left images show that the system finds faces in complex scenes without making many false detects. The upper and middle right images show the system working successfully across different scales. The lower two images show that the same system detects real faces and face drawings equally well.

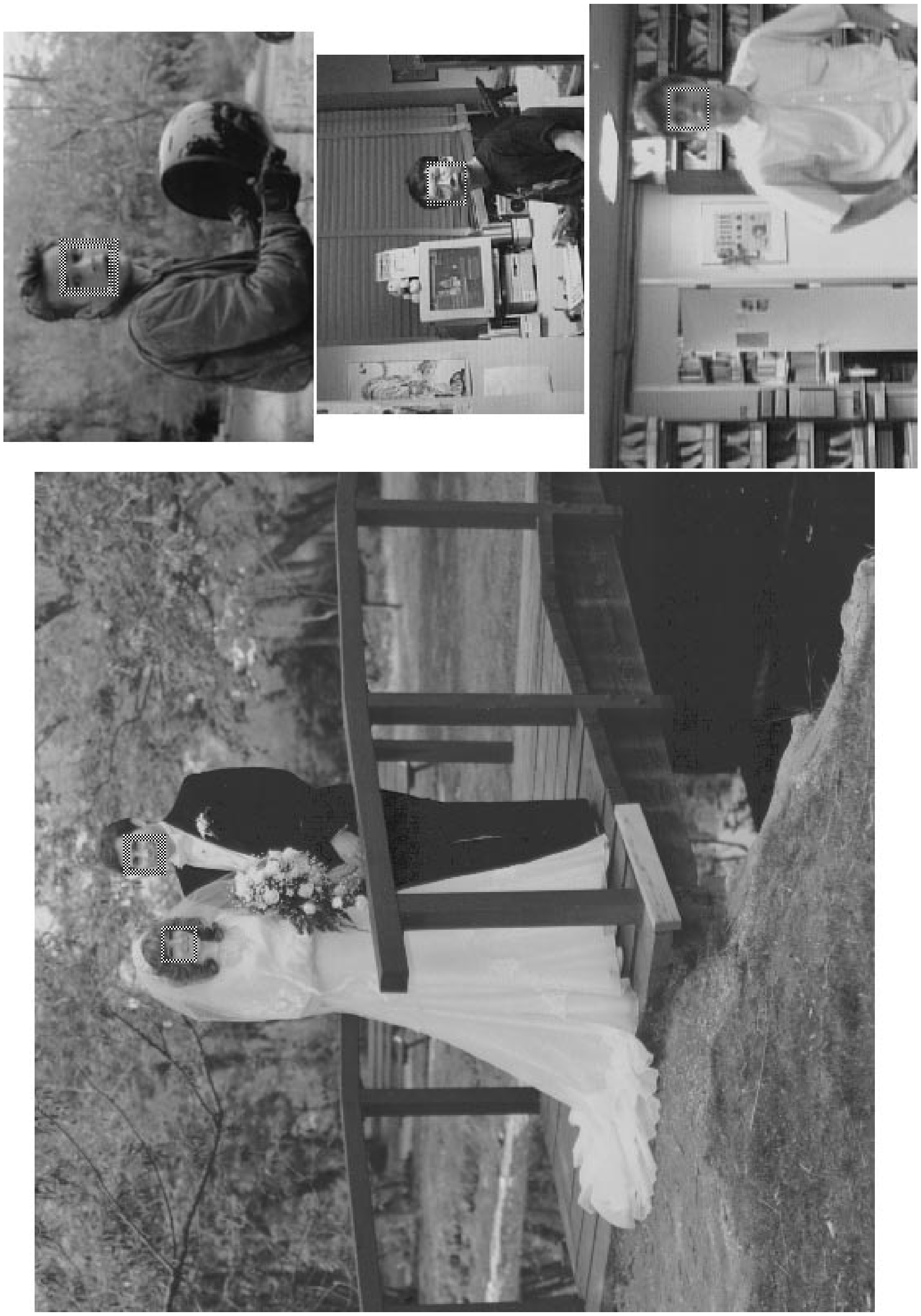


Figure 2-17: The system finds faces successfully without making any false detects in images with highly textured background patterns.

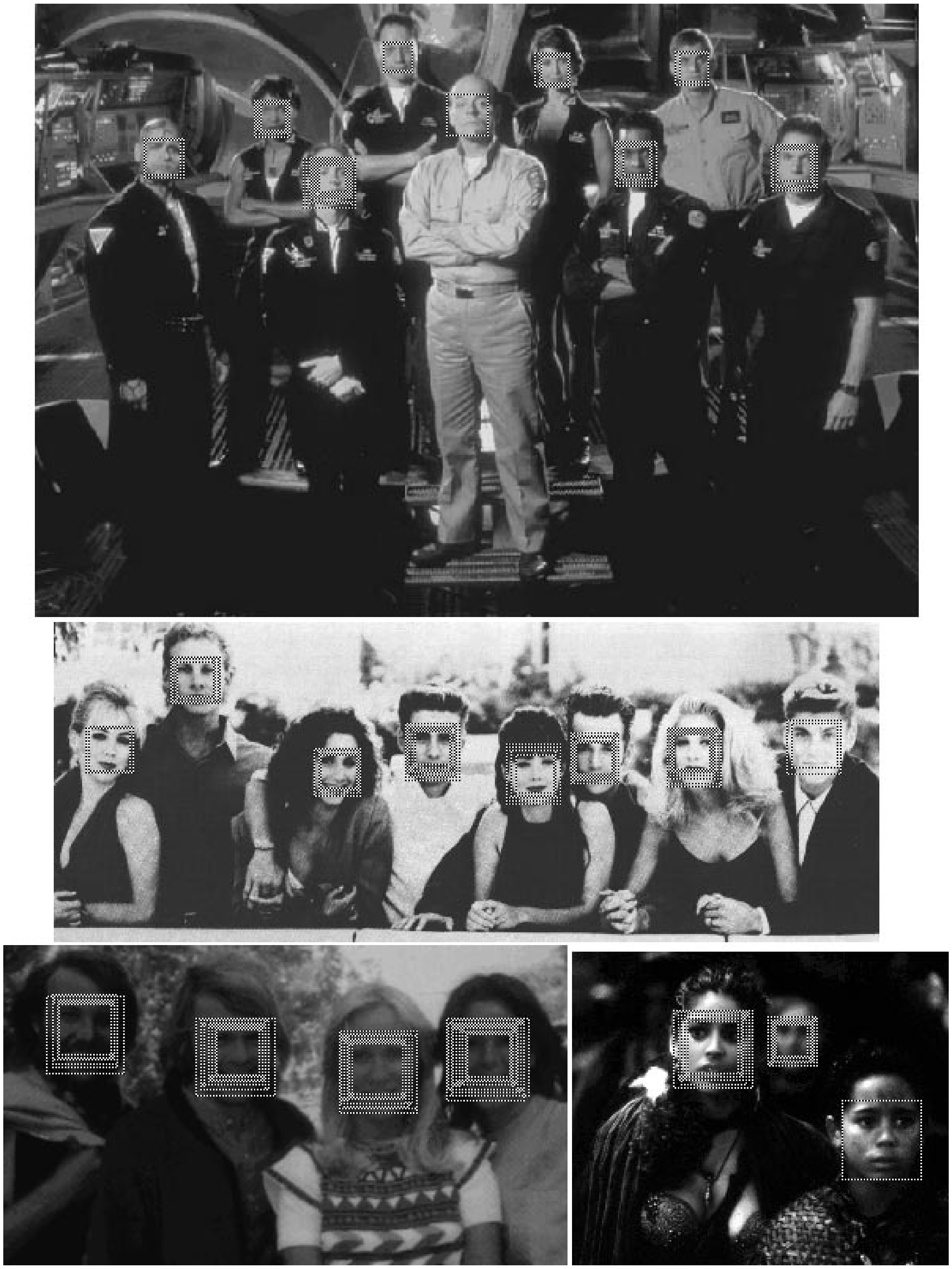


Figure 2-18: More face detection results by our system.

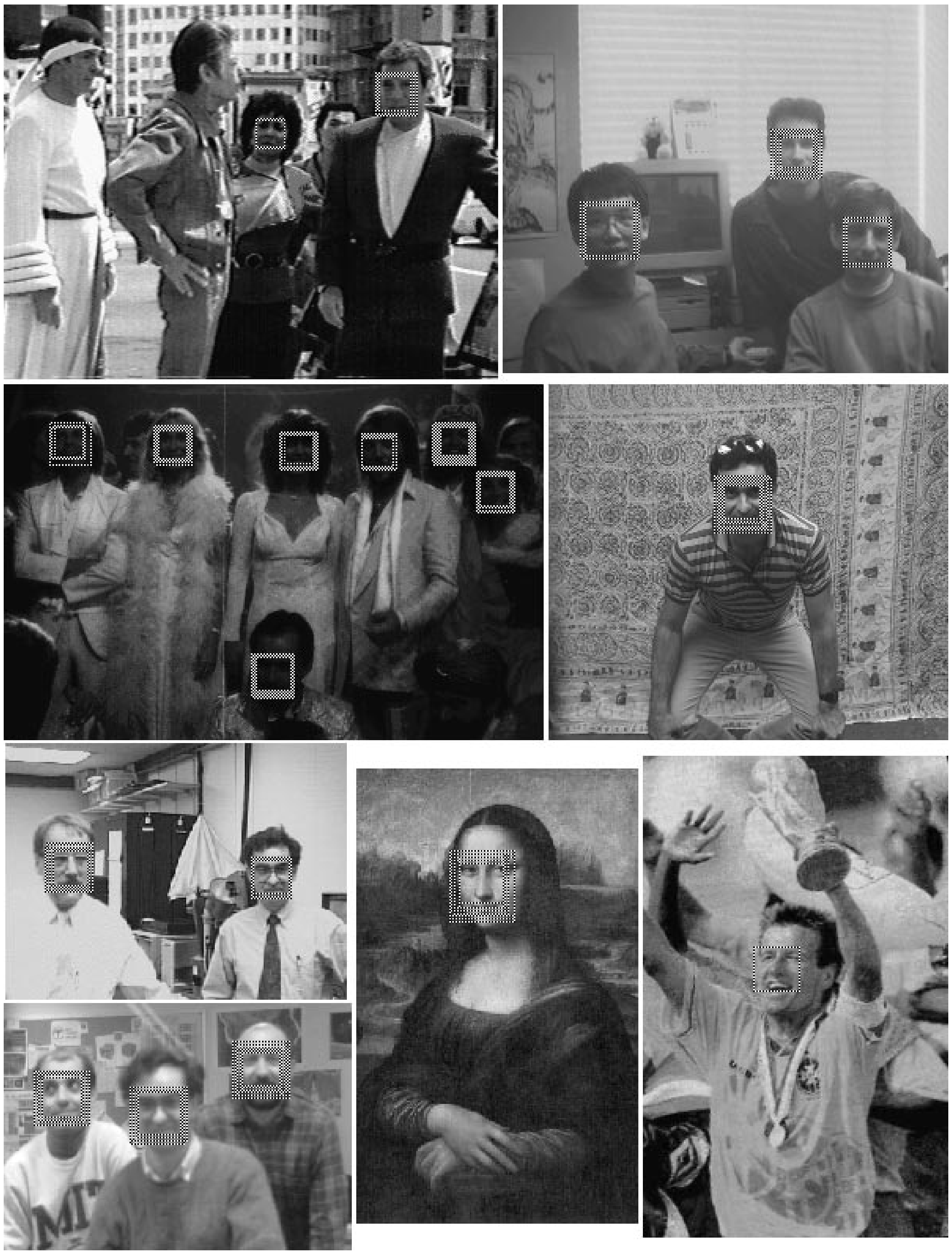


Figure 2-19: More face detection results by our system.

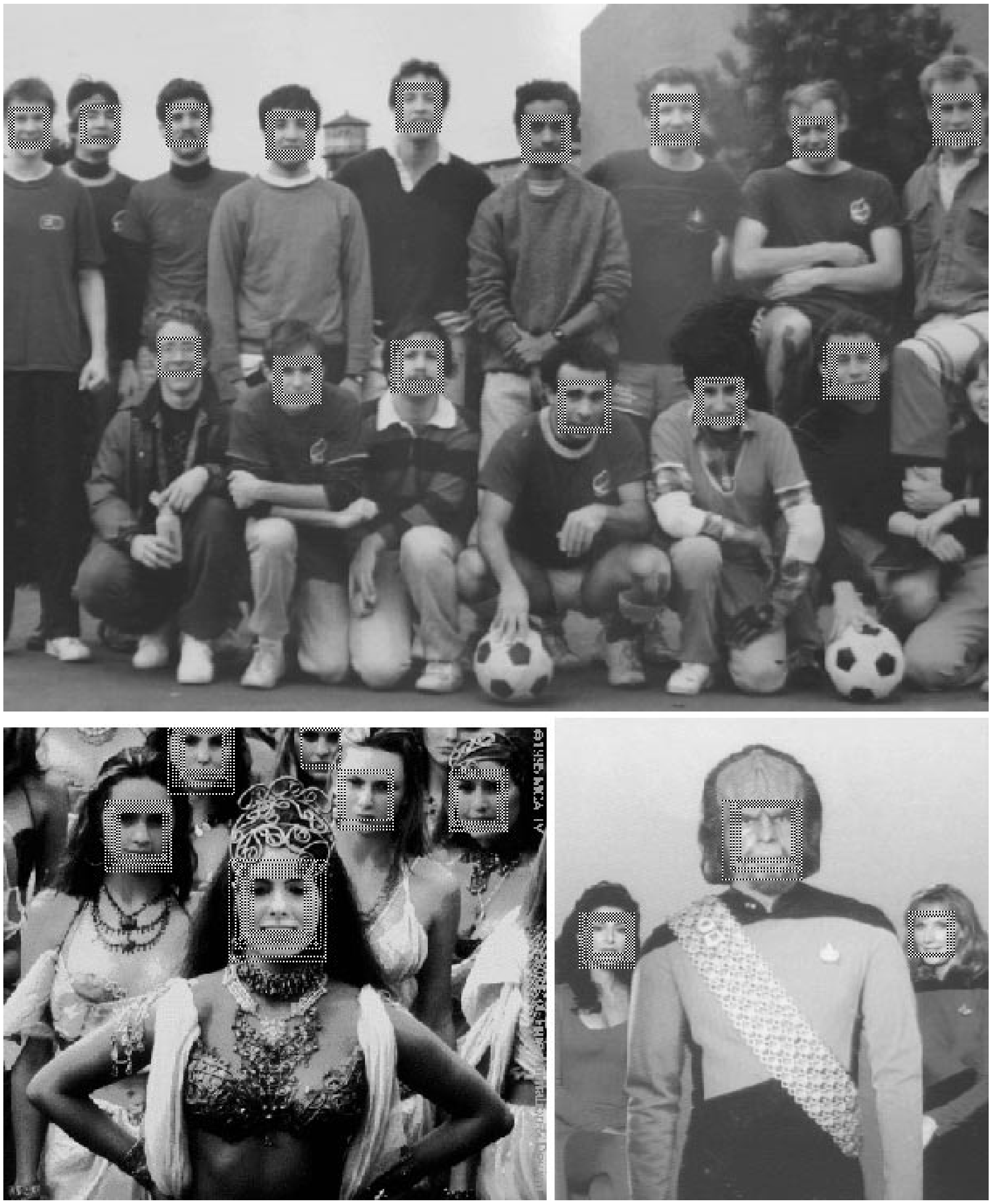


Figure 2-20: More face detection results by our system.

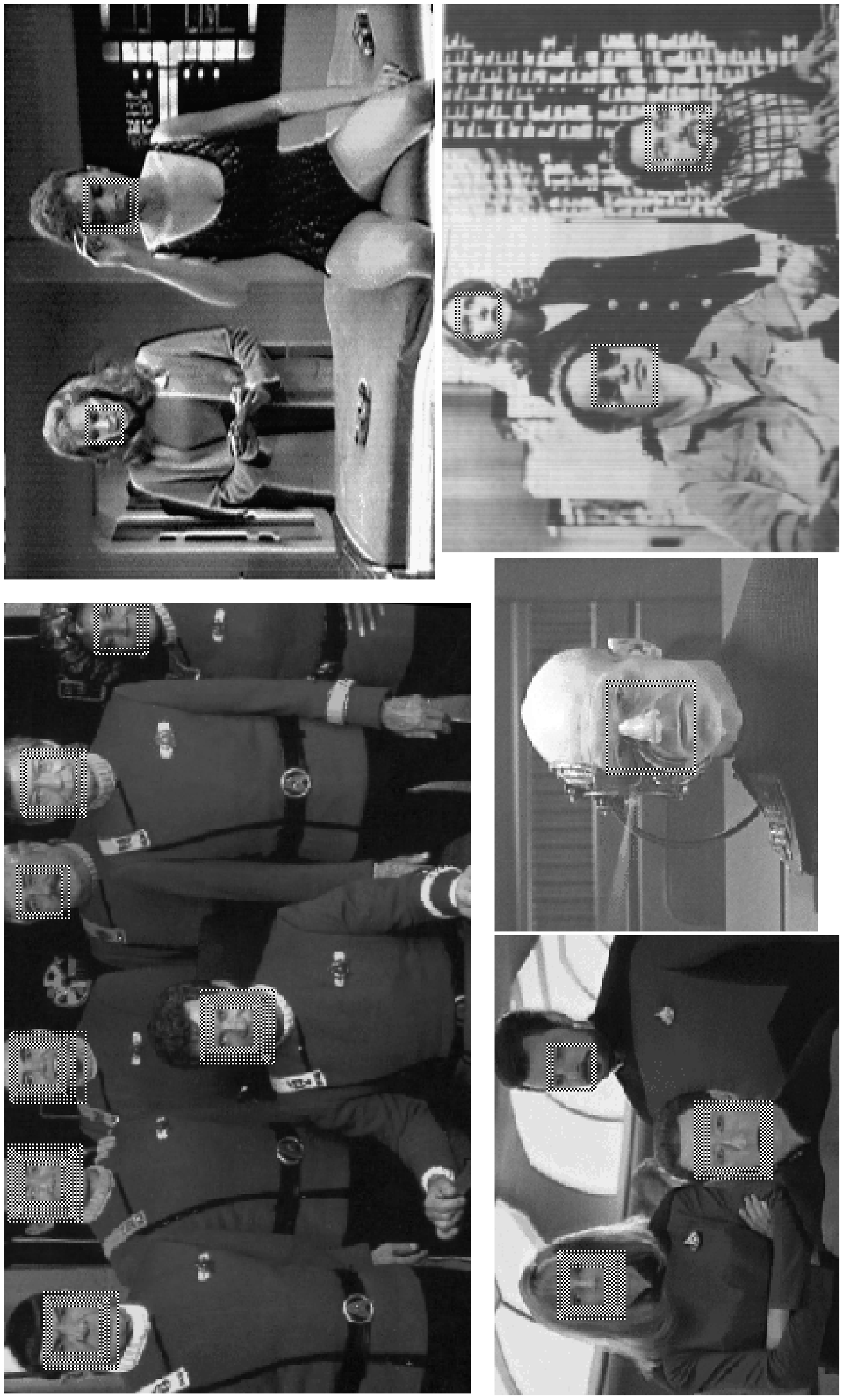


Figure 2-21: More face detection results by our system.

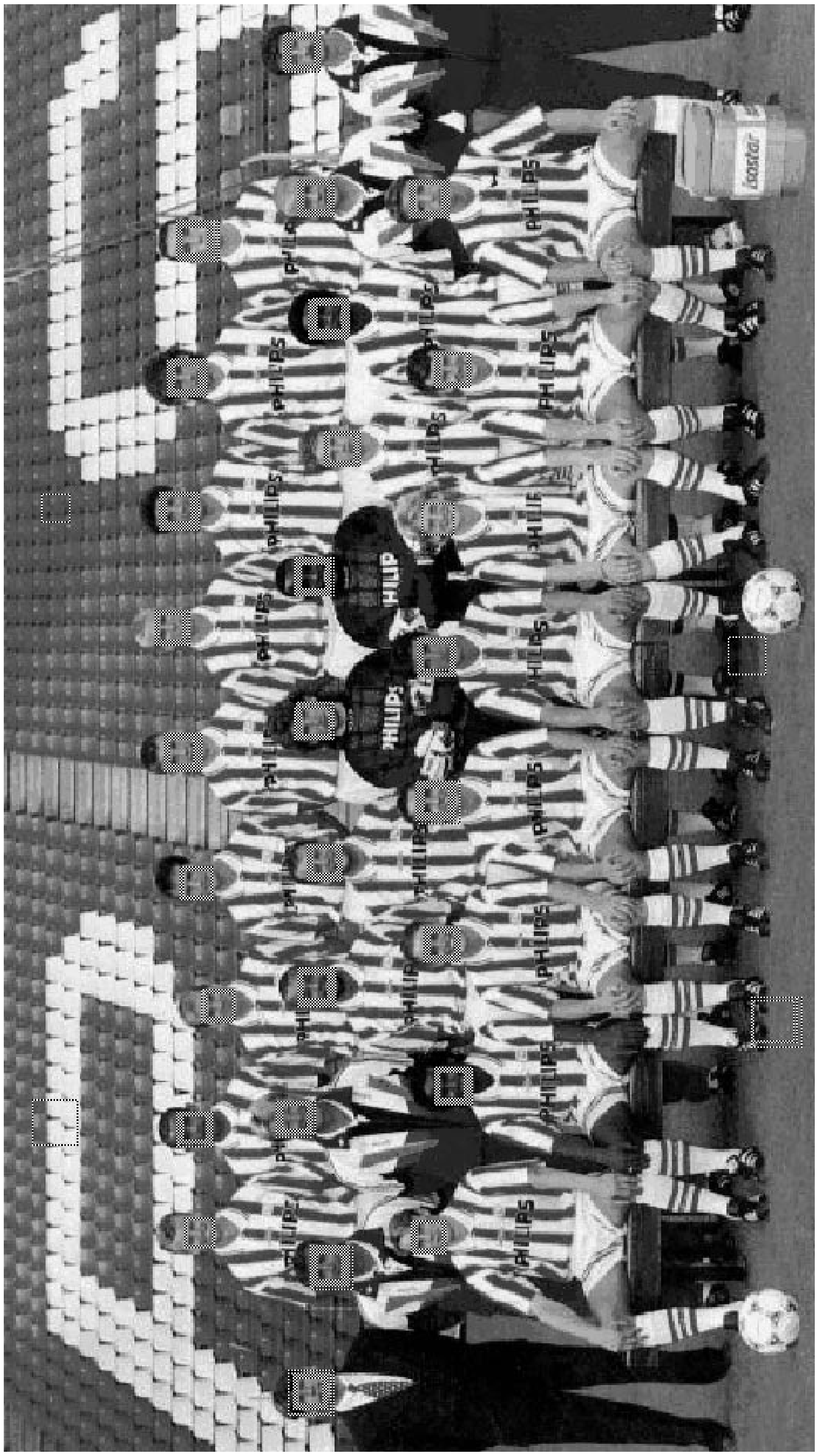


Figure 2-22: The systems finds every face in this image. It makes four false detects: **(1):** Near upper image border directly above second player from the left (top row). **(2):** Near upper image border directly above third player from the right (top row). **(3):** Lower image border between second and third player from the left. **(4):** Lower image border beside soccer ball further to the right.

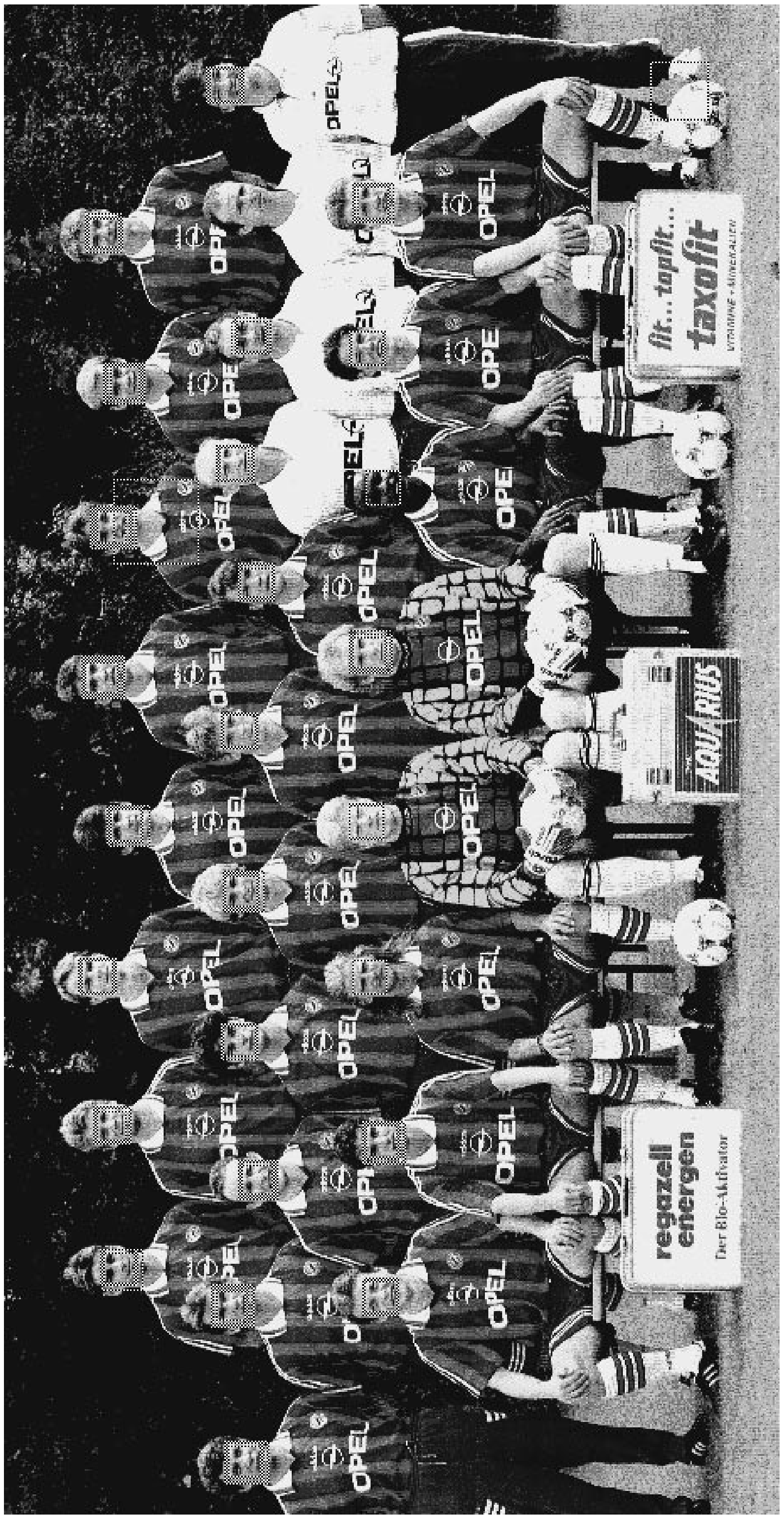


Figure 2-23: The system misses one face (middle row, second person from the right). It also makes two false detects: **(1):** Top row, third person from the right. **(2):** Lower right image corner on soccer ball.

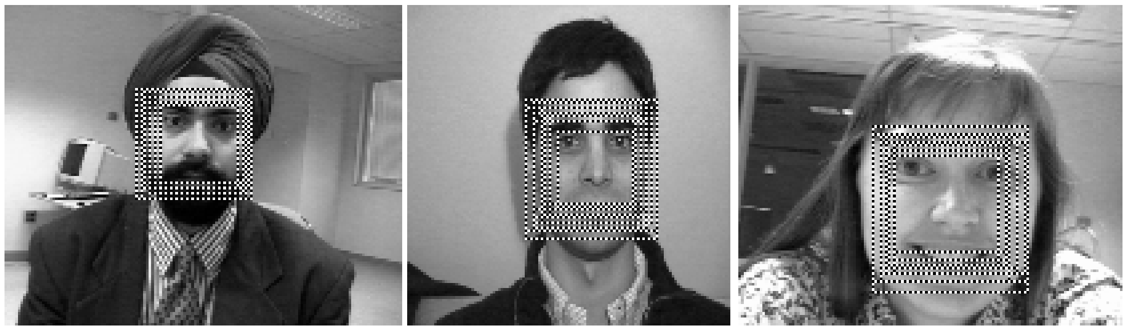


Figure 2-24: Some example images and detection results from the first test database. Even though this database contains mostly face images with simple background patterns, it is still a good test set for gauging our system's ability to successfully identify face patterns, because this operation does not depend on the image background appearance.

detection rate applies only to high quality input images, we still find the result encouraging because often, one can easily replace poorer sensors with better ones to obtain comparable results. For the second database, our system achieves a 79.9% detection rate with only 5 *false positives*. The face patterns it misses are mostly either from low quality newspaper scans or hand drawn pictures. We consider this behavior acceptable because the system is merely degrading gracefully with poorer image quality.

In Chapter 3, we investigate how modifying certain parts of our face detection system architecture affects the system's performance in terms of its face detection rate versus false alarm ratio. The study is part of an effort to identify and characterize the key components of our face detection system, and more generally, our approach for detecting spatially well-defined objects and pattern classes.

2.8 Other Systems

We conclude this chapter by comparing our work with three other face detection systems. Like ours, all three systems pass a test window over the input image to determine whether or not a face exists at each window location. The key difference is in the way each system tests its window patterns for faces.

2.8.1 Sinha

In an early window-based approach for finding faces, Sinha [84] uses a small set of spatial image invariants to describe the space of face patterns. He notes that even though windows containing face patterns may vary considerably in appearance for a variety of reasons, there are some spatial image relationships common and possibly unique to all face window patterns. To detect faces, the idea is to compile an appropriate set of these spatial relationships as invariants, and test for window patterns that satisfy these relationships.

Sinha's image invariance scheme is based on a set of observed brightness invariants between different parts of a human face. The underlying observation is that that while illumination and other changes can significantly alter brightness levels at different parts of a face, the local ordinal structure of brightness distribution remains largely unchanged. For example, the eye regions of a face are almost always darker than the cheeks and the forehead, except possibly under some very unlikely lighting conditions. Similarly, the bridge of the nose is always brighter than the two flanking eye regions. The scheme encodes these observed brightness regularities as a *ratio template*, which it uses to pattern match for faces. A ratio template is a coarse spatial template of a face with a few appropriately chosen sub-regions, roughly corresponding to key facial features. The brightness constraints between facial parts are captured by an appropriate set of pairwise brighter-darker relationships between corresponding sub-regions. An image pattern matches the template if it satisfies all the pairwise brighter-darker constraints.

So far, Sinha's invariance-based approach has only been demonstrated on images with little background clutter. Even though it is unclear how well the approach actually avoids false positive errors in cluttered scenes, we still find the spatial face representation scheme promising, and were encouraged to pursue a window-based face detection approach.

2.8.2 Moghaddam and Pentland

In a later system after ours, Moghaddam and Pentland [61] present a learning-based object finding technique that also models pattern classes by performing density estimation in a high dimensional space using eigenvector decomposition. These probability densities are then used to formulate a maximum-likelihood (ML) estimation framework for finding objects in images.

Their approach shares some common features with ours. Unlike an earlier scheme [69] that uses a unimodal Gaussian distribution to model data, their approach, like ours, uses a multimodel Gaussian mixture to approximate the target data distribution more tightly. To estimate high dimensional Gaussian densities with limited data, they also use a regularization approach that decomposes displacement vectors from a Gaussian centroid into two complementary components, identical to $\mathcal{D}_1$ and $\mathcal{D}_2$ of our 2-Value distance metric. Their approach, however, combines the two distances into a single value robust estimate for high dimensional Gaussian densities, whose form is the product of two lower-dimensional and independent Gaussian densities. This robust estimate is exactly a re-expression of Equation 2.1 as a probabilistic likelihood measure.

So far, Moghaddam and Pentland's approach has only been demonstrated on *localization* problems for human faces and hands. Recall that a *localization* problem differs from a *detection* problem in that the input image contains exactly one target object. Because a localization problem assumes exactly one target object, many *localization* approaches simply treat the localization task as searching an image for the location and scale that best matches a target model. The approaches assume that in any image containing a target object, the region containing the target object should match the model best. Moghaddam and Pentland's formulation for their object finding problem is exactly an object *localization* formulation. They cast the object finding problem as a visual search that returns only the best matching location and scale between a target model and an input image in a probabilistic maximum likelihood sense. We believe that a *detection* problem can be much harder than a *localization* problem, because in the former, one requires more than just a reasonable model to describe target patterns. One also needs reliable thresholds to determine whether or not a match is close enough to be accepted as a target object.

2.8.3 Rowley, Baluja and Kanade

In another recent piece of work after ours, Rowley et. al. [76] report a window-based face detecion approach that uses a highly constrained retinally connected multi-layer perceptron net to classify local image patches. Networks with similar architectures and connection schemes have been previously used in character recognition tasks [28]. To improve performance over a single network, the system arbitrates between the output of multiple networks to obtain a more reliable class prediction for each window pattern. The authors report that even simple arbitration schemes, like ANDing the output of two networks, helps greatly in reducing the number of false positives, with perhaps only a small penalty in the number of missed faces. They argue that this is because individual networks tend to detect roughly the same set of faces, but generally make different false positive errors. So the number of additional missed faces due to an ANDing operation should be small.

Like our technique, their network training procedure also requires a comprehensive but tractable set of prototypical “non-face” images. They use a “bootstrapping” method like ours to selectively add non-face patterns to the training set as training progresses. The “bootstrapping” method helps them reduce the non-face data set to only 9000 examples, which is a very small and tractable number.

In preparing their database of face patterns, Rowley et. al. also introduces additional normalization procedures which we neglected when preparing our own face database. They normalize each original face pattern so that both eyes appear on a horizontal line, and scale the image precisely so that the distance from a point between the eyes to the upper lip is fixed for all images. By normalizing the original face patterns, they were able to generate vitrual face patterns in a more principled fashion.

The individual networks they trained missed only one third the number of face patterns our system missed on a common test database. However, they each had about an order of magnitude more false positives than our system. After arbitration with several schemes, their resulting systems had slightly better detection and false alarm rates than ours.

Chapter 3

Generalizing the Object and Pattern Detection Approach

In this chapter, we generalize our frontal-view human face detection approach into a scheme for detecting spatially well-defined objects and pattern classes in images. We begin by reviewing the key components of our face detection system architecture within the framework of a general object and pattern class detection paradigm. The review attempts to understand the overall system design in terms of its individual components, their functionality, limitations and underlying assumptions. Much of our analysis here will be based on an effort to identify the key components of our face detection system, and design issues that significantly affect the system's performance. We perform this analysis mainly empirically, by studying how the system's performance varies as one changes the architecture of the individual components.

To demonstrate that the underlying pattern detection approach in our face detection system is indeed fairly general, we show three new applications based on the proposed approach. The first is a direct extension of our frontal human face detection system to handle a wider range of poses. We present a new but identically structured system, trained with additional face patterns covering a wider range of views. This application suggests that the approach is well suited for building highly extensible systems, where in principle, one can simply re-train a system with a larger example database to cover new sources of image pattern variations that one wishes to handle.

The second application is about detecting a different class of spatially well-defined objects — human eyes. Our goal here is to show that our proposed object and pattern detection approach works well for more than just human faces. Although there is less pattern variation between human eyes than between human faces, the eye detection problem is still challenging because there can be a lot more random background patterns that resemble eyes than faces, and a successful eye detector must correctly reject these patterns.

The third application deals with a pattern *recognition* task instead of a pattern *detection* problem like those seen so far. Pattern recognition and pattern detection are both instances of a wider class of computer vision problems, called *pattern classification*. The issues influencing both *recognition* and *detection* tasks can be very similar, and their solutions may have a lot in common. In this third application, we demonstrate that our proposed object and pattern detection approach can be a viable solution for certain pattern *recognition* problems as well. Specifically, we look at the task of recognizing isolated hand-printed digits using our underlying object and pattern class identification scheme. There has been a lot of research in hand-printed digit recognition over the past twenty to thirty years, with current state-of-art systems achieving recognition rates comparable to humans. We stress again that our goal here is not an attempt to better the state-of-art performance of hand-printed digit recognition systems. Rather, we wish to demonstrate that our proposed pattern detection approach is truly general enough to even model and capture localized pattern variations in a task that is essentially pattern *recognition* in spirit.

3.1 Overview of the General Object and Pattern Detection Approach

In Chapter 2, we introduced our distribution-based modeling cum example-based learning technique for detecting spatially well-defined objects and pattern classes in images, using a specific example problem on human face detection. In this section, we examine once again the key ideas and components of the underlying approach and system architecture, as a general scheme for detecting spatially well-defined objects and pattern classes. We shall focus exclusively on the pattern identification procedure within the overall search paradigm, which we consider as the most critical and difficult part of our pattern detection approach.

Recall that the pattern identification procedure's task is to correctly label all target input patterns and reject all background patterns. The task can be extremely complex if one has to deal with a wide variety of both target and background patterns. One key difficulty in dealing with highly varied target and background pattern classes is that often, pattern variations within each class can be too complex and significant to be adequately captured by geometric models, correlation templates and other well understood classical parametric modeling techniques. Our approach addresses this problem by modeling the distribution of target patterns directly in an appropriately chosen feature space, using a sufficiently large target pattern sample that covers all the main sources of image variation we wish to account for. The distribution-based modeling scheme statistically encodes observable target pattern variations that may not otherwise be easily parameterized by classical modeling techniques. We then use example-based function approximation schemes to learn a "similarity" measure between new patterns and the distribution-based target model in the chosen feature space. Basically, our approach is to synthesize a pattern identification procedure that computes and thresholds a measure that reflects the "similarity" between a new test pattern and the entire class of target patterns.

We now review the key issues and components of our generic object and pattern class detection approach in greater detail:

3.1.1 Defining a Suitable Feature Space for Representing Target Patterns

The first step in designing a pattern identification procedure is to define an appropriate feature space for modeling the target pattern distribution and classifying new image patterns. What makes a feature space suitable for representing and matching image patterns? Since our ultimate goal is to facilitate pattern classification in an object detection task, an ideal feature space should be one that maps all target image patterns onto an easily identifiable region, and all relevant background patterns onto a different region well separated from the former. A well separated and easily identifiable target pattern distribution can be easily and accurately represented by simple models. With a tractable representation scheme, the classification task becomes one of simply determining whether each new image pattern falls within the target region of the feature space.

For our purpose, we consider a feature space suitable if the target pattern class maps onto a continuous and smoothly varying sample distribution. One can then use existing density estimation techniques to obtain a tight but still fairly simple representation of the target pattern class in the chosen feature space. Because our entire object detection approach is based on representing and performing matches with pattern distributions, we believe that its success depends heavily on the feature space we choose to work in, and in particular, on whether the target pattern class is indeed continuously and smoothly distributed in the feature space, as our modeling schemes would assume.

Given a new detection task, how then can one find a suitable feature space for modeling the target pattern distribution? In general, we believe this can be a very difficult problem that involves understanding the key image attributes that characterize the target pattern class, and having available the types of image feature measurements, if any, that best describe the relevant attributes. Fortunately, we also believe it is often still possible to deduce reasonable feature spaces for many problems, with only some prior knowledge of the task. For example, most pattern classes that are recognized primarily by their structure, such as human faces and the digit class "2", can usually be well represented in a view-based feature space. Similarly, texture-based pattern classes, like roughness defects in lumber and certain tissue anomalies in medical images, can often be adequately represented in a frequency domain based feature space, where the transformed target patterns appear more stable.

In our face detection system, we used a view-based feature space of appropriately masked and normalized 19×19 pixel image window patterns to represent our face distribution. Because faces are highly structured in the image domain, and face patterns with minor spatial or grey-level variations still appear like valid face patterns, one can safely assume that the face pattern distribution is indeed continuous and smoothly varying in our chosen feature space. Similarly, most non-face patterns with minor variations still look like other non-face patterns, which suggests that the non-face pattern distribution is also continuous and smoothly varying in the image domain. For now, we shall assume that in most pattern detection problems, one can at least approach the feature space selection issue reasonably in an empirical fashion.

3.1.2 Modeling the Target Pattern Distribution

Given a suitable feature space to represent image patterns, the next step in designing a pattern identification procedure is to model the target pattern class as a data distribution. During detection, the pattern identification procedure matches new patterns against the distribution-based model to determine whether each new pattern belongs to the target class.

Ideally, we want an exact model that represents the actual target pattern distribution in the chosen feature space. In practice, this is often not possible because one usually does not have available all the target image patterns needed to recover the exact target pattern distribution. Our distribution-based modeling scheme addresses this problem by attempting to localize and approximate the region of target patterns with limited data. Basically, we use a reasonably large sample of target image patterns and a carefully chosen database of distractor patterns to bound the target region in the chosen feature space. We assume that the target pattern sample is sufficiently representative, and covers all the key sources of pattern variation we wish to account for. So, by building a model of this empirical target pattern distribution, one can still obtain a coarse but nevertheless fairly reliable representation of the actual target pattern distribution. We select our distractor pattern sample using the "bootstrap" example selection strategy introduced in the previous chapter. These distractor patterns are negative examples of the target class that lie near the target region boundary in the chosen feature space. We use these distractor patterns to help localize and refine the boundaries of the target model by explicitly carving out regions around the target sample distribution that do not correspond to target image patterns. In Chapter 4, we shall examine the "boot-strap" example selection strategy in greater detail.

Our approach uses a piecewise smooth Gaussian mixture density estimation technique to represent the empirical target pattern distribution in the chosen feature space. The pattern identification procedure uses the resulting distribution-based description as a model for the target pattern class. The piecewise smooth modeling scheme serves two important functions. First, it performs generalization by applying an assumed prior smoothness constraint to the empirical target sample distribution. This results in a stored data distribution function that is well defined even in regions of the feature space where no data samples have been observed. Notice however that this modeling scheme is only reasonable if the actual target pattern distribution is indeed *locally* continuous and smoothly varying as we have assumed. Second, the modeling scheme serves as a tractable means of representing an arbitrarily shaped data distribution using a few Gaussian basis functions. Specifically, our distribution-based Gaussian mixture modeling technique works as follows: We coarsely outline the target pattern sample with a few multi-dimensional Gaussian clusters. We also build a similar Gaussian mixture model for the distractor pattern sample to help localize and refine the boundaries of the actual target pattern distribution. There are two possibly related ways of reasoning about the distractor pattern clusters:

1. One can interpret the distractor pattern clusters as an explicit model for a "near-miss" pattern class in the chosen feature space. "Near misses" are background image patterns that can be easily mistaken as target patterns, often due to certain similarities in image appearance. The "near miss" distribution model provides the pattern identification procedure with additional knowledge to correctly distinguish these patterns from actual target patterns.

2. One can also view the distractor pattern clusters as an explicitly carved out set of regions in and around the target pattern distribution that should not correspond to target patterns. These carved out regions help shape the boundaries of the target pattern distribution by representing concavities in the Gaussian mixture model.

Our final distribution-based model thus consists of a few "positive" example clusters that coarsely approximate the target pattern region in the chosen feature space, and a few "negative" example clusters that help bound the target pattern region. Notice that although our approach uses a Gaussian mixture density estimation technique to model the target pattern class, we believe most piecewise smooth data representation schemes should also lead to satisfactory models that tightly approximate the target pattern distribution. This is true as long as our initial modeling assumptions are valid, i.e., the target pattern distribution is indeed continuous and smoothly varying in our chosen feature space.

3.1.3 Learning a Similarity Measure between New Patterns and the Distribution-based Target Model

The final task in designing a pattern identification procedure is to derive a suitable similarity measure for comparing new image patterns with the distribution-based target model. We approach this problem by: (1) defining a set of distance measurements that coarsely locates each test pattern with respect to the distribution-based target model in the chosen feature space; and (2) learning from examples an approximation function that combines the proposed distance measurements into an empirical similarity measure for matching test patterns against the target pattern class. During detection, one simply computes and thresholds this similarity measure to determine whether a new test pattern belongs to the target pattern class. One can view the empirical similarity function as a class identity predictor for input patterns. From a probabilistic standpoint, the similarity measure can also be interpreted as a conditional probability density, $P(\texttt{Class}(\vec{x}) = \texttt{Target}|\vec{x})$, where $\vec{x}$ is the input test pattern.

We describe first the distance measurements we use for comparing test patterns against our Gaussian mixture target model. Ideally, we want a set of measurements that clearly reflects the test pattern's location relative to the target pattern distribution as represented by the Gaussian mixture model. Such a set of measurements can serve as a highly discriminative set of input features for distinguishing between target and distractor image patterns. The actual measurements we use is a vector of distances between the input test pattern and all the Gaussian model's cluster centroids in the chosen feature space. One way of interpreting the vector of distances is to treat each distance measurement as the test pattern's actual displacement from some reference location along the target pattern distribution. The entire set of all distances can thus be viewed as a crude model-centered reference system that encodes an overall "difference" notion between the test pattern and the entire target pattern class.

Our approach uses a distribution dependent 2-value metric to represent individual distances between test patterns and each model centroid. For each Gaussian cluster in our mixture model, the first distance component is a directionally dependent *Mahalanobis* distance between the test pattern and the cluster centroid, in a vector sub-space spanned by the cluster's larger eigenvectors. This component computes a normalized pattern difference along the main elongation directions of the local data distribution represented by the current cluster. The normalized distance measure penalizes pattern differences less severely along the local data distribution, and more heavily against the local data distribution. This results in a distance value that better reflects the notion of "difference" between a test pattern and the *local* target pattern class. The second distance component is a standard Euclidean distance between the test pattern and its projection in the sub-space of larger eigenvectors. This distance component is a robust measure that accounts for pattern differences in the smaller eigenvector directions, not captured by the first distance component. We use a directionally independent distance measure for the second component because we believe the eigenvectors spanning this orthogonal sub-space may have significantly inaccurate eigenvalues. Normalizing pattern differences in this sub-space can therefore lead to meaningless if not adverse results. Our full set of measurements between each test pattern and the target model is thus a vector of several 2-value distances, whose dimensionality is twice the number of Gaussian clusters in our target model.

Given a set of 2-value distance measurements as input features, our approach uses a trained multi-layer perceptron net to implement a similarity function for comparing new test patterns against the target pattern class. We train the multi-layer perceptron net on distance measurements from a comprehensive but tractable set of target and distractor image pattern to perform the desired similarity computation. During training, we teach the net to output a similarity value of "1" for input distance measurements arising from target patterns, and a "0" otherwise. During detection, the net outputs a continuous similarity value in the $[0, 1]$ range, which can be interpreted as the probability of the test pattern belonging to the target class. One of the most critical issues in any non-trivial learning task is the problem of obtaining high quality example patterns for training. In Chapter 4, we shall examine in greater detail some principled techniques of selecting only useful training examples from among redundant ones to keep the learning task tractable.

In closing, we briefly discuss why we believe multi-layer perceptron nets are suitable, as an approximation function class, for combining our proposed intermediate distance measurements into an empirical indicator function for identifying target patterns. Multi-layer perceptron nets are non-local approximators that partition the input space with hyperplanes into regions of different output classes. We have argued that one can view the intermediate distance measurements we compute as a new model-centered co-ordinate system, where each axis represents the test pattern's distance from one of the model's several Gaussian clusters. In this new model-centered co-ordinate system, one can expect most target and background image patterns to occupy very distinctive regions that can be well separated by a small number of multi-layer perceptron hyperplanes. This is because in the original distribution-based modeling feature space, most target patterns tend to be located near the "positive" model clusters, which, in our new model-centered co-ordinate system, corresponds to a characteristic set of small distance values along certain axes. Most background patterns, on the other hand, tend to be located either near the "negative" clusters, or far away from the entire Gaussian model, which, in the new co-ordinate system, corresponds to a very different characteristic set of distance values. In fact, we shall see in the next section that for our human face detection example, even a single perceptron hyperplane does an almost perfect job at partitioning the space of model-centered distance measurements into separate target and background pattern regions.

3.2 Analyzing the System's Components

We continue our analysis of the proposed object and pattern class detection approach by examining some key design options that can significantly affect the resulting system's performance. We perform this part of the analysis empirically by studying how our human face system's detection and false alarm rates vary as one changes the architecture of certain system components. Specifically, we look at the following three aspects of the existing system design: (1) The classifier architecture, (2) our 2-Value distance metric versus other distance metrics for computing distance feature measurements, and (3) the importance of including negative (i.e. "non-face") clusters in our distribution-based model.

3.2.1 The Existing Human Face Detection System

To begin, we briefly summarize the relevant design features of our original human face detection system as presented in the previous chapter. The system models the face pattern distribution in an appropriately masked and normalized 19×19 pixel image feature space. We use 6 "positive" Gaussian clusters to approximate an empirical face distribution from an example database of 4150 face patterns, and another 6 "negative" Gaussian clusters to model an empirical "near-miss" distribution from a similar database of 6189 carefully chosen non-face patterns.

To match new test patterns against the piecewise smooth Gaussian mixture face model, the system first computes a set of 12 distances between each test pattern's location and the model's 12 cluster centroids in the 19×19 pixel image feature space. One can view the set of 12 distances as a crude "difference" notion between each test pattern and the entire "canonical face" pattern class. Each distance measurement between a test pattern and a cluster centroid is a robust and distribution dependent 2-Value metric presented earlier in Section 2.5.1.

Finally, the system uses a trained multi-layer perceptron net classifier to combine the 12 pairs of distance measurements into a single thresholdable similarity value that indicates whether or not a given test pattern is a face. The current multi-layer perceptron net architecture is as shown in Figure 2-13.

3.2.2 The Experiments

We now describe the experiments we perform to analyze the following three aspects of our system design.

Classifier Architecture

The first experiment investigates how varying the classifier's architecture affects our human face system's overall performance. To do this, we create two new systems with different classifier architectures, and compare their face detection versus false alarm statistics with those of our original system. Our two new classifier architectures are:

1. **A single perceptron unit.** We replace the original multi-layer perceptron net in Figure 2-13 with a single perceptron unit connected directly to the 12 pairs of input terminals. The single perceptron unit computes a sigmoidally thresholded weighted sum of its input feature distance vector. It represents the simplest possible architecture in the family of multi-layer perceptron net classifiers, and its purpose here is to provide an "extreme case" performance figure for multi-layer perceptron net classifiers in this overall pattern detection framework.

2. **A nearest neighbor classifier.** We perform nearest neighbor classification in the vector space of 12 2-Value distances to identify new test patterns. The nearest neighbor classifier works as follows: For each training pattern, we compute and store its 24 value distance feature vector and output class at compile time. When classifying a new test pattern, we compute its 24 value distance feature vector and return the output class of the closest stored pattern in Euclidean space. The nearest neighbor classifier provides us with a performance figure for a different classifier type in our proposed pattern detection framework.

The Distance Metric

The second experiment investigates how using different distance metrics for computing distance feature vectors in the matching stage affects the system's performance. We compare our 2-Value distance metric with three other distance measures: (1) the normalized Mahalanobis distance within a 75 dimensional vector subspace spanned by the prototype cluster's 75 largest eigenvectors — i.e. the first component only ($\mathcal{D}_1$) of our 2-Value distance metric, (2) the Euclidean distance between the test pattern and its projection in the 75 dimensional subspace — i.e. the second component only ($\mathcal{D}_2$) of our 2-Value distance metric, and (3) the standard normalized Mahalanobis distance ($\mathcal{M}_n$) between the test pattern and the prototype centroid within the full image vector space.

To conduct this experiment, we repeat the previous set of classifier experiments three additional times, once for each new distance metric we are comparing. Each new set of experiments differs from the original set as follows: During the matching stage, we use one of the three distance measures above instead of the original 2-Value distance metric to compute feature distance vectors between new test patterns and the 12 prototype centroids. Notice that because the three new distance measures are all single-value measurements, our new distance feature vectors have only 12 values instead of 24 values. This means that we have to modify the classifier architectures accordingly by reducing the number of input terminals from 24 to 12.

Negative Clusters

This experiment looks at how differently the system performs with and without "near-miss" clusters in the distribution-based model. We compare results from our original human face system, whose face model contains both "face" and "non-face" clusters, against results from two new systems whose internal models contain only "face" clusters. The two new systems are:

1. **A system with 12 "face" clusters and no "non-face" clusters.** In this system, we fix the total number of pattern clusters in the canonical face model, and hence the dimensionality of the feature distance vector the matching stage computes. We obtain the 12 "face" clusters by performing elliptical *k-means* clustering with 12 centers on our enlarged canonical face database of 4150 patterns (see Section 2.4.3). The matching stage computes the same 2-Value distance metric that our original system uses; i.e., for each Gaussian cluster, $\mathcal{D}_1$ is the normalized Mahalanobis distance in a subspace of the 75 largest eigenvectors, and $\mathcal{D}_2$ is the Euclidean distance between the test pattern and the subspace.

	Classifier Architecture					
Distance Metric	Multi-Layer		Single Unit		Nearest Nbr	
2-Value	96.3%	3	96.7%	3	65.1%	1
	79.9%	5	84.6%	13		
$\mathcal{D}_1$	91.6%	21	93.3%	15	97.4%	208
(first component)	85.1%	114	85.1%	94		
$\mathcal{D}_2$	91.4%	4	92.3%	3	53.9%	1
(second component)	65.1%	5	68.2%	5		
$\mathcal{M}_n$	84.1%	9	93.0%	13	71.8%	5
(Std. Mahalanobis)	42.6%	5	58.6%	11		

Table 3.1: Summary of performance figures from Experiments 1 (Classifier Architecture) and 2 (Distance Metric). Detection rates versus number of false positives for different classifier architectures and distance metrics. The four numbers for each entry are: **Top Left:** detection rate for first database. **Top Right:** number of false positives for first database. **Bottom Left:** detection rate for second database. **Bottom Right:** number of false positives for second database. Notice that we did not test the *nearest neighbor* architecture on the second database. This is because the test results from the first database already show that the *nearest neighbor* classifier is significantly inferior to the other 2 classifiers in this problem domain.

2. **A system with only 6 "face" clusters.** In this system, we preserve only the "face" clusters from the original system. The matching stage computes the same 2-Value distances between each test pattern and the 6 "face" centroids. Notice that the resulting feature distance vectors have only 6 pairs of values.

We generate two sets of performance statistics for each the two new systems above. For the first set, we use a trained multi-layer perceptron net with 12 hidden units to classify new patterns from their distance feature vectors. For the second set, we replace the multi-layer perceptron net classifier with a trained single perceptron unit classifier.

3.2.3 Performance Statistics and Interpretation

Table 3.1 summarizes the performance statistics for both the *classifier architecture* and *distance metric* experiments, while Table 3.2 shows how the system performs with and without modeling the "near-miss" distribution.

Classifier Architecture

The horizontal rows of Table 3.1 show how different classifier architectures affect the human face system's detection rate versus false alarm occurrences. Quantitatively, the two network-based classifiers have very similar performance figures, while the *nearest neighbor* classifier produces rather different performance statistics. Depending on the distance metric being used, the *nearest neighbor* classifier has either a somewhat higher face detection rate with a lot more false alarms, or a much lower face detection rate with somewhat fewer false alarms than the other two classifiers. Because we do not have a reasonable measure of relative importance between face detection rate and number of false alarms, we empirically rank the performance figures by visually examining their corresponding output images. In doing so, we find that the two network-based classifiers produce results that are a lot more visually appealing than the nearest neighbor classifier.

It is interesting that the two network-based classifiers we tested produce very similar performance statistics, especially on the first test database. The similarity here suggests that the system's performance depends most critically on the classifier type, i.e. a perceptron net, and not on the specific net architecture. It is also somewhat surprising that one can get very encouraging performance figures even with an extremely simple single perceptron unit classifier, especially when using our 2-Value metric to compute distance feature vectors. This suggests that the distance measurements we get with our 2-Value metric are a linearly very separable set of features for "face" and "non-face" patterns. We shall see from the new applications we present later in this chapter that this observation extends to other target pattern classes as well.

Why does the *nearest neighbor* classifier have a much lower face detection rate than the other two network-based classifiers when used with our 2-Value distance metric? We believe this has to do with the much larger number of non-face patterns in our training database than face patterns (43166 non-face patterns versus 4150 face patterns). The good performance figures from the single perceptron classifier suggest that the two pattern classes are linearly highly separable in our 24 value distance feature vector space. Assuming that roughly the same fraction of face and non-face patterns lie along the linear class boundary, we get a boundary population with 10 times more non-face samples than face samples. A new face pattern near the class boundary will therefore have a very high chance of lying nearer a non-face sample than a face sample, and hence be wrongly classified by a nearest neighbor scheme. Notice that despite the huge difference in number of face and non-face training patterns, a perceptron classifier can still locate the face versus non-face class boundary accurately if the two pattern classes are indeed linearly highly separable, and hence still produce good classification results.

Classifier Architecture	Composition of Clusters					
	6 Face & 6 Non-Face		12 Face		6 Face	
Multi-layer Perceptron	96.3%	3	85.3%	21	59.7%	17
	79.9%	5	69.6%	74	60.9%	41
Single Perceptron	96.7%	3	52.1%	6	66.6%	25
	84.6%	13	49.7%	16	55.4%	56

Table 3.2: Summary of performance figures for Experiment 3 (The effect of "Near-Miss" Clusters). Detection rates versus number of false positives for different classifier architectures and composition of clusters in distribution-based model. The four numbers for each entry are: **Top Left:** detection rate for first database. **Top Right:** number of false positives for first database. **Bottom Left:** detection rate for second database. **Bottom Right:** number of false positives for second database.

The Distance Metric

The vertical columns of Table 3.1 show how the human face system's performance changes with different distance metrics in the pattern identification stage. Both the multi-layer perceptron net and single perceptron classifier systems produce better performance statistics with our 2-Value distance metric than with the other three distance measures, especially on images from the second test database. The observation should not at all be surprising. Our 2-Value distance metric consists of both $\mathcal{D}_1$ and $\mathcal{D}_2$, two of the three other distance measures we are comparing against. A system that uses our 2-Value distance should therefore produce performance figures that are at least as good as a similar system that uses either $\mathcal{D}_1$ or $\mathcal{D}_2$ only. In fact, since our 2-Value distance systems actually produce better classification results than the systems using only $\mathcal{D}_1$ or $\mathcal{D}_2$, one can further affirm that the two components $\mathcal{D}_1$ and $\mathcal{D}_2$ do not mutually contain totally redundant information.

Our 2-Value distance metric should also produce classification results that are at least as good as those obtained with a Mahalanobis distance metric. This is because both metrics treat and quantify *distance* in essentially the same way — as a "difference" notion between a test pattern and a local data distribution. Furthermore, the "difference" notions they use are based on two very similar Gaussian generative models of the local data distribution. In our experiments with perceptron-based classifiers, the 2-Value distance metric actually out-performs the Mahalanobis distance metric consistently. We suggest two possible reasons for this difference in performance: (1) As discussed in Section 2.5.5, we have too few sample points in our local data distribution to accurately recover a full Gaussian covariance matrix for computing Mahalanobis distances. By naively trying to do so, we get a distance measure that poorly reflects the "difference" notion we want to capture. (2) The local data distribution we are trying to model may not be truly Gaussian. Our 2-Value distance metric provides an additional degree of freedom that better describes a test pattern's location relative to the local non-Gaussian data distribution.

It is interesting that even a network-based classifier with a "partial" $\mathcal{D}_2$ distance metric almost always out-performs a similar classifier with a "complete" Mahalanobis distance metric. At first glance, the observation can be surprising because unlike the "complete" Mahalanobis distance, the $\mathcal{D}_2$ metric alone does not account for pattern differences in certain image vector space directions. More specifically, the $\mathcal{D}_2$ metric only measures a pattern's Euclidean distance from a cluster's 75 most significant eigenvector subspace, and does not account for pattern differences within the subspace. We believe this comparison truly reveals that without sufficient data to accurately recover a full Gaussian covariance matrix, the resulting Mahalanobis distance can be a very poor notion of "pattern difference".

"Near-Miss" Clusters

Table 3.2 summarizes the performance statistics for comparing systems with and without a "near-miss" distribution model. As expected, the systems with "non-face" clusters clearly out-perform those without "non-face" clusters. Our results suggest that not only do the "near-miss" clusters provide an additional set of distance measurements for pattern detection, the resulting measurements are in fact a very discriminative set of additional classification features.

3.3 New Application 1: Variable Pose Human Face Detection

We conclude this chapter by showing three new vision related applications based on our proposed object and pattern class detection scheme, to demonstrate that the approach is indeed fairly general and extensible. The first example is a direct extension of our frontal view human face detection system to handle a wider range of poses. The new system we present is structurally almost identical to our original frontal view face detection system, with some minor design differences described below. We use the same distribution-based modeling cum learning-based training techniques to construct the new system. Functionally,

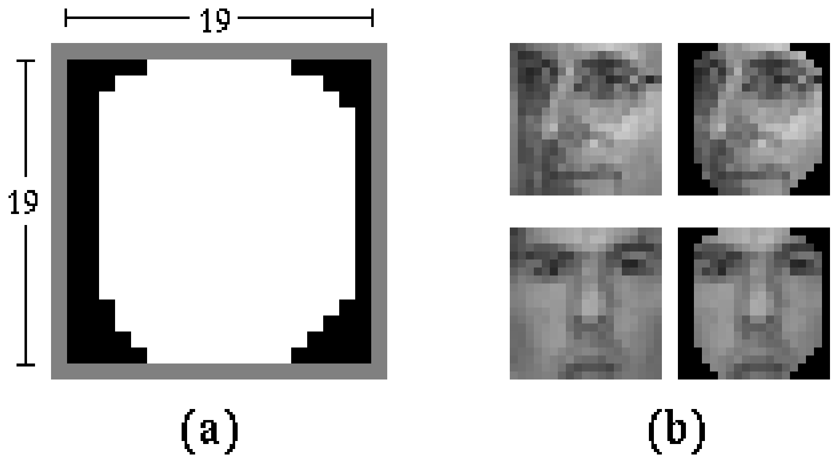


Figure 3-1: **(a):** A 19×19 pixel mask that approximates an "average" silhouette of a left off-plane rotated face view. **(b):** The mask does a reasonable job at eliminating background pixels in both left-rotated and frontal face patterns.

this new application detects vertically oriented human faces with off-plane rotation angles of up to $\pm 45°$.

3.3.1 Implementation Details

Our approach deals with a wider range of face views by using a larger example database that includes both frontal and non-frontal face patterns. The non-frontal face patterns in the database account for pattern variations due to changes in pose. We use this enlarged face database for two purposes: (1) to approximate a new target class distribution of frontal and non-frontal face patterns; and (2) as additional training examples to synthesize an appropriate similarity function for matching test patterns against the extended distribution-based model. In our actual implementation, we consider a smaller target pattern class of only frontal to left off-plane rotated face views. Because there is less pattern variation in this smaller target pattern class, one can still tightly approximate the target distribution with fewer data samples and fewer Gaussian model clusters. To detect right off-plane rotated face views as well, the system matches both an input image pattern and its mirror view against the simplified target model, and accepts the pattern as a face if either match responds positively.

As in our original frontal-view face detection system, the distribution-based modeling stage here also uses an appropriately masked and intensity normalized view-based feature space of fixed-sized image patches to model the extended face distribution. This is possible because we believe the extended target class of frontal and non-frontal face patterns is still continuous and smoothly distributed in a view-based feature space, where incremental changes in object pose also correspond to minor displacements in feature space location. Our chosen feature space is an illumination corrected and histogram equalized 19×19 pixel image space with a slightly different mask pattern as shown in Figure 3-1(a). The new mask pattern approximates an "average" silhouette of left-rotated face views, which eliminates background pixels of both frontal and left-rotated face patterns fairly well. We acknowledge, however, that there are more robust methods for eliminating background pixels of non-frontal face patterns. We shall only briefly describe one possible approach below.

Our final distribution-based target model contains 10 multi-dimensional Gaussian "face" clusters and 10 similar "near-miss" clusters. We obtain the 10 "face" clusters by performing elliptical *k-means* clustering on a combined database of 6845 frontal and left-rotated face patterns, without first partitioning the data set into pattern sub-classes of different pose. Processing all the target patterns together allows the resulting model clusters to generalize across pose as well as other sources of image pattern variation. We obtain our 10 "near-miss" clusters in a similar fashion from a specially chosen database of 11810 face-like patterns, generated by our "boot-strap" example selection and training strategy described in the previous chapter. In building this system, we simply guessed an appropriate number of model clusters to use, by taking the original frontal-view face detection system as a design reference. Our extended system has almost twice as many face data samples as the original system which contains 6 "face" and 6 "non-face" clusters, so we simply increased the current number of model clusters proportionately. One can also argue that our extended system needs a larger number of model clusters because it detects faces over a wider range of pose, and hence requires a more complex model to approximate a larger target pattern class.

The final pattern identification stage uses a trained multi-layer perceptron net with 20 input unit pairs to classify test patterns from their computed distance feature vectors. Each input unit pair receives a 2-value distance measurement between the test pattern's view-based feature space location and one of the model centroids. We train the multi-layer perceptron net on an example database of over 42000 distance feature vectors and their corresponding output classes. The training database includes vectors from all the "face" and "near-miss" patterns used for building our distribution-based model, as well as a random sample of non-face patterns from a large number of natural images.

3.3.2 Results

We have implemented a preliminary variable pose face detection system using the proposed approach and design choices described above. Figures 3-2 to 3-4 show some sample results. Notice that there are still a small but significant number of missed faces and false alarms in the last two images. Because our goal here is merely to demonstrate the extensibility of our pattern detection approach through an example domain of faces, we were satisfied with the current results and have not attempted to further improve the existing system.

Despite having used a larger number of "relevant" training examples and a more complex Gaussian mixture model for estimating the target pattern distribution in this application, our current variable pose system still performs more poorly than our original frontal-view face detection system in terms of classification correctness. We suggest two possible factors contributing to its poorer performance and ways of improving the current results.

1. The extended target class of frontal and off-plane rotated face patterns is significantly more complex than the original frontal face pattern class, such that the number of additional "face" examples we have gathered for modeling is still insufficient with respect to the increase in target class complexity. Although one can reduce *generalization* errors due to overfitting by constraining model complexity, one will still get significant *empirical* errors with too little data to capture all the key variations in a highly complex target class (see for example [65]). One can overcome the problem of insufficient data by collecting more real and virtual face image patterns over the range of rotation angles we wish to handle. Recently, Beymer et. al. [13] [11] have demonstrated a useful technique for learning complex transformations and generating multiple virtual views of an arbitrary human face from a single face view.

2. The mask pattern we are using to eliminate irrelevant background pixels in "face" window patterns (see Figure 3-1(a)) matches only an "average" silhouette of all the left-rotated face views in our training database. There can be many valid face patterns, even among those in our training database, whose individual silhouettes differ significantly from the mask outline. When masked, these face patterns can still con-

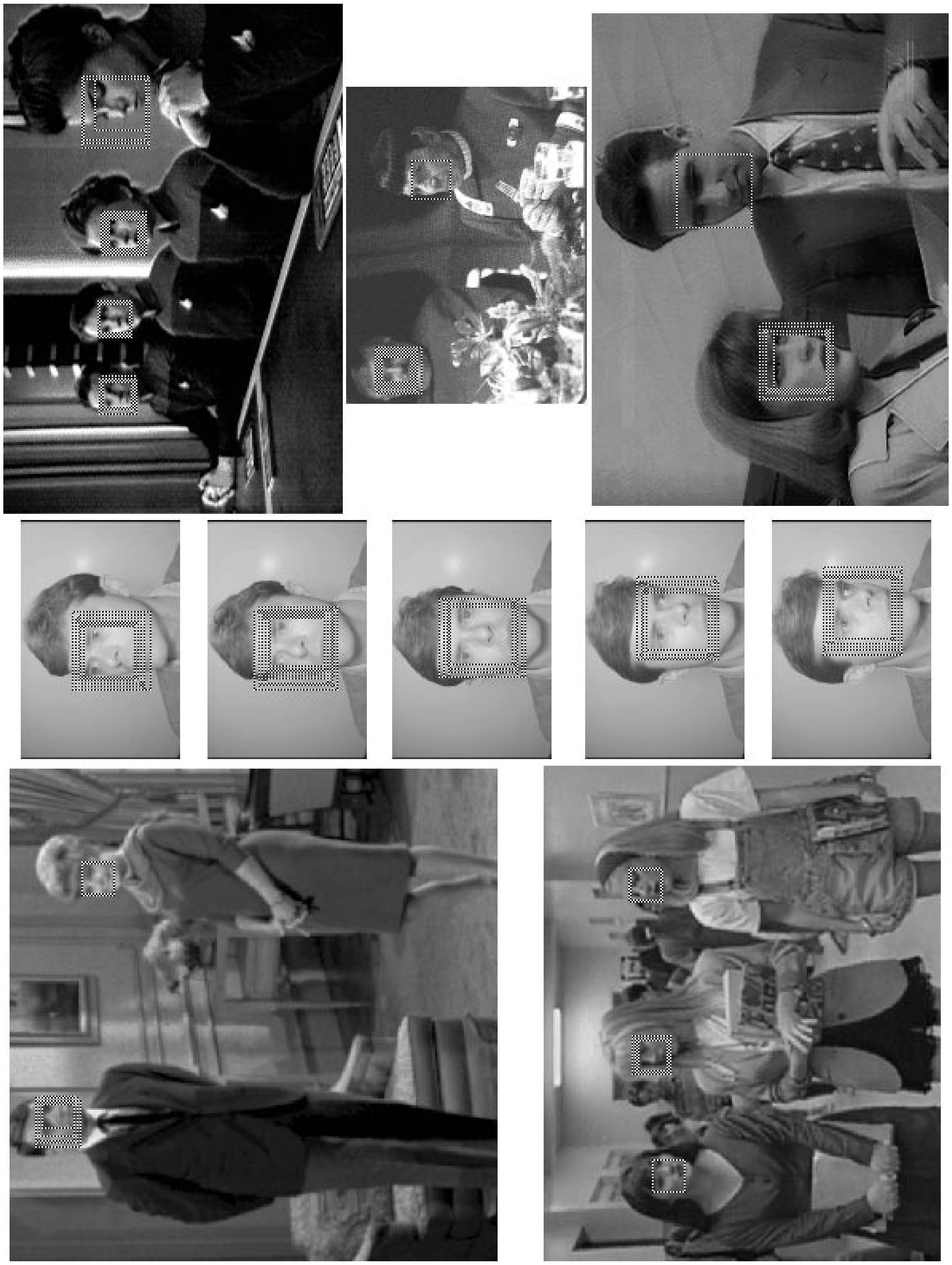


Figure 3-2: Non frontal face detection results. The middle set of 5 pictures shows the system detecting faces successfully over a range of off-plane rotated angles. The other images show non-frontal faces detected in a complex scene.

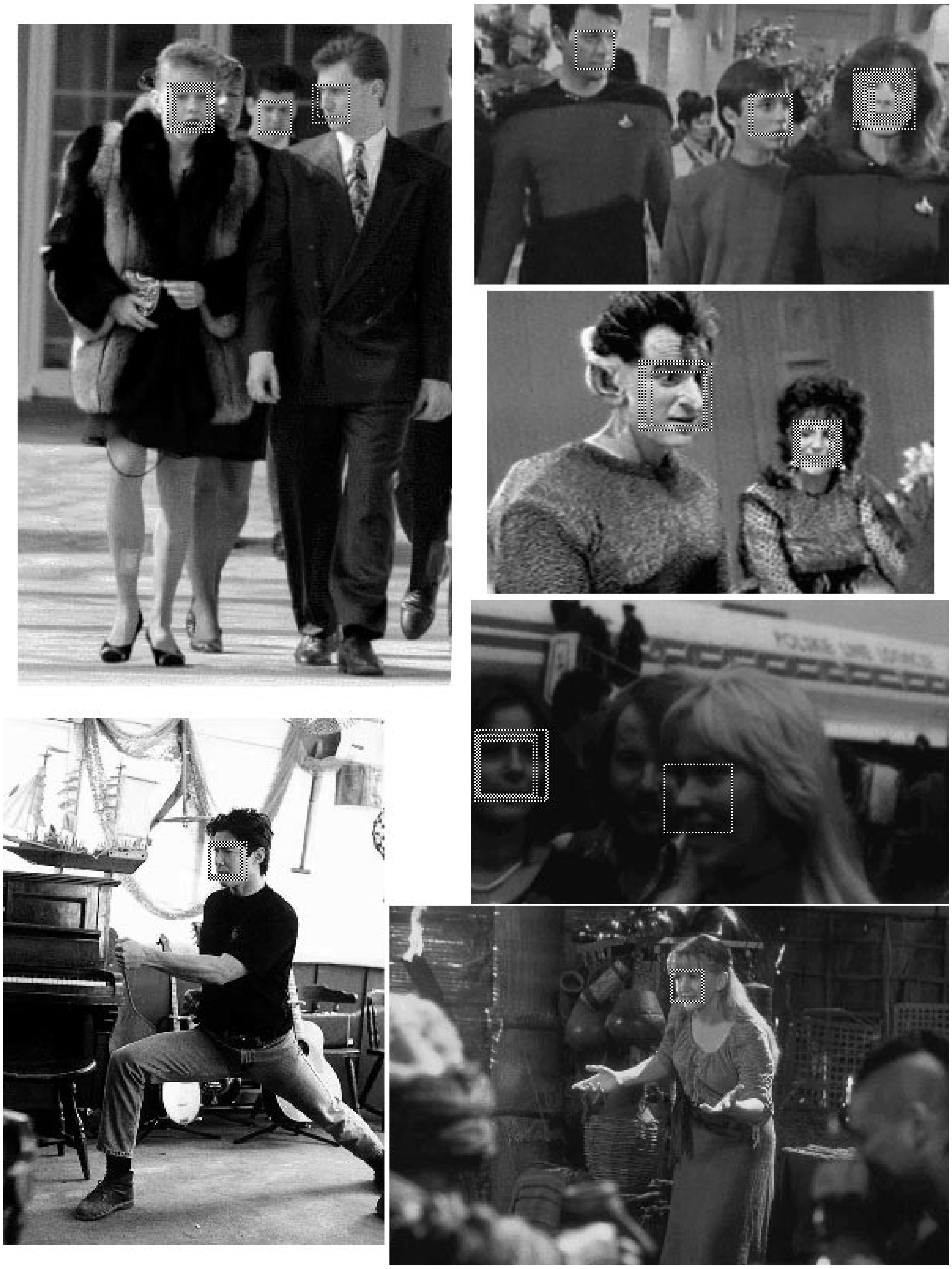
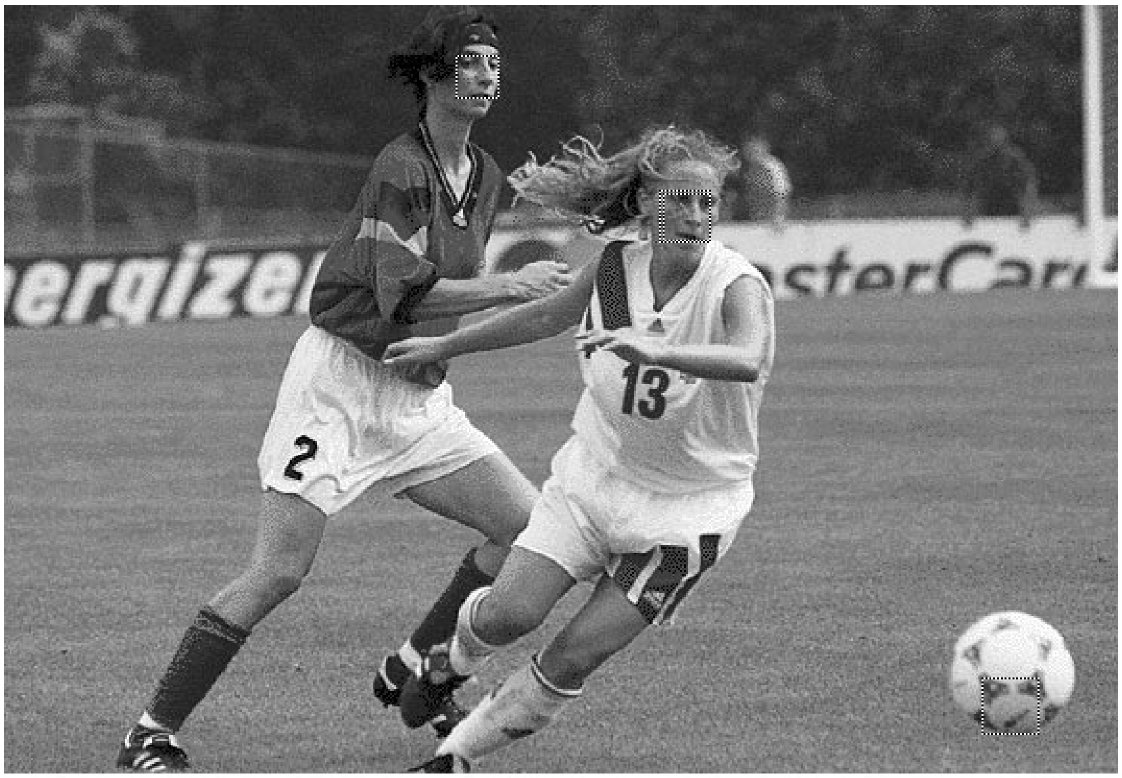


Figure 3-3: More images with frontal and off-plane rotated faces successfully detected.



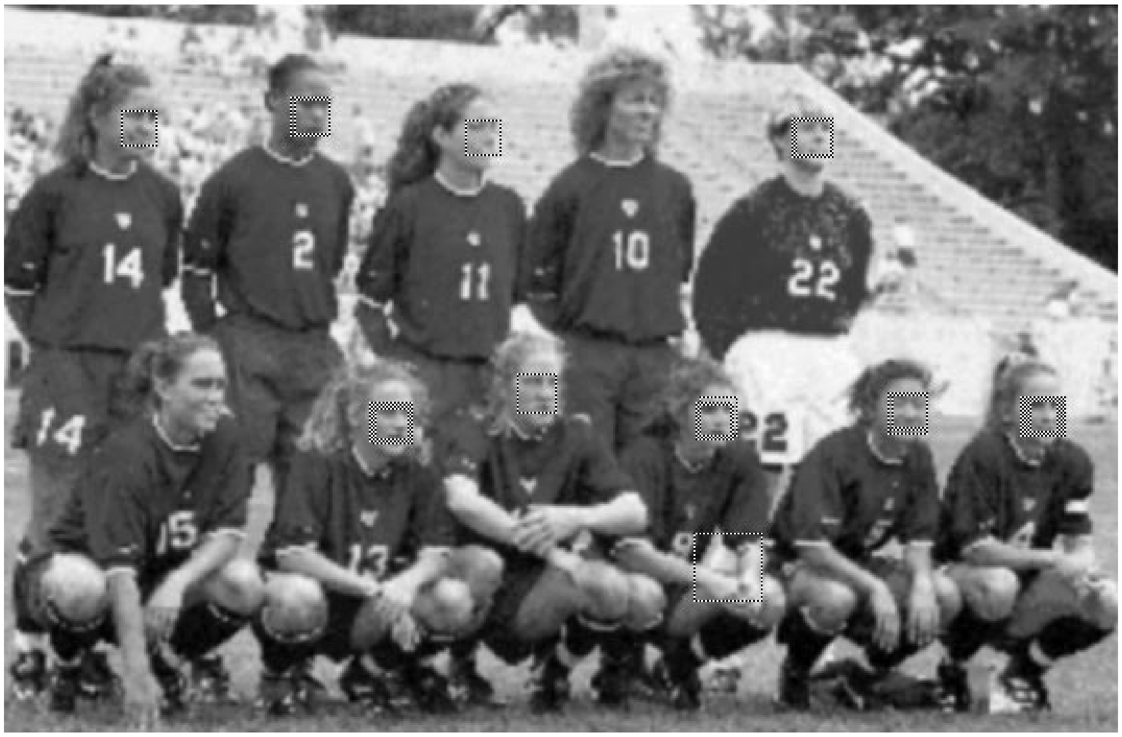


Figure 3-4: **Top:** Image with one false detect on the soccer ball. **Bottom:** Image with two missed faces (top row second from right, and bottom row first from left) and one false detect (arms of bottom row third from right).

tain background pixels that contribute to unwanted structure encoded in the model. To "unlearn" this irrelevant knowledge, we require a much larger number of training samples with the same foreground appearance to reveal that the background pixel values do not affect the example pattern's identity. One can use several mask shapes instead of a single mask pattern to help reduce unwanted background structure introduced into our distribution-based model. During training, we choose the best fitting mask to eliminate irrelevant background pixels from each face pattern in the example database. We then appropriately align each masked face pattern with a fixed outline to model the face distribution in a common view-based feature space. During detection, we try all mask patterns and their corresponding alignment transformations on each test pattern and choose the best output as the match result.

3.4 New Application 2: Human Eye Detection

We present next an application for finding a different class of spatially well-defined objects — human eyes, to demonstrate that our proposed object and pattern class detection approach works well for more than just human faces. Although human eyes appear less varied as a target class than human faces, eye detection is still a challenging problem because even at moderate resolution, there can be a lot more natural background patterns that resemble eyes than faces. To detect human eyes independent of more global structures like human faces, a successful eye finder must not only correctly identify all isolated eye patterns, it must reject a wide range of distractor patterns as well.

We built an eye detection system using the same distribution-based modeling cum learning-based techniques described in our original face detection example. Structurally, the two systems are very similar. Like the original face system, the eye finder uses a "convolution-like" matching paradigm to search the input image for human eye patterns over multiple scales. The embedded pattern identification procedure represents the target pattern class and a "near-miss" distractor class with a Gaussian mixture distribution-based model. The final pattern identification stage uses a trained multi-layer perceptron net to combine feature vectors of 2-value distances into a single similarity measure for classifying target and background patterns. Functionally, our system detects isolated and upright oriented human eye patterns in cluttered scenes independent of other facial features.

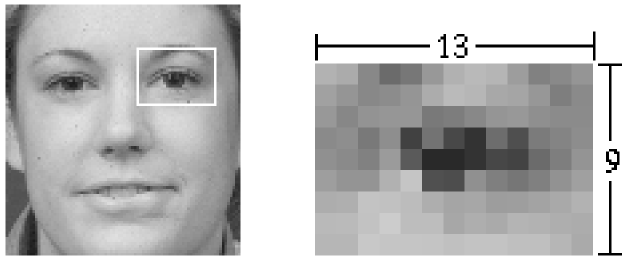


Figure 3-5: The "canonical eye" structure our system uses for finding human eye patterns. It corresponds to a fixed aspect-ratio rectangular patch of the human face, tightly enclosing the left or right eye. We model the distribution of canonical eye patterns in a normalized 13×9 pixel view-based feature space.

3.4.1 Implementation Details

Figure 3-5 shows the "canonical" eye structure our system uses for pattern matching. It corresponds to a fixed aspect-ratio rectangular patch of the human face, tightly enclosing the left or right eye. Because isolated left and right eye patches are structurally almost identical in the image domain, our implementation uses a common model for both pattern classes. To approximate the target class, we collect a data sample of 1114 real eye patterns from images in our frontal face training database. We further increase the number of target examples by artificially generating 5 virtual views from each real eye pattern in our data sample as follows: Each real eye pattern is rotated in-plane by $\pm 5^\circ$ to produce 2 virtual views, and the same is done for its mirror image to produce another 3 virtual views. Our final human eye data set contains altogether 6684 left and right eye patterns, which we use (1) as empirical data to approximate the target class in our distribution-based modeling scheme, and (2) as training examples to synthesize an appropriate similarity function for matching test patterns against the target model, as we had done earlier in both our face detection systems.

Like faces, human eyes are highly structured and relatively stable image patterns. One can therefore reasonably assume that our "canonical" eye target class is continuous and smoothly varying in a view-based feature space, which makes such a feature space suitable for tightly modeling the target distribution. In order to detect human eyes in cluttered scenes where even human faces are fairly small, we choose a moderate resolution intensity normalized image feature space of 13×9 pixels to represent our "canonical eye" target distribution. We decided not to use a lower resolution feature space for modeling, because further scale reduction can obscure essential details that distinguish isolated human eyes from other distractor patterns. In this example, one does not need to mask away "irrelevant" pixels in our chosen view-based feature space, because every pixel in a "canonical" eye patch corresponds to a predictable part of the human face, and contributes to the local image description of a human eye.

Our distribution-based target model contains 16 Gaussian "eye" clusters and 8 "near-miss" clusters. As before, we obtain the 16 "eye" clusters by performing elliptical *k-means* clustering on the enlarged training database of 6684 human eye patterns. In building this system, we arrived at 16 "eye" clusters by increasing the number of Gaussians in our target distribution model, until the classification error rate on an initial *training* database of "eye" and "non-eye" patterns fell to within 5%. We obtain the 8 "near-miss" Gaussians also by clustering a specially chosen database of 8335 "eye-like" patterns, generated using our "boot-strap" example selection strategy. In modeling our "near-miss" pattern distribution, we were able to achieve an almost perfect classification rate using only 8 model clusters, on a final *training* set which includes the additional "non-eye" patterns selected by "boot-strapping".

The final pattern identification stage also uses a trained multi-layer perceptron net with 24 input unit pairs to classify test patterns from their computed distance feature vectors. Each input unit pair receives a 2-value distance measurement between the test pattern's view-based feature space location and one of the model centroids. We train the multi-layer perceptron net on an example database of over 19000 distance feature vectors and their corresponding output classes. The training database includes vectors from all the "eye" and "near-miss" patterns used for building our distribution-based model, as well as a random sample of "non-eye" distractor patterns from a few highly textured natural images.

3.4.2 Results

We have implemented a very preliminary eye finding system by applying only one "boot-strap" cycle to select relevant "non-eye" examples from a few highly textured natural images for training. Figures 3-6 to 3-9 show some sample result the current system produces. Notice that there are still a number of missed eyes and false alarms in several test images, especially among those images with dense background texture. We believe this is partly because even at moderate spatial resolution, one can still lose fine details that are essential

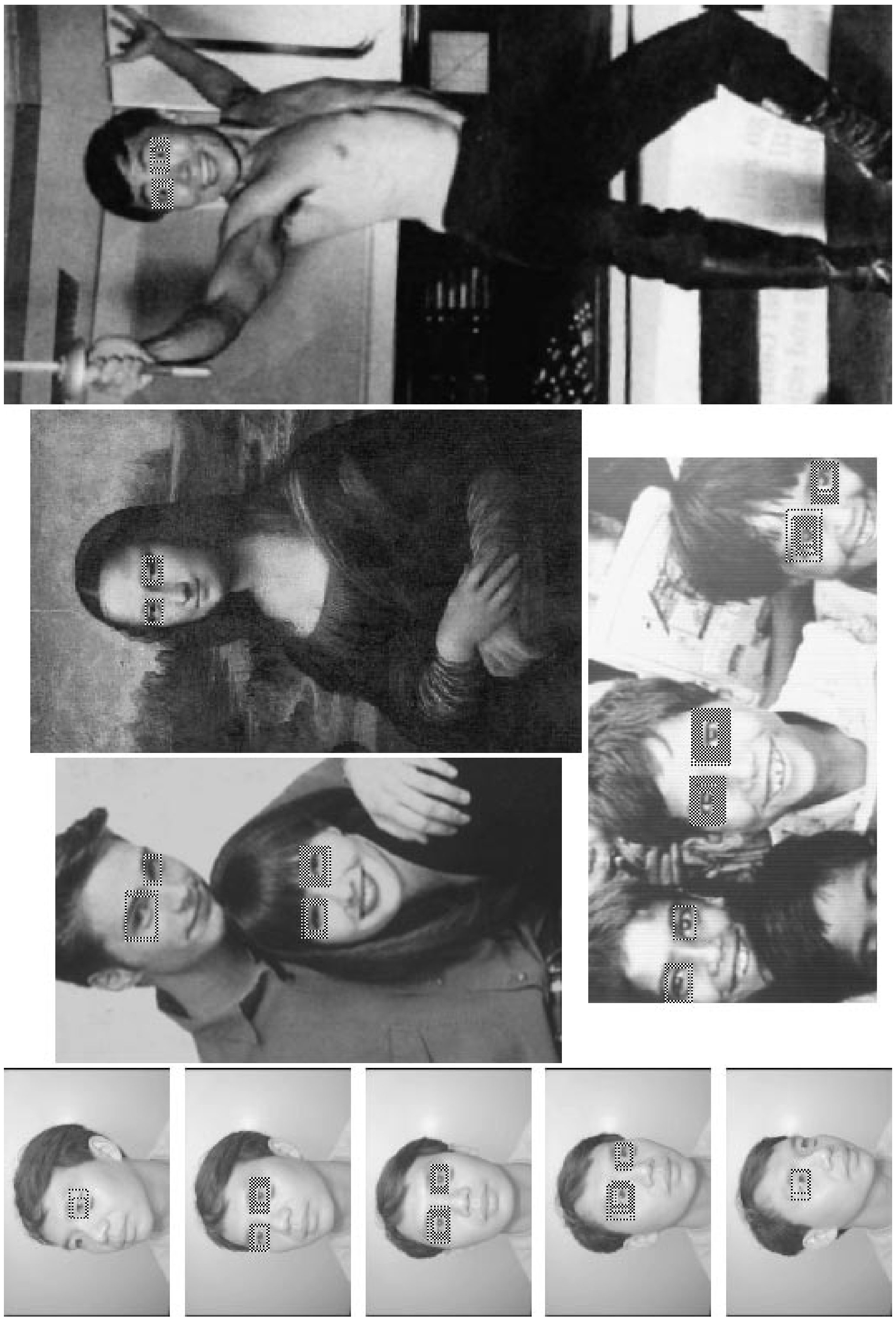


Figure 3-6: Eye detection results. The 5 pictures on the left show the system detecting eyes on frontal and off-plane rotated faces. Because we trained the system with only frontal eye patterns, the system fails at large rotation angles. The other pictures show the system finding eyes successfully in more complex images.

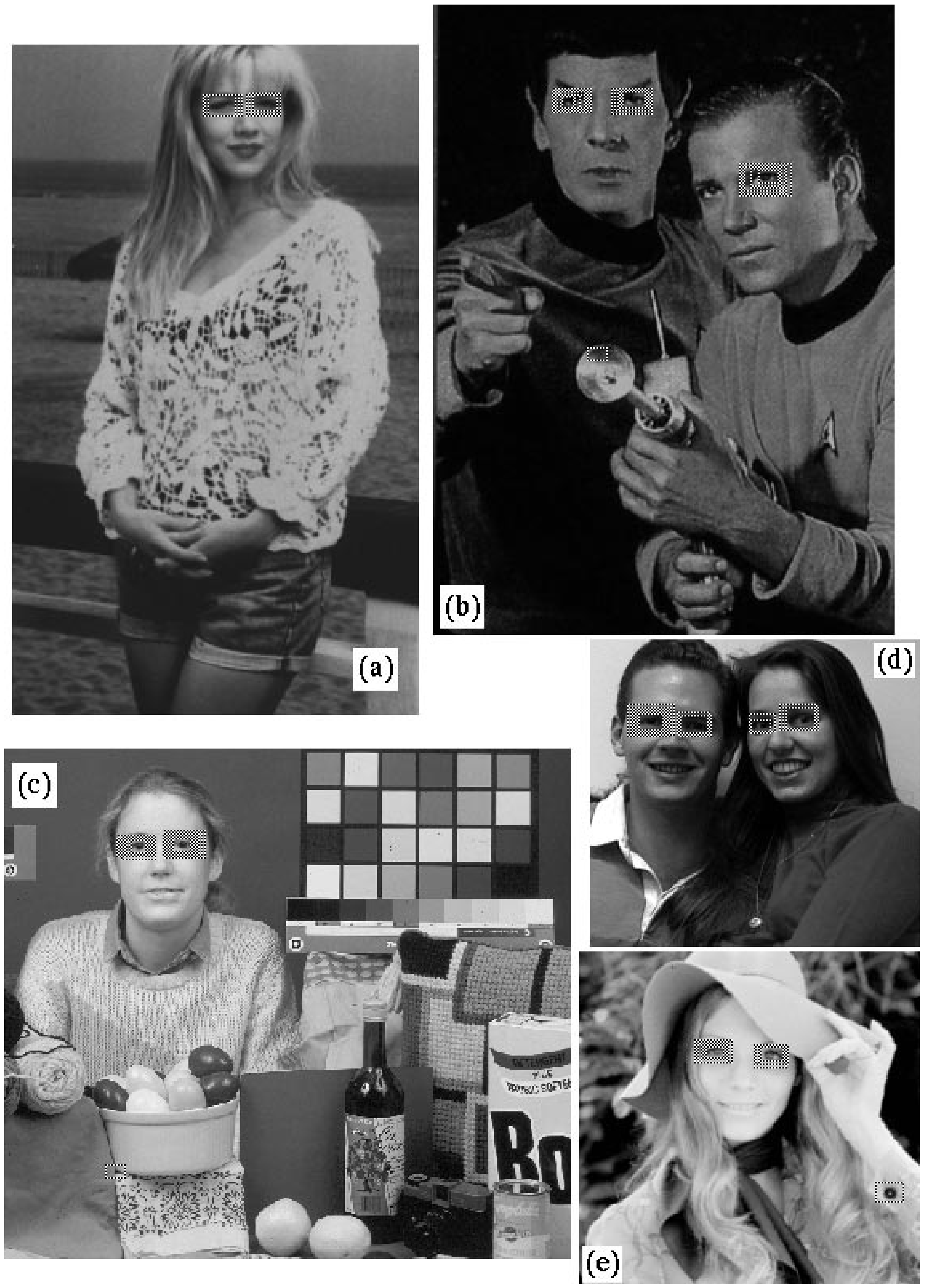


Figure 3-7: **(a):** The system successfully finds Kelly's eyes without any false detects on her textured blouse. **(b):** The system misses Kirk's right eye and makes one false detect on his phaser rifle. **(c):** The system makes only one false detect in this fairly complex scene (bottom left edge of bowl). **(e):** One false detect in this image where the system mistakes a button on the girl's sleeve for an eye.

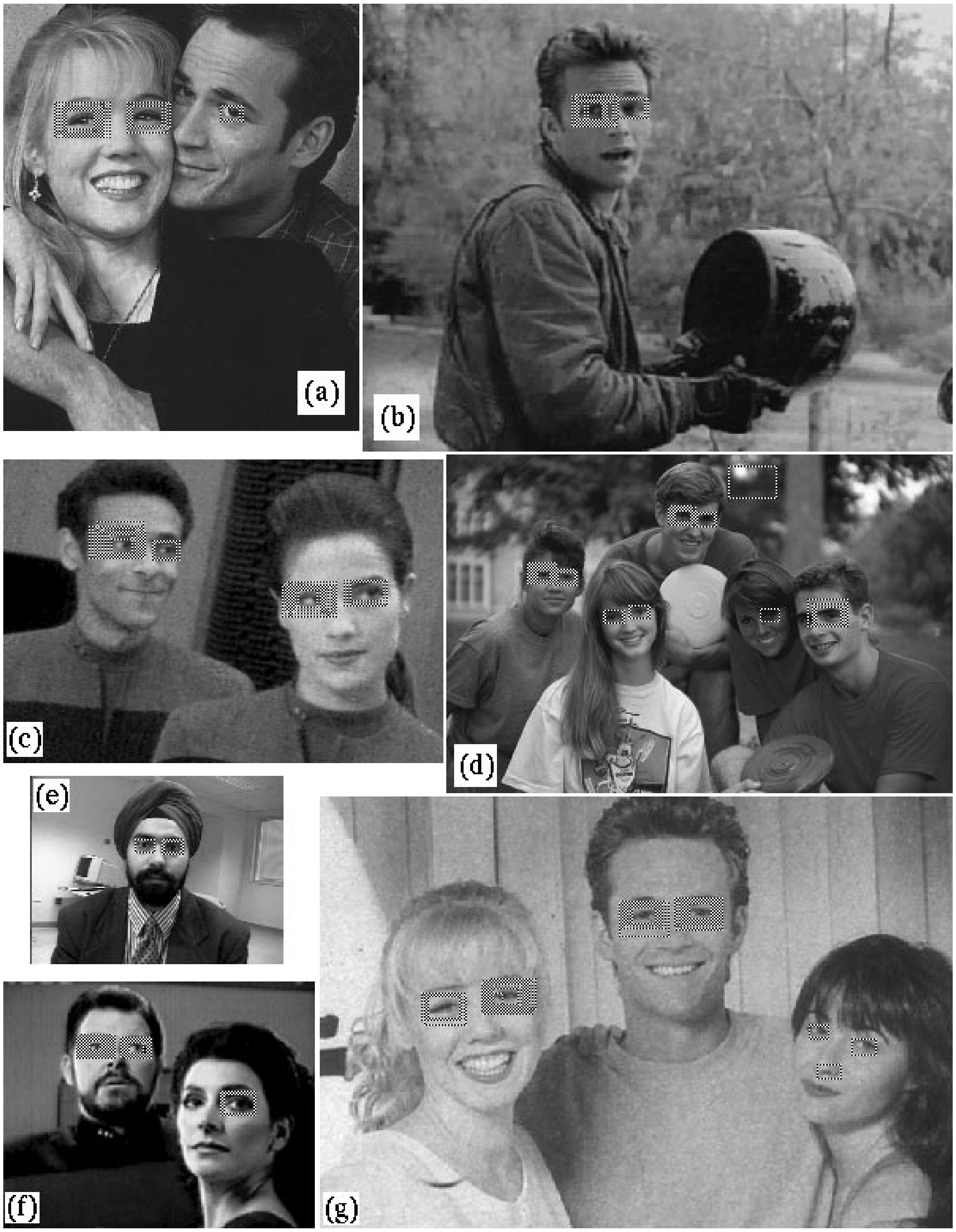


Figure 3-8: **(a):** The system misses Dylan's partly occluded right eye. **(d):** The system makes one false detect and misses the right eye of the two people on the right. **(f):** Misses Deanna Troi's right eye. **(g):** Brenda's nostril is mistaken for an eye. Notice that the system still detects Brenda's eyes successfully even though her head is significantly tilted.

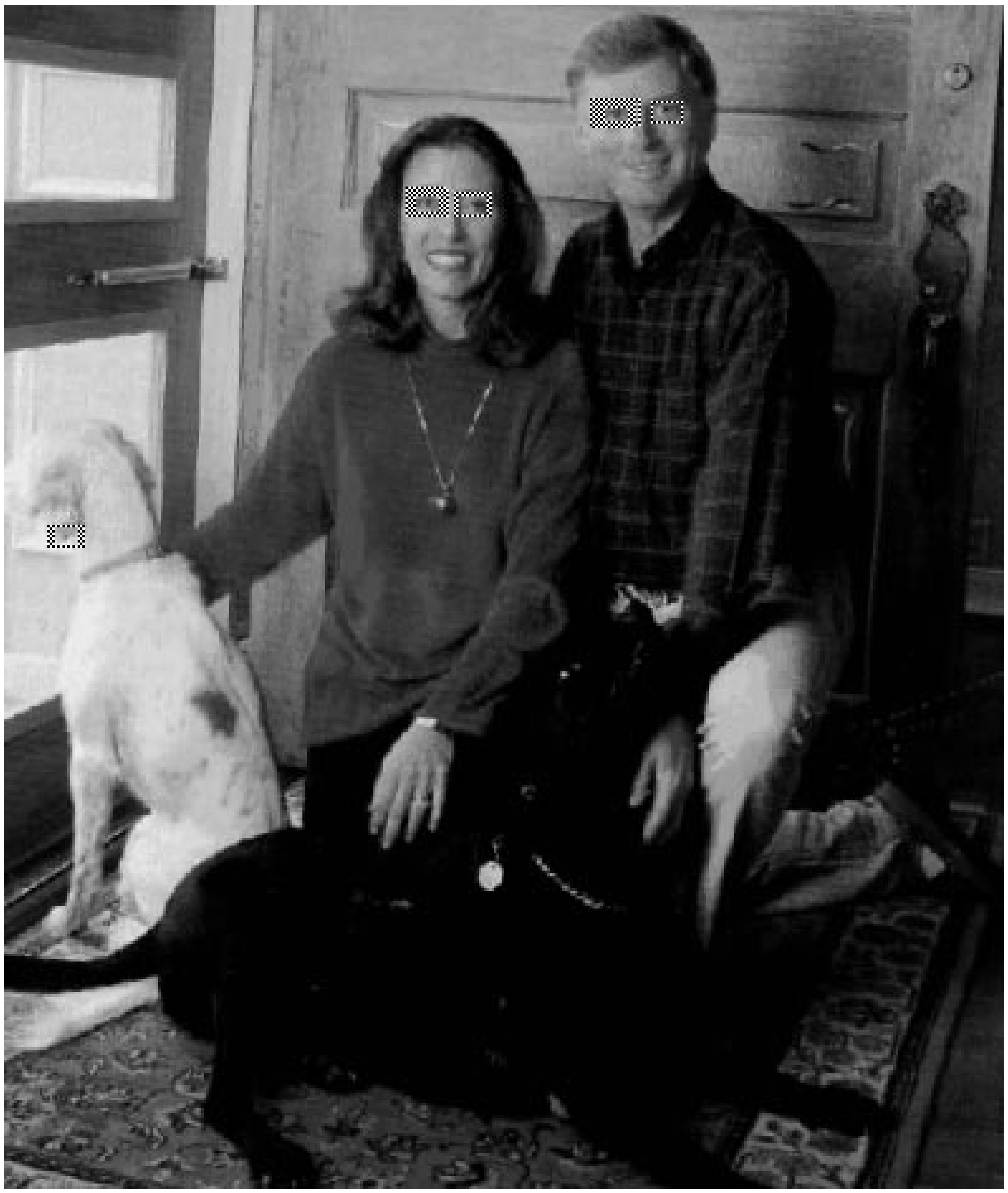


Figure 3-9: A reasonably complex image with potentially confusing textures on the carpet, Dan Quayle's shirt and the wooden door. There is only one false detect on the white dog's ear. The system does not detect the black dog's eye.

for distinguishing isolated eye patterns from very similar looking distractor image patterns. The system otherwise performs reasonably well on high quality frontal illuminated face images, and even detects eyes on vertical faces that are slightly rotated away from the image plane. Because our goal here is to show that our pattern detection approach generalizes to other target pattern domains other than faces, we were satisfied with the current results and have not attempted to improve the existing system with further training.

3.5 New Application 3: Hand Printed Digit Recognition

Our final example application deals with a pattern *recognition* task instead of a pattern *detection* problem like those described so far. We have argued in Chapter 1 that pattern *recognition* and pattern *detection* are both instances of a wider class of computer vision problems, called *pattern classification.* In fact, the pattern identification procedure within our object and pattern detection approach essentially performs a pattern classification task, by determining whether or not each test pattern belongs to the target class.

In this application, we demonstrate that the same pattern identification framework used by our object and pattern detection approach, can also be a viable solution for certain pattern recognition problems. Specifically, we look at the task of recognizing isolated hand-written digits in images. Recognizing hand-written digits by computer is a difficult pattern classification problem because even though each digit class has a common structural description, different hand-written instances of the same digit can still vary significantly in image appearance. Some factors influencing a particular hand-printed digit's appearance include the writer's writing style, the writer's mood, the writing instrument and environmental conditions.

Over the past few decades, there has been a lot of research in isolated hand-printed digit recognition techniques, with current state-of-art systems achieving near human-level recognition rates. Most of the best digit recognition results today are produced by learning-based systems [28] [54] [60] trained on extremely large hand-written digit databases, such as the *National Institute of Standards and Technology* (NIST) digit database which contains more than 200000 patterns. Recently, Simard et. al. [83] have also proposed a very successful learning-based technique for digit recognition that uses a new distance measure (tangent distance) which can be made locally invariant to any specified set of transformations on the input pattern. Their new learning technique simulates the effect of training the system with an enlarged digit database, consisting of the original database patterns and *virtual examples* generated by applying the specified invariant transformations on the original digits.

Although our task here involves building and experimenting with isolated hand-printed digit recognizers, we stress again that our primary goal is not to improve the state-of-art performance of isolated hand-written digit recognition systems. Rather, we wish to demonstrate that the underlying pattern identification framework in our object detection approach is indeed general enough to model and capture localized pattern variations, in a task that is essentially pattern *recognition* in spirit.

3.5.1 The United States Postal Service Digit Database

We perform our experiments using a hand-written digit database compiled by the United States Postal Service (USPS) for digit recognition research. The database contains 9298 digits extracted from the zip-code fields of hand-written postal addresses. Figure 3-10 shows some sample patterns from each of the 10 digit classes. Each digit is hand segmented from its zip-code field and normalized in size to fit within a 16×16 pixel bounding box. The individual digit images are also intensity normalized to fall within a $[-1, 1]$ range, where a -1 value corresponds to a "white" background.

In preparing the digit database, the USPS researchers have also divided the full database into a *training* sample of 7291 digits and a non-overlapping *test* set of 2007 digits. Both the training and test data samples are obtained from a different distribution of writers to make the learning task more challenging. In both data sets, each digit class makes up approximately one tenth the total number of digit samples.

3.5.2 Design of the Digit Recognizer

The digit recognition problem involves constructing a 10-class pattern classifier for identifying new input patterns as one of the 10 digit classes. We have argued in Chapter 1 that one can implement an N-way pattern classifier as N single-class pattern recognizers operating in parallel, with a special arbitration stage to resolve class label conflicts. In this application, we adopt a similar design approach by implementing our 10-class digit recognizer as 10 single-class digit recognizers. Each single-class digit recognizer identifies one of the 10 digit patterns from among the other 9 digit classes. Our arbitration scheme assumes that

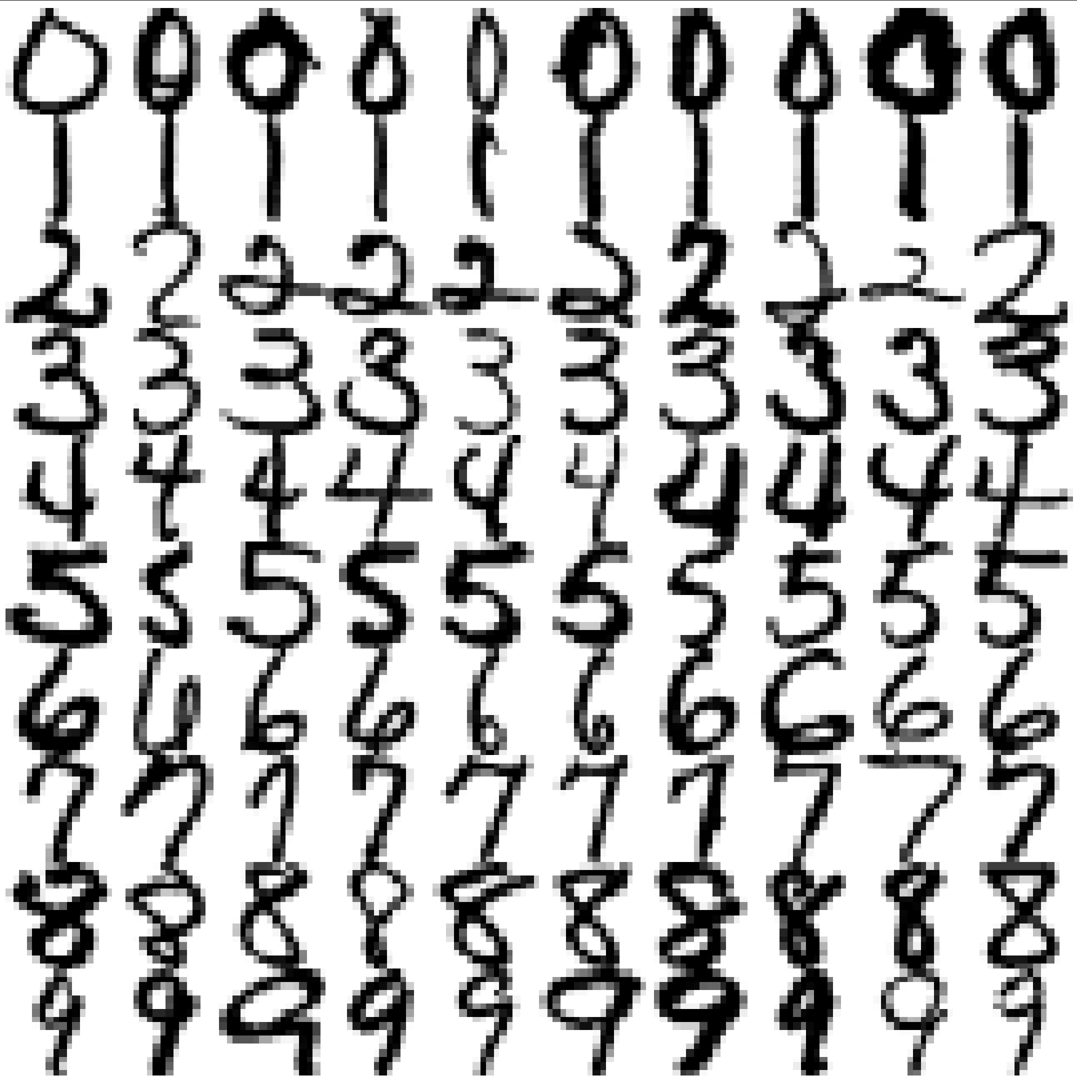


Figure 3-10: Some sample handwritten digit patterns from the United States Postal Services *training* database.

all input patterns belong to exactly one of the 10 digit classes, and simply returns the class label for the single digit recognizer with the strongest response.

We now describe how we build the individual single-class digit recognizers in our 10-class digit recognition system. Basically, we treat each single-class digit recognizer as a 2-way pattern classifier, similar in spirit to the pattern identification stage within our proposed object and pattern class detection framework. In this approach, the key design issue is to appropriately model the target pattern distribution of each digit class in a suitable feature space for pattern matching purposes. Because each digit class has a common overall image structure with minor shape variations between individual patterns, one can reasonably assume that the target pattern distribution for each digit class is continuous and smoothly varying in a view-based feature space. Our implementation models each target digit class directly in the original 16×16 pixel image feature space of normalized digit patterns. For our particular task, we do not need to mask away "irrelevant" pixels in our chosen view-based feature space, because all the hand-segmented digit patterns we are dealing with do not contain unwanted background structure.

In the distribution-based modeling stage, each single-digit recognizer approximates its target distribution with patterns from the USPS *training* database that belong to its target digit class. Because the USPS training database contains, on the average, fewer than 1000 pattern samples per digit class in a 256 dimensional view-based feature space, our actual distribution-based model represents each target pattern class with only 4 multi-dimensional Gaussian clusters to avoid over-fitting the available data with too complex a model. So, each single-class digit recognizer in our overall system has a distribution-based model that contains only 4 Gaussian mixture clusters for representing its target digit class.

We use a slightly different "boot-strap" procedure to obtain "near-miss" pattern samples, and to synthesize "near-miss" distribution models for the 10 single-class digit recognizers. Recall that in this demonstration, we only have available 7291 positive and negative training examples from the USPS *training* database to construct each single-digit classifier. Because we are using the USPS *training* database as our sole data source, one can expect to find only a very small number of "useful" distractor patterns to approximate the "near-miss" distribution for each digit class. In fact, there may not even be enough "useful" distractor patterns for each digit class to reasonably construct a Gaussian mixture "near-miss" distribution model. One solution is to have each single-digit recognizer treat all its non-target

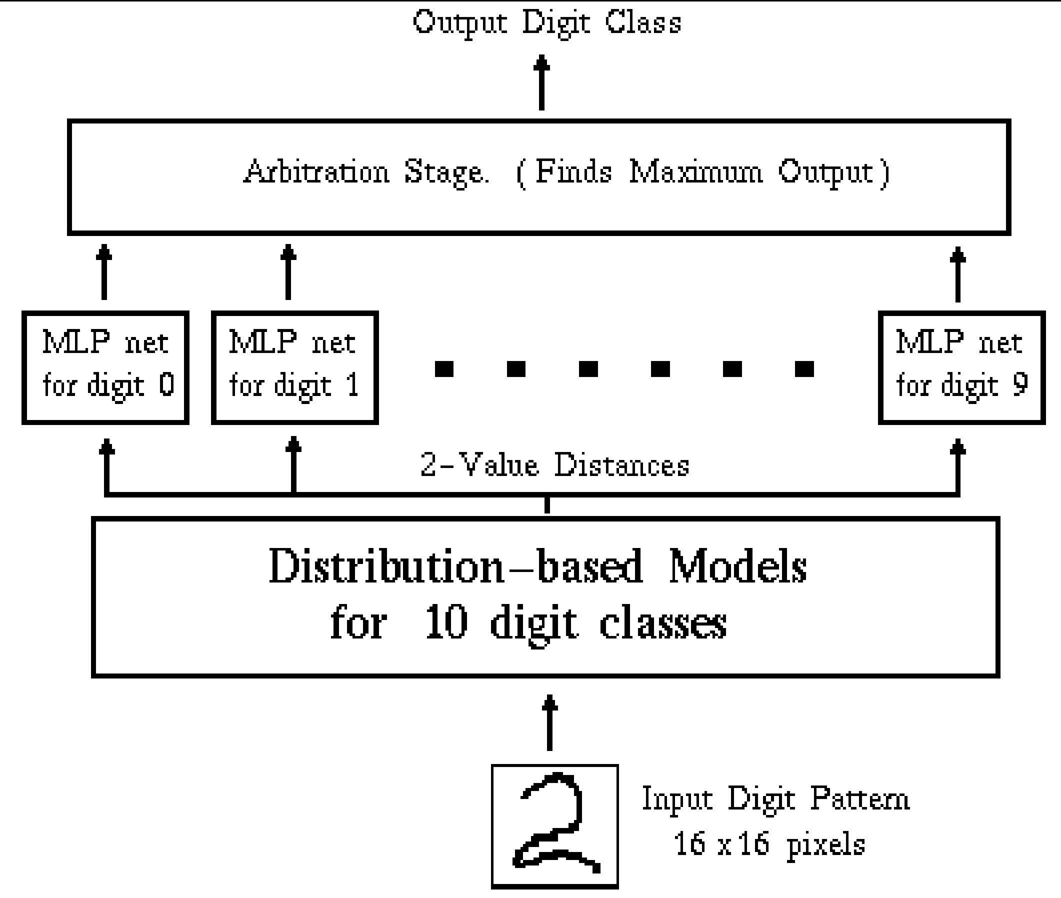

Output Digit Class
Arbitration Stage. (Finds Maximum Output)
MLP net for digit 0
MLP net for digit 1
MLP net for digit 9
2-Value Distances
Distribution-based Models for 10 digit classes
Input Digit Pattern 16 x 16 pixels

Figure 3-11: Design overview of our 10-class digit recognizer. We implement our 10-class digit recognizer as 10 single-class digit recognizers, each separating a given digit from the other 9. For each digit class, we construct a distribution-based model with 4 multi-dimensional Gaussian clusters. The MLP classifier for each digit recognizer receives 2-value distances from all 4 model clusters belonging to its target digit class. It also receives 2-value distance measurements from "relevant" clusters in the other 9 digit classes that help model its "near-miss" distribution. When classifying new digit patterns, the arbitration stage returns the class label of the recognizer with the strongest response.

digits in the USPS *training* database as "useful" distractor patterns. Each single-digit recognizer can then have a "near-miss" distribution model with 36 Gaussian clusters, obtained directly from the target class models of the other 9 single-digit recognizers. Unfortunately, such an approach leads to a set of very high dimensional learning problems in the final pattern classification stage, where we must now train each digit recognizer to identify its target digit class from input feature vectors of 40 2-value distance measurements.

We instead adopt an intermediate approach that also uses additional non-target digit patterns from the USPS *training* database to help approximate the "near-miss" pattern distribution for each digit class, without indiscriminately using the entire set of non-target patterns. Basically, the idea is to use only those non-target pattern samples near each actual "useful" distractor pattern to help locally approximate the "near miss" distribution. The intermediate technique for refining *each* single-digit recognizer works in two steps. In **step one**, we collect all the *false positive* mistakes each initial single-digit recognizer (i.e. whose distribution-based model contains only positive clusters describing the target class distribution) makes on the USPS *training* digit database. This step is identical to the *example selection* phase in our original "boot-strap" procedure. In **step two**, we approximate the local "near-miss" distribution near every *false positive* example obtained from **step one**, using only a "relevant" subset of Gaussian clusters from the other 9 target class models. We determine which Gaussian clusters are "relevant" for the given single-digit recognizer as follows: For each *false positive* example from **step one**, we look up its actual digit class and pick the nearest Gaussian cluster from its digit class model to approximate the local "near-miss" distribution. The set of all clusters so chosen approximates the particular recognizer's overall "near-miss" distribution, and the *final* distribution-based model contains Gaussian clusters that describe both the recognizer's target digit distribution and its "near-miss" distribution.

In the final pattern identification stage, each single-digit recognizer uses a trained multilayer perceptron net to identify instances of its target digit class, based on distance feature measurements between the input digit pattern and the recognizer's *final* distribution-based model. Each input feature vector is an ordered set of 2-value distances between the given test pattern's location and all the recognizer's model centroids in the normalized 16×16 pixel view-based feature space. We train each single-digit recognizer on distance feature vectors with appropriate output class labels from all the digit patterns in the USPS *training* database.

3.5.3 Comparison with Other Techniques

As indicated earlier, our primary goal in this third application is to show that our object *detection* framework also generalizes well to pattern *recognition* tasks. To do this, we compare our final system's digit recognition performance against two other systems based on more traditional digit recognition techniques. Like the original system we implemented, both new systems are also organized as 10 single-digit recognizers with an arbitration stage that simply returns the digit class whose recognizer responds most strongly. The first system uses a classical radial-basis function network with spherical Gaussian centers for each individual digit recognizer. We compute the system's spherical Gaussian centers and their corresponding variances by separately performing *k-means* clustering on each digit class in the USPS *training* database. Our approach is similar to Moody and Darken's algorithm for finding RBF centers [62] in a multi-class data sample. The second system is also Gaussian RBF based with a different method for computing RBF centers. We use a recently developed technique, called *Support Vector Algorithms* [26], to determine the number and location of Gaussian centers for each RBF digit classifier. In building both systems, we use the USPS *training* database as our only source of digit patterns, without artificially enlarging the database with virtual examples. We shall describe the two system in greater detail below.

Classical Spherical Gaussian RBFs

We begin by first describing the classical Gaussian RBF system. A d-dimensional spherical Gaussian RBF network with K centers has the mathematical form:

$$\begin{aligned} g(\vec{x}) &= \sum_{i=1}^{K} w_i \mathcal{G}_i(\vec{x}) - b \\ &= \sum_{i=1}^{K} w_i \frac{1}{(2\pi)^{d/2}\sigma_i^d} \exp\left(-\frac{||\vec{x}-\vec{c_i}||}{2\sigma_i^2}\right) - b \end{aligned}$$

where $\mathcal{G}_i$ is the i^{th} Gaussian basis function with center $\vec{c_i}$ and variance σ_i^2. The weight coefficients w_i combine the Gaussian terms into a single output value and b is an arbitrary bias term. In general, building a Gaussian RBF network for a given learning task involves (1) determining the total number of Gaussian basis functions to use for each output class and for the entire system, (2) locating the Gaussian basis function centers, (3) computing the cluster variance for each Gaussian basis function, and (4) solving for the weight coefficients and bias in the summation term. One can implement a 2-way pattern classifier on input vectors $\vec{x}$ as a Gaussian RBF network by defining an appropriate output threshold that separates the two pattern classes.

In this first system, we implement each individual digit recognizer as a spherical Gaussian RBF network, trained with a classical RBF algorithm. Given a specified number of Gaussian basis functions for each digit class, the algorithm separately computes the Gaussian centers and variances for each of the 10 digit classes to form the system's RBF kernels. The algorithm then solves for an optimal set of weight parameters between the RBF kernels and each output node to perform the desired digit recognition task. Our actual training process constructs all 10 digit recognizers in parallel so one can re-use the same Gaussian basis functions among the 10 digit recognizers. To avoid overfitting the available training data with an overly complex RBF classifier connected to every Gaussian kernel, we use a "boot-strap" like operation that selectively connects each recognizer's output node to only a "relevant" subset of basis functions. The idea is similar to how we choose relevant "near-miss" clusters for each individual digit recognizer in the original system. The full training procedure proceeds as follows:

1. The first training task is to determine an appropriate number of Gaussian kernels for each digit class. This information is needed to initialize our clustering procedure for computing Gaussian RBF kernels. Because the *support vector algorithm* in the second system automatically computes an "optimal" number of RBF kernels for each digit class, we simply use the same figures from the second system to initialize our clustering procedure (see Table 3.3).

2. Our next task is to actually compute the desired number of Gaussian kernels for each digit class. We do this by separately performing classical *k-means* clustering on each digit class in the USPS *training* database. Each clustering operation returns a set of Gaussian centroids and their respective variances for the given digit class. Together, the Gaussian clusters from all 10 digit classes form the system's RBF kernels.

Digit Class	0	1	2	3	4	5	6	7	8	9
Number of Kernels	172	77	217	179	211	231	147	133	194	166

Table 3.3: Number of Gaussian kernels in each digit class used for initializing the classical RBF digit recognition system. These are the number of distinct example patterns from each class that the second system chooses as *support vectors.*

3. For each single-digit recognizer, we build an *initial* RBF network using only Gaussian kernels from its target class. We then separately collect all the false positive mistakes each *initial* digit recognizer makes on the USPS *training* database.

4. In the final training step, we augment each *initial* digit recognizer with additional Gaussian kernels from outside its target class to help reduce mis-classification errors. We determine which Gaussian kernels are "relevant" for each recognizer as follows: For each *false positive* mistake the *initial* recognizer makes during the previous step, we look up the mis-classified pattern's actual digit class and include the nearest Gaussian kernel from its class in the "relevant" set. The *final* RBF network for each single-digit recognizer thus contains every Gaussian kernel from its target class, and several "relevant" kernels from the other 9 digit classes. Because our *final* digit recognizers have fewer weight parameters than a naive system that fully connects all 10 recognizers to every Gaussian kernel, we expect our system to generalize better on new data.

Support Vector Gaussian RBFs

In the classical RBF system, we used a clustering technique that computes Gaussian kernels irrespective of the exact recognition task to be solved. One can view the clustering operation as building a separate distribution-based model for each digit class using spherical Gaussian clusters. The RBF digit recognizers classify new digit patterns by determining how "similar" they are to each of the 10 digit manifolds, based on distance measurements to the Gaussian kernels. In this second system, we build a similar Gaussian RBF-based 10 class digit recognizer using a different initialization technique, called the *support vector algorithm* [26], that concentrates Gaussian kernels at feature space locations critical for the recognition task at hand. The *support vector algorithm* is a general procedure that sieves through example databases for useful data subsets relevant to a given learning task. The algorithm works for many different learning machine architectures, and the resulting data subsets (i.e. the

Digit Recognizer	0	1	2	3	4	5	6	7	8	9
# Support Vectors	274	104	377	361	334	388	236	235	342	263

Table 3.4: Number of support vectors for each each digit recognizer. Notice that for each digit recognizer, the *support vector set* contains both positive and negative example patterns, i.e. patterns from within and outside the target class. The same digit pattern can be a support vector for two or more recognizers. Table 3.3 shows the number of distinct patterns from each digit class selected as support vectors.

support vector sets) for different architectures are often almost identical. Interestingly, for RBF networks, the *support vector sets* also serve well as locations for Gaussian centers. We shall only briefly describe the *support vector algorithm* with particular emphasis on its role as a mechanism for defining and locating Gaussian kernels in RBF networks. The interested reader should refer to the following papers for further details: [102] [14] [26].

The *support vector algorithm* is based in part on the idea of *structural risk minimization* [102], whose motivation can be summarized follows: In example-based function-approximation learning, the goal is to synthesize an approximation function that (1) maps input examples onto their respective output values, and (2) reasonably predicts output values at input locations where no examples are available. This second property is commonly known as the learner's *generalization* ability. Together, one can quantify the above two constraints in terms of a *risk* measure that depends on the number of training examples and the *VC-dimension* [100] [101] [1] (i.e. complexity) of the approximation function class. We refer the reader to [80] for a more detailed and mathematical treatment of *structural risk minimization* and function-approximation learning.

When available training data is limited, one must constrain the learning machine's structural complexity in order to minimize *risk* and generalize reasonably. *Structural risk minimization* chooses the function of "optimal" complexity from an approximation function class so that the resulting *risk* is minimal. The *support vector algorithm* essentially performs *structural risk minimization* on an approximation function class whose structure is a set of hyperplanes. For spherical Gaussian RBF networks, the algorithm minimizes *risk* by determining the number of Gaussian kernels that leads to best *generalization.* In our current RBF *support vector algorithm* formulation, we deal with a structure in which all Gaussian kernels must have the same fixed user-specified variance.

We use the *support vector algorithm* to construct 10 RBF-based single-digit recognizers with fixed Gaussian variances of $\sigma^2 = 38.4$, each trained to separate a given digit from

USPS Database	Classification Error Rate		
	Original	Classical RBF	Support Vector RBF
Training (7291 patterns)	0.33%	1.73%	0.01%
Test (2007 patterns)	5.33%	6.73%	4.88%

Table 3.5: 10-class digit recognition *error rates* for three different system architectures. The first system is based on the pattern identification framework within our proposed object and pattern class detection approach. The other two are the Gaussian RBF-based systems we trained, one with a classical RBF algorithm and the second with the *support vector algorithm*. The test results show that our proposed pattern identification framework compares reasonably well against classical digit recognition architectures, hence suggesting that it is indeed general enough to even model and capture pattern variations in problem domains that are essentially pattern *recognition* in spirit.

the other 9. We experimented with several σ^2 values and chose the setting with the best recognition result on the USPS *test* database. For each single-digit recognizer, the *support vector algorithm* selects a set of positive and negative example digit patterns from the USPS *training* database as Gaussian kernel centers. Table 3.4 shows the number of support vectors selected for each recognizer. Notice that the same digit pattern can be chosen as a support vector for two or more digit recognizers. In Table 3.3, we show the number of distinct patterns from each digit class that have been selected as support vectors. We use these figures in the first classical RBF system as an appropriate number of Gaussian kernels for each digit class.

3.5.4 Results

We ran our original 10-class digit recognizer and the two spherical Gaussian RBF-based systems described above on the USPS *test* digit database. For each test pattern, the arbitration procedure in all three systems simply returns the digit class whose recognizer gives the strongest response. Table 3.5 shows the 10-class digit recognition *error rates* for our original system and the two RBF-based systems. The results should also be compared with values achieved on the same *test* database by a five-layer multi-layer perceptron net, 5.1% [29], a two-layer multi-layer perceptron net, 5.9%, and human performance, 2.5% [16]. On the whole, the test results show that our proposed pattern identification framework compares reasonably well against classical digit recognition architectures in classifying digit patterns, hence suggesting that our framework is indeed general enough to even model and capture pattern variations in problem domains that are essentially pattern *recognition* in spirit.

Chapter 4

Active Example Selection for Function Approximation Learning

One key feature in our proposed object and pattern detection approach is the "boot-strap" idea of sieving through extremely large training data sets for useful examples relevant to the learning problem. We have seen in our face and eye detection scenarios that it can be very difficult to manually obtain a small and representative sample of "non-face" and "non-eye" patterns as training examples. Without a reasonable example selection strategy, the negative example sets in both these scenarios can grow hopelessly large, making the learning problems intractable.

In this chapter, we take a more formal look at the problem of selecting high utility examples for training pattern detection systems. The example selection problem falls under a newly emerging general area of research, called *active learning*, that investigates how learners can pose intelligent queries to teachers under various learning scenarios, to achieve "better" learning results. Active learning differs from traditional example-based learning paradigms in the following way: Rather than passively accepting training examples that randomly describe a target concept, an *active* learner uses information derived from its current state and prior knowledge about the target concept to intelligently gather useful examples from specific input space locations for further training. By carefully generating intelligent queries instead of performing random sampling, one can expect active learning techniques to have faster learning rates and better approximation results than traditional example-based learning algorithms.

Our main focus in this chapter is on active example selection strategies for a *function approximation* based learning framework. Specifically, we address the following three questions:

1. Given a function approximation based learning task and some prior information about the target function, are there principled strategies for selecting useful training data in some "optimal" fashion?

2. Assuming such principled data selection strategies do exist, do these active strategies require fewer examples than classical learning techniques to approximate target functions to the same degree of accuracy?

3. Can one directly apply these active example selection strategies to real-world function approximation learning tasks like our pattern detection scenarios, or easily adapt them into more feasible forms without losing too much of their original flavor?

We begin by proposing an active example selection formulation for function approximation learning to show that one can indeed select high utility examples for a given task in a principled and "optimal" fashion. While the formulation we propose is computationally intractable in its original form for a wide range of approximation function classes, we see it as a possible benchmark for evaluating other active example selection schemes. We next show how the general formulation can be used to derive precise data selection algorithms for three specific approximation function classes: (1) unit step functions, (2) polynomial approximators and (3) Gaussian radial basis function networks. For all three function classes, we provide either theoretical or empirical results suggesting that the active strategy learns the target function with fewer data examples than random sampling. Finally, we consider a reduced version of the original active learning formulation that essentially hunts for new data where approximation "error bars" are high. We show how such a scheme, with minor modifications, leads to the "boot-strap" example selection strategy we have adopted in our object and pattern class detection approach. Although the "boot-strap" strategy loses some of the original active learning flavor and may thus be "sub-optimal" in its choice of new examples, we show empirically that it still outperforms random sampling in training a frontal face detection system, and is therefore still an effective means of dealing with unmanageably large data sets to make learning tasks tractable.

4.1 Background and Approach

We start with an overview of *active learning* and related work. Active learning has appeared in various forms throughout *knowledge engineering* and *machine learning* literature. One early implementation can be found in certain expert systems, where an important component of learning relies on issuing queries to the instructor. For example, Sammut and Banerji [78] use queries about specific examples as part of a strategy for efficiently learning a target concept. Shapiro's Algorithmic Debugging System prompts the user with a variety of query types to locate errors in Prolog programs [82]. In computational learning theory, several types of *active learning* queries have also been defined (see for example [6]) and compared with Valiant's *probably approximately correct* (PAC) model of concept identification under random sampling [99]. Angluin [5], for example, has shown that there are concept classes that can be efficiently learnt with membership and equivalence queries, but not with random sampling in Valiant's PAC model.

Some early connectionist approaches toward active learning include: Ahmad and Omohundro [4] on training networks by selective attention; Hwang et. al. [47] on a query-based neural network learning scheme that generates queries near classification boundaries; Plutowski and White [70] on an efficient feedforward network training technique that selects new training examples with maximum potential utility from among available candidate examples. Both Hwang et. al. [47] and Plutowski et. al. [70] choose new training examples according to information derived from a partially trained network.

Plutowski and White [70] examines the learning task from a more general function approximation standpoint, viz., approximating a target function, $g(\vec{x})$, using a network output function, $F(\vec{w},\vec{x})$, parameterized by weights $\vec{w}$. They design their criteria for selecting new examples to meet two objectives: (1) to maximize the accuracy of fit between network output, $F(\vec{w},\vec{x})$, and the target function, $g(\vec{x})$, and (2) to minimize the approximation's unreliability in the presence of noise. The paper quantifies the above two considerations by proposing an *Integrated Mean Squared Error* (IMSE) measure to be minimized:

$$\begin{aligned} IMSE(\mathbf{x}^n) &= \int\int [g(\vec{x}) - F(l_n(\mathbf{x}^n,\mathbf{y}^n),\vec{x})]^2 \gamma^n(d\mathbf{y}^n|\mathbf{x}^n)\mu(d\vec{x}) \\ &= \int E[(g(\vec{x}) - F(\vec{w_n},\vec{x}))^2|\mathbf{x}^n]\mu(d\vec{x}), \end{aligned}$$

where $\mathbf{x}^n, \mathbf{y}^n$ are the current n pairs of input-output training examples, $l_n(\mathbf{x}^n, \mathbf{y}^n) = \vec{w}_n$ is the learning rule that begets network weights $\vec{w}_n$ from the training examples, $\gamma^n(\mathbf{y}^n|\mathbf{x}^n)$ is the conditional probability of output distribution $\mathbf{y}^n$ given input distribution $\mathbf{x}^n$, $E[\cdot|\mathbf{x}^n]$ is conditional expected network output mean squared error given input distribution $\mathbf{x}^n$, and μ is the probability distribution over the input space $\vec{x}$. To select the next training example, the learning algorithm samples at the next input location $\vec{x_{n+1}}$ that maximally decreases the $IMSE$. Unfortunately, an obvious problem with the approach is that both the $IMSE$ and the analytic expression for its decrement (not shown) assume a *known* target function $g(\vec{x})$. This is seldom a reasonable assumption in real learning scenarios where the target function is *unknown*.

4.1.1 Regularization Theory and Function Approximation — A Review

Our main focus in this chapter is on function approximation based active learning. We briefly review *regularization theory* as a lead in to our active learning formulation.

Let $\mathcal{D}_n = \{(\vec{x_i}, y_i) \in \Re^d \times \Re | i = 1, \ldots, n\}$ be a set of n data points obtained by sampling a function g, possibly in the presence of noise, where d is the input dimensionality. The function approximation task is to recover g, or at least obtain a reasonable estimate of it, by means of an approximator $\hat{g}$. Clearly, the problem is ill-posed [43] because there can be an infinite number of functions that pass through those data points. Some constraints are thus needed to transform the problem into a well-posed one. The *regularization* approach [94] [95] [63] [8] selects a function $\hat{g}$ that minimizes the following functional:

$$H[\hat{g}] = \sum_{i=1}^{n}(y_i - \hat{g}(\vec{x_i}))^2 + \lambda \parallel P\hat{g} \parallel^2 . \tag{4.1}$$

The first term of Equation 4.1 penalizes discrepancies between the solution, $\hat{g}$, and the observed data. The second term, usually called a *stabilizer*, embodies a priori knowledge about the *smoothness* of the solution. P is a constraint operator, usually a linear differential operator, and $\parallel \cdot \parallel$ stands for a *norm* on the function space containing $\hat{g}$, usually the L_2 norm. Together, they favor functions that do not vary too quickly on $\Re^d$. The *regularization parameter*, λ, determines the trade-off between the two terms — data reliability and prior beliefs. Poggio and Girosi have shown that the solution to Equation 4.1 has the following simple form:

$$\hat{g}(\vec{x}) = \sum_{i=1}^{n} c_i G(\vec{x}; \vec{x_i}) + p(\vec{x}), \tag{4.2}$$

where G, p and the coefficients c_i, can all be derived from the constraint operator P, the n data points $(\vec{x_i}, y_i)$, the stabilizer and some boundary conditions (see [71] for details).

For our purpose, it is convenient to adopt a probabilistic interpretation of *regularization* that treats the function $\hat{g}$ and the data set $\mathcal{D}_n$ as random, dependent variables (see [72]). Using Bayes rule, we can express the conditional probability of the function $\hat{g}$ given examples $\mathcal{D}_n$, $\mathcal{P}(\hat{g}|\mathcal{D}_n)$, in terms of the prior probability of $\hat{g}$, $\mathcal{P}(\hat{g})$, and the conditional probability of $\mathcal{D}_n$ given $\hat{g}$, $\mathcal{P}(\mathcal{D}_n|\hat{g})$:

$$\mathcal{P}(\hat{g}|\mathcal{D}_n) \propto \mathcal{P}(\mathcal{D}_n|\hat{g})\mathcal{P}(\hat{g}). \tag{4.3}$$

Equation 4.3 relates to the regularization functional of Equation 4.1 as follows: Suppose noise at each of the n data points is *identically independently Gaussian distributed* with variance σ^2. The conditional probability, $\mathcal{P}(\mathcal{D}_n|\hat{g})$, can be written as:

$$\mathcal{P}(\mathcal{D}_n|\hat{g}) \propto \exp\left(-\sum_{i=1}^{n} \frac{1}{2\sigma^2}(y_i - \hat{g}(\vec{x_i}))^2\right).$$

Similarly, if $\hat{g}$ is a stochastic process [59] [36], we can write $\mathcal{P}(\hat{g})$ as:

$$\mathcal{P}(\hat{g}) \propto \exp\left(-l \parallel P\hat{g} \parallel^2\right),$$

where l is some fixed constant, P and $\parallel \cdot \parallel$ are as defined earlier. Equation 4.3 thus becomes:

$$\begin{aligned}\mathcal{P}(\hat{g}|\mathcal{D}_n) &= K e^{-\sum_{i=1}^{n} \frac{1}{2\sigma^2}(y_i - \hat{g}(\vec{x_i}))^2} \exp\left(-l \parallel P\hat{g} \parallel^2\right) \\ &= K \exp\left(-[\sum_{i=1}^{n} \frac{1}{2\sigma^2}(y_i - \hat{g}(\vec{x_i}))^2 + l \parallel P\hat{g} \parallel^2]\right)\end{aligned}$$

where K is some fixed constant. Taking natural logarithms on both sides and performing some additional algebra yields:

$$-2\sigma^2 \ln \mathcal{P}(\hat{g}|\mathcal{D}_n) + \ln K = \sum_{i=1}^{n}(y_i - \hat{g}(\vec{x_i}))^2 + 2\sigma^2 l \parallel P\hat{g} \parallel^2,$$

which is identically Equation 4.1 with $\lambda = 2\sigma^2 l$ and $H[\hat{g}] = -2\sigma^2 \ln \mathcal{P}(\hat{g}|\mathcal{D}_n) + \ln K$. So, by choosing a function $\hat{g}$ that minimizes $H[\hat{g}]$, *regularization* essentially maximizes the conditional probability $\mathcal{P}(\hat{g}|\mathcal{D}_n)$. In other words, it chooses:

$$\begin{aligned}\hat{g} \in \arg\min_f H[f] &= \arg\max_f \mathcal{P}(f|\mathcal{D}_n) \\ &= \arg\max_f \mathcal{P}(\mathcal{D}_n|f)\mathcal{P}(f),\end{aligned}$$

that is, an a-posteriori most probable function $\hat{g}$ given the set of examples $\mathcal{D}_n$.

4.1.2 A Bayesian Framework

The *active learning* problem for function approximation can be posed as follows: Let $\mathcal{D}_n = \{(\vec{x_i}, y_i) \in \Re^d \times \Re | i = 1, \ldots, n\}$ be a set of n data points sampled from an unknown target function g, possibly in the presence of noise, where d is the input dimensionality. Given an approximation function concept class, $\mathcal{F}$, where each $f \in \mathcal{F}$ has prior probability $\mathcal{P}_{\mathcal{F}}(f)$, one can use *regularization* techniques to approximate g from $\mathcal{D}_n$ (in the Bayes optimal sense) by means of a function $\hat{g} \in \mathcal{F}$. We want a strategy to determine at what input location one should sample the next data point, $(\vec{x_{n+1}}, y_{n+1})$, in order to obtain the "best" possible Bayes optimal approximation of the unknown target function g with our concept class $\mathcal{F}$.

One can use ideas from *optimal experiment design* [33] to approach the active data sampling problem in two stages:

1. **Define what we mean by the "best" possible Bayes optimal approximation of an unknown target function.** We propose an optimality criterion for evaluating the "goodness" of a solution with respect to an *unknown* target function, similar in spirit to the cost function, Equation 4.1, for a *known* target.

2. **Formalize mathematically the task of determining where in input space to sample the next data point.** We express the above mentioned optimality criterion as a cost function to be minimized, and the task of choosing the next sample as one of minimizing the cost function with respect to the input space location of the next sample point.

Earlier work by Cohn [24] and MacKay [58] have tried using similar *optimal experiment design* techniques to collect data with maximum information about the target function. Our work here differs from theirs in two respects. First, we use a different, and perhaps more general, optimality criterion for evaluating solutions to an unknown target function. Specifically, our optimality criterion considers both bias and variance components in the solution's *output* generalization error. In contrast, both MacKay and Cohn use a "less complete" optimality criterion that favors solutions with only small variance components in *model parameter* space. Second, we also examine the important sample complexity issue, i.e., does the active strategy require fewer examples than random sampling to approximate the target to the same degree of accuracy? After completion of this work, we learnt that Sollich [86] had also recently developed a similar formulation to ours, but his analysis is conducted in a statistical physics framework.

4.2 The Active Learning Formulation

In order to optimally select examples for a learning task, one should first have a clear notion of what an "ideal" learning goal is for the task. One can then measure an example's utility in terms of how well the example helps the learner achieve the goal, and devise an active sampling strategy that selects examples with maximum potential utility. In this section, we propose one such learning goal — to find an approximation function $\hat{g} \in \mathcal{F}$ that "best" estimates the *unknown* target function g. We then derive an example utility cost function for the goal and finally present a general procedure for selecting examples.

4.2.1 An Optimality Criterion for Learning an Unknown Target Function

Let g be the target function that we want to estimate by means of an approximation function $\hat{g} \in \mathcal{F}$. If the target function g were known, then one natural measure of how well (or badly) $\hat{g}$ approximates g would be their *Integrated Squared Difference* (ISD) over the input space, $\Re^d$, or over some appropriate region of interest:

$$\delta(\hat{g}, g) = \int_{\vec{x} \in \Re^d} (g(\vec{x}) - \hat{g}(\vec{x}))^2 d\vec{x}. \tag{4.4}$$

In most function approximation tasks, the target g is unknown, so we clearly cannot express the quality of a learning result in terms of g. We propose an alternative scheme for characterizing probabilistically the quality of an approximation result that takes into account only $\hat{g}$, the approximation function itself, and the example data points it approximates, without actually having to know g. Here, our objective notion is similar in spirit to

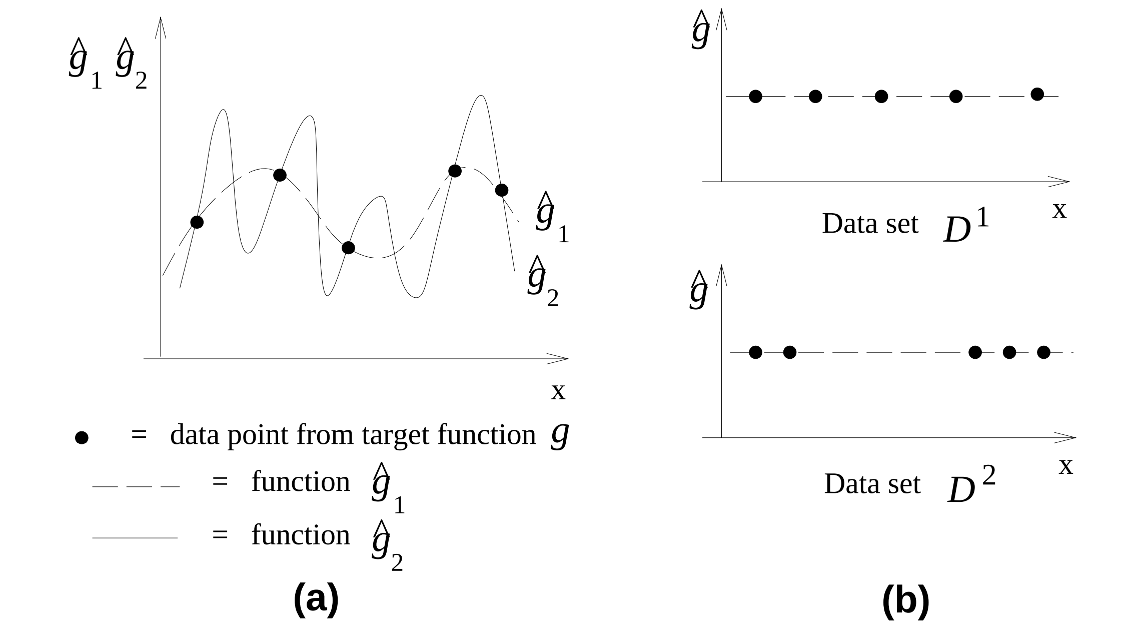


Figure 4-1: **(a):** Two approximation functions, $\hat{g_1}$ and $\hat{g_2}$, for the same set of data points sampled from an unknown target function. $\hat{g_1}$ oscillates less between the data points, so one would normally guess that it is a more probable hypothesis for the unknown target function. **(b):** An approximation function, $\hat{g}$, for two sets of data points sampled from two (possibly different) unknown target functions. Although $\hat{g}$ fits both sets of data points exactly, one might still expect it to more closely resemble the unknown target function of the top system than of the bottom system. This is due to the uneven example distribution in the bottom system.

the *integrated squared difference* "misfit" criterion described above. We elaborate further on what we mean below:

Figure 4-1(a) shows two approximation functions, $\hat{g_1}$ and $\hat{g_2}$, for a set of data points, $\mathcal{D}$, from an *unknown* target function g. Without further knowledge of the target function, g, one would normally guess that $\hat{g_1}$ is a more probable (and hence better) hypothesis for g, because it oscillates less between the data points. This aspect of an approximation function's "goodness" has been fully captured by *regularization*, which assigns $\mathcal{P}(\hat{g_1}|\mathcal{D})$ a higher likelihood value than $\mathcal{P}(\hat{g_2}|\mathcal{D})$.

Figure 4-1(b) shows a function, $\hat{g}$, that approximates two unknown target functions g_1 and g_2, sampled at $\mathcal{D}^1$ and $\mathcal{D}^2$ respectively. Notice that in this example, the approximator $\hat{g}$ fits both data sets exactly, so we have $\hat{g} = \arg\max_{f\in\mathcal{F}} \mathcal{P}(f|\mathcal{D}^1)$ and $\hat{g} = \arg\max_{f\in\mathcal{F}} \mathcal{P}(f|\mathcal{D}^2)$. Intuitively however, one might still expect the actual *misfit* between g_1 and $\hat{g}$ to be smaller than the actual *misfit* between g_2 and $\hat{g}$. This is because $\mathcal{D}^1$ is a more representative data sample for g_1 than $\mathcal{D}^2$ is for g_2, and in both systems, $\hat{g}$ is directly derived from $\mathcal{D}^1$ and $\mathcal{D}^2$ respectively. One can view this *expected misfit* notion between an *unknown* target g and its approximation function $\hat{g}$, as a sense of "uncertainty" that one has in the current solution. The notion is not captured by the *regularization* framework, and as we shall see, depends instead on the distribution of training examples over the input space.

Since our active learning task is to determine the best input space location for sampling next, a reasonable learning goal would be to sample at locations that minimize the *expected misfit* notion between the unknown target g and the resulting approximation $\hat{g}$.

4.2.2 Evaluating a Solution to an Unknown Target — The Expected Integrated Squared Difference

We now formalize the above *expected misfit* notion as a mathematical functional to be minimized. The general idea is as follows: Let $\mathcal{F}$ be the approximation function class in our learning task. Suppose we treat the unknown target function g as a random variable in $\mathcal{F}$, then one way of determining the *expected misfit* between the *regularized* solution, $\hat{g}$, and the unknown target function, g, would be to compute an expected version of some difference measure between them, such as their *integrated squared difference*, $\delta(\hat{g}, g)$ (see Equation 4.4). Taking into account $\mathcal{D}_n$, the n data points seen so far, and $\mathcal{P}_{\mathcal{F}}(g)$, the prior probability of g in $\mathcal{F}$, we have the following a-posteriori likelihood for g: $\mathcal{P}(g|\mathcal{D}_n) \propto \mathcal{P}_{\mathcal{F}}(g)\mathcal{P}(\mathcal{D}_n|g)$. The *expected* integrated squared difference (EISD) between an unknown target, g, and its estimate, $\hat{g}$, given $\mathcal{D}_n$, is thus:

$$E_{\mathcal{F}}[\delta(\hat{g}, g)|\mathcal{D}_n] = \int_{g\in\mathcal{F}} \mathcal{P}(g|\mathcal{D}_n)\delta(\hat{g}, g)dg = \int_{g\in\mathcal{F}} \mathcal{P}_{\mathcal{F}}(g)\mathcal{P}(\mathcal{D}_n|g)\delta(\hat{g}, g)dg. \qquad (4.5)$$

The EISD is intuitively pleasing as an "uncertainty" measure for evaluating a solution to an *unknown* target, because its value decreases with better distributed data samples. The following example illustrates how the measure agrees well with "human intuition". We return to the two function approximation problems described in Figure 4-1(b). In the first system, one intuitively expects a smaller discrepancy between the unknown target and its approximation function than in the second system, even though the same regularized estimate $\hat{g}$ fits both data sets equally well. This is because the data samples $\mathcal{D}^1$ in the first system are more evenly (and hence better) distributed than the samples $\mathcal{D}^2$ in the second system. We now argue that the EISD measure in the first system should indeed be smaller than the EISD measure in the second system.

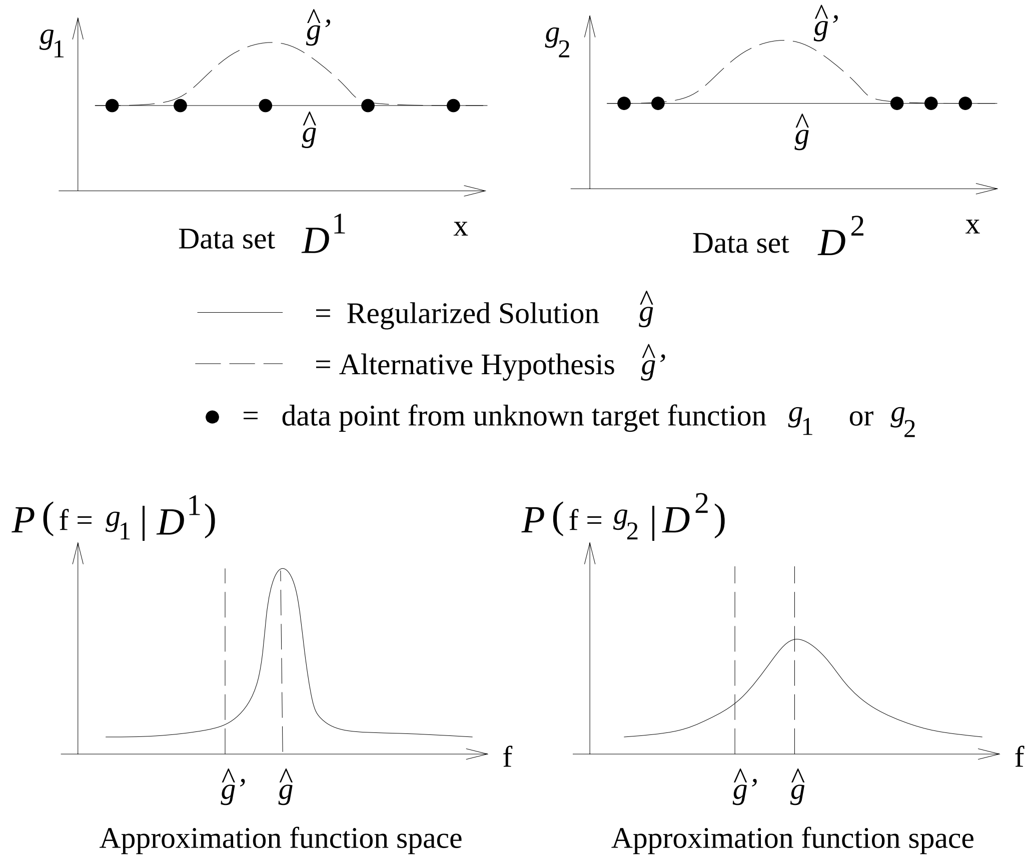

g_1
$\hat{g}'$
$\hat{g}$
Data set D^1
x
g_2
$\hat{g}'$
$\hat{g}$
Data set D^2
x
= Regularized Solution $\hat{g}$
= Alternative Hypothesis $\hat{g}'$
● = data point from unknown target function g_1 or g_2
$P(f = g_1 \mid D^1)$
$P(f = g_2 \mid D^2)$
f
$\hat{g}'$ $\hat{g}$
Approximation function space
f
$\hat{g}'$ $\hat{g}$
Approximation function space

Figure 4-2: **Top Row:** A regularized solution, $\hat{g}$, for two unknown target functions, g_1 (left graph) and g_2 (right graph). The curve $\hat{g}'$ is an alternative hypothesis for the two unknown target functions. There is more evidence against $\hat{g}'$ being a true hypothesis for g_1 than for g_2 because the first system has a data point near the center of the input space where $\hat{g}'$ differs considerably from $\hat{g}$ and the data point. **Bottom Row:** Graphs depicting the a-posteriori probability distribution of the *unknown* target in approximation function space for the two systems. Because there is more evidence against alternative hypotheses like $\hat{g}'$ in the first system than in the second system, we get a sharper peak for the a-posteriori distribution at $\hat{g}$ in the first system than in the second system.

Consider the same two systems in the top row of Figure 4-2, where $\hat{g}$ is the regularized solution for the two unknown target functions g_1 and g_2. The two unknown targets are sampled at $\mathcal{D}^1$ and $\mathcal{D}^2$ respectively, and both data sets contain the same number of data points. Consider next an alternate hypothesis $\hat{g'}$ for g_1 and g_2, that differs slightly from the regularized solution $\hat{g}$ over some region of the input space. Because the first system has better distributed data points than the second system, there is more evidence *against* most alternative hypotheses like $\hat{g'}$ being a viable solution for g_1 than for g_2. Mathematically, this means that for most alternative hypotheses like $\hat{g'}$, the ratio $\mathcal{P}(g_1 = \hat{g'}|\mathcal{D}^1)/\mathcal{P}(g_1 = \hat{g}|\mathcal{D}^1)$ is smaller than the ratio $\mathcal{P}(g_2 = \hat{g'}|\mathcal{D}^2)/\mathcal{P}(g_2 = \hat{g}|\mathcal{D}^2)$. One can therefore expect $E_{\mathcal{F}}[\delta(\hat{g}, g_1)|\mathcal{D}^1] < E_{\mathcal{F}}[\delta(\hat{g}, g_2)|\mathcal{D}^2]$, which agrees well with "human intuition". The bottom row of Figure 4-2 depicts the difference between the two systems graphically. Because most alternative hypotheses are poor solutions for the first data set $\mathcal{D}^1$, the first unknown target g_1 has an a-posteriori probability distribution that is heavily weighted around $\hat{g}$ in approximation function space. The same is less true about the a-posteriori probability distribution for g_2 in the second system. Thus, $\hat{g}$ is a more "stable", and hence a more "certain" solution for g_1 than for g_2.

4.2.3 Selecting the Next Sample Location

Let g be the unknown target function that we want to learn, $\mathcal{D}_n = \{(\vec{x_i}, y_i) \in \Re^d \times \Re | i = 1, \ldots, n\}$ be the set of n examples seen so far, and $\hat{g_n}$ be the current regularized approximation for g. We now formalize the task of determining the best input space location to sample next. Since our learning goal is to minimize the *expected misfit* between g and its regularized solution, a reasonable sampling strategy would be to choose the next example from the input location $\vec{x_{n+1}} \in \Re^d$ that minimizes the EISD between g and its new estimate $\hat{g_{n+1}}$.

How does one predict the new EISD that results from sampling the next data point at location $\vec{x_{n+1}}$? Suppose we also know the target output value (possibly noisy), y_{n+1}, at $\vec{x_{n+1}}$. The EISD between g and its new estimate $\hat{g}_{n+1}$ would then be $E_{\mathcal{F}}[\delta(\hat{g_{n+1}}, g)|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})]$, where $\hat{g_{n+1}}$ can be recovered from $\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})$ via regularization. In reality, we do not know y_{n+1}, but we can derive its conditional probability distribution from $\mathcal{D}_n$, the data samples seen so far. Once again, let $\mathcal{F}$ be the approximation function class for our learning task and $\mathcal{P}_{\mathcal{F}}(f)$ be the prior probability of f in $\mathcal{F}$, then:

$$\mathcal{P}(y_{n+1}|\vec{x_{n+1}}, \mathcal{D}_n) \propto \int_{f \in \mathcal{F}} \mathcal{P}(\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})|f)\mathcal{P}_{\mathcal{F}}(f)df. \qquad (4.6)$$

Because y_{n+1} is a random variable and not a fixed value as we had assumed earlier, this leads to the following *expected* value for the new EISD, if we sample our next data point at $\vec{x_{n+1}}$:

$$\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}}) = \int_{-\infty}^{\infty} \mathcal{P}(y_{n+1}|\vec{x_{n+1}}, \mathcal{D}_n)E_{\mathcal{F}}[\delta(\hat{g_{n+1}}, g)|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})]dy_{n+1}. \qquad (4.7)$$

Notice from Equation 4.5 that $E_{\mathcal{F}}[\delta(\hat{g_{n+1}}, g)|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})]$ in the above expression is actually independent of the unknown target function g, and so $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$ (henceforth referred to as the *total output uncertainty*) is fully computable from available information in the learning model. Clearly, the optimal input location to sample next is the location that minimizes $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$, i.e.:

$$\hat{\vec{x_{n+1}}} = \arg\min_{\vec{x_{n+1}}} \mathcal{U}(g_{n+1}|\mathcal{D}_n, \vec{x_{n+1}}). \qquad (4.8)$$

4.2.4 Summary of the Active Learning Procedure

We summarize the key steps involved in our active learning strategy for finding the optimal next sample location:

1. Compute $\mathcal{P}(g|\mathcal{D}_n)$. This is the a-posteriori likelihood of the different functions g given $\mathcal{D}_n$, the n data points seen so far.

2. Assume a new point $\vec{x_{n+1}}$ to sample.

3. Assume a value y_{n+1} for this $\vec{x_{n+1}}$. One can compute $\mathcal{P}(g|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1}))$ and hence the *expected* integrated squared difference (EISD) between the target and its new estimate $\hat{g_{n+1}}$. This is given by $E_{\mathcal{F}}[\delta(\hat{g_{n+1}}, g)|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})]$ (see Equation 4.5).

4. At the assumed $\vec{x_{n+1}}$, y_{n+1} has a probability distribution given by Equation 4.6. Averaging the resulting EISD over all y_{n+1}'s, we obtain the *total output uncertainty* for $\vec{x_{n+1}}$, given by $\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, \vec{x_{n+1}})$ in Equation 4.7.

5. Sample at the input location $\hat{\vec{x}}_{n+1}$ that minimizes the *total output uncertainty* cost function $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x}_{n+1})$.

Some final remarks about our example selection strategy: Intuitively, a reasonable selection criterion should choose new examples that provide dense information about the target function g. Furthermore, the choice should also take into account the learner's current state, namely $\mathcal{D}_n$ and $\hat{g}_n$, so as to maximize the *net* amount of information gained. Our scheme treats an approximation function's *expected misfit* with respect to the unknown target g (i.e. their EISD) as a measure of *uncertainty* in the current solution. It selects new examples, based on the data that it has already seen, to minimize the *expected* value of the resulting EISD measure. In doing so, it essentially maximizes the net amount of information gained with each new example.

Our main results in this active learning formulation are:

1. a cost function that captures the *expected misfit* optimality criterion (Equation 4.5) for evaluating the solution to an *unknown* target function, and

2. a formal specification for the task of selecting new training examples with maximum potential utility (Equation 4.8).

These results may, in themselves, be interesting from a theoretical standpoint, but in practice, another fundamental concern must also be addressed — the computational complexity issue. Both Equations 4.5 and 4.8, though theoretically computable from available information in the learning model, are clearly intractable in their current form. Nevertheless, we maintain the formulation still serves as a possible "optimal" benchmark for evaluating other active example selection schemes. Later in this chapter, we shall consider a reduced version of the original function approximation based active learning formulation that essentially hunts for new data where approximation "error bars" are high. We also show how such a scheme, with minor modifications, leads to the "boot-strap" example selection strategy we have adopted in our object and pattern class detection approach.

4.3 Comparing Sample Complexity

To demonstrate the usefulness of the above active learning procedure, we show analytically and empirically that the active strategy learns target functions with fewer data examples

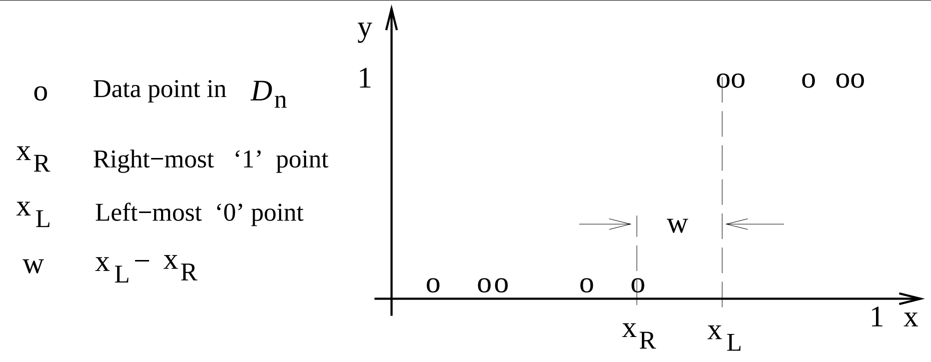


Figure 4-3: Diagram showing the notation used for our unit-step example.

than random sampling for three specific approximation function classes: (1) unit step functions, (2) polynomial approximators and (3) Gaussian radial basis function networks. For all three function classes, one can derive exact analytic data selection algorithms following the key steps outlined in Section 4.2.4.

4.3.1 Unit Step Functions

We first consider the following simple class of one-dimensional unit-step functions described by a single parameter a which takes values in $[0, 1]$. Let us denote the unit-step function by:

$$u(x-a) = \begin{cases} 1 & \text{if } x \geq a \\ 0 & \text{otherwise} \end{cases}$$

The target and approximation function class for this problem is given by:

$$\mathcal{F} = \{u(x-a) | 0 \leq a \leq 1\}$$

Assuming a has an a-priori uniform distribution on $[0, 1]$, we obtain the following prior distribution on the approximation function class:

$$\mathcal{P}_{\mathcal{F}}(g = u(x-a)) = \begin{cases} 1 & \text{if } 0 \leq a \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Suppose we have a *noiseless* data set, $\mathcal{D}_n = \{(x_i, y_i); i = 1, ..n\}$, consistent with some unknown target function $g = u(x-a)$ that the learner has to approximate. We want to find the best input location to sample next, $x \in [0, 1]$, that would provide us with maximal information. Let x_R be the right most point in $\mathcal{D}_n$ whose y value is 0, i.e.,

$x_R = \max_{i=1,..n}\{x_i | y_i = 0\}$ (see Figure 4-3). Similarly, let $x_L = \min_{i=1,..n}\{x_i | y_i = 1\}$ and $w = x_L - x_R$. Following the general procedure outlined in Section 4.2.4, we go through the following steps:

1. Derive $\mathcal{P}(g|\mathcal{D}_n)$. One can show that:
$$\mathcal{P}(g = u(x-a)|\mathcal{D}_n) = \begin{cases} \frac{1}{w} & \text{if } a \in [x_R, x_L] \\ 0 & \text{otherwise} \end{cases}$$

2. Suppose we sample next at a particular $x \in [0,1]$, we would obtain y with the distribution:
$$P(y = 0|\mathcal{D}_n, x) = \begin{cases} \frac{(x_L - x)}{x_L - x_R} = \frac{(x_L - x)}{w} & \text{if } x \in [x_R, x_L] \\ 1 & \text{if } x \leq x_R \\ 0 & \text{otherwise} \end{cases}$$
$$P(y = 1|\mathcal{D}_n, x) = \begin{cases} \frac{(x - x_R)}{x_L - x_R} = \frac{(x - x_R)}{w} & \text{if } x \in [x_R, x_L] \\ 1 & \text{if } x \geq x_L \\ 0 & \text{otherwise} \end{cases}$$

3. For a particular y, the new data set would be $\mathcal{D}_{n+1} = \mathcal{D}_n \cup (x, y)$ and the corresponding EISD can be easily obtained using the distribution $\mathcal{P}(g|\mathcal{D}_{n+1})$. Averaging this over $\mathcal{P}(y|\mathcal{D}_n, x)$ as in **step 4** of the general procedure, we obtain:
$$\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, x) = \begin{cases} \frac{w^2}{12} & \text{if } x \leq x_R \text{ or } x \geq x_L \\ \frac{1}{12w}((x_L - x)^3 + (x - x_R)^3) & \text{otherwise} \end{cases}$$

4. Clearly the new input location that minimizes the *total output uncertainty*, $\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, x)$, measure is the midpoint between x_L and x_R:
$$\hat{x_{n+1}} = \arg\min_{x \in [0,1]} \mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, x) = \frac{x_L + x_R}{2}.$$

Thus, by applying the general procedure to this trivial case of one-dimensional unit-step functions, we get a binary search learning algorithm that queries the midpoint of x_R and x_L. For this function class, one can show analytically in PAC-style [98] that our active data sampling strategy takes fewer examples to learn an unknown target function to a given level of *total output uncertainty* than randomly drawing examples according to a uniform distribution in x.

Theorem 1 *Suppose we want to collect examples so that we are guaranteed with high probability (i.e. probability $> 1 - \delta$) that the* total output uncertainty *is less than ϵ. Then a passive learner would require at least $\frac{1}{\sqrt{48\epsilon}} \ln(1/\delta)$ examples while the active strategy described earlier would require at most $(1/2)\ln(1/12\epsilon)$ examples.*

4.3.2 Polynomial Approximators

We consider next a univariate polynomial target and approximation function class with maximum degree K, i.e.:

$$\mathcal{F} = \{g(x, \vec{a}) = g(x, a_0, \ldots, a_K) = \sum_{i=0}^{K} a_i x^i\}.$$

The model parameters to be learnt are $\vec{a} = [a_0\, a_1\, \ldots\, a_K]^{\mathrm{T}}$ and x is the input variable. We obtain a prior distribution for $\mathcal{F}$ by assuming a zero-mean Gaussian distribution with covariance $\Sigma_{\mathcal{F}}$ on the model parameters $\vec{a}$:

$$\mathcal{P}_{\mathcal{F}}(g(\cdot, \vec{a})) = \mathcal{P}_{\mathcal{F}}(\vec{a}) = \frac{1}{(2\pi)^{(K+1)/2} |\Sigma_{\mathcal{F}}|^{1/2}} \exp(-\frac{1}{2} \vec{a}^{\mathrm{T}} \Sigma_{\mathcal{F}}^{-1} \vec{a}). \tag{4.9}$$

Our task is to approximate an *unknown* target function $g \in \mathcal{F}$ within the input range $[x_{\mathtt{LO}}, x_{\mathtt{HI}}]$ on the basis of sampled data. Let $\mathcal{D}_n = \{(x_i, y_i = g(x_i) + \eta) | i = 1, \ldots, n\}$ be a *noisy* data sample from the unknown target in the input range $[x_{\mathtt{LO}}, x_{\mathtt{HI}}]$, where η is an additive zero-mean Gaussian noise term with variance σ_s^2. We compare two different ways of selecting the next data point: (1) sampling the function at a random point x according to a uniform distribution in $[x_{\mathtt{LO}}, x_{\mathtt{HI}}]$ (i.e. passive learning), and (2) using our active learning framework to derive an exact algorithm for determining the next sampled point.

The Active Strategy

Here, we go through the general active learning procedure outlined in Section 4.2.4 to derive an exact expression for $\hat{x_{n+1}}$, the next query point. We summarize the key derivation steps below:

1. Let $\bar{\mathbf{x}}_{\mathbf{i}} = [1\, x_i\, x_i^2\, \ldots\, x_i^K]^{\mathrm{T}}$ be a power vector of the i^{th} data sample's input value. One can show (see Appendix A.1.1) that the a-posteriori approximation function class distribution, $\mathcal{P}(\vec{a}|\mathcal{D}_n)$, is a multivariate Gaussian centered at $\hat{\vec{a}}$ with covariance Σ_n, where:

$$\hat{\vec{a}} = \Sigma_n(\frac{1}{\sigma_s^2}\sum_{i=1}^{n} \bar{\mathbf{x}}_{\mathbf{i}} y_i)$$

   and:

$$\Sigma_n^{-1} = \Sigma_{\mathcal{F}}^{-1} + \frac{1}{\sigma_s^2}\sum_{i=1}^{n}\left(\bar{\mathbf{x}}_{\mathbf{i}}\bar{\mathbf{x}}_{\mathbf{i}}^T\right). \tag{4.10}$$

2. Deriving the *total output uncertainty* expression $\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, x_{n+1})$ requires several steps (see Appendix A.1.2 and A.1.3). Taking advantage of the Gaussian distribution on both the parameters $\vec{a}$ and the noise term, we eventually get:

$$\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, x_{n+1}) = |\Sigma_{n+1}\mathbf{A}| \propto |\Sigma_{n+1}|, \tag{4.11}$$

   where $\mathbf{A}$ is a constant $(K+1)\times(K+1)$ matrix of numbers whose $(i,j)^{\mathrm{th}}$ element is:

$$\mathbf{A}_{i,j} = \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}} t^{(i+j-2)} dt$$

   Σ_{n+1} has the same form as Σ_n and depends on the previous data, the priors, noise and the next sample location x_{n+1}. When minimized over x_{n+1}, we get $\hat{x_{n+1}}$ as the maximum utility location where the active learner should next sample the unknown target function.

Simulations — Error Rate versus Number of Examples

We perform some simulations to compare the active strategy's sample complexity with that of a passive learner which receives uniformly distributed random training examples on the input domain $[x_{\mathtt{LO}}, x_{\mathtt{HI}}]$. In this experiment, we investigate whether our active example selection strategy learns an unknown target to a smaller average error rate than the passive strategy for the same number of data samples. The experiment proceeds as follows:

We randomly generate 1000 target polynomial functions using a fixed Gaussian prior on the model parameters $\vec{a} = [a_0\, a_1\, \ldots\, a_K]^{\mathrm{T}}$. For each target polynomial, we collect data sets with noisy y values ranging from 3 to 50 samples in size, using both the active and passive sampling strategies. We then assume the same Gaussian priors on the approximation function class to obtain a regularized estimate of the target polynomial for each data set. Because we know the actual target polynomial for each data set, one can compute the actual integrated squared difference between the target and its estimate as an approximation error measure. We compare the two sampling strategies by separately averaging their approximation error rates for each data sample size over the 1000 different target polynomials.

In our simulations, we use polynomials of maximum degree $K = 9$, distributed according to the following independent Gaussian priors on model parameters: for each a_j in $\vec{a} = [a_0\, a_1\, \ldots\, a_9]^{\mathrm{T}}$, we have:

$$\mathcal{P}(a_j) = \frac{1}{\sigma_j\sqrt{2\pi}} \exp\left(-\frac{a_j^2}{2\sigma_j^2}\right),$$

where $\sigma_j = 0.9^{j+1}$. In other words, $\Sigma_{\mathcal{F}}$ of Equation 4.9 is a 10×10 diagonal covariance matrix such that:

$$\Sigma_{\mathcal{F}}(i,j) = \begin{cases} \sigma_{i-1}^2 = 0.9^{2i} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

Qualitatively, our priors favor smooth functions by assigning higher probabilities to polynomials with smaller coefficients, especially for higher powers of x. We also fix the input domain to be $[x_{\mathtt{LO}}, x_{HI}] = [-5, 5]$.

Figure 4-4 shows the average *integrated squared difference* between the 1000 randomly generated target polynomials and their regularized estimates for different data sample sizes. We repeated the same simulations three times, each with a different output noise variance in the data samples: $\sigma_s = 0.1$, 1.0 and 5.0. Notice that the active strategy has a lower average error rate than the passive strategy particularly for smaller data samples. From this experiment, one can conclude empirically that our active sampling strategy learns with fewer data samples than random sampling even when dealing with noisy data.

Simulations — Incorrect Priors

We next investigate how the active learning strategy behaves if the approximation function class $\mathcal{F}$ differs slightly from the actual target class. Specifically, we consider the fol-

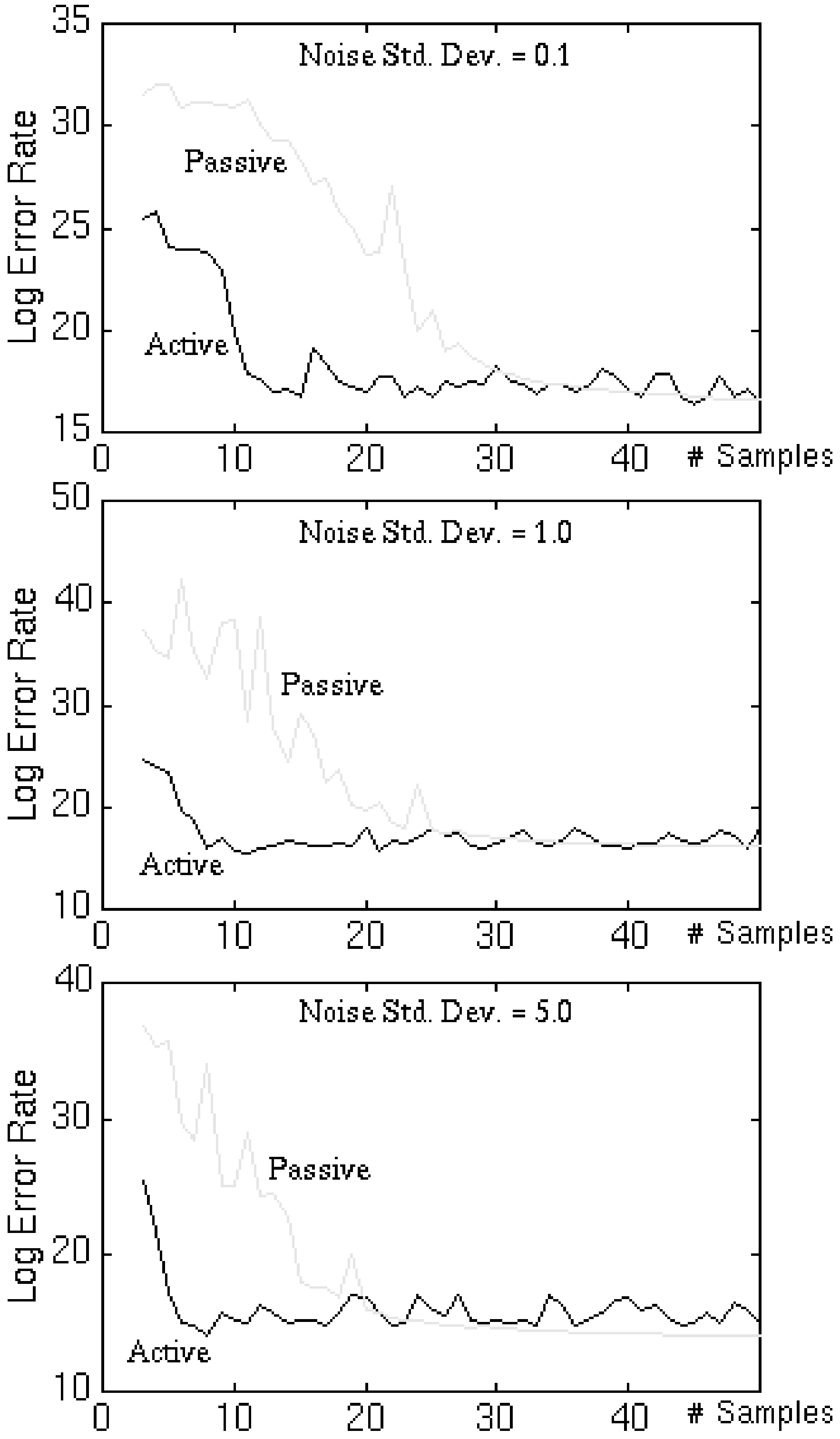


Figure 4-4: Comparing active and passive learning average error rates at different output noise levels for polynomials of maximum degree $K = 9$. We use the same priors on the target and approximation function classes. The three graphs above plot log error rates against number of samples. See text for detailed explanation. The dark and light curves are the active and passive learning error rates respectively.

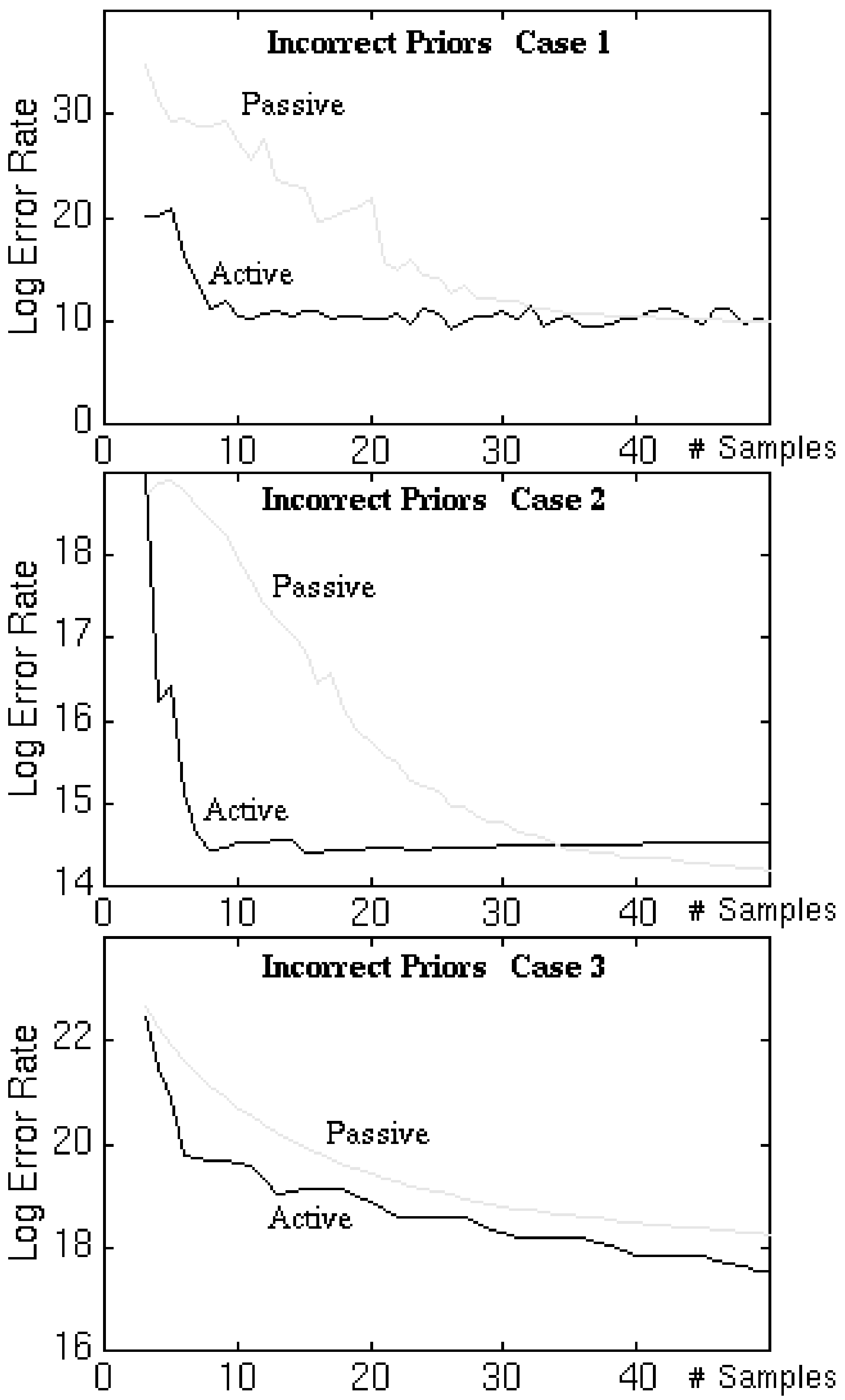


Figure 4-5: Comparing active and passive learning average error rates for slightly different priors between the target and approximation function classes. **Top:** Results for the first case. The approximation function class uses a *higher* degree polynomial with *larger* Gaussian variances on its coefficients ($K = 9$ and $\sigma_j = 0.9^{j+1}$) versus ($K = 8$ and $\sigma_j = 0.8^{j+1}$). **Middle:** The approximation function class uses a *lower* degree polynomial with smaller Gaussian variances on its coefficients ($K = 7$ and $\sigma_j = 0.7^{j+1}$) versus ($K = 8$ and $\sigma_j = 0.8^{j+1}$). **Bottom:** The approximation and target polynomial function classes have smoothness priors that differ in form. In all three cases, the active learning strategy still results in lower approximation error rates than random sampling for the same number of data points.

lowing three cases:

In the first case, the active learner assumes a higher polynomial degree with similar but slightly larger Gaussian variances than the true target priors. We use a 9th degree (i.e. $K = 9$) polynomial function class with Gaussian priors $\sigma_j = 0.9^{j+1}$ to approximate an unknown 8th degree (i.e. $K = 8$) target polynomial with Gaussian priors $\sigma_j = 0.8^{j+1}$. Qualitatively, the approximation function class is *more complex* and favors smooth estimates *less strongly* than the target class.

The second case deals with the exact opposite scenario. The active learner uses a *lower* degree polynomial with similar but slightly smaller Gaussian variances ($K = 7$ and $\sigma_j = 0.7^{j+1}$) to approximate an unknown 8th degree (i.e. $K = 8$) target with Gaussian priors $\sigma_j = 0.8^{j+1}$. Here, the approximation function class is *less complex* and favors smooth estimates *more strongly* than the target class.

In the third case, we consider a polynomial approximation function class $\mathcal{F}$ whose prior distribution has a different form from that of the target class. Let $p \in \mathcal{F}$ be a polynomial in the approximation function class. One can quantify the overall "smoothness" of p by integrating its squared first derivative over the input domain $[x_{\mathtt{LO}}, x_{\mathtt{HI}}]$:

$$\mathcal{Q}(p(\cdot, \vec{a})) = \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}} \left[\frac{dp(x, \vec{a})}{dx}\right]^2 dx. \tag{4.13}$$

The "smoothness" measure above leads to a convenient prior distribution on $\mathcal{F}$ that favors smoothly varying functions:

$$\mathcal{P}_{\mathcal{F}}(p) \propto \exp(-\mathcal{Q}(p(\cdot, \vec{a}))) \exp(-\frac{a_0^2}{2\sigma_0^2}).$$

Here, a_0 is the constant term in the polynomial p, whose coefficients are $\vec{a} = [a_0\ a_1\ \ldots\ a_K]^{\mathrm{T}}$. Although a_0 does not affect the "smoothness" measure in Equation 4.13, we impose on it a Gaussian distribution with variance σ_0^2 so $\mathcal{P}_{\mathcal{F}}(p)$ integrates to 1 over all polynomials in $\mathcal{F}$ like a true probability density function. One can show (see Appendix A.1.4 for detailed derivation) that $\mathcal{P}_{\mathcal{F}}(p)$ has the following general form similar to Equation 4.9, the priors on polynomials with independent Gaussian distributed coefficients:

$$\mathcal{P}_{\mathcal{F}}(p(\cdot, \vec{a})) = \mathcal{P}_{\mathcal{F}}(\vec{a}) = \frac{1}{(2\pi)^{(K+1)/2}|\Sigma_{\mathcal{F}}|^{1/2}} \exp(-\frac{1}{2}\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}).$$

The new covariance term $\Sigma_{\mathcal{F}}$ is as given below:

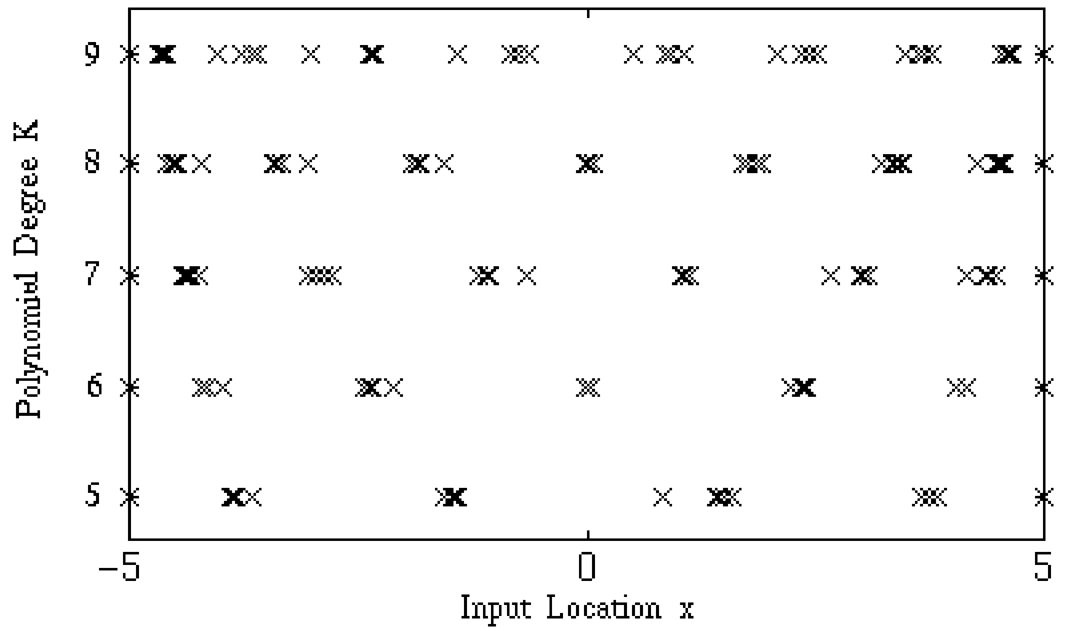


Figure 4-6: Distribution of the first 50 data points the active learner selects as the polynomial degree K of the approximation function class varies from 5 to 9. In all 5 cases, we assume a fixed data output noise level of $\sigma_s = 0.5$. Notice that for polynomials of degree K, the active learner clusters its data samples typically around $K+1$ locations.

$$\Sigma_{\mathcal{F}}^{-1}(i,j) = \begin{cases} 1/\sigma_0^2 & \text{if } i = j = 1 \\ 2\frac{(i-1)(j-1)}{i+j-3}(x_{\mathtt{HI}}^{i+j-3} - x_{\mathtt{LO}}^{i+j-3}) & \text{if } 2 \leq i \leq K+1 \text{ and } 2 \leq j \leq K+1 \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

For this third case, we use an 8^{th} degree (i.e. $K = 8$) polynomial function class with $\sigma_0 = 0.8$ in its "smoothness" prior to approximate a target polynomial class of similar degree with Gaussian priors $\sigma_j = 0.8^{j+1}$. Although the approximation and target function classes have prior distributions that differ somewhat in form, both priors are qualitatively similar in that they favor smoothly varying polynomials.

For all three cases of slightly incorrect priors described above, we compare our active learner's sample complexity with that of a passive learner which receives random samples according to a uniform distribution on $[x_{\mathtt{LO}}, x_{\mathtt{HI}}]$. We repeat the active versus passive function learning simulations performed earlier by generating 1000 target polynomials, collecting noisy data samples ($\sigma_s = 0.5$), computing regularized estimates, averaging and comparing their approximation errors in the same way as before. Figure 4-5 plots the resulting average

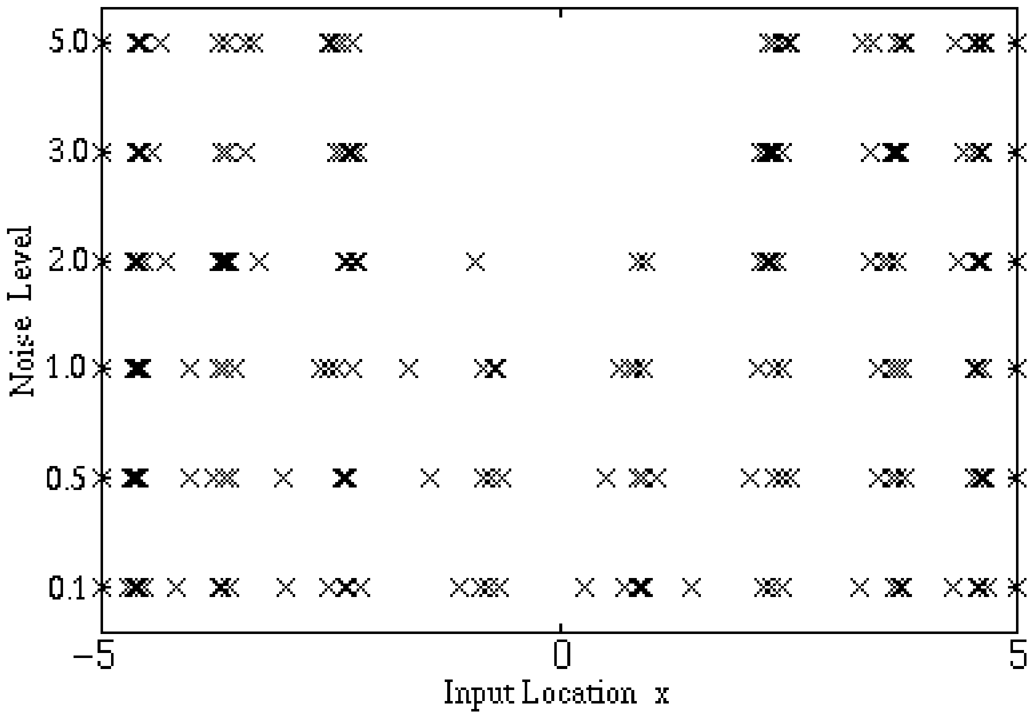


Figure 4-7: Distribution of the first 50 data points the active learner selects for a 9th degree polynomial approximation function class (i.e. $K = 9$), as the assumed data output noise level σ_s varies from 0.1 to 5.0. At higher noise levels, there is less pressure for the active learner to fit the data closely, and so it favors polynomials with small coefficients. For such "lower order" polynomials, one gets better "leverage" from data by sampling away from the origin.

integrated squared difference error rates over a range of data sample sizes for all three cases. Despite the incorrect priors, we see that the active learner still outperforms the passive strategy.

Distribution of Data Points

Notice from Equations 4.10, 4.11, 4.12 and 4.14 that the *total output uncertainty* measure $\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, \vec{x_{n+1}})$ for polynomial approximators (i.e., Equation 4.11) does not depend on the previous y data values actually observed, but only on the previous input locations sampled. In other words, the previously observed y data values do not affect $\hat{x_{n+1}}$, the optimal location to sample next. One can show that this behavior is common to all approximation function classes that are linear in their model parameters [58] [86].

Given a polynomial approximation function class of maximum degree K, one can thus pre-compute the sequence of input locations that our active learner will sample to gain maximum information about the unknown target. There are two sampling trends that are noteworthy here. First, the active strategy does not simply sample the input domain on a uniform grid. Instead, it chooses to cluster its data samples typically around $K+1$ locations. Figure 4-6 shows the first 50 input locations the active learner selects as K varies from 5 to 9, for a fixed data noise level of $\sigma_s = 0.1$. One possible explanation for this clustering behavior is that it takes only $K+1$ data points to recover a K^{th} degree target polynomial in the absence of noise.

Second, as the data noise level σ_s increases, although the number of data clusters remains fixed, the clusters tend to be distributed away from the input origin. Figure 4-7 displays the first 50 input locations the active strategy selects for a 9^{th} degree polynomial approximation function class, as σ_s increases from 0.1 to 5.0. One can explain the observed behavior as follows: For higher noise levels, there is less pressure on the active learner to fit the data closely. Consequently, the prior assumption favoring polynomials with small coefficients dominates. For such "lower order" polynomials, one gets better "leverage" from data by sampling away from the origin. In the extreme case of linear regression, one gets best "leverage" by sampling data at the extreme ends of the input space.

4.3.3 Gaussian Radial Basis Functions

Our final example looks at an approximation function class $\mathcal{F}$ of d-dimensional Gaussian radial basis functions with K fixed centers. Let $\mathcal{G}_i$ be the i^{th} basis function with a fixed center $\vec{c_i}$ and a fixed covariance $\mathcal{S}_i$. The model parameters to be learnt are the weight coefficients denoted by $\vec{a} = [a_1\ a_2\ \cdots\ a_K]^{\mathrm{T}}$. An arbitrary function $r \in \mathcal{F}$ in this class can thus be represented as:

$$\begin{aligned} r(\vec{x}, \vec{a}) &= \sum_{i=1}^{K} a_i \mathcal{G}_i(\vec{x}) \\ &= \sum_{i=1}^{K} a_i \frac{1}{(2\pi)^{d/2}|\mathcal{S}_i|^{1/2}} \exp(-\frac{1}{2}(\vec{x}-\vec{c_i})^{\mathrm{T}} \mathcal{S}_i^{-1} (\vec{x}-\vec{c_i})) \end{aligned}$$

We impose a prior $\mathcal{P}_{\mathcal{F}}()$ on the approximation function class $\mathcal{F}$ by putting a zero-centered Gaussian distribution with covariance $\Sigma_{\mathcal{F}}$ on the model parameters $\vec{a}$. Thus, for an arbitrary function $r(\cdot, \vec{a})$:

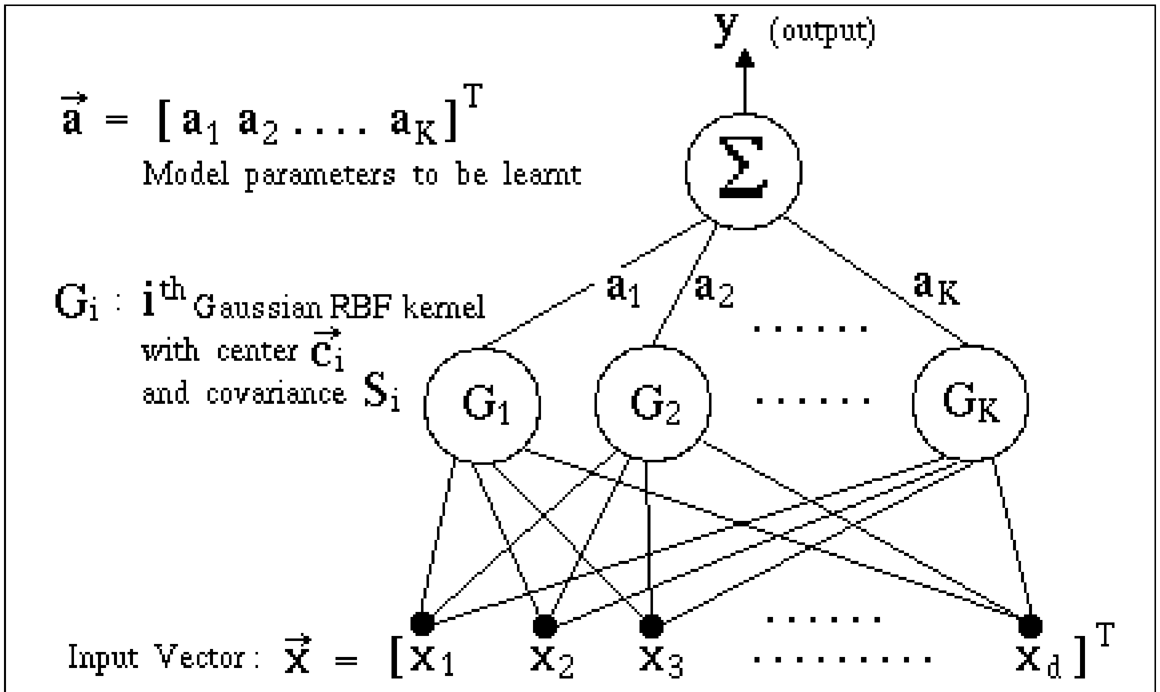

y (output)
$\vec{a} = [a_1\ a_2 \ldots a_K]^T$
Model parameters to be learnt
Σ
G_i : i^{th} Gaussian RBF kernel with center $\vec{c}_i$ and covariance S_i
a_1
a_2
a_K
G_1
G_2
G_K
Input Vector : $\vec{x} = [x_1\ x_2\ x_3 \ldots x_d]^T$

Figure 4-8: Gaussian radial basis function (RBF) model with K fixed centers. The learner's task is to recover the weight coefficients $\vec{a} = [a_1\ a_2\ \cdots\ a_K]^{\mathrm{T}}$.

$$\mathcal{P}_{\mathcal{F}}(r(\cdot, \vec{a})) = \mathcal{P}_{\mathcal{F}}(\vec{a}) = \frac{1}{(2\pi)^{K/2}|\Sigma_{\mathcal{F}}|^{1/2}} \exp(-\frac{1}{2}\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}).$$

Lastly, the learner has access to noisy data of the form $\mathcal{D}_n = \{(\vec{x_i}, y_i = g(\vec{x_i}) + \eta) : i = 1, \ldots, n\}$, where g is an unknown target function and η is a zero-mean additive Gaussian noise term with variance σ_s^2. Thus for every candidate approximation function $r(\cdot, \vec{a}) \in \mathcal{F}$, $\mathcal{P}(\mathcal{D}_n | r(\cdot, \vec{a}))$ has the form:

$$\begin{aligned}
\mathcal{P}(\mathcal{D}_n | r(\cdot, \vec{a})) &\propto \exp\left(-\frac{1}{2\sigma_s^2}\sum_{i=1}^{n}(y_i - r(\vec{x_i}, \vec{a}))^2\right) \\
&= \exp\left(-\frac{1}{2\sigma_s^2}\sum_{i=1}^{n}(y_i - \sum_{t=1}^{K} a_t \frac{\exp\left(-\frac{1}{2}(\vec{x_i} - \vec{c_t})^{\mathrm{T}}\mathcal{S}_t^{-1}(\vec{x_i} - \vec{c_t})\right)}{(2\pi)^{d/2}|\mathcal{S}_t|^{1/2}})^2\right) \\
&= \exp\left(-\frac{1}{2\sigma_s^2}\sum_{i=1}^{n}(y_i - \sum_{t=1}^{K} a_t \mathcal{G}_t(\vec{x_i}))^2\right)
\end{aligned}$$

Given a set of n data points $\mathcal{D}_n$, one can obtain a maximum a-posteriori (MAP) solution to the learning problem by finding a set of model parameters $\hat{\vec{a}}$ that maximizes $\mathcal{P}(r(\cdot,\vec{a})|\mathcal{D}_n) = \mathcal{P}_{\mathcal{F}}(r(\cdot,\vec{a}))\mathcal{P}(\mathcal{D}_n|r(\cdot,\vec{a}))$. Let:

$$\bar{\mathbf{z}}_{\mathbf{i}} = [\mathcal{G}_1(\vec{x_i})\ \mathcal{G}_2(\vec{x_i})\ \ldots\ \mathcal{G}_K(\vec{x_i})]^{\mathrm{T}}$$

be a vector of RBF kernel output values for the i^{th} input value. One can show (see Appendix A.2.1), as in the polynomial case, that the a-posteriori RBF approximation function class distribution $\mathcal{P}(r(\cdot,\vec{a})|\mathcal{D}_n)$ is a multivariate Gaussian centered at $\hat{\vec{a}}$ with covariance Σ_n, where:

$$\Sigma_n^{-1} = \Sigma_{\mathcal{F}}^{-1} + \frac{1}{\sigma_s^2}\sum_{i=1}^{n}(\bar{\mathbf{z}}_{\mathbf{i}}\bar{\mathbf{z}}_{\mathbf{i}}^{\mathrm{T}}) \tag{4.15}$$

$$\hat{\vec{a}} = \Sigma_n\,(\frac{1}{\sigma_s^2}\sum_{i=1}^{n}\bar{\mathbf{z}}_{\mathbf{i}}y_i) \tag{4.16}$$

Notice that $\hat{\vec{a}}$ of Equation 4.16 is also the MAP solution the learner proposes on the basis of the data set $\mathcal{D}_n$, *regardless* of how the data points are selected. We now describe an active strategy for selecting the data optimally.

The Active Strategy

Recall that our goal is to derive an analytical expression for $\mathcal{U}(\hat{g_{n+1}}|D_n, \vec{x_{n+1}})$ in Equation 4.7, the *total output uncertainty* cost function to minimize that yields the optimal location for sampling next. As before, we go through the general active learning procedure outlined in Section 4.2.4 to derive an exact expression for $\hat{x_{n+1}}$, the optimal next query point.

The first derivation step is to obtain an analytical expression for $\mathcal{P}(\vec{a}|\mathcal{D}_n)$, the a-posteriori RBF approximation function class distribution. This is exactly $\mathcal{P}(r(\cdot,\vec{a})|\mathcal{D}_n)$ which we introduced in the series of equations above leading to Equation 4.16.

Deriving the RBF *total output uncertainty* cost function $\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, \vec{x_{n+1}})$ requires several steps (see Appendix A.2.2 and A.2.3). We eventually get:

$$\mathcal{U}(\hat{g_{n+1}}|\mathcal{D}_n, \vec{x_{n+1}}) \propto |\Sigma_{n+1}|. \tag{4.17}$$

Σ_{n+1} has the same form as Σ_n in Equation 4.16 and depends on the previous data sample locations $\{\vec{x_i} : i = 1, \ldots, n\}$, the model priors $\Sigma_{\mathcal{F}}$, the data noise variance σ_s^2, and the next sample location $\vec{x_{n+1}}$. When minimized over $\vec{x_{n+1}}$, we get $\hat{\vec{x_{n+1}}}$ as the maximum utility location where the active learner should next sample the unknown target function.

Like the polynomial class example, our RBF approximation function class $\mathcal{F}$ is also linear in its model parameters. As such, the optimal new sample location $\hat{\vec{x_{n+1}}}$ does not depend on the y data values in $\mathcal{D}_n$, but only on the previously sampled $\vec{x}$ values.

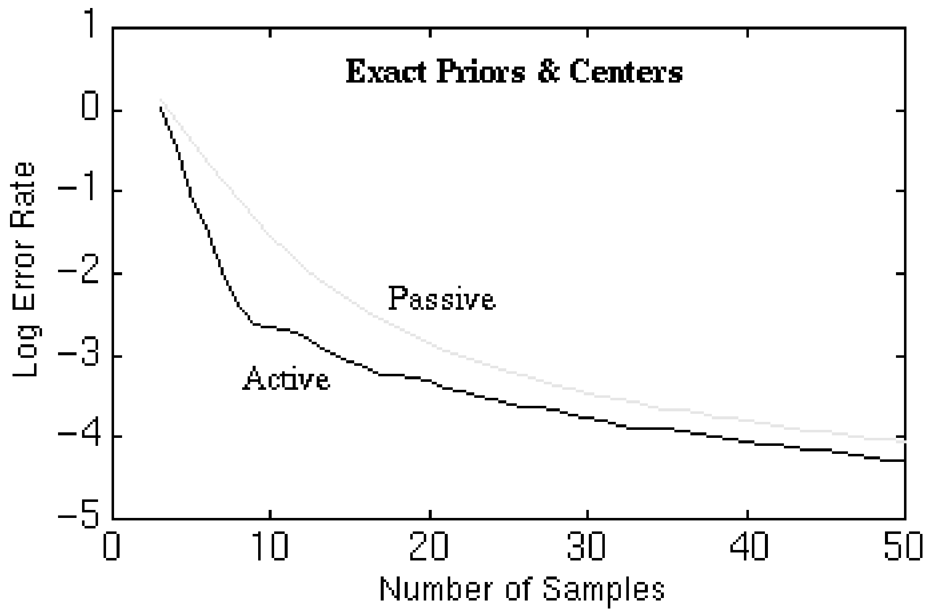
Simulations — Error Rate versus Number of Examples

Does the active strategy for our RBF approximation function class take fewer examples to learn an unknown target than a passive learner that draws random samples according to a uniform distribution on the input domain? We compare sample complexities for the active and passive learners under the following two conditions:

1. **The approximation and target function classes have identical priors.** For simplicity, we perform our simulations in a one-dimensional input domain $[x_{\texttt{LO}}, x_{\texttt{HI}}] = [-5, 5]$. The approximation and target function classes are RBF networks with $K = 8$ fixed centers, arbitrarily located within the input domain. Each RBF kernel has a fixed 1-dimensional Gaussian "covariance" of $\mathcal{S}_i = 1.0$. Finally, we assume identical independent Gaussian priors on the model parameters $\vec{a}$, i.e. $\Sigma_{\mathcal{F}} = \mathbf{I}_K = \mathbf{I}_8$, where $\mathbf{I}_K$ stands for a $K \times K$ *identity* covariance matrix.

2. **The approximation and target function classes have slightly different priors.** We use a similar RBF approximation function class with $K = 8$ fixed centers and a similar 1-dimensional Gaussian kernel "covariances" of $\mathcal{S}_i = 1.0$ for the centers. Each center is slightly displaced from its true location (i.e. its location in the target function class) by a random distance with Gaussian standard deviation $\sigma = 0.1$. The learner's priors on model parameters ($\Sigma_{\mathcal{F}} = 0.9\mathbf{I}_8$) are also slightly different from that of the target class ($\Sigma_{\mathcal{F}} = \mathbf{I}_8$).

The two simulations proceed as follows: We randomly generate 5000 target RBF functions according to the *target* model priors described above. For each target function, we collect data sets with noisy y values ($\sigma_s = 0.1$) ranging from 3 to 50 samples in size, using both the active and passive sampling strategies. We then obtain a regularized estimate of the target function for each data set using Equation 4.16, and finally, we compute the actual integrated squared difference between the target and its estimate as an approximation error



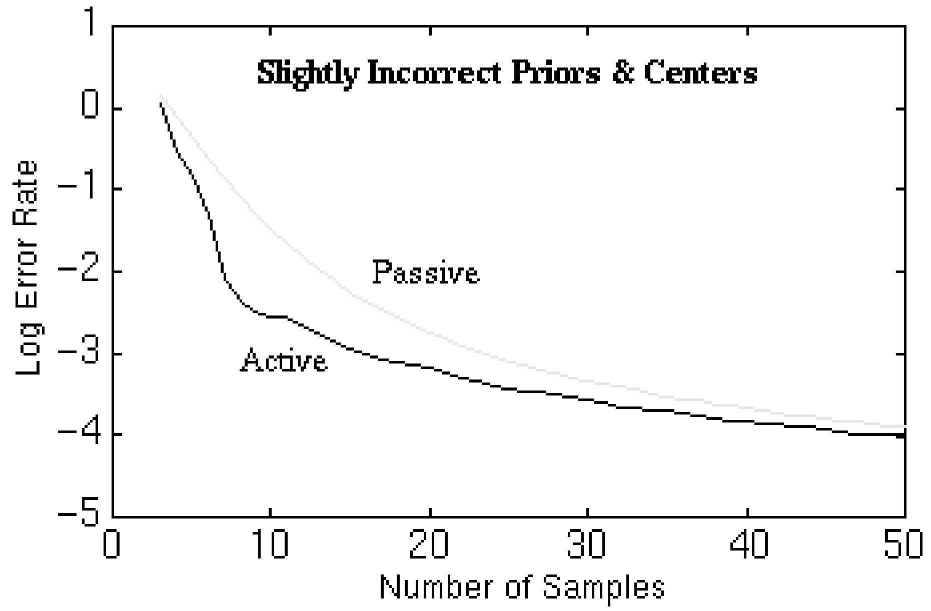


Figure 4-9: Comparing active and passive learning average error rates for Gaussian RBF approximators with $K = 8$ centers. **Top graph:** We use the same priors on the target and approximation function classes. **Bottom graph:** The target and approximation function classes have slightly different center locations and priors on model parameters. In both cases, the active learner has a lower average error rate than the passive learner for the same number of data points.

measure. The graphs in Figure 4-9 plot the average error rates (over the 5000 different target functions) for both the active and passive learners as a function of data sample size. The upper graph shows the learner using exact priors, i.e, that of the target class, while the lower graph is for the case of slightly *incorrect priors*. In both cases, the active learner has a lower average error rate than the passive learner for the same number of data points. This is especially true for small data sets.

4.4 Active Example Selection and the "Boot-strap" Paradigm

Recall from Section 4.2.3 that our active learning strategy chooses its next sample location by minimizing the *total output uncertainty* cost function in Equation 4.7. For convenience, we reproduce the relevant expressions below:

$$\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}}) = \int_{-\infty}^{\infty} \mathcal{P}(y_{n+1}|\vec{x_{n+1}}, \mathcal{D}_n) E_{\mathcal{F}}[\delta(\hat{g_{n+1}}, g)|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})] dy_{n+1}. \quad (4.18)$$

where:

$$E_{\mathcal{F}}[\delta(\hat{g}, g)|\mathcal{D}_n] = \int_{g\in\mathcal{F}} \mathcal{P}(g|\mathcal{D}_n)\delta(\hat{g}, g)dg = \int_{g\in\mathcal{F}} \mathcal{P}_{\mathcal{F}}(g)\mathcal{P}(\mathcal{D}_n|g)\delta(\hat{g}, g)dg.$$

and:

$$\mathcal{P}(y_{n+1}|\vec{x_{n+1}}, \mathcal{D}_n) \propto \int_{f\in\mathcal{F}} P(\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})|f)\mathcal{P}_{\mathcal{F}}(f)df.$$

Clearly, from the three equations above, the cost function $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$ may not have a simple analytical form for many approximation function classes. The current active learning formulation may therefore be computationally intractable for arbitrary approximation function classes in general.

One way of making the active learning task computationally tractable is to define simpler but less "complete" cost functions for measuring the potential utility of new sample locations. To conclude this chapter, we look at one such simplification approach and show how it leads to the "boot-strap" example selection strategy we used for training object and pattern detection systems. We also show empirically that even though the "boot-strap" strategy may be "sub-optimal" in its choice of new examples, it still outperforms random sampling in training a frontal face detection system. As such, we maintain that the "boot-strap" strategy is still an effective means of sifting through unmanageably large data sets that would otherwise make learning intractable.

4.4.1 A Simpler Example Utility Measure

Let g be an *unknown* target function that we want to estimate by means of an approximation function in $\mathcal{F}$, $\mathcal{D}_n = \{(\vec{x_i}, y_i) \in \Re^d \times \Re | i = 1, \ldots, n\}$ be the set of n data points seen so far, and $\hat{g_n} \in \mathcal{F}$ be the current regularized estimate for g. Recall from Section 4.2.3 that our learning goal is to minimize an *expected misfit* notion between g and its regularized solution, and our optimal sampling strategy chooses the next input location $\vec{x_{n+1}} \in \Re^d$ that best minimizes $E_{\mathcal{F}}[\delta(\hat{g_{n+1}}, g) | \mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})]$, the EISD between g and its new estimate $\hat{g_{n+1}}$.

We now describe a different example selection *heuristic*, based on a simpler but less comprehensive example utility measure, that also attempts to efficiently reduce the *expected misfit* between g and $\hat{g_{n+1}}$. Let $\mathcal{L}(\hat{g_n}|\mathcal{D}_n, \vec{x})$ be a *local* "uncertainty" measure for the current estimate $\hat{g_n}$ at $\vec{x}$:

$$\mathcal{L}(\hat{g_n}|\mathcal{D}_n, \vec{x}) = \int_{g \in \mathcal{F}} \mathcal{P}(g|\mathcal{D}_n)(\hat{g_n}(\vec{x}) - g(\vec{x}))^2 dg \tag{4.19}$$

Notice that unlike the EISD which globally characterizes an entire approximation function, $\mathcal{L}(\hat{g_n}|\mathcal{D}_n, \vec{x})$ is just a *local* "error bar" measure between g and $\hat{g_n}$ only at $\vec{x}$. In a loose sense, one can view the *local* "error bar" $\mathcal{L}(\hat{g_n}|\mathcal{D}_n, \vec{x})$ as information that the learner *lacks* at input location $\vec{x}$ for an exact solution. Given such an interpretation, one can also regard $\mathcal{L}(\hat{g_n}|\mathcal{D}_n, \vec{x})$ as an example utility measure, because the learner essentially gains the information it lacks at $\vec{x}$ by sampling there. The new example selection *heuristic* can thus be formulated as choosing the next sample where the new example's *utility value* is greatest, i.e., sampling next where the learner lacks most information:

$$\vec{x_{n+1}} = \arg\max_{\vec{x} \in \Re^d} \mathcal{L}(\hat{g_n}|\mathcal{D}_n, \vec{x}) = \arg\max_{\vec{x} \in \Re^d} \int_{g \in \mathcal{F}} \mathcal{P}(g|\mathcal{D}_n)(\hat{g_n}(\vec{x}) - g(\vec{x}))^2 dg \tag{4.20}$$

We stress again that the new example selection *heuristic* differs from our original active learning formulation only in the example utility cost function it uses. The new heuristic uses a simpler example utility measure, based on the *current* estimate's *local* uncertainty properties instead of the *new* estimate's expected *global* uncertainty properties. In doing so, it implicitly assumes the following:

1. The learning goal is still to minimize the *expected misfit*, i.e., the *total output uncertainty* between g and its estimate $\hat{g}$.

2. One can bring about a proportionate decrease in the *new* estimate's *global* output uncertainty level by *locally* reducing the *current* estimate's output uncertainty at some arbitrary location.

3. There is a better chance of significantly reducing the *local* output uncertainty at a point whose current uncertainty level is high. Furthermore, one can best reduce the *local* uncertainty level at a point by sampling directly at the point.

4. It follows from the previous assumptions that one can most efficiently minimize the new estimate's *global* uncertainty level by gathering new data where the current estimate's *local* output uncertainty level is highest.

Although the above assumptions appear intuitively reasonable, one should still be able to find approximation function classes that do not meet the above assumptions. This suggests that in general, the new *heuristic* may still be a "sub-optimal" sampling strategy with respect to the active learning goal of maximally reducing the *expected misfit* between g and $\hat{g}$ with each new data point. Nevertheless, Equation 4.19 is clearly a much simpler example utility cost function than Equation 4.18, which makes the new heuristic computationally tractable for a much larger range of approximation function classes.

Are there approximation function classes for which the new heuristic and the original active example selection strategy are functionally equivalent? MacKay [58] has shown that if one approximates the current a-posteriori model parameter distribution, i.e. $\mathcal{P}(\vec{a}|\mathcal{D}_n) \equiv \mathcal{P}(g(\cdot,\vec{a})|\mathcal{D}_n)$, as a multi-dimensional Gaussian probability density centered at $\hat{\vec{a}}$, the optimal model parameter estimate, then minimizing the new estimate's *global* uncertainty level reduces to sampling where the current estimate's "error bars" are greatest. MacKay has also observed that for *linear* approximation function classes (i.e., one for which $g(\vec{x},\vec{a}) = \sum_{i=1}^{K} a_i \psi_i(\vec{x})$) with quadratic penalty functions, $\mathcal{P}(\vec{a}|\mathcal{D}_n)$ is *exactly* a multi-dimensional Gaussian probability density centered at $\hat{\vec{a}}$, which in turn suggests that the two sampling strategies are computationally equivalent for such approximation function classes. We refer the interested reader to MacKay's work [58] for further details.

4.4.2 Example Selection in a Real Pattern Detection Training Scenario

We now discuss a variant of the simplified example selection heuristic, used for training a frontal view human face detection system. Recall from Section 2.6.2 that in order to train a face detection system with finite computation resource, one must first acquire a comprehensive but tractably small database of "face" and "non-face" example patterns. For "face" patterns, one can simply collect all frontal face views from mugshot databases and other image archives, and still have a manageably small data set. For "non-face" patterns, the task is more tricky. In essence, any normalized window pattern that does not tightly contain a frontal human face is a valid "non-face" training example. Clearly, our "non-face" example set can grow intractably large if we should include every available "non-face" image patch in our training database.

Notice that our learning scenario for face detection differs slightly from the original active learning scenario presented earlier. In the original setting, one assumes that data measurements are relatively expensive or slow, and we seek the next sample location that best maximizes the expected amount of information gained. In our current scenario, we have an immense amount of available "non-face" data from which we wish to select a small training sample most useful for our learning task. The learner in our face detection scenario also has an added advantage: it already knows the actual output value (i.e., class label) of every candidate "non-face" data point even before they are selected.

Clearly, the current learning scenario reduces to the original active learning setting if we ignore output values (i.e., class labels) of the candidate data points when deciding which new patterns to select. Despite apparent differences in form, both learning scenarios address the same central issue, namely how a learner can select new examples intelligently by estimating the utility of candidate data points. In fact, we shall see shortly that one can still borrow ideas developed for the original active learning scenario to approach example selection in the face detection scenario.

4.4.3 The "Boot-strap" Paradigm

To constrain the number of "non-face" patterns in our training database, we introduced in Section 2.6.2 a "boot-strap" paradigm that incrementally selects "non-face" patterns highly relevant to the learning problem. The "boot-strap" strategy reduces the number of "non-face" patterns needed to train a highly robust face detector. We reproduce the idea below:

1. Start with a small and possibly highly non-representative set of "non-face" examples in the training database.

2. Train a face classifier to output a value of '1' for face examples and '0' for non-face examples using patterns from the current example database.

3. Run the trained face detector on a sequence of images with no faces. Collect all (or a random subset of) the "non-face" patterns that the current system wrongly classifies as "faces" (i.e., an output value of > 0.5). Add these "non-face" patterns to the training database as new negative examples.

4. Return to Step 2.

More generally, one can use the "boot-strap" paradigm to select useful training examples from either pattern class in an arbitrary pattern detection problem:

1. Start with a small and possibly highly non-representative example set in the training database.

2. Train a pattern classifier to output a value of '1' for positive examples and '0' for negative examples using patterns from the current example database.

3. Run the trained pattern classifier on a sequence of images. Collect all (or a random subset of) the wrongly classified patterns and add them to the training database as new correctly labeled examples.

4. Return to Step 2.

At the end of each iteration, the "boot-strap" paradigm augments the current data set with new patterns that the current system classifies wrongly. We argue that this strategy of collecting wrongly classified patterns as new training examples is reasonable, because one can expect these new examples to improve the classifier's performance by steering it away from its current mistakes.

One can reason about the "boot-strap" paradigm as a variant of the previously discussed simplified example selection heuristic. First, as in the original active learning spirit, "boot-strapping" attempts to select only high utility examples for training. During, each example selection step, if "boot-strapping" were to follow the active learning procedure *exactly*, then it should select only the highest utility example available and continue the training process. Instead, we make the "boot-strap" paradigm less restrictive by allowing the learner to select one or more "high utility" examples between training cycles. Notice that the highest utility example may not even be among those selected. Second, the simplified heuristic estimates an example's utility by computing $\mathcal{L}(\hat{g_n}|\mathcal{D}_n, \vec{x})$, the *local* "error bar" measure of Equation 4.19. In "boot-strapping", we take advantage of the already available output value (i.e., class label) at each candidate location to implement a coarse but very simple *local* "error bar" measure for selecting new examples. Points that are classified correctly have low actual output errors. We assume that these points also have low uncertainty "error bars" and so we ignore them as examples containing little new information. Conversely, points that are wrongly classified have high actual output errors. We conveniently assume that these points also have high *local* uncertainty "error bars" and are therefore "high utility" examples suitable for the learning task. We stress again that we are assuming actual output errors and *local* uncertainty "error bars" are highly correlated measurements, which may not always be a valid assumption.

4.4.4 The "Boot-strap" Paradigm and Epsilon Focusing

The "boot-strap" paradigm is also closely related to a recently developed active example selection technique, called *epsilon-focusing* [65]. We briefly highlight their similarities and differences below:

Let $\mathcal{F}$ be a function class on some input domain $\mathcal{X}$, and $h_{\mathcal{F}} : [0,1] \mapsto [0,1]$ be a real valued non-decreasing function characterizing a *local focusing* property of the concept class $\mathcal{F}$. Suppose we want to PAC learn [98] an unknown target $g \in \mathcal{F}$ to within an ϵ *normalized* error rate (i.e., $0 \leq \epsilon \leq 1$) at $(1-\delta)$ confidence, then by the standard Vapnik-Chervonenkis theorem [102], the learner has to draw at most $O\left((1/\epsilon^2)\ln(1/\delta)\right)$ training examples.

The *Epsilon-focusing* idea works as follows: Rather than PAC learning g in one step by collecting the requisite number of examples for ϵ and δ, we split the learning task into k stages. During the first stage, the learner collects a smaller number of examples to make a looser ($\epsilon^{1/k}$ error) estimate of g, but with *higher* confidence $(1 - \delta/k)$. Based on the intermediate hypothesis, the learner then derives an input region of interest where it collects more examples for further training. In general, during the j^{th} stage, the learner estimates g to an $\epsilon^{j/k}/h_{\mathcal{F}}(\epsilon^{(j-1)/k})$ error rate with confidence $(1 - \delta/k)$ by drawing new examples from the region of interest derived during the previous stage. In so doing, the learner keeps refining its estimate by drawing examples from successively smaller input regions and eventually converges onto the target function. Niyogi [65] has shown that the sample complexity for a k-stage *epsilon-focusing* task is:

$$\mathcal{E}(\epsilon, \delta, k) = O\left(\sum_{j=1}^{k}\left(\frac{h_{\mathcal{F}}(\epsilon^{(j-1)/k})}{\epsilon^{j/k}}\right)^2 \ln(k/\delta)\right),$$

which is smaller than the sample complexity for a one step learning task.

There are some striking similarities between the "boot-strap" paradigm and the *epsilon-focusing* idea. First, like *epsilon-focusing*, the "boot-strap" paradigm also breaks the training process into several stages. During each training cycle, the "boot-strap" learner attempts to better estimate its unknown target by collecting new examples that the current system classifies wrongly. One can view these mis-classified examples as data samples from an input *region of interest* where the current estimate differs from the target. Because the learner improves its estimate with each training cycle, the *region of interest* in "boot-strapping" also becomes successively smaller, and so like *epsilon focusing*, the learner eventually converges onto the target function.

Unlike *epsilon-focusing*, the "boot-strap" paradigm does not work towards a specified error bound ϵ and confidence measure δ. In fact, for many real world learning problems like our face detection scenario, it may be difficult if not impossible to actually determine ϵ and δ. Instead, the "boot-strap" learner only seeks to reduce ϵ with each training cycle, and like *epsilon-focusing*, it improves its estimate more quickly than passive learning because it only draws subsequent examples from successively smaller regions of interest.

4.4.5 Sample Complexity of "Boot-strapping"

Does the "boot-strap" strategy yield better classification results with fewer training examples than a passive learner that receives randomly drawn patterns? We show empirically that this is indeed the case for our frontal face detection learning scenario. Using our pro-

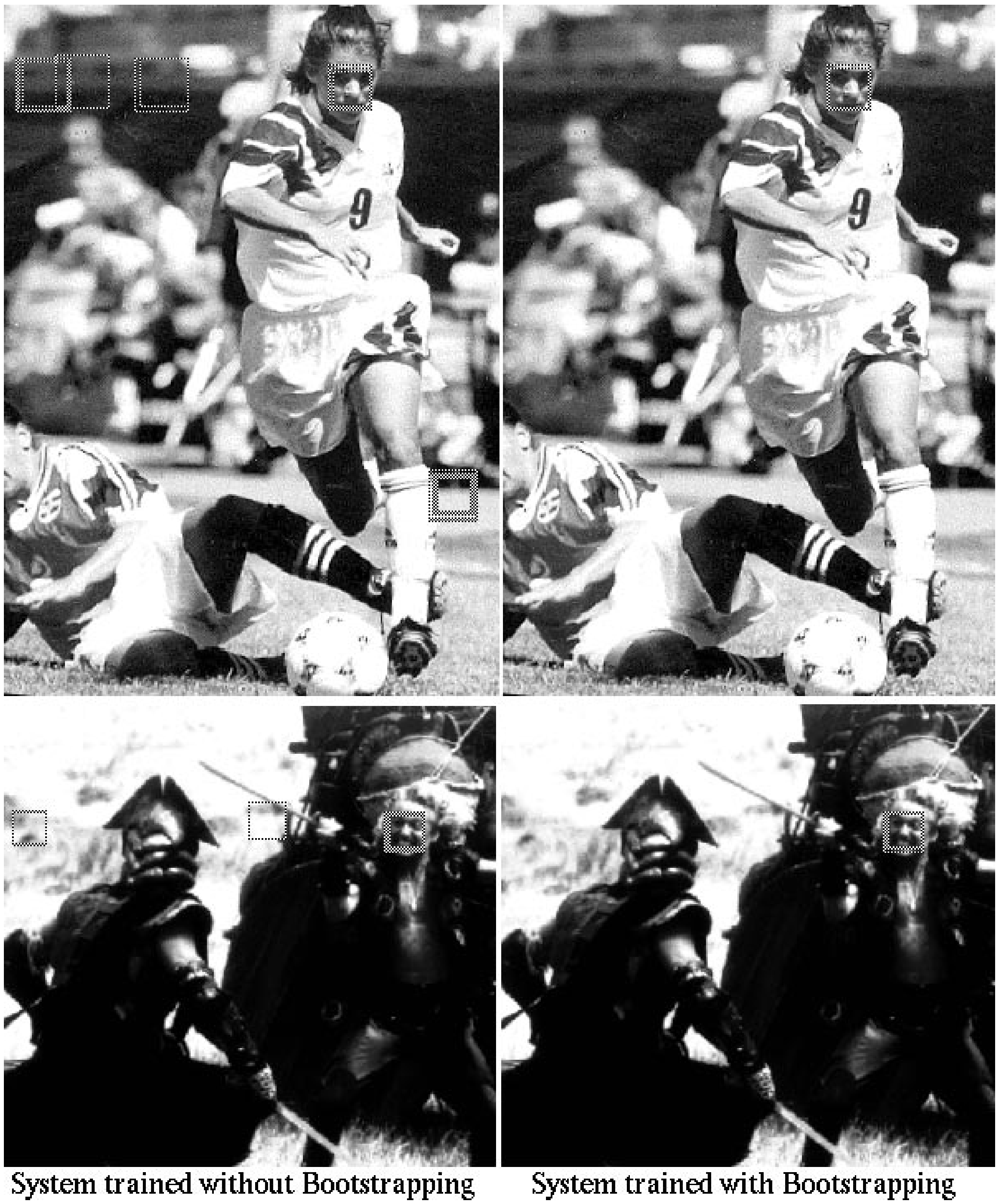

System trained without Bootstrapping
System trained with Bootstrapping

Figure 4-10: Comparing face detection results for two systems: one trained without "boot-strapping" and the other with "boot-strapping". The system trained without "boot-strapping" does a poorer job at discarding non-face patterns. **Top pair:** The system without "boot-strapping" finds the frontal face correctly but also makes four false detects — three near the top-left image corner and one near Mia Hamm's left knee. **Bottom pair:** The system without "boot-strapping" makes two false detects in the complex background, left of Iolaus' face.

posed object and pattern class detection approach presented in Chapter 2, we trained a frontal view face detection system in two “boot-strap” cycles. The final training database contains 4150 “face” patterns and over 40000 “non-face” patterns, of which about 6000 were selected during the second “boot-strap” cycle. As a control, we trained a second system without “boot-strapping”. The second training database contains the same 4150 “face” patterns and another 40000 randomly selected “non-face” patterns. We used the same “face” and “non-face” Gaussian clusters from the first system to model the target and near-miss pattern distributions in the second system.

We ran both systems on a test database of 23 cluttered images with 147 frontal faces. The first system missed 23 faces and produced 13 false detects, while the second system had 15 missed faces and 44 false alarms. Notice that the first system trained with “boot-strapping” has a lower *total* mis-classification rate than the control trained without “boot-strapping”. The “boot-strap” system misses more faces but has a much smaller false alarm rate for non-faces. This is because we have used “boot-strapping” to only select better “non-face” patterns while leaving the total number of “face” patterns unchanged. Consequently, the system is better able to reject non-face patterns at a slight expense of correctly detecting faces.

Notice also that we have in fact helped the control by modeling its “near-miss” distribution with “non-face” clusters from the first system. In Section 3.2, we demonstrated that “non-face” clusters of near-misses found by “boot-strapping” give rise to a very discriminative set of additional features for separating face and non-face patterns. Because the control does not have a well chosen “near-miss” data set, we argue that it could not have independently produced such a highly discriminative set of additional classification features. As such, one can reasonably expect even higher mis-classification rates from a non “boot-strapped” system.

Our comparison suggests that even though the “boot-strap” strategy may be “sub-optimal” in choosing new training patterns, it is still a very simple and effective technique for sifting through unmanageably large data sets for useful examples.

Chapter 5

Extensions

In this thesis, we presented a learning based approach for detecting classes of objects and patterns with spatially well defined boundaries in images. The approach first builds a distribution-based model of the target pattern class in an appropriate feature space to describe the target's variable image appearance. It then learns from examples a similarity measure for matching new patterns against the distribution-based target model. The approach makes few assumptions about the target pattern class and should therefore be fairly general, as long as the target class has predictable image boundaries. We showed that this is indeed the case by demonstrating the technique on a few pattern detection and recognition problems.

We conclude this thesis by considering two possible extensions to our proposed object and pattern detection approach. The first looks at how one can combine output results from several pattern detectors to achieve better detection rates with fewer false alarms. Recently, Rowley et. al. [76] have applied some simple arbitration techniques to a few face detection networks trained with "boot-strapping", and have reported very impressive face detection results. We shall discuss a more powerful network combination and arbitration scheme, called *network boosting* [79] [30], that can potentially lead to systems with arbitrarily high correct classification rates. The second extension is about building hierarchical architectures for dealing with occlusion, and for detecting pattern classes with less predictable boundaries. Recall from an earlier discussion in Chapter 1 that one can detect pattern classes with moderately variable boundaries by defining simpler sub-pattern classes that can be easily isolated and identified in an image. One of the main difficulties in this approach is to reliably identify and locate full target patterns from the spatial distribution and composition of these sub-patterns in an image. In this chapter, we shall briefly explore some possible techniques for performing such a task.

5.1 Improving Detection Rates by Combining Classifiers

One way of building more accurate and reliable decision procedures is to combine multiple classifiers for the given task. Individual classifiers tend to have their own advantages and deficiencies. A collective decision made by a classifier ensemble should therefore be more reliable than one made by any individual classifier alone.

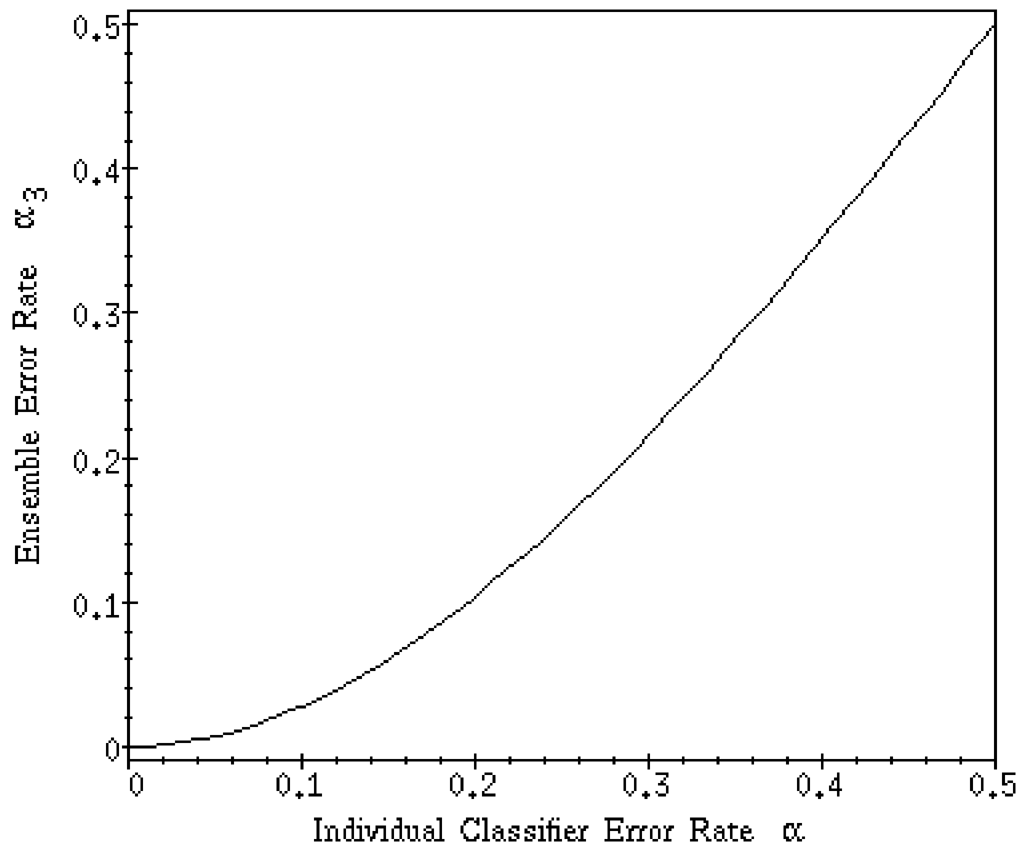
Network *boosting* [79] [30] is a classifier combination technique, based on the *probably approximately correct* (PAC) learning framework [98], that converts a learning architecture with finite error rate into an ensemble of learning machines with an arbitrarily low error rate. The idea works as follows:

Suppose we have three classifiers with almost identical error rates that produce *independent* classification results on a given task. One can obtain a lower error rate decision procedure by passing each new pattern through all three classifiers and using the following output voting scheme: If the first two classifiers agree, then return the common label; otherwise, return the label as given by the third classifier. Notice that what we have just described is simply a *majority* voting scheme on the three classifier output labels. In theory however, one can repeatedly embed this voting process to obtain ensembles of $3^2 = 9$ classifiers, $3^3 = 27$ classifiers, $3^4 = 81$ classifiers and so on, with decreasingly small error rates.

Recently, Drucker et. al. [30] have applied this classifier combination idea to optical character recognition (OCR) problems. They constructed an ensemble of three OCR neural networks and reported a significant overall improvement in character recognition results. We believe one can similarly combine multiple pattern recognizers like ours to build arbitrarily robust object detection systems.

5.1.1 Network Boosting

We now derive the *boosting* procedure which arises from a variant of the PAC learning framework, known as the *weak learning model* [50]. Let the three classifiers in the above



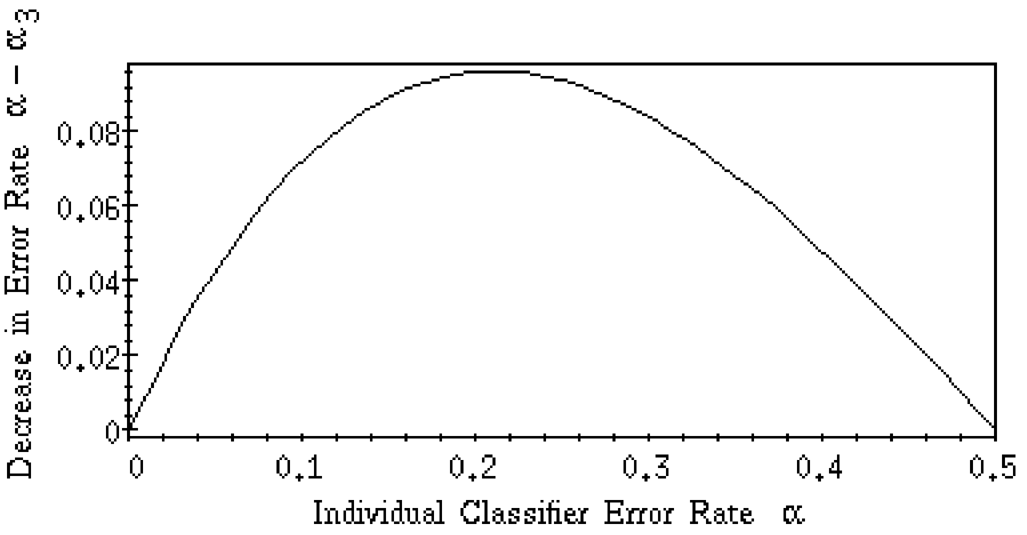


Figure 5-1: **Top:** A 3-classifier ensemble has a lower error rate α_3 than a single classifier α over a wide range of individual classifier error rates $0 < \alpha < 0.5$. Note: a perfect classifier has $\alpha = 0$ while random guessing has $\alpha = 0.5$. Boosting does not improve performance in these two cases. **Bottom:** The actual difference in error rates between a single classifier and a 3-classifier ensemble over the same range of individual classifier error rates.

example have *independent* error rates $\alpha < 0.5$. Using the output voting scheme described earlier, the resulting classifier ensemble makes a mistake only when:

1. Both the first and second classifiers mis-label the input pattern. This occurs with probability α^2.

2. The first and second classifiers disagree (with probability $2\alpha(1-\alpha)$), and the third classifier wrongly labels the input pattern (with probability α). This occurs with joint probability $2\alpha^2(1-\alpha)$.

Adding probabilities for the two cases above, we arrive at the following error rate $\alpha_3 = 3\alpha^2 - 2\alpha^3$ for the classifier ensemble, which can be significantly smaller than the individual classifier error rate for $0 < \alpha < 0.5$.

Figure 5-1 plots the difference in performance between a 3-classifier ensemble and a single classifier over a range of individual classifier error rates $0 < \alpha < 0.5$. The graphs confirm that in order to construct an arbitrarily robust system with *boosting*, the learner only has to produce a sufficiently large number of *independent* classifiers whose individual error rates are slightly better than random guessing ($\alpha = 0.5$).

5.1.2 Maintaining Independence between Classifiers

One major assumption in *weak learning* and *boosting*, is that all classifiers being combined make *independent* errors on new input patterns. When two or more classifiers in an ensemble make correlated predictions, one can show that boosting reduces the combined error rate less significantly. Furthermore, if the first two classifiers produce fully correlated results, both classifiers will always assign identical labels to new patterns, and boosting does not improve the ensemble error rate at all. Hence, when applying boosting to any non-trivial learning problem, one should always try to maintain a reasonable degree of independence between individual classifiers.

Drucker et. al. [30] have proposed a network ensemble training procedure that helps maintain output independence between individual classifiers. Their idea is to train each network classifier with a different example distribution.

1. Assume that the learner has access to an "infinite" stream of training patterns.

2. Draw from the stream a requisite number of data samples and train the first network classifier.

3. Use the first network classifier together with the "infinite" example stream to generate a second training set as follows: Flip a fair coin. If the coin is heads, draw examples from the data stream until the first network mis-classifies a pattern and add this pattern to the second training set. If the coin is tails, draw examples until the first network correctly classifies a pattern and add the pattern to the second training set. When the second data set is sufficiently large, use it to train the second network classifier.

4. Generate a third training set using the first two classifiers as follows: Propagate examples from the "infinite" data stream through the first two classifiers. Add each new pattern to the third training set only if the two classifiers disagree on the output label. When the third data set is sufficiently large, use it to train the third network classifier.

Drucker et. al. have successfully applied the above procedure to train independent OCR networks for their boosting experiment. Clearly, the procedure makes a very strong assumption that the learner has access to an "infinite" data stream. For an OCR scenario, this assumption might still be reasonable because there are very large digit and character databases available that can serve as an almost "infinite" example source. Recently, Simard et. al. [83] have also derived some useful transformations for artificially generating additional digit and character patterns from existing database examples.

In other object and pattern detection problems, the learner may not have access to much data, so the training paradigm outlined above may not be feasible. This is because after training the first classifier, there may not be enough remaining examples to generate independent training sets for the second and third classifiers, unless the individual classifier error rates are all very high. We suggest three possible ways of training several independent classifiers with limited data:

1. Choose a classifier architecture with multiple plausible solutions for a given data set. A highly connected multi-layer perceptron net can be one such architecture. When trained with a backpropagation algorithm, a multi-layer perceptron net initialized with

a different set of parameters can converge onto a different set of weights (i.e. a different solution). Assuming that nets with different weights make relatively independent classification errors, one can still perform boosting using several multi-layer perceptron nets trained on the same data set.

2. Build each classifier using a different machine architecture. Because each machine architecture implements a different approximation function class, one can expect each classifier to generalize differently between existing training examples, and hence make different classification mistakes.

3. Train each classifier on a different set of input features. Feature measurements help enhance certain image properties that can act as useful cues for distinguishing between pattern classes. Classifiers that use different feature measurements are thus relying on different image attributes to identify patterns, and should therefore make different classification mistakes. In Chapters 2 and 3, our face and eye detection systems perform classification in a view-based feature space. To improve detection results with boosting, one can train and combine additional face and eye pattern recognizers that operate on different input image representations, such as a gradient based map or other filtered versions of the input pattern.

5.2 Building Hierarchical Architectures from Simple Detectors

We consider next another classifier combination scheme that builds hierarchical pattern detection architectures using multiple simple pattern recognizers. One classical object modeling paradigm represents complex bodies and patterns in terms of simpler parts. An early example of such a paradigm by Brooks et. al. [18] uses generalized cylinders to describe arbitrarily shaped articulate objects. More recently, Brunelli and Poggio [19] have also proposed a similar face recognition approach that decomposes face patterns into more "rigid" sub-features suitable for template matching.

In this study, we examine a hierarchical scheme that uses multiple spatially well-defined pattern recognizers like ours to take on more complex pattern detection tasks. The overall idea works as follows: We recursively divide a complex target pattern class into several simpler sub-pattern classes that can each be further divided or individually detected in an image. Each sub-pattern class may correspond to some meaningful part of the original target object, such as an eye on a human face or an arm of a humanoid body. At a later stage, we analyze the spatial arrangement of these sub-pattern classes in the image to identify and locate instances of the full target.

We see two advantages in pursuing such a pattern representation and detection scheme:

1. **Handling partial occlusion.** The face and eye systems we presented earlier respond poorly to partially occluded target patterns, because we have trained each system only on full target objects without occlusion. One can account for partial occlusion by adding occluded target examples to the training set. Unfortunately, such an approach can be extremely cumbersome and memory intensive, because one essentially has to collect or generate new target examples with a representative distribution of occluding mask patterns and foreground textures. An alternative approach to occlusion uses a hierarchical paradigm like the above, which divides target patterns into several smaller components that can each be independently detected in an image. The approach assumes one can still successfully find a sufficient number of sub-patterns on a partially occluded target to infer its presence. Computationally, this second approach appears more tractable because it only involves training several independent sub-pattern detectors and an additional stage for integrating their output results.

2. **Detecting articulate pattern classes with less predictable boundaries.** Human faces and eyes are target classes with highly predictable spatial boundaries. Human bodies, on the other hand, are an example of an articulate pattern class with less predictable spatial boundaries. Our proposed object and pattern detection approach does not handle articulate pattern classes with arbitrarily shaped boundaries well, because one cannot simply "segment" these patterns from their variable background images using one of several fixed shape masks. In general, pattern segmentation is still an unsolved computer vision problem, so one may not even have a reliable means of extracting entire articulate patterns from an image for a "template" based detection approach like ours. For some time, vision researchers have considered a hierarchical approach, similar to the classifier combination scheme above, for detecting arbitrarily shaped pattern classes without explicitly performing segmentation. The approach

represents an arbitrarily shaped target class as simpler components with predictable spatial boundaries. One can then use our proposed object and pattern detection scheme to deal with these simpler components, and later analyze the detection results to identify and locate instances of the complex target.

Clearly, the key issue in this hierarchical pattern detection approach is one of integrating intermediate output results for identifying full target patterns. In general, implementing an output combination stage involves: (1) finding sets of individual sub-patterns that arise from the same target; (2) devising a scheme for representing geometric relationships between the individually detected sub-patterns; and (3) determining the significance of each sub-pattern and co-occurrences of sub-patterns as cues for identifying and locating the full target. We now look at some useful tools for implementing an output combination stage.

5.2.1 Combining Sub-Pattern Detection Results with Multi-Layer Perceptron Nets

Multi-layer perceptron nets are a convenient machine architecture for learning and encoding complex relationships between individual features or sets of features in a classification task. For a highly structured and relatively inarticulate target class like human faces, there is usually very little variation in the position and orientation of its sub-pattern components. When detecting such target classes, one can simply take advantage of their highly predictable spatial structure to search for sub-patterns only at specific image locations; i.e., the first problem of finding and grouping together sub-patterns from the same target is trivial.

To identify instances of the target from its sub-pattern components, one can train an appropriately structured multi-layer perceptron net whose input features are all the output values from the individual sub-pattern detectors. Because we are applying each sub-pattern detector at a fixed spatially offset image location, there is no need to recover position and orientation values for each image sub-pattern as input features to the multi-layer perceptron net classifier. Hence, the classifier training process takes care of the third output combination task above, while the second task is usually irrelevant for highly structured and relatively inarticulate target pattern classes.

5.2.2 Handling Sub-Patterns with Variable Position and Orientation

Unlike human faces, human bodies are an example of an articulate pattern class with less predictable spatial boundaries. One can still represent an articulate target class as simpler components with predictable spatial boundaries that can each be independently detected in an image. However, there can be a significant amount of variation in the position and orientation of each sub-pattern component. So, to identify and locate these arbitrarily shaped target patterns from their simpler components, one must also deal with the first and second output combination issues described above; i.e., the problems of finding sub-patterns in an image that belong to the same target, and representing geometric relationships between them as additional cues for identifying the target.

We shall first address the second and simpler issue by introducing position and orientation attributes for each sub-pattern as additional classification features. If one uses a "template matching" like detection paradigm that tests for the target at candidate image locations and orientations, then one can represent each sub-pattern's position in a translationally invariant fashion as its spatial offset from the hypothesized target center. Similarly, one can also describe the orientation of each sub-pattern in a rotationally invariant fashion: define a fixed reference direction for each sub-pattern class, and compute for each image sub-pattern its angular displacement with respect to the hypothesized target orientation. Thus, for an articulate target class, we recover for each of its components, a set of output combination features that includes a detection output value, and an additional vector of translationally and rotationally invariant position and orientation attributes. The additional position and orientation attributes help capture geometric relationships between the individually detected sub-patterns, which the output combination stage relies on as additional cues for identifying the target.

5.2.3 Finding Sets of Sub-Patterns from an Articulate Target

The discussion in Section 5.2.2 assumes a reasonable scheme for finding and grouping together sub-patterns in an image that belong to the same articulate target object. Unfortunately, we believe existing techniques for performing such a task in general are still very much ad hoc and unreliable at best. We conclude this thesis by referring to two current areas of research that may lead to feasible and robust search paradigms for image sub-patterns. Our discussion here will, however, only be speculative and brief, since current research trends in both these areas still appear open and highly exploratory in nature.

Grouping

Grouping [57] [49] is a process that identifies sets of features in a cluttered image likely to have arisen from a single object. It serves mainly as a pre-processing stage that speeds up object recognition by reducing the number of different image feature combinations the recognition stage has to consider while testing for the target. Lowe [57] first demonstrated the idea on an early computational recognition system that makes explicit use of grouping. Jacobs [49] later extended the idea to a geometric model-based object recognition domain. Typical grouping schemes operate on simple low-level image features that their object recognition systems use, such as intensity edges, junctions and corners. Often, these schemes rely heavily on prior assumptions about feature configurations that make them likely target candidates. Such assumptions may be based on domain specific knowledge, like common features that tend to co-exist in the target, or "general" observations like certain image feature configurations being more "salient" in real scenes.

In the hierarchical pattern detection scheme we are considering here, one can treat the sub-pattern components of an articulate target class as highly specific and sophisticated model features, similar in spirit to the simpler features used by current object recognition and *grouping* systems. We have argued earlier that one can reliably detect these sophisticated features using our proposed object and pattern detection approach from Chapter 2. We believe one can also develop similar *grouping* based techniques to identify salient sets of these sophisticated image features that are likely parts from the same target.

Knowledge-based Methods for Directing Search

Over the past twenty to thirty years, *knowledge-based* systems or *expert* systems have been used on a wide range of decision problems, including problems in medical diagnosis [20], molecular structure prediction [56] and software maintenance [81] [82]. In computer vision, researchers have also used rule-based expert systems with rigid object models to direct search for additional image features, based on structures that have already been detected in the scene [18] [17]. The idea works as follows: when the recognition module detects and identifies an image feature as some part of the target, the knowledge-based search module uses the object model to predict new image locations where the recognizer may find other target features. When a sufficient number of target features have been found and "grouped" together in this fashion, the recognizer declares the target present in the scene. We believe one can use the same general techniques developed in this area to also constrain search for the sub-pattern components of an articulate target class.

Like *grouping*, knowledge-based systems also rely heavily on domain specific knowledge, such as common geometric configurations between target components, to predict new image locations for finding additional target features. Recently, researchers have developed a graphical representation for learning structured domain knowledge with statistical data, known as Bayesian Networks [46] [68]. Bayesian Nets are especially suitable for learning and encoding uncertain domain specific knowledge in expert systems. One can construct a Bayesian Net that learns the variable geometric structure of an articulate target class as follows: First, we encode existing information about the articulate target class as a set of graphical nodes and directed arcs. Each graphical node could represent the distribution of some relative location or orientation variable between a pair of sub-pattern components. The directed arcs specify dependencies between the relative position and orientation variables. Next, we use a database of real target examples to update the Bayesian Net, which includes the probability distribution of each node variable and values along each directed arc. Hence, even if our initial domain knowledge about the target class structure may be unreliable or incomplete, one can still improve it through the statistical learning process.

In summary, learning with Bayesian Nets combines the advantages of exploiting existing domain knowledge as in a classical knowledge-based system, and the ability to refine existing knowledge with statistical data as in a traditional connectionist learning-based approach. Research in this area is new and still evolving, and a detailed discussion on applying Bayesian Nets to hierarchical pattern detection is beyond the scope of this thesis. The interested reader should refer to the following papers and others for a more comprehensive guide to the literature [46] [68] [21] [44].

Appendix A

The Active Learning Procedure

This appendix derives the active example selection procedures for polynomial approximators and Gaussian radial basis functions presented in Chapter 4. Specifically, we show the steps leading to Equations 4.11 and 4.17, i.e., the *total output uncertainty* cost functions $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$ for polynomial approximators and Gaussian RBFs respectively.

A.1 Polynomial Approximators

Let $\mathcal{F}$ be a univariate polynomial approximation function concept class with maximum degree K:

$$\mathcal{F} = \{g(x, \vec{a}) = g(x, a_0, \ldots, a_K) = \sum_{i=0}^{K} a_i x^i\}.$$

The model parameters to be learnt are $\vec{a} = [a_0 \, a_1 \, \ldots \, a_K]^{\mathrm{T}}$ and $x \in [x_{\mathtt{LO}}, x_{\mathtt{HI}}]$ is the input variable . The prior distribution on $\mathcal{F}$ is a zero-mean Gaussian distribution with covariance $\Sigma_{\mathcal{F}}$ on the model parameters $\vec{a}$:

$$\mathcal{P}_{\mathcal{F}}(g(\cdot, \vec{a})) = \mathcal{P}_{\mathcal{F}}(\vec{a}) = \frac{1}{(2\pi)^{(K+1)/2}|\Sigma_{\mathcal{F}}|^{1/2}} \exp(-\frac{1}{2}\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}). \tag{A.1}$$

Our task is to approximate an *unknown* target function $g \in \mathcal{F}$ within the input range $[x_{\mathtt{LO}}, x_{\mathtt{HI}}]$ on the basis of *noisy* sampled data: $\mathcal{D}_n = \{(x_i, y_i = g(x_i) + \eta)|i = 1, \ldots, n\}$, where η is an additive zero-mean Gaussian noise term with variance σ_s^2.

A.1.1 The A-Posteriori Function Class Distribution

We first derive the a-posteriori distribution on function class $\mathcal{F}$ given data $\mathcal{D}_n$, i.e., $\mathcal{P}(\vec{a}|\mathcal{D}_n) \propto \mathcal{P}_{\mathcal{F}}(\vec{a})\mathcal{P}(\mathcal{D}_n|\vec{a})$. Since $\mathcal{D}_n$ is sampled under additive zero-mean Gaussian noise with variance σ_s^2, we have:

$$\begin{aligned}\mathcal{P}(\mathcal{D}_n|\vec{a}) &\propto \exp\left(-\frac{1}{2\sigma_s^2}\sum_{j=1}^{n}(y_j - g(x_j,\vec{a}))^2\right)\\ &= \exp\left(-\frac{1}{2\sigma_s^2}\sum_{j=1}^{n}(y_j - \sum_{t=0}^{K} a_t x_j^t)^2\right) \qquad \text{(A.2)}\end{aligned}$$

Let $\bar{\mathbf{x}}_\mathbf{j} = [1\ x_j\ x_j^2\ \ldots\ x_j^K]^{\mathrm{T}}$ be a power vector of the $j^{\mathbf{th}}$ input value. One can expand the exponent term in Equation A.2 as follows:

$$\begin{aligned}\left(y_j - \sum_{t=0}^{K} a_t x_j^t\right)^2 &= y_j^2 + \left(\sum_{t=0}^{K} a_t x_j^t\right)^2 - 2y_i\sum_{t=0}^{K} a_t x_j^t\\ &= y_j^2 + \vec{a}^{\mathrm{T}}(\bar{\mathbf{x}}_\mathbf{j}\bar{\mathbf{x}}_\mathbf{j}^{\mathrm{T}})\vec{a} - y_j\bar{\mathbf{x}}_\mathbf{j}^{\mathrm{T}}\vec{a} - \vec{a}^{\mathrm{T}}\bar{\mathbf{x}}_\mathbf{j} y_j\end{aligned}$$

So:

$$\begin{aligned}\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\left(y_j - \sum_{t=0}^{K} a_t x_j^t\right)^2 &= \frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2 + \vec{a}^{\mathrm{T}}\left(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{x}}_\mathbf{j}\bar{\mathbf{x}}_\mathbf{j}^{\mathrm{T}})\right)\vec{a}\\ &\quad - (\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j\bar{\mathbf{x}}_\mathbf{j}^{\mathrm{T}})\vec{a} - \vec{a}^{\mathrm{T}}(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{x}}_\mathbf{j} y_j)\end{aligned}$$

The polynomial prior distribution $\mathcal{P}_{\mathcal{F}}(\vec{a})$ is given in Equation A.1. The a-posteriori distribution is thus:

$$\begin{aligned}\mathcal{P}(\vec{a}|\mathcal{D}_n) &\propto \mathcal{P}_{\mathcal{F}}(\vec{a})\mathcal{P}(\mathcal{D}_n|\vec{a})\\ &\propto \exp\left(-\frac{1}{2}\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}\right)\exp\left(-\frac{1}{2\sigma_s^2}\sum_{j=1}^{n}(y_j - \sum_{t=0}^{K} a_t x_j^t)^2\right)\\ &= \exp\Bigg[-\frac{1}{2}\Bigg(\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a} + \frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2 + \vec{a}^{\mathrm{T}}\left(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{x}}_\mathbf{j}\bar{\mathbf{x}}_\mathbf{j}^{\mathrm{T}})\right)\vec{a}\\ &\qquad -(\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j\bar{\mathbf{x}}_\mathbf{j}^{\mathrm{T}})\vec{a} - \vec{a}^{\mathrm{T}}(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{x}}_\mathbf{j} y_j)\Bigg)\Bigg]\end{aligned}$$

$$= \exp\left[-\frac{1}{2}\left(\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2 + \vec{a}^{\mathrm{T}}\left(\Sigma_{\mathcal{F}}^{-1} + \frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{x}}_{\mathbf{j}}\bar{\mathbf{x}}_{\mathbf{j}}^{\mathrm{T}})\right)\vec{a} \right.\right.$$
$$\left.\left. -(\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j\bar{\mathbf{x}}_{\mathbf{j}}^{\mathrm{T}})\vec{a} - \vec{a}^{\mathrm{T}}(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{x}}_{\mathbf{j}}y_j)\right)\right] \qquad (A.3)$$

Completing the square in Equation A.3 yields:

$$\mathcal{P}(\vec{a}|\mathcal{D}_n) \propto \exp\left[-\frac{1}{2}\left((\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\Sigma_n^{-1}(\vec{a}-\hat{\vec{a}}) - \hat{\vec{a}}^{\mathrm{T}}\Sigma_n^{-1}\hat{\vec{a}} + \frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2\right)\right] \qquad (A.4)$$

where:

$$\Sigma_n^{-1} = \Sigma_{\mathcal{F}}^{-1} + \frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{x}}_{\mathbf{j}}\bar{\mathbf{x}}_{\mathbf{j}}^{\mathrm{T}}) \qquad (A.5)$$

$$\hat{\vec{a}} = \Sigma_n(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{x}}_{\mathbf{j}}y_j) \qquad (A.6)$$

Notice that neither of the two terms $\hat{\vec{a}}^{\mathrm{T}}\Sigma_n^{-1}\hat{\vec{a}}$ and $\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2$ in Equation A.4 depend on the polynomial model parameters $\vec{a}$. This means that we can rewrite Equation A.4 as:

$$\mathcal{P}(\vec{a}|\mathcal{D}_n) \propto \exp\left[-\frac{1}{2}\left((\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\Sigma_n^{-1}(\vec{a}-\hat{\vec{a}})\right)\right] \qquad (A.7)$$

Clearly, Equation A.7 is multivariate Gaussian in form. To express $\mathcal{P}(\vec{a}|\mathcal{D}_n)$ as a standard probability distribution on $\vec{a}$ that integrates to 1, we simply introduce into Equation A.7 the appropriate normalizing constants:

$$\mathcal{P}(\vec{a}|\mathcal{D}_n) = \frac{1}{(2\pi)^{(K+1)/2}|\Sigma_n|^{1/2}}\exp\left[-\frac{1}{2}\left((\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\Sigma_n^{-1}(\vec{a}-\hat{\vec{a}})\right)\right] \qquad (A.8)$$

Thus, the polynomial a-posteriori function class distribution is a multivariate Gaussian centered at $\hat{\vec{a}}$ (Equation A.6) with covariance Σ_n (Equation A.5).

A.1.2 The Polynomial EISD Measure

Recall from Section 4.2.3 that the *total output uncertainty* cost functions $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$ is simply the *expected* EISD measure between g and its *new* estimate $\hat{g_{n+1}}$, if the learner samples next at $\vec{x_{n+1}}$. We now derive an expression for the EISD between g and its *current* estimate $\hat{g_n}$ given $\mathcal{D}_n$. We shall use this result later to derive an expression for $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$.

The *expected integrated squared difference* (EISD) between an unknown target g and its estimate $\hat{g}$ given $\mathcal{D}_n$ is:

$$E_{\mathcal{F}}[\delta(\hat{g},g)|\mathcal{D}_n] = \int_{g\in\mathcal{F}} \mathcal{P}(g|\mathcal{D}_n)\delta(\hat{g},g)dg$$

where $\delta(\hat{g},g)$ is a standard integrated squared difference measure between two functions over the input space $\Re^d$ or some appropriate region of interest:

$$\delta(\hat{g},g) = \int_{\vec{x}\in\Re^d}(g(\vec{x})-\hat{g}(\vec{x}))^2 d\vec{x}.$$

For our polynomial approximation function class, the *optimal* estimate for g given $\mathcal{D}_n$ has model parameters $\hat{\vec{a}}$ (Equation A.6), since this is where $\mathcal{P}(\vec{a}|\mathcal{D}_n)$ has a global maximum. Let $\hat{\vec{a}} = [\hat{a_0}\ \hat{a_1}\ \ldots\ \hat{a_K}]^{\mathtt{T}}$ and $\bar{\mathbf{x}} = [1\ x\ x^2\ \ldots\ x^K]^{\mathtt{T}}$, one can rewrite $\delta(\hat{g},g)$ in terms of polynomial model parameters as:

$$\begin{aligned}
\delta(\hat{\vec{a}},\vec{a}) &= \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}}[g(x,\vec{a})-g(x,\hat{\vec{a}})]^2dx = \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}}\left[(\sum_{i=0}^{K}a_ix^i)-(\sum_{i=0}^{K}\hat{a}_ix^i)\right]^2dx \\
&= \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}}\left[\sum_{i=0}^{K}(a_i-\hat{a}_i)x^i\right]^2dx = \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}}(\vec{a}-\hat{\vec{a}})^{\mathtt{T}}\bar{\mathbf{x}}\bar{\mathbf{x}}^{\mathtt{T}}(\vec{a}-\hat{\vec{a}})dx \\
&= (\vec{a}-\hat{\vec{a}})^{\mathtt{T}}\left(\int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}}\bar{\mathbf{x}}\bar{\mathbf{x}}^{\mathtt{T}}dx\right)(\vec{a}-\hat{\vec{a}}) \\
&= (\vec{a}-\hat{\vec{a}})^{\mathtt{T}}\mathbf{A}(\vec{a}-\hat{\vec{a}}) \qquad \text{(A.9)}
\end{aligned}$$

where $\mathbf{A}$ is a constant $(K+1)\times(K+1)$ matrix of numbers whose $(i,j)^{\mathtt{th}}$ element is:

$$\mathbf{A}(i,j) = \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}} x^{(i+j-2)}dx$$

Substituting Equations A.8 and A.9 into the EISD expression, we get:

$$\begin{aligned}
E_{\mathcal{F}}[\delta(\hat{g},g)|\mathcal{D}_n] &= E_{\mathcal{F}}[\delta(\hat{\vec{a}},\vec{a})|\mathcal{D}_n] = \int_{\vec{a}\in\Re^{K+1}}\mathcal{P}(\vec{a}|\mathcal{D}_n)\delta(\hat{\vec{a}},\vec{a})d\vec{a} \\
&= \int_{\vec{a}\in\Re^{K+1}}\frac{1}{(2\pi)^{(K+1)/2}|\Sigma_n|^{1/2}}\exp\left[-\frac{1}{2}\left((\vec{a}-\hat{\vec{a}})^{\mathtt{T}}\Sigma_n^{-1}(\vec{a}-\hat{\vec{a}})\right)\right] \\
&\qquad (\vec{a}-\hat{\vec{a}})^{\mathtt{T}}\mathbf{A}(\vec{a}-\hat{\vec{a}})\ d\vec{a} \qquad \text{(A.10)}
\end{aligned}$$

Making the following change of variables: $\vec{q} = \mathbf{A}^{\frac{1}{2}}(\vec{a}-\hat{\vec{a}})$, and noting that the integration bounds on $\vec{q}$ is still $\Re^{K+1}$, Equation A.10 becomes:

$$
\begin{aligned}
E_{\mathcal{F}}[\delta(\hat{g},g)|\mathcal{D}_n] &= E_{\mathcal{F}}[\delta(\hat{\vec{a}},\vec{a})|\mathcal{D}_n] \\
&= \int_{\vec{q}\in\Re^{K+1}} \frac{1}{(2\pi)^{(K+1)/2}|\mathbf{A}|^{1/4}|\Sigma_n|^{1/2}|\mathbf{A}|^{1/4}} \\
&\qquad \exp\left[-\frac{1}{2}\left(\vec{q}^{\,\mathrm{T}}\mathbf{A}^{-\frac{1}{2}}\Sigma_n^{-1}\mathbf{A}^{-\frac{1}{2}}\vec{q}\right)\right]\vec{q}^{\,\mathrm{T}}\vec{q}\,d\vec{q} \\
&= \int_{\vec{q}\in\Re^{K+1}} \frac{1}{(2\pi)^{(K+1)/2}|\Sigma_n\mathbf{A}|^{1/2}} \\
&\qquad \exp\left[-\frac{1}{2}\left(\vec{q}^{\,\mathrm{T}}\mathbf{A}^{-\frac{1}{2}}\Sigma_n^{-1}\mathbf{A}^{-\frac{1}{2}}\vec{q}\right)\right]\vec{q}^{\,\mathrm{T}}\vec{q}\,d\vec{q} \\
&= |\Sigma_n\mathbf{A}| \quad \propto \quad |\Sigma_n| \qquad \text{(A.11)}
\end{aligned}
$$

since $\mathbf{A}$ is just a constant matrix of numbers.

Notice from Equation A.5 that Σ_n depends only on the polynomial function class priors $\Sigma_{\mathcal{F}}$, the output noise variance σ_s^2 and the previously sampled input locations $\{x_1, x_2, \ldots, x_n\}$. It does not depend on the previous y data values actually observed. In other words, the previously observed y data values in $\mathcal{D}_n$ *do not* affect the EISD measure (Equation A.11) for this polynomial function concept class!

A.1.3 The Total Output Uncertainty Measure

The *total output uncertainty* cost function resulting from sampling next at $\vec{x_{n+1}}$ is given by Equation 4.7. We rewrite the expression below in terms of our polynomial model parameters:

$$\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, x_{n+1}) = \int_{y_{n+1}\in\Re} \mathcal{P}(y_{n+1}|x_{n+1},\mathcal{D}_n)E_{\mathcal{F}}[\delta(\hat{\vec{a}},\vec{a})|\mathcal{D}_n\cup(x_{n+1},y_{n+1})]\,dy_{n+1}. \qquad \text{(A.12)}$$

where:

$$\mathcal{P}(y_{n+1}|x_{n+1},\mathcal{D}_n) \propto \int_{\vec{a}\in\Re^{K+1}} \mathcal{P}(\mathcal{D}_n\cup(x_{n+1},y_{n+1})|\vec{a})\mathcal{P}_{\mathcal{F}}(\vec{a})\,d\vec{a}.$$

It is clear from Equation A.12 that $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, x_{n+1})$ is merely the *expected* EISD value between g and its *new* estimate, weighted and averaged over all possible values of y_{n+1} at x_{n+1}. Recall from Equation A.5 however, that for this polynomial function class, the EISD between g and its estimate $\hat{g}$ depends only on the input x_i values in $\mathcal{D}_n$ and not on the observed y_i values. This means that $E_{\mathcal{F}}[\delta(\hat{\vec{a}},\vec{a})|\mathcal{D}_n\cup(x_{n+1},y_{n+1})]$, the new EISD resulting from sampling next at x_{n+1}, does not depend on y_{n+1}! Equation A.12 can therefore be further simplified, which leads to the following closed form expression for the *total output uncertainty* cost function, given also in Equation 4.11:

$$\begin{aligned}
\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, x_{n+1}) &= E_{\mathcal{F}}[\delta(\hat{\vec{a}}, \vec{a})|\mathcal{D}_n \cup (x_{n+1}, y_{n+1})] \int_{y_{n+1}\in\Re} \mathcal{P}(y_{n+1}|x_{n+1}, \mathcal{D}_n)\, dy_{n+1} \\
&= E_{\mathcal{F}}[\delta(\hat{\vec{a}}, \vec{a})|\mathcal{D}_n \cup (x_{n+1}, y_{n+1})] \\
&= |\Sigma_{n+1}\mathbf{A}| \quad \propto \quad |\Sigma_{n+1}| \qquad \text{(A.13)}
\end{aligned}$$

Here, Σ_{n+1} has exactly the same form as Σ_n in Equation A.5, and depends only on the polynomial function class priors $\Sigma_{\mathcal{F}}$, the output noise variance σ_s^2 and the data input locations $\{x_1, x_2, \ldots, x_n, x_{n+1}\}$.

A.1.4 The Polynomial Smoothness Prior

For our polynomial function class, we have assumed the following multi-dimensional Gaussian prior distribution on model parameters $\vec{a} = [a_0\ a_1\ \ldots\ a_K]^{\mathrm{T}}$:

$$\mathcal{P}_{\mathcal{F}}(\vec{a}) = \frac{1}{(2\pi)^{(K+1)/2}|\Sigma_{\mathcal{F}}|^{1/2}} \exp(-\frac{1}{2}\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}),$$

where $\Sigma_{\mathcal{F}}$ is a $(K+1)\times(K+1)$ covariance matrix. If one assumes an *independent* Gaussian distribution with variance σ_i^2 on each parameter a_i, then $\Sigma_{\mathcal{F}}$ is simply a diagonal matrix with the independent variance terms $\{\sigma_i^2|i = 0, \ldots, K\}$ on its principal diagonal.

Let $p \in \mathcal{F}$ be a polynomial function in the approximation concept class. We consider here a slightly different prior distribution on $\mathcal{F}$ based on a global "smoothness" measure:

$$\begin{aligned}
\mathcal{P}_{\mathcal{F}}(\vec{a}) &\propto \exp(-\mathcal{Q}(p(\cdot, \vec{a})))\exp(-\frac{a_0^2}{2\sigma_0^2}) \\
&= \exp\left[-\frac{1}{2}\left(2\mathcal{Q}(p(\cdot, \vec{a})) + \frac{a_0^2}{\sigma_0^2}\right)\right] \qquad \text{(A.14)}
\end{aligned}$$

where the "smoothness" term is:

$$\mathcal{Q}(p(\cdot, \vec{a})) = \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}} \left[\frac{dp(x, \vec{a})}{dx}\right]^2 dx.$$

We shall show that despite the apparent difference in general form, $\mathcal{P}_{\mathcal{F}}(\vec{a})$ still simplifies to a multi-dimensional Gaussian distribution, whose new covariance term $\Sigma_{\mathcal{F}}$ is given by Equation 4.14. First we express $\mathcal{Q}(p(\cdot, \vec{a}))$ in terms of the polynomial model parameters $\vec{a}$:

$$
\begin{aligned}
p(x) &= \sum_{t=0}^{K} a_t x^t \\
\frac{d\,p(x)}{dx} &= \sum_{t=1}^{K} a_t t x^{t-1} \\
\left(\frac{d\,p(x)}{dx}\right)^2 &= \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_K \end{bmatrix}^{\mathtt{T}} \begin{bmatrix} 1 & 2x & 3x^2 & \cdots & Kx^{K-1} \\ 2x & 4x^2 & 6x^3 & & 2Kx^K \\ 3x^2 & 6x^3 & 9x^4 & & \vdots \\ \vdots & & & \ddots & \\ Kx^{K-1} & 2Kx^K & \cdots & & K^2x^{2K-2} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_K \end{bmatrix} \\
\mathcal{Q}(p(\cdot, \vec{a})) &= \int_{x_{\mathtt{LO}}}^{x_{\mathtt{HI}}} \left(\frac{d\,p(x)}{dx}\right)^2 dx \;=\; [a_1\; a_2\; \ldots\; a_K]\, \mathbf{N}\, [a_1\; a_2\; \ldots\; a_K]^{\mathtt{T}}
\end{aligned}
$$

$\mathbf{N}$ is a constant $K \times K$ matrix of numbers whose $(i,j)^{\mathtt{th}}$ element is:

$$\mathbf{N}(i,j) = \frac{ij}{i+j-1}\left(x_{\mathtt{HI}}^{i+j-1} - x_{\mathtt{LO}}^{i+j-1}\right)$$

Next, we substitute the above result into the exponent of Equation A.14. We get:

$$
\begin{aligned}
2\mathcal{Q}(p(\cdot, \vec{a})) + \frac{a_0^2}{\sigma_0^2} &= [a_0\; a_1\; \ldots\; a_K] \begin{bmatrix} 1/\sigma_0^2 & \vec{0}^{\mathtt{T}} \\ \vec{0} & 2\mathbf{N} \end{bmatrix} [a_0\; a_1\; \ldots\; a_K]^{\mathtt{T}} && \text{(A.15)} \\
&= \vec{a}^{\mathtt{T}} \Sigma_{\mathcal{F}}^{-1} \vec{a} \\
\mathcal{P}_{\mathcal{F}}(\vec{a}) &\propto \exp\left[-\frac{1}{2}\left(2\mathcal{Q}(p(\cdot, \vec{a})) + \frac{a_0^2}{\sigma_0^2}\right)\right] \\
&= \exp\left[-\frac{1}{2}\vec{a}^{\mathtt{T}} \Sigma_{\mathcal{F}}^{-1} \vec{a}\right] && \text{(A.16)}
\end{aligned}
$$

where the $K \times K$ matrix $\mathbf{N}$ is as defined above. Thus, from Equation A.16, we see that the new prior distribution $\mathcal{P}_{\mathcal{F}}(\vec{a})$ is indeed multi-dimensional Gaussian in form with covariance $\Sigma_{\mathcal{F}}$ as given below and in Equation 4.14:

$$\Sigma_{\mathcal{F}}^{-1}(i,j) = \begin{cases} 1/\sigma_0^2 & \text{if } i=j=1 \\ 2\frac{(i-1)(j-1)}{i+j-3}(x_{\texttt{HI}}^{i+j-3} - x_{\texttt{LO}}^{i+j-3}) & \text{if } 2 \le i \le K+1 \text{ and } 2 \le j \le K+1 \\ 0 & \text{otherwise} \end{cases} \tag{A.17}$$

A.2 Gaussian Radial Basis Functions

Our next example is a d-dimensional Gaussian Radial Basis Function class $\mathcal{F}$ with K fixed centers. The analysis here is very similar to the polynomial case presented in the previous section. Let $\mathcal{G}_i$ be the i^{th} basis function with a fixed center $\vec{c_i}$ and a fixed covariance $\mathcal{S}_i$. The model parameters to be learnt are the RBF weight coefficients denoted by $\vec{a} = [a_1\ a_2\ \ldots\ a_K]^{\text{T}}$. An arbitrary function $r \in \mathcal{F}$ in this class can be represented as:

$$\begin{aligned} r(\vec{x}, \vec{a}) &= \sum_{i=1}^{K} a_i \mathcal{G}_i(\vec{x}) \\ &= \sum_{i=1}^{K} a_i \frac{1}{(2\pi)^{d/2}|\mathcal{S}_i|^{1/2}} \exp(-\frac{1}{2}(\vec{x} - \vec{c_i})^{\text{T}} \mathcal{S}_i^{-1} (\vec{x} - \vec{c_i})) \end{aligned}$$

The prior distribution on $\mathcal{F}$ is a zero-mean Gaussian distribution with covariance $\Sigma_{\mathcal{F}}$ on the model parameters:

$$\mathcal{P}_{\mathcal{F}}(r(\cdot, \vec{a})) = \mathcal{P}_{\mathcal{F}}(\vec{a}) = \frac{1}{(2\pi)^{K/2}|\Sigma_{\mathcal{F}}|^{1/2}} \exp(-\frac{1}{2}\vec{a}^{\text{T}} \Sigma_{\mathcal{F}}^{-1} \vec{a}). \tag{A.18}$$

Lastly, the learner has access to noisy data of the form $\mathcal{D}_n = \{(\vec{x_i}, y_i = g(\vec{x_i}) + \eta) : i = 1, \ldots, n\}$, where $g \in \mathcal{F}$ is an unknown target function and η is a zero-mean additive Gaussian noise term with variance σ_s^2. The learning task is to recover g from $\mathcal{D}_n$.

A.2.1 The A-Posteriori Function Class Distribution

We first derive the a-posteriori distribution on function class $\mathcal{F}$ given data $\mathcal{D}_n$, i.e., $\mathcal{P}(\vec{a}|\mathcal{D}_n) \propto \mathcal{P}_{\mathcal{F}}(\vec{a})\mathcal{P}(\mathcal{D}_n|\vec{a})$. Since $\mathcal{D}_n$ is sampled under additive zero-mean Gaussian noise with variance σ_s^2, we have:

$$\begin{aligned}
\mathcal{P}(\mathcal{D}_n|\vec{a}) &\propto \exp\left(-\frac{1}{2\sigma_s^2}\sum_{j=1}^{n}(y_j - r(\vec{x_j},\vec{a}))^2\right) \\
&= \exp\left(-\frac{1}{2\sigma_s^2}\sum_{j=1}^{n}(y_j - \sum_{t=1}^{K} a_t \frac{\exp\left(-\frac{1}{2}(\vec{x_j}-\vec{c_t})^{\mathrm{T}}\mathcal{S}_t^{-1}(\vec{x_j}-\vec{c_t})\right)}{(2\pi)^{d/2}|\mathcal{S}_t|^{1/2}})^2\right) \\
&= \exp\left(-\frac{1}{2\sigma_s^2}\sum_{j=1}^{n}(y_j - \sum_{t=1}^{K} a_t\mathcal{G}_t(\vec{x_j}))^2\right) \qquad \text{(A.19)}
\end{aligned}$$

Let $\bar{\mathbf{z}}_\mathbf{j} = [\mathcal{G}_1(\vec{x_j})\ \mathcal{G}_2(\vec{x_j})\ \ldots\ \mathcal{G}_K(\vec{x_j})]^{\mathrm{T}}$ be a vector of kernel output values for the $j^{\mathtt{th}}$ input value. One can expand the exponent term in Equation A.19 as follows:

$$\begin{aligned}
(y_j - \sum_{t=1}^{K} a_t\mathcal{G}_t(\vec{x_j}))^2 &= y_j^2 + \left(\sum_{t=1}^{K} a_t\mathcal{G}_t(\vec{x_j})\right)^2 - 2y_j\sum_{t=1}^{K} a_t\mathcal{G}_t(\vec{x_j}) \\
&= y_j^2 + \vec{a}^{\mathrm{T}}(\bar{\mathbf{z}}_\mathbf{j}\bar{\mathbf{z}}_\mathbf{j}^{\mathrm{T}})\vec{a} - y_j\bar{\mathbf{z}}_\mathbf{j}^{\mathrm{T}}\vec{a} - \vec{a}^{\mathrm{T}}\bar{\mathbf{z}}_\mathbf{j}y_j
\end{aligned}$$

So:

$$\begin{aligned}
\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\left(y_j - \sum_{t=1}^{K} a_t\mathcal{G}_t(\vec{x_j})\right)^2 &= \frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2 + \vec{a}^{\mathrm{T}}\left(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{z}}_\mathbf{j}\bar{\mathbf{z}}_\mathbf{j}^{\mathrm{T}})\right)\vec{a} \\
&\quad - (\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j\bar{\mathbf{z}}_\mathbf{j}^{\mathrm{T}})\vec{a} - \vec{a}^{\mathrm{T}}(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{z}}_\mathbf{j}y_j)
\end{aligned}$$

The Gaussian RBF prior distribution $\mathcal{P}_{\mathcal{F}}(\vec{a})$ is as given in Equation A.18. The a-posteriori distribution is thus:

$$\begin{aligned}
\mathcal{P}(\vec{a}|\mathcal{D}_n) &\propto \mathcal{P}_{\mathcal{F}}(\vec{a})\mathcal{P}(\mathcal{D}_n|\vec{a}) \\
&\propto \exp\left(-\frac{1}{2}\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a}\right)\exp\left(-\frac{1}{2\sigma_s^2}\sum_{j=1}^{n}(y_j - \sum_{t=1}^{K} a_t\mathcal{G}_t(\vec{x_j}))^2\right) \\
&= \exp\Bigg[-\frac{1}{2}\Bigg(\vec{a}^{\mathrm{T}}\Sigma_{\mathcal{F}}^{-1}\vec{a} + \frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2 + \vec{a}^{\mathrm{T}}\left(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{z}}_\mathbf{j}\bar{\mathbf{z}}_\mathbf{j}^{\mathrm{T}})\right)\vec{a} \\
&\qquad - (\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j\bar{\mathbf{z}}_\mathbf{j}^{\mathrm{T}})\vec{a} - \vec{a}^{\mathrm{T}}(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{z}}_\mathbf{j}y_j)\Bigg)\Bigg]
\end{aligned}$$

$$= \exp\left[-\frac{1}{2}\left(\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2 + \vec{a}^{\mathrm{T}}\left(\Sigma_{\mathcal{F}}^{-1} + \frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{z}}_{\mathbf{j}}\bar{\mathbf{z}}_{\mathbf{j}}^{\mathrm{T}})\right)\vec{a} - (\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j\bar{\mathbf{z}}_{\mathbf{j}}^{\mathrm{T}})\vec{a} - \vec{a}^{\mathrm{T}}(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{z}}_{\mathbf{j}}y_j)\right)\right] \quad (A.20)$$

Completing the square in Equation A.20 yields:

$$\mathcal{P}(\vec{a}|\mathcal{D}_n) \propto \exp\left[-\frac{1}{2}\left((\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\Sigma_n^{-1}(\vec{a}-\hat{\vec{a}}) - \hat{\vec{a}}^{\mathrm{T}}\Sigma_n^{-1}\hat{\vec{a}} + \frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2\right)\right] \quad (A.21)$$

where:

$$\Sigma_n^{-1} = \Sigma_{\mathcal{F}}^{-1} + \frac{1}{\sigma_s^2}\sum_{j=1}^{n}(\bar{\mathbf{z}}_{\mathbf{j}}\bar{\mathbf{z}}_{\mathbf{j}}^{\mathrm{T}}) \quad (A.22)$$

$$\hat{\vec{a}} = \Sigma_n\,(\frac{1}{\sigma_s^2}\sum_{j=1}^{n}\bar{\mathbf{z}}_{\mathbf{j}}y_j) \quad (A.23)$$

Notice that as in the polynomial case (see Appendix A.1.1), neither of the two terms $\hat{\vec{a}}^{\mathrm{T}}\Sigma_n^{-1}\hat{\vec{a}}$ and $\frac{1}{\sigma_s^2}\sum_{j=1}^{n} y_j^2$ in Equation A.21 depend on the RBF model parameters $\vec{a}$. This means we can simply remove the two "constant" terms from the exponent and introduce into Equation A.21 the appropriate normalizing constants so $\mathcal{P}(\vec{a}|\mathcal{D}_n)$ becomes a standard probability distribution on $\vec{a}$:

$$\mathcal{P}(\vec{a}|\mathcal{D}_n) = \frac{1}{(2\pi)^{K/2}|\Sigma_n|^{1/2}}\exp\left[-\frac{1}{2}\left((\vec{a}-\hat{\vec{a}})^{\mathrm{T}}\Sigma_n^{-1}(\vec{a}-\hat{\vec{a}})\right)\right] \quad (A.24)$$

Thus, the RBF a-posteriori distribution is a multivariate Gaussian centered at $\hat{\vec{a}}$ (Equation A.23) with covariance Σ_n (Equation A.22).

A.2.2 The RBF EISD Measure

We now derive an expression for the *expected integrated squared difference* (EISD) between an unknown RBF target g and its *current* estimate given $\mathcal{D}_n$. We shall use this result later to derive $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$, the *total output uncertainty* cost function for RBF approximators.

The *expected integrated squared difference* (EISD) between an unknown target g and its estimate $\hat{g}$ given $\mathcal{D}_n$ is:

$$E_{\mathcal{F}}[\delta(\hat{g},g)|\mathcal{D}_n] = \int_{g\in\mathcal{F}}\mathcal{P}(g|\mathcal{D}_n)\delta(\hat{g},g)dg$$

where $\delta(\hat{g}, g)$ is a standard integrated squared difference measure between two functions over the input space $\Re^d$ or some appropriate region of interest:

$$\delta(\hat{g}, g) = \int_{\vec{x}\in\Re^d} (g(\vec{x}) - \hat{g}(\vec{x}))^2 d\vec{x}.$$

For our RBF approximation function class, the *optimal* estimate for g given $\mathcal{D}_n$ has model parameters $\hat{\vec{a}}$ (Equation A.23), since this is where the a-posteriori distribution $\mathcal{P}(\vec{a}|\mathcal{D}_n)$ has a global maximum. Let $\hat{\vec{a}} = [\hat{a_0}\ \hat{a_1}\ \ldots\ \hat{a_K}]^{\mathrm{T}}$ and $\bar{\mathbf{z}} = [\mathcal{G}_1(\vec{x})\ \mathcal{G}_2(\vec{x})\ \ldots\ \mathcal{G}_K(\vec{x})]^{\mathrm{T}}$, one can rewrite $\delta(\hat{g}, g)$ in terms of RBF model parameters as:

$$\begin{aligned}
\delta(\hat{\vec{a}}, \vec{a}) &= \int_{\vec{x}\in\Re^d} [r(x,\vec{a}) - r(x,\hat{\vec{a}})]^2\, d\vec{x} = \int_{\vec{x}\in\Re^d} \left[(\sum_{i=1}^{K} a_i \mathcal{G}_i(\vec{x})) - (\sum_{i=1}^{K} \hat{a}_i \mathcal{G}_i(\vec{x})) \right]^2 d\vec{x} \\
&= \int_{\vec{x}\in\Re^d} \left[\sum_{i=1}^{K} (a_i - \hat{a}_i)\mathcal{G}_i(\vec{x}) \right]^2 d\vec{x} = \int_{\vec{x}\in\Re^d} (\vec{a} - \hat{\vec{a}})^{\mathrm{T}} \bar{\mathbf{z}}\bar{\mathbf{z}}^{\mathrm{T}} (\vec{a} - \hat{\vec{a}})\, d\vec{x} \\
&= (\vec{a} - \hat{\vec{a}})^{\mathrm{T}} \left(\int_{\vec{x}\in\Re^d} \bar{\mathbf{z}}\bar{\mathbf{z}}^{\mathrm{T}}\, d\vec{x} \right) (\vec{a} - \hat{\vec{a}}) \\
&= (\vec{a} - \hat{\vec{a}})^{\mathrm{T}} \mathbf{A} (\vec{a} - \hat{\vec{a}}) \qquad \text{(A.25)}
\end{aligned}$$

where $\mathbf{A}$ is a constant $K \times K$ matrix of numbers whose $(i,j)^{\mathrm{th}}$ element is:

$$\mathbf{A}(i,j) = \int_{\vec{x}\in\Re^d} \mathcal{G}_i(\vec{x})\mathcal{G}_j(\vec{x})\, d\vec{x}$$

Substituting Equations A.24 and A.25 into the EISD expression, we get:

$$\begin{aligned}
E_{\mathcal{F}}[\delta(\hat{g}, g)|\mathcal{D}_n] &= E_{\mathcal{F}}[\delta(\hat{\vec{a}}, \vec{a})|\mathcal{D}_n] = \int_{\vec{a}\in\Re^K} \mathcal{P}(\vec{a}|\mathcal{D}_n)\delta(\hat{\vec{a}}, \vec{a})\, d\vec{a} \\
&= \int_{\vec{a}\in\Re^K} \frac{1}{(2\pi)^{K/2}|\Sigma_n|^{1/2}} \exp\left[-\frac{1}{2} \left((\vec{a} - \hat{\vec{a}})^{\mathrm{T}} \Sigma_n^{-1} (\vec{a} - \hat{\vec{a}}) \right) \right] \\
&\qquad (\vec{a} - \hat{\vec{a}})^{\mathrm{T}} \mathbf{A} (\vec{a} - \hat{\vec{a}})\, d\vec{a} \qquad \text{(A.26)}
\end{aligned}$$

Making the following change of variables as in the polynomial case: $\vec{q} = \mathbf{A}^{\frac{1}{2}}(\vec{a} - \hat{\vec{a}})$, and noting that the integration bounds on $\vec{q}$ is still $\Re^K$, Equation A.26 becomes:

$$\begin{aligned}
E_{\mathcal{F}}[\delta(\hat{g}, g)|\mathcal{D}_n] &= E_{\mathcal{F}}[\delta(\hat{\vec{a}}, \vec{a})|\mathcal{D}_n] \\
&= \int_{\vec{q}\in\Re^K} \frac{1}{(2\pi)^{K/2}|\mathbf{A}|^{1/4}|\Sigma_n|^{1/2}|\mathbf{A}|^{1/4}}
\end{aligned}$$

$$
\begin{aligned}
& \qquad \exp\left[-\frac{1}{2}\left(\vec{q}^{\mathrm{T}}\mathbf{A}^{-\frac{1}{2}}\Sigma_n^{-1}\mathbf{A}^{-\frac{1}{2}}\vec{q}\right)\right]\vec{q}^{\mathrm{T}}\vec{q}\,d\vec{q} \\
= & \int_{\vec{q}\in\Re^K} \frac{1}{(2\pi)^{K/2}|\Sigma_n\mathbf{A}|^{1/2}} \\
& \qquad \exp\left[-\frac{1}{2}\left(\vec{q}^{\mathrm{T}}\mathbf{A}^{-\frac{1}{2}}\Sigma_n^{-1}\mathbf{A}^{-\frac{1}{2}}\vec{q}\right)\right]\vec{q}^{\mathrm{T}}\vec{q}\,d\vec{q} \\
= & |\Sigma_n\mathbf{A}| \quad \propto \quad |\Sigma_n| \qquad (A.27)
\end{aligned}
$$

since $\mathbf{A}$ is just a constant matrix of numbers.

Notice from Equation A.22 that Σ_n depends only on the RBF function class priors $\Sigma_{\mathcal{F}}$, the K fixed Gaussian RBF kernels $\{\mathcal{G}_i(\cdot)|i = 1, \ldots, K\}$, the output noise variance σ_s^2 and the previously sampled input locations $\{x_1, x_2, \ldots, x_n\}$. Like the polynomial case, it does not depend on the previous y data values actually observed. In other words, the previously observed y data values in $\mathcal{D}_n$ *do not* affect the EISD measure (Equation A.27) for this Gaussian RBF concept class!

A.2.3 The RBF Total Output Uncertainty Measure

The *total output uncertainty* cost function is simply the *expected* EISD between g and its *new* estimate $\hat{g_{n+1}}$, if the learner samples next at $\vec{x_{n+1}}$. The cost function is given by Equation 4.7. We rewrite the expression below in terms of our RBF model parameters:

$$
\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}}) = \int_{y_{n+1}\in\Re} \mathcal{P}(y_{n+1}|\vec{x_{n+1}}, \mathcal{D}_n)E_{\mathcal{F}}[\delta(\hat{\vec{a}}, \vec{a})|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})]\, dy_{n+1}. \quad (A.28)
$$

where:

$$
\mathcal{P}(y_{n+1}|\vec{x_{n+1}}, \mathcal{D}_n) \propto \int_{\vec{a}\in\Re^K} \mathcal{P}(\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})|\vec{a})\mathcal{P}_{\mathcal{F}}(\vec{a})\, d\vec{a}.
$$

It is clear from Equation A.28 that $\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}})$ is merely the *new* EISD weighted and averaged over all possible values of y_{n+1} at $\vec{x_{n+1}}$. Recall from Equation A.22 however, that for this RBF concept class, the EISD between g and its estimate $\hat{g}$ depends only on the input $\vec{x_i}$ values in $\mathcal{D}_n$ and not on the observed y_i values. This means that $E_{\mathcal{F}}[\delta(\hat{\vec{a}}, \vec{a})|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})]$, the new EISD resulting from sampling next at $\vec{x_{n+1}}$, does not depend on y_{n+1}! Equation A.28 can therefore be further simplified, which leads to the following closed form expression for the *total output uncertainty* cost function, given also in Equation 4.17:

$$
\begin{aligned}
\mathcal{U}(\hat{g}_{n+1}|\mathcal{D}_n, \vec{x_{n+1}}) &= E_{\mathcal{F}}[\delta(\hat{\vec{a}}, \vec{a})|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})] \int_{y_{n+1} \in \Re} \mathcal{P}(y_{n+1}|\vec{x_{n+1}}, \mathcal{D}_n)\, dy_{n+1} \\
&= E_{\mathcal{F}}[\delta(\hat{\vec{a}}, \vec{a})|\mathcal{D}_n \cup (\vec{x_{n+1}}, y_{n+1})] \\
&= |\Sigma_{n+1}\mathbf{A}| \quad \propto \quad |\Sigma_{n+1}| \qquad \text{(A.29)}
\end{aligned}
$$

Here, Σ_{n+1} has exactly the same form as Σ_n in Equation A.22, and depends only on the polynomial function class priors $\Sigma_{\mathcal{F}}$, the K fixed Gaussian RBF kernels $\{\mathcal{G}_i(\cdot)|i = 1, \ldots, K\}$, the output noise variance σ_s^2 and the data input locations
$\{\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}, \vec{x_{n+1}}\}$.

Bibliography

[1] Y. Abu-Mostafa. The Vapnik-Chervonenkis Dimension: Information versus Complexity in Learning. *Neural Computation*, 1(3):312–317, 1989.

[2] Y. Abu-Mostafa. A Method for Learning from Hints. In S. J. Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural information processings systems 5*, pages 73–80, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[3] Y.S. Abu-Mostafa. Hints and the VC-dimension. *Neural Computation*, 5:278–288, 1993.

[4] S. Ahmad and S. Omohundro. A Network for Extracting the Locations of Point Clusters using Selective Attention. Technical Report TR 90-011, International Computer Science Institute, University of California, Berkeley, 1990.

[5] D. Angluin. Learning k-term DNF Formulas using Queries and Counterexamples. Technical Report YALU/DCS/RR-559, Yale University, Department of Computer Science, 1987.

[6] D. Angluin. Queries and Concept Learning. *Machine Learning*, 2(4):319–342, April 1988.

[7] Alan Bennett and Ian Craw. Finding Image Features Using Deformable Templates and Detailed Prior Statistical Knowledge. In *Proc. British Machine Vision Conference*, pages 233–239, 1991.

[8] M. Bertero. Regularization Methods for Linear Inverse Problems. In C. Talenti, editor, *Inverse Problems*. Springer-Verlag, Berlin, 1986.

[9] M. Betke and N. Makris. Fast Object Recognition in Noisy Images using Simulated Annealing. In *Proceedings of the International Conference on Computer Vision*, pages 523–530, Cambridge, MA, June 1995.

[10] D. Beymer, A. Shashua, and T. Poggio. Example Based Image Analysis and Synthesis. A.I. Memo No. 1431, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1993.

[11] David Beymer and Tomaso Poggio. Face Recognition from One Example View. In *Proceedings of the International Conference on Computer Vision*, Cambridge, MA, 1995.

[12] David J. Beymer. Face Recognition under Varying Pose. A.I. Memo No. 1461, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1993.

[13] David J. Beymer. Vectorizing Face Images by Interleaving Shape and Texture Computations. A.I. Memo No. 1537, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1995.

[14] B. Boser, I. Guyon, and V. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, Pittsburg, PA, 1992. ACM.

[15] Thomas M. Breuel. An Efficient Correspondence based Algorithm for 2D and 3D Model based Recognition. A.I. Memo No. 1259, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1993.

[16] J. Bromley and E. Sackinger. Neural-network and k-Nearest-Neighbor Classifiers. Technical Report 11359-910819-16TM, AT&T, 1991.

[17] R. Brooks. Symbolic Reasoning among 3-D Models and 2-D Images. *Artificial Intelligence*, 17, August 1981.

[18] R. Brooks, G. Russell, and T. Binford. The Acronym Model based Vision System. In *Proceedings IJCAI*, pages 105–113, 1979.

[19] R. Brunelli and T. Poggio. Face Recognition: Features versus Templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(10):1042–1052, 1993.

[20] B. Buchanan and E. Shortliffe. *Rule-Based Expert Programs: the MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, Reading, MA, 1984.

[21] W. Buntine. A Guide to the Literature on Learning Graphical Models. Technical Report IC-95-05, NASA Ames Research Center, 1995.

[22] M. C. Burl, U. Fayyad, P. Perona, P. Smyth, and M. P. Burl. A Trainable Tool for Finding Small Volcanoes in SAR Imagery of Venus. Technical Report CNS TR 34, California Institute of Technology, October 1993.

[23] I. Chakravarty and H. Freeman. Characteristic Views as a Basis for Three-Dimensional Object Recognition. In *SPIE Vol. 336, Robot Vision*, pages 37–45, 1982.

[24] D. Cohn. A Local Approach to Optimal Queries. In D. Touretzky, editor, *Proc. of 1990 Connectionist Summer School*, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

[25] T.F. Cootes, C.J. Taylor, A. Lanitis, D.H. Cooper, and J. Graham. Building and Using Flexible Models Incorporating Grey-level Information. In *Proceedings of the International Conference on Computer Vision*, pages 242–246, Berlin, May 1993.

[26] C. Cortes and V. Vapnik. Support Vector Networks. *Machine Learning (To Appear)*, 1995.

[27] Ian Craw, David Tock, and Alan Bennett. Finding Face Features. In *Proceedings Computer Vision - ECCV*, pages 92–96, 1992.

[28] Y. Le Cun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to Handwritten Zip Code Recognition. *Neural Computation*, 1:541–551, 1989.

[29] Y. Le Cun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten Digit Recognition with a Back-propagation Network. In *Advances in Neural Information Processing Systems*, volume 2, pages 396–404, 1990.

[30] H. Drucker, R. Schapire, and P. Simard. Boosting Performance in Neural Networks. *Interrnational Journal of Pattern Recognition and Artificial Intelligence*, 7(4):705–719, 1993.

[31] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*, chapter 6, pages 211–257. John Wiley and Sons Inc., 1973.

[32] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons Inc., New York, 1973.

[33] V. Fedorov. *Theory of Optimal Experiments*, page 35. Academic Press, New York, 1972.

[34] David Forsyth, Joseph L. Mundy, Andrew Zisserman, Chris Coelho, Aaron Heller, and Charles Rothwell. Invariant Descriptors for 3-D Object Recognition and Pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):971–991, 1991.

[35] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, 1990.

[36] Stuart Geman and Don Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[37] Ziv Gigus and Jitendra Malik. Computing the Aspect Graph for Line Drawings of Polyhedral Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):113–122, February 1990.

[38] W. E. L. Grimson and T. Lozano-Perez. Model-Based Recognition and Localization from Sparse Range Data. In A. Rosenfeld, editor, *Techniques for 3-D Machine Perception.* North-Holland, Amsterdam, 1985.

[39] W. Eric L. Grimson and Tomas Lozano-Perez. Model-Based Recognition and Localization from Sparse Range or Tactile Data. Technical Report A.I. Memo 738, MIT, August 1983.

[40] W. Eric L. Grimson and Tomas Lozano-Perez. Recognition and Localization of Overlapping Parts from Sparse Data in Two and Three Dimensions. 1984.

[41] W. Eric L. Grimson and Tomas Lozano-Perez. Recognition and Localization of Overlapping Parts from Sparse Data. In T. Kanade, editor, *Three-Dimensional Vision Systems.* Kluwer Academic Publishers, Amsterdam, 1985.

[42] W. Eric L. Grimson and Tomas Lozano-Pérez. Localizing Overlapping Parts by Searching the Interpretation Tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):467–482, 1987.

[43] J. Hadamard. *La Theorie des Equations aux Derivees Partielles.* Editions Scientifique, Pekin, 1964.

[44] D. Heckerman. A Tutorial on Learning Bayesian Networks. Technical Report MSR-TR-95-06, Microsoft Research, Advanced Technology Division, Redmond, WA, 1995.

[45] G. Hinton, M. Revow, and P. Dayan. Recognizing Handwritten Digits using Mixture of Linear Models. In D. Touretzky G. Tesauro and J. Alspector, editors, *Advances in Neural Information Processings Systems 7*, San Mateo, CA, 1995. Morgan Kaufman.

[46] R. Howard and J. Matheson. Influence Diagrams. In R. Howard and J. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, volume II, pages 721–762. Strategic Decisions Group, Menlo Park, CA, 1981.

[47] J. Hwang, J. Choi, S. Oh, and R. Marks. Query Learning based on Boundary Search and Gradient Computation of Trained Multi-layer Perceptrons. In *Proceedings IJCNN*, San Diego, CA, 1990. IEEE Press.

[48] Katsushi Ikeuchi and Takeo Kanade. Applying Sensor Models to Automatic Generation of Object Recognition Programs. In *Proceedings of the International Conference on Computer Vision*, pages 228–237, Tampa, FL, December 1988.

[49] D. Jacobs. The Use of Grouping in Visual Object Recognition. Master's thesis, Massachusetts Institute of Technology, 1988.

[50] M. Kearns and L. Valiant. Cryptographic Limitations on Learning Boolean Formulae and Finite Automata. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 443–444, May 1989.

[51] M. Kirby and L. Sirovich. Applications of the Karhunen-Loeve Procedure for the Characterization of Human Faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1):103–108, 1990.

[52] J. J. Koenderink and A. J. van Doorn. The Internal Representation of Solid Shape with Respect to Vision. *Biological Cybernetics*, 32:211–216, 1979.

[53] Matthew R. Korn and Charles R. Dyer. 3-D Multiview Object Representations for Model-based Object Recognition. *Pattern Recognition*, 20(1):91–103, 1987.

[54] Y. Lee. Handwritten Digit Recognition using K Nearest-Neighbor, Radial-Basis Function and Backpropagation Neural Networks. *Neural Computation*, 3:440–449, 1991.

[55] T. Leung, M. Burl, and P. Perona. Finding Faces in Cluttered Scenes using Random Labeled Graph Matching. In *Proceedings of the International Conference on Computer Vision*, pages 637–644, Cambridge, MA, June 1995.

[56] R. Lindsay, B. Buchanan, E. Feigenbaum, and J. Lederberg. *Applications of Artificial Intelligence for Chemical Inference: The DENDRAL Project.* McGraw-Hill, New York, 1980.

[57] David G. Lowe. *Perceptual Organization and Visual Recognition.* Kluwer Academic Publishers, Boston, 1985.

[58] D. MacKay. *Bayesian Methods for Adaptive Models.* PhD thesis, California Institute of Technology, Pasadena, CA, 1992.

[59] J. Marroquin, S. Mitter, and Tomaso Poggio. Probabilistic Solution of Ill-posed Problems in Computational Vision. In *Proceedings Image Understanding Workshop*, pages 293–309, Miami Beach, FL, December 1985.

[60] G. Martin and J. Pittman. Recognizing Hand-printed Letters and Digits using Backpropagation Learning. *Neural Computation*, 3:258–267, 1991.

[61] B. Moghaddam and A. Pentland. Probabilistic Visual Learning for Object Detection. In *Proceedings of the International Conference on Computer Vision*, pages 786–793, Cambridge, MA, June 1995.

[62] J. Moody and C. Darken. Fast Learning in Networks of Locally Tuned Processing Units. *Neural Computation*, 1(2):281–294, 1989.

[63] V. Morozov. *Methods of Solving Incorrectly posed Problems.* Springer-Verlag, Berlin, 1984.

[64] Hiroshi Murase and Shree K. Nayar. Learning Object Models from Appearance. In *Proceedings AAAI*, pages 836–843, Washington, DC, 1993.

[65] P. Niyogi. *The Informational Complexity of Learning from Examples.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1995.

[66] P. Niyogi and F. Girosi. On the Relationship between Generalization Error, Hypothesis Complexity, and Sample Complexity for Radial Basis Functions. Technical Report AIM–1467, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1993.

[67] D. Parker. Learning Logic. Technical Report TR-47, Center of Computational Research in Economics and Management Science, MIT, 1985.

[68] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan-Kaufmann, San Mateo, CA, 1988.

[69] A. Pentland, B. Moghaddam, and T. Starner. View-based and Modular Eigenspaces for Face Recognition. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 84–91, June 1994.

[70] M. Plutowski and H. White. Active Selection of Training Examples for Network Learning in Noiseless Environments. Technical Report CS91-180, Department of Computer Science and Engineering, University of California, San Diego, 1991.

[71] T. Poggio and F. Girosi. A Theory of Networks for Approximation and Learning. Technical Report AIM–1140, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1989.

[72] T. Poggio and F. Girosi. Extensions of a Theory of Networks for Approximation and Learning: Outliers and Negative Examples. Technical Report AIM–1220, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1990.

[73] T. Poggio and T. Vetter. Recognition and Structure from One (2D) Model View: Observations on Prototypes, Object Classes, and Symmetries. A.I. Memo No. 1347, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1992.

[74] Jean Ponce and David J. Kriegman. Computing Exact Aspect Graphs of Curved Objects: Parametric Surfaces. In *Proceedings AAAI*, pages 1074–1079, 1990.

[75] D. Reisfeld, H. Wolfson, and Y. Yeshurun. Detection of Interest Points using Symmetry. In *Proceedings of the International Conference on Computer Vision*, pages 62–65, Dec 1990.

[76] H. Rowley, S. Baluja, and T. Kanade. Human Face Detection in Visual Scenes. Technical Report CMU-CS-95-158, Carnegie Mellon University, 1995.

[77] D. Rumelhart and J. McClelland. *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, Massachusetts, 1986.

[78] C. Sammut and R. Banerji. Learning Concepts by Asking Questions. In J. Carbonell R. Michalski and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach (Vol. 2)*. Morgan Kaufmann, Los Altos, CA, 1986.

[79] R. Schapire. The Strength of Weak Learnability. *Machine Learning*, 5(2):197–227, 1990.

[80] B. Schoelkopf, C. Burges, and V. Vapnik. Extracting Support Data for a Given Task. In *International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, 1995. AAAI Press.

[81] E. Shapiro. Algorithmic Program Disgnosis. In *Proceedings of Ninth ACM Symposium on Principles of Programming Languages*, pages 299–308, Albuquerque, NM, 1982. The Association of Computing Machinery.

[82] E. Shapiro. *Algorithmic Program Debugging.* MIT Press, Cambridge, MA, 1983.

[83] P. Simard, Y. Le Cun, and J. Denker. Efficient Pattern Recognition using a New Transformation Distance. In *Advances in Neural Information Processing Systems*, volume 5, pages 50–58, Denver, Colorado, 1993.

[84] P. Sinha. Object Recognition via Image Invariants: A Case Study. In *Investigative Ophthalmology and Visual Science*, volume 35, pages 1735–1740, Sarasota, Florida, May 1994.

[85] L. Sirovich and M. Kirby. Low-dimensional Procedure for the Characterization of Human Faces. *Journal of the Optical Society of America*, 4(3):519–524, March 1987.

[86] P. Sollich. Query Construction, Entropy, Generalization in Neural Network Models. *Physical Review E*, 49:4637–4651, 1994.

[87] John Stewman and Kevin Bowyer. Creating the Perspective Projection Aspect Graph of Polyhedral Objects. In *Proceedings of the International Conference on Computer Vision*, pages 494–500, Tampa, FL, December 1988.

[88] K. Sung and P. Niyogi. Active Learning for Function Approximation. In *Advances in Neural Information Processings Systems 7*, pages 593–600, Cambridge, MA, 1995. MIT Press.

[89] K. Sung and T. Poggio. Example-based Learning for View-based Human Face Detection. In *Proceedings Image Understanding Workshop*, volume II, pages 843–850, Monterey, CA, November 1994.

[90] K. Sung and T. Poggio. Example-based Learning for View-based Human Face Detection. Technical Report AIM–1521, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, December 1994.

[91] Michael J. Swain and Dana H. Ballard. Color Indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[92] Demetri Terzopoulos and Keith Waters. Analysis and Synthesis of Facial Image Sequences using Physical and Anatomical Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):569–579, 1993.

[93] C. Therrien. *Decision, Estimation and Classification.* John Wiley and Sons, Inc., 1989.

[94] A. Tikhonov. Solution of Incorrectly Formulated Problems and the Regularization Method. *Soviet Math. Dokl.*, 4:1035–1038, 1963.

[95] A. Tikhonov and V. Arsenin. *Solutions of Ill-Posed Problems.* W. H. Winston, Washington, DC, 1977.

[96] M. Turk and A. Pentland. Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[97] S. Ullman and R. Basri. Recognition by Linear Combinations of Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):992–1006, 1991.

[98] L. Valiant. A Theory of Learnable. *Proc. of the 1984 STOC*, pages 436–445, 1984.

[99] L. Valiant. Learning Disjunctions of Conjunctions. In *Proceedings IJCAI*, pages 560–566, Los Angeles, CA, 1985.

[100] V. Vapnik and A. Chervonenkis. On the Uniform Convergence of Relative Frequencies of Events to their Probabilities. *Th. Prob. and its Applications*, 17(2):264–280, 1971.

[101] V. Vapnik and A. Chervonenkis. The Necessary and Sufficient Conditions for the Uniform Convergence of Averages to their Expected Values. *Toeriya Veroyatnostei i Ee Primeneniya*, 26(3):543–564, 1981.

[102] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, Berlin, 1982.

[103] G. Yang and T. Huang. Human Face Detection in a Scene. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 453–458, June 1993.

[104] A. Yuille, P. Hallinan, and D. Cohen. Feature Extraction from Faces using Deformable Templates. *International Journal of Computer Vision*, 8(2):99–111, 1992.