Boston University

CS Department Technical Report-2008-018

INDEXING METHODS FOR EFFICIENT MULTICLASS RECOGNITION

ALEXANDRA STEFAN

Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Arts

BOSTON

UNIVERSITY

BOSTON UNIVERSITY

GRADUATE SCHOOL OF ARTS AND SCIENCES

Thesis

Boston University

CS Department Technical Report-2008-018

INDEXING METHODS FOR EFFICIENT MULTICLASS RECOGNITION

by

ALEXANDRA STEFAN

B.S., University of Bucharest, 2002

Submitted in partial fulfillment of the

requirements for the degree of

Master of Arts

2008

Approved by

First Reader

Stan Sclaroff, PhD
Professor of Computer Science

Second Reader

Margrit Betke, PhD
Associate Professor of Computer Science

Third Reader

George Kollios, PhD
Associate Professor of Computer Science

Acknowledgments

I want to thank my advisor, Professor Stan Sclaroff, for giving me this opportunity and helping me throughout the whole process. I also want to thank the other members of my committee, Professor Margrit Betke and Professor George Kollios, for providing useful feedback. Naturally, I also want to thank my fellow students in the IVC group, for their support and friendship. Special thanks go to Quan Yuan and Vassilis Athitsos, who collaborated closely with me on this project. Last, but not least I want to thank my mother, my father, and my sister. They have always believed in me, but most important of all, they accepted me for who I am and they were there for me in the roughest of times.

Boston University

CS Department Technical Report-2008-018

INDEXING METHODS FOR EFFICIENT MULTICLASS RECOGNITION

ALEXANDRA STEFAN

ABSTRACT

Many real world image analysis problems, such as face recognition and hand pose estimation, involve recognizing a large number of classes of objects or shapes. Large margin methods, such as AdaBoost and Support Vector Machines (SVMs), often provide competitive accuracy rates, but at the cost of evaluating a large number of binary classifiers, thus making it difficult to apply such methods when thousands or millions of classes need to be recognized. This thesis proposes a filter-and-refine framework, whereby, given a test pattern, a small number of candidate classes can be identified efficiently at the filter step, and computationally expensive large margin classifiers are used to evaluate these candidates at the refine step. Two different filtering methods are proposed, ClassMap and OVA-VS (One-vs.-All classification using Vector Search).

ClassMap is an embedding-based method, works for both boosted classifiers and SVMs, and tends to map the patterns and their associated classes close to each other in a vector space. OVA-VS maps OVA classifiers and test patterns to vectors based on the weights and outputs of weak classifiers of the boosting scheme. At runtime, finding the strongest-responding OVA classifier becomes a classical vector search problem, where well-known methods can be used to gain efficiency.

In our experiments, the proposed methods achieve significant speed-ups, in some cases up to two orders of magnitude, compared to exhaustive evaluation of all OVA classifiers. This was achieved in hand pose recognition and face recognition systems where the number of classes ranges from 535 to 48,600.

Contents

List of Figures

List of Abbreviations

1D	.....	One-dimensional
2D	.....	Two-dimensional
3D	.....	Three-dimensional
ACM	.....	Association for Computing Machinery
ASL	.....	American Sign Language
DAGSVM	.....	Directed Acyclic Graph SVM
ECOC	.....	Error-Correcting Output Codes
FRGC	.....	Face Recognition Grand Challenge
IEEE	.....	Institute of Electrical and Electronics Engineers
L_1	.....	Manhattan distance measure
L_2	.....	Euclidean distance measure
LSH	.....	Locality-Sensitive Hashing
OVA	.....	One-vs.-All
OVA-VS	.....	One-vs.-All classification using vector search methods
PCA	.....	Principal Component Analysis
RBF	.....	Radial Basis Function
SIAM	.....	Society for Industrial and Applied Mathematics
SVM	.....	Support Vector Machine

Chapter 1

Introduction

Many real-world computer vision tasks involve recognizing a very large number of classes of objects or shapes - a number that can range from thousands to millions. Examples of such tasks include biometrics-based identification (based on faces and/or fingerprints), hand and human body pose classification, speech and sign language recognition, and generic object recognition using computer vision. An important problem in such domains is designing recognition methods that are scalable and achieve efficient running time in the presence of a large number of classes.

Large margin methods are machine learning techniques that try to maximize the margin of training patterns, where the margin of a pattern, given a classifier, is a measure of confidence in the output of that classifier on the pattern. For example, support vector machines (SVMs) (Vapnik, 1995) are trained by attempting to maximize the minimum margin of any training example (Allwein et al., 2000). SVMs and boosting (Friedman et al., 2000; Schapire and Singer, 1999) have been successful in recent years in various pattern recognition domains. A common way to apply such methods to multiclass problems is to train, for each class, a one-versus-all (OVA) classifier to discriminate between samples from that class and samples from the other classes (Allwein et al., 2000; Torralba et al., 2007). Given a test pattern to recognize, all OVA classifiers are applied to that pattern, and the classification output is the class label associated with the OVA classifier that produced the strongest response.

A major bottleneck of these commonly used techniques is that, given a new pattern to classify, all binary classifiers must be applied to that pattern. This leads to time complexity that is typically at least linear in the number of classes. This time complexity can lead to

prohibitive running times in large multiclass domains with thousands or millions of classes. The main goal of this thesis is to address this problem and propose efficient and scalable methods for large margin multiclass recognition.

This thesis does not address the topic of feature selection and feature extraction. It is simply assumed that some appropriate choices regarding features have already been made. The goal in this thesis is to propose efficient alternatives to exhaustive evaluation of all binary classifiers in order to classify a test pattern. Training and test patterns, for the purposes of our discussion, are feature vectors on which the OVA classifiers can be directly applied.

1.1 Contributions

This thesis proposes a filter-and-refine framework, whereby, given a test pattern, a small number of candidate classes can be identified efficiently at the filter step, and computationally expensive large margin OVA classifiers are used to evaluate these candidates at the refine step. The main contribution consists of proposing two different filtering methods that can be employed in this filter-and-refine framework: ClassMap, and OVA-VS (One-vs.-All classification using Vector Search).

1.1.1 ClassMap

The ClassMap method is a novel paradigm for multiclass recognition, whereby efficient and accurate recognition is framed as a database search problem: given a pattern to classify, the goal is to quickly identify, in a database of classes, a small set of candidate classes for that pattern. The winning candidate can then be selected by applying the individual classifiers available for the candidate classes. Designing an efficient search mechanism for a database of classes is a non-trivial problem. First, classes are abstract entities with no explicit representation given a priori. Second, no distance measure is defined a priori for comparing database objects (classes) to each other and to queries. ClassMap constructs an explicit representation for classes, that can be used to:

1. "store" such classes in an actual database,

2. define a distance measure between classes and from patterns to classes, and

3. accommodate efficient search.

ClassMap is designed for use in domains where the task of multiclass recognition is decomposed into multiple OVA classification problems. ClassMap is not concerned with *how* the classifiers are trained, e.g., using AdaBoost or support vector machines. The only assumptions we make are that:

1. all OVA classifiers have already been trained, and

2. a set of labeled training patterns has been provided.

The key idea in ClassMap is that we can embed both patterns and classes into a common vector space, in a way that patterns tend to get mapped close to their correct classes. Finding the nearest classes of a pattern in this vector space can be done efficiently, using vector comparisons based on the L_1 and L_2 metrics. If we have n classes, instead of applying n OVA classifiers to the query, we perform n vector comparisons. Typically, performing such a vector comparison is significantly faster than evaluating one of the original binary classifiers. Furthermore, vector indexing methods , e.g., (Böhm et al., 2001; Gionis et al., 1999), can be applied to further reduce the number of required vector comparisons. This way, a small number of candidate classes can be quickly identified for each query. Then, only the binary classifiers associated with those classes need to be applied to the query.

We evaluate ClassMap on two datasets, examples in Fig. 1·1: a dataset of hand images where the task is to recognize the handshape and 3D orientation (out of 2430 classes), and a dataset of face images where the task is to recognize the individual (out of 535 classes). In both cases, one-vs.-all (OVA) classifiers are trained. Compared to the brute-force method, where all OVA classifiers are applied on each pattern, ClassMap is between 3 and 28 times faster, with negligible or no loss in classification accuracy.

1.1.2 One-vs.-All Classification Using Vector Search

The second method is called OVA-VS, which stands for "One-vs.-All classification using Vector Search." This method is specific to boosting-based multiclass recognition using OVA classifiers. Our main contribution regarding this method is showing that, given a pattern to classify, identifying the strongest-responding OVA classifier for that pattern is essentially a nearest neighbor search problem in a real vector space $\mathbb{R}^d$. This result leads to two additional contributions:

1. Using the theoretical properties of well-known nearest neighbor search methods, such as locality sensitive hashing (LSH) (Gionis et al., 1999), we show that OVA classification can be approximated with time complexity that is fundamentally sublinear in the number of classes, and the approximation can be as close as desired (accuracy can be traded for efficiency).

2. Using simple vector search methods that are based on dimensionality reduction, we obtain speedups of over two orders of magnitude in a large multiclass problem, where approximately 50,000 hand poses are recognized and speedups of 6 in a face recognition problem with 535 classes. Speedups are measured with respect to the time it takes to perform brute-force evaluation of all OVA classifiers on each test pattern.

1.2 Overview

The remainder of this thesis is organized as follows: Chapter 2 provides background information and the problem definition. Chapter 3 presents related work. Chapters 4 and 5 describe the two main contributions of this thesis, i.e., ClassMap, and OVA-VS. A paper describing the ClassMap formulation was presented in (Athitsos et al., 2007).

Chapter 6 evaluates the performance of the proposed methods on two tasks: hand pose estimation and face recognition where the number of classes ranges from 535 to 48,600. For each of these tasks both an SVM and a Boosted-OVA recognition system was trained. In the experiments, ClassMap, applied to all systems, and OVA-VS, applied to the Boosted-

Figure 1·1: Example recognition tasks with a large number of classes. Top: recognizing handshape and 3D orientation, out of a large number of shape/orientation combinations. Bottom: recognizing person identity, out of a large number of classes. In this paper we propose a method for quickly identifying, given a query, a small number of candidate classes, so as to speed up recognition.

OVA systems, achieved good speedups, of even two orders of magnitude in some cases. As ClassMap requires training, an experiment was conducted to see how it performs on patterns from new classes that were not available at training. The results showed good generalization properties.

Additional discussion about the proposed methods and possible directions for future work are is conducted in Chapter 7.

Chapter 2

Background and Problem Definition

Let $\mathbb{X}$ be a space of patterns, and $\mathbb{Y}$ be a finite set of class labels. Every pattern $X \in \mathbb{X}$ has a class label $L(X) \in \mathbb{Y}$. We use the term *database* as a synonym for $\mathbb{Y}$, the terms *class* and *database object* as synonyms for *class label*, and the term *query* as a synonym for *query pattern*.

In the OVA (Allwein et al., 2000) scheme, for each class $Y \in \mathbb{Y}$ a large margin classifier $C_Y : \mathbb{X} \rightarrow \mathbb{R}$ is trained to discriminate between patterns of class Y and all other patterns. For a given query pattern $Q \in \mathbb{X}$, higher (more positive) responses $C_Y(Q)$ indicate higher confidence that $L(Q) = Y$. The standard approach is to evaluate $C_Y(Q)$ for all $Y \in \mathbb{Y}$, and classify Q as belonging to the class Y for which $C_Y(Q)$ is maximized.

Other decomposition schemes have also been proposed for reducing multiclass to binary classification (Allwein et al., 2000; Dietterich and Bakiri, 1995). In all cases, a set of binary classifiers is defined, and the class label of a test pattern is determined by applying all binary classifiers, and comparing the set of binary classifier outputs to the output code assigned to each class. In this thesis, we focus on the OVA scheme, because it is readily applicable to large multiclass problems. The all-pairs scheme, where a classifier is trained for each pair of classes, requires training a number of classifiers that is quadratic to the number of classes, and thus is not scalable to problems with a large number of classes. Furthermore, the OVA scheme has been shown to be comparable in accuracy to all-pairs and other decomposition schemes in various experimental settings (Rifkin and Klautau, 2004).

Let $\mathbb{C}$ be the set of large margin OVA classifiers for a particular multiclass problem. We assume that all these binary classifiers have already been trained using an existing method, such as boosting or SVMs. We use the term *brute-force classification* for a classification

process that, as described above, in order to classify a query Q needs to compute $C(Q)$ for all $C \in \mathbb{C}$.

Given the above definitions, the problem we want to solve can be stated as follows: we want a classification process that, using the large margin classifiers in $\mathbb{C}$, classifies query patterns Q as accurately as possible and as fast as possible. Ideally, we want the classification process to be significantly faster compared to brute-force classification, but not significantly less accurate.

ClassMap, the first of the two methods proposed in this thesis, addresses the above problem in its general form, as stated in the previous paragraph. OVA-VS, the second method, addresses a more narrow version of the problem, as it is only applicable to settings where the OVA classifiers have been trained using boosting.

Chapter 3

Related Work

Nearest neighbor classification (Duda et al., 2001) is a simple method for multiclass recognition, and has been successfully applied in large multiclass problems, such as face recognition (e.g., (Liu, 2006)), contour matching (e.g., (Grauman and Darrell, 2004)), and articulated pose estimation (e.g., (Shakhnarovich et al., 2003)). Theoretically, k-nearest neighbor classification accuracy becomes optimal as the number of training data approaches infinity (Duda et al., 2001). However, for amounts of training data available in real applications, nearest neighbor classifiers often fall short of the theoretically optimal behavior.

An alternative approach for multiclass recognition is to use large margin classifiers, trained for example via boosting (Friedman et al., 2000; Li and Zhang, 2004; Schapire and Singer, 1999) or support vector machines (SVMs) (Vapnik, 1995). Compared to nearest neighbor methods, large margin methods can be appealing because of their generalization properties and good empirical performance in terms of classification accuracy. The standard strategy for applying large margin methods to a multiclass problem is to decompose the multiclass problem into a set of binary problems (Allwein et al., 2000).

Different types of multiclass-to-binary decompositions can be defined (Dietterich and Bakiri, 1995), such as OVA and all-pairs (Allwein et al., 2000). To classify a query, all binary classifiers are applied, which leads to a time complexity that is linear, for OVA, and a time complexity that is quadratic in the number of classes for all-pairs. An exception is the Directed Acyclic Graph SVM (DAGSVM) method (Platt et al., 2000), that uses the all-pairs scheme but requires a number of classifier evaluations that is only linear to the number of classes, as in the OVA scheme.

The OVA and all-pairs schemes are examples of multiclass-to-binary decomposition

schemes that are defined using error correcting output codes (ECOC) (Dietterich and Bakiri, 1995; Allwein et al., 2000). In ECOC schemes, each class is assigned an ECOC code, which is a unique vector code of some fixed length m. Each of the m entries in the code is constrained to be a value among $\{-1, 0, 1\}$. Then, for every coordinate in this space of m-dimensional vectors, a classifier is trained, where the positive and negative examples come from all classes whose code has, at that coordinate, a value of 1 or -1, respectively. Given a test pattern, all learned binary classifiers are applied, and their values are concatenated to an m-dimensional result vector, where the i-th coordinate is the output of the binary classifier trained based on the i-th coordinates of the class output codes. The pattern is finally classified as belonging to the class whose ECOC code is the closest to the result vector for that pattern.

One way to achieve classification time sublinear in the number of classes is to decompose the multiclass problem into a sublinear number of binary problems. In theory, recognizing n classes can be decomposed to $\log_2 n$ binary problems. This can be done by defining, for each class, a unique bit vector of $log_2 n$ bits. However, such sublinear decompositions are rarely used because they define binary problems with unnatural and hard-to-learn class boundaries, leading to low classification accuracy. OVA and all-pairs decompositions, on the other hand, lead to more natural binary classification boundaries, and this explains the popularity of those decompositions in practice.

A variety of indexing methods can be used to speed up nearest neighbor retrieval and classification (Athitsos, 2006; Böhm et al., 2001; Gionis et al., 1999; Hjaltason and Samet, 2003a; Hjaltason and Samet, 2003b; Shakhnarovich et al., 2003), often achieving significant speedups over brute-force search (Athitsos, 2006; Shakhnarovich et al., 2003). However, for large margin methods, brute-force evaluation of a large number of classifiers is the current state of the art. Torralba et al. (Torralba et al., 2007) proposed a method for improving the efficiency of the OVA scheme by sharing weak classifiers among the OVA models. However, at runtime, all OVA classifiers are applied to each pattern, and thus the complexity of that method is still linear to the number of classes.

The two methods proposed in this thesis perform, given a pattern, a quick search in a database of classes to identify candidate classes. This search task has many conceptual similarities with the classical task of searching for nearest neighbors. For the proposed OVA-VS method, which is applied on top of boosted OVA classifiers, our search task is actually identical to nearest neighbor search, and demonstrating that fact is the main contribution of the OVA-VS method. Therefore, classical nearest neighbor indexing methods (Athitsos, 2006; Böhm et al., 2001; Gionis et al., 1999; Hjaltason and Samet, 2003a; Hjaltason and Samet, 2003b) can be applied in that setting.

On the other hand, for the most general problem definition, where neither the OVA scheme nor boosting-based training are assumed, classical nearest neighbor methods are inapplicable, because of two issues: 1.) database objects are classes, thus living in a different space than patterns. 2.) No distance measure is defined a priori for comparing database objects (classes) to each other and to queries, whereas nearest neighbor indexing methods require such a distance measure to exist. The main contribution of the ClassMap method lies in describing how to overcome these two issues, and thereby obtain a method for efficient search in a database of classes.

We should also mention some additional methods that have been proposed for specific large multiclass problems. Efficient hand pose estimation is achieved in (Ong and Bowden, 2004) by combining hierarchical classifiers into a tree structure. Hierarchical template matching has been used for pedestrian detection (Gavrila and Philomin, 2001) and hand pose estimation (Stenger et al., 2004). Articulated pose can be treated as a multidimensional regression problem, and estimators can be trained that map observations into a continuous pose space (Agarwal and Triggs, 2006; de Campos and Murray, 2006). However, many domains (e.g., face recognition) do not lend themselves readily either to hierarchical decomposition or to regression-based estimation. The two proposed methods, on the other hand, are readily applicable in such domains, as long as a finite set of classes can be defined.

Chapter 4

ClassMap

In this chapter we describe the ClassMap method, which is one of the main contributions of this thesis. ClassMap was introduced in (Athitsos et al., 2007), and it is designed to address the most general version of our problem, where no assumptions are made about the underlying large margin method that is used to train the OVA classifiers.

The key idea in our method is to define an embedding $F : \mathbb{X} \cup \mathbb{Y} \rightarrow \mathbb{R}^{d'}$ that maps both patterns and classes into a common d'-dimensional vector space, and that tends to map queries Q and their corresponding classes $L(Q)$ close to each other. Using such an F, we can efficiently identify for each Q a set of candidate classes, by finding classes Y whose embeddings $F(Y)$ are close to $F(Q)$. It is assumed that measuring the distance between vectors $F(Q)$ and $F(Y)$ is much faster than computing the output of a binary classifier $C \in \mathbb{C}$ on a query Q. Once we obtain a set of candidate classes for Q, we only need to evaluate those binary classifiers C that are related to the candidate classes.

4.1 Jointly Embedding Queries and Classes

In order to keep our formulation general, the only assumptions that we make are that we are given a set $\mathbb{C}$ of already trained binary classifiers, a set $\mathbb{X}_{\text{tr}} \subset \mathbb{X}$ of labeled training examples, and a matrix M storing precomputed outputs $C(X)$ for all $C \in \mathbb{C}$ and $X \in \mathbb{X}_{\text{tr}}$.

Before talking about the embedding, let us see how we hope to learn something about a class, without evaluating its classifier. That is, given a query, Q, a class $Y \in \mathbb{Y}$ and a classifier, C, how can we use C to evaluate whether or not Y is a candidate class for Q? If the responses of C on samples from Y are random, nothing can be done: C offers no information about Y. If the responses follow a distribution, then we can see how likely it

is that $C(Q)$ comes from that distribution. In this thesis we assume the simple case when the distribution is unimodal and thus we can talk about a *typical response* of C for samples from Y. We will use the notation $C(Y)$ for this typical response. $C(Y)$ can be learned using the response of C on a training set of patterns from Y. Mean and median are two possible ways to estimate typical responses. To evaluate Y as a candidate for Q, we look at how far $C(Q)$ is from the $C(Y)$. We can not expect one classifier, C, to discriminate well among all the classes, for example, it may give similar typical responses for several classes. We can also not rely on it providing information about every class. In conclusion, we cannot expect a single classifier to solve the problem, but we hope that several classifiers will. Our experiments show that it does.

The next step is to use the classifiers in $\mathbb{C}$, and the information they give, to build the desired embedding. The solution is simple: any classifier C gives a 1D embedding by mapping a query, Q to $C(Q)$ and a class, Y, to the typical response, $C(Y)$. Moreover, it has the desired property that $C(Q)$ is likely to be close to $C(Y)$ when Q is a pattern of class Y. Several classifiers can be combined to produce an embedding of a higher dimension. Next we give the formal construction for the embedding.

We define an embedding $F : \mathbb{X} \cup \mathbb{Y} \rightarrow \mathbb{R}^{d'}$ that jointly maps patterns and classes into a common d'-dimensional real vector space $\mathbb{R}^{d'}$. We start by defining a simple 1D embedding F^R of both patterns $Q \in \mathbb{X}$ and classes $Y \in \mathbb{Y}$ based on the responses of a single classifier $R \in \mathbb{C}$, which we call a *reference classifier*:

$$F^R(Q) = R(Q) \ . \tag{4.1}$$

$$F^R(Y) = \text{median}\{R(X) : X \in \mathbb{X}_{\text{tr}}, L(X) = Y\} \ . \tag{4.2}$$

We should note that reference classifiers play a role analogous to that of *reference objects* in Lipschitz embeddings (Hjaltason and Samet, 2003a). Note that, for the embedding $F^R(Y)$ of a class Y, we use the median of outputs of R on training examples belonging to class Y. An alternative approach would be to use the mean instead of the median; we chose the median as a statistic that is more robust to outliers.

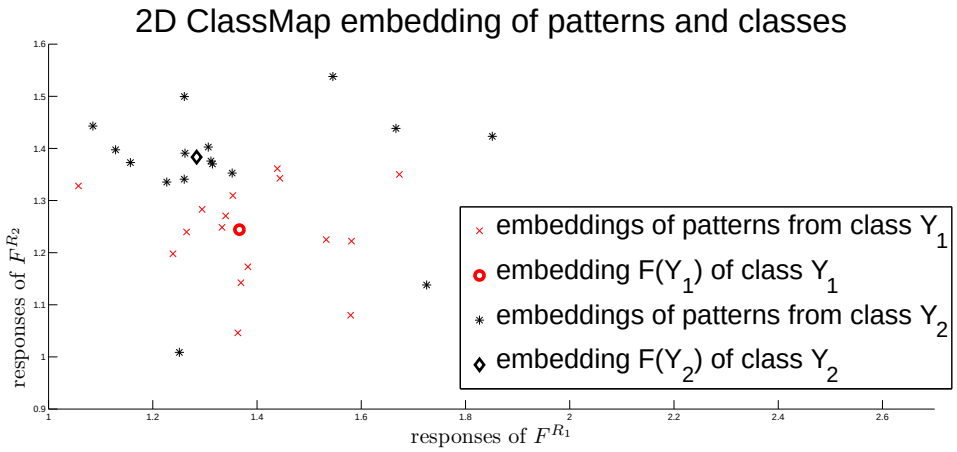

Figure 4·1: A 2D embedding $F = (F^{R_1}, F^{R_2})$, defined using Eqs. 4.3 and 4.4 with two reference classifiers R_1 and R_2, and mapping both patterns and classes into $\mathbb{R}^2$. We show the mappings of patterns belonging to classes Y_1 and Y_2, and the mappings of classes Y_1 and Y_2 themselves. Note that the mapping of each class was obtained by computing, along each dimension, the median mapping of patterns from that class along that dimension. Using the L_1 distance, 28 of the 32 patterns were mapped closer to the embedding of their own class than to the embedding of the other class.

Using 1D embeddings F^R as building blocks, we can define a multidimensional embedding $F : \mathbb{X} \cup \mathbb{Y} \rightarrow \mathbb{R}^{d'}$:

$$F(Q) = (F^{R_1}(Q), \ldots, F^{R_d}(Q)) \ . \tag{4.3}$$

$$F(Y) = (F^{R_1}(Y), \ldots, F^{R_d}(Y)) \ . \tag{4.4}$$

We use the term *ClassMap embeddings* for embeddings F defined this way. Note that any OVA classifier in $\mathbb{C}$ can be used as a reference classifier in the above equations. A simple way to construct such an embedding F in practice is to choose d' classifiers randomly from $\mathbb{C}$.

Query embeddings can be compared to embeddings of classes using any L_p metric. In our implementation we use the L_1 metric as it is more robust to large differences in a single dimension than L_p metrics with p greater than one. A large difference in a single

dimension occurs when, for some reference classifier R_i, $R_i(Q)$ is very different from the median response of R_i on objects of the same class as Q.

An example of a simple ClassMap embedding, computed with real data (faces from the 535-class FRGC2 dataset (Phillips et al., 2005)) is shown on Fig. 4·1. We illustrate a 2D embedding F defined using two OVA classifiers C_{Y_3} and C_{Y_4} as reference classifiers. The figure shows the embeddings of training examples from two other classes Y_1 and Y_2, different from the classes Y_3 and Y_4 that the reference classifiers C_{Y_3} and C_{Y_4} were trained to recognize. We see in the figure that, in most cases, F maps patterns closer to the embedding of their own class than to the embedding of the other class, when distances are measured using the L_1 metric. That is exactly the behavior that we want to exploit to achieve efficient search in the database of classes: given a query Q, we expect the embedding F of its true class $L(Q)$ to be a relatively close neighbor of $F(Q)$.

Using a ClassMap embedding we can drastically reduce the number of required classifier evaluations, by simply finding the classes whose embeddings are close to the embedding of the query. Essentially we substitute vector comparisons for classifier evaluations. This scheme leads to efficiency gains in two ways: first, we assume that measuring the distance between two vectors is much faster than applying a classifier on a pattern. Second, since the query and all database classes are mapped to a common vector space, efficient vector indexing methods, e.g., LSH (Gionis et al., 1999), can be used to speed up nearest neighbor search in that space.

4.2 A Simple ClassMap Implementation

In this section we sketch a simple end-to-end implementation that specifies both the off-line steps of preprocessing and embedding construction, and the online process of multiclass recognition using ClassMap.

ClassMap takes as input the following data:

- A set $\mathbb{C}$ of large margin OVA classifiers $C : \mathbb{X} \to \mathbb{R}$.

- A set $\mathbb{X}_{\text{tr}} \subset \mathbb{X}$ of training examples, with class labels.
- A matrix M of classifier outputs $C(X)$ for each pair $(C, X) : C \in \mathbb{C}, X \in \mathbb{X}_{\text{tr}}$.

The first objective is to construct a ClassMap embedding F. A simple approach is to first choose d', i.e., the dimensionality of the embedding, and then simply choose randomly d' reference classifiers $R_1, \ldots, R_d$ from $\mathbb{C}$ and apply Equations 4.3 and 4.4. The last preprocessing step is to compute and store $F(Y)$ for each class $Y \in \mathbb{Y}$.

Once preprocessing is done, we can proceed to the runtime phase of ClassMap, i.e., classification of previously unseen query patterns. For the runtime phase we adapt the filter-and-refine framework (Hjaltason and Samet, 2003a). We have used two different versions of filter-and-refine retrieval in our experiments, described as follows:

Version 1 (simple OVA)

- **Input**: query $Q \in \mathbb{X}$, to be classified.
- **Embedding step**: compute $F(Q)$.
- **Filter step**: rank all classes Y by the distance of their embeddings $F(Y)$ from $F(Q)$.
- **Refine step**: for some user-specified number p, compute all responses $C_Y(Q)$ such that class Y was ranked in the top p classes by the filter step. Assign Q to the class Y of the classifier C_Y producing the highest response.

Version 2 (OVA + threshold)

- **Input**: query $Q \in \mathbb{X}$, to be classified, threshold t (same for all queries) .
- **Embedding step**: compute $F(Q)$. If, during computing $F(Q)$, for some reference classifier C_Y it is the case that $C_Y(Q) \geq t$, then assign Q to class Y corresponding to C_Y and finish.
- **Filter step**: rank all classes Y by the distance of their embeddings $F(Y)$ from $F(Q)$.

- **Refine step**: given user-specified p: start computing $C_Y(Q)$, according to the order in which Y was ranked at the filter step. If, for some Y, $C_Y(Q) \geq t$, then assign Q to class Y and finish. If p classes have already been considered, classify Q as in the simple OVA case.

We note that there is a large amount of flexibility in designing the refine step. In addition to the two versions provided above, several other versions may be reasonable choices for specific domains. Our focus in this thesis is not the specific implementation of the refine step, but the design of appropriate embeddings F to be used for the filter step, within the general filter-and-refine framework.

The rationale of the OVA + threshold version of the refine step is that higher responses $C_Y(Q)$ indicate higher confidence that Q indeed belongs to class Y. Responses $C_Y(Q)$ higher than some threshold t may be so conclusive that we can safely classify Q, without performing any more classifier evaluations. The threshold t can be learned from training examples, so as to rarely lead to incorrect decisions.

Regardless of the particular implementation of the refine step, the key idea in filter-and-refine classification is that the filter step, using efficient vector comparisons, can quickly identify a relatively small set of candidate classes. Then, the refine step uses more expensive computations (classifier evaluations) to choose one among those candidates.

4.3 Optimizing Embedding Quality

In order for F to be useful for filter-and-refine classification, F should tend to map queries closer to their own class than to other classes. In other words, if $Q \in \mathbb{X}$ is a random query of class $L(Q)$, and $Y \neq L(Q)$ is a random class in the database, we want it to hold as often as possible that $F(Q)$ be closer to $F(L(Q))$ than to $F(Y)$. Instead of choosing random reference classifiers, as suggested in Section 4.2, we can optimize embeddings according to this criterion.

In particular, let Δ be the distance measure used in $\mathbb{R}^{d'}$, and let $Q \in \mathbb{X}, Y_1 = L(Q), Y_2 \neq L(Q)$. For every embedding $F : \mathbb{X} \cup \mathbb{Y} \rightarrow \mathbb{R}^{d'}$ we define a corresponding classifier

$\tilde{F} : \mathbb{X} \times \mathbb{Y} \times \mathbb{Y} \rightarrow \mathbb{R}$, as follows:

$$\tilde{F}(Q, Y_1, Y_2) = \Delta(F(Q), F(Y_2)) - \Delta(F(Q), F(Y_1)) \ . \tag{4.5}$$

In words, the task of $\tilde{F}$ is to decide whether $L(Q) = Y_1$ or $L(Q) = Y_2$. The decision simply depends on whether $F(Q)$ is closer to $F(Y_1)$ or to $F(Y_2)$. Positive and negative outputs of $\tilde{F}$ correspond respectively to decisions that $L(Q) = Y_1$ and $L(Q) = Y_2$. We want to construct an F so that the error rate of $\tilde{F}$ on triples $(Q, Y_1, Y_2) : Y_1 = L(Q), Y_2 \neq L(Q)$ is minimized.

For the sake of clarity we should emphasize that two entirely different types of classifiers appear in our formulation: The first type is large margin OVA classifiers $C : \mathbb{X} \rightarrow \mathbb{R}$. The second type is classifiers $\tilde{F}$ associated with embeddings F, where $\tilde{F} : \mathbb{X} \times \mathbb{Y} \times \mathbb{Y} \rightarrow \mathbb{R}$ maps a triple (Q, Y_1, Y_2) to a real number. In the remainder of the paper we will refer to classifiers of type $\tilde{F}$ using the term *triple-classifiers*. 184 Every $R \in \mathbb{C}$ can be used to define a 1D embedding F^R. A triple-classifier $\tilde{F}^R$ is expected to act as a weak classifier, with possibly high error rate, but better performance than a random guess (see, for example, Fig. 4·1). We will now discuss how to combine many such weak triple-classifiers into an optimized strong triple-classifier, and a corresponding optimized multidimensional embedding, using AdaBoost. This idea comes from (Athitsos, 2006), where AdaBoost is used to optimize embeddings for nearest neighbor retrieval.

The inputs we give to the embedding construction algorithm are the same as in Section 4.2: a set $\mathbb{C}$ of large margin classifiers, a set $\mathbb{X}_{\text{tr}} \subset \mathbb{X}$ of labeled training examples, and a matrix M of classifier outputs $C(X)$ for each pair $(C, X) : C \in \mathbb{C}, X \in \mathbb{X}_{\text{tr}}$.

Embedding construction is performed as follows:

1. We define for each $R \in \mathbb{C}$ a 1D embedding F^R.

2. For each F^R we also define the corresponding weak triple-classifier $\tilde{F}^R$.

3. We construct a set $\mathbb{T}$ of training triples $(X, L(X), Y)$ such that $X \in \mathbb{X}_{\text{tr}}, Y \in \mathbb{Y} - \{L(X)\}$.

4. We run AdaBoost (Schapire and Singer, 1999) using $\mathbb{T}$ as training data, so as to combine many weak triple-classifiers of type $\tilde{F}^R$ into a strong triple-classifier H:

$$H = \sum_{i=1}^{d'} (\alpha_i \tilde{F}^{R_i}) \ , \tag{4.6}$$

where each R_i is an element of $\mathbb{C}$ and each weight α_i is a positive real number. AdaBoost is used to choose R_i and α_i.

5. Based on strong classifier H we define a d'-dimensional embedding F_{out} and a distance measure $\Delta : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \rightarrow \mathbb{R}$ as follows:

$$F_{\text{out}}(Q) = (F^{R_1}(Q), ..., F^{R_{d'}}(Q)) \ . \tag{4.7}$$

$$F_{\text{out}}(Y) = (F^{R_1}(Y), ..., F^{R_{d'}}(Y)) \ . \tag{4.8}$$

$$\Delta((u_1, ..., u'_d), (v_1, ..., v_{d'})) = \sum_{i=1}^{d'} (\alpha_i |u_i - v_i|) \ . \tag{4.9}$$

Eqs. 4.7 and 4.8 use the definition of F^R given in Eqs. 4.1 and 4.2. In Eq. 4.9, $(u_1, ..., u_{d'})$ and $(v_1, ..., v_{d'})$ are d'-dimensional vectors that F_{out} maps patterns and classes to.

Following the proof in (Athitsos, 2006) for AdaBoost-trained embeddings, it holds that $H = \tilde{F}_{\text{out}}$, where $\tilde{F}_{\text{out}}$ is the triple-classifier constructed from F_{out} according to 4.5. In other words, the classifier H trained via AdaBoost misclassifies a triple $(Q, L(Q), Y \neq L(Q))$ iff, under distance Δ, F_{out} maps Q closer to Y than to $L(Q)$. Therefore, AdaBoost directly optimizes the error rate of classifier $\tilde{F}_{\text{out}}$, which is exactly the measure we wanted to optimize for the purposes of using F_{out} for filter-and-refine classification.

4.4 Summary

ClassMap is a novel approach for speeding up recognition in the presence of a large number of classes. The ClassMap formulation can be applied in settings where multiclass recognition is performed using large margin OVA classifiers. The key idea is to relate patterns

and classes by constructing a joint embedding, that maps both patterns and classes into a common vector space. Using this embedding, a small number of candidate classes for each query can be quickly identified using simple vector comparisons.

Chapter 5

Boosted OVA-Based Classification Using Vector Search Methods

This chapter describes OVA-VS (One-vs.-All classification using Vector Search). We designed this method to speed up multiclass recognition in the special case where boosted OVA classifiers are used.

In that setting, for each class $Y, \in \mathbb{Y}$ a boosted classifier $C_Y : \mathbb{X} \rightarrow \mathbb{R}$ is trained to discriminate between patterns of class Y and all other patterns. Classifier C_Y is of the following form:

$$C_Y(Q) = \sum_{i=1}^{d} \alpha_{Y,i} w_i(Q) \tag{5.1}$$

where each w_i is a weak classifier with weight $\alpha_{Y,i}$ (Allwein et al., 2000; Torralba et al., 2007). Note that the weights $\alpha_{Y,i}$ depend on the class Y. On the other hand, without loss of generality, in our formulation, the weak classifiers w_i do not depend on class Y: we define $\mathbb{W} = \{w_1, \ldots, w_d\}$ to be the union of all weak classifiers appearing in any C_Y. If some weak classifier w_i is not used in some strong classifier C_Y, we simply set $\alpha_{Y,i} = 0$.

Higher (more positive) responses of $C_Y(Q)$ indicate higher confidence that $L(Q) = Y$. To classify a query $Q \in \mathbb{X}$, the standard approach is to evaluate $C_Y(Q)$ for all $Y \in \mathbb{Y}$, and classify Q as belonging to the class Y for which $C_Y(Q)$ is maximized. More specifically, if we denote as $C(Q)$ the output of the multiclass classifier C for pattern Q, $C(Q)$ is defined as:

$$C(Q) = \text{argmax}_{Y \in \mathbb{Y}} C_Y(Q) \ . \tag{5.2}$$

In order for the outputs $C_Y(Q)$ to be comparable for a given Q and different class labels

Y, weights $\alpha_{Y,i}$ must be normalized. We assume that weights are normalized so that the norm of each vector $(\alpha_{Y,1}, \ldots, \alpha_{Y,d})$ is 1, for all $Y \in \mathbb{Y}$.

At runtime, given a pattern Q to classify, the standard approach is to apply all OVA classifiers C_Y, and identify the Y such that C_Y gives the strongest response. Clearly, this approach has complexity linear to the number of classes. In this chapter we show show that the strongest-responding C_Y classifier can be found efficiently, using vector search methods, without needing to evaluate $C_Y(Q)$ for all Y. This topic is addressed in the next sections.

5.1 Reduction to Vector Search

The core observation underlying OVA-VS is that, for boosted OVA-based multiclass recognition, both test patterns and OVA classifiers can be represented as vectors, specifying points on the surface of a unit hypersphere. Finding for a test pattern Q the strongest-responding OVA classifier C_Y can be done by doing nearest neighbor search on those points.

In particular, we denote by $V(Q)$ and $V(C_Y)$ respectively the vectors corresponding to test pattern Q and OVA classifier C_Y. These vectors are defined as follows:

$$V_{orig}(Q) = (w_1(Q), \ldots, w_d(Q)) \ , \tag{5.3}$$

$$V(Q) = \frac{V_{orig}(Q)}{\|V_{orig}(Q)\|} \ , \tag{5.4}$$

$$V(C_Y) = (\alpha_{Y,1}, \ldots, \alpha_{Y,d}) \ , \tag{5.5}$$

where w_i and $\alpha_{Y,i}$ are the weak classifiers and weights used in Equation 5.1. As a reminder, the same weak classifiers w_i are used in the definition of all strong classifiers C_Y. It is the weights $\alpha_{Y,i}$ that differentiate OVA classifiers C_Y from each other. As defined in the above equations, vector $V_{orig}(Q)$ is the concatenation of the responses of all weak classifiers w_i on Q. Vector $V(Q)$ is obtained by normalizing $V_{orig}(Q)$ to unit length ($\|V\|$ denotes the norm of vector V).

Using these definitions, Equation 5.2 can be rewritten as follows:

$$C(Q) = \text{argmax}_{Y \in \mathbb{Y}} C_Y(Q) \tag{5.6}$$

$$= \text{argmax}_{Y \in \mathbb{Y}} (V_{orig}(Q) \cdot V(C_Y)) \tag{5.7}$$

$$= \text{argmax}_{Y \in \mathbb{Y}} (V(Q) \cdot V(C_Y)) \ , \tag{5.8}$$

where $V_1 \cdot V_2$ denotes the dot product between vectors V_1 and V_2.

We recall from Section 2 that weights α_{y_i} are normalized so that the norm of each vector $V(C_Y) = (\alpha_{Y,1}, \ldots, \alpha_{Y,d})$ is 1. Also, $V(Q)$ is by definition a vector of norm 1. Therefore, for all test patterns $Q \in \mathbb{X}$ and all OVA classifiers C_Y, their vector representations $V(Q)$ and $V(C_Y)$ are unit vectors in d-dimensional real vector space $\mathbb{R}^d$.

Equation 5.8 indicates that, to classify pattern Q, we need to find the class Y that maximizes dot product $V(Q) \cdot V(C_Y)$. However, since both $V(Q)$ and $V(C_Y)$ are unit vectors, we can easily show that maximizing $V(Q) \cdot V(C_Y)$ is the same as minimizing the Euclidean distance between $V(Q)$ and $V(C_Y)$, because the dot product and the Euclidean distance for unit vectors are related as follows:

$$\|V(Q) - V(C_Y)\|^2 = 2 - 2(V(Q) \cdot V(C_Y)) \ . \tag{5.9}$$

The above equation can be easily derived as follows:

$$\|V(Q) - V(C_Y)\|^2 = (V(Q) - V(C_Y)) \cdot (V(Q) - V(C_Y)) \tag{5.10}$$

$$= (V(Q) \cdot V(Q)) + (V(C_Y) \cdot V(C_Y)) - 2(V(Q) \cdot V(C_Y)) \tag{5.11}$$

$$= 2 - 2(V(Q) \cdot V(C_Y)) \ , \tag{5.12}$$

using the fact that $(V(Q) \cdot V(Q)) = (V(C_Y) \cdot V(C_Y)) = 1$.

Intuitively, this can be explained as follows: first, it is clear that normalizing OVA classifiers is necessary in order to be able to compare their responses. Second, query vectors can be normalized without loss of generality. This way, both query vectors and

classifier vectors are mapped to the surface of the unit hypersphere. We also observe that the response of an OVA classifier on a query is, by definition, the dot product between the weights of the weak classifiers and the responses of those weak classifiers on the query. In general, to obtain a high dot product, classifiers with negative weights and positive weights should tend to have negative and positive responses respectively on the query. Therefore, if an OVA classifier gives a high response on a query, it means that the dot product of their associated unit vectors is high, which then implies that those unit vectors should be close to each other.

Consequently, we have established that, given a test pattern Q, finding the strongest-responding OVA classifier C_Y is reduced to finding the nearest neighbor of $V(Q)$ among all vectors $V(C_Y)$ by finding the class with greatest dot product. The next section describes how to use that fact for speeding up multiclass recognition.

5.2 Using Nearest Neighbor Search Methods for OVA-Based Classification

So far we have established that, to classify a test pattern Q, it suffices to find the nearest neighbor of $V(Q)$ among all vectors $V(C_Y)$. Clearly, vectors $V(C_Y)$ can be computed off-line and stored in a database, and indexing methods can be used to preprocess this database and facilitate nearest neighbor search. We can use any of the numerous indexing methods that have been proposed for nearest neighbor search in vector spaces (Böhm et al., 2001; Gionis et al., 1999; Hjaltason and Samet, 2003b; Kanth et al., 1998; Li et al., 2002; Sakurai et al., 2000; Tuncel et al., 2002; Weber and Böhm, 2000; Weber et al., 1998).

One method that has become popular in recent years is locality sensitive hashing (LSH) (Gionis et al., 1999). LSH can be used to find nearest neighbors, with high probability, in time $O(d \log n)$, where d is the dimensionality of the space (equal to the number of weak classifiers in our setting) and n is the number of database vectors (equal to the number of classes in our setting). The probability of successful retrieval can be set as high as desired, at the cost of increasing time complexity. For any fixed probability of success, retrieval

time is sublinear in the number of vectors in the database (Gionis et al., 1999).

LSH is applied to a vector space $\mathbb{R}^d$ as follows: let $\mathcal{H}$ be a family of hash functions $h : \mathbb{R}^d \rightarrow \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers. As described in (Gionis et al., 1999), $\mathcal{H}$ is called *locality sensitive* if there exist real numbers r_1, r_2, p_1, p_2 such that $r_1 < r_2$, $p_1 > p_2$, and for any $X_1, X_2 \in \mathbb{X}$:

$$D(X_1, X_2) < r_1 \Rightarrow \Pr_{h \in \mathcal{H}}(h(X_1) = h(X_2)) \geq p_1 \ . \tag{5.13}$$

$$D(X_1, X_2) > r_2 \Rightarrow \Pr_{h \in \mathcal{H}}(h(X_1) = h(X_2)) \leq p_2 \ . \tag{5.14}$$

Given a locality sensitive family $\mathcal{H}$, LSH indexing works as follows: first, we pick integers k and l. Then, we construct l hash functions $g_1, g_2, \ldots, g_l$, as concatenations of k functions chosen randomly from $\mathcal{H}$:

$$g_i(X) = (h_{i1}(X), h_{i2}(X), \ldots, h_{ik}(X)) \ . \tag{5.15}$$

Each database object X is stored in each of the l hash tables defined by the functions g_i. Given a query object $Q \in \mathbb{X}$, the retrieval process first identifies all database objects that fall in the same bucket as Q in at least one of the l hash tables, and then exact distances are measured between the query and those database objects. In our implementation, binary hash functions are defined based on random unit vectors V: for each V, a binary hash function is defined by computing the dot product between V and the vector to be hashed, and thresholding the result.

Although LSH has good time complexity in theory, this theoretical complexity does not necessarily translate to good performance in practice, and this is shown in some of our experiments in Section 6.3. An alternative is to use dimensionality reduction. Since the set of vectors $V(C_Y)$ is computed off-line, we can use those vectors for an additional off-line step, where dimensionality reduction is used to produce lower-dimensional approximate representations of those vectors. Given a test pattern Q, its vector $V(Q)$ can be projected to the lower-dimensional space online, and then it can be compared to the lower-dimensional

projections of the database vectors.

Dimensionality reduction can be used within a filter-and-refine retrieval framework (Ferhatosmanoglu et al., 2001): given an integer parameter p, filter-and-refine works as follows:

- **Input:** A test pattern Q, and its vector representation $V(Q)$.
- **Filter step:** Project $V(Q)$ to the lower-dimensional space, and find the nearest neighbors of the projection of $V(Q)$ among the projections of the database vectors. Keep the top p nearest neighbors.
- **Refine step:** For each of the top p nearest neighbors, apply the corresponding C_Y to Q.
- **Output:** Return the C_Y yielding the strongest response $C_Y(Q)$, among the C_Y's evaluated during the refine step.

One dimensionality reduction approach that we have experimented with is principal component analysis (PCA) (Jolliffe, 1986). In particular, we computed the principal components of the set of vector representations of all OVA classifiers. A disadvantage of PCA is that the cost of computing the PCA projection of the query can be significant, as demonstrated in the experiments. Another dimensionality reduction approach that we have used, that does not incur a significant projection cost, is to simply sample a subset of the original dimensions. To choose d' out of the original d dimensions, we tried two different sampling methods:

- **Sampling-1:** choose the first d' dimensions. The rationale here is that the first d' dimensions correspond to the d' weak classifiers that were chosen first during training, and thus convey the most information.
- **Sampling-2:** choose d' dimensions randomly.

The cost of using a PCA-based filter-and-refine process is approximately $2 \cdot [d'(n+d) + p \cdot d]$ flops. This number reflects the total cost of all steps: projecting $(d' \cdot d)$, filtering $(n \cdot d')$,

and refinement ($p \cdot d$). If we use sampling instead of PCA, the projection cost becomes independent of d, which can be an important source of computational savings, especially for larger d's and smaller n's. The cost of OVA classification is approximately $2 \cdot (n \cdot d)$ flops.

Naturally, dimensionality reduction and LSH can also be combined with each other in various ways. For example, since LSH behaves better in smaller dimensions, we can first apply dimensionality reduction to project vectors into a lower-dimensional space, and then we can use LSH to index that space. We tested this combination in an experiment which showed that for the same method parameters, LSH applied to the original space had poor accuracy vs. efficiency trade-offs, and applied to the lower-dimensional space it achieved significantly better trade-offs.

5.3 A Note on Sharing Classifiers

In the previous sections we have shown that multiclass recognition using boosted OVA classifiers can be performed efficiently using nearest neighbor search methods. However, it is well-known that nearest neighbor search methods perform worse as the dimensionality of the space increases. For example, the running time of LSH is linear to the dimensionality. In the worst case, the set of weak classifiers used by any classifier C_Y is disjoint of the set of weak classifiers used by all other OVA classifiers. We have seen that PCA is one way to reduce the dimensionality of the space. An additional, and complimentary, method is to reduce the number d of unique weak classifiers appearing in the OVA classifiers by forcing all OVA classifiers to use the same weak classifiers, thus reducing the total number of unique weak classifiers (Torralba et al., 2007). It turns out that forcing OVA classifiers to share weak classifiers is also beneficial with respect to classification accuracy, and training methods have been recently proposed that enforce the sharing of weak classifiers (Torralba et al., 2007). Using such training methods also leads to reduced dimensionality in the vector representation of OVA classifiers and test patterns, thus facilitating the application of vector indexing methods like those proposed in this thesis.

Chapter 6

Experiments

The methods proposed in this thesis are designed to be applied on top of existing OVA classification systems. They use the response of the OVA classifiers. Therefore each dataset depends on the brute-force classification system (e.g. SVM OVAs, boosted OVAs), and the *original* dataset, that is, the dataset of patterns to be classified (e.g. hand images, face images, video files of signs, audio files of sounds). The datasets used in our experiments were generated from two original datasets: a dataset of hand images, where the task is to estimate the handshape and the 3D orientation, and the Face Recognition Grand Challenge (FRGC) Version 2 dataset (Phillips et al., 2005) of 2D face images. Example images from these datasets are shown in Fig. 1·1. The hands dataset contains hand images of 81 basic handshapes defined in American Sign Language (ASL) at 30 3D orientations yielding 2,430 number of classes. The faces dataset contains all 2D face images in the FRGC2 dataset, amounting to 36,817 face images from 535 subjects (i.e., 535 classes). The OVA systems trained to classify the patterns in these datasets are:

1. SVMs with a linear kernel for hand pose estimation,
2. SVMs with a Gaussian RBF kernel for face recognition,
3. jointly boosted classifiers for hand pose estimation, and
4. jointly boosted classifiers for face recognition.

The above methods are our *brute-force methods*: they require the application of all OVAs for classification. ClassMap was applied to all the datasets. OVA-VS was applied

to the last two datasets. Since by definition OVA-VS requires boosted classifiers, it cannot be applied to datasets 1 and 2.

Next, we describe each of the two datasets, and then we provide and discuss the results of our experiments.

6.1 Datasets

The Hand Dataset

This dataset contains hand images of 81 basic hand shapes defined in American Sign Language (ASL). There are 30 different out-of-plane view angles for each shape, for a total of $81 \times 30 = 2,430$ hand pose classes. The inplane orientation is fixed. For each class, 200 synthetic images were generated using Poser 5 (Curious Labs, 2002). For each synthetic hand image, cluttered background from random real images was added to the regions outside the hand silhouette.

From each hand image, a histogram of oriented gradient feature vector (Dalal and Triggs, 2005) of dimension 2,025 was extracted. The image was normalized to 48 by 48 pixels, which was divided into cells of size 6 by 6, with neighboring cells overlapping by half. For each cell, nine edge orientation bins were evenly spaced between 0 to 180 degrees. Bins in each cell were normalized with the surrounding 3 by 3 cells. All the bins from all the cells were vectorized into a feature vector of 2025 feature components for a hand sample.

To generate a domain with large number of classes, we enhanced the dataset of hand images, to include 20 in-plane rotations per viewpoint, for a total of 81 handshapes $\times$ 30 out-of-plane rotations $\times$ 20 in-plane rotations $= 48,600$ hand pose classes.

For evaluation, in addition to the synthetic hand images, we also used a second test set of 992 real hand images, collected from 7 subjects and with cluttered background. The real test images cover 13 out of the 2,430 classes. We collected real images from only a few classes in order to facilitate the extremely laborious process of manually annotating the ground truth for those images. Furthermore, because of the difficulties in visually

estimating the 3D hand orientation on an image, we assigned to each hand image four different class labels (out of the possible 2,430 class labels). Each of those four class labels corresponded to the same handshape and a 3D orientation within 30 degrees of the manually labeled orientation. The classification result is considered correct iff it was equal to one of those four labels.

The Face Dataset

This dataset contains all 2D face images in the FRGC2 dataset (Phillips et al., 2005), amounting to 36817 face images from 535 subjects (i.e., 535 classes).

The original resolution of the face images was either 1704×2272, or 1200×1600. All images were converted to gray images and normalized to 100 by 100 pixels. A PCA space was learned from 4,000 uniformly sampled training faces of all the subjects. The features of face images were their projections on the top 2,509 PCA components, which accounts for 99.9% of the variance.

We also created an alternative, smaller test set, which we call the faces-25 test set. In faces-25 we included all test images from classes for which at least 25 training examples were available for the embedding construction algorithm. The faces-25 set was useful for illustrating how performance of ClassMap was affected when the number of training examples per class becomes too small. Images in the faces-25 test set were still classified against all 535 classes.

6.2 Experimental Evaluation of the ClassMap Method

SVM classifiers were used for both the hands and the faces datasets. For the hands dataset, for each class, a linear kernel SVM was trained using 150 positive samples of that class, and 48,000 negative samples containing images uniformly sampled from hand images of other classes. When ClassMap was applied for this dataset, out of the remaining 50 samples per class, 25 were used as training during embedding construction via AdaBoost, and 25 for testing.

For the faces dataset, the 535 OVAs face classifiers were trained using SVMs with a Gaussian RBF kernel with variance 10. For each OVA classifier the positive training samples contain the training samples of a specific class and all training samples from the other classes are negative training samples. In particular, for each subject the sample images were split into three disjoint sets: half of the face images were used for training OVA classifiers, 1/4 were used for the embedding construction of ClassMap, and 1/4 were used for testing ClassMap.

In our experiments we evaluated five different methods: brute-force classification, and four different variations of ClassMap, that use different ways to construct ClassMap embeddings, and different versions of filter-and-refine retrieval (see Section 4.2). The methods we have used are the following:

- **Brute force**: evaluate all OVA classifiers; select the class whose classifier produced the highest response.
- **CM-RRC**: use version 1 (simple OVA) of filter-and-refine classification, and use a randomly generated ClassMap embedding.
- **CM-RRC-Thr**: use version 2 (OVA+threshold) of filter-and-refine classification, and use a randomly generated ClassMap embedding.
- **CM-Boosted**: use version 1 (simple OVA) of filter-and-refine classification, and a ClassMap embedding constructed using AdaBoost.
- **CM-Boosted-Thr**: use version 2 (OVA+threshold) of filter-and-refine classification, and a ClassMap embedding constructed using AdaBoost.

In training CM-Boosted embeddings, using the algorithm of Section 4.3, different training triples were used for each dataset. Within each dataset, there was a single run of ClassMap. The number of triples was the same (i.e., was 3 million) in all runs. The CM-Boosted embeddings had 45 dimensions ($d = 45$) for the hand dataset, and 84 dimensions for the face dataset. The number of dimensions is simply the number of reference classifiers

to which AdaBoost assigned non-zero weights. The same embedding was used for both real and synthetic test images of hands, and the same embedding was used for both the faces and the faces-25 test sets. To make comparisons fair, the same dimensions (45 for hands and 84 for faces) were also used for the CM-RRC embeddings.

For the face dataset, for version 2 of the filter-and-refine classification of Section 4.2, the threshold t was set to -0.38. This value was chosen by considering the training examples: -0.38 was the smallest value that would increase classification error on the training examples by no more than 0.05%.

6.2.1 Results

Performance was measured based on classification accuracy and efficiency. Fig. 6·1 and Fig. 6·2 display the results obtained for the four test sets (real hand images, synthetic hand images, faces, and faces-25) using brute force, CM-Boosted, and (for the faces and faces-25 sets) CM-Boosted-Thr. To evaluate different accuracy-vs.-efficiency trade-offs, we varied parameter p of the refine step (Section 4.2), which specifies the maximum number of candidate classes to consider. One measure of efficiency is the number of OVA classifier evaluations per query. OVA classifiers were evaluated at the embedding step, in order to produce the embedding of the query, and at the refine step, where the OVA classifiers corresponding to the candidate classes are evaluated. However, the difference between the 23.9% accuracy of ClassMap and the 22.5% accuracy of brute force search is not statistically significant, as it has a probability of about 15.3% of occurring by chance. Therefore, we cannot make the claim that ClassMap actually improves the classification accuracy on this dataset. We can, however, state that we achieved results comparable to those of brute force search, at a significant speedup.

On the hand images, the brute-force classification accuracy was 22.5% for the real images and 95.3% for the synthetic images.

At the cost of 85 classifier evaluations per query ($d = 45, p = 40$), CM-Boosted produced 23.9% classification accuracy for the real images and 95.3% for the synthetic images.

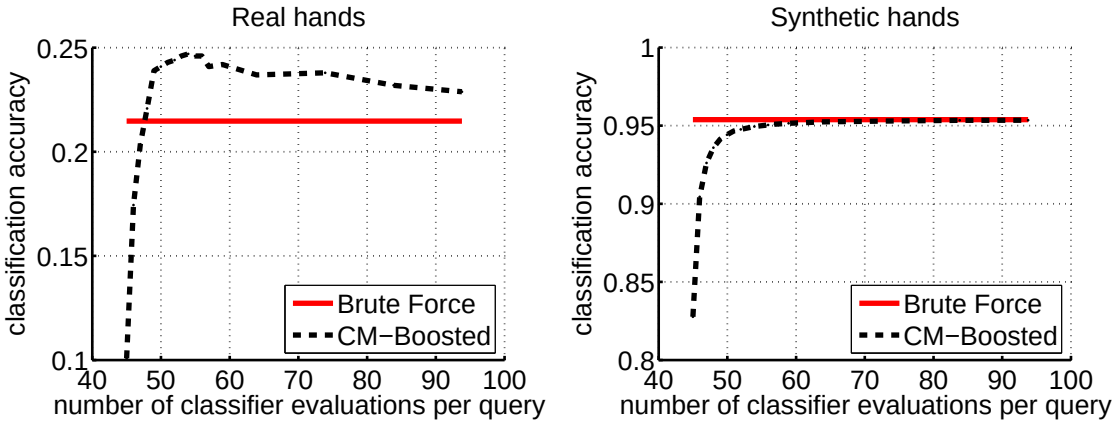

Figure 6·1: Results on the real and synthetic test sets of hand images test sets. For method CM-Boosted we plot accuracy vs. number of OVA classifier evaluations per query. We also show as a solid horizontal line the brute force classification accuracy, attained at a cost of 2430 evaluations per query.

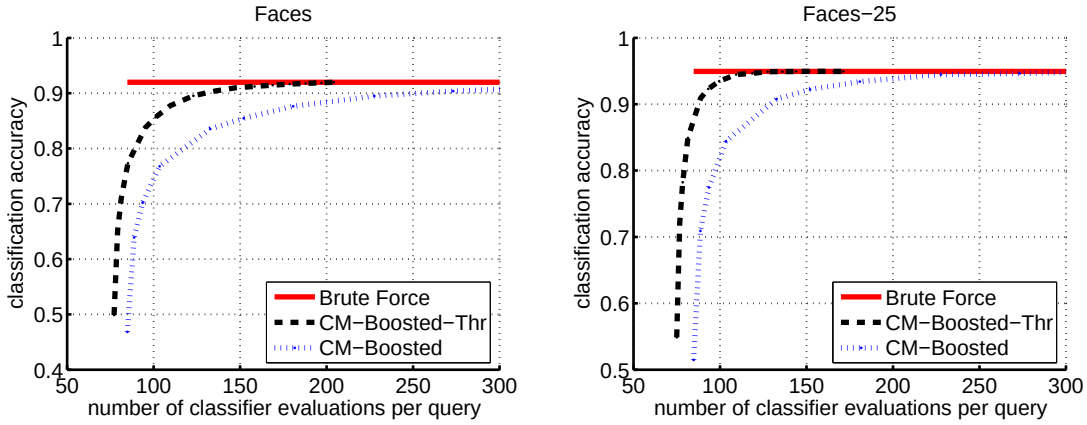

Figure 6·2: Results on the faces and faces-25 test sets. For methods CM-Boosted and CM-Boosted-Thr we plot accuracy vs. number of OVA classifier evaluations per query. We also show as a solid horizontal line the brute force classification accuracy, attained at a cost of 535 classifier evaluations per query.

Interestingly, for the real hand images, CM-Boosted was both faster and more accurate than brute force. A pattern misclassified via brute force can be classified correctly via the filter-and-refine method, if the OVA classifier(s) producing false alarms were not associated with candidate classes considered during the refine step.

In terms of running time for the hand dataset, the speed of brute force classification is 28 images/second, and the speed of CM-Boosted (at $d = 45, p = 40$, and with no loss in classification accuracy) is 781 images/second, which is 28 times faster than brute force. In a detection setting, where hundreds or thousands of image windows are classified separately in order to determine where the hand is located, the speed-up factor of 28 produced by ClassMap can make a big difference in practice.

We note that the classification accuracy on the dataset of real hand images for the SVM-OVA was 22.5% which is relatively low for a large margin method. Recognizing handshapes in arbitrary 3D orientations remains a challenging task, as evidenced by the high error rates in our experiments. However, these error rates correspond, in some sense, to a worst-case scenario, where no prior information is available as to what 3D orientations and handshapes are most likely to be observed. Such prior information is oftentimes available in real-world systems, and can come from the following sources:

- Specific usage scenarios, where the user is typically facing in a certain direction with respect to the camera and makes handshapes with a limited range of 3D orientations.
- Knowledge of specific human-computer communication protocols, that involve a relatively small number of gestures, thus restricting possible handshapes and 3D orientations.
- Use of multiple cameras, which can resolve ambiguities that are unavoidable in systems that only use a single camera.
- Use of linguistic constraints in the context of sign language recognition. For example, given the handshape of the dominant hand there is a relatively small number of possible handshapes for the non-dominant hand.

- Use of information from multiple consecutive frames in a video sequence. The method described in this paper can be a source of hypotheses for initializing a hand tracker. Such a tracker can use information from multiple frames to improve upon the accuracy of estimates made based on a single frame.

For the face dataset, ClassMap again provides a more efficient alternative to brute-force classification. Brute-force classification achieves an accuracy of 92.0%, and it takes 2.17 seconds to classify a face image. At a cost of 178 classifier evaluations per query, the CM-Boosted-Thr method yields an accuracy of 91.6%, and it takes 0.73 seconds to classify a face image, which is 3.0 times faster than brute force. At the refine step, CM-Boosted-Thr runs the OVAs in the order given by the filter step. If a classifier, C_Y, gives a response higher than a threshold t, the process is stopped and the query is assigned class Y. Otherwise it evaluates all of the p classifiers, as CM-Boost. This flrexibility allows CM-Boost-Thr to go up to $p = 400$ for the difficult queries and still achieve the cost of 178 per query on average.

In order to investigate the accuracy trade-off with respect to the number of samples per class available for training of ClassMap, we selected a subset of the test set, which we refer to as faces-25. Faces-25 contains only samples from classes that provided at least 25 samples for training. For this experiment, the embedding was trained using all of the 535 classifiers and the database consisted of the embedding of all 535 classes. At a cost of 128 classifier evaluations per query, the CM-Boosted-Thr method yielded an accuracy of 94.9%, equal to the accuracy of brute force, at the speed of 0.53 seconds per image, which is 4.1 faster than brute force.

Fig. 6·3 compares the single embedding constructed via AdaBoost vs. results from 100 randomly constructed embeddings. Interestingly, for the real hand images, several random embeddings performed better than the AdaBoost-constructed embedding. Since AdaBoost in general converges only to a local optimum, it is always possible that a random construction turns out to be better than this local optimum. In the faces dataset, the AdaBoost-constructed embedding does outperform all 100 CM-RRC embeddings. Still the randomly constructed ClassMap embeddings performed better than brute force, achieving

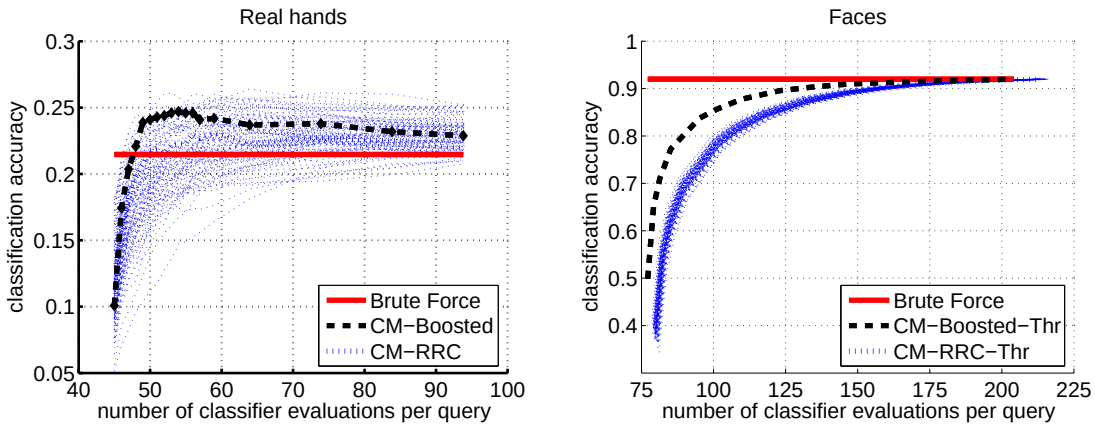

Figure 6·3: Comparing AdaBoost-based embedding construction to random embedding construction. For the real hand test set and the faces test set, we compare the single embedding constructed via AdaBoost vs. results from 100 randomly constructed embeddings.

similar accuracy at more than twice the speed.

One last experiment measured the generalization ability of CM-Boosted embeddings. In this experiment, AdaBoost used 435 classifiers and examples from only 435 out of 535 classes. The remaining 100 classes and classifiers were presented to the system after the embedding was learned. Then, for each new class (out of the 100), using the embedding (based on the 435 classifiers) and the training samples of the class, the class embedding was computed and added to the database of embedded classes. The test set included only examples from the left-out 100 classes. In Fig. 6·4 we compare the results to those obtained (on the same test set from the left-out 100 classes) using the original embedding, that was trained on all 535 classes. Not surprisingly, performance was worse for the embedding that was trained without examples from the classes in the test set. At the same time, performance on those left-out classes was still better than that of CM-RRC embeddings and brute force. For example, at a cost of 110 classifier evaluations per query, the embedding trained without examples from the test classes achieved an accuracy of 84.5%, which is worse than the 87% accuracy of the embedding trained with all 535 classes, but is better than than the median accuracy (78.5%) and the max accuracy (82.5%) attained using

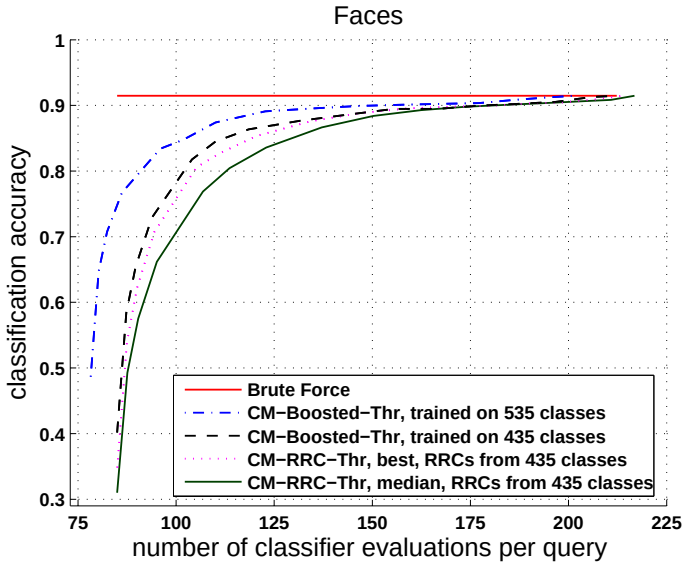

Figure 6·4: Measuring generalization of ClassMap embeddings to patterns from classes not available during AdaBoost training. We show results from an embedding optimized on all 535 classes, and an embedding optimized on only 435 classes, leaving out 100 classes. We also show results from the best performing out of 100 CM-RRC embeddings (with reference classifiers chosen from the 435 classes), and the median result from those 100 embeddings. The test set was the set of test patterns from the left-out 100 classes. Each test pattern was classified against all 535 classes.

100 randomly constructed embeddings. This experiment shows that ClassMap could be applicable in a dynamic setting, where new classes and classifiers can be inserted after the embedding was trained, without requiring a new embedding to be constructed.

6.2.2 Summary of ClassMap results

In summary, on the hand and face dataset, ClassMap yielded 3 to 28 times faster classification compared to brute force, with negligible or no loss in accuracy, in the presence of large numbers of classes (2,430 and 535 classes respectively). On the face dataset, ClassMap also worked well when applied to query patterns from classes that were not available during embedding construction.

6.3 Experimental Evaluation of the OVA-VS Method

In contrast to ClassMap, the OVA-VS method can only be applied if the OVA classifiers have been trained via boosting. For that purpose, we jointly trained boosted OVA classifiers using the shared features approach of Torralba et al. (Torralba et al., 2007). When applied to the faces dataset of 535 classes and the enhanced hands dataset of 48600 classes, this method produced a number of 10,000 and respectively 3,000 unique weak classifiers appearing in the OVA classifiers.

For evaluation we used a test set of 281 synthetic hand images (also generated using Poser) and a test set of 300 face images. Naturally, the sets of test images were disjoint from the sets of training images. We evaluated seven different methods: brute-force search, ClassMap, and five variants of OVA-VS; each of those five variants uses a different approach for the filtering step. In more detail, the seven methods we tried are:

- **Brute-force search:** application of all OVA classifiers to the query.
- **ClassMap (CM-Boosted version).** For comparison purposes, we ran ClassMap on top of the boosted OVA classifiers which we used as basis for the different variants of the OVA-VSmethod. We note that the results provided here differ from those of

Section 6.2.1, as the results in Section 6.2.1 were applied on top of OVA classifiers trained using SVMs, and not using boosting.

- **OVA-VS-PCA:** Use PCA within the filter-and-refine framework, as described in Section 5.2.
- **OVA-VS-Sampling-1:** Use the Sampling-1 method within the filter-and-refine framework, as described in Section 5.2.
- **OVA-VS-Sampling-2:** Use the Sampling-2 method within the filter-and-refine framework, as described in Section 5.2.
- **OVA-VS-LSH:** locality sensitive hashing applied to the 3,000-dimensional space defined by the weak classifiers.
- **OVA-VS-PCA+LSH**: Use PCA to reduce dimensionality from 3,000 to 20 dimensions, normalize to one, and then use LSH to index the 20-dimensional unit hypersphere.

Performance is measured in terms of speed-up with respect to brute force, and classification accuracy. By definition, brute force achieves a speed-up factor of 1. The classification accuracy of brute force is 90.75% for the synthetic hands dataset and 87.0% for the faces dataset. For the PCA, Sampling-1, and Sampling-2 methods, the parameters that need to be chosen are d', i.e., the dimensionality of the lower-dimensional space, and p, i.e., the number of OVA classifiers to be evaluated at the refine step. Naturally, classification accuracy and running time depend on both p and d'.

We now proceed to describe the experimental results obtained by these methods for the synthetic hands dataset and the FRGC dataset.

6.3.1 Results on the Synthetic Hands Dataset

Figure 6·12 compares the performance of PCA and CM-Boosted on the synthetic hands dataset. PCA gave significantly better overall results than ClassMap. To illustrate that,

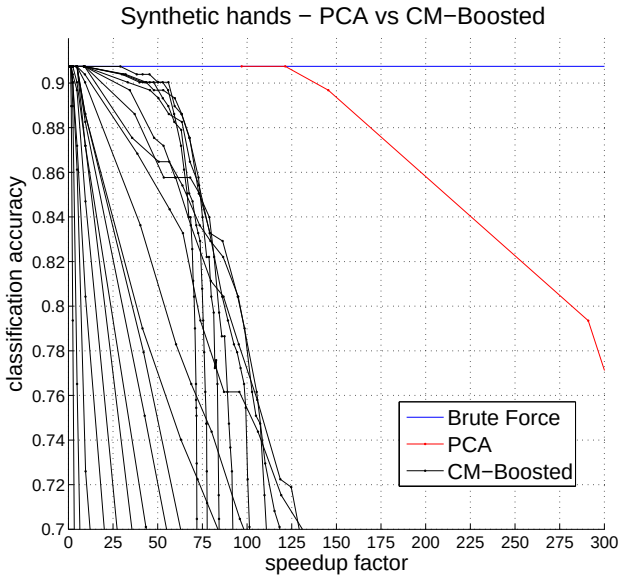

Figure 6·5: Classification accuracy vs. speedup factor on the synthetic hands dataset, for brute force, PCA, and CM-Boosted. All three methods were applied on top of boosted OVA classifiers.

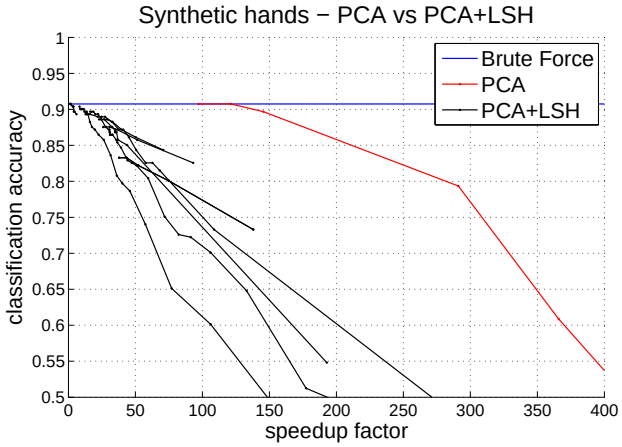

Figure 6·6: Classification accuracy vs. speedup factor on the synthetic hands dataset, for brute force, PCA, and PCA+LSH. All three methods were applied on top of boosted OVA classifiers.

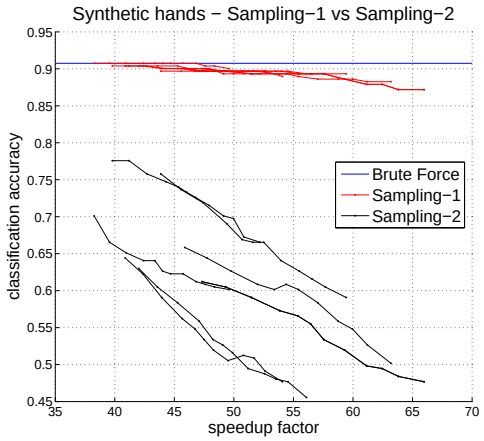

Figure 6·7: Classification accuracy vs. speedup factor on the synthetic hands dataset, for brute force, Sampling-1, and Sampling-2. All three methods were applied on top of boosted OVA classifiers.

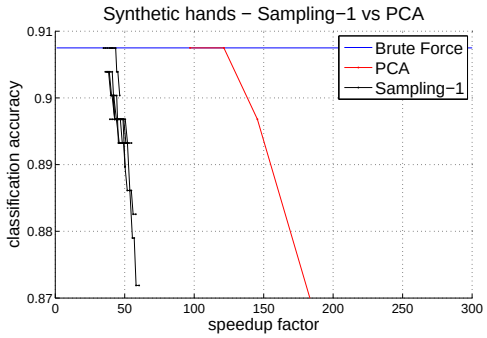

Figure 6·8: Classification accuracy vs. speedup factor on the synthetic hands dataset, for brute force, Sampling-1, and PCA. All three methods were applied on top of boosted OVA classifiers.

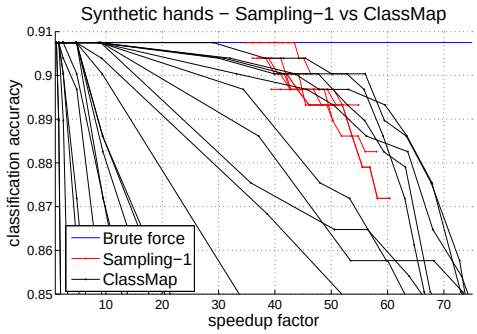

Figure 6·9: Classification accuracy vs. speedup factor on the synthetic hands dataset, for brute force, Sampling-1, and CM-Boosted. All three methods were applied on top of boosted OVA classifiers.

we plotted a single performance curve for PCA, obtained by constraining d and p so that the running time of the filter and the refine step were equal. In contrast, for CM-Boosted, we plotted a family of curves, each curve corresponding to a different embedding dimensionality, ranging from 1 to 90 dimensions. The single PCA curve corresponds to much better accuracy vs. efficiency trade-offs than any of the results obtained using different d' and p parameters for CM-Boosted. As a highlight, PCA gave a speed-up factor of 120 over brute-force search for a classification accuracy of 90.75% (equal to that of brute-force search), whereas ClassMap gave a speed-up factor of 29 for that accuracy. PCA yielded this result for $d' = 12$ and $p = 194$.

LSH did not work as well for this dataset, both by itself or in conjunction with PCA. Figure 6·6 illustrates that the performance of PCA+LSH was much worse than that of using just PCA. These results for PCA+LSH correspond to a 20-dimensional PCA projection, LSH parameter k ranging from 8 to 24 and LSH parameter l going up to 6000. We note that larger values of l place a heavy burden on memory, as the amount of memory needed for LSH is $O(ln)$ is the number of classes. Using $l = 10000$ required about 5GB of memory. LSH by itself performed even worse: for example, the best speed-up obtained for an accuracy of 80% was a factor of 2.65, and even for an accuracy of 64% the best obtained speed-up was a factor of 5.

Figures 6·7, 6·8, and 6·9 illustrate the performance of sampling-based dimensionality reduction, using variants Sampling-1 and Sampling-2. For each of these variants, multiple curves are plotted, each curve corresponding to a different number d' of sampled dimensions, ranging from 20 to 40. We note on Figure 6·7 that Sampling-1 worked significantly better than Sampling-2, as expected, since Sampling-1 selects the most informative dimensions, whereas Sampling-2 selects dimensions randomly. As Figure 6·8 shows, PCA produced much better results than Sampling-1 (and consequently better results than Sampling-2 as well).

In Figure 6·9 we observe that the best speedup obtained by Sampling-1 for the highest accuracy setting of 90.75% was a factor of 43, whereas ClassMap, for that accuracy,

gave a speedup factor of 29. This is an interesting result, considering the simplicity of the Sampling-1 method vs. the complexity of implementing ClassMap. Using the vector representation proposed by OVA-VS allowed us to use the inherent structure of that vector space to our advantage, whereas ClassMap, due to its more general formulation, did not have access to this vector representation.

In summary, PCA gave the best results for the synthetic hands dataset, highlighted by a speedup factor of 120 with no loss in classification accuracy over brute-force search. There was no clear winner between Sampling-1 and ClassMap. Sampling-1 outperformed Sampling-2, as expected. PCA+LSH performed rather poorly, and LSH on the original space performed even more poorly.

6.3.2 Results on the FRGC Dataset

While the PCA variant of OVA-VS gave good results for the synthetic hands dataset, that was not the case for the FRGC dataset. Figure 6·10 plots the results attained using several different combinations of d' and p for PCA and ClassMap. We note that for an accuracy of 83.5%, which is 3.5% lower than that of brute-force search, PCA achieved a speedup factor of only 1.6. As seen on the same figure, ClassMap also did not work very well, achieving a speedup of 3.3 for the same accuracy of 83.5%, and a speedup of 1.8 for an accuracy of 87%, which is equal to the accuracy of brute-force search.

One reason for the poor performance of both PCA and ClassMap is the relatively small number of classes in this dataset, only 535, compared to the 48,600 classes of the synthetic hands dataset. As a result, generating a single dimension of a PCA projection, or a single dimension of a ClassMap embedding, are operations that incur 1/535 of the cost of brute-force search, compared to 1/48600 for the hands dataset. Figure 6·11 plots classification accuracy vs. filter-and-refine cost, omitting from that cost the cost of generating PCA projections and ClassMap embeddings for the queries. This cost is representative of the performance we could expect if we had a much larger number of classes, that would make the projection and embedding costs negligible. In that case we see that PCA performs

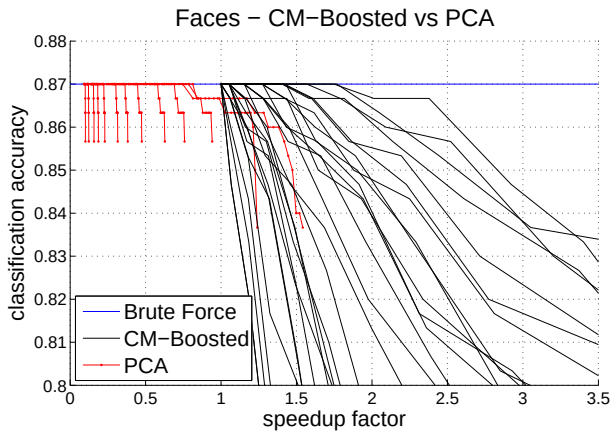

Figure 6·10: Classification accuracy vs. speedup factor on the face dataset, for brute force, CM-Boost, and PCA. Both methods were applied on top of boosted OVA classifiers.

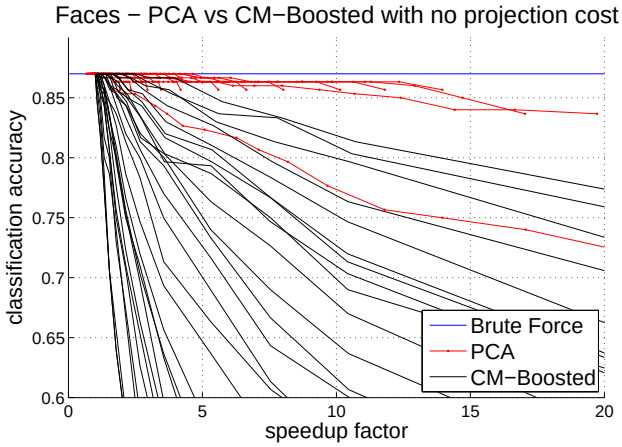

Figure 6·11: Classification accuracy vs. speedup factor on the face dataset, for brute force, PCA, and CM-Boosted, ignoring the cost of computing the PCA projection on the query, and also ignoring the cost of computing the CM-Boosted embedding of the query. Both methods were applied on top of boosted OVA classifiers.

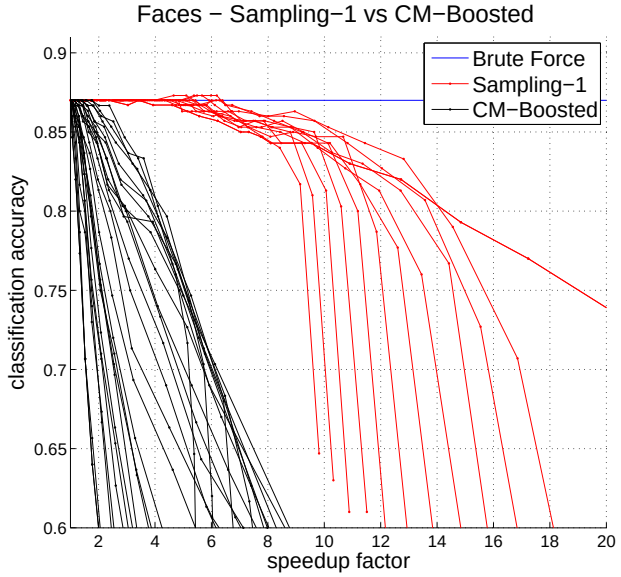

Figure 6·12: Classification accuracy vs. speedup factor on the face dataset, for brute force, Sampling-1, and CM-Boosted. All three methods were applied on top of boosted OVA classifiers.

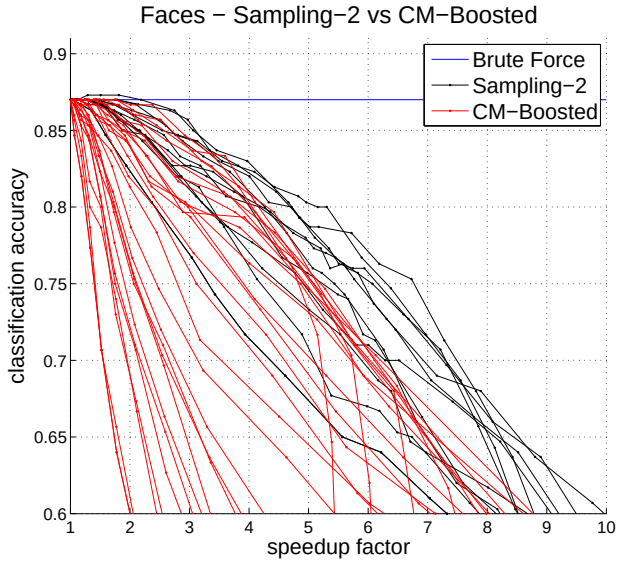

Figure 6·13: Classification accuracy vs. speedup factor on the face dataset, for brute force, Sampling-2, and CM-Boosted. All three methods were applied on top of boosted OVA classifiers.

better than ClassMap, obtaining, for example, a speedup factor of 12.3 for a classification accuracy of 86.3%. For that same accuracy, ClassMap gives a speedup factor of 4.2%.

While the cost of computing the lower-dimensional projection is significant for PCA for this dataset, the sampling methods Sampling-1 and Sampling-2 do not incur such a cost. As a result, Sampling-1 produced the best results out of the seven methods evaluated on this dataset. Figure 6·12 shows the performance of Sampling-1 compared to ClassMap. Sampling-1 produced a speedup of 6.4 for the same accuracy as brute-force search, compared to a speedup of 1.8 for ClassMap and for the same accuracy. For an accuracy of 85.5% (compared to 87% for brute-force search), Sampling-1 produced a speedup of 10.2, which is an order of magnitude.

As expected, Sampling-2 performed worse than Sampling-1. At the same time, the performance of Sampling-2 was actually slightly better than that of ClassMap, as shown in Figure 6·13.

LSH performed poorly for this dataset, as it did for the hands dataset. For example, applied on top of a 20-dimensional PCA projection, and for an accuracy of 81%, the speedup attained using PCA is only a factor of 1.55.

In summary, Sampling-1 gave the best results for the FRGC dataset, producing a speedup factor of 6.4 while achieving the accuracy of brute-force search. Sampling-1 and ClassMap gave similar performance, while PCA and LSH performed poorly. The relatively small number of classes in this dataset makes it hard to obtain good results for the PCA and ClassMap methods.

6.3.3 Summary of OVA-VS results

In summary, our experiments with the different variants of the OVA-VS method show that, in each dataset, one of these four variants, significantly outperformed ClassMap. For the hands dataset, for an accuracy equal to that of brute-force search, the PCA variant of OVA-VS achieved a speedup factor of 120, compared with a speedup factor of 29 for ClassMap. For the FRGC dataset, again for an accuracy equal to that of brute-force

search, the Sampling-1 variant of OVA-VS achieved a speedup factor of 6.4, compared with a speedup factor of 1.8 for ClassMap. These results are expected, as the vector representation proposed by OVA-VS does not lose any information, whereas ClassMap embeddings are lossy.

With respect to the PCA variant of OVA-VS, we have seen that the relatively poor performance of that variant on the FRGC dataset was due to the relatively small number of classes, which made the PCA projection cost a large fraction of the cost of brute-force search. As the number of classes increase, we expect PCA performance to improve.

With respect to the LSH and PCA+LSH variants, we have seen that the results of those variants on our datasets were rather mediocre. At the same time, there are recent methods for improving the performance of LSH (Andoni and Indyk, 2006; Panigrahy, 2006), that we have not implemented. In future work, we plan to evaluate the performance of those methods when integrated into the LSH variant of OVA-VS. Also, we should note that the task of LSH in the PCA+LSH variant is rather more challenging than the typical task of LSH, which is to find a few nearest neighbors of the query. For example, in the hands dataset, for a 20-dimensional embedding, the true nearest neighbor of some queries (i.e., the vector representation of the strongest responding classifier for those queries) is not included in the 100 nearest neighbors after we project to the PCA space. In such cases, the task of LSH is to retrieve a not-so-near neighbor of the query, and this leads to worse performance.

Chapter 7

Discussion and Future work

We have described two methods, ClassMap and OVA-VS, for speeding up large margin classification in the presence of a large number of classes. The key differences between the two methods can be summarized as follows:

- **Generality.** ClassMap does not make assumptions about the large margin method used for training and the binary decomposition scheme. OVA-VS is designed for application on top of boosted OVA classifiers. On the other hand, ClassMap makes the relatively strong assumption that the outputs of an OVA classifier on patterns of any specific class follow a unimodal distribution. No such assumption is made for OVA-VS.

- **Training data.** ClassMap requires additional training data, on top of the training data used to train the classifiers. OVA-VS requires no additional training data.

- **Cost of mapping.** For ClassMap, the embedding of each test pattern is relatively costly, as it involves applying the reference classifiers on the pattern. For OVA-VS, the mapping of each pattern to a vector is automatic, obtained immediately based on the responses of the weak classifiers on that pattern.

- **Quality of mapping.** The ClassMap embedding is lossy: the nearest class in the embedding space is not necessarily the class that is computed using brute-force classification. On the other hand, the mapping used in OVA-VS is lossless: the vector representation is such that the strongest-responding OVA classifier (that would be identified using brute force) is the nearest neighbor of the pattern in the vector space.

Our experiments with hand recognition and face recognition systems showed that each method achieves significant speed-ups, that are over one order of magnitude in some cases, with relatively small losses in classification accuracy.

In the case where the OVA classifiers are not trained using boosting, OVA-VS is not applicable, but ClassMap is applicable and has been shown in the experiments to be a significantly better alternative than brute-force search. When boosted OVA classifiers are used, then the OVA-VS method becomes applicable, and we can take advantage of the lossless vector representation proposed by that method. As shown in the experiments, in both datasets a variant of OVA-VS produced results significantly better than those of ClassMap. On the hands dataset, the PCA variant gave the best results. On the FRGC dataset, while PCA did not perform as well due to the relatively small number of classes, the Sampling-1 variant produced the best results, and significantly outperformed ClassMap.

With respect to the performance of PCA and Sampling-1 variants of the OVA-VS method, we note that computing the PCA projection of a query requires time $O(dd')$, where d is the number of weak classifiers (and thus the dimensionality of the vector representation of classifiers and patterns) and d' is the dimensionality of the lower dimensional space obtained using PCA or Sampling-1. This cost of $O(dd')$ is independent of the number of classes. As the number of classes becomes smaller, the $O(dd')$ cost becomes a larger fraction of the overall running time of OVA-VS. The Sampling-1 method does not incur this $O(dd')$ cost and thus becomes more attractive computationally, compared to PCA, as the number of classes decreases. This difference of $O(dd')$ in the running time contributed to the fact that Sampling-1 outperformed PCA on the FRGC dataset, where there are only 535 classes, whereas PCA outperformed Sampling-1 on the hands dataset, where the number of classes is 48,600.

For ClassMap, the filter step compares the embedding of the query to the embeddings of all classes. This step takes negligible time in our experiments, but can become a bottleneck for really large numbers of classes. However, by mapping classes to a vector space, ClassMap allows application of numerous vector indexing methods (e.g., LSH (Gionis et al.,

1999)) for speeding up the filter step. Using vector indexing methods can lead to recognition time that is sublinear in the number of classes, thus allowing efficient recognition even with significantly more classes than the numbers used in our experiments. Integrating such indexing methods with ClassMap is an interesting topic for future work.

Another possible improvement for ClassMap is to explicitly model the probability distribution of the responses of reference classifiers for the patterns of each class. At the filter step, instead of measuring L_1 distances between the embedding of the query and the embeddings of the database classes, we could compute the probability of the pattern belonging to each class, based on the responses of the reference classifiers. A challenge in implementing such a probabilistic approach is handling classes with a small number of training examples.

For OVA-VS, an interesting topic for future exploration is to try a larger number of vector indexing methods, in order to identify methods that tend to work well in practice within the proposed framework. For the LSH variant of OVA-VS, we plan to evaluate recently proposed methods for improving LSH performance (Andoni and Indyk, 2006; Panigrahy, 2006). Finally, as OVA-VS is only applicable to boosting-based classification, it will be interesting to investigate whether similar methods can also be designed for other types of large margin classifiers, such as support vector machines.

It will also be interesting to integrate the hand recognition module in an actual application, such as ASL sign recognition (Athitsos et al., 2008), to obtain a better picture of how domain-specific constraints affect hand pose recognition accuracy.

References

Agarwal, A. and Triggs, B. (2006). Recovering 3D human pose from monocular images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(1):44–58.

Allwein, E. L., Schapire, R. E., and Singer, Y. (2000). Reducing multiclass to binary: a unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141.

Andoni, A. and Indyk, P. (2006). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 459–468.

Athitsos, V. (2006). *Learning Embeddings for Indexing, Retrieval, and Classification, with Applications to Object and Shape Recognition in Image Databases*. PhD thesis, Boston University.

Athitsos, V., Neidle, C., Sclaroff, S., Nash, J., Yuan, A. S. Q., and Thangali, A. (2008). The American Sign Language lexicon video dataset. In *IEEE Workshop on Computer Vision and Pattern Recognition for Human Communicative Behavior Analysis (CVPR4HB)*.

Athitsos, V., Stefan, A., Yuan, Q., and Sclaroff, S. (2007). ClassMap: Efficient multiclass recognition via embeddings. In *IEEE International Conference on Computer Vision (ICCV)*.

Böhm, C., Berchtold, S., and Keim, D. A. (2001). Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373.

Curious Labs (2002). *Poser 5 Reference Manual*. Curious Labs, Santa Cruz, CA.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 886–893.

de Campos, T. E. and Murray, D. W. (2006). Regression-based hand pose estimation from multiple cameras. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 782–789.

Dietterich, T. G. and Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. Wiley-Interscience.

Ferhatosmanoglu, H., Tuncel, E., Agrawal, D., and Abbadi, A. E. (2001). Approximate nearest neighbor searching in multimedia databases. In *IEEE International Conference on Data Engineearing (ICDE)*, pages 503–511.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28(2):337–374.

Gavrila, D. and Philomin, V. (2001). Real-time object detection for "smart" vehicles. In *IEEE International Conference on Computer Vision (ICCV)*, pages 87–93.

Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity search in high dimensions via hashing. In *International Conference on Very Large Databases (VLDB)*, pages 518–529.

Grauman, K. and Darrell, T. J. (2004). Fast contour matching using approximate earth mover's distance. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages I: 220–227.

Hjaltason, G. and Samet, H. (2003a). Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 25(5):530–549.

Hjaltason, G. R. and Samet, H. (2003b). Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems (TODS)*, 28(4):517–580.

Jolliffe, I. (1986). *Principal Component Analysis*. Springer-Verlag.

Kanth, K. V. R., Agrawal, D., and Singh, A. (1998). Dimensionality reduction for similarity searching in dynamic databases. In *ACM International Conference on Management of Data (SIGMOD)*, pages 166–176.

Li, C., Chang, E., Garcia-Molina, H., and Wiederhold, G. (2002). Clustering for approximate similarity search in high-dimensional spaces. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 14(4):792–808.

Li, S. Z. and Zhang, Z. Q. (2004). Floatboost learning and statistical face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(9):1112–1123.

Liu, C. (2006). Capitalize on dimensionality increasing techniques for improving face recognition grand challenge performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(5).

Ong, E. J. and Bowden, R. (2004). A boosted classifier tree for hand shape detection. In *Automatic Face and Gesture Recognition (AFGR)*, pages 889–894.

Panigrahy, R. (2006). Entropy based nearest neighbor search in high dimensions. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1186–1195.

Phillips, P. J., Flynn, P. J., Scruggs, T., Bowyer, K. W., Chang, J., Hoffman, K., Marques, J., Min, J., and Worek, W. (2005). Overview of the face recognition grand challenge. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 947–954.

Platt, J., Cristianini, N., and Shawe-Taylor, J. (2000). Large margin DAGS for multiclass classification. In *Neural Information Processing Systems (NIPS)*, pages 547–553.

Rifkin, R. and Klautau, A. (2004). In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141.

Sakurai, Y., Yoshikawa, M., Uemura, S., and Kojima, H. (2000). The A-tree: An index structure for high-dimensional spaces using relative approximation. In *International Conference on Very Large Data Bases (VLDB)*, pages 516–526.

Schapire, R. and Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336.

Shakhnarovich, G., Viola, P., and Darrell, T. (2003). Fast pose estimation with parameter-sensitive hashing. In *IEEE International Conference on Computer Vision (ICCV)*, pages 750–757.

Stenger, B., Thayananthan, A., Torr, P. H. S., and Cipolla, R. (2004). Hand pose estimation using hierarchical detection. In *ECCV Workshop on Human Computer Interaction*, pages 105–116.

Torralba, A., Murphy, K. P., and Freeman, W. T. (2007). Sharing visual features for multiclass and multiview object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 29(5):854–869.

Tuncel, E., Ferhatosmanoglu, H., and Rose, K. (2002). VQ-index: An index structure for similarity searching in multimedia databases. In *Proceedings of ACM Multimedia*, pages 543–552.

Vapnik, V. (1995). *The nature of statistical learning theory.* Springer-Verlag New York, Inc.

Weber, R. and Böhm, K. (2000). Trading quality for time with nearest-neighbor search. In *International Conference on Extending Database Technology (EDBT)*, pages 21–35.

Weber, R., Schek, H.-J., and Blott, S. (1998). A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *International Conference on Very Large Data Bases (VLDB)*, pages 194–205.