To appear in Proceedings of IEEE International Conference on Computer Vision (ICCV), October 2007.

ClassMap: Efficient Multiclass Recognition via Embeddings

Vassilis Athitsos[1], Alexandra Stefan[2], Quan Yuan[2], and Stan Sclaroff[2]
[1] Computer Science and Engineering Department, University of Texas at Arlington, USA
[2] Computer Science Department, Boston University, USA

Abstract

In many computer vision applications, such as face recognition and hand pose estimation, we need systems that can recognize a very large number of classes. Large margin classification methods, such as AdaBoost and SVMs, often provide competitive accuracy rates, but at the cost of evaluating a large number of binary classifiers. We propose an embedding-based method for efficient multiclass recognition. In our method, patterns and classes are mapped to vectors in such a way that patterns and their associated classes tend to get mapped close to each other. This way, given a test pattern, a small set of candidate classes can be identified efficiently using simple vector comparisons. In experiments with 3D hand pose recognition (2430 classes) and face recognition (535 classes), our method is between 3 and 28 times faster compared to evaluating all binary classifiers, with negligible or no loss in classification accuracy.

Figure 1. Example recognition tasks with a large number of classes. Top: recognizing hand shape 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.

1. Introduction

A fundamental computer vision and pattern recognition problem is efficient and accurate recognition of a very large number of classes. Real-world vision systems should eventually be able to recognize thousands of different objects, thousands of different signs of a sign language, thousands or millions of human body or hand poses, and faces of thousands or millions of individuals. Training an individual one-vs.-all (OVA) classifier for each class is becoming a common approach, and is a natural way to apply theoretically and empirically sound large margin methods, such as AdaBoost [20] and SVMs [24], to a multiclass setting [2]. The question we address in this paper is the following: assuming that we have trained all these individual classifiers, how can we use them efficiently?

The standard way such individual OVA classifiers are used today is essentially brute-force search: given a pattern to classify, all classifiers are applied so as to find the classifier producing the strongest response. This strategy can quickly become impractical as the number of classes becomes large. To achieve a scalable solution, we propose a novel paradigm, 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.

This work was supported by NSF grants IIS 0308213, IIS 0329009, and CNS 0202067.

ClassMap, the method proposed in this paper, 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.

In general, the proposed method is designed for use in domains where the task of multiclass recognition is decomposed into multiple binary classification problems. In addition to the one-vs.-all (OVA) scheme, where a classifier is trained for each class, our method can also be used with the all-pairs scheme, where a classifier is trained for each pair of classes, or with more general error-correcting output codes [2, 8]. Our method 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 binary 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 computationally light 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 our method on two datasets, illustrated in Fig. 1: a dataset of hand images for which we want to recognize the hand shape and 3D orientation (out of 2430 classes), and a dataset of face images where we want 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, our method is between 3 and 28 times faster, with negligible or no loss in classification accuracy.

2. Related Work

Nearest neighbor classification [9] is a simple method for multiclass recognition, and has been successfully applied in large multiclass problems, such as face recognition (e.g., [16]) and articulated pose estimation (e.g., [21]). Theoretically, k-nearest neighbor classification accuracy becomes optimal as the number of training data approaches infinity [9]. 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 [10, 15, 20] or support vector machines (SVMs) [24]. 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 [2].

Different types of multiclass-to-binary decompositions can be defined using error-correcting output codes [2, 8]. The most commonly used decompositions are into all-pairs problems, where a classifier is trained to discriminate between each pair of classes, or into one-vs.-all (OVA) problems, where, for each class, an OVA classifier is trained to discriminate between that class and all other classes. To classify a query, all binary classifiers are applied. An exception is the DAGSVM method [19], 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. Feature sharing for boosted classifiers [23] can significantly reduce the feature extraction cost, but does not reduce the number of required classifier evaluations.

A variety of indexing methods can be used to speed up nearest neighbor retrieval and classification [3, 4, 12, 13, 14, 21], often achieving significant speedups over brute-force search [3, 21]. However, for large margin methods, brute-force evaluation of a large number of classifiers is the current state of the art. ClassMap, the method proposed in this paper, is designed to provide an efficient alternative to the brute-force method for large margin multiclass recognition.

Our method performs, 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. However, classical nearest neighbor indexing methods [3, 4, 12, 13, 14] are inapplicable to our setting, because of two issues: 1). In our problem, 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 this paper 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 [17] by combining hierarchical classifiers into a tree structure. Hierarchical template matching has been used for pedestrian detection [11] and hand pose estimation [22]. Articulated pose can be treated as a multidimensional regression problem, and estimators can be trained that map observations into a continuous pose space [1, 7]. However, many domains (e.g., face recognition) do not lend themselves readily either to hierarchical decomposition or to regression-based estimation. Our method, on the other hand, is readily applicable in such domains, as long as a finite set of classes can be defined.

3. Problem Definition and Overview

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*.

We assume that we have chosen a decomposition of the multiclass recognition problem into a family of binary problems. While our method can be applied with more general decompositions, for brevity we only consider one-vs.-all (OVA) and all-pairs decompositions. In the OVA 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. Higher (more positive) responses $C_Y(Q)$ indicate higher confidence that $L(Q) = Y$. To classify a query $Q \in \mathbb{X}$, the standard approach (which we want to speed up using ClassMap) 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.

In the all-pairs scheme, for each pair of classes (Y_1, Y_2) a classifier $C_{Y_1,Y_2} : \mathbb{X} \rightarrow \mathbb{R}$ is trained to discriminate between class Y_1 and class Y_2. To classify a query $Q \in \mathbb{X}$, the standard approach (which, again, we want to speed up using ClassMap) is to: 1). compute $C_{Y_1,Y_2}(Q)$ for all pairs (Y_1, Y_2), and 2). determine the final classification output using a voting scheme, where each C_{Y_1,Y_2} votes for Y_1 and/or Y_2, with a weight that depends on the response $C_{Y_1,Y_2}(Q)$. Different voting schemes can be used [2].

Let $\mathbb{C}$ be the set of all binary classifiers corresponding to the decomposition scheme we have chosen. For example, $\mathbb{C}$ can be a set of all-pairs or OVA classifiers. 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.

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. 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 $\mathbb{X}_{\rm tr} \subset \mathbb{X}$ of labeled training examples, and a matrix M storing precomputed outputs $C(X)$ for all $C \in \mathbb{C}$ and $X \in \mathbb{X}_{\rm tr}$.

Under these assumptions, the only way to obtain information about a query $Q \in \mathbb{X}$ is to evaluate $C(Q)$ for some classifiers $C \in \mathbb{C}$. Our goal is to classify Q while evaluating $C(Q)$ for as few classifiers C as possible. The question then becomes: if we only compute $C(Q)$ for a relatively small number of classifiers, how can we use that information to quickly generate a short list of candidate classes for Q? Alternatively, given a class $Y \in \mathbb{Y}$, how can we obtain a quick evaluation of whether Y is a candidate class for Q? The answer is that we can compare how well each computed output $C(Q)$ matches a *typical output* of C on training examples from class Y.

More formally, 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*:

$$F^R(Q) = R(Q) \,. \tag{1}$$

$$F^R(Y) = \text{median}\{R(X) : X \in \mathbb{X}_{\rm tr}, L(X) = Y\} \,. \tag{2}$$

We should note that reference classifiers play a role analogous to that of *reference objects* in Lipschitz embeddings [13]. 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. We treat this median as a typical output of R on patterns from 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.

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

$$F(Q) = (F^{R_1}(Q), \ldots, F^{R_d}(Q)) \,. \tag{3}$$

$$F(Y) = (F^{R_1}(Y), \ldots, F^{R_d}(Y)) \,. \tag{4}$$

We use the term *ClassMap embeddings* for embeddings F defined this way. Note that any OVA or all-pairs 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 $\mathbb{C}$.

An example of a simple ClassMap embedding, computed with real data (faces from the 535-class FRGC2 dataset [18]) is shown on Fig. 2. We illustrate a 2D embedding F defined using two OVA classifiers C_{Y_3} and C_{Y_4}

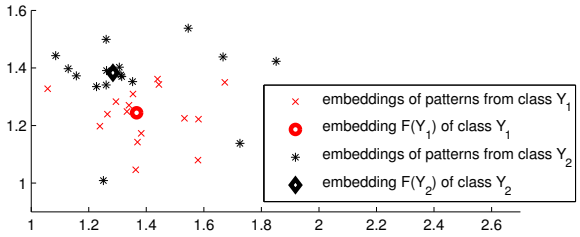

Figure 2. A 2D embedding $F = (F^{R_1}, F^{R_2})$, defined using Eqs. 3 and 4 with two reference classifiers R_1 and R_2, and mapping both patterns and classes into $\mathbb{R}^2$. We show the mappings of patterns belonging to classes Y_1 and Y_2, and the mappings of classes Y_1 and Y_2 themselves. Note that the mapping of each class is 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 are mapped closer to the embedding of their own class than to the embedding of the other class.

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. 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 [12], can be used to speed up nearest neighbor search in that space.

5. 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.

Our method takes as input the following data:

- A set $\mathbb{C}$ of large margin classifiers $C : \mathbb{X} \to \mathbb{R}$, corresponding to some multiclass-to-binary decomposition.
- 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 3 and 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 our method, i.e., classification of previously unseen query patterns. For the runtime phase we adapt the filter-and-refine framework [13]. Given a query $Q \in \mathbb{X}$, we perform the following steps:

- **Embedding step**: compute $F(Q)$.
- **Filter step**: rank all classes Y by the distance of their embeddings $F(Y)$ from $F(Q)$.
- **Refine step, version 1 (all-pairs)**: for some user-specified number p, compute all responses $C_{Y_1,Y_2}(Q)$ such that classes Y_1 and Y_2 were ranked in the top p classes by the filter step. Determine the class label for Q by voting according to those responses [2].
- **Refine step, version 2 (simple OVA)**: 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.
- **Refine step, version 3 (OVA + threshold)**: given user-specified p and t: 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 three versions provided above, several other versions may be reasonable choices for specific domains. Our focus in this paper 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.

6. 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 5, 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:

$$\tilde{F}(Q, Y_1, Y_2) = \Delta(F(Q), F(Y_2)) - \Delta(F(Q), F(Y_1)) \; . \tag{5}$$

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 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*.

Every $R \in \mathbb{C}$ can be used to define a 1D embedding F^R. 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. 2). 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 [3], 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 5: a set $\mathbb{C}$ of large margin classifiers, a set $\mathbb{X}_{\rm tr} \subset \mathbb{X}$ of labeled training examples, and a matrix M of classifier outputs $C(X)$ for each pair $(C, X) : C \in \mathbb{C}, X \in \mathbb{X}_{\rm tr}$.

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 $X \in \mathbb{X}_{\rm tr}, Y \in \mathbb{Y} - \{L(X)\}$.

4. We run AdaBoost [20] 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:

$$H = \sum_{i=1}^{d} (\alpha_i \tilde{F}^{R_i}) \; , \tag{6}$$

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

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

$$F_{\rm out}(Q) = (F^{R_1}(Q), ..., F^{R_d}(Q)) \; . \tag{7}$$

$$F_{\rm out}(Y) = (F^{R_1}(Y), ..., F^{R_d}(Y)) \; . \tag{8}$$

$$\Delta((u_1, ..., u_d), (v_1, ..., v_d)) = \sum_{i=1}^{d} (\alpha_i |u_i - v_i|) \; . \tag{9}$$

Eqs. 7 and 8 use the definition of F^R given in Eqs. 1 and 2. In Eq. 9, $(u_1, ..., u_d)$ and $(v_1, ..., v_d)$ are d-dimensional vectors that $F_{\rm out}$ maps patterns and classes to.

Following the proof in [3] for AdaBoost-trained embeddings, it holds that $H = \tilde{F}_{\rm out}$. In other words, the classifier H trained via AdaBoost misclassifies a triple $(Q, L(Q), Y \neq L(Q))$ iff, under distance Δ, $F_{\rm out}$ maps Q closer to Y than to $L(Q)$. Therefore, AdaBoost directly optimizes the error rate of $\tilde{F}_{\rm out}$, which is exactly the measure we wanted to optimize for the purposes of using $F_{\rm out}$ for filter-and-refine classification.

7. Experiments

We evaluate our method on two datasets, shown in Fig. 1: a dataset of hand images, where we want to estimate the handshape and the 3D orientation, and the Face Recognition Grand Challenge (FRGC) Version 2 dataset [18].

7.1. Datasets

7.1.1 The Hand Dataset

This dataset contains hand images of 81 basic hand shapes defined in American Sign Language (ASL). There are 30 different 3D orientations for each shape, for a total of $81 \times 30 = 2430$ hand pose classes.

Using Poser 5 [5], we generated 200 synthetic images for each class: 150 for training OVA classifiers, 25 for use as training during embedding construction via AdaBoost, and 25 for testing. For each synthetic hand image, cluttered background from random real images was added to

the regions outside the hand silhouette. 2430 OVA classifiers were trained using SVMs with a linear kernel. Histogram of Oriented Gradient features [6] were used. All histogram bins were stored as a 2025-dimensional vector.

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 2430 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 2430 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 is equal to one of those four labels.

7.1.2 The Face Dataset

The face set contains all 2D face images in the Face Recognition Grand Challenge Version 2 (FRGC2) dataset [18]. There are 36817 face images from 535 subjects (i.e., 535 classes). These face images were partitioned into three disjoint sets: for each subject, half of the face images were used for training OVA classifiers, 1/4 were used for embedding construction, and 1/4 were used for testing. The features of face images were their projections on the top 2509 PCA components, which accounts for 99.9% of the variance. The 535 one-vs-all face classifiers were trained using SVMs with an RBF kernel.

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 is useful for illustrating how performance of our method is 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.

7.2. Methods and Parameters

In our experiments we evaluate five different methods: brute-force classification, and four different variations of ClassMap, as follows:

- **Brute force**: evaluate all OVA classifiers; select the class whose classifier produced the highest response.
- **CM-RRC**: use filter-and-refine classification, with version 2 of the refine step (see Section 5), and use a randomly generated ClassMap embedding.
- **CM-RRC-Thr**: use filter-and-refine classification, with version 3 (OVA+threshold) of the refine step, and use a randomly generated ClassMap embedding.
- **CM-Boosted**: use filter-and-refine classification, with version 2 of the refine step, and a ClassMap embedding constructed using AdaBoost.
- **CM-Boosted-Thr**: use filter-and-refine classification, with version 3 (OVA+threshold) of the refine step, and a ClassMap embedding constructed using AdaBoost.

The number of training triples used to train CM-Boosted embeddings, using the algorithm of Section 6, was 3 million for both the hand dataset and the face dataset. 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 3 of the refine step of Section 5, 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%.

7.3. Results

Performance is measured based on classification accuracy and efficiency. Fig. 3 displays 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 produce different accuracy-vs.-efficiency trade-offs, we vary parameter p of the refine step (Section 5), 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 are 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.

On the hand images, brute-force classification accuracy is 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 produces 23.9% classification accuracy for the real images and 95.3% for the synthetic images. Interestingly, for the real hand images, CM-Boosted is 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 are not associated with candidate classes considered during the refine step.

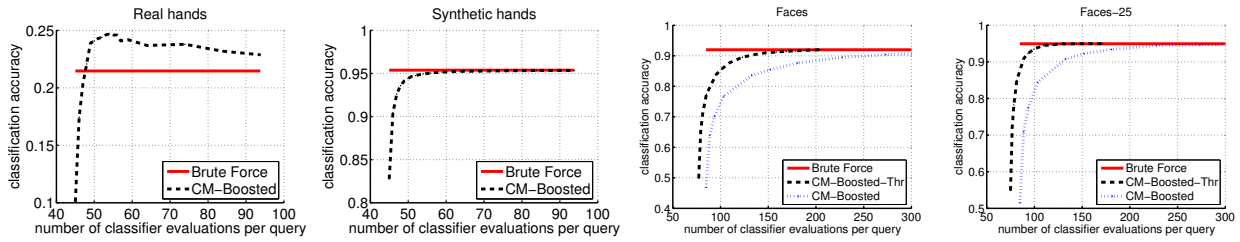

Figure 3. Results on the real and synthetic test sets of hand images, and on the faces and faces-25 test sets. For methods CM-Boosted and CM-Boosted-Thr we plot accuracy vs. number of OVA classifier evaluations per query. We also show as a solid horizontal line the brute force classification accuracy, attained at a cost of 2430 and 535 classifier evaluations per query respectively for hands and for faces.

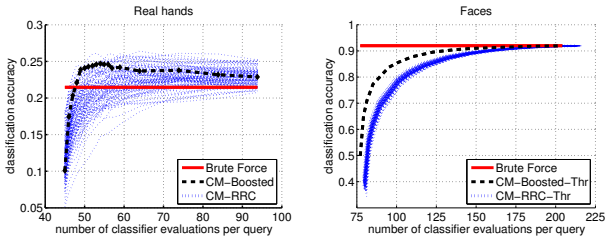

Figure 4. Comparing AdaBoost-based embedding construction to random embedding construction. For the real hand test set and the faces test set, we compare the single embedding constructed via AdaBoost vs. results from 100 randomly constructed embeddings.

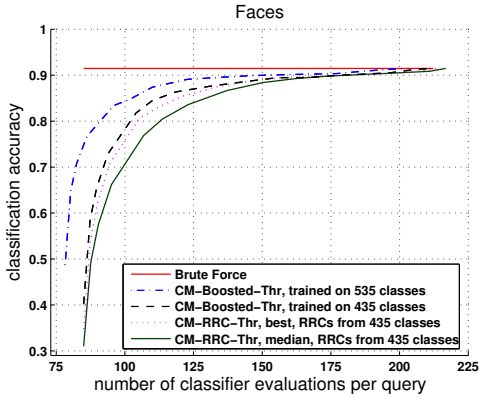

Figure 5. Measuring generalization of ClassMap embeddings to patterns from classes not available during AdaBoost training. We show results from an embedding optimized on all 535 classes, and an embedding optimized on only 435 classes, leaving out 100 classes. We also show results from the best performing 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 here is the set of test patterns from the left-out 100 classes. Each test pattern is classified against all 535 classes.

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 $d = 45, p = 40$) 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 our method can make a big difference in practice.

For the face dataset our method 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. We also note that the CM-Boosted-Thr method gives better results than the CM-Boosted method.

The results on the faces-25 dataset illustrate that our method achieves better accuracy/efficiency trade-offs for classes with 25 or more training examples. At a cost of 128 classifier evaluations per query, the CM-Boosted-Thr method yields 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. 4 compares the single embedding constructed via AdaBoost vs. results from 100 randomly constructed embeddings. Interestingly, for the real hand images, several random embeddings perform 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 perform better than brute force, achieving similar accuracy at more than twice the speed.

One last experiment measures the generalization ability of CM-Boosted embeddings. In this experiment, AdaBoost uses training examples from only 435 of the 535 classes of the face dataset, and the test set includes only examples from the left-out 100 classes. For each test example, all 535 classes are still considered as possible outputs. In Fig. 5 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 is 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 is 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 achieves 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 100 randomly constructed embeddings. This experiment shows that our method 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.

8. Discussion

We have presented a novel approach for speeding up recognition in the presence of a large number of classes. The key idea has been 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.

In experiments with datasets of hand and face images, and in the presence of large numbers of classes (2430 and 535 classes respectively), our method leads to 3 to 28 times faster classification compared to brute force, with negligible or no loss in accuracy. In our experiments, our method also works well on query patterns from classes that were not available during embedding construction.

In our implementation, 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 [12]) for speeding up the filter step. Using vector indexing methods can lead to recognition time that is sublinear to the number of classes, thus allowing efficient recognition even with significantly more classes than the numbers used in our experiments.

References

[1] A. Agarwal and B. Triggs. Recovering 3D human pose from monocular images. *PAMI*, 28(1):44–58, 2006.

[2] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *JMLR*, 1:113–141, 2000.

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

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

[5] Curious Labs, Santa Cruz, CA. *Poser 5 Reference Manual*, August 2002.

[6] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, pages 886–893, 2005.

[7] T. E. de Campos and D. W. Murray. Regression-based hand pose estimation from multiple cameras. In *CVPR*, volume 1, pages 782–789, 2006.

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

[9] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, 2001.

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

[11] D. Gavrila and V. Philomin. Real-time object detection for "smart" vehicles. In *ICCV*, pages 87–93, 2001.

[12] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[13] G. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *PAMI*, 25(5):530–549, 2003.

[14] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *TODS*, 28(4):517–580, 2003.

[15] S. Z. Li and Z. Q. Zhang. Floatboost learning and statistical face detection. *PAMI*, 26(9):1112–1123, 2004.

[16] C. Liu. Capitalize on dimensionality increasing techniques for improving face recognition grand challenge performance. *PAMI*, 28(5), 2006.

[17] E. J. Ong and R. Bowden. A boosted classifier tree for hand shape detection. In *AFGR*, pages 889–894, 2004.

[18] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek. Overview of the face recognition grand challenge. In *CVPR*, pages 947–954, 2005.

[19] J. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin DAGS for multiclass classification. In *NIPS*, pages 547–553, 2000.

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

[21] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. In *ICCV*, pages 750–757, 2003.

[22] B. Stenger, A. Thayananthan, P. H. S. Torr, and R. Cipolla. Hand pose estimation using hierarchical detection. In *ECCV Workshop on HCI*, pages 105–116, 2004.

[23] A. Torralba, K. P. Murphy, and W. T. Freeman. Sharing visual features for multiclass and multiview object detection. *PAMI*, 29(5):854–869, 2007.

[24] V. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.