Sharing Visual Features for Multiclass and Multiview Object Detection

Antonio Torralba, Kevin P. Murphy and William T. Freeman

AI Memo 2004-008 April 2004

Abstract

We consider the problem of detecting a large number of different classes of objects in cluttered scenes. Traditional approaches require applying a battery of different classifiers to the image, at multiple locations and scales. This can be slow and can require a lot of training data, since each classifier requires the computation of many different image features. In particular, for independently trained detectors, the (run-time) computational complexity, and the (training-time) sample complexity, scales linearly with the number of classes to be detected. It seems unlikely that such an approach will scale up to allow recognition of hundreds or thousands of objects.

We present a multi-class boosting procedure (joint boosting) that reduces the computational and sample complexity, by finding common features that can be shared across the classes (and/or views). The detectors for each class are trained jointly, rather than independently. For a given performance level, the total number of features required, and therefore the computational cost, is observed to scale approximately logarithmically with the number of classes. The features selected jointly are closer to edges and generic features typical of many natural structures instead of finding specific object parts. Those generic features generalize better and reduce considerably the computational cost of an algorithm for multi-class object detection. [1].

[1]This work was sponsored in part by the Nippon Telegraph and Telephone Corporation as part of the NTT/MIT Collaboration Agreement, and by DARPA contract DABT63-99-1-0012.

I. Introduction

A long-standing goal of machine vision has been to build a system which is able to recognize many different kinds of objects in a cluttered world. This would allow machines to truly see and interact in the visual world.

Progress has been made on restricted versions of this goal. In particular, it is now possible to recognize individual *instances* of highly textured objects, such as magazine covers or toys, despite clutter, occlusion and affine transformations, by using object-specific features [19], [27]. In addition, it is possible to recognize *classes* of objects, generalizing over intra-class variation, when the objects are presented against simple uniform backgrounds [22], [23], [17], [7], [20]. However, combining these two approaches — dealing with clutter and generalizing across intra-class variability — remains very challenging.

The most challenging problem in object detection is to differentiate the object from the background. In the case that we want to detect C classes of objects, the problem of discriminating any object class against the background is more difficult than discriminating between the C classes for a patch known to contain one of the objects. One of the reasons for this is that the background generates many more distractors than the $C - 1$ other object classes.

The current state of the art for the problem of class-level object detection in clutter uses classifier methods, such as boosting and support vector machines, to distinguish the class from the background. Researchers have made reliable detectors for objects of a single class, such as faces (e.g., [31]) or pedestrians (e.g. [24]), seen over a limited range of viewing conditions. However, such approaches seem unlikely to scale up to the detection of hundreds or thousands of different object classes, or to many different views of objects, because each classifier computes many image features independently. These features typically involve convolutions with part templates [9], [30] or with a set of basis filters [31], [24]. Computing these features is slow, and it requires a lot of data to determine which features are useful. We believe that an essential ingredient in the scale-up to multi-object detection will be the sharing of image features for classification across multiple objects and views; that is, training the classes jointly instead of independently.

In this paper, we develop a new object classification architecture that explicitly learns to share features across multiple object classes. The basic idea is an extension of the boosting algorithm [25], [10], which has been shown to be useful for detecting individual object classes in cluttered scenes [31]. Rather than training C binary classifiers independently, we train them jointly. The basic idea of the algorithm is that for each possible subset of classes, we find a feature that is most useful for distinguishing that subset from the background; we then pick the best such feature/subset, and repeat, until the overall error (across all classes) stops decreasing, or until we reach a limit on the number of features we can afford to compute (to bound the run-time cost). (The details will be given below.) The result is that many fewer features are needed to achieve a desired level of performance than if we were to train the classifiers independently. This results in a faster classifier (since there are fewer features to compute) and one which works better (since the features are fit to larger, shared data sets).

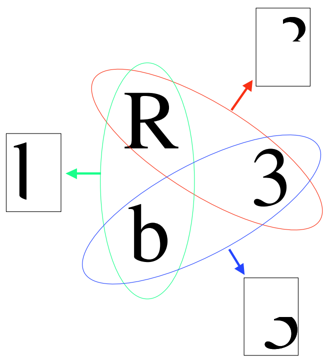


Fig. 1. Objects may share features in a way that cannot be represented as a tree. In this example, we can see how each pair of objects shares a part.

Section II summarize previous work on multiclass object detection and multiclass classifiers. We describe the joint boosting algorithm in Section III, and illustrate its performance on some artificial data sets. In Section IV, we show how the algorithm can be used to detect different classes and views of objects in real world images.

II. Previous work

We first discuss work from the computer vision literature, and then discuss work from the machine learning community.

A. Multi-class object detection

As mentioned in the introduction, it is helpful to distinguish four categories of work: object recognition (which assumes the object has been segmented out from the background) vs object detection (which assumes the object is buried in a cluttered scene); and class-level (e.g., all cars) vs instance-level (e.g., my red Toyota). In this paper, we are concerned with class-level object detection in clutter.

Fei-Fei, Fergus and Perona [8] propose a model based on the geometric configuration of parts; each part is represented as a local PCA template. They impose a prior on the model parameters for each class, which encourages each class to be similar, and allows the system to learn from small sample sizes. However, the parts themselves are not shared across classes.

Blanchard and Geman [3] use decision trees to build a hierarchical multi-class classifier. Our work differs from this since we do not create a hierarchy. Hence we can partition the classes into multiple overlapping sets, and compute features for each such partition. This is important as objects share features in more complicated patterns than a tree (See Figure 1).

Amit, Geman and Fian [2] describe a system for multiclass and multi-pose object detection based in a coarse-to-fine search. They model the joint distribution of poses between different objects in order to get better results than using independent classifiers.

Krempp, Geman and Amit [15] present a system that learns to reuse parts for detecting several object categories. The system is trained incrementally by adding one new category at each step and adding new parts as needed. They apply their system to detecting mathematical characters on a background composed of other characters. They show that the number of parts grows logarithmically with respect to the number of classes, as we have found. However, they do not jointly optimize the shared feature, and they have not applied their technique to real-world images.

B. *Multi-class classification*

The algorithm proposed in this paper relates to a general framework developed by Dietterich and Bakiri [6] for converting binary classifiers into multiple-class classifiers using error-correcting output codes (ECOC).

The idea is to construct a code matrix μ with entries in $\{-1, 0, +1\}$. There is one row per class and one column for each subset being considered. We fit a binary classifier for each column; the 1's in the column specify which classes to group together as positive examples, and the -1's specify which classes to treat as negative examples; the 0 classes are ignored. Given an example v, we apply each column classifier to produce a bit-vector, $(f_1(v), \ldots, f_n(v))$, where n is the number of columns. We then find the row which is closest in Hamming distance to this bit-vector, and output the corresponding class (row number).

The goal is to design encodings for the classes that are resistant to errors (misclassifications of the individual bits). There are several possible code matrices: (1) μ has size $C \times C$, and has $+1$ on the diagonal and -1 everywhere else; this corresponds to one-against-all classification. (2) μ has size $C \times \binom{C}{2}$ in which each column corresponds to a distinct pair of labels z_1, z_2; for this column, μ has $+1$ in row z_1, -1 in row z_2 and 0 in all other rows; this corresponds to building all pairs of i vs j classifiers [12]. (3) μ has size $C \times 2^C - 1$, and has one column for every non-empty subset; this is the complete case. (4) μ is designed randomly and is chosen to ensure that the rows are maximally dissimilar (i.e., so the resulting code has good error-correcting properties).

Allwein et. al. [1] show that the popular one-against-all approach is often suboptimal, but that the best code matrix to use is problem dependent. Although our algorithm starts with a complete code matrix (all possible subsets), it learns which subsets are actually worth using. It is possible to do a greedy search for the best code (see Section III-C); thus our algorithm effectively learns the code matrix, by alternating between fitting the column (subset) classifiers, and searching for good subsets, in a stage-wise fashion. The "bunching" algorithm in [5] also attempts to simultaneously learn the code matrix and solve the classification problem, but is more complicated than our algorithm, and has not been applied to object detection.

A difference between our approach and the ECOC framework is how we use the column (subset) classifiers. In ECOC, they classify an example by running each column classifier, and looking for the closest matching row in the code matrix. In our algorithm, we add the output of the individual column (subset) classifiers together, as in a standard additive model (which boosting fits).

Another set of related work is called "multiple task learning" [4]. This is also concerned with training classifiers for multiple classes simultaneously, and sharing features amongst them. However, as far as we know, this has not been done for boosting, or for the object detection task.

III. The joint boosting algorithm

Boosting [25], [10] provides a simple way to sequentially fit additive models of the form

$$H(v, c) = \sum_{m=1}^{M} h_m(v, c),$$

where c is the class label, and v is the input feature vector. In the boosting literature, the h_m are often called weak learners. It is common to define these to be simple decision or regression stumps of the form $h_m(v) = a\delta(v^f > \theta) + b$, where v^f denotes the f'th component (dimension) of the feature vector v, θ is a threshold, δ is the indicator function, and a and b are regression parameters (note that b does not contribute to the final classification). We can estimate the optimal a and b by weighted least squares, and can find the optimal feature f and threshold θ by exhaustive search through the data (see Section III-A for details).

Boosting was originally designed to fit binary classifiers. There are two common approaches to extend boosting to the multi-class case. The simplest is to train multiple separate binary classifiers (e.g., as in one-vs-all, or using some

$$\begin{tikzcd}
 & G^{12} \arrow[r] \arrow[dr] & G^{1} \\
G^{123} \arrow[ur] \arrow[r] \arrow[dr] & G^{13} \arrow[ur] \arrow[dr] & G^{2} \\
 & G^{23} \arrow[ur] \arrow[r] & G^{3}
\end{tikzcd}$$

Fig. 2. All possible ways to share features amongst 3 classifiers. In this representation, each classifier $H(v, c)$ is constructed by adding, only once, all the nodes that connect to each of the leaves. The leaves correspond to single classes. Each node corresponds to a different grouping of classes.

kind of error-correcting output code). Another approach is to ensure the weights at each stage m define a normalized probability distribution over all the classes [10]. However, even in the latter case, the weak-learners are class-specific.

We propose to share weak-learners across classes. For example, if we have 3 classes, we might define the following classifiers:

$$H(v,1) = G^{1,2,3}(v) + G^{1,2}(v) + G^{1,3}(v) + G^{1}(v)$$

$$H(v,2) = G^{1,2,3}(v) + G^{1,2}(v) + G^{2,3}(v) + G^{2}(v)$$

$$H(v,3) = G^{1,2,3}(v) + G^{1,3}(v) + G^{2,3}(v) + G^{3}(v)$$

where each $G^{S(n)}(v)$ is itself an additive model of the form $G^{S(n)}(v) = \sum_{m=1}^{M_n} h_m^n(v)$. The n refers to a node in the "sharing graph" (see Figure 2), which specifies which functions can be shared between classifiers. $S(n)$ is the subset of classes that share the node n. In the example shown in Fig. 5, we have $S(1) = \{1, 2, 3\}$, which means that the additive model $G^{S(1)}$ is useful for distinguishing all 3 classes from background; similarly, $S(3) = \{1, 3\}$, which means that the additive model $G^{S(3)}$ is useful for distinguishing classes 1 and 3 from the background (the background may or may not include class 2 — see discussion below). If we allow all possible subsets, the graph will have $2^C - 1$ nodes, but the algorithm will figure out which subsets are most useful.

The decomposition is not unique (different choices of functions $G^{S(n)}(v)$ give the same functions $H(v, c)$). But we are interested in the choices of $G^{S(n)}(v)$ that minimize the computational cost. We impose the constraint that $\sum_n M_n = M$, where M is the total number of functions that have to be learned. So the total of constructed functions is equal to the number of rounds of boosting.

If the classifiers are trained independently, then $M = O(C)$. By jointly training, we can use a number of features that is sub-linear in the number of classes.

For instance, suppose we have C classes, and, for each class, the feature vectors reside within some sphere in a D-dimensional space. Further, suppose the weak classifiers are hyperplanes in the D-dimensional space. If the classes are arranged into a regular grid, then, by sharing features, we need $M = 2DC^{1/D}$ hyperplanes to approximate the hyperspherical decision boundary with hypercubes.

Note that asymptotically (provided enough complexity and training samples) a multi-class classifier might converge to the same performance as a classifier that shares features across classes. However, we are interested in the complexity needed to achieve a particular performance with a reduced set of training data.

The natural structure of object categories and the regularities in the building blocks that compose visual objects (edges, parts, etc.) makes the problem of object detection a good candidate for applying multiclass procedures that share features across classifiers.

A. The joint boosting algorithm

The idea of the algorithm is that at each boosting round, we examine various subsets of classes, $S \subseteq C$, and considering fitting a weak classifier to distinguish that subset from the background. We pick the subset that maximally reduces the error on the weighted training set for *all* the classes. The best weak learner $h(v, c)$ is then added to the strong learners $H(v, c)$ for all the classes $c \in S$, and their weight distributions are updated.

We decide to optimize the following multiclass cost function:

$$J = \sum_{c=1}^{C} E\left[e^{-z^c H(v,c)}\right] \qquad (1)$$

where z^c is the membership label (± 1) for class c. The term $z^c H(v, c)$ is called the "margin", and is related to the generalization error (out-of-sample error rate).

We chose to base our algorithm on the version of boosting called "gentleboost" [11], because it is simple to implement, numerically robust, and has been shown experimentally [18] to outperform other boosting variants for the face detection task. The optimization of J is done using adaptive Newton steps [11] which corresponds to minimize a weighted squared error at each step. At each step m, the function H is updated as: $H(v, c) := H(v, c) + h_m(v, c)$, where h_m is chosen so as to minimize a second order Taylor approximation of the cost function:

$$\arg\min_{h_m} J(H + h_m) \simeq \arg\min_{h_m} \sum_{c=1}^{C} E\left[e^{-z^c H(v,c)}(z^c - h_m)^2\right] \qquad (2)$$

Replacing the expectation with an empirical expectation over the training data, and defining weights $w_i^c = e^{-z_i^c H(v_i,c)}$ for example i and class c, this reduces to minimizing the weighted squared error:

$$J_{wse} = \sum_{c=1}^{C} \sum_{i=1}^{N} w_i^c (z_i^c - h_m(v_i, c))^2. \qquad (3)$$

```
1) Initialize the weights $w_i^c = 1$ and set $H(v_i, c) = 0$, $i = 1..N$, $c = 1..C$.
2) Repeat for $m = 1, 2, \ldots, M$
   a) Repeat for $n = 1, 2, \ldots, 2^C - 1$
      i) Fit shared stump:
         $h_m(v,c) = \begin{cases} a\delta(v_i^f > \theta) + b & \text{if } c \in S(n) \\ k^c & \text{if } c \notin S(n) \end{cases}$
      ii) Evaluate error
         $J_{wse}(n) = \sum_{c=1}^{C} \sum_{i=1}^{N} w_i^c (z_i^c - h_m(v_i, c))^2$
3) Find best sharing by selecting $n = \arg\min_n J_{wse}(n)$, and pick the corresponding shared feature $h_m(v,c)$.
4) Update
         $H(v_i, c) := H(v_i, c) + h_m(v_i, c)$
         $w_i^c := w_i^c e^{-z_i^c h_m(v_i, c)}$
```

Fig. 3. Joint boosting with regression stumps. v_i^f is the f'th feature of the i'th training example, $z_i^c \in \{-1, +1\}$ are the labels for class c, and w_i^c are the *unnormalized* example weights. N is the number of training examples, and M is the number of rounds of boosting.

The resulting optimal function at round m, h_m, is defined as:

$$h_m(v,c) = \begin{cases} a\delta(v_i^f > \theta) + b & \text{if } c \in S(n) \\ k^c & \text{if } c \notin S(n) \end{cases} \tag{4}$$

with parameters $(a, b, f, \theta, n, k^c)$, for each $c \notin S(n)$, i.e., a total of $5 + C - |S(n)|$ parameters. Here $\delta(x)$ is the indicator function, and n denotes a node in the "sharing graph" (Figure 2), which corresponds to a subset of classes $S(n)$. By adding a shared stump, the complexity of the multi-class classifier increases with constant rate, independent of the number of classes sharing the stump. Only the classes that share the feature at round m will have a reduction of their classification error.

The minimization of (3) gives the parameters:

$$b = \frac{\sum_{c \in S(n)} \sum_i w_i^c z_i^c \delta(v_i^f \leq \theta)}{\sum_{c \in S(n)} \sum_i w_i^c \delta(v_i^f \leq \theta)}, \tag{5}$$

$$a + b = \frac{\sum_{c \in S(n)} \sum_i w_i^c z_i^c \delta(v_i^f > \theta)}{\sum_{c \in S(n)} \sum_i w_i^c \delta(v_i^f > \theta)}, \tag{6}$$

$$k^c = \frac{\sum_i w_i^c z_i^c}{\sum_i w_i^c} \quad c \notin S(n) \tag{7}$$

For all the classes c in the set $S(n)$, the function $h_m(v,c)$ is a shared regression stump. For the classes that do not share this feature ($c \notin S(n)$), the function $h(v,c)$ is a constant k^c different for each class. This constant prevents sharing features due to the asymmetry between positive and negative samples for each class. These constants do not contribute to the final classification. Fig. 3 summarizes the algorithm.

As we do not know which is the best sharing $S(n)$, at each iteration we search over all $2^C - 1$ possible sharing patterns to find them one that minimizes eq. (3). Obviously this is very slow. In Section III-B, we discuss a way to speed this up by a constant factor, by reusing computation at the leaves to compute the score for interior nodes of the sharing graph. In Section III-C, we discuss a greedy search heuristic that has complexity $O(C^2)$ instead of $O(2^C)$.

B. Efficient computation of shared regression stumps

To evaluate the quality of a node in the sharing graph, we must find the optimal regression stump which is slow, since it involves scanning over all features and all N thresholds (where N is the number of training examples). However, we can propagate most of the computations from the leaves to higher nodes, as we now discuss.

At each boosting round, and for each isolated class (the leaves of the graph), we compute the parameters a and b for a set of predefined thresholds and for all features, so as to minimize the weighted square error. Then, the parameters a and b for each threshold and feature at any other internal node can be computed simply as a weighted combination of the errors at the leaves that are connected with the node. The best regression parameters for a subset of classes S is:

$$b_S(f, \theta) = \frac{\sum_{c \in S} b_c(f, \theta) w_c^+(f, \theta)}{\sum_{c \in S} w_c^+(f, \theta)} \tag{8}$$

with $w_c^+(f, \theta) = \sum_{i=1}^N w_i^c \delta(v_i^f > \theta)$. Similarly for a_S. For each feature f, and each threshold θ, the joint weighted regression error, for the set of classes $S(n)$, is:

$$\begin{aligned} J_{wse}(n) &= (1 - \hat{a}_s^2) \sum_{c \in S(n)} w_c^+ + (1 - b_s^2) \sum_{c \in S(n)} w_c^- + \\ &\quad + \sum_{c \notin S(n)} \sum_{i=1}^N w_i^c (z_i^c - k^c)^2 \end{aligned} \tag{9}$$

with $\hat{a}_s = a_s + b_s$. The first two terms correspond to the weighted error in the classes sharing a feature. The third term is the error for the classes that do not share a feature at this round. This can be used instead of Eq. 3, for speed.

C. Approximate search for the best sharing

As currently described, the algorithm requires computing features for all possible $2^C - 1$ subsets of classes, so it does not scale well with the number of classes. Instead of searching among all possible $2^C - 1$ combinations, we use best-first search and a forward selection procedure. This is similar to techniques used for feature selection but here

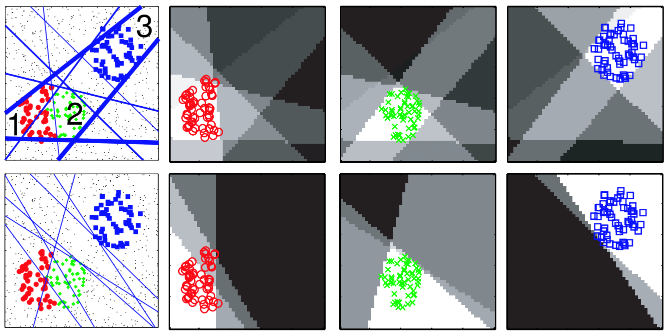


Fig. 4. Illustration of joint boosting (top row) and independent boosting (bottom row) on a toy problem in which there are three object classes and one background class. 50 samples from each class are used for training, and we use 8 rounds of boosting. Left: The thickness of the lines indicates the number of classes sharing each regression stump. Right: whiter colors indicate that the class is more likely to be be present (since the output of boosting is the log-odds of class presence [10]).

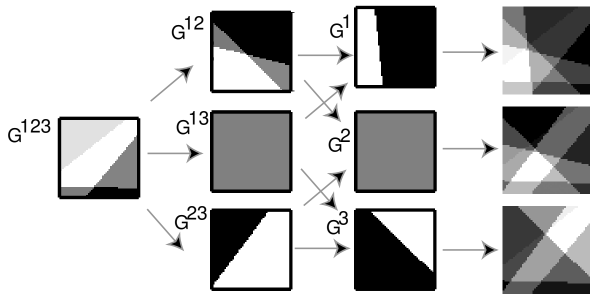


Fig. 5. Decision boundaries learned by all the nodes in the sharing graph of Figure 2 for the problem in Figure 4

we group classes instead of features (see [14] for a review of feature selection techniques).

At each round, we have to decide which classes are going to share a feature. We start by computing all the features for the leaves (single classes) as described in the previous section. We select first the class that has the best reduction of the error. Then we select the second class that has the best error reduction jointly with the previously selected class. We iterate until we have added all the classes. We select the sharing that provides the largest error reduction. The complexity is quadratic in the number of classes, requiring us to explore $C(C+1)/2$ possible sharing patterns instead of $2^C - 1$. We can improve the approximation by using beam search considering at each step the best $N_c < C$ classes. We found empirically that using this approximate optimization (with $N_c = 1$) the performance of the final classifier did not differ from an exhaustive search (Fig. 6).

The experimental results in Section IV on object detection show that the algorithm scales well with the dimensionality of the feature vector and the number of object classes.

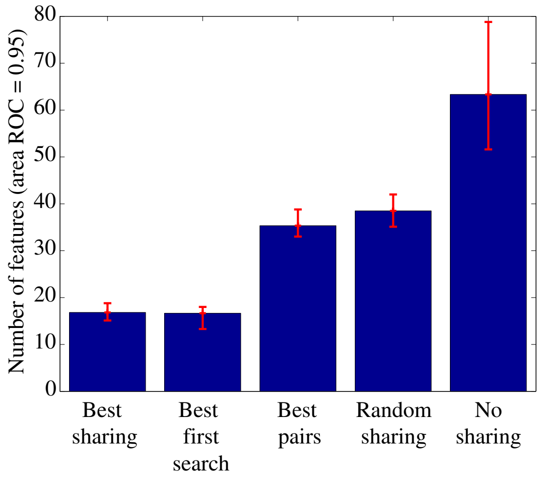


Fig. 6. Comparison of number of stumps needed to achieve the same performance (area under ROC equal to 0.95) when using exact search, best-first, best pair, random sharing and no sharing at each round. We use a toy data set with $C = 9$ classes plus a background class in $D = 2$ dimensions. Both exact search and the approximate best-first search provide the lower complexities. The differences increase as more classes are used (Fig. 7).

D. *Example of joint boosting on a toy problem*

To illustrate the benefits of joint boosting, we compared joint boosting with independent boosting on a toy data set, which consists of C spherical "clouds" of data in D dimensions, embedded in a uniform "sea" of background distractors. Some results are shown in Fig. 4. This clearly illustrates the benefit of sharing features when we can only afford to compute a small number (here, 8) of stumps[1]. In this case, the first shared function has the form $G^{123}(v) = \sum_{m=1}^{3} h_m^{123}(v)$, meaning that the classifier which separates classes 1,2,3 vs. the background has 3 decision boundaries. The other nodes have the following number of boundaries: $M_{123} = 2$, $M_{12} = 2$, $M_{23} = 2$, $M_{13} = 0$, $M_1 = 1$, $M_2 = 0$, $M_3 = 1$, so there are no pure boundaries for class 2 in this example.

Fig. 6 illustrates the differences between the exact search of the best sharing, the best first approximate search, the best pairs only, a random sharing and one-vs-all (no sharing). For this experiments we use only two dimensions, 25 training samples per class, and 8,000 samples for the background. For each search algorithm the graph shows the number of stumps needed to achieve a fixed level of performance (area under the ROC $= 0.95$). In this result

[1]In this 2D example, the feature vectors are the projection of the coordinates onto lines at 60 different angles coming from the origin. For the higher dimensional experiments described below, we use raw coordinate values as the features.

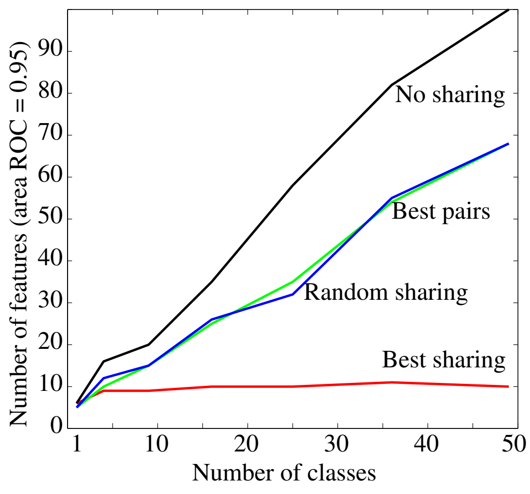


Fig. 7. Complexity of the multiclass classifier as a function of the number of classes. The complexity of a classifier is evaluated here as the number of stumps needed for achieving a predefined level of performance (area under the ROC of 0.95).

we use $C = 9$ classes so that it is still efficient to search for the best sharing at each round using exact search. First we can see that using the exact best sharing or the one obtained using the approximate search (best first) provides similar results. The complexity of the resulting multiclass classifier (17 stumps) is smaller than the complexity of a one-vs-all classifier that requires 63 stumps for achieving the same performance.

Fig. 7 illustrates the dependency of the complexity of the classifier as a function of the number of classes when using different sharing patterns. For this experiments we use 2 dimensions, 25 training samples per class, and 40,000 samples for the background. As expected, when no sharing is used (one-vs-all classifier), the complexity grows linearly with the number of classes. When the sharing is allowed to happen only between pairs of classes, then the complexity is lower that the one-vs-all but still grows linearly with the number of classes. The same thing happens with random sharing. A random sharing will be good for at least two classes at each round (and for D classes in D dimensions). It also performs better than one-vs-all but still complexity grows linearly with respect to the number of classes. However, when using the best sharing at each round (here using best-first search), then the complexity drops dramatically and the dependency between complexity and number of classes follows a logarithmic curve.

IV. Joint boosting applied to multiclass object detection

Having described the joint boosting algorithm in general, we now explain how to apply it to object detection.

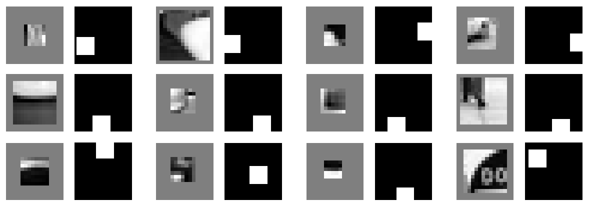


Fig. 8. Each feature is composed of a template (image patch on the left) and a binary spatial mask (on the right) indicating the region in which the response will be averaged. The patches vary in size from 4x4 pixels to 14x14.

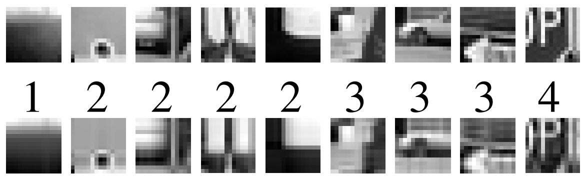


Fig. 9. Each feature is approximated by a linear combination of separable filters. The top-row shows the original patches, the middle row is the order of the approximation required to have $|\hat{g}_f^T g_f| > 0.95$, and the bottom row is the reconstruction of the patch. Using patches of $14 * 14$, each convolution requires 196 operations per pixel and per feature on average. The separable approximation requires 70 operations per pixel and per feature on average.

The need for sharing features arises in several situations in object detection:

1) Multi-class object detection: we want to share features that are common to different types of objects.

2) Multi-view object detection: several points of views of an object may share common visual appearances. For example, a ball is very similar under different points of view. On the other hand, the appearance of an object such as a car or flat-panel monitor will change dramatically from different viewpoints, requiring that different features be used.

3) Location and scale invariant object detection: we want to scan the image in location and scale to find the target. Hence we have to apply the same classifier many times in the image. If we can share computations at different locations and scales, the search will be more efficient.

In this section, we concentrate on the first problem (view-specific, multi-class detection).

A. *Dictionary of features*

In the study presented here we used 21 object categories (13 indoor objects: screen, keyboard, mouse, mouse pad, speaker, computer, trash, poster, bottle, chair, can, mug, light; 7 outdoor objects: frontal view car, side view car, traffic light, stop sign, one way sign, do not enter sign; and also heads and pedestrians.).

First we build a dictionary of image patches by randomly extracting patches from images of the 21 objects that we want to detect. The objects were normalized in size in order to be centered in a square window of 32x32 pixels. To each patch we associated a spatial mask that indicates the location from which the patch was extracted from the original image (see Figure 8 for some examples of the features). We generated a total of 2000 patches.

For each image region of standardized size (32x32 pixels), we compute a feature vector of size 2000. The vector of features computed at location x and scale σ is given by:

For each image region of standardized size (here 32x32), we compute a feature vector of size 2000. Here we slightly modify the notation from before, to make explicit the location in the image at which the features are computed:

$$v^f(x,\sigma) = (w_f * |I_\sigma \otimes g_f|^p)^{1/p} \tag{10}$$

where $\otimes$ represents the normalized correlation between the image I_σ and the patch/filter g_f, and $*$ represents the convolution operator. $w_f(x)$ is a spatial mask/ window. $v(x,\sigma)$ is the vector of features computed at location x and scale σ, and v^f is the f'th component of the vector. Features corresponding to different scales σ are obtained by scaling the image. The exponent p allows us to generate different types of features. For example, by setting $p = 1$, the feature vector encodes the average of the filter responses, which are good for describing textures ([21]). By setting $p > 10$, the feature vector becomes $v^f \simeq \max_{x \in S_w} \{|I_\sigma \otimes g_f|\}$, where $S_w(x)$ is the support of the window for a feature at the location x. This is good for template matching [30]. By changing the spatial mask, we can change the size and location of the region in which the feature is evaluated. This provides a way of generating features that are well localized (good for part-based encoding and template matching) and features that provide a global description of the patch (good for texture-like objects, e.g. a bookshelf).

For the experiment presented in this section we set $p = 10$, and we took the window w_n to be localized around the region from where the patch was extracted in the original image, c.f., [30].

During training we extract thousands of patches from training images and we have to compute the convolution of each patch with each of the whole training images. The computational cost can be reduced by approximating each patch g_f with a linear combination of 1D separable filters:

$$\hat{g}_f = \sum_{n=1}^{r} u_n v_n^T \tag{11}$$

where u_n and v_n are 1D filters and r is the order of the approximation. This decomposition can be obtained by applying SVD to the matrix g_f [29], [16]. For each filter, we chose r so that $|\hat{g}_f^T g_f|$ is larger than 0.95. Fig. 9 shows examples of patches 14x14 pixels and their separable approximations. Using patches of 14x14, each convolution requires 196 operations per pixel and per feature on average. The separable approximation requires 70 operations per pixel and per feature on average.

Convolution with the masks w_f can be implemented in a small number of operations using the integral image trick [31].

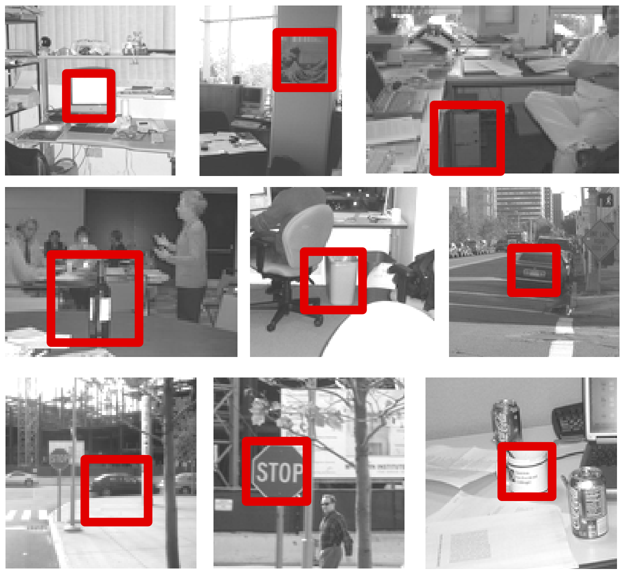


Fig. 10. Examples of correct detections of several object detectors trained jointly (screen, poster, cpu, bottle, trash, car frontal, car side, stop sign, mug).

B. *Results on multiclass object detection*

For training we used a hand-labeled database of 2500 images. We train a set of 21 detectors using joint and independent boosting. In both cases, we limit the number of features to be the same in order to compare performance for the same computational cost. Each feature is defined by the parameters: $\{f, a, b, \theta\}$, where f specifies an entry from the dictionary, $\{w_f(x), g_f(x), p_f\}$, and the parameters $\{a, b, \theta\}$ define the regression stump.

Figure 10 shows some sample detection results when running the detectors on whole images by scanning each location and scale. Figure 11 summarizes the performances of the detectors for each class. For the test, we use an independent set of images (images from the web, and taken with a digital camera). All the detectors have better performances when trained jointly, sometimes dramatically so.

Note that as we reduce the number of features and training samples all the results get worse. In particular, when training the detectors independently, if we allow fewer features than classes, then some classifiers will have no features, and will perform at chance level (a diagonal line on the ROC). Even for the classifiers that get some

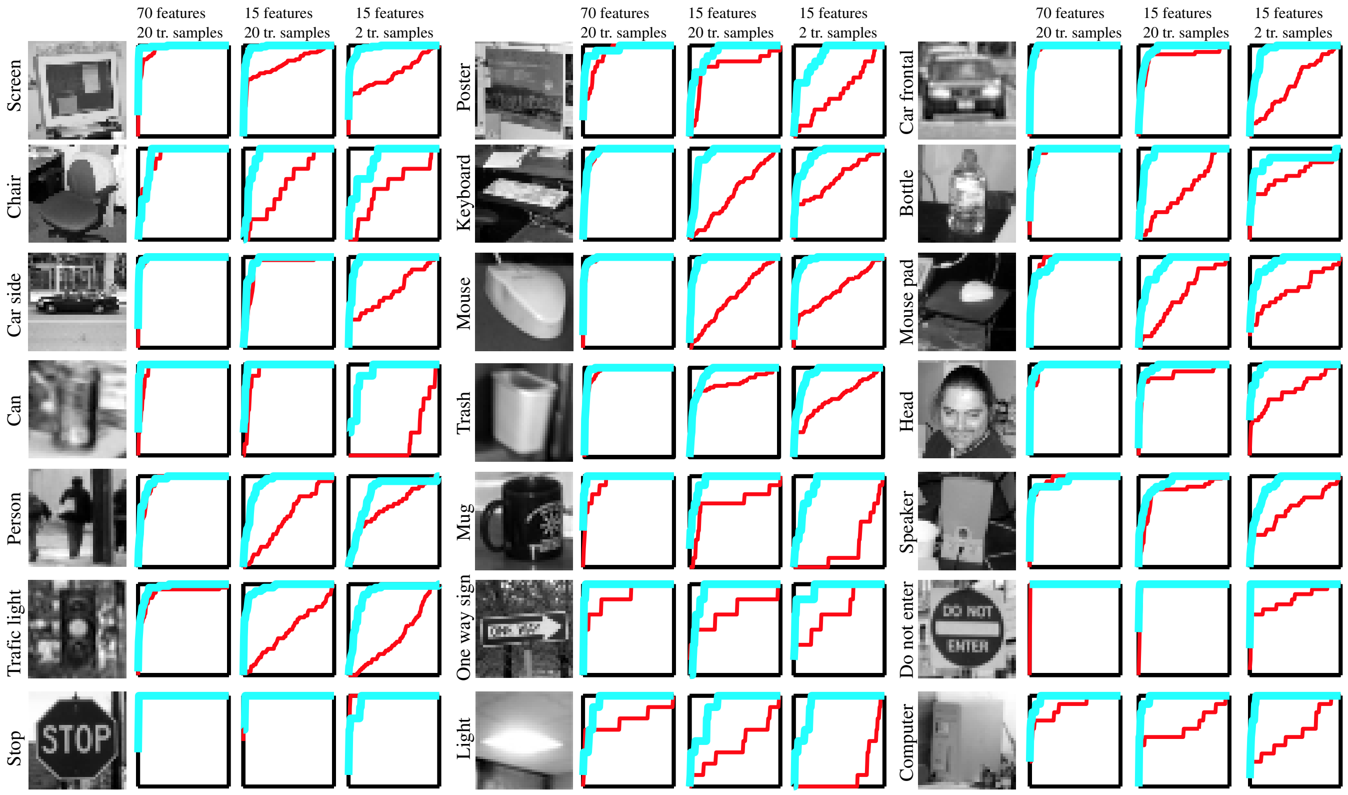


Fig. 11. ROC curves for 21 objects (red (lower curve) = isolated detectors, blue (bottom curve) = joint detectors). For each object we show the ROC obtained with different training parameters. From left to right: i) 70 features in total (on average $70/21 \simeq 3.3$ features per object) and 20 training samples per object, ii) 15 features and 20 training samples, and iii) 15 features and 2 training samples.

features, the performance can be bad — sometimes it is worse than chance (below the digonal), because there is not enough data to reliably pick the good features or to estimate their parameters. Conversely, the jointly trained detectors perform well even as we reduce the amount of computation time and training data.

Figure 12 shows how the performance of the multiclass detectors improve as we add rounds to the multiclass boosted classifier and compares with respect to independent boosted classifiers. The horizontal axis of the figure corresponds to the number of features (rounds of boosting) used for all the object classes. The vertical axis shows the area under the ROC for the test set, averaged across all object classes. When enough training samples are provided, and many boosting rounds are allowed, then both joint and independent classifiers will converge to the same performance, as both are equivalent additive classifiers. However, when only a reduced number of rounds are allowed (in order to reduce computational cost), the joint training outperforms the isolated detectors (see also Fig. 11).

As more and more objects are trained jointly we expect a larger improvement with respect to independent training, as it will be possible to find more object sets that share relevant features. We are currently increasing our database to work with more objects.

C. Feature sharing

By training the objects using joint boosting, at each round we find what is the feature that best reduces the total multiclass classification error. Figure 13 shows an example of a feature shared between two objects at one of the boosting rounds. The selected feature can help discriminate both trashcans and heads against the background, as is shown by the distribution of positive and negative samples a long the feature dimension. As this feature reduces the error in two classes at once, it has been chosen over other more specific features that might have been performed better on a single class, but which would have result in worst performance when considering the multiclass loss function.

Figure 14 shows the evolution of the number of objects sharing features for each boosting round.

Figure 15 shows the final set of features selected (the parameters of the regression stump are not shown) and the sharing matrix provided by jointBoosting that specifies

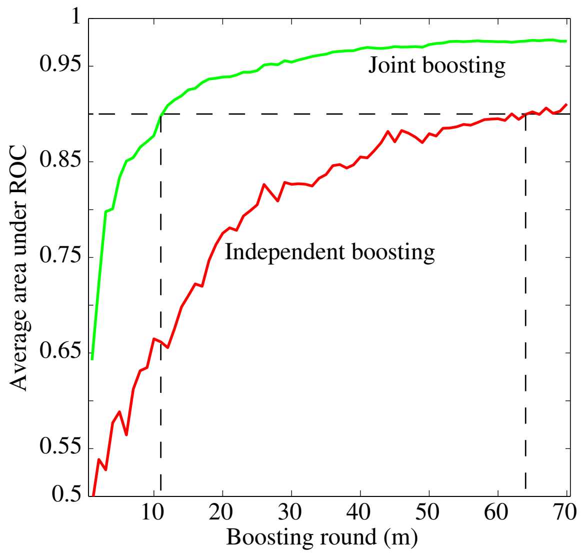


Fig. 12. Evolution of classification performance of the test set as a function of number of boosting rounds (or features). Performance is measured as the average area below the ROC across all classes. Chance level is 0.5 and perfect detection for all objects correspond to area= 1. Both joint and independent detectors are trained using up to 70 features (boosting rounds), 20 training samples per object and 21 object classes. The dashed lines indicate the number of features needed when using joint or independent boosting for the same performance.

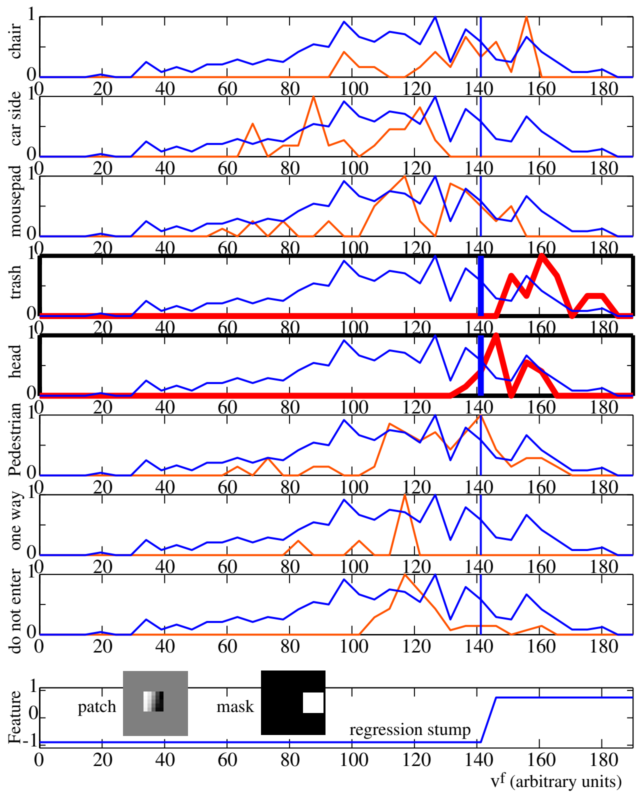


Fig. 13. Example of a shared feature (obtained at round 4 of joint boosting) between two objects (heads and trash-cans) when training 8 objects jointly. The shared feature is shown at the bottom of the figure. It is defined by an image feature (template and mask) and a regression stump (a, b and θ). For each object, the blue graph shows an empirical approximation to $p(v^f|z^c = 0)$ (negative examples), and the red graph shows $p(v^f|z^c = 1)$ (positive examples). The x-axis represent the feature indices f on an arbitrary scale.

how the different features are shared across the 21 object classes. Each column corresponds to one feature and each row shows the features used for each object. A white entry in cell (i, j) means that object i uses feature j. The features are sorted according to the number of objects that use each feature. From left to right the features are sorted from generic features (shared across many classes) to class-specific features (shared among very few objects).

D. *Specific vs. generic features*

One important consequence of training object detectors jointly is in the nature of the features selected for multi-class object detection.

When training objects jointly, the system will look for features that generalize across multiple classes instead on focusing on class-specific features. The features selected jointly are closer to edges and generic features typical of many natural structures. Those features generalize better and reduce considerably the computational cost of an algorithm for multi-class object detection.

Other studies have argued about the superiority of class-specific features against generic features (e.g., [30]). Object detection algorithms based on the configuration of parts (e.g., [30], [13], [9]) usually find parts that are class specific. For instance, faces are decomposed into parts that look like eyes, mouth, etc.; car detectors generally identify parts that correspond to meaningful object parts such as wheels. However, we argue that those parts are too specific for building efficient multiclass object detection algorithms that can scale to large number of objects with low computational cost.

The human visual system is able to detect such meaningful parts such as eyes, wheels, etc. In our framework, those parts can also be detected as we can detect single objects. There is no special status for a part. A wheel is detected in the same way that we detect a car. Of course, different objects can interact contextually. In that sense, a wheel detector will improve the performances of a car detector. But the car detector will not rely directly on the wheel detector as in a part-based detection approach.

Fig. 16 illustrates the difference between class-specific and generic features. In this figure we show the features selected for detecting a traffic sign. This is a well-defined object with a very regular shape. Therefore, a detector

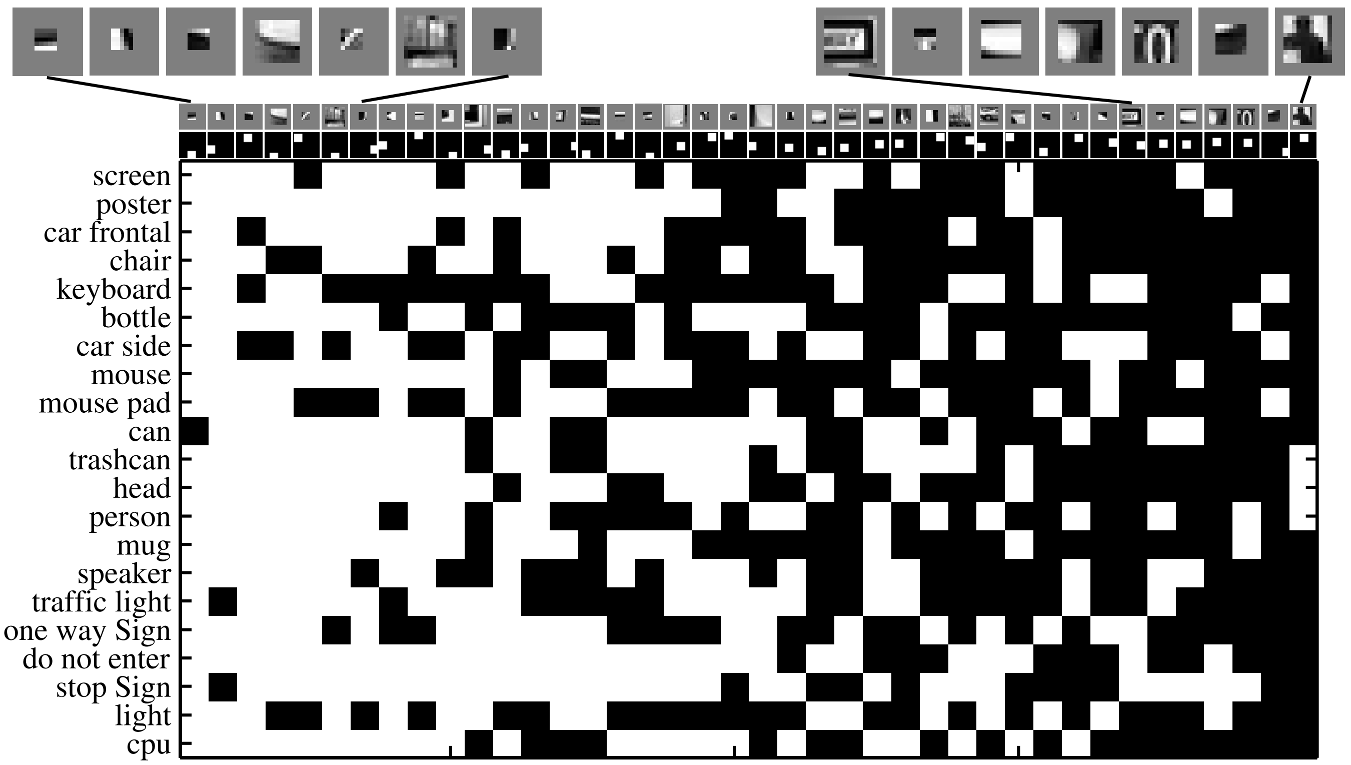


Fig. 15. Matrix that relates features to classifiers, which shows which features are shared among the different object classes. The features are sorted from left to right from more generic (shared across many objects) to more specific. Each feature is defined by one filter, one spatial mask and the parameters of the regression stump (not shown). These features were chosen from a pool of 2000 features in the first 40 rounds of boosting.

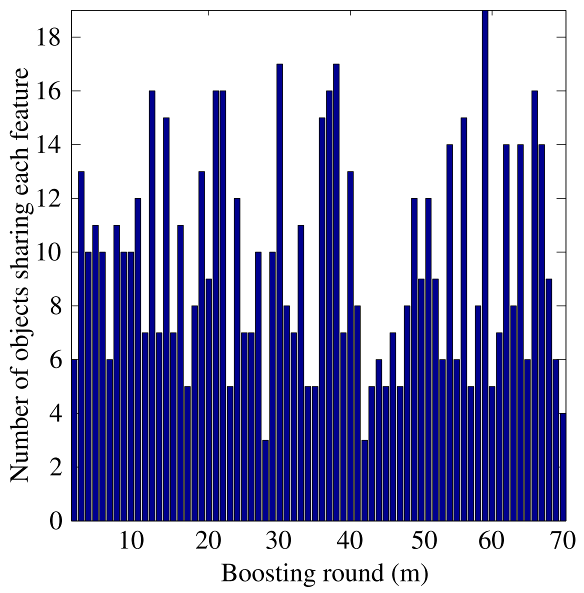


Fig. 14. This graph shows how many objects share the same feature at each round of boosting during training. Here we train 21 objects jointly using 20 training samples for each object. Note that a feature shared among 10 objects is in fact using $20 * 10 = 200$ training samples.

based on template matching will be able to perform perfectly. Indeed, when training a single detector using boosting, most of the features are class-specific and behave as a template matching detector (see fig. 16b). But when we need to detect thousands of other objects, we cannot afford to develop such specific features for each object. This is what we observe when training the same detector jointly with 20 other objects. The new features (fig. 16c) are more generic (configuration of edges) which can be reused by other objects. Although the features are less optimal for a particular object, the fact that we can allocate more features for each object results in better performance (fig. 11).

E. Computational cost

One important consequence of feature sharing is that the number of features needed grows sub-linearly with respect to the number of classes. Fig. 17 shows the number of features necessary (vertical axis) to obtain a fixed performance as a function of the number of object classes to be detected (horizontal axis). When using C independent classifiers, the complexity grows linearly as expected. However, when joint boosting is used, the complexity is compatible with

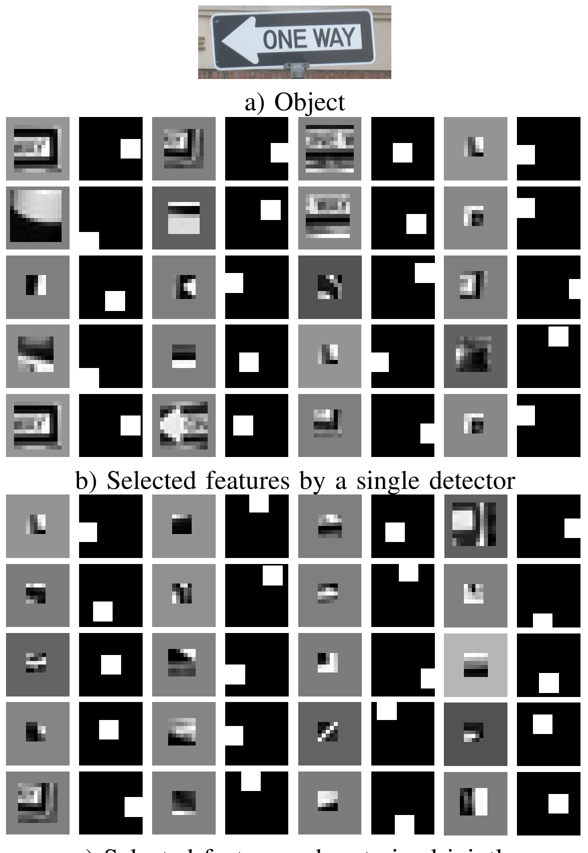

a) Object
b) Selected features by a single detector
c) Selected features when trained jointly

Fig. 16. Specific vs. generic features for object detection. (a) An object with very little intra-class variation. (b) When training an independent detector, the system learns template-like filters. (c) When training jointly, the system learns more generic, wavelet-like filters.

$log(C)$. (A similar result has been reported by Krempp, Geman and Amit ([15]) using character detection as a test bed.) In fact, as more and more objects are added, we can achieve good performance in all the object classes even using fewer features than objects.

F. Learning from few examples

Another important consequence of joint training is that the amount of training data requires is reduced. Fig. 11 shows the ROC for the 21 objects trained with 20 samples per object, and also with only 2 samples per objects. When reducing the amount of training, some of the detectors trained in isolation perform worse than chance level (which will be the diagonal on the ROC), which means that the selected features were misleading. This is due to the lack of training data, which hurts the isolated method more.

G. Grouping of object categories

We can measure similarity between two objects by the number of features that they have in common. Figure 18 shows the result of a greedy clustering algorithm using this similarity measure. Objects that are close in the tree are objects that share many features, and therefore share most of their computations. The same idea can be used to group features (results not shown).

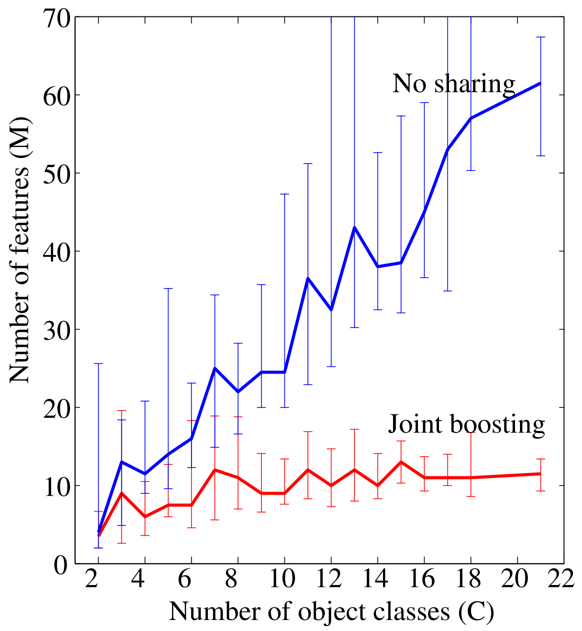


Fig. 17. Number of features needed in order to reach a fix level of performance (area under the ROC equal to 0.9). The results are averaged across 8 training sets (and different combinations of objects). The error bars show the variability between the different runs.

H. Loss function for multiclass object detection

We have given the same weight to all errors. But some mislabelings might be more important than others. For instance, it is not a big error if a mug is mislabeled as a cup, or if a can is mislabeled as a bottle. However, if a frontal view of a car is mislabeled as a door that could be hazardous. Changing the loss function will have consequences for deciding which objects will share more features. The more features that are shared by two objects, the more likely it is that they are going to be confused at the detection stage. We leave exploring this issue to future work.

V. Multiview object detection

An important problem in object detection is to deal with the large variability in appearances and poses that an object can have in a scene. Most object detection algorithms deal with the detection of one object under a particular point of view (e.g., frontal faces). When building view invariant object detectors, the standard approach is to discretize the

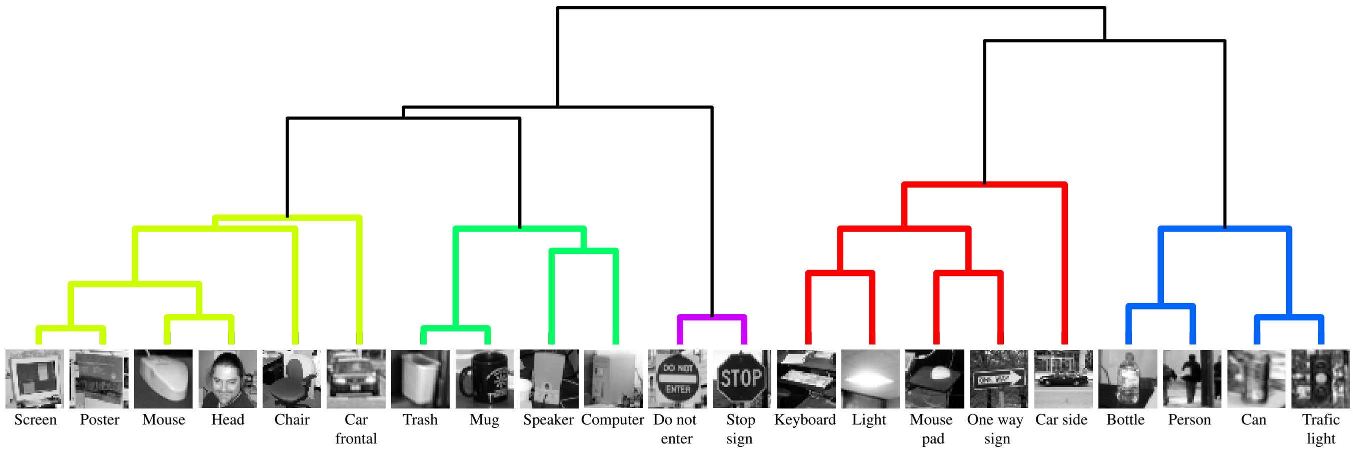


Fig. 18. Clustering of objects according to the similarity induced by the joint Boosting procedure. Objects that are close in the tree are objects that share many features and therefore, share most of the computations when running the classifiers on images. This clustering is obtained by training jointly 21 objects, using 70 stumps and 50 training samples per object.

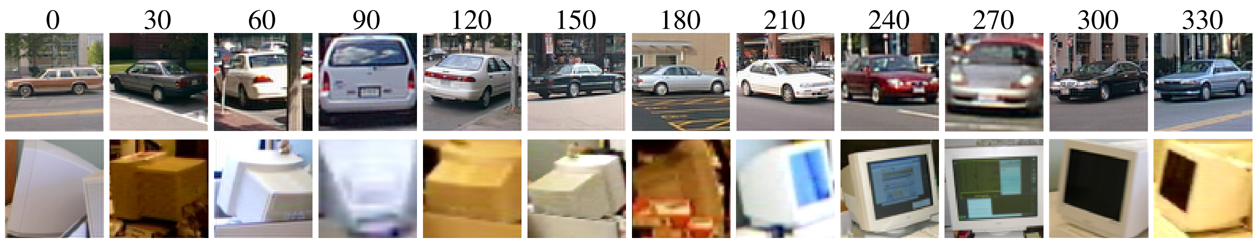


Fig. 19. Examples of pose variations for cars and screens (the angles are approximate).

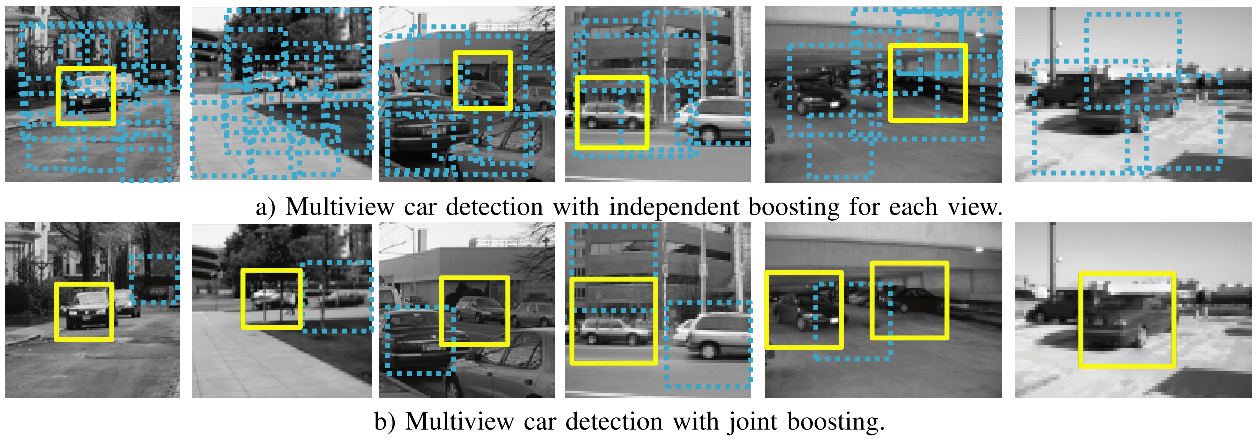

a) Multiview car detection with independent boosting for each view.
b) Multiview car detection with joint boosting.

Fig. 20. View invariant car detection (dashed boxes are false alarms, and solid boxes are correct detections). The figure shows a comparison of car detections with a battery of binary classifiers for each view trained individually (a), and jointly (b). The joint training provides more robust classifiers with the same complexity. In both cases, the classifiers were trained using 20 samples per view (12 views), and use 70 stumps in total. Both classifiers are set in order to provide 80% detection rate. The independent training of each view provides poor results with over 8 false alarms per image. When training the classifiers using joint boosting, the detector has 1 false alarm per image on average. Images are about 128x128 and contain more than 17000 patches to be classified.

space of poses, and to implement a set of binary classifiers, each one tuned to a particular pose (e.g., [26]).

In the case of multiple views, some objects have poses that look very similar. For instance, in the case of a car, both frontal and back views have many common features, and both detectors should share a lot of computations. However, in the case of a computer monitor, the front and back views are very different, and we will not be able to share features. By sharing features we can find a good trade-off between specificity of the classifier (training on very specific views) and computational complexity (by sharing features between views).

One problem when discretizing the space of poses is to decide how fine the discretization should be. The finer the sampling, the more detectors we will need and hence the larger the computational cost. However, when training the detectors jointly, the computational cost does not blow up in this way: If we sample too finely, then the sharing will increase as the detectors become more and more correlated.

Figure 20 shows the results of multiview car detectors and compares the classifiers obtained using independent boosting training for each view and joint boosting. In both cases, we limit the number of stumps to 70 and training is performed with 20 samples per view (12 views). Both classifiers have the same computational cost. The top row shows typical detection results obtained by combining 12 independent binary classifiers, each one trained to detect one specific view. When the detection threshold is set to get 80% detection rate, independent classifiers produce over 8 false alarms per image on average (average obtained on 200 images not used for training). The bottom row shows the results obtained when trained jointly the twelve view specific classifiers. For 80% detection rate, the joint classifier results in 1 false alarm per image. Images for the test were 128x128, which produced more than 17000 patches to be classified. The detector is trained on square patches 24x24 pixels. Fig. 21 summarizes the result showing the ROC for both detectors.

We trained a set of classifiers $H(v, c, \theta_i)$, for each class c and pose θ_i (with some tolerance). For those patches in which the detector is above the detection threshold, $max_i \{H(v, c, \theta_i)\} > th$, we can estimate the pose of the object as $\theta = argmax_{\theta_i} \{H(v, c, \theta_i)\}$. For some objects, it is very difficult to estimate the direction (for instance, for cars, the error is centered around π), so we might get ties in the pose detector outputs. Figure 22 shows some results in the estimation of the pose of a car.

VI. Feature sharing applied to face detection and recognition

Feature sharing may be useful in systems requiring different levels of categorization. If we want to build a system performing both class detection (e.g. chairs vs. background) and instance-level categorization (e.g., recognition of specific chairs), a common approach is to use a two stage system: the first stage is built by training a generic class detector (to detect any chair), and the second stage is built by training a dedicated classifier to discriminate between specific instances (e.g., my chair vs. all others).

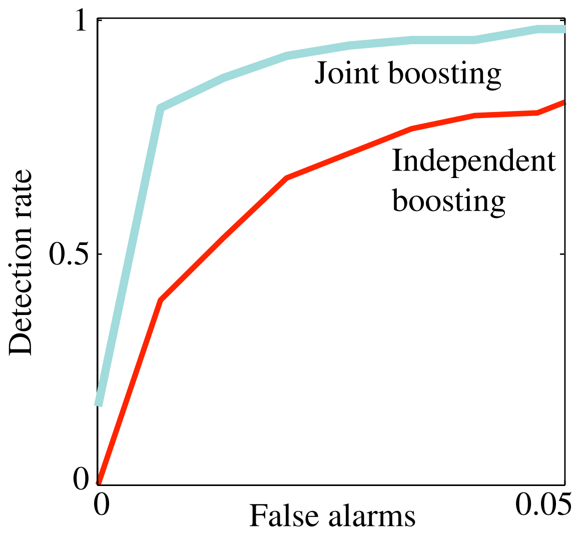


Fig. 21. ROC for view invariant car detection. The graph compares the ROC for the multiview classifier trained using joint boosting for 12 views and using independent boosting for each view. In both cases, the classifier is trained with 20 samples per view and only 70 features (stumps) are used.

By applying the feature sharing approach, we can train one set of classifiers to detect specific instances of objects. The algorithm will find the commonalities between the object instances deriving: 1) generic class features (shared among all instances), and 2) specific class features (used for discriminating among classes). This provides a natural solution that will adapt the degree of feature sharing as a function of intra-class variability.

One example of multiple level of categorization is in the field of face detection and recognition. Most systems for face recognition are built using two stages: the first stage performing object detection and the second one performing face recognition on the patches classified as faces. The face detection stage is built by training a classifier to discriminate between all faces and the background.

To illustrate the feature sharing approach, we have trained a system to do face detection and emotion recognition (the same approach will apply for other intra-class discriminations like person recognition, gender classification, etc.). We use the MacBrain Face Stimulus database (Fig. 23). There are 16 emotions and 40 faces per emotion. We use 5 faces of each class to build the feature dictionary (2000 features). For training we used 20 additional faces

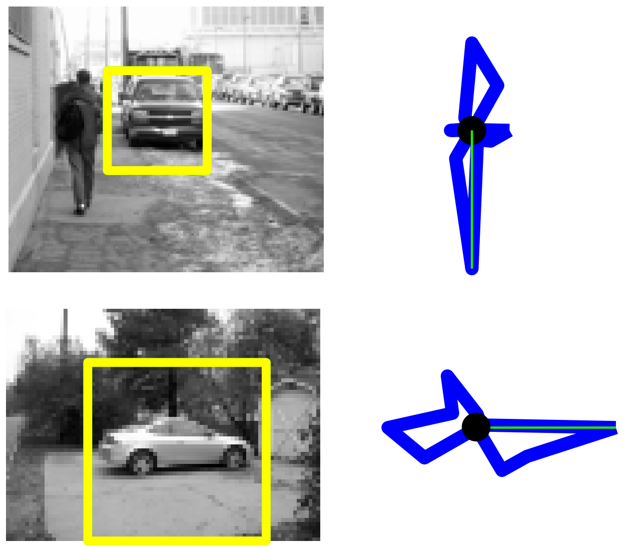


Fig. 22. Examples of estimation of pose of cars. The polar plot corresponds to $H(v, \theta_i)$.

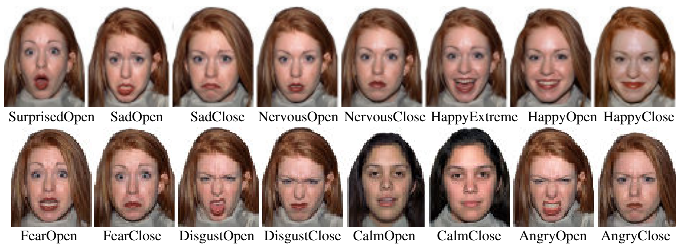
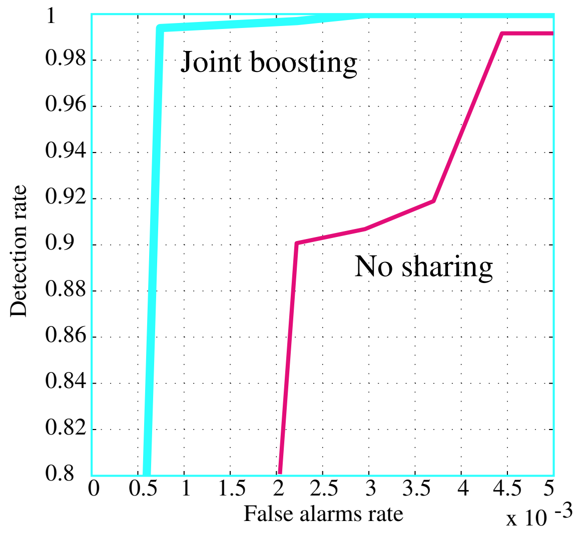


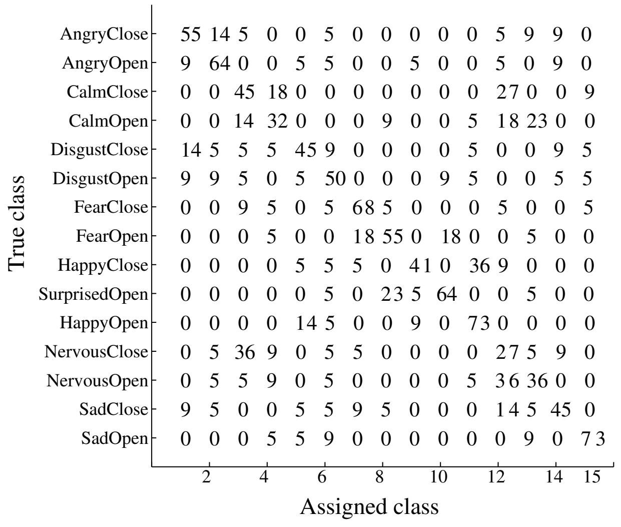
Fig. 23. Example of the emotions used. Development of the MacBrain Face Stimulus Set was overseen by Nim Tottenham and supported by the John D. and Catherine T. MacArthur Foundation Research Network on Early Experience and Brain Development. Please contact Nim Tottenham at tott0006@tc.umn.edu for more information concerning the stimulus set.

and 1000 background patches selected randomly from images. The test is performed on the remaining faces and additional background patches. The joint classifier is trained to differentiate the faces from the background (detection task) and also to differentiate between the different emotions (recognition task).

Fig. 24 shows the features selected and the sharing between the different emotion categories by applying joint boosting. The first 5 features are shared across all classes. Therefore, they contribute exclusively to the task of detection and not to the recognition. For instance, the smiling-face detector will have a collection of features that are generic to all faces, as part of the difficulty of the classification is in the localization of the face itself in a cluttered scene. The training of a specific class detector will benefit from having examples from other expressions. Note that the features used for the recognition (not shared among all classes) also contribute to the detection.

Fig. 25 summarizes the performances of the system on




True class
Assigned class
2
4
6
8
10
12
14
15

AngryClose	55	14	5	0	0	5	0	0	0	0	0	5	9	9	0
AngryOpen	9	64	0	0	5	5	0	0	5	0	0	5	0	9	0
CalmClose	0	0	45	18	0	0	0	0	0	0	0	27	0	0	9
CalmOpen	0	0	14	32	0	0	0	9	0	0	5	18	23	0	0
DisgustClose	14	5	5	5	45	9	0	0	0	0	5	0	0	9	5
DisgustOpen	9	9	5	0	5	50	0	0	0	9	5	0	0	5	5
FearClose	0	0	9	5	0	5	68	5	0	0	0	5	0	0	5
FearOpen	0	0	0	5	0	0	18	55	0	18	0	0	5	0	0
HappyClose	0	0	0	0	5	5	5	0	41	0	36	9	0	0	0
SurprisedOpen	0	0	0	0	0	5	0	23	5	64	0	0	5	0	0
HappyOpen	0	0	0	0	14	5	0	0	9	0	73	0	0	0	0
NervousClose	0	5	36	9	0	5	5	0	0	0	0	27	5	9	0
NervousOpen	0	5	5	9	0	5	0	0	0	0	5	36	36	0	0
SadClose	9	5	0	0	5	5	9	5	0	0	0	14	5	45	0
SadOpen	0	0	0	5	5	9	0	0	0	0	0	0	9	0	73

Fig. 25. This figure evaluates the performances of the joint classifier by splitting both tasks, detection and recognition. (Top) ROC for face detection, and, (Bottom) confusion matrix for emotion classification with 30 rounds of joint boosting and 15 emotion categories. The numbers correspond to percentages.

detection and emotion recognition. The efficiency of the final system will also be a function of the richness of the dictionary of image features used. Here we use image patches and normalize correlation for computing image features as in the previous sections.

VII. Sharing features across locations and scales

In object detection, the classifier is a function $H(v; x, \sigma, c)$, which returns +1 when the object class c is present at location x and with scale σ, and -1 otherwise. If the classifier only uses features that are local to a particular location/scale, it can be written as $H(v; x, \sigma, c) =$

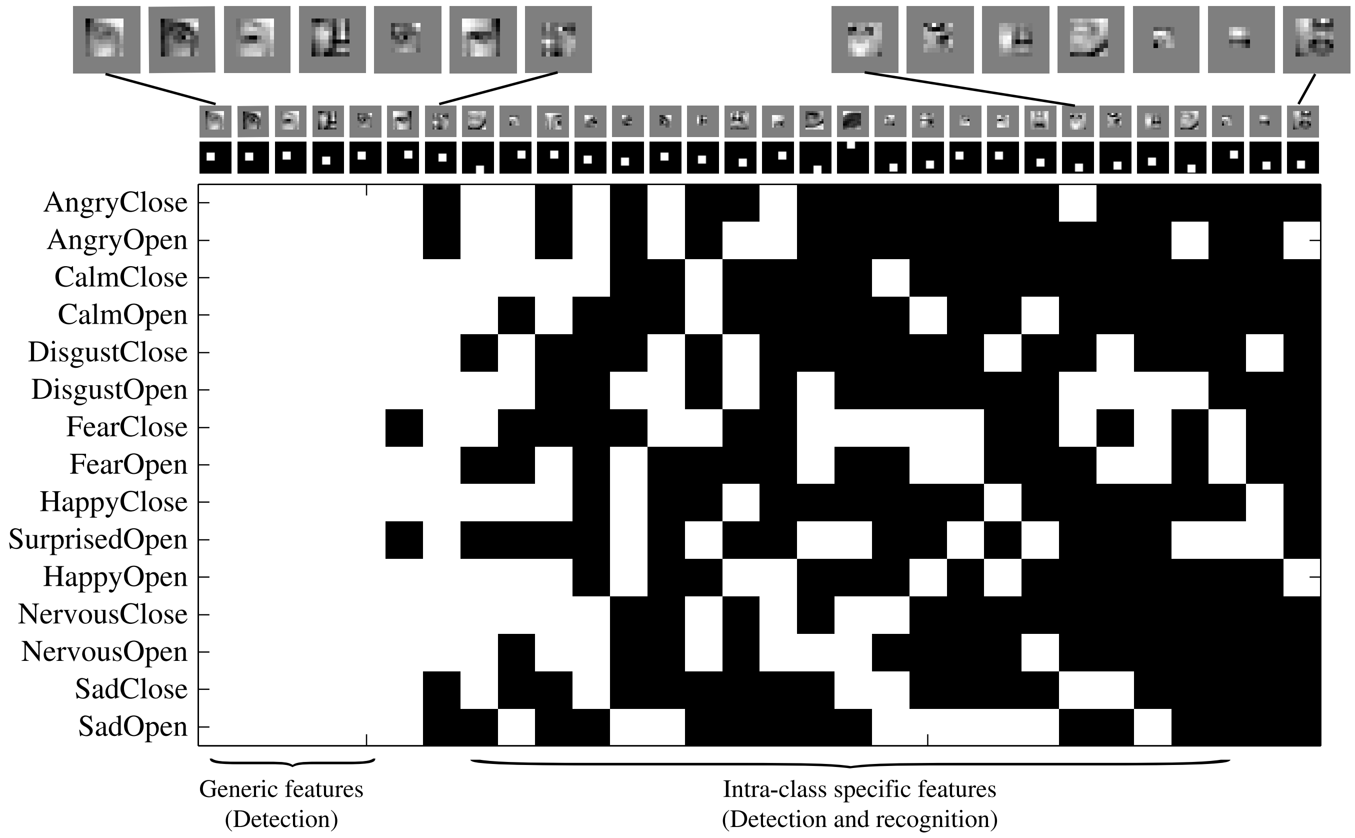


Fig. 24. Sharing matrix for face detection and emotion classification. This matrix shows the features selected using 30 rounds of boosting. The (face) generic features are used to distinguish faces from non-faces (detection task), while the intra-class specific features perform both detection (distinguish faces from the background) and recognition (distinguish among face categories).

$H(v(x, \sigma); c)$. We will call this a "pointwise classifier". However, we can imagine using non-local features that are shared across many locations and scales, as well as or instead of being shared across classes. Hence we can write $H(v; x, \sigma, c) = \sum_{m \in S_{x,\sigma,c}} h_m(v)$, where $S_{x,\sigma,c}$ denotes a partition on classes, locations and scales.

We can build a graph for feature sharing across the different partitions. A pointwise classifier will correspond to one in which the set $S_{x,\sigma,c}$ contains only features that apply at location x, σ. At the other extreme we can have features that are associated with a set that contains all locations and scales in the image. Just as a feature that is shared across all object classes is relatively cheap computationally, so is a feature that is shared across all locations and scales. Such global features can be used to do contextual priming of objects [28]. As locations are more numerous than object classes, having features that are shared across many locations will provide a considerable computational advantage.

VIII. Conclusion

We have introduced a new algorithm, joint boosting, for jointly training multiple classifiers so that they share as many features as possible. The result is a classifier that runs faster (since it computes fewer features) and requires less data to train (since it can share data across classes) than independently trained classifiers. In particular, the number of features required to reach a fixed level of performance grows sub-linearly with the number of classes (for the number of classes that we explored), as opposed to the linear growth observed with independently trained classifiers.

We have applied the joint boosting algorithm to the problem of multi-class, multi-view object detection in clutter. The jointly trained classifier significantly outperforms standard boosting (which is a state-of-the-art method for this problem) when we control for computational cost (by ensuring that both methods use the same number of features). We believe the computation of shared features will be an essential component of object recognition algorithms as we scale up to large numbers of objects.

References

[1] Erin Allwein, Robert Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *J. of Machine Learning Research*, pages 113–141, 2000.

[2] Y. Amit, D. Geman, and X. Fan. Computational strategies for model-based scene interpretation for object detection, 2003.

[3] G. Blanchard and D. Geman. Hierarchical testing designs for pattern recognition. Technical report, U. de Paris-Sud, Mathematiques, 2003.

[4] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[5] O. Dekel and Y. Singer. Multiclass learning by probabilistic embeddings. In *Advances in Neural Info. Proc. Systems*, 2002.

[6] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via ECOCs. *J. of AI Research*, 2:263–286, 1995.

[7] S. Edelman and S. Duvdevani-Bar. A model of visual recognition and categorization.

[8] L. Fei-Fei, R. Fergus, and P. Perona. A bayesian approach to unsupervised one-shot learning of object categories. In *IEEE International Conference on Computer Vision (ICCV'03)*, Nice, France, 2003.

[9] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2003.

[10] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Technical report, Dept. of Statistics, Stanford University, 1998.

[11] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of statistics*, 38(2):337–374, 2000.

[12] T. Hastie and R. Tibshirani. Classification by pairwise coupling. *Annals of Statistics*, 26:451–471, 1998.

[13] B. Heisele, T. Serre, S. Mukherjee, and T. Poggio. Feature reduction and hierarchy of classifiers for fast object detection in video images. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2001.

[14] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 1.

[15] S. Krempp, D. Geman, and Y. Amit. Sequential learning of reusable parts for object detection. Technical report, CS Johns Hopkins, 2002. http://cis.jhu.edu/cis-cgi/cv/cisdb/pubs/query?id=geman.

[16] T. Kubota and C. O. Alford. Computation of orientational filters for real-time computer vision problems i: implementation and methodology. *Real-time Imaging*, 1:261–281, 1995.

[17] B. Leibe and B. Schiele. Analyzing appearance and contour based methods for object categorization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'03)*, Madison, WI, June 2003.

[18] R. Lienhart, A. Kuranov, and V. Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *DAGM*, 2003.

[19] David G. Lowe. Object recognition from local scale-invariant features. In *Proc. of the International Conference on Computer Vision ICCV, Corfu*, pages 1150–1157, 1999.

[20] S. Mahamud, M. Hebert, and J. Shi. Object recognition using boosted discriminants. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'01)*, Hawaii, December 2001.

[21] J. Malik, S. Belongie, J. Shi, and T. Leung. Textons, contours and regions: Cue integration in image segmentation. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 1999.

[22] Bartlett W. Mel. SEEMORE: Combining color, shape and texture histogramming in a neurally-inspired approach to visual object recognition. *Neural Computation*, 9(4):777–804, 1997.

[23] H. Murase and S. Nayar. Visual learning and recognition of 3-d objects from appearance. *Intl. J. Computer Vision*, 14:5–24, 1995.

[24] C. Papageorgiou and T. Poggio. A trainable system for object detection. *Intl. J. Computer Vision*, 38(1):15–33, 2000.

[25] R. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2001.

[26] Henry Schneiderman and Takeo Kanade. A statistical model for 3D object detection applied to faces and cars. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2000.

[27] S.Lazebnik, C. Schmid, and J. Ponce. Affine-invariant local descriptors and neighborhood statistics for texture recognition. In *Intl. Conf. on Computer Vision*, 2003.

[28] A. Torralba. Contextual priming for object detection. *International Journal of Computer Vision*, 52(2):169–191, 2003.

[29] S. Treitel and J. Shanks. The design of multistage separable planar filters. *IEEE Trans. Geosci. Electron.*, 9(1):10–27, 1971.

[30] M. Vidal-Naquet and S. Ullman. Object recognition with informative features and linear classification. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2003.

[31] Paul Viola and Michael Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.