Sharing visual features for multiclass and multiview object detection

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

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. We present a multi-task learning procedure, based on boosted decision stumps, 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 run-time cost of the classifier, is observed to scale approximately logarithmically with the number of classes. The features selected by joint training are generic edge-like features, whereas the features chosen by training each class separately tend to be more object-specific. The generic features generalize better and considerably reduce the computational cost of multi-class object detection.

***Index Terms*— Object detection, interclass transfer, sharing features, boosting, multiclass**

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. Although the general problem remains unsolved, progress has been made on restricted versions of this goal. One succesful special case considers the problem of detecting individual *instances* of highly textured objects, such as magazine covers or toys, despite clutter, occlusion and affine transformations. The method exploits features which are invariant to various transformations, yet which are very specific to a particular object [24], [31]. This can be used to solve tasks such as “find an object that looks just like this one”, where the user presents a specific instance; but it cannot be used to solve tasks such as “find an object that looks like a car”, which requires learning an appearance model of a generic car.

The problem of detecting a generic category of object in clutter is often posed as a binary classification task, namely distinguishing between object class and background class. Such a classifier can be turned into a detector by sliding it across the image (or image pyramid), and classifying each such local window [26], [16], [1]. Alternatively, one can extract local windows at locations and scales returned by an interest point detector and classify these, either as an object or as part of an object (see e.g., [12]). In either case, the classifier will be applied to a large number of image locations, and hence needs to be fast and to have a low false positive rate. Various classifiers have been used, such as SVMs [26], naive Bayes [30], mixtures of Gaussians [12], boosted decision stumps [37], etc. In addition, various types of image features have been considered, ranging from generic wavelets [30], [37] to class-specific fragments [16], [36]. Since it is expensive to compute these features at run-time, many classifiers will try to select a small subset of useful features.

The category-level object detection work mentioned above is typically only concerned with finding a single class of objects (most work has concentrated on frontal and profile faces and cars). To handle multiple classes, or multiple views of a class, separate classifiers are trained and applied independently. There has been work on training a single multi-class classifier, to distinguish between different classes of object, but this typically assumes that the object has been separated from the background (see e.g., [25], [22]).

In this paper [33], we consider the combined problem of distinguishing classes from the background and from each other. This is harder than standard multi-class isolated object classification problems, because the background class is very heterogeneous in appearance (it represents “all other classes”), and is much more likely to appear than the various object classes (since most of the image is background).

The first key insight of our work is that training multiple binary classifiers at the same time needs less training data, since many classes share similar features (e.g., computer screens and posters can both be distinguished from the background by looking for the feature “edges in a rectangular arrangement”). This observation has previously been made in the multi-task learning literature (see e.g., [6], [32]). However, nearly all of this work focuses on feedforward neural networks, whereas we use a quite different kind of classifier, based on boosted decision stumps[29].

The second key insight of our work is that training multiple binary classifiers at the same time results in a much faster classifier at run time, since the computation of many of the features can be shared for the different classes. This observation has previously been made in the neural network literature [20], [21]. However, in these systems, the architecture of the

A. Torralba and W. T. Freeman are at the Department of Electrical Engineering and Computer Science from the Massachusetts Institute of Technology.

K. P. Murphy is at the Departments of computer science and statistics from the University of British Columbia.

network(and hence its computational complexity) is fixed in advance, whereas we effectively learn the structure subject to the constraint that the classifier have a given run-time complexity.

Our extensive empirical results, on 21 object classes, show that the number of features needed when training jointly grows roughly logarithmically with the number of classes (c.f., [18]), whereas independent training shows linear growth. Since the number of features is fewer, the classifier is faster, and the amount of training data (needed to select the features and estimate their parameters) is less. We also show that the features which are chosen when training jointly are generic, edge-like features (reminiscent of V1 cells); this is similar to the results of unsupervised learning methods such as ICA. However, the features chosen when training independently are more class-specific, similar to the results in [36]. Our algorithm will smoothly interpolate between generic and class-specific features, depending on the amount of training data and the bound on the computational complexity of the classifier.

The paper is organized as follows. We describe the multiclass boosting algorithm in Section II, and illustrate its performance on some artificial data sets. In Section III, we show how the algorithm can be used to learn to detect 21 different classes of objects in cluttered, real world images. In Section IV, we show how the algorithm can be used to learn to detect different views of an object class (we focus on cars). The intuition behind this view-based approach is that a car seen from the side is essentially a different visual class than a car seen from the front, but the angles in between share many features in common. In Section VI, we show how the algorithm can be used to perform both face detection and recognition. The idea here is that we first learn to classify a patch as face vs background, and then learn features that discriminate between the face classes. In section VII, we summarize previous work on multiclass object detection and multiclass classifiers. We conclude in Section VIII.

II. Multiclass boosting with feature sharing

A. Boosting for binary classification

We start with a brief review of boosting for binary classification problems [29], [28], [14]. Boosting provides a simple way to sequentially fit additive models of the form

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

where v is the input feature vector, M is the number of boosting rounds, and $H(v) = \log P(z=1|v)/P(z=-1|v)$ is the log-odds of being in class $+1$, where z is the class membership label (± 1). Hence $P(z=1|v) = \sigma(H(v))$, where $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid or logistic function. In the boosting literature, the $h_m(v)$ are often called weak learners, and $H(v)$ is called a strong learner. Boosting optimizes the following cost function one term of the additive model at a time:

$$J = E\left[e^{-zH(v)}\right] \tag{1}$$

The term $zH(v)$ is called the "margin", and is related to the generalization error (out-of-sample error rate). The cost function can be thought of as a differentiable upper bound on the misclassification rate [28] or as an approximation to the likelihood of the training data under a logistic noise model [14]. There are many ways to optimize this function. We chose to base our algorithm on the version of boosting called "gentleboost" [14], because it is simple to implement, numerically robust, and has been shown experimentally [23] to outperform other boosting variants for the face detection task. In gentleboost, the optimization of J is done using adaptive Newton steps, which corresponds to minimizing a weighted squared error at each step. Specifically, at each step m, the function H is updated as $H(v) := H(v) + h_m(v)$, where h_m is chosen so as to minimize a second order Taylor approximation of the cost function:

$$\arg\min_{h_m} J(H + h_m) \simeq \arg\min_{h_m} E\left[e^{-zH(v)}(z - h_m)^2\right] \tag{2}$$

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

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

where N is the number of training examples. How we minimize this cost depends on the specific form of the weak learners h_m. It is common to define the weak learners to be simple functions of the form $h_m(v) = a\delta(v^f > \theta) + b\delta(v^f \leq \theta)$, 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. In this way, the weak learners perform feature selection, since each one picks a single component f. These weak learners are called decision or regression "stumps", since they can be viewed as degenerate decision trees with a single node. We can find the best stump just as we would learn a node in a decision tree: we search over all possible features f to split on, and for each one, we search over all possible thresholds θ induced by sorting the observed values of f; given f and θ, we can estimate the optimal a and b by weighted least squares. Specifically, we have

$$a = \frac{\sum_i w_i z_i \delta(v_i^f > \theta)}{\sum_i w_i \delta(v_i^f > \theta)}, \tag{4}$$

$$b = \frac{\sum_i w_i z_i \delta(v_i^f \leq \theta)}{\sum_i w_i \delta(v_i^f \leq \theta)} \tag{5}$$

We pick the f and θ, and corresponding a and b, with the lowest cost (using Equation 3), and add this weak learner to the previous ones for each training example: $H(v_i) := H(v_i) + h_m(v_i)$. Finally, boosting makes the following multiplicative update to the weights on each training sample:

$$w_i := w_i e^{-z_i h_m(v_i)}$$

This update increases the weight of examples which are missclassified (i.e., for which $z_i H(v_i) < 0$), and decreases the weight of examples which are correctly classified. The overall algorithm is summarized in Figure 1.

```
1) Initialize the weights $w_i = 1$ and set $H(v_i) = 0$, $i = 1..N$.
2) Repeat for $m = 1, 2, \dots, M$
    a) Fit stump: $h_m(v_i) = a\delta(v_i^f > \theta) + b\delta(v_i^f \le \theta)$
    b) Update class estimates for examples $i = 1, \dots, N$: $H(v_i) := H(v_i) + h_m(v_i)$
    c) Update weights for examples $i = 1, \dots, N$: $w_i := w_i e^{-z_i h_m(v_i)}$
```

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

B. Sharing features: basic idea

In the multiclass case, we modify the cost function as in Adaboost.MH [29]:

$$J = \sum_{c=1}^{C} E\left[e^{-z^c H(v,c)}\right] \tag{6}$$

where z^c is the membership label (± 1) for class c and

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

where $H(v, c) = \log P(z^c = 1|v)/P(z^c = -1|v)$. Our algorithm for minimizing this cost function differs from Adaboost.MH [29] in the structure of the weak classifiers h_m. The key idea is that at each round m, the algorithm will choose a subset of classes $S(m)$ that will share a feature and that will have their classification error reduced. The weak classifier is obtained by fitting a binary decision stump as outlined above (some small modifications are required when we share classes, which are explained below.). We consider multiple overlapping subsets of classes, rather than a hierarchical partitioning, because some features may be shared between classes in a way that is not tree-structured (see Figure 2).

We will present two methods for choosing the best subset of classes at each round: the first is based on exhaustive search of all possible subsets, which has complexity $O(2^C)$; the second is based on greedy search (forward selection), which has complexity $O(C^2)$. We will show that, at least on artificial data, the greedy approach is a very good approximation to the exhaustive approach.

C. Toy problem

Before we explain in detail how JointBoost works, we illustrate its behavior on a toy data set. We consider the problem of discriminating among C classes, which consists of C spherical "clouds" of data in D dimensions, embedded in a uniform "sea" of background distractors. So, the classification task requires discriminating among the C classes and also against the background class. In Figure 3, we consider $C = 3$ classes (plus a background class) in $D = 2$ dimensions. In this 2D example, the feature vectors are the projection of the coordinates onto lines at 60 different angles coming from the origin. It is intuitively clear that some features (lines) are useful for separating multiple classes from the background,

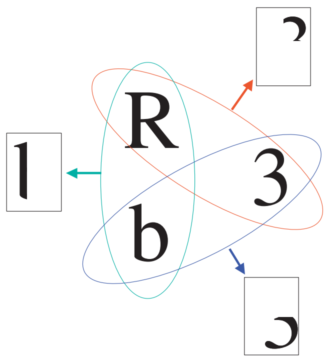


Fig. 2. Objects may share features in a way that cannot be represented as a tree. In this example, we can see how each pair of objects shares a part: the R and the 3 share the crescent-shaped fragment in the top right; the R and the b share the vertical line on the left; and the b and the 3 share the semi-circle-shaped fragment on the bottom right.

and thus can be fruitfully shared. In our formulation, the multiclass classifier is composed by three binary classifiers that can share features (stumps). Each binary problem classifies one class against the others and the background. Our goal is to figure out which features to share amongst which classes.

Figure 4.a shows all subsets of 3 classes arranged in a lattice (ordered by subset inclusion). Let the set at node n in this graph be denoted $S(n)$. At each round, JointBoost will consider each of one of these subsets as a possible candidate to share a stump and will learn a weak classifier for that subset. If we sum up all the weak learners associated with subset $S(n)$, we get a strong learner, which we can denote $G^{S(n)}(v)$. (If subset $S(n)$ was never chosen by the algorithm, then $G^{S(n)}(v) = 0$.) Finally, for each class c, we can find all subsets $S(n)$ that contain c, and sum up their additive models to give the final form of the 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 $G^{S(n)}(v) = \sum_{m=1}^{M_n} h_m^n(v)$.

If we apply the JointBoost algorithm to the data in Fig. 3, but restrict it to 8 rounds (so it can choose exactly 8 features), the result is the model shown in Fig. 4.b. 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: $M_{123} = 3$, $M_{12} = 2$, $M_{23} = 1$, $M_{13} = 0$, $M_1 = 1$, $M_2 = 0$, $M_3 = 1$, so there are no pure boundaries for class 2 in this example (indicated by the blank G^2 square in Figure 4.b). The decomposition is not unique as different choices of functions $G^{S(n)}(v)$ can give the same classifiers $H(v, c)$. 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 (i.e., the number of rounds of boosting).

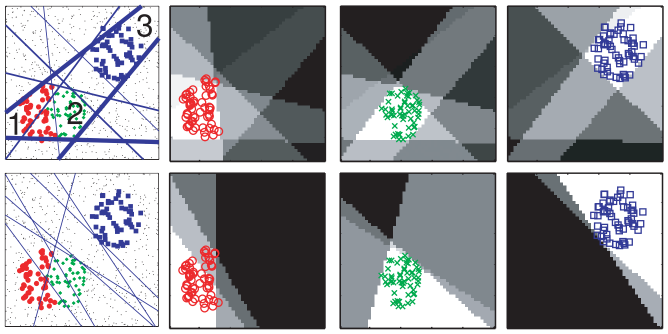


Fig. 3. Illustration of feature sharing (top row) and independent features (bottom row) on a toy problem in which there are three object classes and one background class. 50 samples from each class are used for training, and we use 8 rounds of boosting. Left: The thickness of the lines indicates the number of classes sharing each stump. Right: whiter colors indicate that the class is more likely to be present. Note that for the same computational resources, feature sharing gives better separation of the 3 classes from the background class.

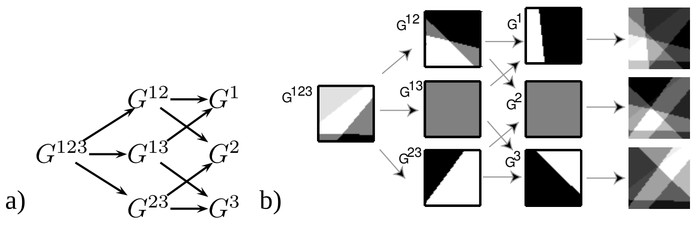


Fig. 4. a) All possible ways to share features amongst 3 classifiers. The sets are shown in a lattice ordered by subset inclusion. The leaves correspond to single classes. b) Decision boundaries learned by all the nodes in the sharing graph for the problem in Fig. 3

D. Shared stumps

We now explain in more detail how JointBoost works. Proceeding as in the regular gentleBoost algorithm, we must solve the following weighted least squares problem at each iteration:

$$J_{wse} = \sum_{c=1}^{C} \sum_{i=1}^{N} w_i^c (z_i^c - h_m(v_i, c))^2 \tag{7}$$

where $w_i^c = e^{-z_i^c H(v_i, c)}$ are the weights for example i and for the classifier for class c. Note that each training example i has C weights, w_i^c, one for each binary problem. It is important to note that the weights cannot be normalized for each binary problem independently, but a global normalization does not affect the results. z_i^c is the membership label (± 1) for example i for class c[1].

For classes in the chosen subset, $c \in S(n)$, we can fit a regression stump as before. For classes not in the chosen subset, $c \notin S(n)$, we define the weak learner to be a class-specific constant k^c. The form of a shared stump is:

$$h_m(v, c) = \begin{cases} a_S & \text{if } v_i^f > \theta \text{ and } c \in S(n) \\ b_S & \text{if } v_i^f \leq \theta \text{ and } c \in S(n) \\ k_S^c & \text{if } c \notin S(n) \end{cases} \tag{8}$$

[1]For each binary classification problem we can consider as negative examples all the other classes and the background or just the background class (in such a case we can set the weights to $w_i^c > 0$ for samples in the class c ($z_i^c = 1$) or in the background class and we set $w_i^c = 0$ for samples i in one of other classes $C - c$).

The purpose of the class-specific constant k_S^c is to prevent a class being chosen for sharing just due to the imbalance between negative and positive training examples. (The constant gives a way to encode a prior bias for each class, without having to use features from other classes that happen to approximate that bias.) Note that this constant changes the way features are shared, especially in the first iterations of boosting. Therefore, in order to add a class to the shared subset we need to have a decrease of the classification error that is larger than just using a constant as weak classifier. This insures that the shared features are really providing additional discriminative information.

At iteration n, the algorithm will select the best stump and a classes subset. For a subset $S(n)$, the parameters of the stump are set to minimize Equation 7. Note that the class labels z_i^c do not change with the shared subset selected. The class labels z_i^c define the C binary classification problems that we are trying to solve jointly. When a stump is shared among several classes, the error for each shared class increases with respect to a stump optimized just for that class. However, because more classes have their classification error reduced when the stump is shared, the total multiclass error decreases (see also section III-E).

Minimizing Equation 7 gives

$$a_S(f, \theta) = \frac{\sum_{c \in S(n)} \sum_i w_i^c z_i^c \delta(v_i^f > \theta)}{\sum_{c \in S(n)} \sum_i w_i^c \delta(v_i^f > \theta)}, \tag{9}$$

$$b_S(f, \theta) = \frac{\sum_{c \in S(n)} \sum_i w_i^c z_i^c \delta(v_i^f \leq \theta)}{\sum_{c \in S(n)} \sum_i w_i^c \delta(v_i^f \leq \theta)}, \tag{10}$$

$$k^c = \frac{\sum_i w_i^c z_i^c}{\sum_i w_i^c} \quad c \notin S(n). \tag{11}$$

Thus each weak learner contains 4 parameters (a, b, f, θ) for the positive class, $C - |S(n)|$ parameters for the negative class, and 1 parameter to specify which subset $S(n)$ was chosen.

Fig. 5 presents the simplest version of the algorithm, which involves a search over all $2^C - 1$ possible sharing patterns at each iteration. Obviously this is very slow. In Section II-E, 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 II-F, we discuss a greedy search heuristic that has complexity $O(C^2)$ instead of $O(2^C)$.

E. Efficient computation of shared regression stumps

To evaluate the quality of a node in the sharing graph, we must find the optimal regression stump, a slow computation, 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_c and b_c for a set of predefined thresholds and for all features, so as to minimize the weighted square error. Then, the parameters a_S and b_S for each threshold and feature at any other internal

```
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:
         $$h_m^n(v_i, c) = \begin{cases} a_S & \text{if } v_i^f > \theta \text{ and } c \in S(n) \\ b_S & \text{if } v_i^f \leq \theta \text{ and } c \in S(n) \\ k^c & \text{if } c \notin S(n) \end{cases}$$
      ii) Evaluate error
         $$J_{wse}(n) = \sum_{c=1}^{C} \sum_{i=1}^{N} w_i^c (z_i^c - h_m^n(v_i, c))^2$$
   b) Find best subset: $n^* = \arg\min_n J_{wse}(n)$.
   c) Update the class estimates
         $$H(v_i, c) := H(v_i, c) + h_m^{n^*}(v_i, c)$$
   d) Update the weights
         $$w_i^c := w_i^c e^{-z_i^c h_m^{n^*}(v_i, c)}$$
```

Fig. 5. Boosting with shared 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.

node can be computed simply as a weighted combination of the parameters at the leaves that are connected with that node. The best regression parameters for a subset of classes S is:

$$a_S(f, \theta) = \frac{\sum_{c \in S} a_c(f, \theta) w_+^c(f, \theta)}{\sum_{c \in S} w_+^c(f, \theta)} \quad (12)$$

$$b_S(f, \theta) = \frac{\sum_{c \in S} b_c(f, \theta) w_-^c(f, \theta)}{\sum_{c \in S} w_-^c(f, \theta)} \quad (13)$$

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

$$\begin{aligned} J_{wse}(n) &= (1 - a_s^2) \sum_{c \in S(n)} w_+^c + (1 - b_s^2) \sum_{c \in S(n)} w_-^c + \\ &+ \sum_{c \notin S(n)} \sum_{i=1}^{N} w_i^c (z_i^c - k^c)^2 \end{aligned} \quad (14)$$

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. 7, for speed.

F. 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 we group classes instead of features.

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 first select 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 keep adding the next best class, until we have added all the classes. We then pick the set, from the C we have considered, with the largest error reduction. This set can have any size between 1 and C.

Since at each step we must consider adding one from $O(C)$ classes, and there are C steps, the overall complexity of this algorithm is $O(C^2)$. This is much better than $O(2^C)$ required for exhaustive search. We can improve the approximation by using beam search, considering at each step the best $N_c < C$ classes.

To compare the exhaustive and greedy search procedures, we return to the toy data shown in Fig. 3. We consider $D = 2$ dimensions but $C = 9$ classes (so that we can afford to consider all possible subsets). For this experiment, the features are the raw coordinate values; we use 25 training samples per class, and 8,000 samples for the background.

Fig. 6.a illustrates the differences between exact search for the best sharing, the best first approximate search, the best pairs only, a random sharing and no sharing. For each search algorithm the graph shows the number of stumps needed to achieve a fixed level of performance (area under the ROC $= 0.95$). 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 to achieve the same performance.

Fig. 6.b illustrates the dependency of the complexity of the classifier as a function of the number of classes when using different sharing patterns. For these 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. What is perhaps a bit surprising is that, even though random sharing exhibits linear complexity, it still performs about as well as the best pair. The reason is that a random sharing will be good for at least two classes at each round (in general, for D classes in D dimensions). 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.

The above scaling results are on low-dimensional artificial data, but the experimental results in Section III show that the algorithm also scales to handle 21 object classes and feature vectors of size 2000.

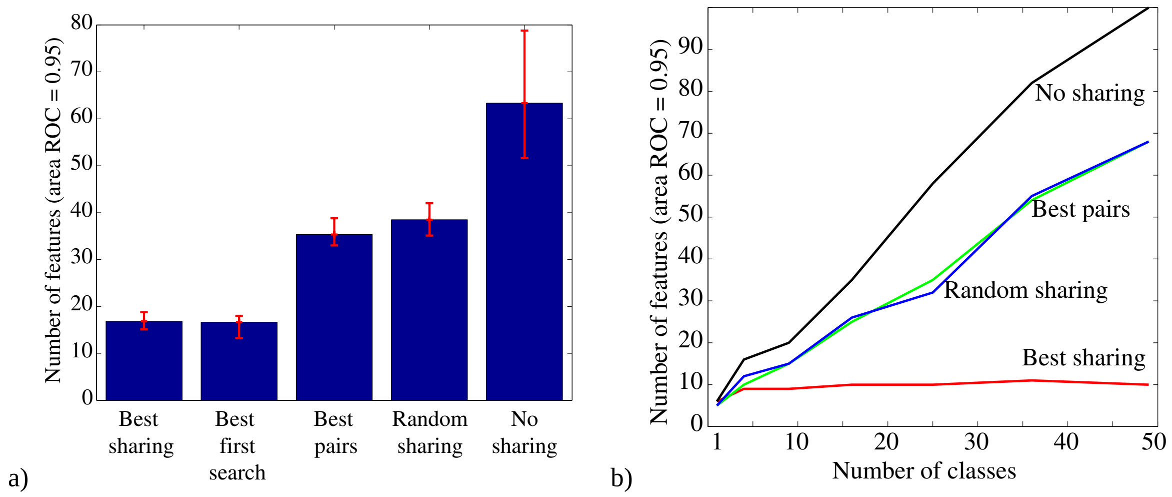


Fig. 6. a) Comparison of number of stumps needed to achieve the same performance (area under ROC equal to 0.95) when using exact search, best-first, best pair, random sharing and no sharing at each round. We use a toy data set with $C = 9$ classes plus a background class in $D = 2$ dimensions. b) Complexity 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).

III. Multiclass object detection

In this section, 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, pedestrians); and heads (which can occur indoors and outdoors). We used hand-labeled images from the *LabelMe* database of objects and scenes [27], available at `labelme.csail.mit.edu`.

A. Features

The features we use are inspired by the fragments proposed by [36]. As in [36], first we build a dictionary of features by extracting a random set of $D = 2000$ patches or fragments from a subset of the training images from all the classes (with objects normalized in scale so that they fit in a bounding box of 32x32 pixels). The fragments have sizes ranging from 4x4 to 14x14 pixels. When we extract a fragment g_f, we also record the location with respect to the object center from which it was taken (within the 32x32 window); this is represented by a binary spatial mask w_f (we fix the mask to be a square of 7x7 pixels centered on the original fragment location). See Figure 7.a for some examples. Once the dictionary is built, for each image we compute the features by performing the following steps for each of the 2000 fragments f:

1) For training, we first scale the images so that the target object fits in a bounding box of 32x32 pixels. We crop the images so that they are not larger than 128x128 pixels. We will use the background around the object to collect negative training samples.
2) Apply normalized cross correlation between each fragment g_f and the training images. Normalized cross correlation can be speed up by approximating each patch g_f with a linear combination of 1D separable filters [35], [19].
3) Perform elementwise exponentiation of the result, using exponent p. With a large exponent, this has the effect of performing template matching. With $p = 1$, the feature vector encodes the average of the filter responses, which are good for describing textures. In this paper, we use $p = 10$; this is good for template matching as it approximates a local maximum operator (although other values of p will be useful for objects defined as textures like buildings, grass, etc.).
4) Convolve the response with the spatial mask w_f (to test if the fragment occurs in the expected location). This corresponds to make each feature to vote for the expected object center. Convolution with the binary, rectangular masks w_f can be implemented in a small number of operations using the integral image [37].

This will give us a very large set of training vectors. To reduce the number we use only a sparse set of locations. From each image in the training set we extract positive training vectors by sampling the feature outputs at the object center and negative training vectors by sampling randomly in the background (Fig. 7). We do not use samples that are inside the object bounding box. For each chosen location, we get a vector of size equal to the number of features in the dictionary. Using 2000 fragments give us a 2000 dimensional feature vector for each location. However, by only using M rounds of boosting, we will select at most M of these features, so the run time complexity of the classifier is bounded by M.

At test time, objects are detected by applying the classifier to the jet of feature responses at each image location. As objects were normalized in scale for the training images, objects are only detected at a normalized scale of 32x32 pixels. Scale invariance is obtained by scanning scale by scaling down the image in small steps. This evaluation of features for all image locations and scales, can be summarized as:

$$v^f(x, y, \sigma) = (w_f * |I_\sigma \otimes g_f|^p) \quad (15)$$

where I_σ is the image at scale σ, g_f is the fragment, w_f is the spatial mask, $\otimes$ represents the normalized correlation, and $*$ represents the convolution operator.

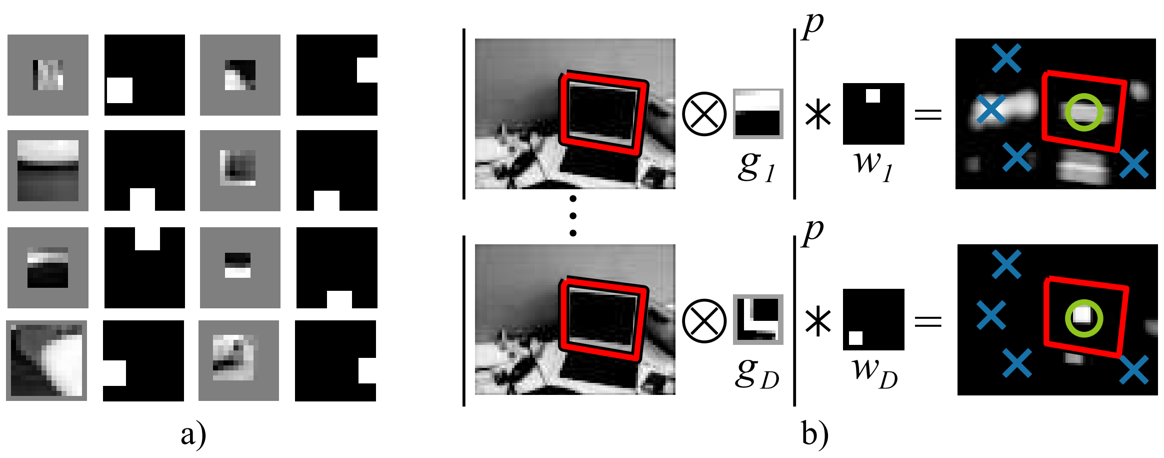


Fig. 7. a) Each feature is composed of a template (image patch on the left) and a binary spatial mask (on the right) indicating the region in which the response will be averaged. The patches vary in size from 4x4 pixels to 14x14. b) Each feature is computed by applying normalize correlation with the template. From each image, we get positive ($z^c = 1$) and negative (background, $z^c = -1\ \forall c$) training samples by sampling the set of responses from all the features in the dictionary at various points in the background and in the center of each target object.

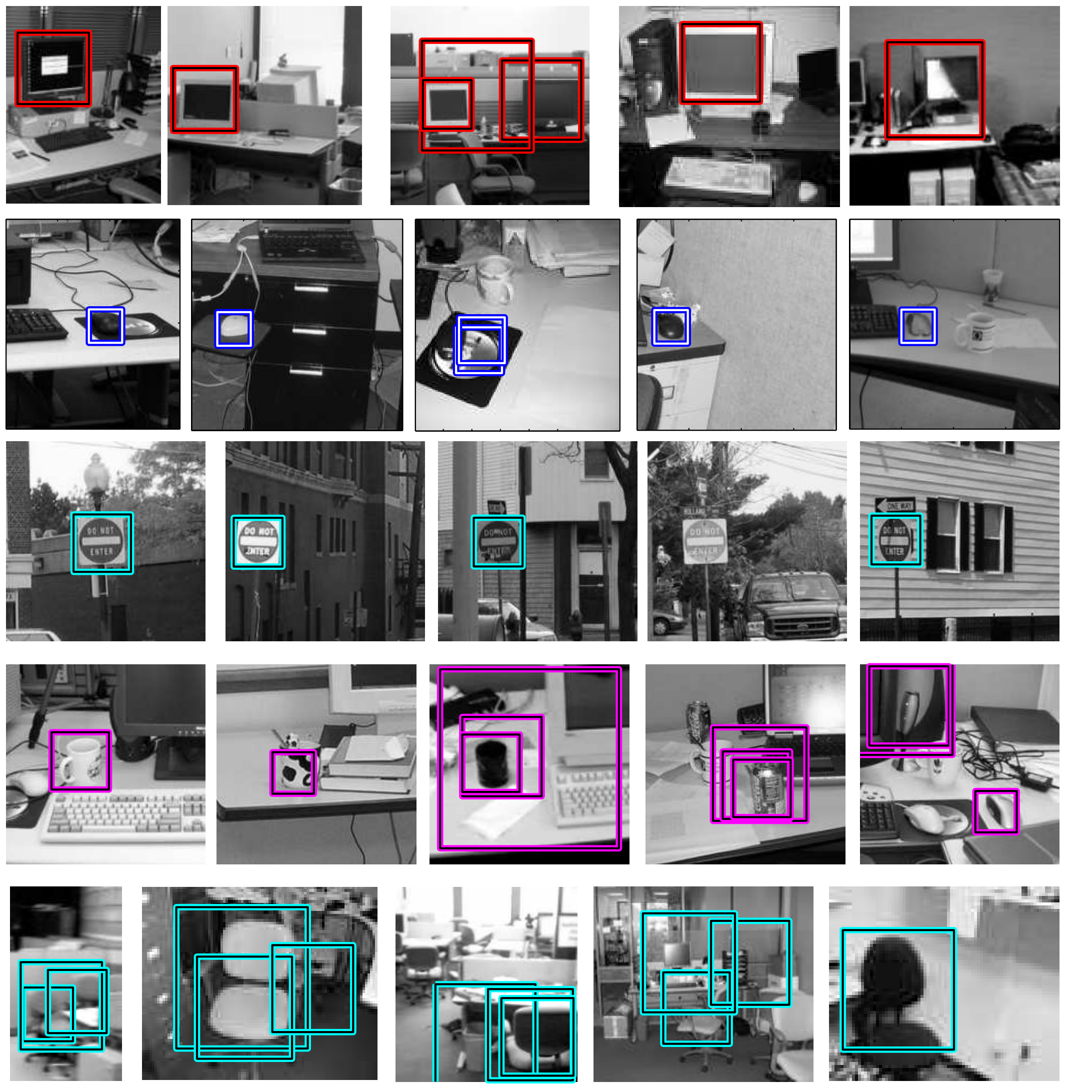


Fig. 8. Examples of typical detections for computer screen, mouse, do-not-enter sign, mug and chairs (results are the first 5 images processed from a typical run). For each row, only the output of one object class detector is shown. The results are obtained training 21 object classes using 50 training samples per class and 1000 background samples. The classifier uses 500 features (rounds of boosting). Images are cropped so that the difficulty of detecting all the object classes is the same independent of their real size. Images have about 180x180 pixels. Detections are performed by scanning the image across locations and scales. Scale is explored by scaling the image with steps of 0.9.

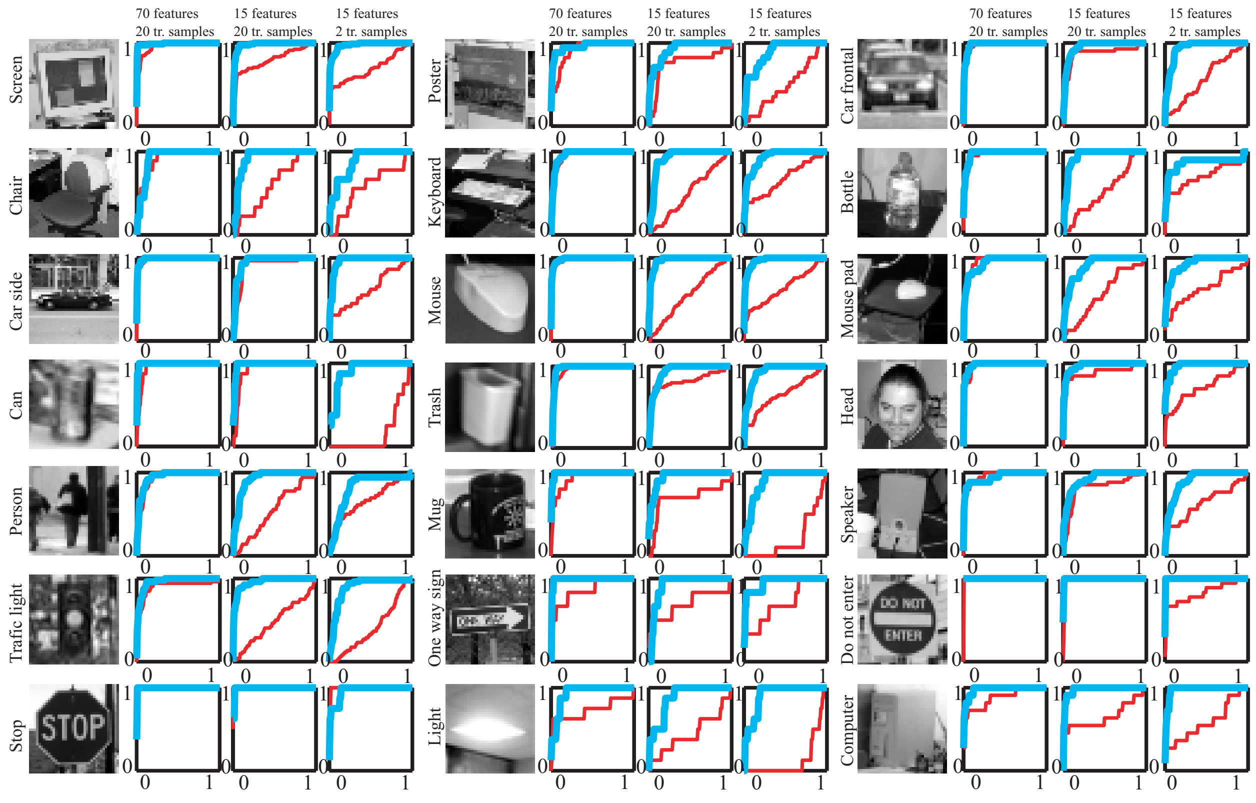
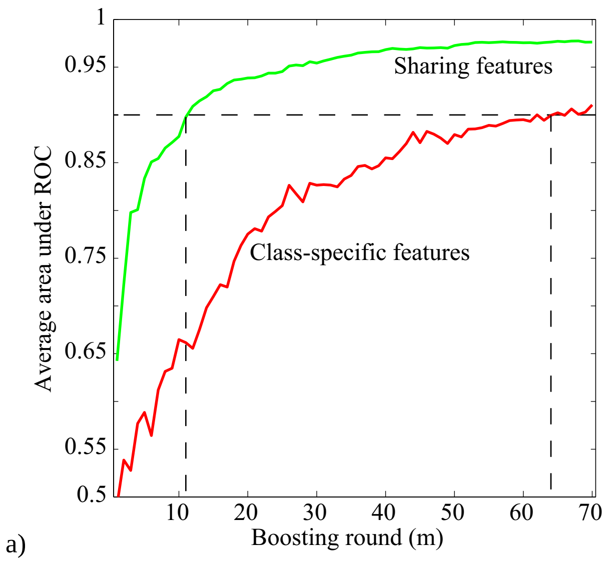


Fig. 9. ROC curves for 21 objects (red (lower curve) = isolated detectors, blue (top curve) = joint detectors). ROC is obtained by running the detector on entire images and sampling the detector output at the location of the target and on the background. For each graph, the horizontal axis is the false alarm ratio and the vertical axis is the ratio of correct detections. 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. In the second and third cases, there are fewer features than classes, so training each class separately will inevitably result in some classifiers performing at chance (shown by diagonal ROC lines).



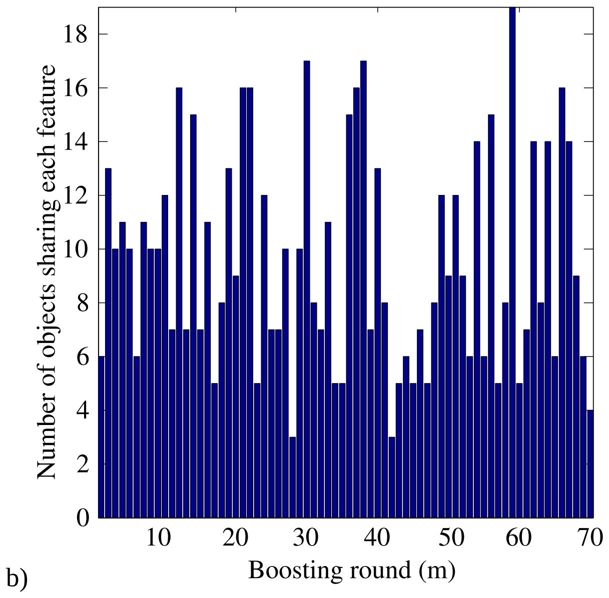


Fig. 10. a) Evolution of classification performance of the test set as a function of number of boosting rounds (or features). Performance is measured as the average area below the ROC across all classes. Chance level is 0.5 and perfect detection for all objects correspond to area$= 1$. Both joint and 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 training for the same performance. b) This graph shows how many objects share the same feature at each round of boosting during training. Note that a feature shared among 10 objects is in fact using $20 * 10 = 200$ training samples.

B. Results on multiclass object detection

Figure 8 shows some sample detection results when running the detectors with shared features on whole images by scanning each location and scale, and finding the local maxima. Figure 9 summarizes the performances of the detectors for each class. For the test, we use an independent set of images. All the detectors have better performances when trained jointly, sometimes dramatically so. When separate classifiers are trained, we require that exactly the same number of features (weak learners) are used in total (summing across classes) as in the joint classifier, to ensure that the run-time complexity of the two approaches is comparable.

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

Figure 10.a show performance of both methods improves as we allow more rounds of boosting. 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 have the same functional form. However, when only a reduced number of rounds are allowed (in order to reduce computational cost), the joint training outperforms the isolated detectors. Furthermore, we expect the relative advantage of joint training to get larger and larger as more classes are added.

C. Feature sharing

To gain some insight into how the algorithm works, it is helpful to examine which features it selects and why. Fig. 11 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 along the feature dimension.

Figure 10.b shows the evolution of the number of objects sharing features for each boosting round. We expected to see that the features chosen initially would be shared by many classes, and the features chosen later would be more class-specific, but this is not what is observed.

Figure 12 shows the final set of features selected (the parameters of the regression stump are not shown) and the sharing matrix that specifies 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

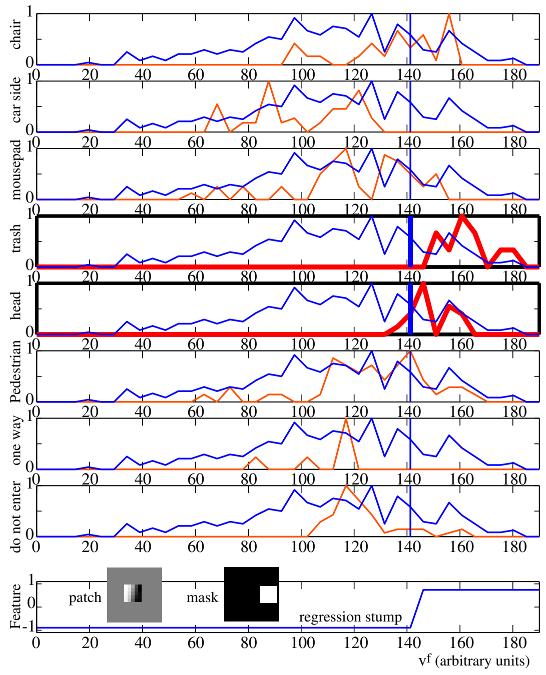


Fig. 11. Example of a shared feature (obtained at round 4 of boosting) between two objects (heads and trash-cans) when training 8 objects jointly. The shared feature is shown at the bottom of the figure. It is defined by an image feature (template and mask) and a regression stump (a, b and θ). For each object, the blue graph shows an empirical approximation to $p(v^f|z^c = -1)$ (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.

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).

We can measure similarity between two object classes by counting the number of features that they have in common and normalizing by the number of features used by each class (normalized correlation). Figure 13 shows the result of a greedy clustering algorithm using this simple 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).

D. Specific vs. generic features

One consequence of training object detectors jointly is in the nature of the features selected for multiclass object detection. When training objects jointly, the system will look for features that generalize across multiple classes. These features tend to be edges and generic features typical of many natural structures, similar to the response properties of V1 cells. Similar results have been obtained using unsupervised learning methods, such as ICA, applied to image patches, but we obtained our results using supervised, discriminative methods (similar to a neural network).

The generality of the features we find is in contrast to the claim in [36] that class-specific features (of intermediate complexity) are best. When training classifiers independently, we find that class-specific features are indeed best, since

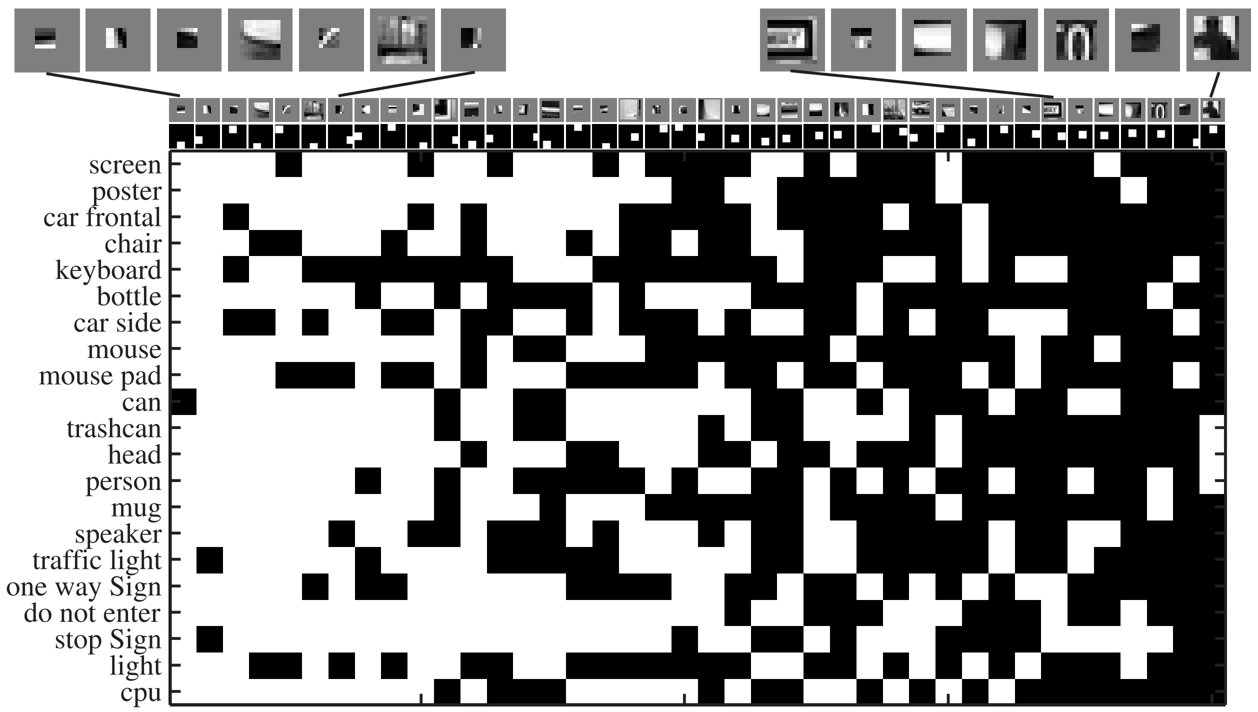


Fig. 12. Matrix that relates features to classifiers, which shows which features are shared among the different object classes. The features are sorted from left to right from more generic (shared across many objects) to more specific. Each feature is defined by one filter, one spatial mask and the parameters of the regression stump (not shown). These features were chosen from a pool of 2000 features in the first 40 rounds of boosting.

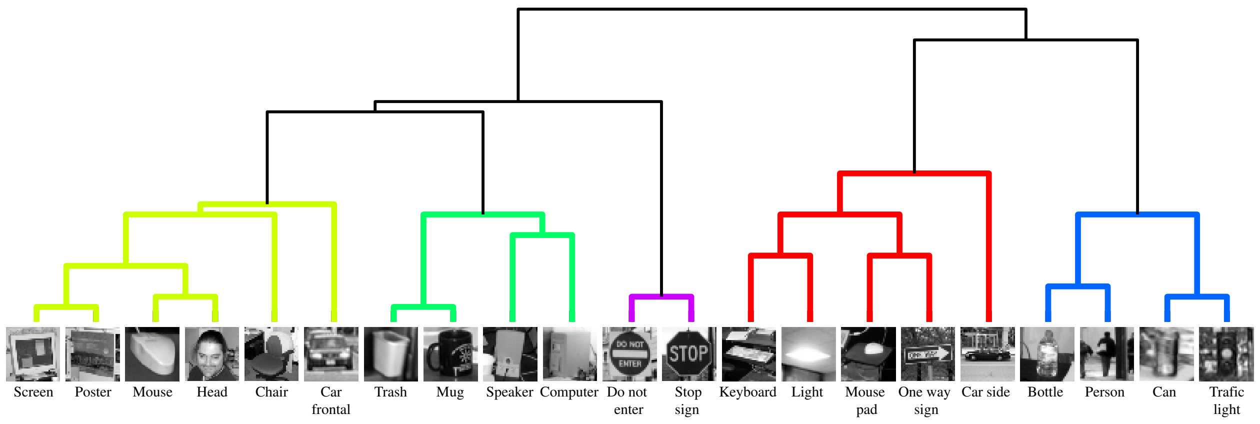


Fig. 13. Clustering of objects according to the number of shared features. Objects that are close in the tree are objects that share more features and therefore share most of the computations when running the classifiers on images. This clustering is obtained by training jointly 21 objects, using 70 stumps and 50 training samples per object.

they are more discriminative and therefore fewer are needed. However, in cases where we cannot afford to have a large number of features, it is better to use generic features, since they can be shared.

Fig. 14 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. When training a single detector using boosting, most of the features are class-specific (the selected features are pieces of the target object despite that the algorithm could chose pieces coming from other 20 object categories) and behave like a template matching detector (see Fig. 14b). 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. 14c) are more generic (configuration of edges) which can be reused by other objects.

E. The number of features needed is approximately logarithmic in the number of classes

One important consequence of feature sharing is that the number of features needed grows sub-linearly with respect to the number of classes. Fig. 15.a 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

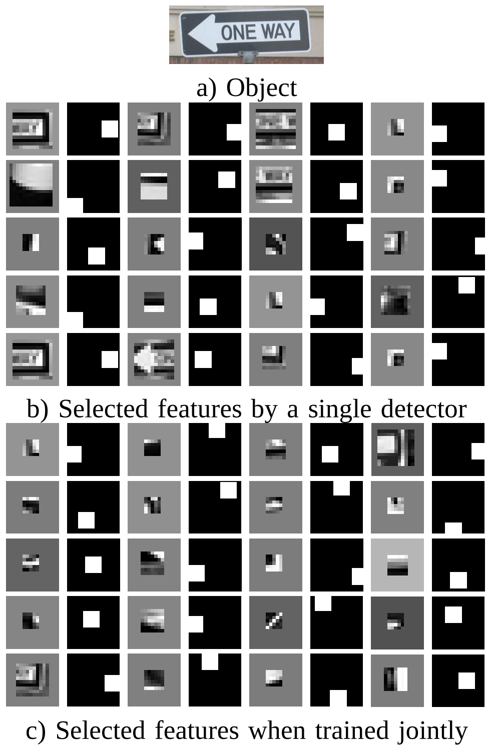

ONE WAY
a) Object
b) Selected features by a single detector
c) Selected features when trained jointly

Fig. 14. 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 trained jointly with 20 other classes, the system learns more generic, wavelet-like filters.

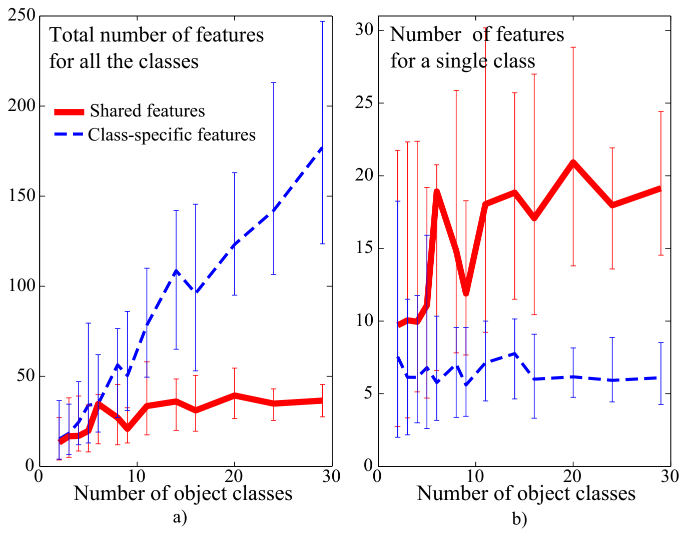


Fig. 15. Comparison of the efficiency of class-specific and shared features to represent many object classes (in this experiment we used 29 object classes by adding to previous 21 classes also frontal faces, parking meter, pot, paper cup, bookshelf, desk, laptop, and fire hydrant). a) Total number of features needed to reach a given classification performance for all the objects (area under the ROC equal to 0.95). The results are averaged across 20 training sets and different combinations of objects. Error bars correspond to 80% interval. As we increase the number of objects to be represented the number of features required to keep performance constant increase linearly for class-specific features and sub-linearly for shared features. b) Number of features allocated for each object class. When sharing features, the features become less informative for a single class, and we therefore need more features per class to achieve the same performance compared to using class-specific features.

complexity grows linearly, as expected. However, when shared features are used, the complexity seems to grow as $\log(C)$. (A similar result has been reported by Krempp, Geman and Amit ([18]) using character detection as a test bed.)

When the system is required to represent an increasing number of object categories, each shared feature becomes less informative for a single object class and, therefore, more features are required for achieving the same detection performance than if we were using class-specific features (Fig. 15.b). However, the fact that we can allocate more features for each object by reusing features from other object classes results in a reduced set of features (Fig. 15.a). Fig. 15.b explains why class-specific features are the preferred representation when studying representations for single object classes. Although this is the goal of some computer vision applications (e.g., car detection), the human visual system is confronted with a more general multiclass object recognition problem.

Both graphs in Fig. 15 show a trade-off between the efficiency of the multiclass object representation and the representation of a single object class. A useful strategy would be to devote class-specific features for classes of special interest. For instance, faces play an important role in human vision and area IT contains cells selective for faces and parts of faces. Face-specific features emerge when we indicate to the algorithm that a larger efficiency is required for that object class (this is done by increasing the penalty of classification errors for the face-class). The resulting visual dictionary contains generic features (shared across many object classes) and face-specific features devoted to an efficient encoding of faces (see Section VI).

F. 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.

IV. MULTIVIEW OBJECT DETECTION

When building view invariant object detectors, the standard approach is to discretize the space of poses, and to implement a set of binary classifiers, each one tuned to a particular pose (e.g., [30]). In this section, we discuss how to train a single multiview classifier that exploits features that are shared across 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, we find that many of the views are quite similar, and hence can share many features.

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

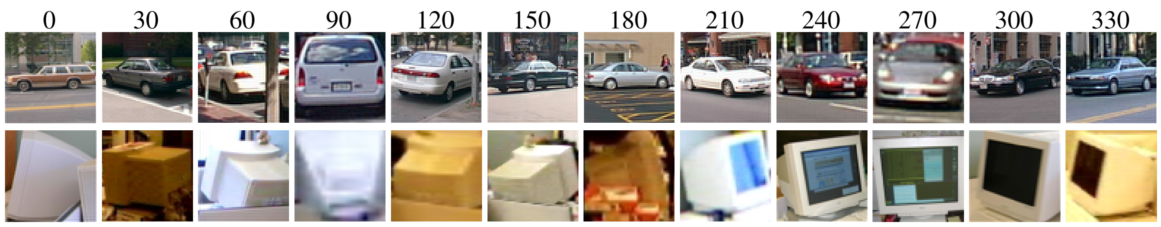
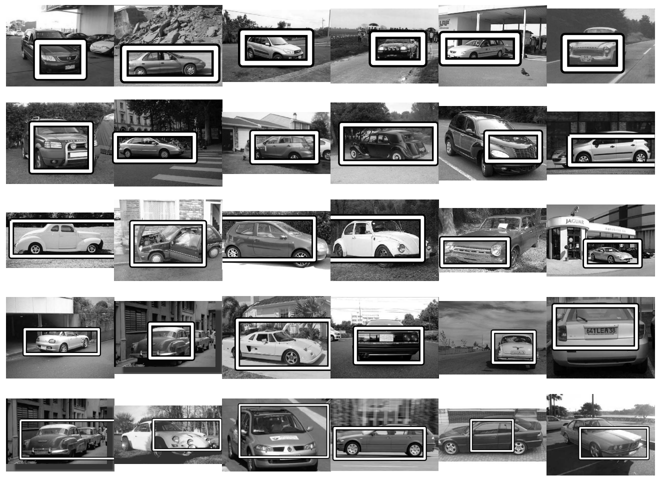


Fig. 16. Examples of pose variations for cars and screens from the LabelMe dataset (the angles are approximate).

a) 

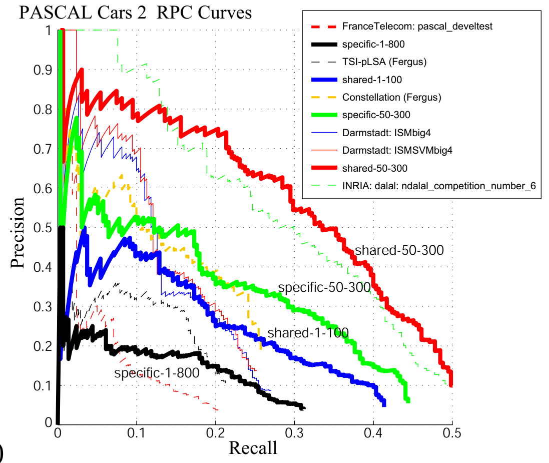
b) 

Fig. 17. a) Detection results on images from the PASCAL collection (cars test set 2, [8]). The classifier is trained on 12 views of cars from the LabelMe dataset (50 positive examples for each view and 12860 background samples) and uses 300 shared features. The detection results are organized according to the confidence of the detector (from high precision/low recall to low precision/high recall). The first row are randomly selected among the most confident detections. Each row represents a different point in the precision-recall curve. b) Precision-recall curves comparing our algorithm with algorithms evaluated during the PASCAL challenge.

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. Our algorithm will share features as much as possible, but only if it does not hurt performance.

Fig. 17 shows the detection results obtained on the PASCAL dataset [8] which contains a challenging set of cars with multiple views. We trained a set of classifiers $H(v, c, \theta_i)$, for the car class 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 $\theta = argmax_{\theta_i} \{H(v, c, \theta_i)\}$. Fig. 17.a shows some detection results ranked according to confidence of the detector. The different aspect ratios of the bounding boxes correspond to the hypothesized car orientations.

Fig. 17.b compares performances with respect to other algorithms from the PASCAL challenge [8]and also from [11]. Our algorithm is evaluated in four versions: 1) one training sample per view, 800 features (rounds of boosting), and no sharing (referenced in the figure as *specific-1-800*), 2) one training sample/view, 100 features, and sharing (*shared-1-100*), 3) 50 training samples/view, 300 features, and no sharing (*specific-50-300*), and 4) 50 training samples/view, 300 features with sharing (*shared-50-300*). Versions 1 and 2 evaluate the ability of the algorithms to generalize from few training examples (note that without sharing features, generalization is poor and it is not a function of how many features are used by the classifier, see next section). Versions 3 and 4 evaluate performances for same computational cost. Note that if the algorithm can use as much training data as he wants, and use as many computations as needed, then there will not be any difference between sharing and no sharing features in this framework.

V. Learning from few examples: multiclass vs multiview

Another important consequence of joint training is that the amount of training data required is reduced. Fig. 9 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. In the case where we are training C object class detectors and we have N positive training examples for each class, by jointly training the detectors we expect that the performance will be equivalent to training each detector independently with N^e positive examples for each class, with $N \leq N^e \leq NC$. The number of equivalent training samples N^e will depend on the degree of sharing between objects.

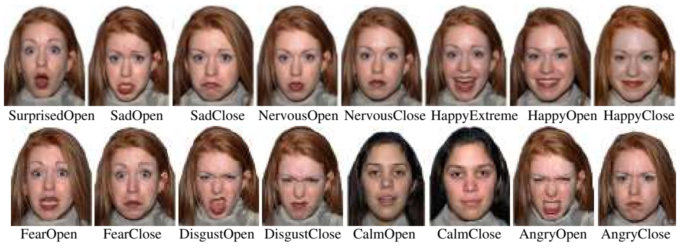


Fig. 19. Example of the emotions used.

To get an estimate of how much larger N^e is compared to N, we ran two experiments in which the classes have different degrees of similarity. In the first experiment, we used 12 different object classes; in the second, we use 12 different views of a car (see previous section). For this comparison, we used 600 features in the dictionary, and 1000 negative examples in the two experiments. We used for training and test images from the LabelMe dataset.

Intuitively, we expect that more features will be shared in the multi*view* case than in the multi*class* case. The experiment confirms this intuition. Specifically, we find that in the multiclass case, each feature was shared amongst 5.4 classes on average, whereas in the multiview case, each feature was shared amongst 7 classes on average. In Fig. 18, we see that that in the multiclass case, $N^e \approx 2.1N$ (i.e., we need to double the size of the training set to get the same performance out of class-specific features), and that in the multiview case, $N^e \approx 4.8N$ (i.e., joint training effectively increases the training set by almost a factor of 5).

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 to perform both class detection (e.g. faces vs. background) and instance-level categorization (e.g., recognition of specific faces), a common approach is to use a two stage system: the first stage is built by training a generic class detector (to detect any face), and the second stage is built by training a dedicated classifier to discriminate between specific instances (e.g., my face vs. all others).

By applying the feature sharing approach, we can train one classifier to solve both tasks. The algorithm will find the commonalities between the object instances, deriving generic class features (shared among all instances) and 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.

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[2] database (Fig. 19). 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 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. 20 shows the features selected and the sharing between the different emotion categories. 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 (i.e., not shared among all classes) also contribute to the detection.

Fig. 21 summarizes the performances of the system on 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 normalized correlation for computing image features, as in the previous sections.

Recently it has become popular to detect objects by detecting their parts, and checking that they satisfy certain spatial constraints (see e.g., [12], [10]). Our algorithm implicitly does this: the spatial mask is a way of requiring that the fragment occurs in the desired place. However, the fragments that are chosen do not have any special semantic meaning [36]. For example, Fig. 20 shows the features we learn for faces; they do not have a clean correspondence with nameable parts like eyes, nose, mouth, etc.

VII. Related work

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

A. *Multi-class object detection*

There has been a large amount of work on object detection and classification. Here we only mention results that are concerned with multi-class object detection in clutter.

Perhaps the closest previous work is by Krempp, Geman and Amit [18]. They 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

[2]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.

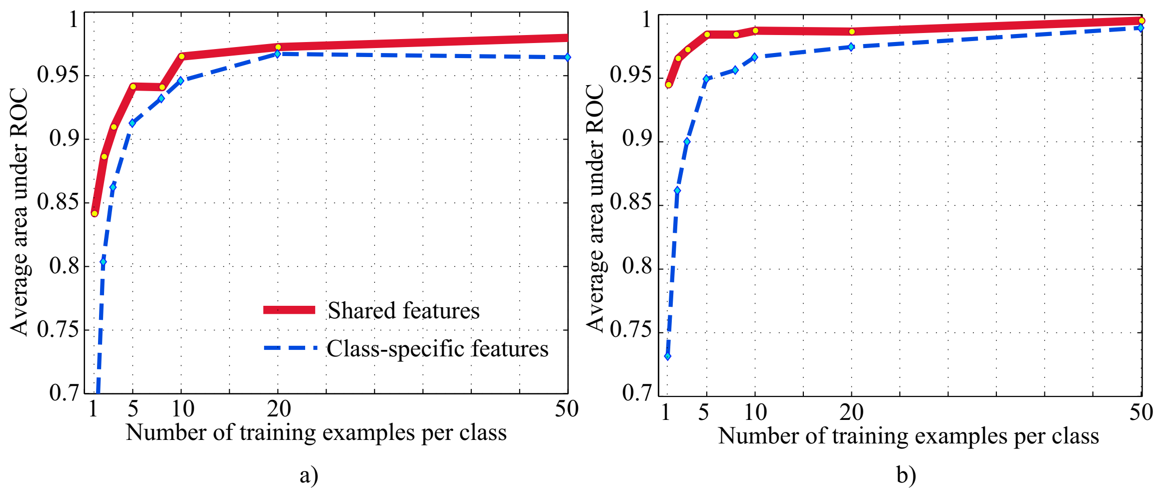


Fig. 18. Detection performance as a function of number of training examples per class. (a) 12 objects of different categories. (b) 12 views of the same object class. Sharing features improves the generalization when few training samples are available, especially when the classes have many features in common (case b). The boosting procedure (both with class-specific and shared features) is run for as many rounds as necessary to achieve maximal performance on the test set.

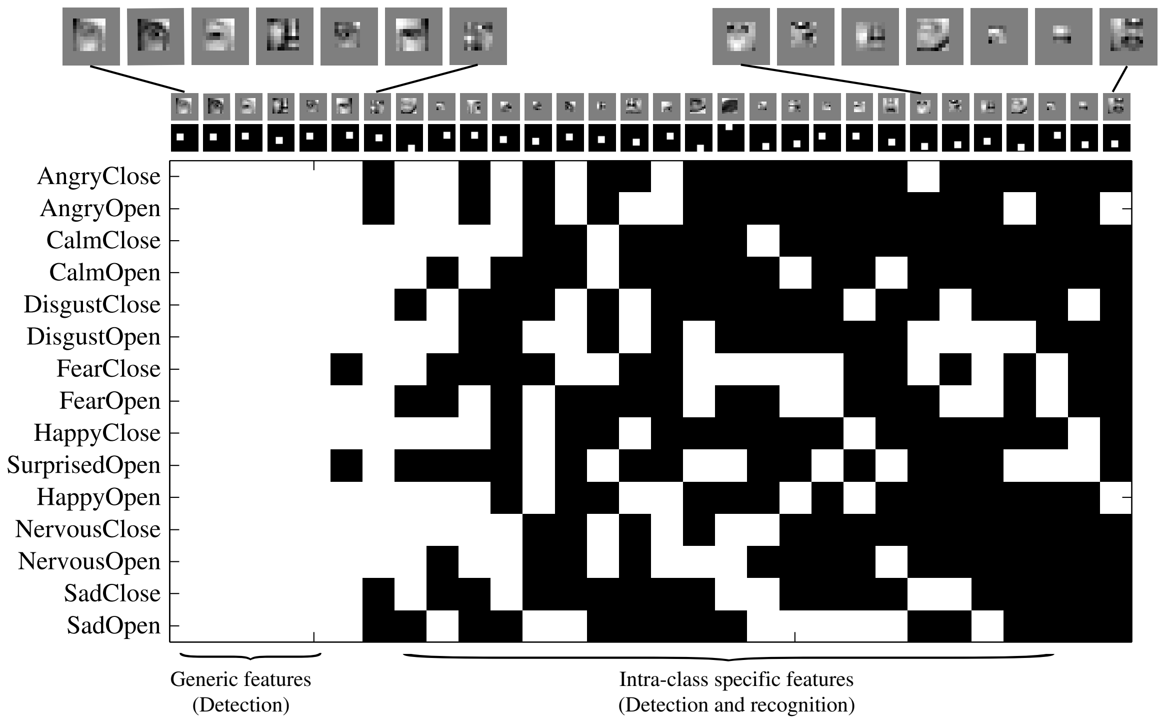
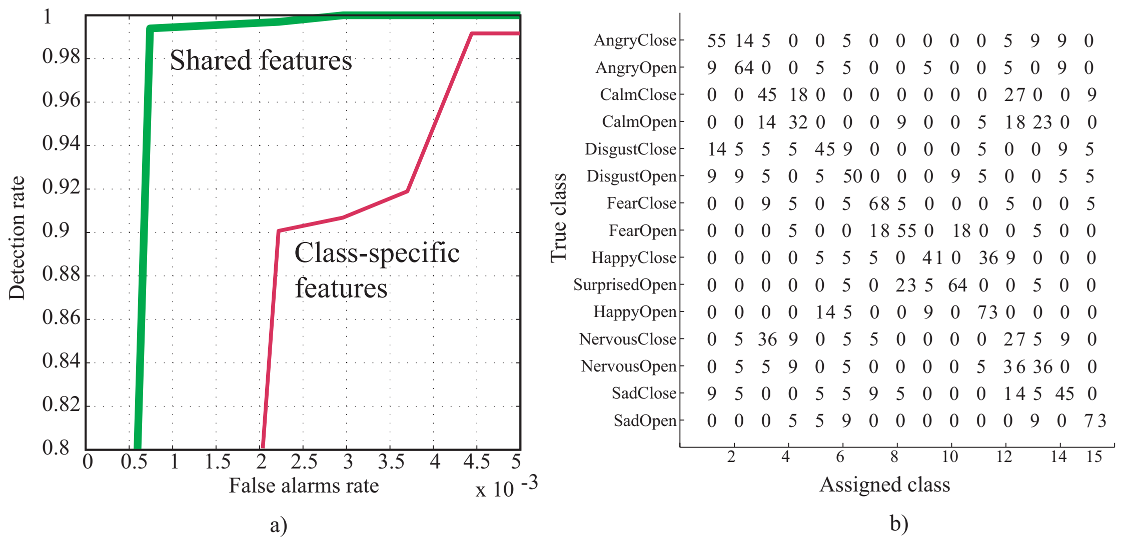


Fig. 20. Sharing matrix for face detection and emotion classification. This matrix shows the features selected using 30 rounds of boosting. The (face) generic features are used to distinguish faces from non-faces (detection task), while the intra-class specific features perform both detection (distinguish faces from the background) and recognition (distinguish among face categories). Here, the degree of sharing is larger than the sharing obtained in the multiclass and multiview experiments.

have found. However, they do not jointly optimize the shared features, and they have not applied their technique to real-world images.

A related piece of work is by Amit, Geman and Fian [3]. They describe a system for multiclass and multi-pose object detection 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. Their CTF search yields candidate locations which are then validated using a generative model.

Fei-Fei, Fergus and Perona [9] 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, this is a generative model, not a discriminative one, and has run-time complexity $\binom{d}{N}$, where d is the number of interest point detections and N is the number of model parts. Hence it is too expensive to detect objects in



	1	2	3	4	5	6	7	8	9	10	11	12	13	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. 21. This figure evaluates the performances of the joint classifier by splitting both tasks, detection and recognition. a) ROC for face detection and, b) confusion matrix for emotion classification with 30 shared features and 15 emotion categories. The numbers correspond to percentages.

really cluttered images.

LeCun, Huang, and Bottou [21] use Convolutional Neural Networks in order to learn to classify several toy objects on backgrounds of moderate complexity. Since the hidden layers are shared by all the classes, they learn common features. We discuss this in more detail below, when we discuss multi-task learning.

More recently, Bernstein and Amit [5] show that one can use clustering (EM applied to a mixture of bernoulli-product models) to discover 'features', or little patches, which can then serve as a universal dictionary for subsequent generative classifiers. In particular, the codebook or dictionary is constructed by clustering patches of binary oriented-edge filtered images; new images are then recoded in terms of which codeword occurs where; a new mixture model, one per class, is then fit this transformed data. However, the dictionary of patches is shared across classes. They demonstrate results on handwritten digits, Latex symbols and the UIUC car-side dataset. Transferring knowledge between objects to improve generalization has also been studied in several recent papers [4], [17], [34].

B. *Multi-class classification*

As mentioned in the introduction, the insight that learning to solve multiple tasks at once is easier than solving them separately has been exploited in the field of "multiple task learning" [6] or "inductive transfer" [32]. The vast majority of this work has focused on the case where the classifier to be learned is a feedforward neural network. In this case, the hidden layer is naturally shared amongst the output classes.[3]

The algorithm proposed in this paper is also related to the idea of error correcting output codes (ECOC) developed by Dietterich and Bakiri [7]. This is a way of converting binary classifiers into multi-class classifiers [2]. The idea of ECOC 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. A binary classifier is fit 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, each column classifier is applied to produce a bit-vector, $(f_1(v), \dots, f_n(v))$, where n is the number of columns. The estimated class label is the one corresponding to the row which is closest in Hamming distance to this bit-vector.

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 [15]. (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. [2] show that

[3]An additive model of boosted stumps is like a 2 layer perceptron, where the m'th hidden unit acts like a weighted linear threshold unit: $h_m(v) = a\delta(v^f > \theta) + b$. The main difference from standard multi-layer perceptrons is the learning procedure: instead of learning all parameters at once using backpropagation (gradient descent), the parameters are learned sequentially using weighted least squares plus exhaustive search (although boosting can be viewed as gradient descent in a function space [13].) In practice, boosting is orders of magnitude faster than backprop. It is also more general in the sense that the weak learners do not have to be simple linear threshold units (decision stumps).

the popular one-against-all approach is often suboptimal, but that the best code matrix is problem dependent. Our algorithm learns the best possible subset to use at each round. Another 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.

VIII. Conclusion

We have introduced an algorithm for multi-class object detection that shares features across objects. 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, as opposed to the linear growth observed with independently trained classifiers.

We have applied the 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 object classes.

IX. Acknowledgments

We acknowledge support from NSF contract IIS-0413232, the National Geospatial-Intelligence Agency (NGA-NEGI), DARPA contract DABT63-99-1-0012, and the Nippon Telegraph and Telephone Corporation as part of the NTT/MIT Collaboration Agreement.

References

[1] S. Agarwal, A. Awan, and D. Roth. Learning to detect objects in images via a sparse, part-based representation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(11):1475–1490, 2004.

[2] E. Allwein, R. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *J. of Machine Learning Research*, pages 113–141, 2000.

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

[4] E. Bart and S. Ullman. Cross-generalization: learning novel classes from a single example by feature replacement. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2005.

[5] E. Bernstein and Y. Amit. Part-based statistical models for object classification and detection. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2005.

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

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

[8] M. Everingham, A. Zisserman, C. Williams, L. Van Gool, M. Allan, C. Bishop, O. Chapelle, N. Dalal, T. Deselaers, G. Dorko, S. Duffner, J. Eichhorn, J. Farquhar, M. Fritz, C. Garcia, T. Griffiths, F. Jurie, D. Keysers, M. Koskela, J. Laaksonen, D. Larlus, B. Leibe, H. Meng, H. Ney, B. Schiele, C. Schmid, E. Seemann, J. Shawe-Taylor, A. Storkey, S. Szedmak, B. Triggs, I. Ulusoy, V. Viitaniemi, and J. Zhang. The 2005 pascal visual object classes challenge. In *First PASCAL Challenges Workshop*. Springer-Verlag, 2005.

[9] L. Fei-Fei, R. Fergus, and P. Perona. A bayesian approach to unsupervised one-shot learning of object categories. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2003.

[10] P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. *Intl. J. Computer Vision*, 61(1), 2005.

[11] R. Fergus. *Visual Object Category Recognition*. PhD thesis, University of Oxford, 2005.

[12] 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.

[13] J. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2001.

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

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

[16] 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.

[17] M. Fink K. Levi and Y. Weiss. Learning from a small number of training examples by exploiting object categories. In *Workshop of Learning in Computer Vision*, 2004.

[18] 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.

[19] 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.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[21] Y. LeCun, Fu-Jie Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of CVPR'04*. IEEE Press, 2004.

[22] 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.

[23] R. Lienhart, A. Kuranov, and V. Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *DAGM 25th Pattern Recognition Symposium*, 2003.

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

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

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

[27] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. Technical Report 025, MIT AI Lab, 2005.

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

[29] R. Schapire and Y. Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

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

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

[32] S. Thrun and L. Pratt, editors. *Machine Learning. Special issue on Inductive Transfer*. 1997.

[33] A. Torralba, K. Murphy, and W. Freeman. Sharing features: efficient boosting procedures for multiclass object detection. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 762-769, 2004.

[34] E. Sudderth, A. Torralba, W.T. Freeman, and A. Willsky. Learning hierarchical models of scenes, objects, and parts. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2005.

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

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

[37] P. Viola and M. Jones. Robust real-time object detection. *Intl. J. Computer Vision*, 57(2):137–154, 2004.