Shared Features for Multiclass Object Detection

Antonio Torralba[1], Kevin P. Murphy[2], and William T. Freeman[1]

[1] Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, USA
[2] Departments of computer science and statistics University of British Columbia, Canada

Abstract. We consider the problem of detecting a large number of different classes of objects in cluttered scenes. We present a 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). Shared features, emerge in a model of object recognition trained to detect many object classes efficiently and robustly, and are preferred over class-specific features. Although that class-specific features achieve a more compact representation for a single category, the whole set of shared features is able to provide more efficient and robust representations when the system is trained to detect many object classes than the set of class-specific features. Classifiers based on shared features need 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").

1 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 [9,15]. 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 [1,6,20]. 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., [4]). 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 [17], naive Bayes [22], mixtures of Gaussians [4], boosted decision stumps [26], etc. In addition, various types of image features have been considered, ranging from generic wavelets [21,26] to class-specific fragments [6,25]. 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., [12,16]).

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 [24] 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., [3,23]). 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[19]. Transfering knowledge between objects to improve generalization has also been studied in several recent papers [2,13,22].

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 [10,11]. However, in these systems, the architecture of the network (and hence its computational complexity) is fixed in advance, whereas we effectively learn the structure.

2 Sharing Features

As objects tend to share many properties, an efficient visual dictionary of objects should capture those commonalities. Since objects are typically embedded in cluttered backgrounds, the representations have to be robust enough to allow for reliable discrimination between members of an object class and background distractors (non-objects). Here we show that shared features emerge in a model of object recognition trained to detect many object classes efficiently and robustly, and are preferred over class-specific features. Note that edge features emerge here from a visual recognition task, rather than from a statistical criterion such as sparse coding or maximizing statistical independence. We show that, although that class-specific features achieve a more compact representation for a single category, the whole set of shared features is able to provide more efficient and robust representations when the system is trained to detect many object classes than the set of class-specific features.

Fig. 1 illustrates the difference between two representations for objects. The first representation (Fig. 1a-left), obtained when training a set of classifiers to detect each object independently, is based on class-specific features of intermediate complexity, which have been shown to maximize the information delivered about the presence of an object class [25]. One drawback of class-specific features is that they might be too finely tuned, preventing them from being useful for other objects classes. The second representation is obtained when training the system to detect 29 object classes by allowing the classifiers to share features. The resulting representation is based on a vocabulary of shared visual features where each feature is used by a subset of the 29 object classes. Each object is represented as configurations of simple features that resemble edge and line detectors instead of relying on configurations of class-specific features.

Our learning algorithm, based on multiclass Boosting [19], is an iterative procedure that adds one feature at each step in order to build a dictionary of visual features. Each feature is found by selecting, from all possible class groupings and features, the combination that provides the largest reduction of the multiclass error rate. The feature added in the first iteration will have to be as informative as possible for as many objects as possible, since only the object classes for which the feature is used will have their error rate reduced. In the second iteration the same selection process is repeated but with a larger weight given to the training examples that were incorrectly classified by the previous feature. Once the second feature is selected, new weights are given to each training example to penalize more the examples incorrectly classified using both features. This process is iterated until a desired level of performance is reached. The algorithm has the flexibility to select class-specific features if it finds that the different object classes do not share any visual property.

2.1 Boosting for Binary Classification

Boosting [5,19,20] 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 $H(v) = \log P(c = 1|v)/P(c = 0|v)$ is the log-odds of being in class c. (Hence $P(c = 1|v) = \sigma(H(v))$, 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.

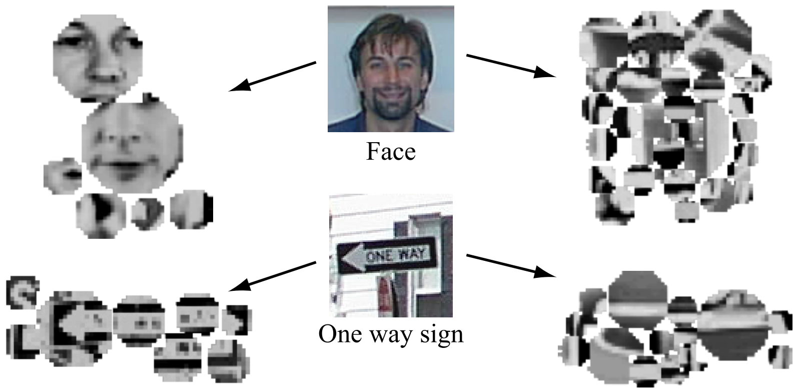


Fig. 1. Example of specific (left) and generic (right) features, and their class-conditional response distributions. Two possible representations of objects (e.g., face and one-way sign). The number of features used for each representation is selected so that both representations achieve the same detection performance (area under ROC is 0.95). The first representation (left) uses class-specific features (optimized for detecting each object class in cluttered scenes). Each feature is shown in object-centered coordinates. In contrast, the second representation is built upon the best features that can be shared across 29 object categories.

Boosting optimizes the following cost function one term of the additive model at a time:

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

where z is the class membership label (± 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 [20] or as an approximation to the likelihood of the training data under a logistic noise model [5].

There are many ways to optimize this function. We chose to base our algorithm on the version of boosting called "gentleboost" [5], because it is simple to implement, numerically robust, and has been shown experimentally [14] 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 $h_m(v) = a\delta(v^f > \theta) + b$, 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 we can replace $\sum_m b_m$ by a single global offset in the final strong classifier.) 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

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

$$a + b = \frac{\sum_i w_i z_i \delta(v_i^f > \theta)}{\sum_i w_i \delta(v_i^f > \theta)}, \tag{5}$$

We pick the f and θ, and corresponding a and b, with the lowest cost (using Eq. 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 Fig. 2.

2.2 Multiclass Boosting and Shared Stumps

In the multiclass case, we modify the cost function as in Adaboost.MH [19]:

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

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

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

```
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:
           $h_m(v_i) = a\delta(v_i^f > \theta) + b$
   (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. 2. 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.

where $H(v, c) = \log P(c = 1|v)/P(c = 0|v)$, so $P(c|v) = e^{H(v,c)}/\sum_{c'} e^{H(c',v)}$ (the softmax function).

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[1] for example i and for the classifier for class c. Here, we use the same procedure as in Adaboost.MH, but we change the structure of the multiclass weak classifiers. The key idea is that at each round m, the algorithm will choose a subset of classes $S(m)$ to be considered "positive"; examples from the remaining classes can be considered "negative" (i.e., part of the background) or ignored. This gives us a binary classification problem, which can be solved by fitting a binary decision stump as outlined above. (Some small modifications are required when we share classes, which are explained below.) The goal is to pick a subset and a weak learner that reduces the cost for all the classes. At the next round, a different subset of classes may be chosen. 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:

$$h_m(v, c) = \begin{cases} a\delta(v_i^f > \theta) + b & \text{if } c \in S(n) \\ k^c & \text{if } c \notin S(n) \end{cases} \tag{8}$$

[1] Note that each training example has C weights, 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.

The purpose of the class-specific constant k^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 does not contribute to the final strong classifier, but it changes the way features are shared, especially in the first iterations of boosting.

```
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:
          $$h_m^n(v_i, c) = \begin{cases} a\delta(v_i^f > \theta) + b & \text{if } c \in S(n) \\ k^c & \text{if } c \notin S(n) \end{cases}$$
       ii. Evaluate error
          $$J_{wse}(n) = \sum_{c=1}^{C} \sum_{i=1}^{N} w_i^c (z_i^c - h_m(v_i, c))^2$$
   (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. 3. 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.

Minimizing Eq. 7 gives

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

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

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

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. 3 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. 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 (see [7] for a review of feature selection techniques). We start by computing the best feature for each leaf (single class), and pick the class that maximally reduces the overall 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. Finally we select from all the sets we have examined the one 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. However, we have found empirically that the maximally greedy strategy (using $N_c = 1$) gives results which are as good as exhaustive search.

Fig. 4 compares two features, one optimized for one class only (faces) and another selected for optimal sharing.

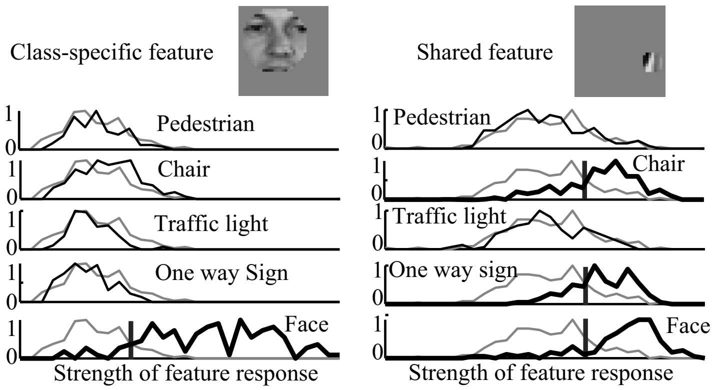


Fig. 4. These graphs compare the behavior of the two types of features for five object classes. Each graph shows the response distribution of a feature for non-objects (gray) and objects (black) of several classes. On the left side, the feature is class-specific and is optimized for detecting faces. The distributions show that the feature responds strongly when a face is present in its receptive field and weakly for non-objects or other object classes. When other object classes are present, the feature has no selectivity and is unable to discriminate between other object classes and non-objects, as expected. The plots on the right show the behavior of a feature selected for optimal sharing. This feature has elevated activation for a variety of objects (chairs, one-way signs, and faces).

3 Multiclass Object Detection

We are interested in the problem of object detection in cluttered scenes. In the rest of sections, we provide some experimental results and discuss some of the benefits of sharing features between a large number of object detectors.

3.1 LabelMe Database for Multiclass Object Detection

One important problem when developing algorithms for multiclass object detection is the lack of databases with labeled data. Most of existing databases for object recognition are inadequate for the task of learning to detect many object categories in cluttered real-world images. For this reason we have build a large database of hand-labeled images. Fig. 5 shows some examples of annotated images from the LabelMe database [18]. The LabelMe database and the online annotation tool for labeling new objects, can be found at:

http://www.csail.mit.edu/∼brussell/research/LabelMe/intro.html

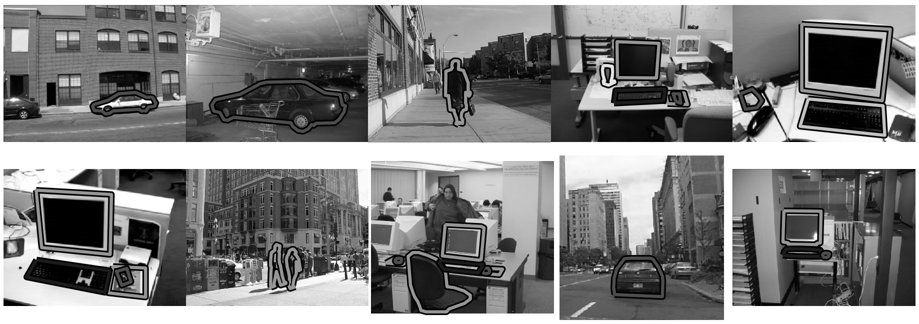


Fig. 5. Some examples of images from the LabelMe database

For the experiments 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, pedestrians); and heads (which can occur indoors and outdoors).

3.2 Local Features

For each 32x32 window in the image, we compute a feature vector. The features we use are inspired by the fragments proposed by [25]. Specifically, we extract a random set of 2000 patches or fragments from a subset of the 32x32 training images from all the classes (Fig. 6). The fragments have sizes ranging from 4x4 to 14x14 pixels. When we extract a fragment g_f, we also record the location from which it was taken (within the 32x32 window); this is represented by a binary spatial mask w_f. To compute the feature vector for a 32x32 window, we perform the following steps for each of the 2000 fragments f:

1. Apply normalized cross correlation between the window and the fragment to find where the fragment occurs;
2. 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.
3. Weight the response with the spatial mask (to test if the fragment occurs in the expected location).
4. Sum the result across all 32x32 pixels.
5. Perform element wise exponentiation using exponent $1/p$.

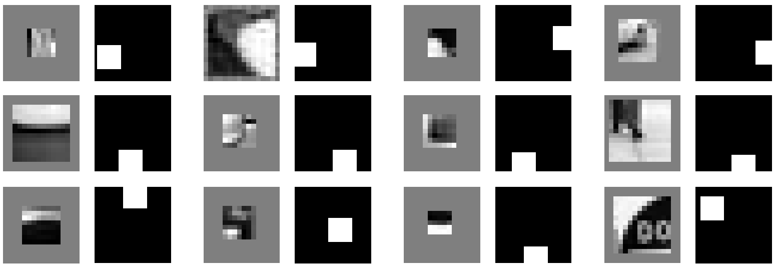


Fig. 6. Each feature is composed of a template (image patch on the left) and a binary spatial mask (on the right) indicating the region in which the response will be averaged. The patches vary in size from 4x4 pixels to 14x14.

This procedure converts each 32x32 window into a single positive scalar for each fragment f. This operation, for all image locations and scales, can be summarized as:

$$v^f(x, y, \sigma) = (w_f * |I_\sigma \otimes g_f|^p)^{1/p} \tag{12}$$

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.

In this chapter, we use $p = 10$; this is good for template matching as it approximates a local maximum operator (although we feel that other values of p will be useful for objects defined as textures like buildings, grass, etc.). Using 2000 fragments give us a 2000 dimensional feature vector for each window. 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.

3.3 Dictionary of Visual Shared Features

One important 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.

Fig. 7 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 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).

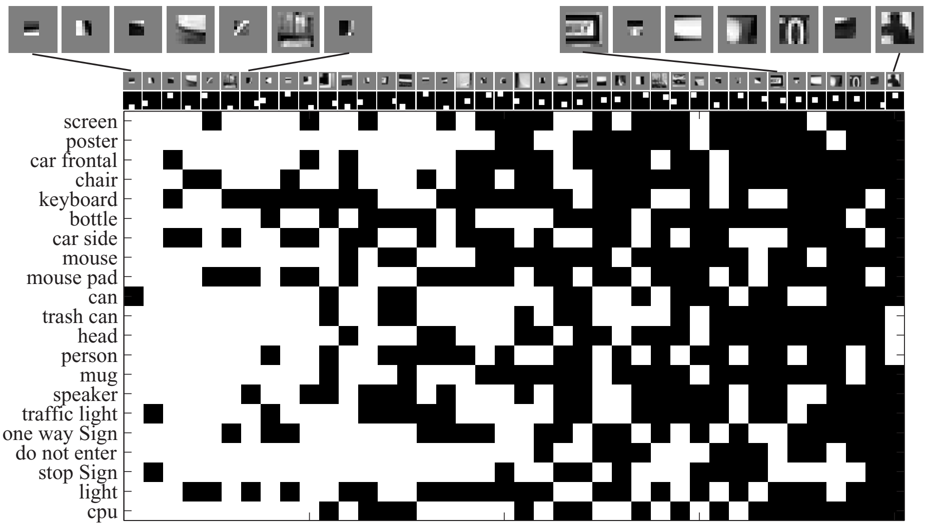


Fig. 7. Matrix that relates features to classifiers, which shows which features are shared among the different object classes. The features are sorted from left to right from more generic (shared across many objects) to more specific. Each feature is defined by one filter, one spatial mask and the parameters of the regression stump (not shown). These features were chosen from a pool of 2000 features in the first 40 rounds of boosting.

Fig. 1 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 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 like a template matching detector. 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 are more generic (configuration of edges) which can be reused by other objects.

Fig. 8 shows some typical results for the detection of office objects. Note that not all the objects achieve the same performance after training. The figure shows some results for the detection of computer monitors, keyboards and mouse pads. The three classifiers have been trained jointly with 18 other objects.

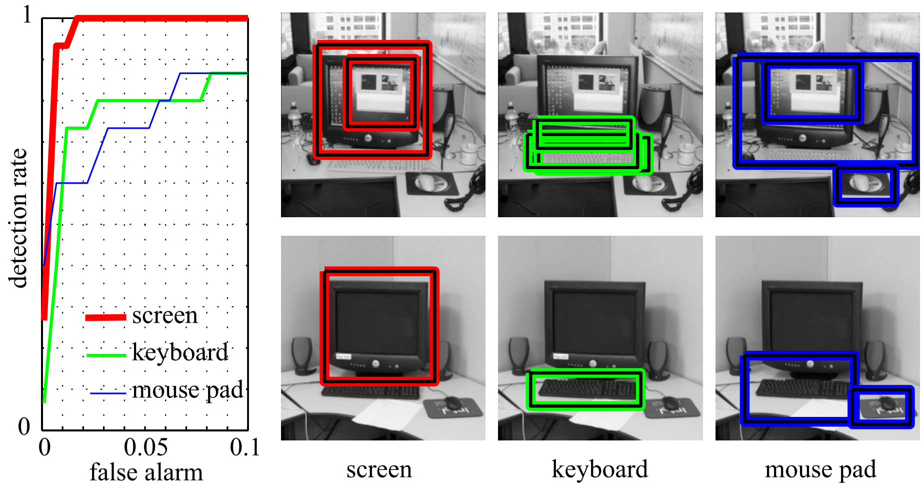


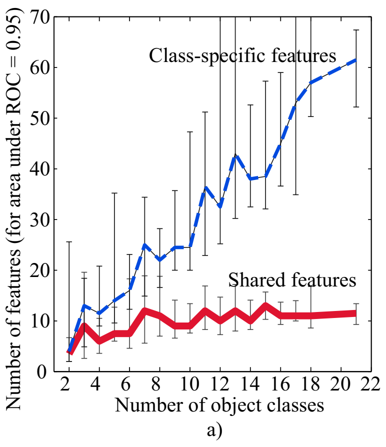
Fig. 8. ROC for detection of screens, keyboards and mouse pads when trained jointly with other 18 objects. On the right, we show some typical results of the detector output on images with size 256x256 pixels. The multiclass classifier, trained using boosting, uses 500 features (stumps) shared across 21 object classes.

3.4 Generalization and Effective Training Set

When building a vision system able to detect thousands of objects, using a set of independent classifiers will require a large amount of computations that will grow linearly with respect to the number of object classes. Most of those computations are likely to be redundant.

One important consequence of feature sharing is that the number of features needed grows sub-linearly with respect to the number of classes. Fig. 9.a shows the number of features necessary to obtain a fixed performance as a function of the number of object classes to be detected. When using C independent classifiers, the complexity grows linearly as expected. However, when sharing features among classifiers, the complexity grows sublinearly. (A similar result has been reported by Krempp, et. al ([8]) 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.

Another important consequence of joint training is that the amount of training data required is reduced. If different classes share common features, the learning of such features should benefit from the multiple classes reducing the amount of data required for each class. In the case where we are training C object class



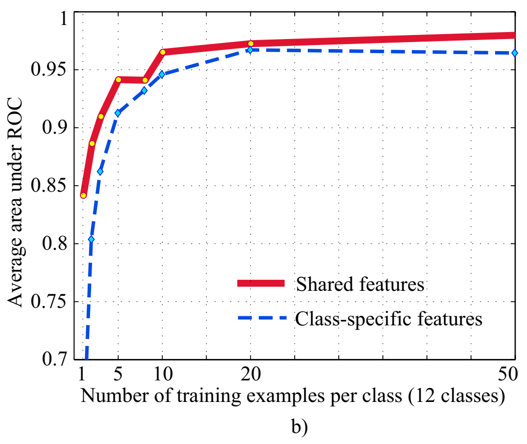


Fig. 9. Efficiency and generalization improve when objects are trained jointly allowing them to share features. a) Number of features needed in order to reach a fix level of performance (area under the ROC equal to 0.95). The results are averaged across 20 training sets. The error bars show the variability between the different runs (80% interval). b) Detection performance as a function of number of training examples per class when training 12 detectors of different object categories.

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.

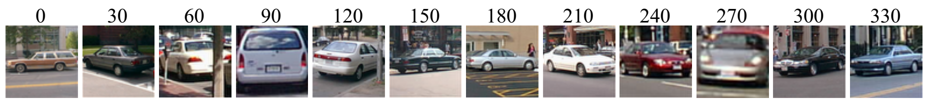
Fig. 9.b shows the detection performance as a function of number of training examples per class when training 12 detectors of different object categories (we used 600 features in the dictionary, and 1000 negative examples). Sharing features improves the generalization when few training samples are available, especially when the classes have many features in common. 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. From Fig. 9.b, we get that $N^e \approx 2.1N$ (i.e., we need to double the size of the training set to get the same performance out of class-specific features).

3.5 Multiview Object Detection

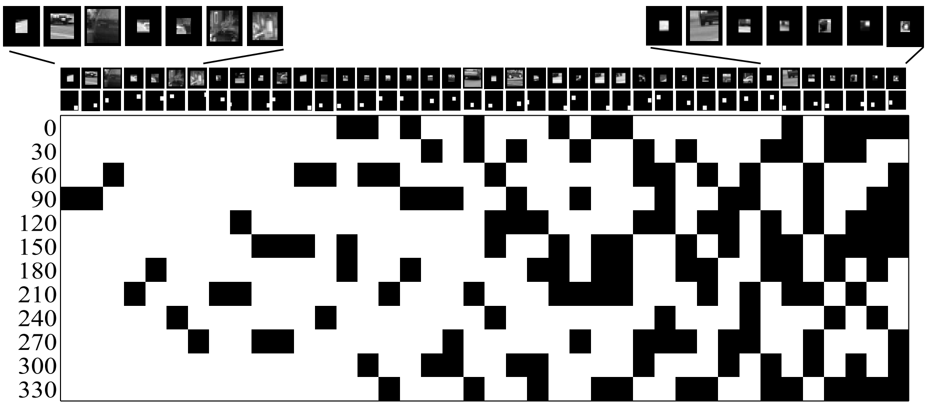
In the case of multiple views, some objects have poses that look very similar. For instance, in the case of a car, both frontal and back views have many common features, and both detectors should share a lot of computations. However, in the case of a computer monitor, the front and back views are very different, and we will not be able to share many features. Our algorithm will share features as much as possible, but only if it does not hurt performance.

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). By sharing features we could have a set of features shared across all views, not very specific and trying to solve the view invariant detection problem, and then a set of features with less sharing and more specific to few views of the object. Our goal is to implement an object detector that works for many views of the object and that can provide an estimation of the pose of the object.

Fig. 10 shows a dictionary of features (here localized image patches) build, using multiclass Boosting, for the task of multiview car detection. Here we trained 12 detectors each one tuned to one orientation.



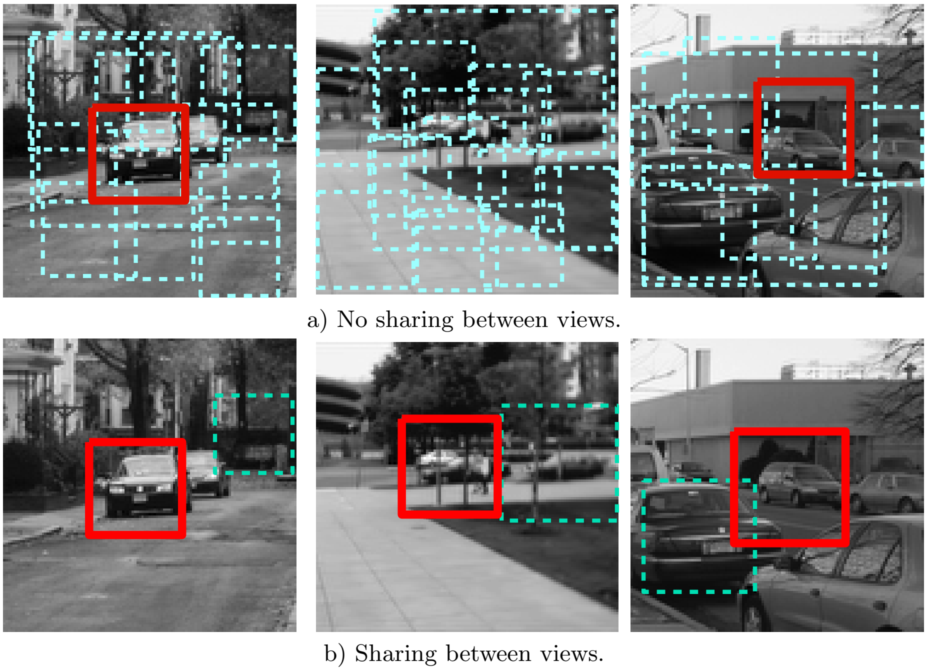
a)



b)

Fig. 10. Matrix that relates features to classifiers, which shows which features are shared among the different car views (orientation is discretized in 12 groups). The features are sorted from left to right from more generic (shared across many objects) to more specific.

Fig. 11 shows the results of multiview car detectors and compares the classifiers obtained with specific and shared features. 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, whereas the joint classifier results in about 1 false alarm per image (averages obtained on 200 images not used for training). Test images were


a) No sharing between views.

b) Sharing between views.

Fig. 11. View invariant car detection (dashed boxes are false alarms, and solid boxes are correct detections). a) No feature sharing, b) feature sharing. The joint training provides more robust classifiers with the same complexity.

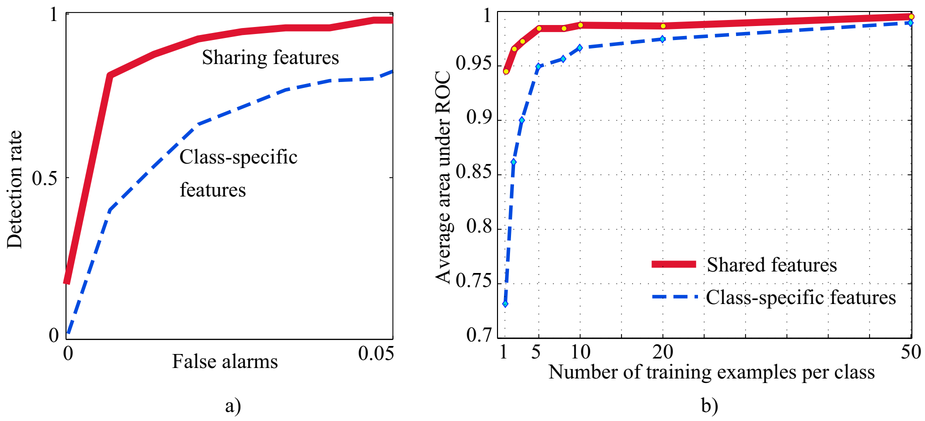


Fig. 12. a) ROC for view invariant car detection. b) Detection performance as a function of number of training examples per class (each class correspond to one out of the 12 car orientations) when using view-specific and shared features. Detection performance is measured as the average area under the ROC for all the classes.

128x128 pixels, which produced more than 17000 patches to be classified. The detector is trained on square regions of size 24x24 pixels. Fig. 12 summarizes the result showing the ROC for detectors using specific and shared features.

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. In order to be comparable with the results of fig. 9.b, we used 600 features in the dictionary (created from patches extracted from cars), and 1000 negative examples. Specifically, we find that in the multiclass case (fig. 9.b), 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. 12, we obtain that the equivalent training set size is $N^e \approx 4.8N$ (i.e., joint training effectively increases the training set for every class by almost a factor of 5).

4 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 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.

Acknowledgments

This work was sponsored in part by the Nippon Telegraph and Telephone Corporation as part of the NTT/MIT Collaboration Agreement, by DARPA contract DABT63-99-1-0012. We acknowledge support from NSF contract IIS-0413232.

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. 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.
3. R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.
4. 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.
5. J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of statistics*, 28(2):337–374, 2000.

6. 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.
7. R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 1.
8. 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.
9. S. Lazebnik, C. Schmid, and J. Ponce. Affine-invariant local descriptors and neighborhood statistics for texture recognition. In *Intl. Conf. on Computer Vision*, 2003.
10. 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.
11. 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.
12. 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.
13. K. Levi, M. Fink, and Y. Weiss. Learning from a small number of training examples by exploiting object categories. In *Workshop of Learning in Computer Vision*, 2004.
14. 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.
15. 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.
16. H. Murase and S. Nayar. Visual learning and recognition of 3-d objects from appearance. *Intl. J. Computer Vision*, 14:5–24, 1995.
17. C. Papageorgiou and T. Poggio. A trainable system for object detection. *Intl. J. Computer Vision*, 38(1):15–33, 2000.
18. B. C. Russell, A. Torralba, K. P. Murphy, W. T. Freeman. LabelMe: a database and web-based tool for image annotation. MIT AI Lab Memo AIM-2005-025, September, 2005.
19. R. Schapire and Y. Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.
20. R. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2001.
21. 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.
22. E. Sudderth, A. Torralba, W.T. Freeman, and A. Willsky. Learning hierarchical models of scenes, objects, and parts. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2005.
23. S. Thrun and L. Pratt, editors. *Machine Learning. Special issue on Inductive Transfer*. 1997.
24. 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*, 2004.
25. M. Vidal-Naquet and S. Ullman. Object recognition with informative features and linear classification. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2003.
26. P. Viola and M. Jones. Robust real-time object detection. *Intl. J. Computer Vision*, 57(2):137–154, 2004.