Sharing features: efficient boosting procedures for multiclass object detection

Antonio Torralba Kevin P. Murphy William T. Freeman
Computer Science and Artificial Intelligence Lab., MIT
Cambridge, MA 02139

Abstract

We consider the problem of detecting a large number of different object classes in cluttered scenes. Traditional approaches require applying a battery of different classifiers to the image, which can be slow and require much training data. We present a multi-class boosting procedure (joint boosting) that reduces both the computational and sample complexity, by finding common features that can be shared across the classes. The detectors for each class are trained jointly, rather than independently. For a given performance level, the total number of features required is observed to scale approximately logarithmically with the number of classes. In addition, we find that the features selected by independently trained classifiers are often specific to the class, whereas the features selected by the jointly trained classifiers are more generic features, such as lines and edges.

1. Introduction

A long-standing goal of machine vision has been to build a system that is able to recognize many different kinds of objects in cluttered scenes. Progress has been made on restricted versions of this goal. In particular, it is now possible to recognize *instances* of highly textured objects, such as magazine covers or toys, despite clutter, occlusion and affine transformations, by using object-specific features [12, 19]. In addition, it is possible to recognize many *classes* of objects, generalizing over intra-class variation, but only when the objects are presented against simple backgrounds [14, 15, 10, 4, 13].

The problem of detecting classes of objects in cluttered images has proved more challenging. Most current approaches slide a window across the image, and apply a binary classifier to each such window; the classifier, which discriminates between the class or the background, is trained using standard machine learning techniques, such as boosting [22] or support vector machines [16]). However, such approaches seem unlikely to scale up to the detection of hundreds or thousands of different object classes because each classifier is trained and run independently.

In this paper, we develop a new object classification architecture that explicitly learns to share features across multiple object classes (classifiers). The basic idea is an extension of the boosting algorithm [17, 6], which has been shown to be useful for detecting individual object classes in cluttered scenes [22]. Rather than training C binary classifiers independently, we train them jointly. The result is that many fewer features are needed to achieve a desired level of performance than if we were to train the classifiers independently. This results in a faster classifier (since there are fewer features to compute) and one which works better (since the features are fit to larger, shared data sets).

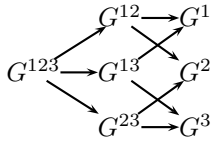


Figure 1: All possible ways to share features amongst 3 classifiers. Each classifier $H(v, c)$ is constructed by adding, only once, all the nodes that connect to each of the leaves. The leaves correspond to single classes.

2. Sharing features

Boosting [17, 6] provides a simple way to sequentially fit additive models of the form

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

where c is the class label, v is the input feature vector, and M is the number of rounds. In the boosting literature, the h_m are often called weak learners. It is common to define these to be simple decision or regression stumps of the form $h_m(v) = a\delta(v^f > \theta) + b$, where v^f denotes the f'th component (dimension) of the feature vector v, θ is a threshold, δ is the indicator function, and a and b are regression parameters (note that b does not contribute to the final classification).

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

ing classifiers:

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

where each $G^{S(n)}(v)$ is itself an additive model of the form $G^{S(n)}(v) = \sum_{m=1}^{M_n} h_m^n(v)$. The n refers to a node in the "sharing graph", which specifies which functions can be shared between classifiers (Fig.1). $S(n)$ is the subset of classes that share the node n.

The decomposition is not unique (different choices of functions $G^{S(n)}(v)$ give the same functions $H(v,c)$). But we are interested in the choices of $G^{S(n)}(v)$ that minimize the computational cost. We impose the constraint that $\sum_n M_n = M$, where M is the total number of functions that have to be learned. If the classifiers are trained independently, we find we need $O(C)$ functions, whereas if the classifiers are trained jointly, the required number of features grows sub-linearly with the number of classes.

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

2.1. The joint boosting algorithm

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

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

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

We chose to base our algorithm on the version of boosting called "gentleboost" [7], because it is simple to implement, numerically robust, and has been shown experimentally [11] to outperform other boosting variants for the face detection task. The optimization of J is done using adaptive Newton steps [7] which corresponds to minimizing a weighted squared error at each step. Specifically, at step m, the function H is updated as: $H(v,c) := H(v,c) + h_m(v,c)$, where h_m is chosen so as to minimize a second order Taylor approximation of the cost function. Replacing the expectation in Equation 1 with an empirical expectation over the training data, and defining weights $w_i^c = e^{-z_i^c H(v_i,c)}$ for example i and class c, results in minimizing the weighted squared error:

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

The resulting optimal function at round m is given by

$$h_m(v,c) = \begin{cases} a\delta(v_i^f > \theta) + b & \text{if } c \in S(n) \\ k^c & \text{if } c \notin S(n) \end{cases} \tag{3}$$

with parameters $(a, b, f, \theta, n, k^c)$, for each $c \notin S(n)$, i.e., a total of $5 + C - |S(n)|$ parameters. The minimization of (2) gives the parameters:

$$b = \frac{\sum_{c \in S(n)} \sum_i w_i^c z_i^c \delta(v_i^f \le \theta)}{\sum_{c \in S(n)} \sum_i w_i^c \delta(v_i^f \le \theta)}, \tag{4}$$

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

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

Figure 2: Joint boosting with regression stumps. v_i^f is the f'th feature of the i'th training example, $z_i^c \in \{-1, +1\}$ are the labels for class c, and w_i^c are the *unnormalized* example weights. N is the number of training examples, and M is the number of rounds of boosting.

$$k^c = \frac{\sum_i w_i^c z_i^c}{\sum_i w_i^c} \quad c \notin S(n) \tag{6}$$

By adding a shared stump, the complexity of the multiclass classifier increases with constant rate, independent of the number of classes sharing the stump. Only the classes that share the feature at round m will have a reduction of their classification error.

For all the classes c in the set $S(n)$, the function $h_m(v, c)$ is a shared regression stump. For the classes that do not share this feature ($c \notin S(n)$), the function $h(v, c)$ is a constant k^c different for each class. This constant prevents sharing features due to the asymmetry between the number of positive and negative samples for each class. However, these constants do not contribute to the final classification. Fig. 2 summarizes the algorithm.

As we do not know which is the best sharing $S(n)$, in principle we need to search over all $2^C - 1$ possible sharing patterns at each iteration, to find the one that minimizes Eq. (2). Obviously this would be very slow. Below, we discuss a greedy search heuristic that has complexity $O(C^2)$ instead of $O(2^C)$.

2.2. Efficient computation of shared stumps

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 [8] 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 empiricially that the maximally greedy strategy (using $N_c = 1$) gives results which are as good as exhaustive search.

To evaluate the quality of a node in the sharing graph, we must find the optimal regression stump (using Equations 4, 5 and 6), which is slow. However, it turns out that we can propagate most of the computations from the leaves to higher nodes, as we now discuss. We start by computing the parameters a and b for a set of predefined thresholds and for all features, so as to minimize the weighted square error. Then, the parameters a and b for each threshold and feature at any other internal node can be computed simply as a weighted combination of the parameters at the leaves that are connected with the node. The best regression parameters for a subset of classes S is:

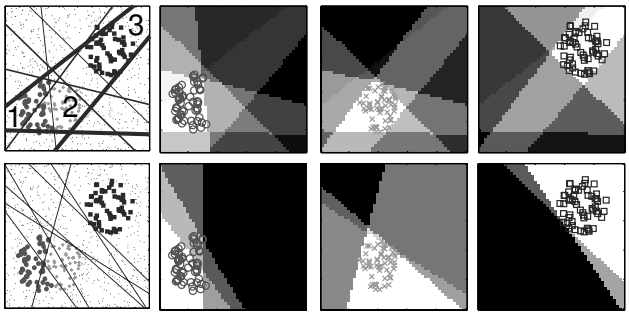


Figure 3: Illustration of joint boosting (top row) and independent boosting (bottom row) on a toy problem in which there are three object classes and one background class. 50 samples from each class are used for training, and we use 8 rounds of boosting. Left: The thickness of the lines indicates the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

$$b_S(f, \theta) = \frac{\sum_{c \in S} b_c(f, \theta) w_c^+(f, \theta)}{\sum_{c \in S} w_c^+(f, \theta)} \tag{7}$$

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

$$\begin{aligned} J_{wse}(n) \quad = \quad & (1 - \hat{a}_s^2) \sum_{c \in S(n)} w_c^+ + (1 - b_s^2) \sum_{c \in S(n)} w_c^- + \\ & + \sum_{c \notin S(n)} \sum_{i=1}^N w_i^c (z_i^c - k^c)^2 \end{aligned} \tag{8}$$

with $\hat{a}_s = a_s + b_s$. The first two terms correspond to the weighted error in the classes sharing a feature. The third term is the error for the classes that do not share a feature at this round. This can be used instead of Eq. 2, for speed.

2.3. Example of sharing on a toy problem

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

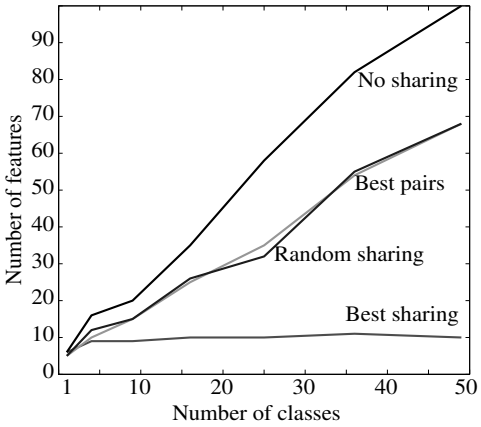


Figure 4: Complexity of the multiclass classifier as a function of the number of classes. The complexity of a classifier is evaluated here as the number of stumps needed for achieving a predefined level of performance (area under the ROC of 0.95).

Fig. 4 illustrates the dependency of the complexity of the classifier as a function of the number of classes when using different sharing patterns when the classifiers are required to achieve the same level of performance. For this experiments we use 2 dimensions, 25 training samples per class, and 40,000 samples for the background. As expected, when no sharing is used, the complexity grows linearly with the number of classes. When sharing is only allowed between pairs of classes, the complexity still grows linearly with the number of classes. The same thing occurs with random sharing, since, in 2D, random sharing will be good for at least two classes at each round (and for D classes in D dimensions). However, when using the best sharing at each round, the complexity drops dramatically, and the dependency between complexity and number of classes follows a logarithmic curve.

3. Multiclass object detection

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

3.1. Dictionary of features

In the study presented here we used 21 object categories[1] (13 indoor objects: screen, keyboard, mouse, mouse pad, speaker, computer, trash cans, poster, bottle, chair, can, mug, light; 7 outdoor objects: frontal view car, side view car, traffic light, stop sign, one way sign, do not enter sign; and 2 objects that can occur indoors or outdoors: heads and pedestrians).

[1] A labeled database of indoor and outdoor scenes, and a extended version of this paper [20], are available at http://web.mit.edu/torralba/www/multiclass.html

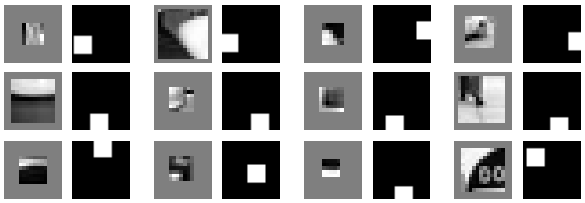


Figure 5: Each feature is composed of a template (image patch on the left) and a binary spatial mask (on the right) indicating the region in which the response will be averaged. The patches vary in size from 4x4 pixels to 14x14.

Figure 6: Examples of correct detections of classifiers trained jointly (screen, poster, cpu, car side, stop sign, mug).

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

$$v^f(x,\sigma) = (w_f * |I_\sigma \otimes g_f|^p)^{1/p} \quad (9)$$

where $\otimes$ represents the normalized correlation between the image I_σ at scale σ and the filter g_f, and $*$ represents the convolution operator with spatial mask (window) $w_f(x)$. The filters g_f are generated by randomly extracting patches from images of the 21 objects, after they where resized to 32x32 pixels. We generated a total of 2000 patches from each class (see Fig. 5).

The exponent p allows us to generate different types of features. For example, by setting $p = 1$, the feature vector encodes the average of the filter responses, which are good for describing textures. By setting $p > 10$, the feature vector becomes $v^f \simeq \max_{x \in S_w}\{|I_\sigma \otimes g_f|\}$, where $S_w(x)$ is the support of the window for a feature at the location x. This is good for template matching [21]. By changing the spatial mask, we can change the size and location of the region in which the feature is evaluated. This provides a way of generating features that are well localized (good for part-based encoding and template matching) and features that provide a global description of the patch (good for texture-like objects).

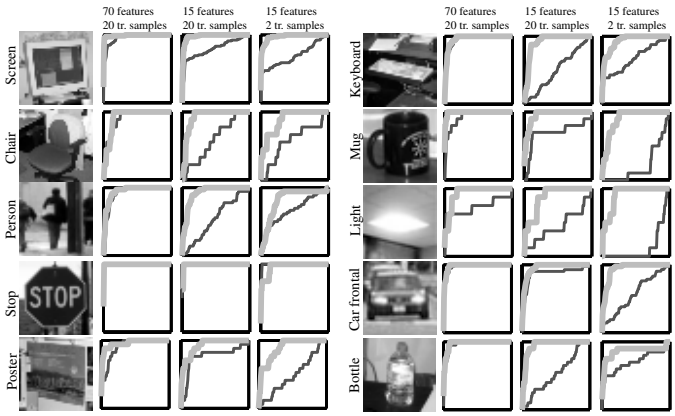


Figure 7: ROC curves for some of the 21 objects used (thin (lower curve) = isolated detectors, thick (bottom curve) = joint detectors). From left to right: i) 70 features in total (on average $70/21 \simeq 3.3$ features per object) and 20 training samples per object, ii) 15 features and 20 training samples, and iii) 15 features and 2 training samples.

For the experiment presented in this section we set $p = 10$, and we took the window w_n to be localized within the 32x32 region near where the patch was extracted in the original image, c.f., [21].

3.2. Results

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

Fig. 6 shows some sample detection results when running the detectors on whole images by scanning each location and scale. Figure 7 summarizes the performances of the detectors for each class. For the test set, we use an independent set of images (images from the web, and taken with a digital camera). All the detectors have better performance when trained jointly, sometimes dramatically so.

By training the objects using joint boosting, at each round we find which feature best reduces the total multiclass classification error. Fig. 8 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. As this feature reduces the error in two classes at once, it has been chosen over other more specific features that might have been performed better on a single class, but

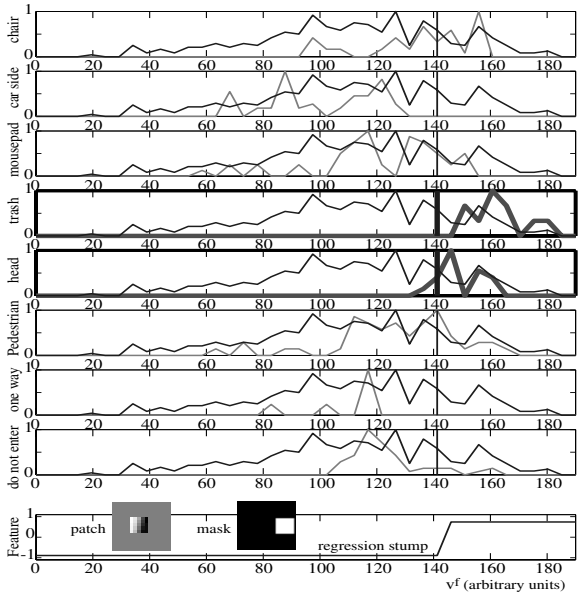


Figure 8: Example of a shared feature between two objects (heads and trash-cans) when training 8 objects jointly. The shared feature is shown at the bottom of the figure. For each object, the thin graph shows an empirical approximation to $p(v^f|z^c = 0)$, and the thick graph shows $p(v^f|z^c = 1)$.

which would have resulted in worst performance when considering the multiclass loss function.

Fig. 9 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 objects. 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. From left to right the features are sorted from generic features (shared across many classes) to class-specific features (shared among very few objects).

When training detectors jointly, the system will look for features that generalize across multiple classes, such as edges and other generic features (Fig. 9). Conversely, when we train the detectors independently, the system learns class-specific features, that look more like part-templates. The disadvantage of class-specific features is that we cannot afford to compute enough of them if we have a large number of classes, since the features of each class have to relearn object extraction functions common across classes.

3.3. Computational and sample complexity

One important consequence of feature sharing is that the number of features needed grows sub-linearly with respect to the number of classes. Fig. 10 shows the number of features necessary to obtain a fixed performance as a function

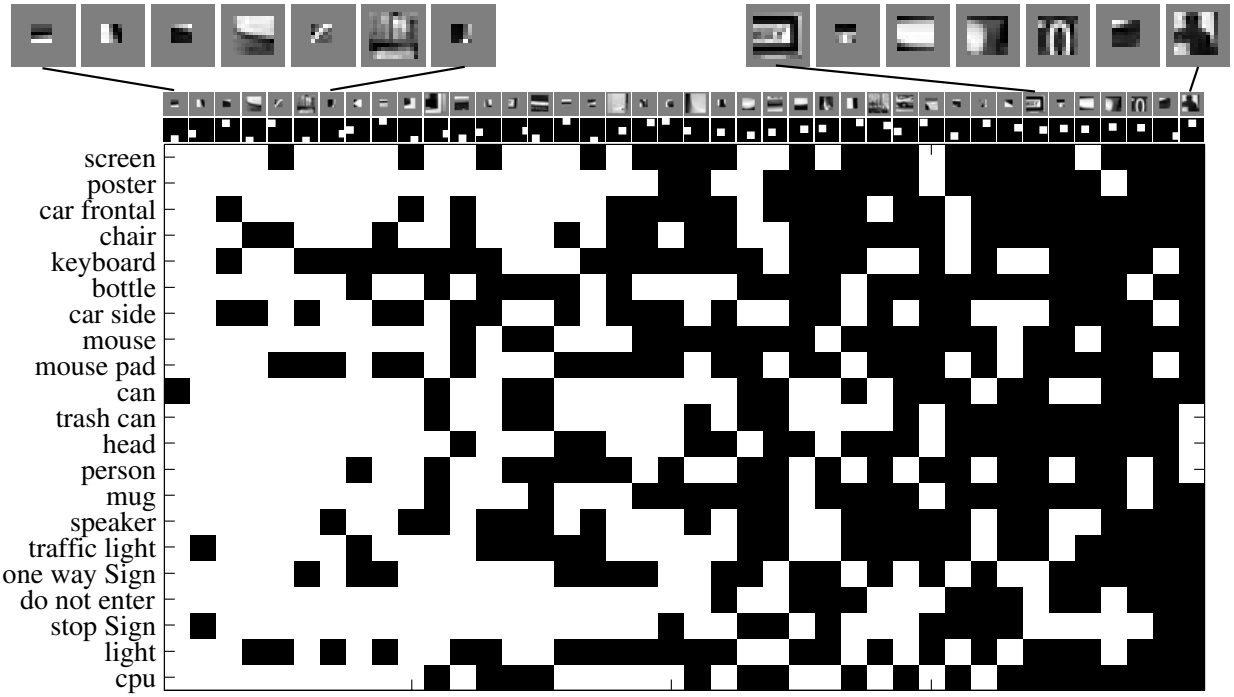


Figure 9: Matrix that relates features to classifiers, which shows which features are shared among the different object classes. The features are sorted from left to right from more generic (shared across many objects) to more specific. Each feature is defined by one filter, one spatial mask and the parameters of the regression stump (not shown). These features were chosen from a pool of 2000 features in the first 40 rounds of boosting.

of the number of object classes to be detected. When using C independent classifiers, the complexity grows linearly as expected. However, when joint boosting is used, the complexity is compatible with $log(C)$. (A similar result has been reported by Krempp, et. al ([9]) 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. Fig. 7 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.

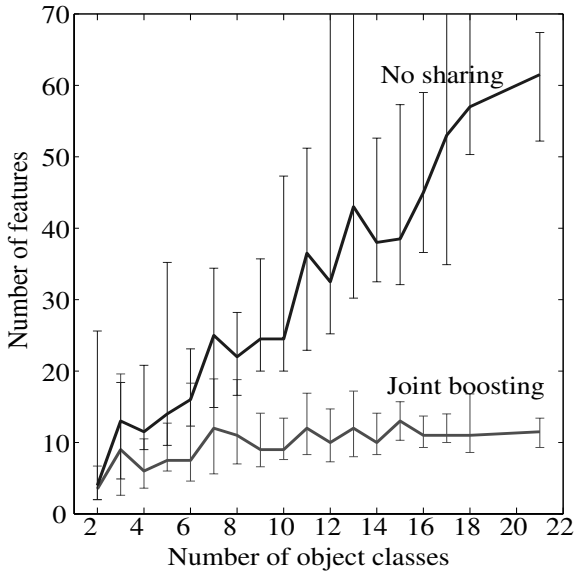


Figure 10: Number of features needed in order to reach a fix level of performance (area under the ROC equal to 0.9). The results are averaged across 8 training sets. The error bars show the variability between the different runs.

4. Multiview object detection

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

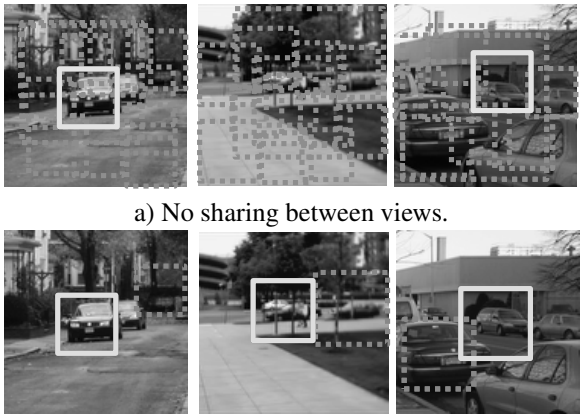
Figure 11: Examples of pose variations for cars and screens.

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., [18]).

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

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

Fig. 12 shows the results of multiview car detectors and compares the classifiers obtained using independent boosting for each view and joint boosting. In both cases, we limit the number of stumps to 70 and training is performed with 20 samples per view (12 views). Both classifiers have the same computational cost. The top row shows typical detection results obtained by combining 12 independent binary classifiers, each one trained to detect one specific view. When the detection threshold is set to get 80% detection rate, independent classifiers produce over 8 false alarms per image on average, whereas the joint classifier results in about 1 false alarm per image (averages obtained on 200 images not used for training). Test images were 128x128 pixels, which produced more than 17000 patches to be classified. The detector is trained on square regions of size 24x24 pixels. Fig. 13 summarizes the result showing the ROC for both detectors.



a) No sharing between views.

b) Sharing between views.

Figure 12: 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.

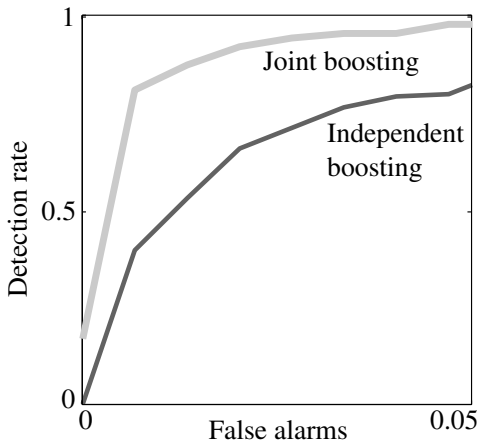


Figure 13: ROC for view invariant car detection.

5. Previous work

This paper builds on previous work in two separate communities, namely computer vision and machine learning. We will discuss relevant papers from each in turn.

Fei-Fei, Fergus and Perona [5] propose a parts-based object representation and impose a prior on the model parameters for each class, which encourages classes to be similar. However, the parts themselves are not shared across classes. Krempp, Geman and Amit [9] present a system that learns to reuse parts for detecting several object categories. The system is trained incrementally. They apply their system to detecting overlapping characters. They show that the number of parts grows logarithmically with respect to the number of classes, which we also find. However, they do not jointly optimize the shared features, and they have not applied their technique to real-world images.

In the machine learning community, Caruana [2] has studied the problem of "multiple task learning", but as far as we know, this has never been applied to the object detection task. A closer connection is to the general framework developed by Dietterich and Bakiri [3] for converting binary classifiers into multiple-class classifiers using error-correcting output codes (ECOC). A difference between our approach

and the ECOC framework is how we use the subsets of classifiers. In ECOC, they classify an example by running each set of classifiers, and look for the closest matching row in the code matrix. As we saw in Section 2, in our algorithm, we add the output of the individual subset classifiers together, as in a standard additive model. Allwein et. al. [1] show that the popular one-against-all approach is often suboptimal, but that the best code matrix to use is problem dependent. Although our algorithm starts with a complete code matrix, it learns which subsets are actually worth using.

6. Conclusion

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

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

Acknowledgments

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

References

[1] 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.

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

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

[4] S. Edelman and S. Duvdevani-Bar. A model of visual recognition and categorization. *Phil. Trans. Royal Soc. B*, 352:1191–1202, 1997.

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

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

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

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

[9] S. Krempp, D. Geman, and Y. Amit. Sequential learning of reusable parts for object detection. Technical report, CS Johns Hopkins, 2002.

[10] 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.

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

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

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

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

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

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

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

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

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

[20] A. Torralba, K. Murphy, and W. Freeman. Sharing visual features for multiclass and multiview object detection. Technical report, CSAIL Technical report, MIT, 2004.

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

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