Object Detection with Discriminatively Trained Part Based Models

Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester and Deva Ramanan

Abstract—We describe an object detection system based on mixtures of multiscale deformable part models. Our system is able to represent highly variable object classes and achieves state-of-the-art results in the PASCAL object detection challenges. While deformable part models have become quite popular, their value had not been demonstrated on difficult benchmarks such as the PASCAL datasets. Our system relies on new methods for discriminative training with partially labeled data. We combine a margin-sensitive approach for data-mining hard negative examples with a formalism we call *latent SVM*. A latent SVM is a reformulation of MI-SVM in terms of latent variables. A latent SVM is semi-convex and the training problem becomes convex once latent information is specified for the positive examples. This leads to an iterative training algorithm that alternates between fixing latent values for positive examples and optimizing the latent SVM objective function.

Index Terms—Object Recognition, Deformable Models, Pictorial Structures, Discriminative Training, Latent SVM

1 Introduction

Object recognition is one of the fundamental challenges in computer vision. In this paper we consider the problem of detecting and localizing generic objects from categories such as people or cars in static images. This is a difficult problem because objects in such categories can vary greatly in appearance. Variations arise not only from changes in illumination and viewpoint, but also due to non-rigid deformations, and intraclass variability in shape and other visual properties. For example, people wear different clothes and take a variety of poses while cars come in a various shapes and colors.

We describe an object detection system that represents highly variable objects using mixtures of multiscale deformable part models. These models are trained using a discriminative procedure that only requires bounding boxes for the objects in a set of images. The resulting system is both efficient and accurate, achieving state-of-the-art results on the PASCAL VOC benchmarks [11]–[13] and the INRIA Person dataset [10].

Our approach builds on the pictorial structures framework [15], [20]. Pictorial structures represent objects by a collection of parts arranged in a deformable configuration. Each part captures local appearance properties of an object while the deformable configuration is characterized by spring-like connections between certain pairs of parts.

Deformable part models such as pictorial structures provide an elegant framework for object detection. Yet it has been difficult to establish their value in practice. On difficult datasets deformable part models are often outperformed by simpler models such as rigid templates [10] or bag-of-features [44]. One of the goals of our work is to address this performance gap.

While deformable models can capture significant variations in appearance, a single deformable model is often not expressive enough to represent a rich object category. Consider the problem of modeling the appearance of bicycles in photographs. People build bicycles of different types (e.g., mountain bikes, tandems, and 19th-century cycles with one big wheel and a small one) and view them in various poses (e.g., frontal versus side views). The system described here uses mixture models to deal with these more significant variations.

We are ultimately interested in modeling objects using "visual grammars". Grammar based models (e.g. [16], [24], [45]) generalize deformable part models by representing objects using variable hierarchical structures. Each part in a grammar based model can be defined directly or in terms of other parts. Moreover, grammar based models allow for, and explicitly model, structural variations. These models also provide a natural framework for sharing information and computation between different object classes. For example, different models might share reusable parts.

Although grammar based models are our ultimate goal, we have adopted a research methodology under which we gradually move toward richer models while maintaining a high level of performance. Improving performance by enriched models is surprisingly difficult. Simple models have historically outperformed sophisticated models in computer vision, speech recognition, machine translation and information retrieval. For example, until recently speech recognition and machine translation systems based on n-gram language models outperformed systems based on grammars and phrase

- *P.F. Felzenszwalb is with the Department of Computer Science, University of Chicago. E-mail:* pff@cs.uchicago.edu
- *R.B. Girshick is with the Department of Computer Science, University of Chicago. E-mail:* rbg@cs.uchicago.edu
- *D. McAllester is with the Toyota Technological Institute at Chicago. E-mail:* mcallester@tti-c.org
- *D. Ramanan is with the Department of Computer Science, UC Irvine. E-mail:* dramanan@ics.uci.edu

structure. In our experience maintaining performance seems to require gradual enrichment of the model.

One reason why simple models can perform better in practice is that rich models often suffer from difficulties in training. For object detection, rigid templates and bag-of-features models can be easily trained using discriminative methods such as support vector machines (SVM). Richer models are more difficult to train, in particular because they often make use of latent information.

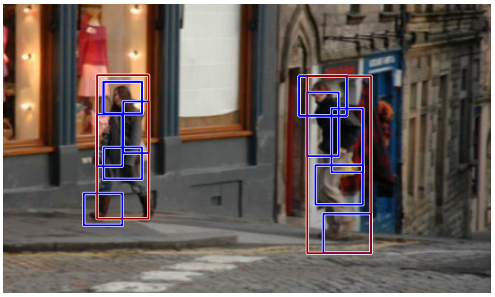
Consider the problem of training a part-based model from images labeled only with bounding boxes around the objects of interest. Since the part locations are not labeled, they must be treated as latent (hidden) variables during training. More complete labeling might support better training, but it can also result in inferior training if the labeling used suboptimal parts. Automatic part labeling has the potential to achieve better performance by automatically finding effective parts. More elaborate labeling is also time consuming and expensive.

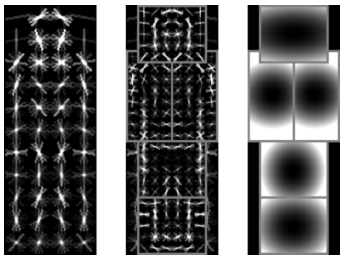
The Dalal-Triggs detector [10], which won the 2006 PASCAL object detection challenge, used a single filter on histogram of oriented gradients (HOG) features to represent an object category. This detector uses a sliding window approach, where a filter is applied at all positions and scales of an image. We can think of the detector as a classifier which takes as input an image, a position within that image, and a scale. The classifier determines whether or not there is an instance of the target category at the given position and scale. Since the model is a simple filter we can compute a score as $\beta \cdot \Phi(x)$ where β is the filter, x is an image with a specified position and scale, and $\Phi(x)$ is a feature vector. A major innovation of the Dalal-Triggs detector was the construction of particularly effective features.

Our first innovation involves enriching the Dalal-Triggs model using a star-structured part-based model defined by a "root" filter (analogous to the Dalal-Triggs filter) plus a set of parts filters and associated deformation models. The score of one of our star models at a particular position and scale within an image is the score of the root filter at the given location plus the sum over parts of the maximum, over placements of that part, of the part filter score on its location minus a deformation cost measuring the deviation of the part from its ideal location relative to the root. Both root and part filter scores are defined by the dot product between a filter (a set of weights) and a subwindow of a feature pyramid computed from the input image. Figure 1 shows a star model for the person category.

In our models the part filters capture features at twice the spatial resolution relative to the features captured by the root filter. In this way we model visual appearance at multiple scales.

To train models using partially labeled data we use a latent variable formulation of MI-SVM [3] that we call *latent SVM* (LSVM). In a latent SVM each example x is





(a) (b) (c)

Fig. 1. Detections obtained with a single component person model. The model is defined by a coarse root filter (a), several higher resolution part filters (b) and a spatial model for the location of each part relative to the root (c). The filters specify weights for histogram of oriented gradients features. Their visualization show the positive weights at different orientations. The visualization of the spatial models reflects the "cost" of placing the center of a part at different locations relative to the root.

scored by a function of the following form,

$$f_\beta(x) = \max_{z \in Z(x)} \beta \cdot \Phi(x, z). \quad (1)$$

Here β is a vector of model parameters, z are latent values, and $\Phi(x, z)$ is a feature vector. In the case of one of our star models β is the concatenation of the root filter, the part filters, and deformation cost weights, z is a specification of the object configuration, and $\Phi(x, z)$ is a concatenation of subwindows from a feature pyramid and part deformation features.

We note that (1) can handle very general forms of latent information. For example, z could specify a derivation under a rich visual grammar.

Our second class of models represents an object category by a mixture of star models. The score of a mixture model at a particular position and scale is the maximum over components, of the score of that component model at the given location. In this case the latent information, z, specifies a component label and a configuration for that component. Figure 2 shows a mixture model for the bicycle category.

To obtain high performance using discriminative training it is often important to use large training sets. In the case of object detection the training problem is highly unbalanced because there is vastly more background than objects. This motivates a process of searching through

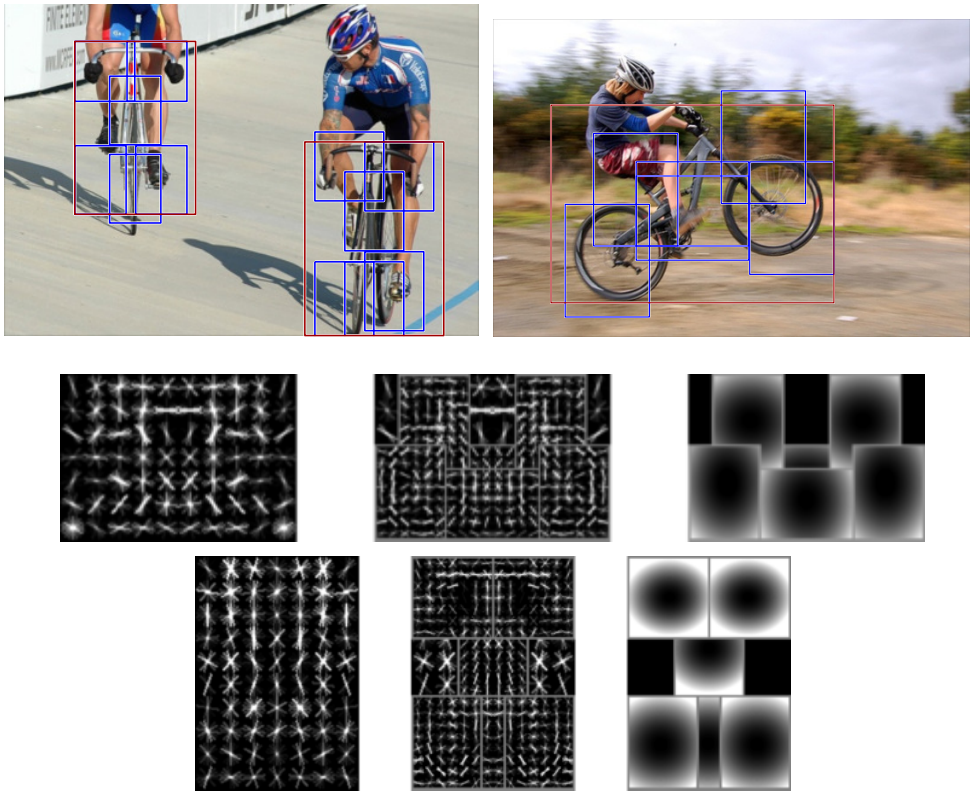


Fig. 2. Detections obtained with a 2 component bicycle model. These examples illustrate the importance of deformations mixture models. In this model the first component captures sideways views of bicycles while the second component captures frontal and near frontal views. The sideways component can deform to match a "wheelie".

the background data to find a relatively small number of potential false positives, or hard negative examples.

A methodology of data-mining for hard negative examples was adopted by Dalal and Triggs [10] but goes back at least to the bootstrapping methods used by [38] and [35]. Here we analyze data-mining algorithms for SVM and LSVM training. We prove that data-mining methods can be made to converge to the optimal model defined in terms of the entire training set.

Our object models are defined by filters that score subwindows of a feature pyramid. We have investigated feature sets similar to the HOG features from [10] and found lower dimensional features which perform as well as the original ones. By doing principal component analysis on HOG features the dimensionality of the feature vector can be significantly reduced with no noticeable loss of information. Moreover, by examining the principal eigenvectors we discover structure that leads to "analytic" versions of low-dimensional features which are easily interpretable and can be computed efficiently.

We have also considered some specific problems that arise in the PASCAL object detection challenge and similar datasets. We show how the locations of parts in an object hypothesis can be used to predict a bounding box for the object. This is done by training a model specific predictor using least-squares regression. We also demonstrate a simple method for aggregating the output of several object detectors. The basic idea is that objects of some categories provide evidence for, or against, objects of other categories in the same image. We exploit this idea by training a category specific classifier that rescores every detection of that category using its original score and the highest scoring detection from each of the other categories.

2 Related Work

There is a significant body of work on deformable models of various types for object detection, including several kinds of deformable template models (e.g. [7], [8], [21], [43]), and a variety of part-based models (e.g. [2], [6], [9], [15], [18], [20], [28], [42]).

In the constellation models from [18], [42] parts are constrained to be in a sparse set of locations determined by an interest point operator, and their geometric arrangement is captured by a Gaussian distribution. In contrast, pictorial structure models [15], [20] define a matching problem where parts have an individual match cost in a dense set of locations, and their geometric arrangement is captured by a set of "springs" connecting pairs of parts. The patchwork of parts model from [2] is similar, but it explicitly considers how the appearance model of overlapping parts interact.

Our models are largely based on the pictorial structures framework from [15], [20]. We use a dense set of possible positions and scales in an image, and define a score for placing a filter at each of these locations.

The geometric configuration of the filters is captured by a set of deformation costs ("springs") connecting each part filter to the root filter, leading to a star-structured pictorial structure model. Note that we do not model interactions between overlapping parts. While we might benefit from modeling such interactions, this does not appear to be a problem when using models trained with a discriminative procedure, and it significantly simplifies the problem of matching a model to an image.

The introduction of new local and semi-local features has played an important role in advancing the performance of object recognition methods. These features are typically invariant to illumination changes and small deformations. Many recent approaches use wavelet-like features [30], [41] or locally-normalized histograms of gradients [10], [29]. Other methods, such as [5], learn dictionaries of local structures from training images. In our work, we use histogram of gradient (HOG) features from [10] as a starting point, and introduce a variation that reduces the feature size with no loss in performance. As in [26] we used principal component analysis (PCA) to discover low dimensional features, but we note that the eigenvectors we obtain have a clear structure that leads to a new set of "analytic" features. This removes the need to perform a costly projection step when computing dense feature maps.

Significant variations in shape and appearance, such as caused by extreme viewpoint changes, are not well captured by a 2D deformable model. Aspect graphs [31] are a classical formalism for capturing significant changes that are due to viewpoint variation. Mixture models provide a simpler alternative approach. For example, it is common to use multiple templates to encode frontal and side views of faces and cars [36]. Mixture models have been used to capture other aspects of appearance variation as well, such as when there are multiple natural subclasses in an object category [5].

Matching a deformable model to an image is a difficult optimization problem. Local search methods require initialization near the correct solution [2], [7], [43]. To guarantee a globally optimal match, more aggressive search is needed. One popular approach for part-based models is to restrict part locations to a small set of possible locations returned by an interest point detector [1], [18], [42]. Tree (and star) structured pictorial structure models [9], [15], [19] allow for the use of dynamic programming and generalized distance transforms to efficiently search over all possible object configurations in an image, without restricting the possible locations for each part. We use these techniques for matching our models to images.

Part-based deformable models are parameterized by the appearance of each part and a geometric model capturing spatial relationships among parts. For generative models one can learn model parameters using maximum likelihood estimation. In a fully-supervised setting training images are labeled with part locations and models can often be learned using simple methods [9], [15]. In a weakly-supervised setting training images may not specify locations of parts. In this case one can simultaneously estimate part locations and learn model parameters with EM [2], [18], [42].

Discriminative training methods select model parameters so as to minimize the mistakes of a detection algorithm on a set of training images. Such approaches directly optimize the decision boundary between positive and negative examples. We believe this is one reason for the success of simple models trained with discriminative methods, such as the Viola-Jones [41] and Dalal-Triggs [10] detectors. It has been more difficult to train part-based models discriminatively, though strategies exist [4], [23], [32], [34].

Latent SVMs are related to hidden CRFs [32]. However, in a latent SVM we maximize over latent part locations as opposed to marginalizing over them, and we use a hinge-loss rather than log-loss in training. This leads to an an efficient coordinate-descent style algorithm for training, as well as a data-mining algorithm that allows for learning with very large datasets. A latent SVM can be viewed as a type of energy-based model [27].

A latent SVM is equivalent to the MI-SVM formulation of multiple instance learning (MIL) in [3], but we find the latent variable formulation more natural for the problems we are interested in.[1] A different MIL framework was previously used for training object detectors with weakly labeled data in [40].

Our method for data-mining hard examples during training is related to working set methods for SVMs (e.g. [25]). The approach described here requires relatively few passes through the complete set of training examples and is particularly well suited for training with very large data sets, where only a fraction of the examples can fit in RAM.

The use of context for object detection and recognition has received increasing attention in the recent years. Some methods (e.g. [39]) use low-level holistic image features for defining likely object hypothesis. The method in [22] uses a coarse but semantically rich representation of a scene, including its 3D geometry, estimated using a variety of techniques. Here we define the context of an image using the results of running a variety of object detectors in the image. The idea is related to [33] where a CRF was used to capture co-occurrences of objects, although we use a very different approach to capture this information.

A preliminary version of our system was described in [17]. The system described here differs from the one in [17] in several ways, including: the introduction of mixture models; here we optimize the true latent SVM objective function using stochastic gradient descent, while in [17] we used an SVM package to optimize a heuristic approximation of the objective; here we use new features that are both lower-dimensional and more informative;

1. We defined a latent SVM in [17] before realizing the relationship to MI-SVM.

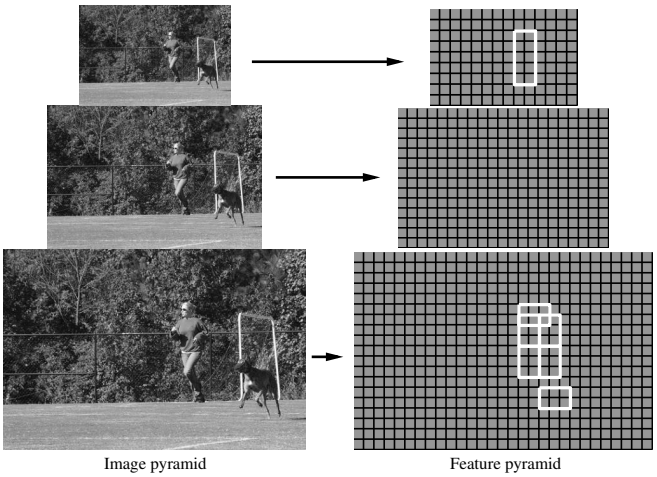


Fig. 3. A feature pyramid and an instantiation of a person model within that pyramid. The part filters are placed at twice the spatial resolution of the placement of the root.

we now post-process detections via bounding box prediction and context rescoring.

3 Models

All of our models involve linear filters that are applied to dense feature maps. A feature map is an array whose entries are d-dimensional feature vectors computed from a dense grid of locations in an image. Intuitively each feature vector describes a local image patch. In practice we use a variation of the HOG features from [10], but the framework described here is independent of the specific choice of features.

A filter is a rectangular template defined by an array of d-dimensional weight vectors. The response, or score, of a filter F at a position (x, y) in a feature map G is the "dot product" of the filter and a subwindow of the feature map with top-left corner at (x, y),

$$\sum_{x',y'} F[x', y'] \cdot G[x + x', y + y'].$$

We would like to define a score at different positions and scales in an image. This is done using a feature pyramid, which specifies a feature map for a finite number of scales in a fixed range. In practice we compute feature pyramids by computing a standard image pyramid via repeated smoothing and subsampling, and then computing a feature map from each level of the image pyramid. Figure 3 illustrates the construction.

The scale sampling in a feature pyramid is determined by a parameter λ defining the number of levels in an octave. That is, λ is the number of levels we need to go down in the pyramid to get to a feature map computed at twice the resolution of another one. In practice we have used $\lambda = 5$ in training and $\lambda = 10$ at test time. Fine sampling of scale space is important for obtaining high performance with our models.

The system in [10] uses a single filter to define an object model. That system detects objects from a particular category by computing the score of the filter at each position and scale of a HOG feature pyramid and thresholding the scores.

Let F be a $w \times h$ filter. Let H be a feature pyramid and $p = (x, y, l)$ specify a position (x, y) in the l-th level of the pyramid. Let $\phi(H, p, w, h)$ denote the vector obtained by concatenating the feature vectors in the $w \times h$ subwindow of H with top-left corner at p in row-major order. The score of F at p is $F' \cdot \phi(H, p, w, h)$, where F' is the vector obtained by concatenating the weight vectors in F in row-major order. Below we write $F' \cdot \phi(H, p)$ since the subwindow dimensions are implicitly defined by the dimensions of the filter F.

3.1 Deformable Part Models

Our star models are defined by a coarse root filter that approximately covers an entire object and higher resolution part filters that cover smaller parts of the object. Figure 3 illustrates an instantiation of such a model in a feature pyramid. The root filter location defines a detection window (the pixels contributing to the part of the feature map covered by the filter). The part filters are placed λ levels down in the pyramid, so the features at that level are computed at twice the resolution of the features in the root filter level.

We have found that using higher resolution features for defining part filters is essential for obtaining high recognition performance. With this approach the part filters capture finer resolution features that are localized to greater accuracy when compared to the features captured by the root filter. Consider building a model for a face. The root filter could capture coarse resolution edges such as the face boundary while the part filters could capture details such as eyes, nose and mouth.

A model for an object with n parts is formally defined by a $(n + 2)$-tuple $(F_0, P_1, \ldots, P_n, b)$ where F_0 is a root filter, P_i is a model for the i-th part and b is a real-valued bias term. Each part model is defined by a 3-tuple (F_i, v_i, d_i) where F_i is a filter for the i-th part, v_i is a two-dimensional vector specifying an "anchor" position for part i relative to the root position, and d_i is a four-dimensional vector specifying coefficients of a quadratic function defining a deformation cost for each possible placement of the part relative to the anchor position.

An object hypothesis specifies the location of each filter in the model in a feature pyramid, $z = (p_0, \ldots, p_n)$, where $p_i = (x_i, y_i, l_i)$ specifies the level and position of the i-th filter. We require that the level of each part is such that the feature map at that level was computed at twice the resolution of the root level, $l_i = l_0 - \lambda$ for $i > 0$.

The score of a hypothesis is given by the scores of each filter at their respective locations (the data term) minus a deformation cost that depends on the relative position of each part with respect to the root (the spatial prior),

plus the bias,

$$\text{score}(p_0,\ldots,p_n) = \sum_{i=0}^{n} F_i' \cdot \phi(H,p_i) - \sum_{i=1}^{n} d_i \cdot \phi_d(dx_i,dy_i) + b, \quad (2)$$

where

$$(dx_i,dy_i) = (x_i,y_i) - (2(x_0,y_0)+v_i) \quad (3)$$

gives the displacement of the i-th part relative to its anchor position and

$$\phi_d(dx,dy) = (dx,dy,dx^2,dy^2) \quad (4)$$

are deformation features.

Note that if $d_i = (0,0,1,1)$ the deformation cost for the i-th part is the squared distance between its actual position and its anchor position relative to the root. In general the deformation cost is an arbitrary separable quadratic function of the displacements.

The bias term is introduced in the score to make the scores of multiple models comparable when we combine them into a mixture model.

The score of a hypothesis z can be expressed in terms of a dot product, $\beta \cdot \psi(H,z)$, between a vector of model parameters β and a vector $\psi(H,z)$,

$$\beta = (F_0',\ldots,F_n',d_1,\ldots,d_n,b). \quad (5)$$

$$\psi(H,z) = (\phi(H,p_0),\ldots\phi(H,p_n), -\phi_d(dx_1,dy_1),\ldots,-\phi_d(dx_n,dy_n),1). \quad (6)$$

This illustrates a connection between our models and linear classifiers. We use this relationship for learning the model parameters with the latent SVM framework.

3.2 Matching

To detect objects in an image we compute an overall score for each root location according to the best possible placement of the parts,

$$\text{score}(p_0) = \max_{p_1,\ldots,p_n} \text{score}(p_0,\ldots,p_n). \quad (7)$$

High-scoring root locations define detections while the locations of the parts that yield a high-scoring root location define a full object hypothesis.

By defining an overall score for each root location we can detect multiple instances of an object (we assume there is at most one instance per root location). This approach is related to sliding-window detectors because we can think of $\text{score}(p_0)$ as a score for the detection window specified by the root filter.

We use dynamic programming and generalized distance transforms (min-convolutions) [14], [15] to compute the best locations for the parts as a function of the root location. The resulting method is very efficient, taking $O(nk)$ time once filter responses are computed, where n is the number of parts in the model and k is the total number of locations in the feature pyramid. We briefly describe the method here and refer the reader to [14], [15] for more details.

Let $R_{i,l}(x,y) = F_i' \cdot \phi(H,(x,y,l))$ be an array storing the response of the i-th model filter in the l-th level of the feature pyramid. The matching algorithm starts by computing these responses. Note that $R_{i,l}$ is a cross-correlation between F_i and level l of the feature pyramid.

After computing filter responses we transform the responses of the part filters to allow for spatial uncertainty,

$$D_{i,l}(x,y) = \max_{dx,dy} \left(R_{i,l}(x+dx,y+dy) - d_i \cdot \phi_d(dx,dy)\right). \quad (8)$$

This transformation spreads high filter scores to nearby locations, taking into account the deformation costs. The value $D_{i,l}(x,y)$ is the maximum contribution of the i-th part to the score of a root location that places the anchor of this part at position (x,y) in level l.

The transformed array, $D_{i,l}$, can be computed in linear time from the response array, $R_{i,l}$, using the generalized distance transform algorithm from [14].

The overall root scores at each level can be expressed by the sum of the root filter response at that level, plus shifted versions of transformed and subsampled part responses,

$$\text{score}(x_0,y_0,l_0) = R_{0,l_0}(x_0,y_0) + \sum_{i=1}^{n} D_{i,l_0-\lambda}(2(x_0,y_0)+v_i) + b. \quad (9)$$

Recall that λ is the number of levels we need to go down in the feature pyramid to get to a feature map that was computed at exactly twice the resolution of another one.

Figure 4 illustrates the matching process.

To understand equation (9) note that for a fixed root location we can independently pick the best location for each part because there are no interactions among parts in the score of a hypothesis. The transformed arrays $D_{i,l}$ give the contribution of the i-th part to the overall root score, as a function of the anchor position for the part. So we obtain the total score of a root position at level l by adding up the root filter response and the contributions from each part, which are precomputed in $D_{i,l-\lambda}$.

In addition to computing $D_{i,l}$ the algorithm from [14] can also compute optimal displacements for a part as a function of its anchor position,

$$P_{i,l}(x,y) = \operatorname*{argmax}_{dx,dy} \left(R_{i,l}(x+dx,y+dy) - d_i \cdot \phi_d(dx,dy)\right). \quad (10)$$

After finding a root location (x_0,y_0,l_0) with high score we can find the corresponding part locations by looking up the optimal displacements in $P_{i,l_0-\lambda}(2(x_0,y_0)+v_i)$.

3.3 Mixture Models

A mixture model with m components is defined by a m-tuple, $M = (M_1,\ldots,M_m)$, where M_c is the model for the c-th component.

An object hypothesis for a mixture model specifies a mixture component, $1 \le c \le m$, and a location for each

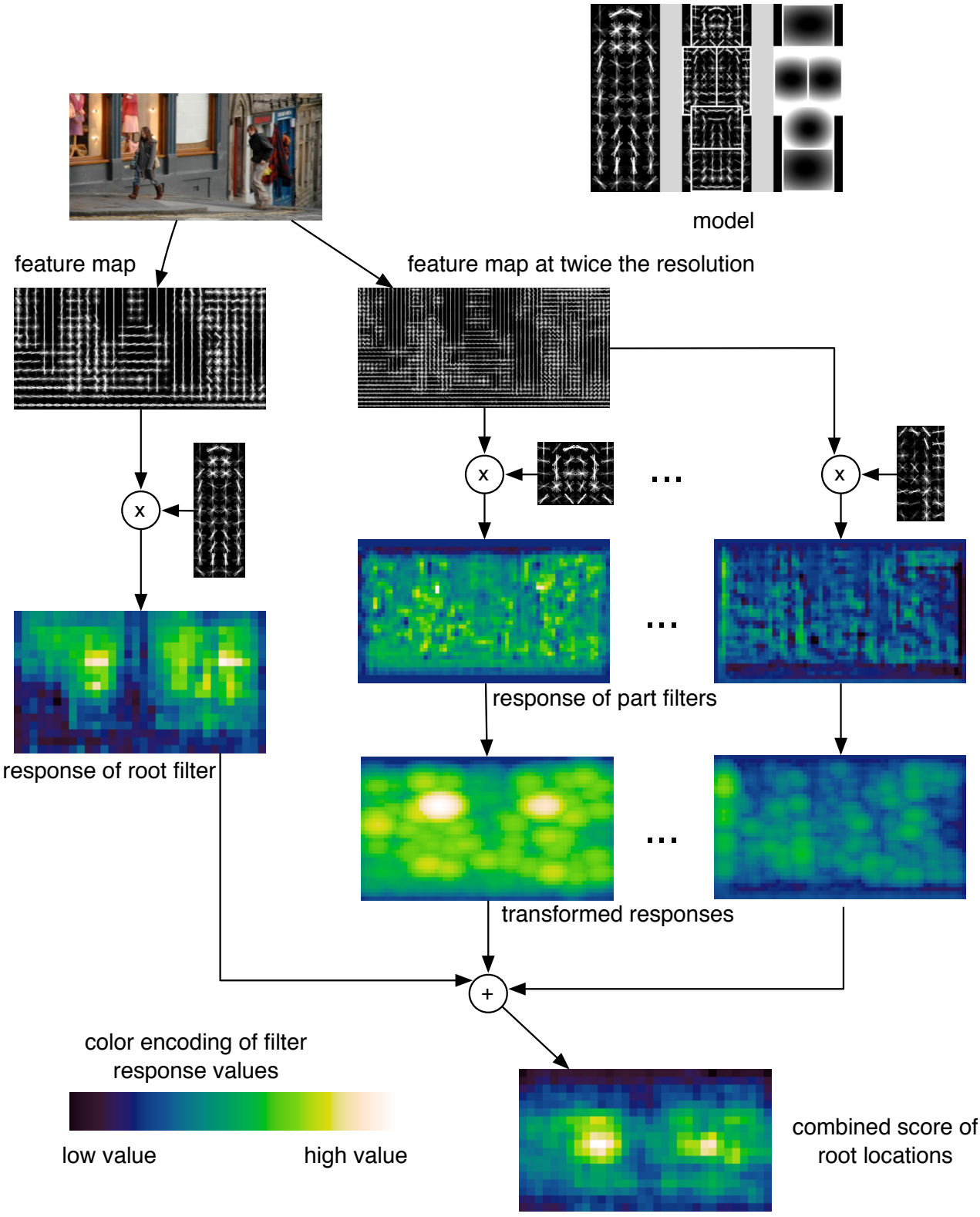


Fig. 4. The matching process at one scale. Responses from the root and part filters are computed a different resolutions in the feature pyramid. The transformed responses are combined to yield a final score for each root location. We show the responses and transformed responses for the “head” and “right shoulder” parts. Note how the “head” filter is more discriminative. The combined scores clearly show two good hypothesis for the object at this scale.

filter of M_c, $z = (c, p_0, \ldots, p_{n_c})$. Here n_c is the number of parts in M_c. The score of this hypothesis is the score of the hypothesis $z' = (p_0, \ldots, p_{n_c})$ for the c-th model component.

As in the case of a single component model the score of a hypothesis for a mixture model can be expressed by a dot product between a vector of model parameters β and a vector $\psi(H, z)$. For a mixture model the vector β is the concatenation of the model parameter vectors for each component. The vector $\psi(H, z)$ is sparse, with non-zero entries defined by $\psi(H, z')$ in a single interval matching the interval of β_c in β,

$$\beta = (\beta_1, \ldots, \beta_m). \tag{11}$$

$$\psi(H, z) = (0, \ldots, 0, \psi(H, z'), 0, \ldots, 0). \tag{12}$$

With this construction $\beta \cdot \psi(H, z) = \beta_c \cdot \psi(H, z')$.

To detect objects using a mixture model we use the matching algorithm described above to find root locations that yield high scoring hypotheses independently for each component.

4 Latent SVM

Consider a classifier that scores an example x with a function of the form,

$$f_\beta(x) = \max_{z \in Z(x)} \beta \cdot \Phi(x, z). \tag{13}$$

Here β is a vector of model parameters and z are latent values. The set $Z(x)$ defines the possible latent values for an example x. A binary label for x can be obtained by thresholding its score.

In analogy to classical SVMs we train β from labeled examples $D = (\langle x_1, y_1 \rangle, \ldots, \langle x_n, y_n \rangle)$, where $y_i \in \{-1, 1\}$, by minimizing the objective function,

$$L_D(\beta) = \frac{1}{2}||\beta||^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i f_\beta(x_i)), \tag{14}$$

where $\max(0, 1 - y_i f_\beta(x_i))$ is the standard hinge loss and the constant C controls the relative weight of the regularization term.

Note that if there is a single possible latent value for each example ($|Z(x_i)| = 1$) then f_β is linear in β, and we obtain linear SVMs as a special case of latent SVMs.

4.1 Semi-convexity

A latent SVM leads to a non-convex optimization problem. However, a latent SVM is *semi-convex* in the sense described below, and the training problem becomes convex once latent information is specified for the positive training examples.

Recall that the maximum of a set of convex functions is convex. In a linear SVM we have that $f_\beta(x) = \beta \cdot \Phi(x)$ is linear in β. In this case the hinge loss is convex for each example because it is always the maximum of two convex functions.

Note that $f_\beta(x)$ as defined in (13) is a maximum of functions each of which is linear in β. Hence $f_\beta(x)$ is convex in β and thus the hinge loss, $\max(0, 1 - y_i f_\beta(x_i))$, is convex in β when $y_i = -1$. That is, the loss function is convex in β for negative examples. We call this property of the loss function *semi-convexity*.

In a general latent SVM the hinge loss is not convex for a positive example because it is the maximum of a convex function (zero) and a concave function ($1 - y_i f_\beta(x_i)$).

Now consider a latent SVM where there is a single possible latent value for each positive example. In this case $f_\beta(x_i)$ is linear for a positive example and the loss due to each positive is convex. Combined with the semi-convexity property, (14) becomes convex.

4.2 Optimization

Let Z_p specify a latent value for each positive example in a training set D. We can define an auxiliary objective function $L_D(\beta, Z_p) = L_{D(Z_p)}(\beta)$, where $D(Z_p)$ is derived from D by restricting the latent values for the positive examples according to Z_p. That is, for a positive example we set $Z(x_i) = \{z_i\}$ where z_i is the latent value specified for x_i by Z_p. Note that

$$L_D(\beta) = \min_{Z_p} L_D(\beta, Z_p). \tag{15}$$

In particular $L_D(\beta) \leq L_D(\beta, Z_p)$. The auxiliary objective function bounds the LSVM objective. This justifies training a latent SVM by minimizing $L_D(\beta, Z_p)$.

In practice we minimize $L_D(\beta, Z_p)$ using a "coordinate descent" approach:

1) *Relabel positive examples:* Optimize $L_D(\beta, Z_p)$ over Z_p by selecting the highest scoring latent value for each positive example,
   $z_i = \text{argmax}_{z \in Z(x_i)} \beta \cdot \Phi(x_i, z)$.
2) *Optimize beta:* Optimize $L_D(\beta, Z_p)$ over β by solving the convex optimization problem defined by $L_{D(Z_p)}(\beta)$.

Both steps always improve or maintain the value of $L_D(\beta, Z_p)$. After convergence we have a relatively strong local optimum in the sense that step 1 searches over an exponentially-large space of latent values for positive examples while step 2 searches over all possible models, implicitly considering the exponentially-large space of latent values for all negative examples.

We note, however, that careful initialization of β may be necessary because otherwise we may select unreasonable latent values for the positive examples in step 1, and this could lead to a bad model.

The semi-convexity property is important because it leads to a convex optimization problem in step 2, even though the latent values for the negative examples are not fixed. A similar procedure that fixes latent values for all examples in each round would likely fail to yield good results. Suppose we let Z specify latent values for all examples in D. Since $L_D(\beta)$ effectively maximizes over negative latent values, $L_D(\beta)$ could be much larger than $L_D(\beta, Z)$, and we should not expect that minimizing $L_D(\beta, Z)$ would lead to a good model.

4.3 Stochastic gradient descent

Step 2 (*Optimize Beta*) of the coordinate descent method can be solved via quadratic programming [3]. It can also be solved via stochastic gradient descent. Here we describe a gradient descent approach for optimizing β over an arbitrary training set D. In practice we use a modified version of this procedure that works with a cache of feature vectors for $D(Z_p)$ (see Section 4.5).

Let $z_i(\beta) = \text{argmax}_{z \in Z(x_i)} \beta \cdot \Phi(x_i, z)$.

Then $f_\beta(x_i) = \beta \cdot \Phi(x_i, z_i(\beta))$.

We can compute a sub-gradient of the LSVM objective function as follows,

$$\nabla L_D(\beta) = \beta + C \sum_{i=1}^{n} h(\beta, x_i, y_i) \tag{16}$$

$$h(\beta, x_i, y_i) = \begin{cases} 0 & \text{if } y_i f_\beta(x_i) \geq 1 \\ -y_i \Phi(x_i, z_i(\beta)) & \text{otherwise} \end{cases} \tag{17}$$

In stochastic gradient descent we approximate ∇L_D using a subset of the examples and take a step in its negative direction. Using a single example, $\langle x_i, y_i \rangle$, we approximate $\sum_{i=1}^{n} h(\beta, x_i, y_i)$ with $nh(\beta, x_i, y_i)$. The resulting algorithm repeatedly updates β as follows:

1) Let α_t be the learning rate for iteration t.
2) Let i be a random example.
3) Let $z_i = \text{argmax}_{z \in Z(x_i)} \beta \cdot \Phi(x_i, z)$.
4) If $y_i f_\beta(x_i) = y_i(\beta \cdot \Phi(x_i, z_i)) \geq 1$ set $\beta := \beta - \alpha_t \beta$.
5) Else set $\beta := \beta - \alpha_t(\beta - Cny_i \Phi(x_i, z_i))$.

As in gradient descent methods for linear SVMs we obtain a procedure that is quite similar to the perceptron algorithm. If f_β correctly classifies the random example x_i (beyond the margin) we simply shrink β. Otherwise we shrink β and add a scalar multiple of $\Phi(x_i, z_i)$ to it.

For linear SVMs a learning rate $\alpha_t = 1/t$ has been shown to work well [37]. However, the time for convergence depends on the number of training examples, which for us can be very large. In particular, if there are many "easy" examples, step 2 will often pick one of these and we do not make much progress.

4.4 Data-mining hard examples, SVM version

When training a model for object detection we often have a very large number of negative examples (a single image can yield 10^5 examples for a scanning window classifier). This can make it infeasible to consider all negative examples simultaneously. Instead, it is common to construct training data consisting of the positive instances and "hard negative" instances.

Bootstrapping methods train a model with an initial subset of negative examples, and then collect negative examples that are incorrectly classified by this initial model to form a set of hard negatives. A new model is trained with the hard negative examples and the process may be repeated a few times.

Here we describe a data-mining algorithm motivated by the bootstrapping idea for training a classical (non-latent) SVM. The method solves a sequence of training problems using a relatively small number of hard examples and converges to the *exact* solution of the training problem defined by a large training set. This requires a margin-sensitive definition of hard examples.

We define hard and easy instances of a training set D relative to β as follows,

$$H(\beta, D) = \{\langle x, y \rangle \in D \mid y f_\beta(x) < 1\}. \tag{18}$$

$$E(\beta, D) = \{\langle x, y \rangle \in D \mid y f_\beta(x) > 1\}. \tag{19}$$

That is, $H(\beta, D)$ are the examples in D that are incorrectly classified or inside the margin of the classifier defined by β. Similarly $E(\beta, D)$ are the examples in D that are correctly classified and outside the margin. Examples on the margin are neither hard nor easy.

Let $\beta^*(D) = \text{argmin}_\beta L_D(\beta)$.

Since L_D is strictly convex $\beta^*(D)$ is unique.

Given a large training set D we would like to find a small set of examples $C \subseteq D$ such that $\beta^*(C) = \beta^*(D)$.

Our method starts with an initial "cache" of examples and alternates between training a model and updating the cache. In each iteration we remove easy examples from the cache and add new hard examples. A special case involves keeping all positive examples in the cache and data-mining over negatives.

Let $C_1 \subseteq D$ be an initial cache of examples. The algorithm repeatedly trains a model and updates the cache as follows:

1) Let $\beta_t := \beta^*(C_t)$ (train a model using C_t).
2) If $H(\beta_t, D) \subseteq C_t$ stop and return β_t.
3) Let $C'_t := C_t \backslash X$ for any X such that $X \subseteq E(\beta_t, C_t)$ (shrink the cache).
4) Let $C_{t+1} := C'_t \cup X$ for any X such that $X \subseteq D$ and $X \cap H(\beta_t, D) \backslash C_t \neq \emptyset$ (grow the cache).

In step 3 we shrink the cache by removing examples from C_t that are outside the margin defined by β_t. In step 4 we grow the cache by adding examples from D, including at least one new example that is inside the margin defined by β_t. Such example must exist otherwise we would have returned in step 2.

The following theorem shows that when we stop we have found $\beta^*(D)$.

Theorem 1: Let $C \subseteq D$ and $\beta = \beta^*(C)$. If $H(\beta, D) \subseteq C$ then $\beta = \beta^*(D)$.

Proof: $C \subseteq D$ implies $L_D(\beta^*(D)) \geq L_C(\beta^*(C)) = L_C(\beta)$. Since $H(\beta, D) \subseteq C$ all examples in $D \backslash C$ have zero loss on β. This implies $L_C(\beta) = L_D(\beta)$. We conclude $L_D(\beta^*(D)) \geq L_D(\beta)$, and because L_D has a unique minimum $\beta = \beta^*(D)$. □

The next result shows the algorithm will stop after a finite number of iterations. Intuitively this follows from the fact that $L_{C_t}(\beta^*(C_t))$ grows in each iteration, but it is bounded by $L_D(\beta^*(D))$.

Theorem 2: The data-mining algorithm terminates.

Proof: When we shrink the cache C'_t contains all examples from C_t with non-zero loss in a ball around β_t. This implies $L_{C'_t}$ is identical to L_{C_t} in a ball around

β_t, and since β_t is a minimum of L_{C_t} it also must be a minimum of $L_{C'_t}$. Thus $L_{C'_t}(\beta^*(C'_t)) = L_{C_t}(\beta^*(C_t))$.

When we grow the cache $C_{t+1} \backslash C'_t$ contains at least one example $\langle x, y \rangle$ with non-zero loss at β_t. Since $C'_t \subseteq C_{t+1}$ we have $L_{C_{t+1}}(\beta) \geq L_{C'_t}(\beta)$ for all β. If $\beta^*(C_{t+1}) \neq \beta^*(C'_t)$ then $L_{C_{t+1}}(\beta^*(C_{t+1})) > L_{C'_t}(\beta^*(C'_t))$ because $L_{C'_t}$ has a unique minimum. If $\beta^*(C_{t+1}) = \beta^*(C'_t)$ then $L_{C_{t+1}}(\beta^*(C_{t+1})) > L_{C'_t}(\beta^*(C'_t))$ due to $\langle x, y \rangle$.

We conclude $L_{C_{t+1}}(\beta^*(C_{t+1})) > L_{C_t}(\beta^*(C_t))$. Since there are finitely many caches the loss in the cache can only grow a finite number of times. □

4.5 Data-mining hard examples, LSVM version

Now we describe a data-mining algorithm for training a latent SVM *when the latent values for the positive examples are fixed*. That is, we are optimizing $L_{D(Z_p)}(\beta)$, and not $L_D(\beta)$. As discussed above this restriction ensures the optimization problem is convex.

For a latent SVM instead of keeping a cache of examples x, we keep a cache of (x, z) pairs where $z \in Z(x)$. This makes it possible to avoid doing inference over all of $Z(x)$ in the inner loop of an optimization algorithm such as gradient descent. Moreover, in practice we can keep a cache of feature vectors, $\Phi(x, z)$, instead of (x, z) pairs. This representation is simpler (its application independent) and can be much more compact.

A feature vector cache F is a set of pairs (i, v) where $1 \leq i \leq n$ is the index of an example and $v = \Phi(x_i, z)$ for some $z \in Z(x_i)$. Note that we may have several pairs $(i, v) \in F$ for each example x_i. If the training set has fixed labels for positive examples this may still be true for the negative examples.

Let $I(F)$ be the examples indexed by F. The feature vectors in F define an objective function for β, where we only consider examples indexed by $I(F)$, and for each example we only consider feature vectors in the cache,

$$L_F(\beta) = \frac{1}{2}||\beta||^2 + C \sum_{i \in I(F)} \max(0, 1 - y_i(\max_{(i,v) \in F} \beta \cdot v)). \quad (20)$$

We can optimize L_F via gradient descent by modifying the method in Section 4.3. Let $V(i)$ be the set of feature vectors v such that $(i, v) \in F$. Then each gradient descent iteration simplifies to:

1) Let α_t be the learning rate for iteration t.
2) Let $i \in I(F)$ be a random example indexed by F.
3) Let $v_i = \text{argmax}_{v \in V(i)} \beta \cdot v$.
4) If $y_i(\beta \cdot v_i) \geq 1$ set $\beta = \beta - \alpha_t \beta$.
5) Else set $\beta = \beta - \alpha_t(\beta - C n y_i v_i)$.

Now the size of $I(F)$ controls the number of iterations necessary for convergence, while the size of $V(i)$ controls the time it takes to execute step 3. In step 5 $n = |I(F)|$.

Let $\beta^*(F) = \text{argmin}_\beta L_F(\beta)$.

We would like to find a small cache for $D(Z_p)$ with $\beta^*(F) = \beta^*(D(Z_p))$.

We define the hard feature vectors of a training set D relative to β as,

$$H(\beta, D) = \{(i, \Phi(x_i, z_i)) \mid z_i = \underset{z \in Z(x_i)}{\text{argmax}} \beta \cdot \Phi(x_i, z) \text{ and } y_i(\beta \cdot \Phi(x_i, z_i)) < 1\}. \quad (21)$$

That is, $H(\beta, D)$ are pairs (i, v) where v is the highest scoring feature vector from an example x_i that is inside the margin of the classifier defined by β.

We define the easy feature vectors in a cache F as

$$E(\beta, F) = \{(i, v) \in F \mid y_i(\beta \cdot v) > 1\} \quad (22)$$

These are the feature vectors from F that are outside the margin defined by β.

Note that if $y_i(\beta \cdot v) \leq 1$ then (i, v) is not considered easy even if there is another feature vector for the i-th example in the cache with higher score than v under β.

Now we describe the data-mining algorithm for computing $\beta^*(D(Z_p))$.

The algorithm works with a cache of feature vectors for $D(Z_p)$. It alternates between training a model and updating the cache.

Let F_1 be an initial cache of feature vectors. Now consider the following iterative algorithm:

1) Let $\beta_t := \beta^*(F_t)$ (train a model).
2) If $H(\beta, D(Z_p)) \subseteq F_t$ stop and return β_t.
3) Let $F'_t := F_t \backslash X$ for any X such that $X \subseteq E(\beta_t, F_t)$ (shrink the cache).
4) Let $F_{t+1} := F'_t \cup X$ for any X such that $X \cap H(\beta_t, D(Z_p)) \backslash F_t \neq \emptyset$ (grow the cache).

Sstep 3 shrinks the cache by removing easy feature vetors. Step 4 grows the cache by adding "new" feature vectors, including at least one from $H(\beta_t, D(Z_p))$. Note that over time we will accumulate multiple feature vectors from the same negative example in the cache.

We can show this algorithm will eventually stop and return $\beta^*(D(Z_p))$. This follows from arguments analogous to the ones used in Section 4.4.

5 Training Models

Now we consider the problem of training models from images labeled with bounding boxes around objects of interest. This is the type of data available in the PASCAL datasets. Each dataset contains thousands of images and each image has annotations specifying a bounding box and a class label for each target object present in the image. Note that this is a weakly labeled setting since the bounding boxes do not specify component labels or part locations.

We describe a procedure for initializing the structure of a mixture model and learning all parameters. Parameter learning is done by constructing a latent SVM training problem. We train the latent SVM using the coordinate descent approach described in Section 4.2 together with the data-mining and gradient descent algorithms that work with a cache of feature vectors

from Section 4.5. Since the coordinate descent method is susceptible to local minima we must take care to ensure a good initialization of the model.

5.1 Learning parameters

Let c be an object class. We assume the training examples for c are given by positive bounding boxes P and a set of background images N. P is a set of pairs (I, B) where I is an image and B is a bounding box for an object of class c in I.

Let M be a (mixture) model with fixed structure. Recall that the parameters for a model are defined by a vector β. To learn β we define a latent SVM training problem with an implicitly defined training set D, with positive examples from P, and negative examples from N.

Each example $\langle x, y \rangle \in D$ has an associated image and feature pyramid $H(x)$. Latent values $z \in Z(x)$ specify an instantiation of M in the feature pyramid $H(x)$.

Now define $\Phi(x, z) = \psi(H(x), z)$. Then $\beta \cdot \Phi(x, z)$ is exactly the score of the hypothesis z for M on $H(x)$.

A positive bounding box $(I, B) \in P$ specifies that the object detector should "fire" in a location defined by B. This means the overall score (7) of a root location defined by B should be high.

For each $(I, B) \in P$ we define a positive example x for the LSVM training problem. We define $Z(x)$ so the detection window of a root filter specified by a hypothesis $z \in Z(x)$ overlaps with B by at least 50%. There are usually many root locations, including at different scales, that define detection windows with 50% overlap. We have found that treating the root location as a latent variable is helpful to compensate for noisy bounding box labels in P. A similar idea was used in [40].

Now consider a background image $I \in N$. We do not want the object detector to "fire" in any location of the feature pyramid for I. This means the overall score (7) of every root location should be low. Let $\mathcal{G}$ be a dense set of locations in the feature pyramid. We define a different negative example x for each location $(i, j, l) \in \mathcal{G}$. We define $Z(x)$ so the level of the root filter specified by $z \in Z(x)$ is l, and the center of its detection window is (i, j). Note that there is a very large number of negative examples obtained from each image. This is consistent with the requirement that a scanning window classifier should have low false positive rate.

The procedure `Train` is outlined below. The outermost loop implements a fixed number of iterations of coordinate descent on $L_D(\beta, Z_p)$. Lines 3-6 implement the *Relabel positives* step. The resulting feature vectors, one per positive example, are stored in F_p. Lines 7-14 implement the *Optimize beta* step. Since the number of negative examples implicitly defined by N is very large we use the LSVM data-mining algorithm. We iterate data-mining a fixed number of times rather than until convergence for practical reasons. At each iteration we collect hard negative examples in F_n, train a new model using gradient descent, and then shrink F_n by removing easy feature vectors. During data-mining we grow the cache by iterating over the images in N sequentially, until we reach a memory limit.

Data:
Positive examples $P = \{(I_1, B_1), \ldots, (I_n, B_n)\}$
Negative images $N = \{J_1, \ldots, J_m\}$
Initial model β

Result: New model β

```
1  F_n := ∅
2  for relabel := 1 to num-relabel do
3  |   F_p := ∅
4  |   for i := 1 to n do
5  |   |   Add detect-best(β,I_i,B_i) to F_p
6  |   end
7  |   for datamine := 1 to num-datamine do
8  |   |   for j := 1 to m do
9  |   |   |   if |F_n| ≥ memory-limit then break
10 |   |   |   Add detect-all(β,J_j,−(1 + δ)) to F_n
11 |   |   end
12 |   |   β :=gradient-descent(F_p ∪ F_n)
13 |   |   Remove (i, v) with β · v < −(1 + δ) from F_n
14 |   end
15 end
```

Procedure `Train`

The function `detect-best`(β, I, B) finds the highest scoring object hypothesis with a root filter that significantly overlaps B in I. The function `detect-all`(β, I, t) computes the best object hypothesis for each root location and selects the ones that score above t. Both of these functions can be implemented using the matching procedure in Section 3.2.

The function `gradient-descent`(F) trains β using feature vectors in the cache as described in Section 4.5. In practice we modified the algorithm to constrain the coefficients of the quadratic terms in the deformation models to be above 0.01. This ensures the deformation costs are convex, and not "too flat". We also constrain the model to be symmetric along the vertical axis. Filters that are positioned along the center vertical axis of the model are constrained to be self-symmetric. Part filters that are off-center have a symmetric part on the other side of the model. This effectively reduces the number of parameters to be learned in half.

5.2 Initialization

The LSVM coordinate descent algorithm is susceptible to local minima and thus sensitive to initialization. This is a common limitation of other methods that use latent information as well. We initialize and train mixture models in three phases as follows.

Phase 1. Initializing Root Filters: For training a mixture model with m components we sort the bounding boxes in P by their aspect ratio and split them into m groups of equal size $P_1, \ldots, P_m$. Aspect ratio is used as a simple indicator of extreme intraclass variation. We train

m different root filters $F_1, \ldots, F_m$, one for each group of positive bounding boxes.

To define the dimensions of F_i we select the mean aspect ratio of the boxes in P_i and the largest area not larger than 80% of the boxes. This ensures that for most pairs $(I, B) \in P_i$ we can place F_i in the feature pyramid of I so it significantly overlaps with B.

We train F_i using a standard SVM, with no latent information, as in [10]. For $(I, B) \in P_i$ we warp the image region under B so its feature map has the same dimensions as F_i. This leads to a positive example. We select random subwindows of appropriate dimension from images in N to define negative examples. Figures 5(a) and 5(b) show the result of this phase when training a two component car model.

Phase 2. Merging Components: We combine the initial root filters into a mixture model with no parts and retrain the parameters of the combined model using `Train` on the full (unsplit and without warping) data sets P and N. In this case the component label and root location are the only latent variables for each example. The coordinate descent training algorithm can be thought of as a discriminative clustering method that alternates between assigning cluster (mixture) labels for each positive example and estimating cluster "means" (root filters).

Phase 3. Initializing Part Filters: We initialize the parts of each component using a simple heuristic. We fix the number of parts at six per component, and using a small pool of rectangular part shapes we greedily place parts to cover high-energy regions of the root filter.[2] A part is either anchored along the central vertical axis of the root filter, or it is off-center and has a symmetric part on the other side of the root filter. Once a part is placed, the energy of the covered portion of the root filter is set to zero, and we look for the next highest-energy region, until six parts are chosen.

The part filters are initialized by interpolating the root filter to twice the spatial resolution. The deformation parameters for each part are initialized to $d_i = (0, 0, .1, .1)$. This pushes part locations to be fairly close to their anchor position. Figure 5(c) shows the results of this phase when training a two component car model. The resulting model serves as the initial model for the last round of parameter learning. The final car model is shown in Figure 9.

6 Features

Here we describe the 36-dimensional histogram of oriented gradients (HOG) features from [10] and introduce an alternative 13-dimensional feature set that captures essentially the same information.[3] We have found that augmenting this low-dimensional feature set to include both contrast sensitive and contrast insensitive features, leading to a 31-dimensional feature vector, improves performance for most classes of the PASCAL datasets.

6.1 HOG Features

6.1.1 Pixel-Level Feature Maps

Let $\theta(x, y)$ and $r(x, y)$ be the orientation and magnitude of the intensity gradient at a pixel (x, y) in an image. As in [10], we compute gradients using finite difference filters, $[-1, 0, +1]$ and its transpose. For color images we use the color channel with the largest gradient magnitude to define θ and r at each pixel.

The gradient orientation at each pixel is discretized into one of p values using either a contrast sensitive (B_1), or insensitive (B_2), definition,

$$B_1(x, y) = \text{round}\left(\frac{p\theta(x, y)}{2\pi}\right) \bmod p \quad (23)$$

$$B_2(x, y) = \text{round}\left(\frac{p\theta(x, y)}{\pi}\right) \bmod p \quad (24)$$

Below we use B to denote either B_1 or B_2.

We define a pixel-level feature map that specifies a sparse histogram of gradient magnitudes at each pixel. Let $b \in \{0, \ldots, p-1\}$ range over orientation bins. The feature vector at (x, y) is

$$F(x, y)_b = \begin{cases} r(x, y) & \text{if } b = B(x, y) \\ 0 & \text{otherwise} \end{cases} \quad (25)$$

We can think of F as an oriented edge map with p orientation channels. For each pixel we select a channel by discretizing the gradient orientation. The gradient magnitude can be seen as a measure of edge strength.

6.1.2 Spatial Aggregation

Let F be a pixel-level feature map for a $w \times h$ image. Let $k > 0$ be a parameter specifying the side length of a square image region. We define a dense grid of rectangular "cells" and aggregate pixel-level features to obtain a cell-based feature map C, with feature vectors $C(i, j)$ for $0 \le i \le \lfloor (w-1)/k \rfloor$ and $0 \le j \le \lfloor (h-1)/k \rfloor$. This aggregation provides some invariance to small deformations and reduces the size of a feature map.

The simplest approach for aggregating features is to map each pixel (x, y) into a cell $(\lfloor x/k \rfloor, \lfloor y/k \rfloor)$ and define the feature vector at a cell to be the sum (or average) of the pixel-level features in that cell.

Rather than mapping each pixel to a unique cell we follow [10] and use a "soft binning" approach where each pixel contributes to the feature vectors in the four cells around it using bilinear interpolation.

6.1.3 Normalization and Truncation

Gradients are invariant to changes in bias. Invariance to gain can be achieved via normalization. Dalal and Triggs [10] used four different normalization factors for

2. The "energy" of a region is defined by the norm of the positive weights in a subwindow.

3. There are some small differences between the 36-dimensional features defined here and the ones in [10], but we have found that these differences did not have any significant effect on the performance of our system.

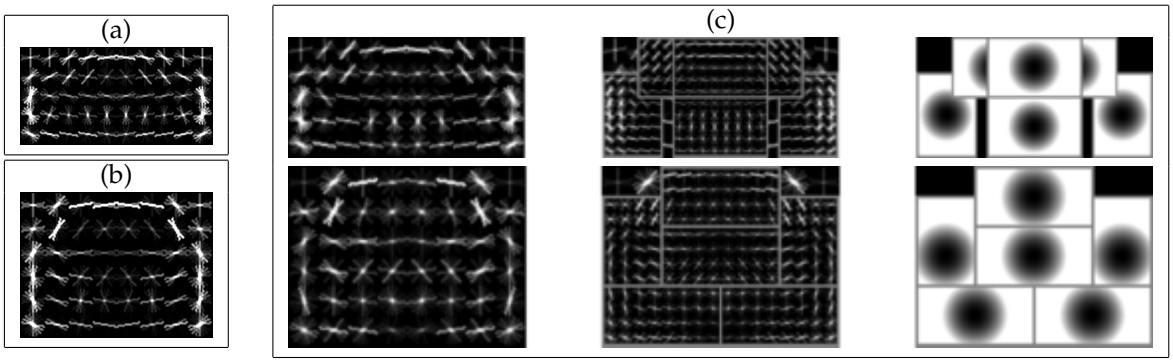


Fig. 5. (a) and (b) are the initial root filters for a car model (the result of Phase 1 of the initialization process). (c) is the initial part-based model for a car (the result of Phase 3 of the initialization process).

the feature vector $C(i,j)$. We can write these factors as $N_{\delta,\gamma}(i,j)$ with $\delta,\gamma \in \{-1,1\}$,

$$N_{\delta,\gamma}(i,j) = (||C(i,j)||^2 + ||C(i+\delta,j)||^2 + ||C(i,j+\gamma)||^2 + ||C(i+\delta,j+\gamma)||^2)^{\frac{1}{2}}. \quad (26)$$

Each factor measures the "gradient energy" in a square block of four cells containing (i,j).

Let $T_\alpha(v)$ denote the component-wise truncation of a vector v by α (the i-th entry in $T_\alpha(v)$ is the minimum of the i-th entry of v and α). The HOG feature map is obtained by concatenating the result of normalizing the cell-based feature map C with respect to each normalization factor followed by truncation,

$$H(i,j) = \begin{pmatrix} T_\alpha(C(i,j)/N_{-1,-1}(i,j)) \\ T_\alpha(C(i,j)/N_{+1,-1}(i,j)) \\ T_\alpha(C(i,j)/N_{+1,+1}(i,j)) \\ T_\alpha(C(i,j)/N_{-1,+1}(i,j)) \end{pmatrix} \quad (27)$$

Commonly used HOG features are defined using $p = 9$ contrast insensitive gradient orientations (discretized with B_2), a cell size of $k = 8$ and truncation $\alpha = 0.2$. This leads to a 36-dimensional feature vector. We used these parameters in the analysis described below.

6.2 PCA and Analytic Dimensionality Reduction

We collected a large number of 36-dimensional HOG features from different resolutions of a large number of images and performed PCA on these vectors. The principal components are shown in Figure 6. The results lead to a number of interesting discoveries.

The eigenvalues indicate that the linear subspace spanned by the top 11 eigenvectors captures essentially all the information in a HOG feature. In fact we obtain the same detection performance in all categories of the PASCAL 2007 dataset using the original 36-dimensional features or 11-dimensional features defined by projection to the top eigenvectors. Using lower dimensional features leads to models with fewer parameters and speeds up the detection and learning algorithms. We note however that some of the gain is lost because we need to perform a relatively costly projection step when computing feature pyramids.

Recall that a 36-dimensional HOG feature is defined using 4 different normalizations of a 9 dimensional histogram over orientations. Thus a 36-dimensional HOG feature is naturally viewed as a 4×9 matrix. The top eigenvectors in Figure 6 have a very special structure: they are each (approximately) constant along each row or column of their matrix representation. Thus the top eigenvectors lie (approximately) in a linear subspace defined by sparse vectors that have ones along a single row or column of their matrix representation.

Let $V = \{u_1,\ldots,u_9\} \cup \{v_1,\ldots,v_4\}$ with

$$u_k(i,j) = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{otherwise} \end{cases} \quad (28)$$

$$v_k(i,j) = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{otherwise} \end{cases} \quad (29)$$

We can define a 13-dimensional feature by taking the dot product of a 36-dimensional HOG feature with each u_k and v_k. Projection into each u_k is computed by summing over the 4 normalizations for a fixed orientation. Projection into each v_k is computed by summing over 9 orientations for a fixed normalization.[4]

As in the case of 11-dimensional PCA features we obtain the same performance using the 36-dimensional HOG features or the 13-dimensional features defined by V. However, the computation of the 13-dimensional features is much less costly than performing projections to the top eigenvectors obtained via PCA since the u_k and v_k are sparse. Moreover, the 13-dimensional features have a simple interpretation as 9 orientation features and 4 features that reflect the overall gradient energy in different areas around a cell.

We can also define low-dimensional features that are contrast sensitive. We have found that performance on some object categories improves using contrast sensitive features, while some categories benefit from contrast insensitive features. Thus in practice we use feature vectors that include both contrast sensitive and insensitive information.

4. The 13-dimensional feature is not a linear projection of the 36-dimensional feature into V because the u_k and v_k are not orthogonal. In fact the linear subspace spanned by V has dimension 12.

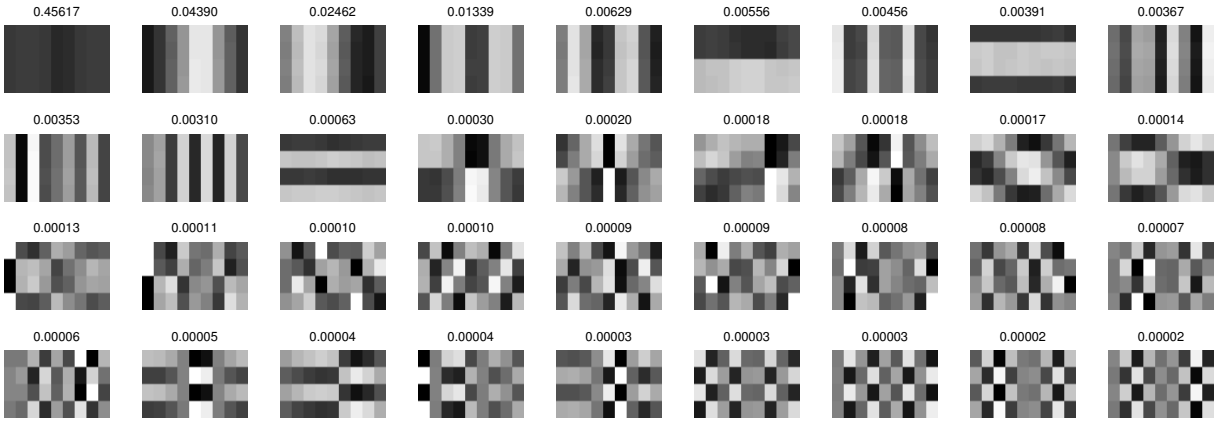


Fig. 6. PCA of HOG features. Each eigenvector is displayed as a 4 by 9 matrix so that each row corresponds to one normalization factor and each column to one orientation bin. The eigenvalues are displayed on top of the eigenvectors. The linear subspace spanned by the top 11 eigenvectors captures essentially all of the information in a feature vector. Note how all of the top eigenvectors are either constant along each column or row of the matrix representation.

Let C be a cell-based feature map computed by aggregating a pixel-level feature map with 9 contrast insensitive orientations. Let D be a similar cell-based feature map computed using 18 contrast sensitive orientations. We define 4 normalization factors for the (i, j) cell of C and D using C as in equation (26). We can normalize and truncate $C(i, j)$ and $D(i, j)$ using these factors to obtain $4 * (9 + 18) = 108$ dimensional feature vectors, $F(i, j)$. In practice we use an analytic projection of these 108-dimensional vectors, defined by 27 sums over different normalizations, one for each orientation channel of F, and 4 sums over the 9 contrast insensitive orientations, one for each normalization factor. We use a cell size of $k = 8$ and truncation value of $\alpha = 0.2$. The final feature map has 31-dimensional vectors $G(i, j)$, with 27 dimensions corresponding to different orientation channels (9 contrast insensitive and 18 contrast sensitive), and 4 dimensions capturing the overall gradient energy in square blocks of four cells around (i, j).

Finally, we note that the top eigenvectors in Figure 6 can be roughly interpreted as a two-dimensional separable Fourier basis. Each eigenvector can be roughly seen as a sine or cosine function of one variable. This observation could be used to define features using a finite number of Fourier basis functions instead of a finite number of discrete orientations.

The appearance of Fourier basis in Figure 6 is an interesting empirical result. The eigenvectors of a $d \times d$ covariance matrix Σ form a Fourier basis when Σ is circulant, i.e., $\Sigma_{i,j} = k(i - j \bmod d)$ for some function k. Circulant covariance matrices arise from probability distributions on vectors that are invariant to rotation of the vector coordinates. The appearance of a two-dimensional Fourier basis in Figure 6 is evidence that the distribution of HOG feature vectors on natural images have (approximately) a two-dimensional rotational invariance. We can rotate the orientation bins and independently rotate the four normalizations blocks.

7 Post Processing

7.1 Bounding Box Prediction

The desired output of an object detection system is not entirely clear. The goal in the PASCAL challenge is to predict the bounding boxes of objects. In our previous work [17] we reported bounding boxes derived from root filter locations. Yet detection with one of our models localizes each part filter in addition to the root filter. Furthermore, part filters are localized with greater spatial precision than root filters. It is clear that our original approach discards potentially valuable information gained from using a multiscale deformable part model.

In the current system, we use the complete configuration of an object hypothesis, z, to predict a bounding box for the object. This is implemented using functions that map a feature vector $g(z)$, to the upper-left, (x_1, y_1), and lower-right, (x_2, y_2), corners of the bounding box. For a model with n parts, $g(z)$ is a $2n + 3$ dimensional vector containing the width of the root filter in image pixels (this provides scale information) and the location of the upper-left corner of each filter in the image.

Each object in the PASCAL training data is labeled by a bounding box. After training a model we use the output of our detector on each instance to learn four linear functions for predicting x_1, y_1, x_2 and y_2 from $g(z)$. This is done via linear least-squares regression, independently for each component of a mixture model.

Figure 7 illustrates an example of bounding prediction for a car detection. This simple method yields small but noticible improvements in performance for some categories in the PASCAL datasets (see Section 8).

7.2 Non-Maximum Suppression

Using the matching procedure from Section 3.2 we usually get multiple overlapping detections for each instance of an object. We use a greedy procedure for eliminating repeated detections via non-maximum suppression.

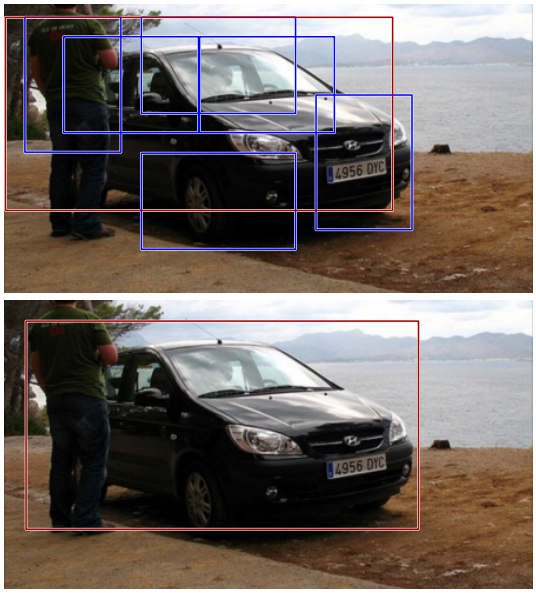


Fig. 7. A car detection and the bounding box predicted from the object configuration.

After applying the bounding box prediction method described above we have a set of detections D for a particular object category in an image. Each detection is defined by a bounding box and a score. We sort the detections in D by score, and greedily select the highest scoring ones while skipping detections with bounding boxes that are at least 50% covered by a bounding box of a previously selected detection.

7.3 Contextual Information

We have implemented a simple procedure to rescore detections using contextual information.

Let $(D_1, \ldots, D_k)$ be a set of detections obtained using k different models (for different object categories) in an image I. Each detection $(B, s) \in D_i$ is defined by a bounding box $B = (x_1, y_1, x_2, y_2)$ and a score s. We define the context of I in terms of a k-dimensional vector $c(I) = (\sigma(s_1), \ldots, \sigma(s_k))$ where s_i is the score of the highest scoring detection in D_i, and $\sigma(x) = 1/(1+\exp(-2x))$ is a logistic function for renormalizing the scores.

To rescore a detection (B, s) in an image I we build a 25-dimensional feature vector with the original score of the detection, the top-left and bottom-right bounding box coordinates, and the image context,

$$g = (\sigma(s), x_1, y_1, x_2, y_2, c(I)). \tag{30}$$

The coordinates $x_1, y_1, x_2, y_2 \in [0, 1]$ are normalized by the width and height of the image. We use a category-specific classifier to score this vector to obtain a new score for the detection. The classifier is trained to distinguish correct detections from false positives by integrating contextual information defined by g.

To get training data for the rescoring classifier we run our object detectors on images that are annotated with bounding boxes around the objects of interest (such as provided in the PASCAL datasets). Each detection returned by one of our models leads to an example g that is labeled as a true positive or false positive detection, depending on whether or not it significantly overlaps an object of the correct category.

This rescoring procedure leads to a noticible improvement in the average precision on several categories in the PASCAL datasets (see Section 8). In our experiments we used the same dataset for training models and for training the rescoring classifiers. We used SVMs with quadratic kernels for rescoring.

8 Empirical Results

We evaluated our system using the PASCAL VOC 2006, 2007 and 2008 `comp3` challenge datasets and protocol. We refer to [11]–[13] for details, but emphasize that these benchmarks are widely acknowledged as difficult testbeds for object detection.

Each dataset contains thousands of images of real-world scenes. The datasets specify ground-truth bounding boxes for several object classes. At test time, the goal is to predict the bounding boxes of all objects of a given class in an image (if any). In practice a system will output a set of bounding boxes with corresponding scores, and we can threshold these scores at different points to obtain a precision-recall curve across all images in the test set. For a particular threshold the precision is the fraction of the reported bounding boxes that are correct detections, while recall is the fraction of the objects found.

A predicted bounding box is considered correct if it overlaps more than 50% with a ground-truth bounding box, otherwise the bounding box is considered a false positive detection. Multiple detections are penalized. If a system predicts several bounding boxes that overlap with a single ground-truth bounding box, only one prediction is considered correct, the others are considered false positives. One scores a system by the average precision (AP) of its precision-recall curve across a testset.

We trained a two component model for each class in each dataset. Figure 9 shows some of the models learned on the 2007 dataset. Figure 10 shows some detections we obtain using those models. We show both high-scoring correct detections and high-scoring false positives.

In some categories our false detections are often due to confusion among classes, such as between horse and cow or between car and bus. In other categories false detections are often due to the relatively strict bounding box criteria. The two false positives shown for the person category are due to insufficient overlap with the ground-truth bounding box. In the cat category we often detect the face of a cat and report a bounding box that is too small because it does not include the rest of the cat. In fact, the top 20 highest scoring false positive bounding boxes for the cat category correspond to the face of a cat. This is an extreme case but it gives an explanation for our low AP score in this category. In many of the positive

training examples for cats only the face is visible, and we learn a model where one of the components corresponds to a cat face model, see Figure 9.

Tables 1 and 2 summarize the results of our system on the 2006 and 2007 challenge datasets. Table 3 summarizes the results on the more recent 2008 dataset, together with the systems that entered the official competition in 2008. Empty boxes indicate that a method was not tested in the corresponding object class. The entry labeled "UofCTTIUCI" is a preliminary version of the system described here. Our system obtains the best AP score in 9 out of the 20 categories and the second best in 8. Moreover, in some categories such as person we obtain a score significantly above the second best score.

For all of the experiments shown here we used the objects not marked as difficult from the `trainval` datasets to train models (we include the objects marked as truncated). Our system is fairly efficient. Using a Desktop computer it takes about 4 hours to train a model on the PASCAL 2007 `trainval` dataset and 3 hours to evaluate it on the `test` dataset. There are 4952 images in the `test` dataset, so the average running time per image is around 2 seconds. All of the experiments were done on a 2.8Ghz 8-core Intel Xeon Mac Pro computer running Mac OS X 10.5. The system makes use of the multiple-core architecture for computing filter responses in parallel, although the rest of the computation runs in a single thread.

We evaluated different aspects of our system on the longer-established 2006 dataset. Figure 8 summarizes results of different models on the person and car categories. We trained models with 1 and 2 components with and without parts. We also show the result of a 2 component model with parts and bounding box prediction. We see that the use of parts (and bounding box prediction) can significantly improve the detection accuracy. Mixture models are important in the car category but not in the person category of the 2006 dataset.

We also trained and tested a 1 component model on the INRIA Person dataset [10]. We scored the model with the PASCAL evaluation methodology (using the PASCAL development kit) over the complete test set, including images without people. We obtained an AP score of .869 in this dataset using the base system with bounding box prediction.

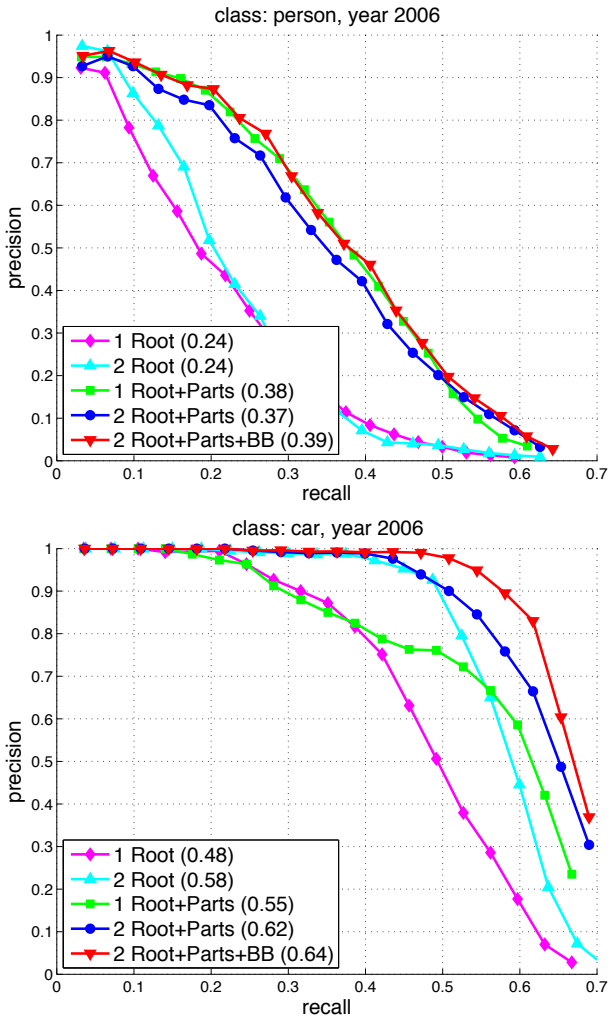


Fig. 8. Precision/Recall curves for models trained on the person and car categories of the PASCAL 2006 dataset. We show results for 1 and 2 component models with and without parts, and a 2 component model with parts and bounding box prediction. In parenthesis we show the average precision score for each model.

9 Discussion

We described an object detection system based on mixtures of multiscale deformable part models. Our system relies heavily on new methods for discriminative training of classifiers that make use of latent information. It also relies heavily on efficient methods for matching deformable models to images. The resulting system is both efficient and accurate, leading to state-of-the-art results on difficult datasets.

Our models are already capable of representing highly variable object classes, but we would like to move towards richer models. The framework described here allows for exploration of additional latent structure. For example, one can consider deeper part hierarchies (parts with parts) or mixture models with many components. In the future we would like to build grammar based models that represent objects with variable hierarchical structures. These models should allow for mixture models at the part level, and allow for reusability of parts, both in different components of an object and among different object models.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. IIS 0746569 (Pedro Felzenszwalb and Ross Girshick), IIS 0811340 (David McAllester) and IIS 0812428 (Deva Ramanan).

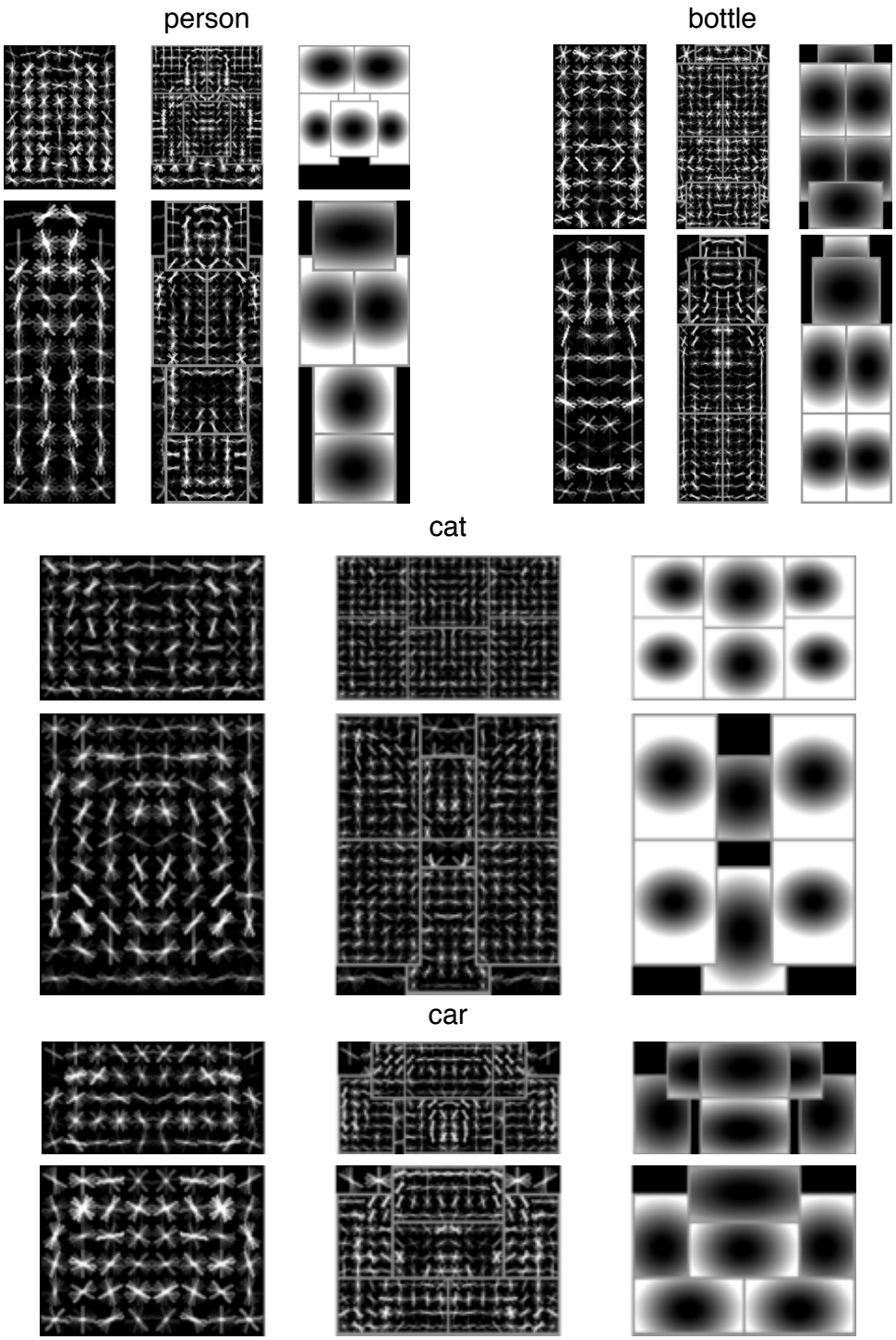


Fig. 9. Some of the models learned on the PASCAL 2007 dataset.

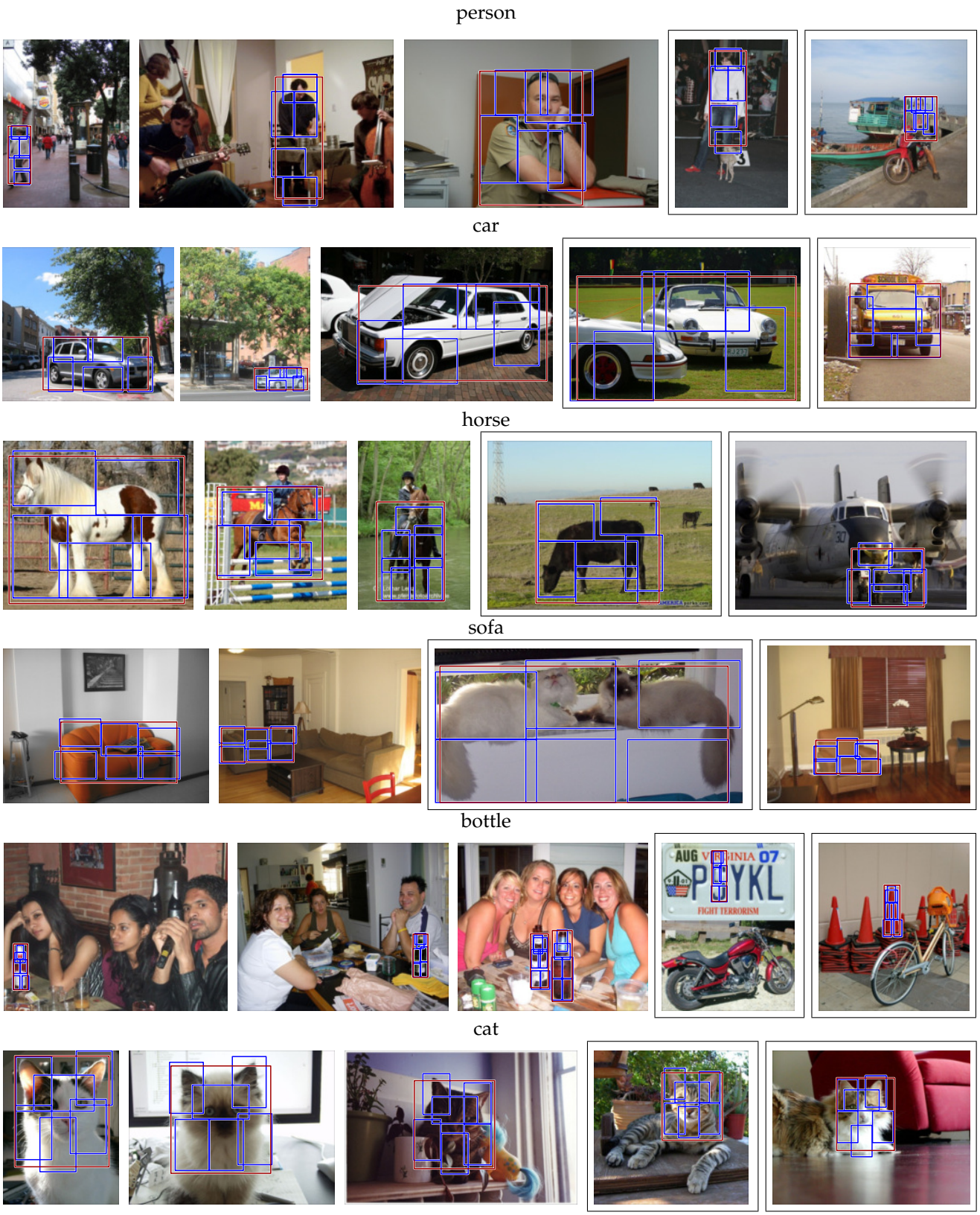


Fig. 10. Examples of high-scoring detections on the PASCAL 2007 dataset, selected from the top 20 highest scoring detections in each class. The framed images (last two in each row) illustrate false positives for each category. Many false positives (such as for person and cat) are due to the bounding box scoring criteria.

	bike	bus	car	cat	cow	dog	horse	mbik	pers	sheep
a) base	.619	.490	.615	.188	.407	.151	.392	.576	.363	.404
b) BB	.620	.493	.635	.190	.417	.153	.386	.579	.380	.402
c) context	.623	.502	.631	.236	.437	.185	.429	.625	.401	.431

TABLE 1
PASCAL VOC 2006 results. (a) average precision scores of the base system, (b) scores using bounding box prediction, (c) scores using bounding box prediction and context rescoring.

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbik	pers	plant	sheep	sofa	train	tv
a) base	.290	.546	.006	.134	.262	.394	.464	.161	.163	.165	.245	.050	.436	.378	.350	.088	.173	.216	.340	.390
b) BB	.287	.551	.006	.145	.265	.397	.502	.163	.165	.166	.245	.050	.452	.383	.362	.090	.174	.228	.341	.384
c) context	.328	.568	.025	.168	.285	.397	.516	.213	.179	.185	.259	.088	.492	.412	.368	.146	.162	.244	.392	.391

TABLE 2
PASCAL VOC 2007 results. (a) average precision scores of the base system, (b) scores using bounding box prediction, (c) scores using bounding box prediction and context rescoring.

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbik	pers	plant	sheep	sofa	train	tv
a) base	.336	.371	.066	.099	.267	.229	.319	.143	.149	.124	.119	.064	.321	.353	.407	.107	.157	.136	.228	.324
b) BB	.339	.381	.067	.099	.278	.229	.331	.146	.153	.119	.124	.066	.322	.366	.423	.108	.157	.139	.234	.328
c) context	.351	.402	.117	.114	.284	.251	.334	.188	.166	.114	.087	.078	.347	.395	.431	.117	.181	.166	.256	.347
d) rank	2	1	1	1	1	1	2	2	1	2	4	5	2	2	1	1	2	2	3	1
(UofCTTIUCI)	.326	.420	.113	.110	.282	.232	.320	.179	.146	.111	.066	.102	.327	.386	.420	.126	.161	.136	.244	.371
CASIA Det	.252	.146	.098	.105	.063	.232	.176	.090	.096	.100	.130	.055	.140	.241	.112	.030	.028	.030	.282	.146
Jena	.048	.014	.003	.002	.001	.010	.013		.001	.047	.004	.019	.003	.031	.020	.003	.004	.022	.064	.137
LEAR PC	.365	.343	.107	.114	.221	.238	.366	.166	.111	.177	.151	.090	.361	.403	.197	.115	.194	.173	.296	.340
MPI struct	.259	.080	.101	.056	.001	.113	.106	.213	.003	.045	.101	.149	.166	.200	.025	.002	.093	.123	.236	.015
Oxford	.333	.246					.291			.125			.325	.349						
XRCE Det	.264	.105	.014	.045	.000	.108	.040	.076	.020	.018	.045	.105	.118	.136	.090	.015	.061	.018	.073	.068

TABLE 3
PASCAL VOC 2008 results. Top: (a) average precision scores of the base system, (b) scores using bounding box prediction, (c) scores using bounding box prediction and context rescoring, (d) ranking of final scores relative to systems in the 2008 competition. Bottom: the systems that participated in the competition (UofCTTIUCI is a preliminary version of our system and we don't include it in the ranking).

REFERENCES

[1] Y. Amit and A. Kong, "Graphical templates for model registration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 3, pp. 225–236, 1996.

[2] Y. Amit and A. Trouve, "POP: Patchwork of parts models for object recognition," *International Journal of Computer Vision*, vol. 75, no. 2, pp. 267–282, 2007.

[3] S. Andrews, I. Tsochantaridis, and T. Hofmann, "Support vector machines for multiple-instance learning," in *Advances in Neural Information Processing Systems*, 2003.

[4] A. Bar-Hillel and D. Weinshall, "Efficient learning of relational object class models," *International Journal of Computer Vision*, vol. 77, no. 1, pp. 175–198, 2008.

[5] E. Bernstein and Y. Amit, "Part-based statistical models for object classification and detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[6] M. Burl, M. Weber, and P. Perona, "A probabilistic approach to object recognition using local photometry and global geometry," in *European Conference on Computer Vision*, 1998.

[7] T. Cootes, G. Edwards, and C. Taylor, "Active appearance models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 6, pp. 681–685, 2001.

[8] J. Coughlan, A. Yuille, C. English, and D. Snow, "Efficient deformable template detection and localization without user initialization," *Computer Vision and Image Understanding*, vol. 78, no. 3, pp. 303–319, June 2000.

[9] D. Crandall, P. Felzenszwalb, and D. Huttenlocher, "Spatial priors for part-based recognition using statistical models," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[10] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[11] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results." [Online]. Available: http://www.pascal-network.org/challenges/VOC/voc2007/

[12] ——, "The PASCAL Visual Object Classes Challenge 2008 (VOC2008) Results." [Online]. Available: http://www.pascal-network.org/challenges/VOC/voc2008/

[13] M. Everingham, A. Zisserman, C. K. I. Williams, and L. Van Gool, "The PASCAL Visual Object Classes Challenge 2006 (VOC2006) Results." [Online]. Available: http://www.pascal-network.org/challenges/VOC/voc2006/

[14] P. Felzenszwalb and D. Huttenlocher, "Distance transforms of sampled functions," Cornell University CIS, Tech. Rep. 2004-1963, 2004.

[15] ——, "Pictorial structures for object recognition," *International Journal of Computer Vision*, vol. 61, no. 1, 2005.

[16] P. Felzenszwalb and D. McAllester, "The generalized A* architecture," *Journal of Artificial Intelligence Research*, vol. 29, pp. 153–190, 2007.

[17] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[18] R. Fergus, P. Perona, and A. Zisserman, "Object class recognition by unsupervised scale-invariant learning," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[19] ——, "A sparse object category model for efficient learning and exhaustive recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[20] M. Fischler and R. Elschlager, "The representation and matching of pictorial structures," *IEEE Transactions on Computer*, vol. 22, no. 1, 1973.

[21] U. Grenander, Y. Chow, and D. Keenan, *HANDS: A Pattern-Theoretic Study of Biological Shapes*. Springer-Verlag, 1991.

[22] D. Hoiem, A. Efros, and M. Hebert, "Putting objects in perspective," *International Journal of Computer Vision*, vol. 80, no. 1, October 2008.

[23] A. Holub and P. Perona, "A discriminative framework for mod-

elling object classes," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.
[24] Y. Jin and S. Geman, "Context and hierarchy in a probabilistic image model," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.
[25] T. Joachims, "Making large-scale svm learning practical," in *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. MIT Press, 1999.
[26] Y. Ke and R. Sukthankar, "PCA-SIFT: A more distinctive representation for local image descriptors," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2004.
[27] Y. LeCun, S. Chopra, R. Hadsell, R. Marc'Aurelio, and F. Huang, "A tutorial on energy-based learning," in *Predicting Structured Data*, G. Bakir, T. Hofman, B. Schölkopf, A. Smola, and B. Taskar, Eds. MIT Press, 2006.
[28] B. Leibe, A. Leonardis, and B. Schiele, "Robust object detection with interleaved categorization and segmentation," *International Journal of Computer Vision*, vol. 77, no. 1, pp. 259–289, 2008.
[29] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, November 2004.
[30] C. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," in *IEEE International Conference on Computer Vision*, 1998.
[31] W. Plantinga and C. Dyer, "An algorithm for constructing the aspect graph," in *Foundations of Computer Science, 1985., 27th Annual Symposium on*, 1986, pp. 123–131.
[32] A. Quattoni, S. Wang, L. Morency, M. Collins, and T. Darrell, "Hidden conditional random fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 10, pp. 1848–1852, October 2007.
[33] A. Rabinovich, A. Vedaldi, C. Galleguillos, E. Wiewiora, and S. Belongie, "Objects in context," in *IEEE International Conference on Computer Vision*, 2007.
[34] D. Ramanan and C. Sminchisescu, "Training deformable models for localization," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.
[35] H. Rowley, S. Baluja, and T. Kanade, "Human face detection in visual scenes," Carnegie Mellon University, Tech. Rep. CMU-CS-95-158R, 1995.
[36] H. Schneiderman and T. Kanade, "A statistical method for 3d object detection applied to faces and cars," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2000.
[37] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Pegasos: Primal estimated sub-gradient solver for SVM," in *International Conference on Machine Learning*, 2007.
[38] K. Sung and T. Poggio, "Example-based learning for view-based human face detection," Massachussets Institute of Technology, Tech. Rep. A.I. Memo No. 1521, 1994.
[39] A. Torralba, "Contextual priming for object detection," *International Journal of Computer Vision*, vol. 53, no. 2, pp. 169–191, July 2003.
[40] P. Viola, J. Platt, and C. Zhang, "Multiple instance boosting for object detection," in *Advances in Neural Information Processing Systems*, 2005.
[41] P. Viola and M. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, May 2004.
[42] M. Weber, M. Welling, and P. Perona, "Towards automatic discovery of object categories," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2000.
[43] A. Yuille, P. Hallinan, and D. Cohen, "Feature extraction from faces using deformable templates," *International Journal of Computer Vision*, vol. 8, no. 2, pp. 99–111, 1992.
[44] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid, "Local features and kernels for classification of texture and object categories: A comprehensive study," *International Journal of Computer Vision*, vol. 73, no. 2, pp. 213–238, June 2007.
[45] S. Zhu and D. Mumford, "A stochastic grammar of images," *Foundations and Trends in Computer Graphics and Vision*, vol. 2, no. 4, pp. 259–362, 2007.