Generalized Sparselet Models for Real-Time Multiclass Object Recognition

Hyun Oh Song, Ross Girshick, Stefan Zickler, Christopher Geyer, Pedro Felzenszwalb, and Trevor Darrell

Abstract—The problem of real-time multiclass object recognition is of great practical importance in object recognition. In this paper, we describe a framework that simultaneously utilizes shared representation, reconstruction sparsity, and parallelism to enable real-time multiclass object detection with deformable part models at 5Hz on a laptop computer with almost no decrease in task performance. Our framework is trained in the standard structured output prediction formulation and is generically applicable for speeding up object recognition systems where the computational bottleneck is in multiclass, multi-convolutional inference. We experimentally demonstrate the efficiency and task performance of our method on PASCAL VOC, subset of ImageNet, Caltech101 and Caltech256 dataset.

Index Terms—Object detection, sparse coding, deformable part models, real-time vision

1 Introduction

Real-time category level recognition is a core requirement for visual competence in everyday environments. Domains of modest complexity typically have hundred to thousands of categories, and as one considers unconstrained search problems, the space of possible categories becomes practically unlimited. On top of that, modern object models [1], [2] consists of mixture of hundreds to thousands of object filters.

As the number of categories grows, individual models are increasingly likely to become redundant. In the case of part-based models this redundancy can be exploited by constructing models with shared parts. In this regard, shared intermediate representations are highly appealing due to their potential for gains in computational and statistical efficiency. These representations appear under a variety of guises, such as steerable filter banks [3], low-rank approximations for collaborative filtering [4], and shared part models for object detection [5], [6], [7].

Recently, sparselets [8], [9] were introduced as a new shared intermediate representation for multiclass object detection with deformable part models (DPMs) [1]. In this application, each sparselet can be thought of as a small, generic part (e.g., a corner or edge) that is shared between all object categories. The parts of a DPM, for any class, are then constructed by tiling sparse linear combinations ("activations") of the sparselet mini-parts. With this representation, sparselets reconstruct approximate part responses using a sparse matrix-vector product instead of exhaustive convolutions.

In contrast to standard applications of sparse coding, where features are encoded as sparse combinations of over-complete dictionary elements, sparselet models learn a dictionary of *model parameters*, and the models themselves are encoded as sparse combinations of dictionary elements. This leads to a compression of the models that can be exploited to speed-up computation.

The computational efficiency gains of this approach were demonstrated in a GPU sparselets implementation of DPM detection that outperformed a baseline GPU implementation by a factor of 3 to 5×, and outperformed the CPU version of the cascade algorithm in [10] by a factor of 15×, with almost no loss in detection average precision. The sparsity level used in this construction naturally trades off a decrease in detection accuracy for greater speed. However, the reconstructive method for learning activations proposed in [8] is brittle, and pushing slightly beyond these speedup factors leads to a substantial loss in detection accuracy.

This paper also helps unify sparselets with the steerable part models of [11]. The fundamental differences between the two methods lies in how they accelerate inference and how they are trained. Steerable part models use a small part dictionary with dense linear combinations and discriminative training, whereas sparselets use a larger dictionary with sparse linear combination, and a reconstructive error training paradigm. With regard to dictionary size and linear combination density, the two approaches can be viewed as operating at different points within the same algorithm design space. The remaining difference, then, lies in the training method. This paper unifies the two approaches by showing how to train sparselet activations discriminatively, or alternately, how to train steering coefficients sparsely.

2 Related Work

Our work is related to three strands of active research: (1) part sharing with compositional models [5], [6], [7], [12],

- *H.O. Song, R. Girshick and T. Darrell are with the Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA 94720. E-mail: {song, rbg, trevor}@eecs.berkeley.edu.*
- *S. Zickler is with iRobot, Bedford, MA 01730. E-mail: szickler@irobot.com.*
- *C. Geyer is with Berkshire Grey Inc, Boston, MA. E-mail: cmgeyer@gmail.com.*
- *P. Felzenszwalb is with the Department of Engineering and Computer Science, Brown University, Providence, RI 02912. E-mail: pff@brown.edu.*

Manuscript received 14 Mar. 2013; revised 13 Mar. 2014; accepted 1 Apr. 2014. Date of publication 3 Sept. 2014; date of current version 4 Apr. 2015. Recommended for acceptance by D. Ramanan.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPAMI.2014.2353631

[13], (2) sparse coding and dictionary learning [14], [15], [16], and (3) modeling and learning with low-rank approximations [3], [17], [18]. None of these methods, however, simultaneously exploit shared interclass information *and* discriminative sparsity learning to speed up inference while maintaining task performance.

A preliminary version of our system was described in [8], [9]. First we introduce the notion of generalized sparselets in structured output prediction problems [19], [20] and analyze the computational gains in efficiency. Also, we formulate a discriminative sparselet activation training framework and several regularization schemes that lead to improved sparsity and task performance. We experimentally demonstrate that the proposed sparselet activation learning algorithm substantially outperforms reconstructive sparselets and generalizes to previously unseen object categories.

The paper is structured as follows. In Section 3, we start with a brief overview of sparselets [8] and formulate structured output prediction with generalized sparselets [9]. In Section 4, we describe how discriminative sparselet activation training fits into the framework and discuss several regularization methods for sparse activation learning. In Section 5, we discuss important applications of the proposed approach to multiclass object detection with mixtures of deformable part models [1] and to multiclass image classification. Before we conclude in Section 7, we provide experimental results on sensitivity of the sparselet dictionary learned form random subset of object classes, the effect of different sparselet block sizes, multiclass object detection, multiclass image classification, and wall clock run time experiments with and without GPU in Section 6.

3 Sparselets

In general, convolution of a feature pyramid with thousands of object model filters becomes the major computational bottleneck in multiclass object detection tasks. The *sparselet model* tackles this problem by learning a dictionary of "universal" object models that allows filter convolutions to be computed as sparse linear combinations of sparselet convolutions. The sparselet dictionary size is independent of the number of classes, and as the number classes increases the speedup offered by the method approaches the ratio between the number of classes and the reconstruction sparsity.

3.1 Sparse Reconstruction of Object Models

Formally, a *sparselet model* is defined by a dictionary $\mathbf{S} = [\mathbf{s}_1, \dots, \mathbf{s}_d]$ in $\mathbb{R}^{m \times d}$, where each column $\mathbf{s}_i$ in $\mathbb{R}^m$ is called a *sparselet*. We formulate the following optimization problem to compute a sparselet model that reconstructs a matrix of linear object filters $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_K] \in \mathbb{R}^{m \times K}$ collected from a set of trained models

$$\begin{aligned} &\min_{\alpha_{ij}, \mathbf{s}_j} \sum_{i=1}^{K} ||\mathbf{w}_i - \sum_{j=1}^{d} \alpha_{ij} \mathbf{s}_j||_2^2 \\ &\text{subject to} \quad ||\boldsymbol{\alpha_i}||_0 \leq \lambda_0 \quad \forall i = 1, \dots, K \\ &\qquad\qquad\quad ||\mathbf{s}_j||_2^2 \leq 1 \quad \forall j = 1, \dots, d. \end{aligned} \tag{1}$$

Although the above optimization is NP-hard, greedy algorithms such as orthogonal matching pursuit algorithm (OMP) [21] can be used to efficiently compute an approximate solution. OMP iteratively estimates the optimal matching projections of the input signal onto the dictionary $\mathbf{S}$. The above optimization problem is convex with respect to $\mathbf{S}$ if $\boldsymbol{\alpha}_i$ is fixed, and so we can optimize the objective in a coordinate descent fashion by iterating between updating $\boldsymbol{\alpha}_i$ while fixing $\mathbf{S}$ and vice versa. For our experiments we use the online dictionary learning algorithm from [15].

3.2 Precomputation and Efficient Reconstruction

We can precompute convolutions for all sparselets, and by linearity of convolution we can then use the activation vectors estimated for a target object detector to approximate the convolution response we would have obtained from convolution with the original filters. Denoting the feature pyramid of an image as Ψ, we have

$$\Psi * \mathbf{w}_i \approx \Psi * \sum_j \alpha_{ij} \mathbf{s}_i = \sum_j \alpha_{ij} (\Psi * \mathbf{s}_i). \tag{2}$$

Concretely, we can recover individual part filter responses via sparse matrix multiplication (or lookups) with the activation vector replacing the heavy convolution operation as shown in Eq. (3):

$$\begin{bmatrix} \text{------}\Psi * \mathbf{w}_1\text{------} \\ \text{------}\Psi * \mathbf{w}_2\text{------} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \text{-----}\Psi * \mathbf{w}_K\text{-----} \end{bmatrix} \approx \begin{bmatrix} \text{-----}\boldsymbol{\alpha}_1\text{-----} \\ \text{-----}\boldsymbol{\alpha}_2\text{-----} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \text{-----}\boldsymbol{\alpha}_K\text{-----} \end{bmatrix} \begin{bmatrix} \text{------}\Psi * \mathbf{s}_1\text{------} \\ \text{------}\Psi * \mathbf{s}_2\text{------} \\ \vdots \\ \text{------}\Psi * \mathbf{s}_d\text{------} \end{bmatrix}, \tag{3}$$

where the first matrix on the right hand side is the matrix of sparse activation vectors and the second matrix is a matrix of all sparselet responses. The sparselet responses $\Psi * \mathbf{s}_i$ are independent of any filter, and thus their cost can be amortized over all filters from all object models. Fig. 1 shows the overview of the approach. In the remainder of this section we present a generalization of this technique. First, we illustrate how to generalize sparselets for simple multiclass linear classifiers, and then for any linear structured output prediction model.

3.3 Generalized Sparselets for Structured Output Prediction

Consider a set of K linear classifiers parameterized by the weight vectors $\mathbf{w}_1, \dots, \mathbf{w}_K$ each in $\mathbb{R}^n$. An input feature vector $\mathbf{x} \in \mathbb{R}^n$ is assigned to a class $f_{\mathbf{w}}(\mathbf{x}) \in \{1, \dots, K\}$ according to the rule

$$f_{\mathbf{w}}(\mathbf{x}) = \underset{k \in \{1, \dots, K\}}{\operatorname{argmax}} \, \mathbf{w}_k^{\mathsf{T}} \mathbf{x}. \tag{4}$$

Our objective is to reduce the computational cost of computing Eq. (4).

We begin by partitioning each parameter vector $\mathbf{w}_k$ into several m-dimensional *blocks*. A block is a subvector of parameters chosen so that the set of all blocks admits a sparse

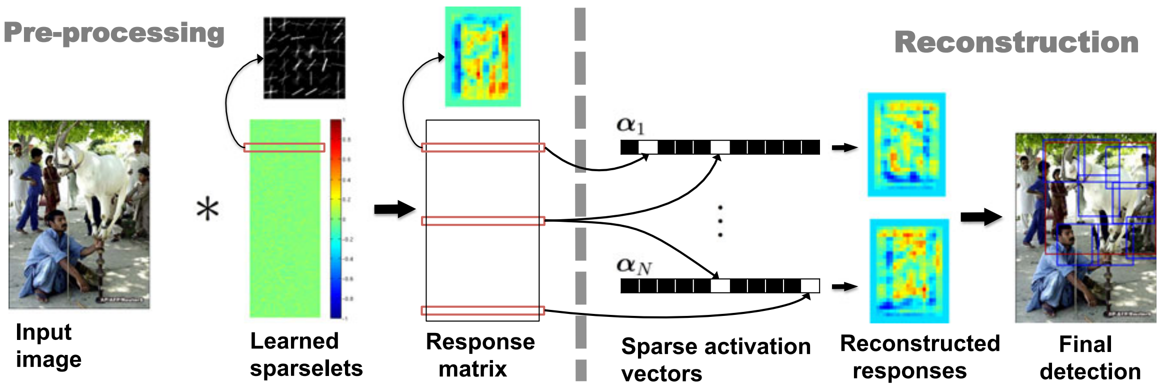

Fig. 1. Overview diagram of object detection with sparselets. Once we evaluate the image with learned sparselets, the reconstruction phase can be done via efficient sparse matrix vector multiplications.

representation over $\mathbf{S}$. Concretely, in the examples that follow blocks will be chosen to be fragments of part filters in a deformable part model (see Fig. 3), or simply contiguous subvectors of the parameters in a bag-of-visual-words classifier. For clarity, we will assume that $n = pm$ for some positive integer p. We can rewrite each linear classifier in terms of its blocks, $\mathbf{b}_{ki}$ in $\mathbb{R}^m$, such that $\mathbf{w}_k = (\mathbf{b}_{k1}^\mathsf{T}, \ldots, \mathbf{b}_{kp}^\mathsf{T})^\mathsf{T}$. Similarly, we can partition an input feature vector into m-dimensional subvectors, $\mathbf{c}_i$ in $\mathbb{R}^m$, such that $\mathbf{x} = (\mathbf{c}_1^\mathsf{T}, \ldots, \mathbf{c}_p^\mathsf{T})^\mathsf{T}$.

Given a sparselet model $\mathbf{S}$, we can approximate any vector $\mathbf{b} \in \mathbb{R}^m$ as a sparse linear combination of the sparselets in $\mathbf{S}$

$$\mathbf{b} \approx \mathbf{S}\alpha = \sum_{\substack{i=1,\\ \alpha_i \neq 0}}^{d} \alpha_i \mathbf{s}_i, \tag{5}$$

where $\alpha = (\alpha_1, \ldots, \alpha_d)^\mathsf{T} \in \mathbb{R}^d$ is a *sparselet activation vector* for $\mathbf{b}$. The quality of the approximation depends on the fixed dictionary and the chosen activation vector. Now, the dot product in Eq. (4) can be approximated as

$$\begin{aligned}\mathbf{w}_k^\mathsf{T}\mathbf{x} &= \left(\mathbf{b}_{k1}^\mathsf{T}, \ldots, \mathbf{b}_{kp}^\mathsf{T}\right)\left(\mathbf{c}_1^\mathsf{T}, \ldots, \mathbf{c}_p^\mathsf{T}\right)^\mathsf{T} \\ &= \sum_{i=1}^{p} \mathbf{b}_{ki}^\mathsf{T}\mathbf{c}_i \approx \sum_{i=1}^{p} (\mathbf{S}\alpha_{ki})^\mathsf{T}\mathbf{c}_i = \sum_{i=1}^{p} \alpha_{ki}^\mathsf{T}(\mathbf{S}^\mathsf{T}\mathbf{c}_i).\end{aligned} \tag{6}$$

We note two important properties of Eq. (6): (1) the *sparselet responses* $\mathbf{S}^\mathsf{T}\mathbf{c}_i$ are independent of any particular classifier, and (2) the subsequent product with α_{ki} can be computed efficiently by accessing only the nonzero elements of α_{ki}. In the following, let λ_0 be the average number of nonzero elements in each α_{ki}.

Multiclass classification is a special case of structured output prediction. To complete the description of generalized sparselets for structured output prediction, consider the linear discriminant function

$$f_{\mathbf{w}}(\mathbf{x}) = \underset{\mathbf{y} \in \mathcal{Y}}{\operatorname{argmax}}\, \mathbf{w}^\mathsf{T}\Phi(\mathbf{x}, \mathbf{y}), \tag{7}$$

where the input $\mathbf{x}$ comes from an arbitrary input space $\mathcal{X}$, and $f_{\mathbf{w}}$ outputs an element from the label space $\mathcal{Y}$. In the following section, we give concrete examples of how the weight vector $\mathbf{w}$ is partitioned into *blocks*and analyze the computational cost for three scenarios: multiclass classification, multiclass convolutional classifiers and part based models.

3.4 Computational Cost Analysis

We can analyze generalized sparselets for multiclass classification by looking at the cost of computing $\mathbf{b}_{ki}^\mathsf{T}\mathbf{c}_i$ for a single block i and for all classes k. The original classifiers require Km additions and multiplications. The generalized sparselet approach has a shared cost of dm operations for computing the sparselet responses, $\mathbf{r}_i = \mathbf{S}^\mathsf{T}\mathbf{c}_i$, and a cost of $K\lambda_0$ operations for computing $\alpha_{ki}^\mathsf{T}\mathbf{r}_i$ for all classes. The overall speedup is thus $Km/(dm + K\lambda_0)$. To make this value large, the dictionary size d should be much smaller than the number of classes K, and the average number of nonzero coefficients in the activation vectors should be much less than the sparselet size m. As the number of classes becomes large, the cost of computing sparselet responses becomes fully amortized which leads to a maximum theoretical speedup of m/λ_0 [8]. This emphasizes the importance of a sparse representation, in contrast, for example, to the dense steering coefficients in [11]. This analysis shows that generalized sparselets are most applicable to multiclass problems with a large number of classes. This is a regime of growing interest, especially in computer vision as exemplified by new datasets such as ImageNet [22], which includes more than 10,000 categories [23]. In Section 6.5 we show results on the Caltech-{101, 256} [24], [25] datasets demonstrating that even with only one or two hundred classes generalized sparselets can accelerate simple linear classifiers.

Analysing the speedup for general structured prediction problems requires reasoning about the feature map $\Phi(x, y)$ and the inference algorithm used to compute the $argmax$ in Eq. (7). Let $\mathcal{A}$ be an algorithm such that $\mathcal{A}(\mathbf{w}, x)$ computes $f_{\mathbf{w}}(x)$, i.e. it solves the argmax in Eq. (7). To analyze the speedup, we will build a bipartite graph $\mathcal{G} = (\mathcal{B} \cup \mathcal{C}, \mathcal{E})$ that encodes certain computations performed by $\mathcal{A}$. The graph depends on $\mathcal{A}$'s inputs $\mathbf{w}$ and x, but to lighten notation we will omit this dependence. As before, we assume $\mathbf{w}$ is partitioned into a set of blocks $\{\mathbf{b}_i\}$ in $\mathbb{R}^m$ such that $\mathbf{w} = (\mathbf{b}_1^\mathsf{T}, \ldots, \mathbf{b}_p^\mathsf{T})^\mathsf{T}$, each of which will be approximated with sparselets.

Each node in $\mathcal{G}$ corresponds to a vector in $\mathbb{R}^m$ (either a block of features or a block of model parameters). With a slight abuse of notation we will label each node with the vector that it is in correspondence with. Similarly, we will label the edges with a pair of vectors (i.e., nodes), each in

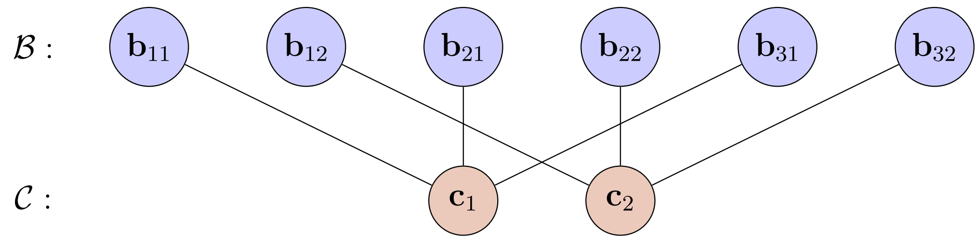

Fig. 2. Computation graph for a multiclass problem with $K = 3$. Let the sparselet size be m and the number of blocks be $p = 2$. We define $\mathbf{w} = (\mathbf{w}_1^\mathsf{T}, \mathbf{w}_2^\mathsf{T}, \mathbf{w}_3^\mathsf{T})^\mathsf{T}$ in $\mathbb{R}^{Kpm}$. Each per-class classifier $\mathbf{w}_k$ in $\mathbb{R}^{pm}$ is partitioned into p blocks such that $\mathbf{w}_k = (\mathbf{b}_{k1}^\mathsf{T}, \mathbf{b}_{k2}^\mathsf{T})^\mathsf{T}$. An input vector $\mathbf{x}$ in $\mathbb{R}^{pm}$ is partitioned into subvectors such that $\mathbf{x} = (\mathbf{c}_1^\mathsf{T}, \mathbf{c}_2^\mathsf{T})^\mathsf{T}$. The feature map $\Phi(\mathbf{x}, k)$ in $\mathbb{R}^{Kpm}$ is defined as: $\Phi(\mathbf{x}, 1) = (\mathbf{x}^\mathsf{T}, 0, \dots, 0)^\mathsf{T}$; $\Phi(\mathbf{x}, 2) = (0, \dots, 0, \mathbf{x}^\mathsf{T}, 0, \dots, 0)^\mathsf{T}$; $\Phi(\mathbf{x}, 3) = (0, \dots, 0, \mathbf{x}^\mathsf{T})^\mathsf{T}$. The edges in the graph encode the dot products computed while solving $\operatorname{argmax}_{k\in\{1,2,3\}} \mathbf{w}^\mathsf{T}\Phi(\mathbf{x}, k)$.

$\mathbb{R}^m$. We define the first set of disconnected nodes in $\mathcal{G}$ to be the set of all blocks in $\mathbf{w}$: $\mathcal{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_p\}$. We will define the second set of disconnected nodes, $\mathcal{C}$, next.

Any algorithm that computes Eq. (7) performs some number of computations of the form $\mathbf{b}^\mathsf{T}\mathbf{c}$, for a block $\mathbf{b} \in \mathcal{B}$ and some vector $\mathbf{c} \in \mathbb{R}^m$. The vectors $\mathbf{c}$ appearing in these computations are most likely subvectors of $\Phi(x, y)$ arising from various values of y. The graph $\mathcal{G}$ encodes all unique computations of this form. Conceptually, $\mathcal{C}$ could be constructed by running algorithm $\mathcal{A}$ and adding each *unique* vector $\mathbf{c}$ that appears in a computation of the form $\mathbf{b}^\mathsf{T}\mathbf{c}$ to $\mathcal{C}$. The edge set $\mathcal{E}$ is constructed by connecting a node $\mathbf{b} \in \mathcal{B}$ to a node in $\mathbf{c} \in \mathcal{C}$ if and only if $\mathcal{A}$ performs the computation $\mathbf{b}^\mathsf{T}\mathbf{c}$. For a specific algorithm $\mathcal{A}$, we can construct $\mathcal{G}$ analytically (instead of by running the algorithm as just described). An example graph for a multiclass classification problem is given in Fig. 2.

The edges in $\mathcal{G}$ encode exactly all of the computations of the form $\mathbf{b}^\mathsf{T}\mathbf{c}$ and therefore we can use it to analyze the computational costs of $\mathcal{A}$ with and without generalized sparselets.

Obviously, not all of the computation performed by $\mathcal{A}$ are of the form captured by the graph. For example, when generalized distance transforms are used by $\mathcal{A}$ to solve in the computation of Eq. (7) for deformable part models, the cost of computing the distance transforms is outside of the scope of $\mathcal{G}$ (and outside the application of sparselets). We let the quantity $T(\mathbf{w}, x)$ account for all computational costs not represented in $\mathcal{G}$.

We are now ready to write the number of operations performed by $\mathcal{A}(\mathbf{w}, x)$. First, without sparselets we have

$$T_{\text{Original}}(\mathbf{w}, x) = T(\mathbf{w}, x) + m \sum_{\mathbf{c}\in\mathcal{C}} \deg(\mathbf{c}), \tag{8}$$

where $\deg(\mathbf{v})$ is the degree of a node $\mathbf{v}$ in $\mathcal{G}$. The second term in Eq. (8) accounts for the m additions and multiplications that are performed when computing $\mathbf{b}^\mathsf{T}\mathbf{c}$ for a pair of nodes $(\mathbf{b}, \mathbf{c}) \in \mathcal{E}$.

When sparselets are applied, the cost becomes

$$T_{\text{Sparselets}}(\mathbf{w}, x) = T(\mathbf{w}, x) + dm|\mathcal{C}| + \lambda_0 \sum_{\mathbf{c}\in\mathcal{C}} \deg(\mathbf{c}), \tag{9}$$

The second term in Eq. (9) accounts for the cost of precomputing the sparselet responses, $\mathbf{r} = \mathbf{S}^\mathsf{T}\mathbf{c}$ (cost dm), for each node in $\mathcal{C}$. The third term accounts for the sparse dot product $\alpha(\mathbf{b})^\mathsf{T}\mathbf{r}$ (cost λ_0) computed for each pair $(\mathbf{b}, \mathbf{c}) \in \mathcal{E}$, where $\alpha(\mathbf{b})$ is the sparselet activation vector for $\mathbf{b}$.

The speedup is the ratio $T_{\text{Original}}/T_{\text{sparselets}}$

$$\frac{T(\mathbf{w}, x) + m\sum_{i=1}^{|\mathcal{C}|} \deg(\mathbf{c}_j)}{T(\mathbf{w}, x) + dm|\mathcal{C}| + \lambda_0 \sum_{i=1}^{|\mathcal{C}|} \deg(\mathbf{c}_j)}. \tag{10}$$

In all of the examples we consider in this paper, the degree of each node in $\mathcal{C}$ is a single constant: $\deg(\mathbf{c}) = Q$ for all $\mathbf{c} \in \mathcal{C}$. In this case, the speedup simplifies to the following:

$$\frac{T(\mathbf{w}, x) + Q|\mathcal{C}|m}{T(\mathbf{w}, x) + dm|\mathcal{C}| + Q|\mathcal{C}|\lambda_0}. \tag{11}$$

This analysis shows that sparselets are not applicable to inference algorithms where $T(\mathbf{w}, x)$ is large. However, in many interesting cases, such as those discussed below, $T(\mathbf{w}, x)$ is small. If we narrow our scope to only consider the speedup restricted to the operations of $\mathcal{A}$ affected by sparselets, we can ignore the $T(\mathbf{w}, x)$ terms and note that the $|\mathcal{C}|$ factors cancel

$$\frac{Qm}{dm + Q\lambda_0}. \tag{12}$$

This narrowing is justified in the multiclass classification case (with K classes) where the cost $T(\mathbf{w}, x)$ amounts to computing the maximum value of K numbers, which is negligible compared to the other terms. We also observe that $Q = K$, yielding the following speedup:

$$\frac{Km}{dm + K\lambda_0}. \tag{13}$$

The computation graph for a simple multiclass example with $K = 3$ is given in Fig. 2. For more intuition, we consider two examples below.

Multiclass convolutional classifiers. Consider the multiclass setting and let $\mathbf{w} = (\mathbf{w}_1^\mathsf{T}, \dots, \mathbf{w}_K^\mathsf{T})^\mathsf{T}$ in $\mathbb{R}^{Kn}$. As before, each $\mathbf{w}_k$ is partitioned into p blocks. But now, instead of an n-dimensional input feature vector, consider larger input vectors $\mathbf{x} \in \mathbb{R}^q$, $q \gg n$, and the feature map $\Phi(\mathbf{x}, (k, y)) = (0, \dots, 0, \mathbf{x}_{y:n}^\mathsf{T}, 0, \dots, 0)^\mathsf{T}$. We write $\mathbf{x}_{y:n}$ to denote the length-n subvector of $\mathbf{x}$ starting at position y. This subvector is placed into the kth "slot" of Φ (corresponding to the slot for $\mathbf{w}_k$ in $\mathbf{w}$). The label space $\mathcal{Y}$ consists of all valid (class, position) pairs (k, y). This setup is equivalent to the problem of searching for the subvector of $\mathbf{x}$ that has maximum correlation with a weight vector in $\{\mathbf{w}_k\}$. A concrete example of this is multiclass object detection with Dalal and Triggs style scanning window detectors [26]. In contrast to the non-convolutional multiclass setting, now each block of $\mathbf{w}$ must be multiplied with each subvector of $\mathbf{x}$ while scanning for the maximum response (imagine "sliding" each $\mathbf{w}_k$ over $\mathbf{x}$ while computing a dot product at each position), and thus $Q = Kp$.

Part-Based Models. Another common situation that leads to a large Q value is when $\mathbf{w}$ parameterizes a set of "parts" and $f_{\mathbf{w}}(x)$ computes the optimal assignment of the parts to locations y in the input x. For example, a location y might be a position in a sentence or an image. In this problem setting, there is a (typically very large) pool of feature vectors, where each vector in the pool describes one location in x. The feature map $\Phi(x, y)$ acts on a label y by installing the selected subset of local feature vectors into the appropriate slots of Φ. These problems typically also involve pairwise interactions between the labels assigned to some pairs of parts. When these interactions form a tree, dynamic programming can be used to efficiently compute the optimal label assignments. In the dynamic programming algorithm, the dot product between each part model and each local feature vector must be evaluated. As a concrete example, consider the deformable part models of [1]. For this model, the dynamic programming algorithm implicitly generates the large set of local feature vectors through the convolution of each part with a histogram of oriented gradients (HOG) feature image [1], [26]. Given object detectors for K classes, each with N parts, each of which is partitioned into p blocks, this model and algorithm result in $Q = KNp$. The part-based structure of this problem increases sparselet response reuse by a factor of N.

4 Discriminative Activation of Generalized Sparselets

Throughout the rest of this paper we consider linear models defined by parameter vectors that are partitioned into K slots: $\mathbf{w} = (\mathbf{w}_1^\mathsf{T}, \ldots, \mathbf{w}_K^\mathsf{T})^\mathsf{T}$. In the multiclass setting, slots correspond to the individual classifiers. More generally, slots might be structures like the filters in a deformable part model. Generalized sparselets may be applied to any subset of the slots. For a slot $\mathbf{w}_k$ to which sparselets are applied, it is further partitioned into p_k blocks: $\mathbf{w}_k = (\mathbf{b}_{k1}^\mathsf{T}, \ldots, \mathbf{b}_{kp_k}^\mathsf{T})^\mathsf{T}$. The $\{\mathbf{w}_k\}$ may have different dimensions, as long as each is a multiple of the sparselet dimension m.

In [8], the task of learning the sparselet model $\mathbf{S}$ from a training set of parameter blocks $\{\mathbf{b}_{ki}\}$ was naturally posed as a sparse coding dictionary learning problem [14], [15]. The objective was to find a dictionary $\mathbf{S}$ and activation vectors $\{\alpha_{ki}\}$ that minimize reconstruction error, subject to an ℓ_0-pseudo-norm sparsity constraint on each activation vector. Then, given the learned dictionary $\mathbf{S}$, the activation vectors for a model $\mathbf{w}$ (either previously unseen or from the training set) were learned by minimizing reconstruction error, subject to the same sparsity constraint.

The experimental results in [8] show that task performance (average precision for object detection) quickly degrades to undesirable levels as the activation vectors are made increasingly sparse. This result is intuitive given the reconstructive activation vector learning method used in [8]. When reconstruction error is low (i.e. low sparsity), the original decision boundary of the model is roughly preserved. However, as sparsity increases, and the reconstruction error becomes larger, the decision boundary of the reconstructed model changes in an uncontrolled way and may no longer be discriminative for the target task.

Our solution is to replace reconstruction error with a discriminative objective. To do this (assuming a fixed dictionary), we propose to rewrite the original optimization problem used for training the linear model in terms of sparselet responses, which now act as training features, and the activation vectors, which now act as the model parameters. To achieve sparsity, we augment this new objective function with a sparsity inducing regularizer. As we show below, the somewhat obvious choice of ℓ_1 regularization leads to unsatisfactory results, motivating the development of an alternative approach.

4.1 Learning Discriminative Activation Vectors

Here we consider learning the activation vectors for a predictor $\mathbf{w}$ in the structural SVM (SSVM) framework [19], [20]. The SSVM training equation is

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \frac{1}{M}\sum_{i=1}^{M} \max_{\hat{y}\in\mathcal{Y}} \big(\mathbf{w}^\mathsf{T}\Phi(x_i, \hat{y}) + \Delta(y_i, \hat{y})\big) - \mathbf{w}^\mathsf{T}\Phi(x_i, y_i), \tag{14}$$

where $\Delta(y, y')$ is a loss function. Given a fixed sparselet model $\mathbf{S}$, we can rewrite Eq. (14) in terms of the activation vector parameters and sparselet responses. For clarity, assume the slots of $\mathbf{w}$ have been arranged so that slots 1 through s are represented with sparselets, and slots $s+1$ through K are not.[1] For each slot $\mathbf{w}_k = (\mathbf{b}_{k1}^\mathsf{T}, \ldots, \mathbf{b}_{kp_k}^\mathsf{T})^\mathsf{T}$ that is represented by sparselets, we define a corresponding activation parameter vector $\alpha_k = (\alpha_{k1}^\mathsf{T}, \ldots, \alpha_{kp_k}^\mathsf{T})^\mathsf{T} \in \mathbb{R}^{p_k d}$. Let $\alpha = (\alpha_1^\mathsf{T}, \ldots, \alpha_s^\mathsf{T})^\mathsf{T}$ and $\tilde{\mathbf{w}} = (\mathbf{w}_{s+1}^\mathsf{T}, \ldots, \mathbf{w}_K^\mathsf{T})^\mathsf{T}$, and define the new model parameter vector $\beta = (\alpha^\mathsf{T}, \tilde{\mathbf{w}}^\mathsf{T})^\mathsf{T}$.

We transform the feature vector in a similar manner. For a feature vector slot $\Phi_k(x, y) = (\mathbf{c}_1^\mathsf{T}, \ldots, \mathbf{c}_{p_k}^\mathsf{T})^\mathsf{T}$ that will be represented by sparselets, we transform the features into sparselet responses: $\tilde{\Phi}_k(x, y) = (\mathbf{c}_1^\mathsf{T}\mathbf{S}, \ldots, \mathbf{c}_{p_k}^\mathsf{T}\mathbf{S})^\mathsf{T} \in \mathbb{R}^{p_k d}$. The fully transformed feature vector is $\tilde{\Phi}(x, y) = (\tilde{\Phi}_1^\mathsf{T}(x, y), \ldots, \tilde{\Phi}_s^\mathsf{T}(x, y), \Phi_{s+1}^\mathsf{T}(x, y), \ldots, \Phi_K^\mathsf{T}(x, y))^\mathsf{T}$. The resulting objective is

$$\beta^* = \operatorname*{argmin}_{\beta} R(\alpha) + \frac{\lambda}{2}\|\tilde{\mathbf{w}}\|_2^2 + \frac{1}{M}\sum_{i=1}^{M} \max_{\hat{y}\in\mathcal{Y}} \Big(\beta^\mathsf{T}\tilde{\Phi}(x_i, \hat{y}) + \Delta(y_i, \hat{y})\Big) - \beta^\mathsf{T}\tilde{\Phi}(x_i, y_i), \tag{15}$$

where $R(\alpha)$ is a regularizer applied to the activation vectors.

1. This flexibility lets us leave slots where sparselets don't make sense unchanged, e.g. a bias parameter slot.

4.2 Inducing Sparsity

We consider three sparsity inducing regularizers R.

I. **Lasso penalty** [27]
$R_{\text{Lasso}}(\alpha) = \lambda_1 \|\alpha\|_1$

II. **Elastic net penalty** [28]
$R_{\text{EN}}(\alpha) = \lambda_1 \|\alpha\|_1 + \lambda_2 \|\alpha\|_2^2$

III. **Combined ℓ_0 and ℓ_2 penalty**
$R_{0,2}(\alpha) = \lambda_2 \|\alpha\|_2^2$ subject to $\|\alpha\|_0 \le \lambda_0$

The first two regularizers lead to convex optimization problems, however the third does not. We consider two alternative methods for approximately minimizing Eq. (15) when $R(\alpha) = R_{0,2}(\alpha)$. Both of these methods employ a two step process. In the first step, a subset of the activation coefficients is selected to satisfy the constraint $\|\alpha\|_0 \le \lambda_0$. In the second step, the selection of nonzero variables is fixed (thus satisfying the sparsity constraint) and the resulting convex optimization problem is solved. We consider the following variable selection strategies.

III-A. *Overshoot, rank, and threshold (ORT).* In this method, we first apply either R_{Lasso} or R_{EN} with λ_1 set to *overshoot* the target number of nonzero variables λ_0. We then rank the nonzero activation coefficients by their magnitudes and select the λ_0 variables with the largest magnitudes. Each variable in the selected variable set's complement is thresholded to zero.

III-B. *Orthogonal matching pursuit.* In this method, we select the nonzero variables by minimizing the reconstruction error between parameter blocks and their sparse coding approximation subject to the constraint $\|\alpha\|_0 \le \lambda_0$. In practice, we use orthogonal matching pursuit [21] as implemented in SPAMS software package [15]. This produces the same initial set of activation vectors as the baseline method [8]. However, we then learn the selected variables discriminatively according to Eq. (15).

5 Application of Generalized Sparselets

We first focus on the application of our novel sparselet activation vector learning approach to object detection with mixtures of deformable part models [1] in order to facilitate direct comparison with the results in [8]. In brief, the deformable part model from [1] is specified by a root filter that models the global appearance of an object class and a set of N part filters that capture local appearance. The part filters are attached to the root filter by flexible "springs" that allow the model to match the image with a deformed arrangement of parts. In practice, several DPMs are combined into one mixture model to better represent more extreme variation in object class appearance.

A DPM is matched to an image by maximizing a score function over latent variables z. Let $z = (c, \rho_0, \ldots, \rho_N)$ specify a mixture component $c \in \{1, \ldots, C\}$, root filter location ρ_0, and part filter locations $\rho_1, \ldots, \rho_N$ for a model with C components and N part filters. The score function can be written as

$$\begin{aligned}\text{score}(x, z) = w_c + \sum_{i=0}^{N} \mathbf{w}_{ci}^{\mathsf{T}} \psi_{ci}(x, \rho_i) \\ + \sum_{i=1}^{N} \mathbf{d}_{ci}^{\mathsf{T}} \delta_{ci}(\rho_0, \rho_i) = \mathbf{w}^{\mathsf{T}} \Phi(x, z),\end{aligned} \tag{16}$$

where $\mathbf{w}_{ci}$ are the weights in filter i of component c, $\mathbf{d}_{ci}$ are the quadratic deformation parameters specifying the stiffness of the spring connecting the root filter and part filter i of component c, and w_c is a score bias. The feature functions $\psi_{ci}(x, \rho_i)$ and $\delta_{ci}(\rho_0, \rho_i)$ are local image features (HOG) and deformation features, respectively. The score can be written as a single dot production between

$$\begin{aligned}\mathbf{w} = (w_1, \ldots, w_C, \mathbf{w}_{10}^{\mathsf{T}}, \ldots, \mathbf{w}_{1N}^{\mathsf{T}}, \ldots, \mathbf{w}_{C0}^{\mathsf{T}}, \ldots, \mathbf{w}_{CN}^{\mathsf{T}}, \\ \mathbf{d}_{11}^{\mathsf{T}}, \ldots, \mathbf{d}_{1N}^{\mathsf{T}}, \ldots, \mathbf{d}_{C1}^{\mathsf{T}}, \ldots, \mathbf{d}_{CN}^{\mathsf{T}})^{\mathsf{T}}\end{aligned} \tag{17}$$

and a sparse cumulative feature vector $\Phi(x, z)$ that is laid out with the same slots as $\mathbf{w}$.

We apply sparselets to *all* filter slots of $\mathbf{w}$, i.e., the $\{\mathbf{w}_{ci}\}$. The part filters all have the same 6×6 shape, but the root filters, both within a mixture model and across classes, have a variety of dimensions. Unlike [8] and [11] we decompose the root filters, not just the part filters. To do this, we employ 3×3 sparselets and pad the root filters with an extra one or two rows and columns, as needed, to ensure that their dimensions are multiples of 3. Summed over the models for all 20 object classes [29] in the PASCAL VOC 2007 dataset [30], there are a total of 4954 3×3 subfilters. In our experiments below, we represent *all* of these subfilters by sparse linear combinations of only 256 sparselets — effectively achieving more than an order of magnitude reduction in the number of model parameters. The HOG image features are 32-dimensional, leading to a sparselet size of $m = 288$. Our dictionary is thus undercomplete — which is desirable from a runtime perspective. Our experimental results confirm that the sparselets spans a sufficient subspace to represent the subfilters in the 20 PASCAL classes (Section 6.3), as well as to generalize to previously unseen classes from the ImageNet dataset (Section 6.4). Our DPM sparselets are visualized in Fig. 3.

5.1 Latent SVM

The DPMs in [1] are learned by optimizing a latent SVM (LSVM):

$$\begin{aligned}\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{\lambda}{2} \|\mathbf{w}\|^2 \\ + \frac{1}{M} \sum_{i=1}^{M} \max\Big(0, 1 - y_i \max_{z \in \mathcal{Z}(x_i)} \mathbf{w}^{\mathsf{T}} \Phi(x_i, z)\Big).\end{aligned} \tag{18}$$

The objective function in Eq. (18) is not convex in $\mathbf{w}$ and in practice a local optimum is found by coordinate descent on an auxiliary function that upper bounds Eq. (18) (see [1] for details). The coordinate descent algorithm alternates between two steps. In the first step, the set $\mathcal{Z}(x_i)$ is made singleton—for each *positive* example—by setting its only member to be an optimal latent value assignment for example x_i. This step results in a convex optimization problem that has the same form as a structural SVM. It is therefore

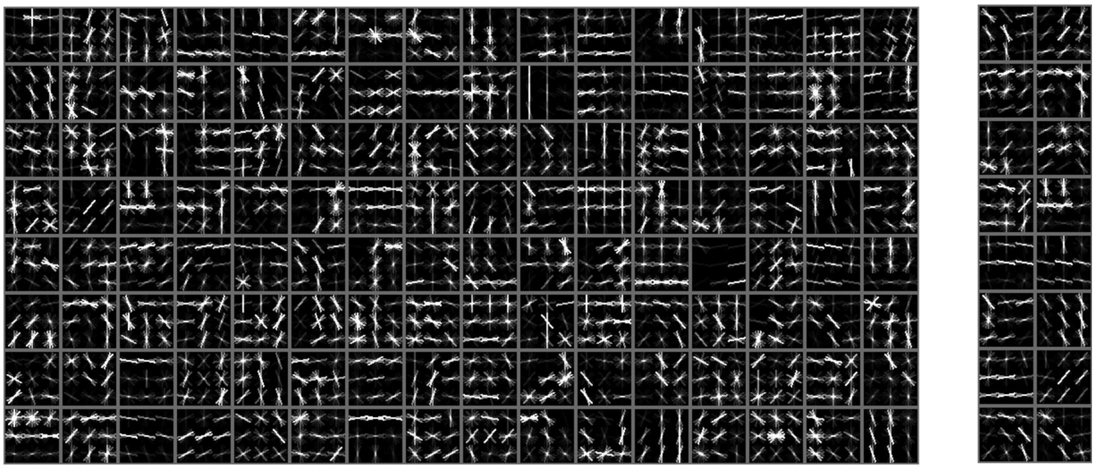

Fig. 3. (Left) 128 of the 256 sparselets learned from 20 DPMs trained on the PASCAL VOC 2007 dataset. (Right) The top 16 sparselets activated for the motorbike category.

straightforward to apply discriminative activation learning to a LSVM: we follow the same coordinate descent scheme and apply the SSVM problem transformation from Section 4.1 to the LSVM's convex optimization subproblem.

Our implementation is based on the `voc-release4` source code from [29]. To optimize the transformed objective function Eq. (15) when $R(\alpha)$ is either $R_{\text{Lasso}}(\alpha)$ or $R_{\text{EN}}(\alpha)$, we modified the default stochastic subgradient descent (SGD) code to implement the truncated gradient descent update of Langford et al. [31]. This method achieves actual sparsity by shrinking parameters and then truncating small values every few SGD iterations.

5.2 Visualizing Learned DPM Sparselets

Each DPM sparselet can be visualized as a 3×3 filter. In Fig. 3 (left) we show the positive weights of 128 of the 256 sparselets that we learned from DPMs for the 20 classes from the PASCAL VOC 2007 dataset. Regular structures, such as horizontal, vertical, and diagonal edges, as well as arcs and corners, are visible. We can order the sparselets activated for a particular category model by sorting them by the magnitude of their activation coefficients. Fig. 3 (right) shows the top 16 sparselets for the motorbike category. Some of the activated sparselets resemble circular fragments of wheels.

5.3 Image Classification

To illustrate generalized sparselets applicability beyond DPMs, we evaluated our approach on the Caltech-101 [24] (102 classes, including background) and Caltech-256 (257 classes) [25] datasets. Since our aim is not state-of-the-art accuracy, but rather to demonstrate our learning method, we implemented sparselets atop a basic, publicly available image classification framework. Specifically, we used the `phow_caltech101` method included in VLFeat [32]. This approach trains one-against-all linear SVMs using bag-of-visual-words features (600 word vocabulary, 2×2 and 4×4 spatial pooling, and an approximate χ^2 feature map [33]). In Section 6.5 we experiment with two block sizes, $m \in \{100, 200\}$. These values of m lead to 36,720 (or 18,360) blocks in total for the 102 Caltech-101 classifiers, and 92,520 (or 46,260) blocks in total for the 257 Caltech-256 classifiers. We represent all of these blocks as sparse linear combinations of $d = 40$ sparselets.

6 Experiments

We performed six sets of experiments. The first experiment evaluates the sensitivity of the sparselet dictionary learned from random subset of object classes. The second experiment was designed to evaluate the effect of sparselet block size when the precomputation time is fixed versus when the representation space is fixed. The third experiment compares each of the regularization methods described in Section 4.2. The fourth experiment was designed to evaluate how well a set of sparselets learned on one set of models generalizes to a new set of models when learning the activation vectors in our discriminative framework. The fifth experiment shows results in multiclass classification. The last experiment compares wall clock run time with and without GPU.

6.1 Sensitivity to Heterogeneity of Object Classes Used in Dictionary Construction

To test how sensitive the learned dictionary of sparselets is with respect to the set of object classes used for the dictionary construction, we designed the following experiment on PASCAL VOC 2007 [30] dataset. For a test object class, we ran five trials of constructing the dictionary from five randomly chosen object models excluding the test object class (five different dictionaries per class). Empirically (Table 1), a dictionary learned from a randomly chosen subset of the object classes shows negligible loss in average precision compared to the dictionary learned from all the classes with insignificant standard deviation. This also shows the dictionary learned on a subset of object classes generalizes to previously unseen object classes.

6.2 Effect of Different Sparselet Block Sizes

In practice we can divide a part filter into smaller subfilters before computing the sparselet representation. The subfilter size (which equals the sparselet size) determines certain runtime and memory tradeoffs. Let F be a $h_F \times w_F \times l$ filter, and let the sparselet size be $h_s \times w_s \times l$. We require that h_s and w_s are divisors of h_F and w_F, respectively, and divide F

TABLE 1
Statistics of Average Precision for All 20 Classes over Five Trials of Constructing the Dictionary from Five Randomly Chosen Classes (Five Different Dictionaries Per Class)

	Average Precision				
	Mean	**Std**	**Max**	**Min**	**Full Dict**
aeroplane	0.2922	0.0065	0.3024	0.2851	0.2966
bicycle	0.5542	0.0133	0.5747	0.5430	0.5688
bird	0.0690	0.0365	0.0948	0.0143	0.0343
boat	0.1236	0.0075	0.1352	0.1175	0.1394
bottle	0.1995	0.0104	0.2170	0.1903	0.2298
bus	0.4788	0.0129	0.4883	0.4581	0.5009
car	0.5479	0.0036	0.5526	0.5436	0.5640
cat	0.1296	0.0287	0.1631	0.0870	0.1432
chair	0.2008	0.0062	0.2075	0.1934	0.2057
cow	0.2226	0.0028	0.2258	0.2196	0.2384
diningtable	0.2136	0.0068	0.2215	0.2051	0.2381
dog	0.0574	0.0287	0.1054	0.0344	0.0590
horse	0.5408	0.0036	0.5442	0.5367	0.5542
motorbike	0.4611	0.0105	0.4721	0.4446	0.4724
person	0.3538	0.0080	0.3622	0.3459	0.3834
pottedplant	0.1048	0.0084	0.1163	0.0954	0.1127
sheep	0.1435	0.0148	0.1575	0.1236	0.1622
sofa	0.2940	0.0262	0.3144	0.2584	0.3191
train	0.4205	0.0102	0.4319	0.4039	0.4481
tvmonitor	0.3884	0.0059	0.3966	0.3826	0.3791

The last column (Full Dict) denotes the result when all 20 classes were used to construct the dictionary.

into an $h_F/h_s \times w_F/w_s$ array of tiled subfilters. We approximate (or "reconstruct") a filter response by summing over approximate subfilter responses.

Given precomputed sparselet responses, reconstructing the response to F requires at most $\lambda_0(h_F/h_s)(w_F/w_s)$ operations. Low-cost approximation is essential, so we fix the reconstruction budget for F at $\lambda_0(h_F/h_s)(w_F/w_s) \leq B_R$. Within this budget, we can use fewer, smaller sparselets, or more, larger sparselets. We consider two other budget constraints. *Precomputation time* B_P: convolving an input with the entire sparselet dictionary requires $h_s w_s l d$ operations. For a fixed budget $h_s w_s l d \leq B_P$, we can use more, smaller sparselets, or fewer, larger sparselets. *Representation space* B_S: the space required to store the intermediate representation is proportional to the dictionary size d.

We evaluated the filter reconstruction errors for four different subfilter sizes. For these experiments we fixed the reconstruction budget $\lambda_0(h_F/h_s)(w_F/w_s) = 112$ by setting λ_0 to be $\{112, 28, 13, 3\}$ for subfilter size of $\{6\times 6, 3\times 3, 2\times 2, 1\times 1\}$. Fig. 4 (left) shows the result as the subfilter sizes vary while both the precomputation time and reconstruction time budget is fixed. We set the dictionary size $d = \{128, 512, 1152, 4608\}$ for $\{6\times 6, 3\times 3, 2\ \times 2, 1\times 1\}$ sized sparselets to fix the precomputation budget $h_s w_s d = 4608$.

For fixed reconstruction and precomputation budgets B_R and B_P, we studied the effect of varying sparselet size. Empirically, filter reconstruction error always decreases as we decrease sparselet size. When there are not too many classes, the precomputation time is not fully amortized and we would like to make B_P small. For a fixed, small B_P we minimize reconstruction error by setting h_s and w_s to small values as shown is Fig. 4 (top). However, as we make the sparselets smaller, d grows, possibly making the representation space budget B_S too large. In our experiments, we balance memory usage with sparselet size by setting h_s and w_s to 3.

When precomputation is amortized, minimizing precomputation time is less important. However, in this case we are still concerned with keeping the intermediate representation reasonably small. Fig. 4 (bottom) shows the results as the subfilter sizes vary while both the representation space and reconstruction time budget is fixed. We fixed the dictionary size $d = 512$ for this experiment. By fixing the response and representation space budgets, we observe that using more, larger sparselets minimizes reconstruction error (at the expense of requiring a larger precomputation budget) as shown in Fig. 4 (bottom).

6.3 Comparison of Regularization Methods

We evaluated the reconstructive sparselets [8] and the discriminatively trained activation vectors [9] on the PASCAL VOC 2007 dataset [30]. Fig. 5 (left) shows the mean average precision (mAP) at various activation vector sparsity levels. We set the sparsity regularization constant λ_1 to $\{0.010, 0.015, 0.020\}$ for the lasso penalty ("R-Lasso") and to $\{0.025, 0.030, 0.035\}$ for the elastic net penalty ("R-EN"). For the combined ℓ_0 and ℓ_2 penalty, λ_0 was set to $\{48, 32, 16, 8, 4, 2\}$.

The ℓ_1-based regularization methods were very difficult to tune. Adjusting λ_1 to hit a desired sparsity level requires an expensive grid search. Additionally, the ratio between hinge-loss and the regularization term varied significantly between different classes, leading to a wide range of sparsity levels. Ultimately, these methods also underperformed in terms of mAP. Combined ℓ_0 and ℓ_2 regularization ("R–0,2 ORT" and "R–0,2 OMP"), in contrast, produces exactly the desired sparsity level and outperforms all other methods by a large margin. One interesting observation is that the mAP margin grows as the activation vectors become increasingly sparse.

6.4 Universality and Generalization to Previously Unseen Categories

To test the hypothesis that our learned dictionary of sparselets, in conjunction with the proposed discriminative activation training, are "universal" and generalize well, we used the sparselet dictionary learned from 20 PASCAL classes and evaluated detection performance on novel classes from the ImageNet [22] dataset. We selected nine categories (sailboat, bread, cake, candle, fish, goat, jeep, scissors and tire) that have substantial appearance changes from the PASCAL classes. Fig. 5 (right) shows that our method generalizes well to novel classes and maintains competitive detection performance even in the high sparsity regime.

6.5 Image Classification with Generalized Sparselets

Fig. 6 compares classification accuracy versus speedup factor (averaged over six machines with different CPU types). Generalized sparselets consistently provide a good speedup, however only the discriminatively trained

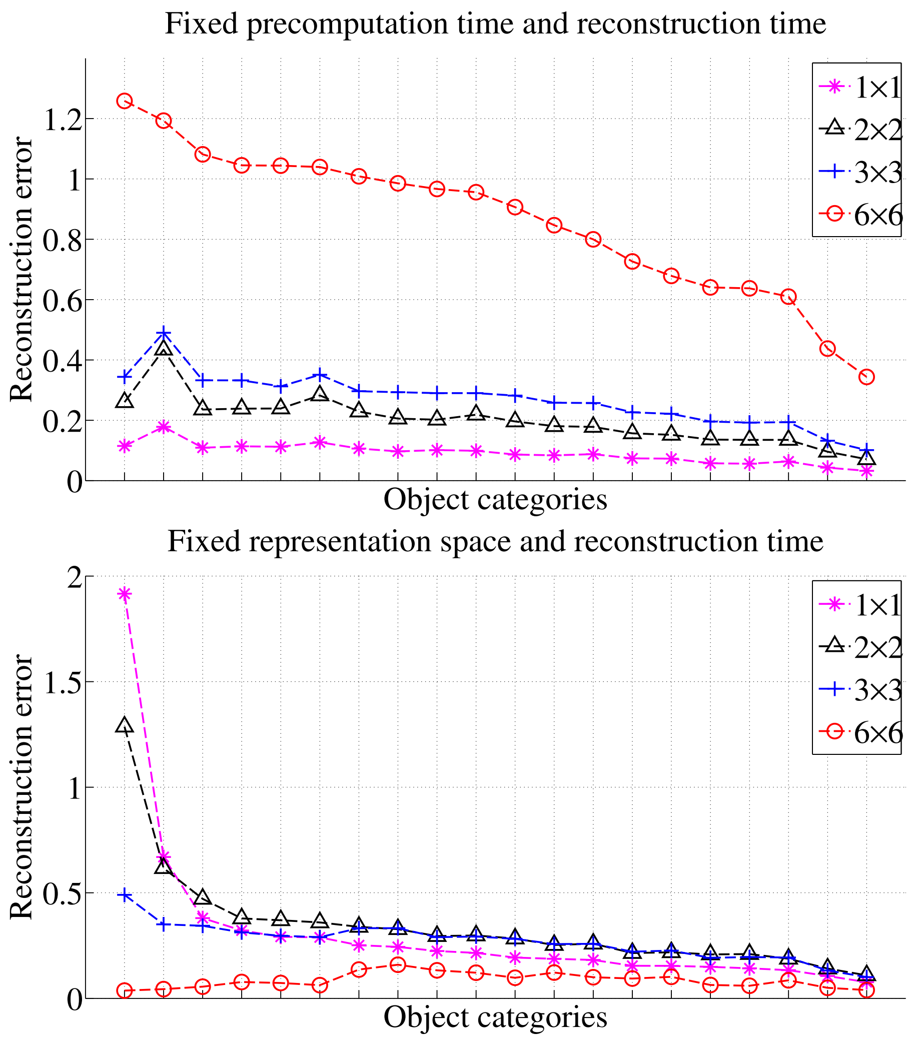

Fig. 4. Reconstruction error for all 20 object categories from PASCAL 2007 dataset as sparselet parameters are varied. The precomputation time is fixed in the top figure and the representation space is fixed on the bottom. Object categories are sorted by the reconstruction error by 6×6 in the top figure and by 1×1 in the bottom figure.

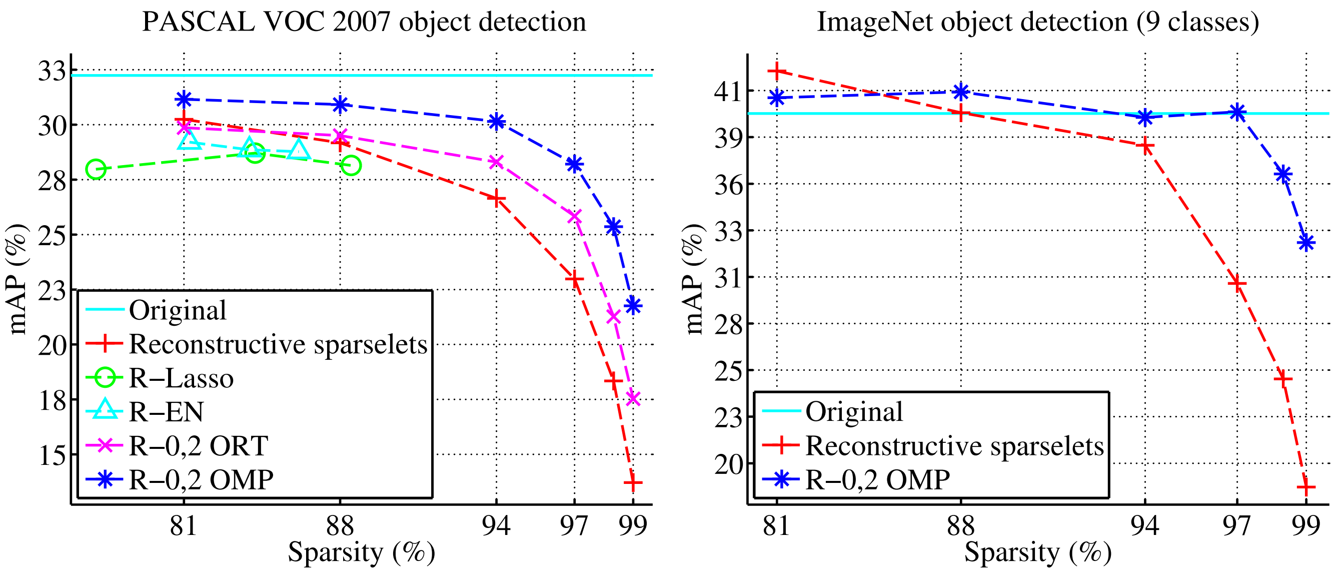

Fig. 5. Mean average precision (mAP) *vs.* sparsity for object detection on the PASCAL 2007 dataset (left) and for nine classes from ImageNet (right). The dictionary learned from the PASCAL detectors was used for the novel ImageNet classes. “Original” is the original linear model; “Reconstructive sparselets” is the baseline method from [8]; the remaining methods correspond to discriminative learning [9] with each of the regularizers described in Section 4.2.

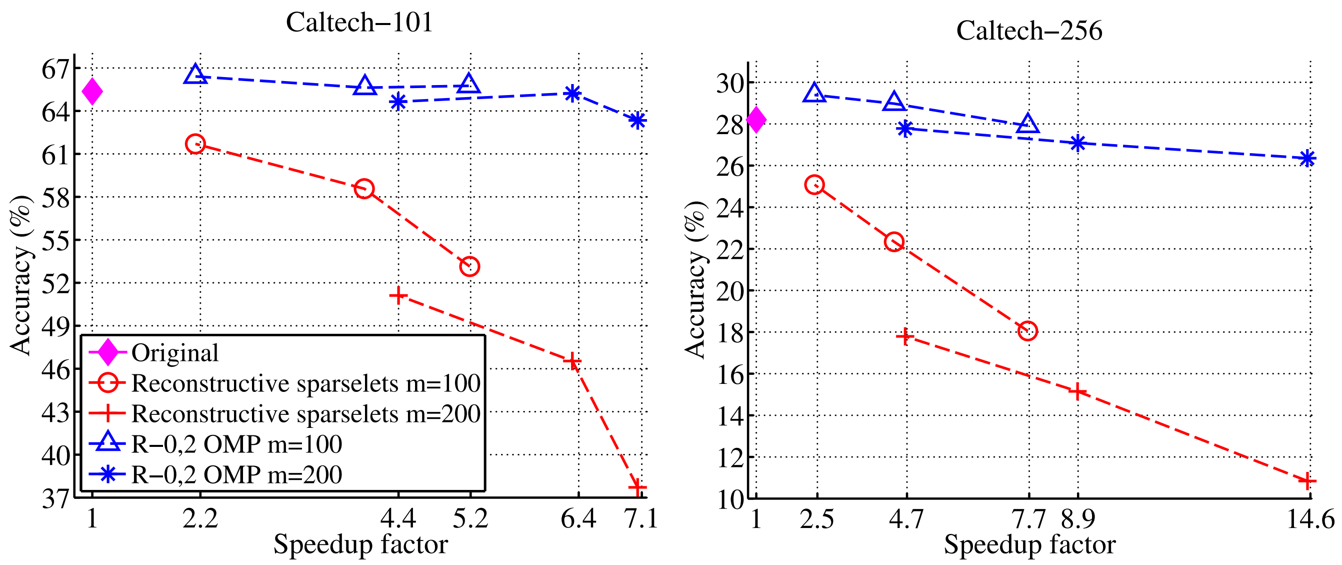

Fig. 6. Average classification accuracy *vs.* speedup factor for Caltech-{101, 256}.

sparselet activation models provide high accuracy, occasionally besting the original classifiers. In these experiments, we used a fixed dictionary size $d = 40$. We explored two block sizes $m = 100$ or 200. Each curve shows results at three sparsity levels: 0.6, 0.8, and 0.9. We trained and tested with 15 images per class on both datasets. As predicted by our cost analysis, increasing the class count (from 102 to 257) magnifies the speedup factor.

6.6 Run Time Experiments

We performed two sets of experiments to measure the wall clock runtime performance with and without GPU.

GPU experiment. Fig. 7 shows the relative comparisons for DPM implementation on GPU, reconstructive sparselets and discriminatively activated sparselets. For sparselets, the dictionary size K was set to 256 and the sparsity parameter λ_0 was varied in the following range $\{48, 32, 16, 8, 4, 2\}$ which corresponds to $\{81, 88, 94, 97, 98, 99\}$ percent sparsity

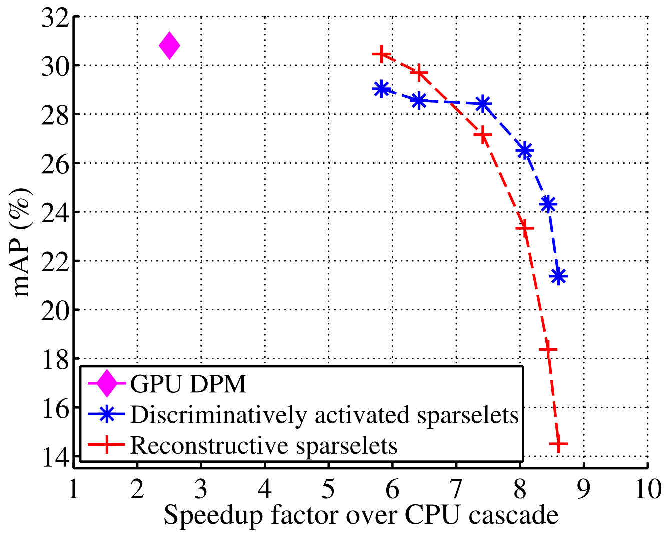

Fig. 7. Run time comparison for DPM implementation on GPU, reconstructive sparselets and discriminatively activated sparselets in contrast to CPU cascade.

respectively. As a reference for comparison, the CPU cascade took about 17.1 seconds per image to detect all 20 classes.

In all the GPU experiments, detection thresholds were automatically adjusted at runtime via binary search to deliver 5000 ± 10 detections per object class, per frame. This was done to ensure sufficient detection coverage for approaching each object category's maximum-recall point, while limiting memory consumption and runtime of the non-maximum suppression algorithm to a known upper bound.

The CPU cascade experiment was performed on a quad-core Intel Core i5-2400 CPU @ 3.10 GHz with 8 GB of RAM. GPU experiments were conducted on NVIDIA GeForce GTX 580 with 3 GB of memory.

Single core CPU experiment. On a single core CPU experiment (Intel Core i7) with 8 GB memory, we compare our method against our efficient baseline implementation of DPM which utilizes SSE floating point SIMD instructions.

The sparsity levels $\{81, 88, 94, 97, 98, 99\}$ percent resulted in $\{2.63, 3.99, 10.92, 15.81, 19.90, 22.57\}$ times speedup in filter convolution stage and $\{1.83, 2.25, 3.16, 3.39, 3.51, 3.54\}$ times speedup in end-to-end detection of 20 PASCAL classes per image, respectively. The variance of the experiments over test images were insignificant. The wall clock convolution and end-to-end time for detecting 20 classes per image per core for the baseline DPM code was 46.64 and 59.77 seconds respectively. For comparison, the speedup factor for the cascade method with respect to the baseline DPM code was $3.04\times$ per image per core for the end-to-end detection.

Even though the convolution stage is substantially accelerated via sparselets, other stages of the DPM framework (i.e. distance transform, zero padding filter responses) upper bounds the maximum possible end-to-end detection speedup (if the convolution stage and zero padding convolution responses takes 0 seconds) to be about $4\times$ per core. However, our implementation of sparselets nearly reaches maximum possible speedup (up to $3.5\times$). In both GPU and CPU implementations, discriminatively activated sparselets significantly outperformed reconstructive sparselets in the high speedup, high sparsity regime.

For completeness, we plan to maintain the source code at the following link:

https://github.com/rksltnl/sparselet-release1

7 Conclusion

We described an efficient object recognition model that simultaneously utilizes model redundancy and reconstruction sparsity to enable real-time multiclass object recognition. We also showed that the framework generalizes to any structured output prediction problem. The experimental results show that a fixed number of sparselets learned from one dataset generalizes to novel objects from other datasets. This allows for reusing the pretrained sparselets with various other designs of activation vectors. In the future, we would like to design more flexible activation vectors to enable computational complexity which is logarithmic in number of object classes.

Acknowledgments

H. Song was supported by Samsung Scholarship Foundation. P. Felzenszwalb and R. Girshick were supported in part by NSF grant IIS-0746569. T. Darrell was supported by DARPA contract W911NF-10-2-0059, by NSF awards IIS-0905647, IIS-0819984, and support from Toyota and Google.

References

[1] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part based models,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, Sep. 2010.

[2] L. Bourdev and J. Malik, “Poselets: Body part detectors trained using 3d human pose annotations,” in *Proc. 12th IEEE Int. Conf. Comput. Vis.*, 2009, pp. 1365–1372.

[3] W. Freeman and E. Adelson, “The design and use of steerable filters,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 9, pp. 891–906, Sep. 1991.

[4] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Application of dimensionality reduction in recommender systems—A case study,” in *Proc. ACM WebKDD Workshop*, 2000.

[5] L. Zhu, Y. Chen, A. Torralba, W. Freeman, and A. Yuille, “Part and appearance sharing: Recursive compositional models for multi-view multi-object detection,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2010, pp. 1919–1926.

[6] P. Ott and M. Everingham, “Shared parts for deformable part-based models,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2011, pp. 1513–1520.

[7] R. Girshick, P. Felzenszwalb, and D. McAllester, “Object detection with grammar models,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 442–450.

[8] H. Song, S. Zickler, T. Althoff, R. Girshick, M. Fritz, C. Geyer, F. Felzenszwalb, and T. Darrell, “Sparselet models for efficient multiclass object detection,” in *Proc. 12th Eur. Conf. Comput. Vis.*, 2012, pp. 802–815.

[9] R. Girshick, H. Song, and T. Darrell, “Discriminatively activated sparselets,” in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 196–204.

[10] P. Felzenszwalb, R. Girshick, and D. McAllester, “Cascade object detection with deformable part models,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2010, pp. 2241–2248.

[11] H. Pirsiavash and D. Ramanan, “Steerable part models,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2012, pp. 3226–3233.

[12] A. Torralba, K. Murphy, and W. Freeman, “Sharing visual features for multiclass and multiview object detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 5, pp. 854–869, May 2007.

[13] S. Fidler, M. Boben, and A. Leonardis, “Learning hierarchical compositional representations of object structure,” in *Object Categorization: Computer and Human Vision Perspectives*, S. Dickinson, A. Leonardis, B. Schiele, and M. J. Tarr, Eds. Cambridge, U.K.: Cambridge Univ. Press, 2009.

[14] K. Kreutz-Delgado, J. Murray, B. Rao, K. Engan, T. Lee, and T. Sejnowski, “Dictionary learning algorithms for sparse representation,” *Neural Comput.*, vol. 15, no. 2, pp. 349–396, 2003.

[15] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, “Online dictionary learning for sparse coding,” in *Proc. 26th Int. Conf. Mach. Learn.*, 2009, pp. 689–696.

[16] J. Mairal, F. Bach, and J. Ponce, “Task-driven dictionary learning,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 4, pp. 791–804, Apr. 2012.

[17] R. Manduchi, P. Perona, and D. Shy, “Efficient deformable filter banks,” *IEEE Trans. Signal Process.*, vol. 46, no. 4, pp. 1168–1173, Apr. 1998.

[18] L. Wolf, H. Jhuang, and T. Hazan, “Modeling appearances with low-rank SVM,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2007, pp. 1–6.

[19] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, “Large margin methods for structured and interdependent output variables,” *J. Mach. Learn. Res.*, vol. 6, no. 2, pp. 1453–1484, 2006.

[20] B. Taskar, C. Guestrin, and D. Koller, “Max-margin Markov networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2003, pp. 25–32.

[21] S. Mallat and Z. Zhang, “Matching pursuit with time-frequency dictionaries,” *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3397–3415, Dec. 1993.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2009, pp. 248–255.

[23] J. Deng, A. Berg, K. Li, and L. Fei-Fei, “What does classifying more than 10,000 image categories tell us?” in *Proc. 11th Eur. Conf. Comput. Vis.*, 2010, pp. 71–84.

[24] L. Fei-Fei, R. Fergus, and P. Perona, “One-shot learning of object categories,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 594–611, Apr. 2006.

[25] G. Griffin, A. Holub, and P. Perona. (2007). Caltech-256 object category dataset. California Inst. Technol., Pasadena, CA, USA, Tech. Rep. 7694. [Online]. Available: http://authors.library.caltech.edu/7694

[26] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2005, pp. 886–893.

[27] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *J. Royal Statist. Soc. Ser. B*, vol. 73, pp. 267–288, 1996.

[28] H. Zou and T. Hastie, “Regularization and variable selection via the elastic net,” *J. Royal Statist. Soc. Ser. B*, vol. 67, pp. 301–320, 2005.

[29] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester. (2010). Discriminatively trained deformable part models, release 4 [Online]. Available: http://people.cs.uchicago.edu/pff/latent-release4/

[30] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. (2007). The PASCAL visual object classes challenge 2007 (VOC2007) results [Online]. Available: http://www.pascal-network.org/challenges/VOC/voc2007/workshop/ index.html

[31] J. Langford, L. Li, and T. Zhang, “Sparse online learning via truncated gradient,” *J. Mach. Learn. Res.*, vol. 10, pp. 777–801, 2009.

[32] A. Vedaldi and B. Fulkerson. (2008). VLFeat: An open and portable library of computer vision algorithms [Online]. Available: http://www.vlfeat.org/

[33] A. Vedaldi and A. Zisserman, “Efficient additive kernels via explicit feature maps,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 3, pp. 480–492, Mar. 2012.

Hyun Oh Song is working toward the PhD degree in the Department of Computer Science at UC Berkeley. He also spent time at INRIA Rhone-Alpes as a visiting student researcher in Fall 2013. He has been supported by Samsung Lee Kun Hee scholarship foundation. His research interests are machine learning, computer vision, optimization algorithms, and artificial intelligence.

Ross Girshick received the the PhD degree in computer vision at The University of Chicago under the supervision of Pedro Felzenszwalb in April 2012. He is currently a postdoctoral fellow working with Jitendra Malik at UC Berkeley. His main research interests are in computer vision, AI, and machine learning. He is particularly focused on building models for object detection and recognition. These models aim to incorporate the "right" biases so that machine learning algorithms can understand image content from moderate to large-scale datasets. During his PhD, he spent time as a research intern at Microsoft Research Cambridge, United Kingdom working on human pose estimation from depth images. He has also participated in several first-place entries into the PASCAL VOC object detection challenge, and was awarded a "life-time achievement" prize for his work on deformable part models.

Stefan Zickler received the BA degree in cognitive science from SUNY Buffalo in 2005, and the PhD degree in computer science from Carnegie Mellon University in 2010. He is a principal robotics engineer at iRobot Corporations research group where he currently works on perception research and development. His doctoral thesis focused on physics-based real-time robot motion planning in adversarial multi-body environments. He has been the leader of Carnegie Mellons RoboCup Small Size League robot soccer team. He joined iRobot Corporation in 2010 where he has since worked on developing approaches for real-time object detection, activity recognition and robot autonomy.

Christopher Geyer started his career in computer vision in the GRASP Lab at the University of Pennsylvania, where he received the BSE and PhD degrees in computer science in 1999 and 2002, respectively. He is a scientist at Boston-based Berkshire Grey, Inc. As a post-doctoral researcher at the University of California, Berkeley from 2002 to 2005, he led a team to develop an autonomous landing capability for unmanned rotorcraft for the U.S. Defense Advanced Research Projects Agency (DARPA). From 2005 to 2008, he led research in perception for aerial vehicles at Carnegie Mellon University (CMU), and developed technology for sensing and avoiding general aviation aircraft for unmanned aerial vehicles (UAVs). While at CMU, he was also a member of CMU's Tartan Racing team, the team that won first place in the DARPA Urban Challenge. From 2008 to 2013, he led research and development in perception for unmanned and robotic systems at iRobot Corporation. In 2013, he was selected by the National Academy of Engineering to be a Gilbreth Lecturer. His interests include computer vision, robotics, human-robot interaction, and autonomy.

Pedro Felzenszwalb received the BS degree in computer science from Cornell University in 1999. He received the MS and PhD degrees in computer science from the Massachusetts Institute of Technology in 2001 and 2003. He was a postdoc at Cornell University from 2003 to 2004. He was a professor of computer science at the University of Chicago from 2004 to 2011. He joined Brown University in 2011, where he is currently an associate professor of engineering and computer science. His work has been supported by the National Science Foundation, including a CAREER award received in 2008. His main research interests are in computer vision, geometric algorithms and artificial intelligence. In 2010, he received the Longuet-Higgins Prize for a fundamental contribution to computer vision that withstood the test of time. He was a program chair for the 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) and is currently an associate editor of the *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Trevor Darrell received the BSE degree from the University of Pennsylvania in 1988, and the SM and PhD degrees from MIT in 1992 and 1996, respectively. He started his career in computer vision as an undergraduate researcher in Ruzena Bajcsy's GRASP lab. He is in the faculty of the CS Division of the EECS Department at UCB and is the vision group lead at ICSI. His interests include computer vision, machine learning, computer graphics, and perception-based human computer interfaces. He was previously in the faculty of the MIT EECS department from 1999-2008, where he directed the Vision Interface Group. He was a member of the research staff at Interval Research Corporation from 1996-1999.