Sparselet Models for Efficient Multiclass Object Detection

Hyun Oh Song[1], Stefan Zickler[2], Tim Althoff[1], Ross Girshick[3], Mario Fritz[4], Christopher Geyer[2], Pedro Felzenszwalb[5], and Trevor Darrell[1]

[1]UC Berkeley, [2]iRobot, [3]University of Chicago, [4]MPI Informatics, [5]Brown University
[1]{song,althoff,darrell}@eecs.berkeley.edu
[2]{szickler,cgeyer}@irobot.com [3]rbg@cs.uchicago.edu
[4]mfritz@mpi-inf.mpg.de [5]pff@brown.edu

Abstract. We develop an intermediate representation for deformable part models and show that this representation has favorable performance characteristics for multi-class problems when the number of classes is high. Our model uses sparse coding of part filters to represent each filter as a sparse linear combination of shared dictionary elements. This leads to a universal set of parts that are shared among all object classes. Reconstruction of the original part filter responses via sparse matrix-vector product reduces computation relative to conventional part filter convolutions. Our model is well suited to a parallel implementation, and we report a new GPU DPM implementation that takes advantage of sparse coding of part filters. The speed-up offered by our intermediate representation and parallel computation enable real-time DPM detection of 20 different object classes on a laptop computer.

Keywords: Sparse Coding, Object Detection, Deformable Part Models

1 Introduction

Scalable category-level recognition is a core requirement for visual competence in everyday environments: domains of modest complexity typically have hundreds to thousands of categories, and as one considers unconstrained search problems, the space of possible categories becomes practically unlimited. 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.

We introduce an intermediate representation into part-based recognition models whereby a fixed set of shared “basis” parts is used to reconstruct category responses and infer detections. We learn the set of basis parts so that reconstructing the response of a target model is extremely efficient: our method decomposes model parameters as sparse linear combinations of the basis elements. With this representation we can reconstruct approximate part responses using a sparse matrix-vector product instead of exhaustive convolutions.

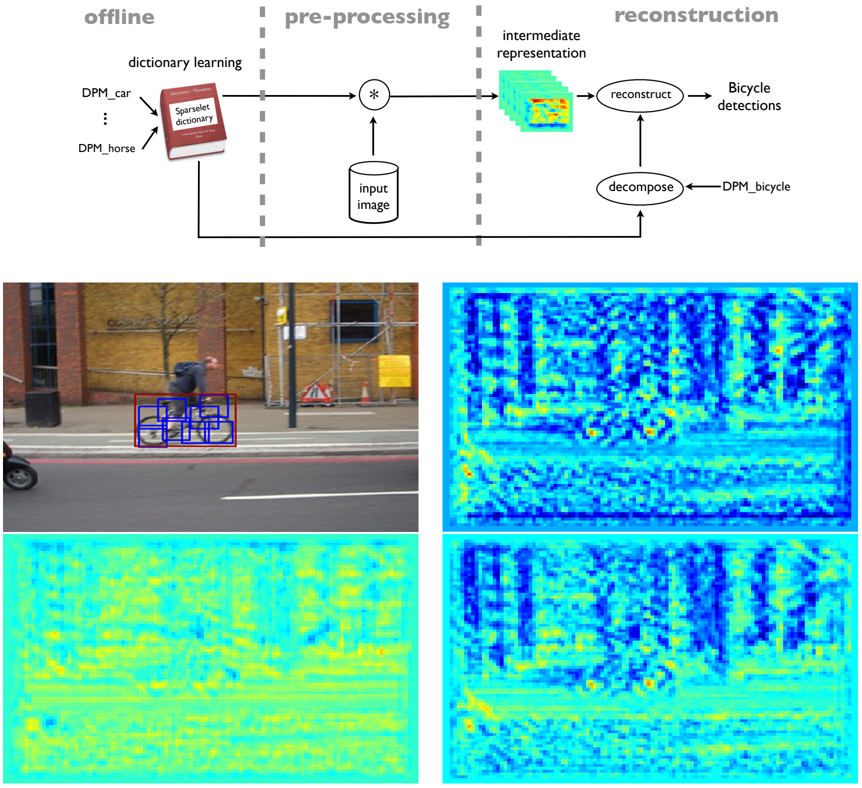
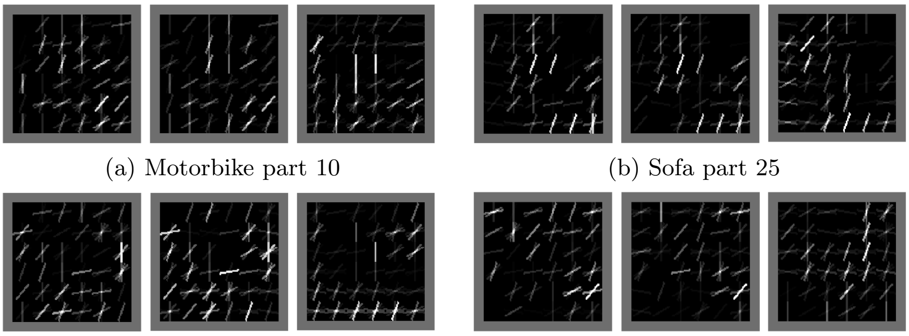


Fig. 1: Top: System concept. Middle row: example bicycle detection and true part filter responses for a wheel part from the bicycle model. Bottom row: reconstructed responses using SVD and sparselets (each using 20 bases). Our method maintains part response specificity (peaked responses at the wheels), while SVD reconstruction fails to maintain the sharpness. Best viewed in color.

In contrast to standard applications of sparse coding, where features are encoded as sparse combinations of dictionary elements, we 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.

As the number of object categories increases the speed-up provided by our "sparselet" model is the ratio of the original part filter size to the average number of non-zero coefficients in the sparse representation. For retrieval of a novel ("post-hoc") category in a large media corpus, low-level features and basis convolutions can be computed in a pre-processing step. For online detection, our representation is especially well-suited to a parallel architecture, where the memory and processor architecture allows for efficient reconstruction in a direct fashion (e.g., without a cascade). The method is also applicable to a cascaded CPU im-



(a) Motorbike part 10

(b) Sofa part 25

(c) Aeroplane part 27

(d) Cat part 34

Fig. 2: Each block shows a randomly selected part, its sparselet reconstruction, and its SVD reconstruction. In both cases 20 out of 216 bases were used. Note how sparselets preserve filter structure better than singular vectors.

plementation, but the speed improvement can be limited by memory cache issues relative to a GPU implementation. Our CUDA implementation of a sparselet-based DPM model is approximately 35 times faster than the fastest single CPU cascaded DPM implementations on benchmark PASCAL evaluations.

We evaluate our methods using the PASCAL VOC, ImageNet, and TRECVID MED datasets. We show 1) real-time performance on PASCAL VOC using 20 categories on a laptop computer, 2) that sparselets learned on PASCAL can effectively reconstruct detectors trained on the ImageNet dataset, and 3) that post-hoc detectors trained on ImageNet using PASCAL-derived bases can be effective at detecting objects related to TRECVID MED activities.

Although we focus on object and activity recognition, the conceptual contribution of this paper – sparse decomposition of filters – is generally applicable to a variety of multiclass classification settings where linear models are employed.

2 Related Work

Deformable part models have been proven to yield high accuracy on benchmark challenges, yet are computationally demanding. Previous efforts have addressed hypothesis pruning in a cascaded implementation [1], and in coarse-to-fine search schemes [2]. However, relatively little attention has been paid to the problem of scaling such models to detect hundreds or thousands of categories at near real-time speeds, or to quickly search large repositories of media for a category of interest not known a-priori.

Recent work has explored schemes for part sharing, including [3]. Our method implicitly shares part prototypes and can provide a significant improvement in speed by compressing the effective number of parts used in typical DPM models. Other authors have explored group regularization of a prototype representation [4], and linear manifold [5] and/or topic models [6] over visual classifier parameter spaces, but have not addressed the reconstruction of new models, nor the advantages of sparsity for large-scale detection.

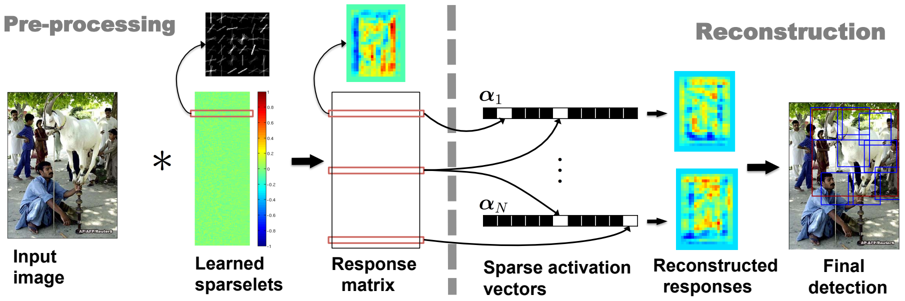


Fig. 3: Overview diagram of the our method. Once we evaluate the image with learned sparselets, the reconstruction phase can be done via efficient sparse matrix vector multiplications.

A complementary line of work uses classifier hierarchies to speed up classification by pruning classes while descending the classifier tree. We note a few, focusing on recent works with vision results. Some learn a tree in a top-down fashion [7, 8] by spectral clustering on the affinity matrix. Others optimize a discriminative objective [9, 10]. [11] builds a taxonomy of object classes based on shared features. These approaches may preemptively discard a correct detection if it falls on the wrong side of a hard decision boundary. Attempts have been made to address this using relaxed hierarchies [12, 13]. We note that sparselets may be combined with these approaches to further accelerate them.

3 Sparselets

In this paper we propose and evaluate sparse prototype representations for encoding deformable object models. We use a dictionary of parts learned in a sparse coding framework. This approach, which we term "sparselets", has the advantage of leading not only to fast and accurate multiclass object detection, but also offers an efficient intermediate representation for detecting new object categories. It is important to note that our approach learns a dictionary over *model parameters*, not over observed features, as is the customary application of existing sparse coding methods in the recognition literature.

While sparselets are applicable to a range of settings, we focus in this paper on the star-structured deformable part models (DPM) from [14]. Briefly, a DPM is composed of low-resolution root filters that describe an object category's global appearance and high-resolution part filters that capture local appearance. These filters are weight vectors over HOG [15] style features learned by optimizing a latent SVM [14]. The main computational bottleneck in applying these models is convolving their filters with a HOG feature pyramid. This fact is demonstrated by the cascade algorithm for detection with DPMs [1], which dramatically reduces detection time by computing filter convolutions at a reduced set of locations that is chosen based on learned pruning thresholds. Sparselets offer a complementary, shared-part approach for reducing filter convolution costs when the number of categories becomes large.

3.1 Sparse reconstruction of models

Sparse coding of model filters has several desirable properties, most significant among them for our post-hoc scenario being the relative efficiency of a sparse reconstruction at query time. Our objective is to find a generic dictionary of filters $D = \{D_1, D_2, \ldots, D_K\}$, called *sparselets*, that optimally approximates the part filters $P = \{P_1, P_2, \cdots, P_N\}$ pooled from a set of training models, subject to a sparsity constraint. Explicitly, we formulate the optimization problem

$$\begin{aligned} \min_{\alpha_{ij}, D_j} \quad & \sum_{i=1}^{N} ||\mathrm{vec}\,(P_i) - \sum_{j=1}^{K} \alpha_{ij} D_j||_2^2 \\ \text{subject to} \quad & ||\boldsymbol{\alpha}_i||_0 \leq \epsilon \qquad \forall i = 1, ..., N \\ & ||D_j||_2 = 1 \qquad \forall j = 1, ..., K \end{aligned} \tag{1}$$

where $P_i \in \mathbb{R}^{h \times h \times l}$ is a part filter, $D_j \in \mathbb{R}^{lh^2}$ is a dictionary element, $\boldsymbol{\alpha}_i \in \mathbb{R}^K$ is an activation vector, ϵ imposes a cap on the number of activations, h is the filter size (we assume square filters for simplicity), and l is the feature dimension.

Although the above optimization is NP-hard, greedy algorithms such as orthogonal matching pursuit algorithm (OMP) [16, 17] can be used to efficiently compute an approximate solution. OMP iteratively estimates the optimal matching projections of the input signal onto the dictionary D. The above optimization problem is convex with respect to D 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 D and vice versa. For our experiments we use the online dictionary learning algorithm from [18]. Figure 2 shows randomly chosen part filters from models trained on the PASCAL VOC 2007 [19] dataset and compares our sparselet reconstruction with $\epsilon = 20$ to the SVD-based reconstruction using 20 singular bases out of the full set of 216.

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 $\Psi * P_i \approx \Psi * (\sum_j \alpha_{ij} D_j) = \sum_j \alpha_{ij} (\Psi * D_j)$, where $*$ denotes the convolution operator. 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 Eqn (2):

$$\begin{bmatrix} \text{---} \; \Psi * P_1 \; \text{---} \\ \text{---} \; \Psi * P_2 \; \text{---} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \text{---} \; \Psi * P_N \; \text{---} \end{bmatrix} \approx \begin{bmatrix} \text{---} \; \boldsymbol{\alpha}_1 \; \text{---} \\ \text{---} \; \boldsymbol{\alpha}_2 \; \text{---} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \text{---} \; \boldsymbol{\alpha}_N \; \text{---} \end{bmatrix} \begin{bmatrix} \text{---} \; \Psi * D_1 \; \text{---} \\ \text{---} \; \Psi * D_2 \; \text{---} \\ \vdots \\ \text{---} \; \Psi * D_K \; \text{---} \end{bmatrix} = AM, \qquad (2)$$

where M is a matrix of all sparselet responses, A is the matrix of sparse activation vectors. As the number of classes increases, we amortize the time required to compute the intermediate representation M.

In practice the score of a DPM can be reconstructed as following,

$$\begin{aligned} \text{score}_{\text{recon}}(\omega) &= m_0(\omega) + \sum_{i=1}^{N} \max_{\delta} s_i(\omega + \delta) - d_i(\delta) \\ \text{where } s_i(\omega) &= \sum_{\substack{j=1 \\ \forall \alpha_{ij} \neq 0}}^{K} \alpha_{ij} (\Psi * D_j)[\omega]. \end{aligned} \qquad (3)$$

Here d_i are quadratic deformation costs, δ is a displacement and ω is a position and scale in a feature pyramid. After precomputation, the reconstructed part filter score, $s_i(\omega)$, simplifies to

$$s_i(\omega) = \sum_{\substack{j=1 \\ \forall \alpha_{ij} \neq 0}}^{K} \alpha_{ij} M_j[\omega]. \qquad (4)$$

Note that the summation is only over non-zero elements of the sparse vector $\boldsymbol{\alpha}_i$. Additionally, this could be efficiently implemented as sparse matrix multiplications or lookups. Figure 1 shows a sample reconstruction and Figure 3 summarizes our framework.

For online detection, we convolve the query image with the sparselets and do sparse reconstruction in turn per frame. For dictionary size K, total number of filters N, filter size h and feature dimension l, an exhaustive convolution based detection scheme requires approximately Nlh^2 operations per feature pyramid location. Our scheme requires approximately $Klh^2 + N\mathbb{E}[\, ||\boldsymbol{\alpha}_i||_0 \,]$ operations. The first term is from convolution with sparselets and the second term is the average activation level from the sparse reconstruction. Note that the convolution time (Klh^2) is not dependent on number of classes or model filters (N) and depends only on the size of sparselet dictionary (K) which is fixed and does not need to grow with the number of classes.

As the number of classes or model filters grows, the convolution time gets amortized and the speedup factor becomes the ratio between the complexity of the convolution kernel and the average activation. Therefore, the sparsity in the sparselet activation vector becomes the key, yielding the speedup

$$\frac{lh^2}{\mathbb{E}[\ ||\boldsymbol{\alpha}_i||_0\ ]}$$

For example, reconstructing response from a 6 by 6 kernel with feature dimension 6 and average activation level of 20, we would get more than an order of magnitude speedup in terms of number of arithmetic operations.

4 Implementation

4.1 CPU Cascaded Sparselets

We first implemented our model using the cascade code of [1], and interleaved sparse reconstruction with the cascaded search. In this implementation we used a sparselet representation for the filters over PCA features as well. We found that a sparse reconstruction model was still successful. We report results in the following section on the ability of sparselet models to reconstruct held-out and/or post-hoc categories; that is, categories which were not used in training the part dictionary model. While reconstruction can be accurate with a small number of bases, as is shown below, the memory cache behavior on a conventional CPU limited the overall speedup of the method, even when the theoretical ratio of the size of the convolution window to the number of sparse bases was relatively large. To address this, we turned to a GPU implementation, described in the following section.

4.2 Vanilla DPM and Sparselets Implementations on GPU

One of our contributions is the highest reported throughput for DPM-based methods, which is due to both the use of GPUs and sparselets. We describe here our CUDA[1] implementation of both a “Vanilla” DPM approach, which follows the classic implementation in [14], as well as sparselets. We expected DPM to benefit from porting to the GPU since a large fraction of the computation time of the original implementation is spent on convolution, HOG computation and distance transforms - all operations that are parallelizable. GPUs offer a massively parallel computing architecture and have been successfully used to speed up similar algorithms, e.g. HOG-based pedestrian detection [15].

[1] Compute Unified Device Architecture (CUDA) is the programming paradigm for Nvidia GPUs. Briefly, the GPU code is specified in kernels written in CUDA - essentially C/C++ with some additional language constructs. Each core executes compiled kernel code in a single thread. See [20] for more information.

The main steps in the GPU implementation are as follows. First, the input image is transferred to the GPU. Second, image pyramids and HOG features are computed as in [14] on the GPU (as are all subsequent operations). Third, filter responses to root, part or part basis filters are computed at each of S scales in the HOG pyramid. For Vanilla DPM, we make SR kernel calls for all R root filter responses, and SN kernel calls for all P part filter responses. A single kernel call instantiates kernel threads for each HOG cell in the given pyramid level. For Sparselets, we make SR kernel calls for the root filters, where each kernel call runs over all HOG pixels in the pyramid level; then, we make S kernel calls for the part bases, where each kernel call runs over each HOG cell in the pyramid level *and* each of the K part bases.To reconstruct all N part responses, we make SN kernel calls, where each part filter is reconstructed from the part bases responses.

We then compute bounded distance transforms (BDT) of the part filter responses. In the cascade implementation of DPM [1], part location offsets are bounded. Furthermore, all deformation penalties learned in training are separable. We therefore employ two one-dimensional BDTs in the horizontal and vertical directions. Each is implemented as a *min* convolution with a length-11 quadratic penalty filter. A total of $2SN$ kernel calls evaluate the horizontal, then vertical, BDT for all scales and parts. Finally, all root filter responses and BDT outputs are summed into a single score via SR kernel calls. Those part locations with scores above threshold are put in a list – added asynchronously by kernel threads and protected by an atomic lock on the GPU. The list of candidate detections is then copied to CPU memory. All remaining operations such as non-maxima suppression are performed on the CPU and follow the original implementation.

4.3 Sparselet size

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 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 $\epsilon(h_F/h_s)(w_F/w_s)$ operations. Low-cost approximation is essential, so we fix the reconstruction budget for F at $\epsilon(h_F/h_s)(w_F/w_s) \leq B_R$. 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 $h_s w_s l K$ operations. For a fixed budget $h_s w_s l K \leq B_P$, 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 K.

					Reconstruction error			
$h_s \times w_s$	K	$h_s w_s K$	ϵ	$(h_F/h_s)(w_F/w_s)$	**bicycle**	**car**	**cat**	**person**
6 × 6	128	4608	112	1	1.0645	1.0349	0.8521	1.1939
3 × 3	512	4608	28	4	0.3116	0.3360	0.2552	0.4573
2 × 2	1152	4608	13	9	0.2298	0.2706	0.1763	0.4007
1 × 1	4608	4608	3	36	0.1062	0.1200	0.0820	0.1635
6 × 6	512	18432	112	1	0.0893	0.0528	0.1134	0.0472
3 × 3	512	4608	28	4	0.3116	0.3360	0.2552	0.4573
2 × 2	512	2048	13	9	0.3833	0.5962	0.2561	1.2280
1 × 1	512	512	3	36	0.3172	0.6599	0.1817	1.8594

Table 1: Reconstruction error for four classes as sparselet parameters are varied. The reconstruction budget (B_R) is fixed for the whole table. The precomputation budget (B_P) is fixed in the top half of the table and varies in the bottom half.

For fixed reconstruction and precomputation budgets B_R and B_P, we studied the effect of varying sparselet size. Empirically (Table 1), filter reconstruction error always decreases as we decrease sparselet size.

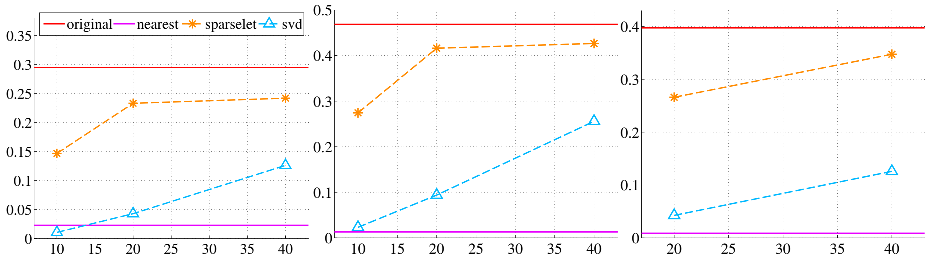
When there are not too many classes, the precomputation time is not fully amortized and we would like to make B_P small. For a fixed, small B_P we minimize reconstruction error by setting h_s and w_s to small values. However, as we make the sparselets smaller, K grows, possibly making the representation space budget B_S too large. In our GPU 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. 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). Therefore, in the CPU-based experiments which focused on the offline setting we use larger 6 × 6 sparselets.

5 Experiments

5.1 Evaluation on unseen categories

We performed experiments to analyze the detection performance of reconstructed models for previously unseen categories at a given level of sparsity. We experimented with three datasets: PASCAL VOC 2007 [19], ImageNet [21], and TRECVID [22]. To compute ground truth AP, we ran cascaded deformable part models [1] trained on the held-out category. For a baseline, we extracted singular vectors learned from the training models and estimated the reconstruction



(a) 20 PASCAL categories (b) 9 PASCAL categories (c) 9 ImageNet categories

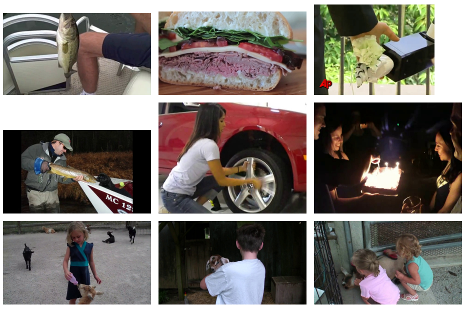
Fig. 4: Results on held-out evaluation on PASCAL VOC 2007 [19] and ImageNet[21] dataset. Y-axes show class averaged AP. X-axes represent number of bases used in reconstruction for SVD and sparselets.

weight vectors from the previously unseen query category models using a reconstruction from a linear basis of the top-k singular vectors. We also explored a nearest-neighbor-of-parts baseline where the query object model retrieves closest matching part filters (in L2 distance) from the pool of training object models. The global detection threshold was fixed to -1.1 for all object models throughout the experiment for consistency. This number was roughly the saturation threshold for AP evaluation.

To test the reconstruction generalization performance on previously unseen category models we first performed leave one class out evaluation where we used dictionaries and the set of singular vectors that are trained on all other classes.

Figure 4(a) shows the experimental results in AP for all 20 classes from PASCAL VOC 2007 test dataset [19]. Figure 4(b) shows AP for 9 categories that have AP above 0.3. This shows that the reconstruction error is relatively small for well performing models compared to mediocre DPM models. Our intuition is that well performing models have relatively higher classification margin than others and are more tolerant to approximation errors. We can see from Figure 4 that our sparse reconstruction method preserves most of AP on average with only 20 bases while SVD reconstruction does not quite preserve the AP at the same level of reconstruction budget. Although nearest-neighbor-of-parts baseline has poor performance overall, on subset of categories that DPM detector has poor detection performance (e.g. bird, dog and potted plant), it worked as well as or a slightly better than the original held-out query models. Please refer to the supplementary material for full per class AP table.

Next, we investigated whether a part dictionary model trained with PASCAL categories would work on other datasets. We manually selected a set of object categories which are related to events in the 2011 TRECVID MED challenge [22]: sailboat, bread, cake, candle, fish, goat, jeep, scissors, and tire. We trained DPM models using data sampled from ImageNet [21] for these classes. We used the ImageNet bounding boxes for training, and manually annotated 200 additional test images that had at least one instance of the above object categories with



Methods	Average Precision
original	0.4853
svd 20	0.0571
svd 40	0.1167
sparselet 20	0.3323
sparselet 40	0.3911

Fig. 5: Left: Example keyframes on TRECVID [22] dataset. Right: Average precision of retrieved examples using models trained with ImageNet training data reconstructed with sparselet representations learned from PASCAL categories. Numbers 20 and 40 represent the number of bases used in the reconstruction.

bounding boxes. We tested how the dictionary learned from PASCAL models in the previous experiment performed when approximating previously unseen novel categories trained and tested on ImageNet imagery.

Figure 4(c) shows the AP for the 9 categories: the dictionary of parts learned from PASCAL does transfer to novel categories from ImageNet domain. With the domain change however, we can see that SVD reconstruction with 40 bases preserves only about 12% to 15% of the original AP while sparselet reconstruction with 40 bases preserves about 88% to 93% of the AP. Again we can observe the same behavior that well performing DPM models are more tolerant on reconstruction errors than mediocre models. Please refer to the supplementary material for average AP plot of subset of classes which have AP above 0.3.

Finally, we tested how well these classifiers performed and were reconstructed on TRECVID imagery. The contents of TRECVID videos are highly variant, for example, “wedding ceremony” varies from a traditional catholic mass, to a Hindi ceremony, to home-made music videos, so precision is only high at relatively low recall in this model transfer scenario even for the baseline model. Figure 5 shows example frames from the event kit and category averaged precision for top 50 retrieved examples (based on sampled keyframes annotated using AMT [23]). We can see that sparse reconstruction significantly improves the precision as compared to SVD reconstruction for an equivalently compact intermediate representation.

5.2 Runtimes and Performance on the PASCAL VOC 2007 Dataset

In this section we present results of experiments examining runtime, and mean average precision (AP) of *Cascade DPM* [1] on a CPU, *Vanilla DPM* on a GPU, and *Sparselets* on a GPU. Figure 6 (top) plots runtime at 640×480 resolution as a function of the number of objects, where for sparselets we use $(K, \epsilon) = (512, 4)$. Runtimes were measured on a 3.1 GHz Intel Core i5 CPU, and an Nvidia GTX 580 GPU with 512 cores running at 772 MHz. We used the 20-object VOC 2007 dataset [19] to measure runtime, and we repeated VOC 2007 classes to get to 60 classes. The dashed portion is an extrapolation based on a linear model of runtime. Overall there is about a 32× speedup of sparselets on a GPU over Cascade DPM on a CPU - equally attributable to the GPU and sparselets.

We also evaluated mean AP for various configurations of (K, ϵ). Figure 6 (bottom) presents mean AP on VOC 2007 vs. speedup over the Cascade DPM on CPU implementation. The lower/blue group of points represent sweet spots in the "online" case and is based on runtimes measured on 20 classes. The upper/orange group is based on the "post-hoc" case where we measure the incremental runtime of a single class only. In both cases we only show those points not dominated by any other configuration - neither AP greater nor runtime faster. In the online case, we achieve speedups up to 25× without significant loss of AP over Vanilla DPM on a GPU. In the post-hoc case where precomputation is not counted, larger K and lower ϵ are favored, only the latter of which counts against per-class runtime, thus achieving even larger speedups - up to 35×.

In the supplemental video we show sparselets detecting 20 PASCAL object classes on video in real time on a laptop computer.

6 Conclusion

We introduced sparse intermediate representations that enable real-time multiclass object detection and efficient post-hoc category retrieval. Our results show that sparselets exploit the intrinsic redundancy among model filters and can generalize to previously unseen categories from other domains. Our model is well suited to a parallel implementation, and we report a new GPU DPM implementation which takes advantage of sparse coding of part filters. We achieve state of the art performance one to two orders of magnitude faster than the fastest current deformable part model implementations.

Acknowledgments. S. Zickler and C. Geyer were supported by DARPA contract W911NF-10-C-0081. 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. An earlier version of this work appeared in [24].

References

1. Felzenszwalb, P.F., Girshick, R.B., McAllester, D.A.: Cascade object detection with deformable part models. In: CVPR. (2010)
2. Pedersoli, M., Vedaldi, A., Gonzàlez, J.: A coarse-to-fine approach for fast deformable object detection. In: CVPR. (2011)
3. Ott, P., Everingham, M.: Shared parts for deformable part-based models. In: CVPR. (2011) 1513–1520
4. Pirsiavash, H., Ramanan, D., Fowlkes, C.: Bilinear classifiers for visual recognition. In: NIPS. (2009)

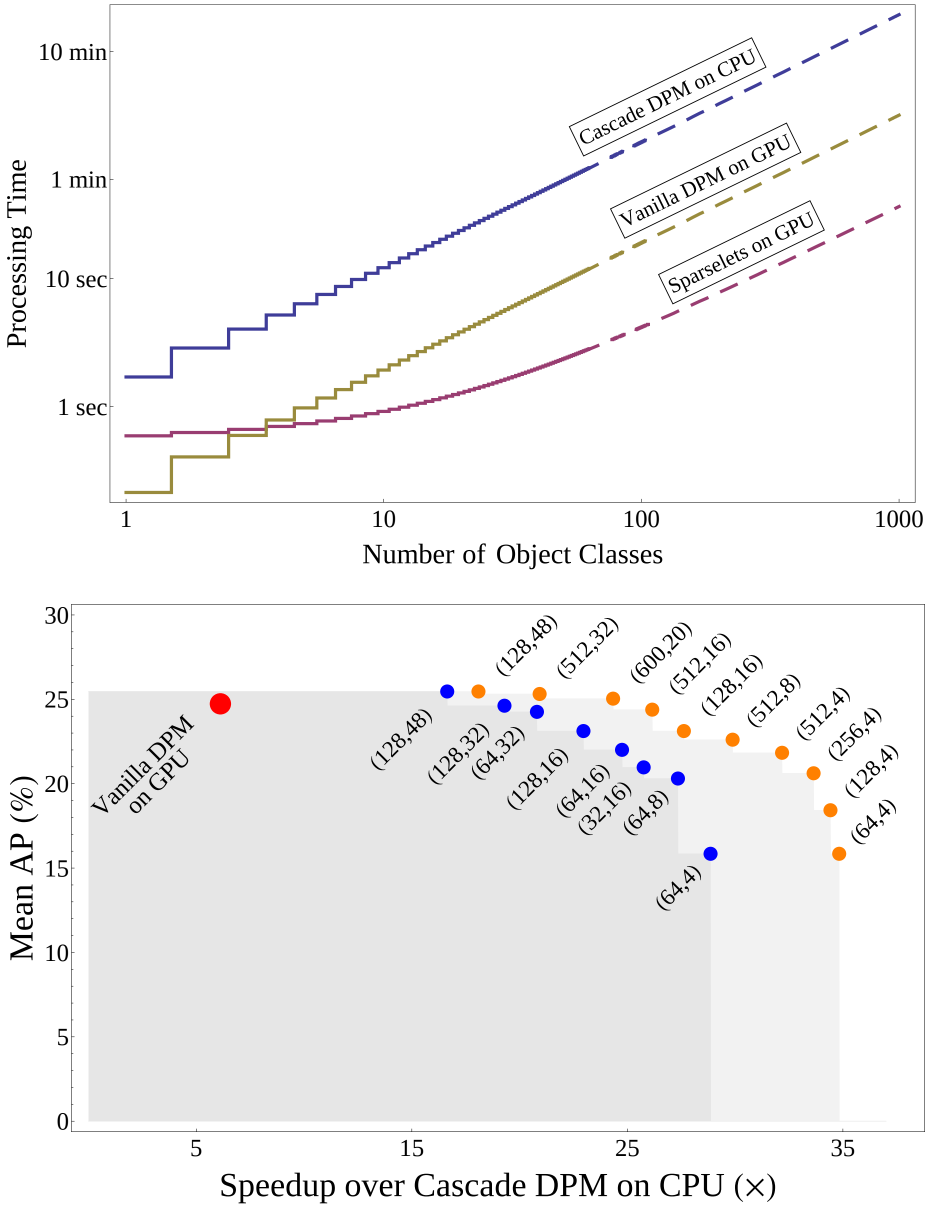


Fig. 6: Top: Comparison of cascade algorithm on CPU vs. vanilla DPM on GPU vs. sparselets accelerated DPM on GPU as number of object classes grows. Sparselet accelerated DPM on GPU offers approximately 35 times faster compared to the cascade implementation on CPU. Frame size used in this experiment was 640×480. Bottom: Speedup vs. Mean AP. Blue dots are for "online" results measuring end-to-end time. Orange dots are for "post-hoc" case. The tuple in parenthesis denote (K, ϵ).

5. Quattoni, A., Collins, M., Darrell, T.: Transfer learning for image classification with sparse prototype representations. In: CVPR. (2008)
6. Fritz, M., Schiele, B.: Decomposition, discovery and detection of visual categories using topic models. In: CVPR. (2008)
7. Griffin, G., Perona, P.: Learning and using taxonomies for fast visual categorization. In: CVPR. (2008)
8. Bengio, S., Weston, J., Grangier, D.: Label embedding trees for large multi-class tasks. In: NIPS. (2010)
9. Binder, A., Müller, K.R., Kawanabe, M.: On taxonomies for multi-class image categorization. International Journal of Computer Vision **99**(3) (2012) 281–301
10. Lai, K., Bo, L., Ren, X., Fox, D.: A scalable tree-based approach for joint object and pose recognition. In: Twenty-Fifth Conference on Artificial Intelligence (AAAI). (August 2011)
11. Razavi, N., Gall, J., Gool, L.J.V.: Scalable multi-class object detection. In: CVPR. (2011) 1505–1512
12. Marszalek, M., Schmid, C.: Constructing category hierarchies for visual recognition. In: Proceedings of the 10th European Conference on Computer Vision: Part IV. ECCV '08, Berlin, Heidelberg, Springer-Verlag (2008) 479–491
13. Gao, T., Koller, D.: Discriminative learning of relaxed hierarchy for large-scale visual recognition. In: ICCV. (2011)
14. Felzenszwalb, P., Girshick, R., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part-based models. IEEE Transactions on Pattern Analysis and Machine Intelligence **32**(9) (2010) 1627–1645
15. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: CVPR. (2005)
16. Cotter, S.F., Rao, B.D., Kreutz-Delgado, K., Adler, J.: Forward sequential algorithms for best basis selection. IEEE Proceedings Vision Image and Signal Processing **146**(5) (1999) 235
17. Mallat, S.G., Zhang, Z.: Matching pursuits with time-frequency dictionaries. IEEE Transactions on Signal Processing **41**(12) (1993) 3397–3415
18. Mairal, J., Bach, F., Ponce, J., Sapiro, G.: Online learning for matrix factorization and sparse coding. Journal of Machine Learning Research **11** (2010) 19–60
19. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.
20. NVIDIA: CUDA Technology. http://www.nvidia.com/CUDA
21. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR. (2009)
22. Smeaton, A.F., Over, P., Kraaij, W.: Evaluation campaigns and trecvid. In: MIR '06: Proceedings of the 8th ACM International Workshop on Multimedia Information Retrieval, New York, NY, USA, ACM Press (2006) 321–330
23. Amazon Mechanical Turk. http://www.mturk.com
24. Song, H.O., Fritz, M., Althoff, T., Darrell, T.: Don't look back: Post-hoc category detection via sparse reconstruction. Technical Report UCB/EECS-2012-16, EECS Department, University of California, Berkeley (Jan 2012)