Hierarchical Cascade of Classifiers for Efficient Poselet Evaluation

Bo Chen[1]
bchen3@caltech.edu
Pietro Perona[1]
perona@caltech.edu
Lubomir Bourdev[2]
lubomir@fb.com

[1] Computation and Neural Systems
California Institute of Technology
California, USA

[2] Facebook AI Research,
Menlo Park, California, USA

Abstract

Poselets have been used in a variety of computer vision tasks, such as detection, segmentation, action classification, pose estimation and action recognition, often achieving state-of-the-art performance. Poselet evaluation, however, is computationally intensive as it involves running thousands of scanning window classifiers. We present an algorithm for training a hierarchical cascade of part-based detectors and apply it to speed up poselet evaluation. Our cascade hierarchy leverages common components shared across poselets. We generate a family of cascade hierarchies, including trees that grow logarithmically on the number of poselet classifiers. Our algorithm, under some reasonable assumptions, finds the optimal tree structure that maximizes speed for a given target detection rate. We test our system on the PASCAL dataset and show an order of magnitude speedup at less than 1% loss in AP.

1 Introduction

Poselets [3] have been used to achieve state-of-the-art performance on problems such as object detection [3], attribute classification [4], fine-grain classification [35], action recognition [21, 34], action detection [27], and pose estimation [21, 33]. While Convolutional Neural Nets have lead to recent significant gain across multiple vision problems [16, 30], the current best methods for person detection [5] and attributes of people [36] result from combining poselets and CNNs.

One of the biggest drawbacks of poselets is their evaluation speed. A typical poselet system runs a large number of poselet classifiers, which are linear filters, on scanning windows of an image. For example, for a typical PASCAL [11] image, the implementation provided by [3] uses 150 poselets in the person category and performs about $90K * 150 = 13M$ classifier evaluations. Running all PASCAL categories on a photo (assuming each consists of 150 poselets) would amount to more than 250M evaluations, which takes three minutes on an Intel Xeon CPU at 2.67Ghz. This demanding computation is the bottleneck for large-scale deployment, as many applications require fixed and low max latency, such as the self-driving car, face correction in cameras, or interactive face tagging.

To address the computational complexity we develop a hierarchical cascade of classifiers that quickly filters out background and non-promising poselet types for each scanning window. Our algorithm organizes the poselet types in a hierarchy, where poselet types similar in appearance can be filtered out or detected together (Fig. 1). Under certain assumptions, our algorithm identifies the optimal (fastest) hierarchy for a given target detection rate. Our experiments on the PASCAL dataset show more than an order of magnitude speedup in the person category for a small and controllable loss in accuracy. We also demonstrate a logarithmic dependence of the computational complexity on the number of poselet types.

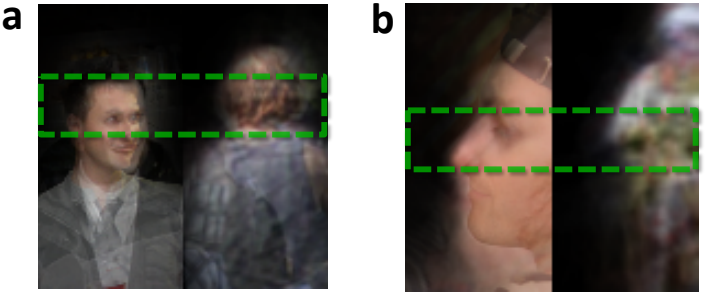

Figure 1: We can leverage the common appearance shared by different poselet types to train efficient multi-way classifiers. **(a)** A front-facing and back-facing poselet have similar appearance. **(b)** Poselets capturing completely unrelated body parts can also share similarity. The oval shape of a profile face, at a crude level, resembles the oval shape of a cropped out pedestrian figure. Even if two poselets look very different, they may share parts of the pattern. Our algorithm automatically finds such shared patterns (horizontal stripes highlighted in dashed boxes) and uses them to build classifiers to detect both poselet types together.

Since our algorithm focuses on the complexity in relation to the number of poselet types, it complements approaches that attempt to reduce the number of scanning windows (e.g. [17, 26]). Moreover, although developed for poselets, our algorithm is general and applicable to speeding up any part-based detector with a large number of parts such as the Deformable Parts Model [13] and deep convolutional nets [30]. Poselets are a good setup because of the large number of part classifiers. In addition, our experiments show that the poselets model has built-in redundancy which means that our efficient search can lose a large number of the poselets with very minimal overall reduction in AP at the object level hypothesis.

2 Related Work

Although we are the first to propose learning a hierarchical cascade for speeding up poselets while providing detection rate guarantees, the literature on speeding up part-based detectors in general is vast. Here we overview without the intention of being extensive. One common approach is to reduce detection to a binary classification problem at each sliding window and use active evaluation [6, 18, 19, 25] to reduce the number of evaluations per window. These approaches select the best part classifier based on the output of selected classifiers. The drawback is that the computational cost of selecting the next classifier is typically linear in the number of parts, thus limiting scalability.

Another family of approaches is cascaded classifiers [2, 9, 14, 28, 32], which performs cheap classifiers to reject unlikely part hypotheses early. By cleverly adjusting the cascade

procedure, one can achieve substantial speedup with accuracy guarantees [14, 28]. Yet most cascades treat each part independently, hence the cost scales linearly in the number of parts. A notable exception is [12] which performs cascades on a hierarchy of part detectors that is given a priori.

Our algorithm belongs to the group of methods that learns a hierarchical partitioning [1, 8, 15, 22, 31] of part detectors. The hierarchy gives us logarithmic growth in computational cost. While most methods learn the hierarchy in a greedy, layer-wise fashion (e.g. [1, 15, 22]) our algorithm finds the globally optimal structure within a reasonable search space.

Numerical methods [7, 10, 29] have successfully accelerated the evaluation of linear filters, but they provide little accuracy guarantee. Alternatively, effective search methods have been proposed to avoid exhaustive sliding windows [17, 20, 23, 24, 26], which can be combined with ours to provide further speedups.

3 Cascade Hierarchy

Here we describe the general concept of cascade hierarchy for speeding up part-based detectors. We will adapt it to poselets in Sec. 5.

Consider a scanning window detector that evaluates part classifiers at each window and collects **positive examples**, i.e. features computed from a scanning windows with positive scores, for further processing. The goal of a cascade hierarchy is to recover as many positive examples with as few evaluations as possible.

A cascade hierarchy (Fig. 2a) is a tree where each node represents a classifier designed to distinguish between examples of a set of parts $\mathcal{P}$ and those of the background class. $\mathcal{P}$ is further partitioned into $K_{\mathcal{P}}$ subsets and handled by each of the $K_{\mathcal{P}}$ child nodes. We use $\mathcal{P}$ to refer to both the set of parts and the node, whenever the context is clear. Any example that fails the classifier for $\mathcal{P}$ is filtered out from further consideration. The examples that pass are evaluated by all of $\mathcal{P}$'s children. All examples that reach a leaf are evaluated by the traditional part-classifiers for all parts at that leaf node.

The following performance metrics are of interest. For a node $\mathcal{P}$, **retention rate** $R_{\mathcal{P}}$ is the fraction of all examples that pass through, and **detection rate** $d_{\mathcal{P}}$ is the fraction of positive examples for set $\mathcal{P}$ that pass through:

$$R_{\mathcal{P}} \triangleq \frac{\#\text{Examples go through } \mathcal{P}}{\#\text{Examples enter } \mathcal{P}}, \qquad d_{\mathcal{P}} \triangleq \frac{\#\text{Positives go through } \mathcal{P}}{\#\text{Positives enter } \mathcal{P}} \tag{1}$$

Since the vast majority of examples are negatives, the speed of a single node is inversely proportional to the retention rate. The speed of a cascade hierarchy can be characterized using the expected number of examples classified at each node plus the cost of running the traditional part classifiers on the retained examples (Sec. 4). In addition we can measure accuracy using the detection rate: at any node $\mathcal{P}$, since the node filters some examples and passes the rest to the children nodes $\mathcal{P}'$, the overall detection rate $d_{\mathcal{T}(\mathcal{P})}$ of the tree rooted at $\mathcal{P}$ must be the product of the $\mathcal{P}$'s detection rate $d_{\mathcal{P}}$ and the expected detection rate of the children subtrees $d_{\mathcal{T}(\mathcal{P}')}$: $d_{\mathcal{T}(\mathcal{P})} = d_{\mathcal{P}}\mathbb{E}[d_{\mathcal{T}(\mathcal{P}')}]$. At last, once a classifier is trained for a node $\mathcal{P}$, we can adjust its threshold to trade-off the retention rate $R_{\mathcal{P}}$ and the detection rate $d_{\mathcal{P}}$ for node $\mathcal{P}$: $R_{\mathcal{P}}$ is a non-decreasing function of $d_{\mathcal{P}}$.

The accuracy and speed can be affected by three key factors: (1) the tree structure $\mathcal{T}$ that describes the partitioning of parts, (2) the classifiers at individual nodes, and (3) the desired detection rate of each node. These factors must be orchestrated to maintain a desired overall

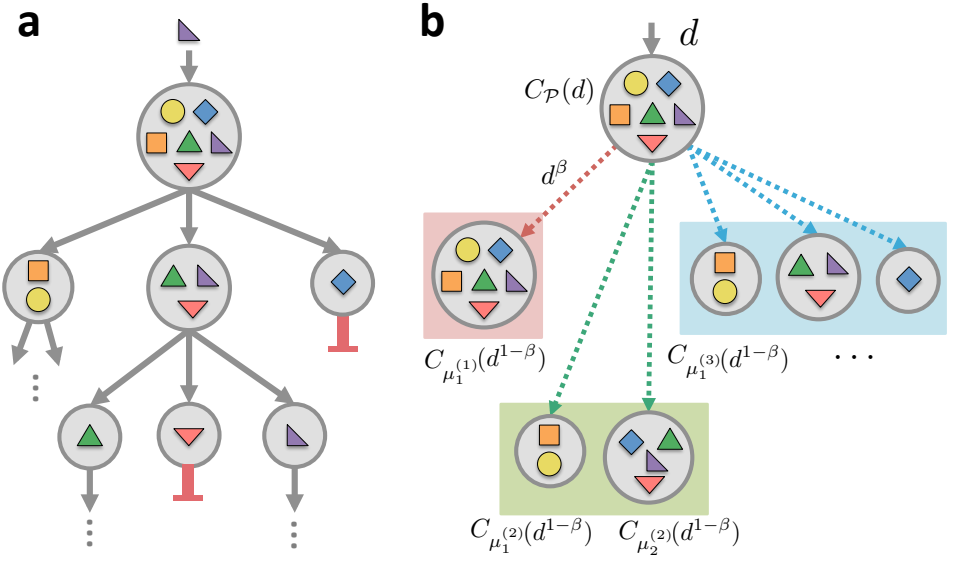

Figure 2: Cascade hierarchy. **(a)** Inference: each node is a classifier trained to let through examples of a set of parts $\mathcal{P}$ (represented as shapes within the node). The classifiers are trained to filter out background examples, but are lenient in letting through parts outside of $\mathcal{P}$. Thus an example (purple triangle) may go through multiple nodes on the same layer (e.g. first two nodes on the middle layer). Through paths are indicated with arrows, and blocked path an upside-down T. **(b)** Learning with target detection rate d: each node $\mathcal{P}$ train a classifier to guarantee a detection rate of $d_{\mathcal{P}} = d^{\beta}$. $\mathcal{P}$ incurs an evaluation cost $C_{\mathcal{P}}(d)$, and imposes a new target detection rate $d^{1-\beta}$ on each children. There are many ways to split $\mathcal{P}$ (listed in three colored boxes), and the split that yields the lowest cost (Eq. 2) should be chosen as the subtree of $\mathcal{P}$. Computing the exact cost could be done recursively (Eq. 3 and 4) until a leaf is reached, but would require exhaustively exploring all possible tree structures. To efficiently find the globally optimal hierarchy, we use branch-and-bound (Algo. 1) that prunes unlikely splits by estimating upper and lower bounds on the costs.

detection rate. For example, having more child nodes means paying more cost upfront for evaluating their classifiers, but each child will handle fewer parts and thus their classifiers are stronger and more efficient. Moreover, the classifier at individual nodes must be efficient to evaluate and yet effective to distinguish between parts and the background. Finally, a node $\mathcal{P}$ with a low local detection rate $d_{\mathcal{P}}$ can drastically filter out examples for its children and potentially make them faster, but to achieve the overall target detection rate $d_{\mathcal{T}(\mathcal{P})}$ it must also ask its child classifiers to be more conservative and slower.

We propose an algorithm that explores these three factors by considering a discrete set of possible tradeoffs at each node. Our parameterization is general and allows for building families of tree classifiers with desired properties. Specifically, if we set $K_{\mathcal{P}} = |\mathcal{P}|$ for the root and $K_{\mathcal{P}} = 1$ for the rest of the tree, we recover the traditional cascade. If we constrain the retention rate $R_{\mathcal{P}}$ at any node $\mathcal{P}$ such that $\mathbb{E}[R_{\mathcal{P}} K_{\mathcal{P}}] \leq 1$, then the number of evaluations at each level of our tree will not increase, which results in a family of trees whose evaluation time grows logarithmically on the number of parts.

4 Algorithm

We describe the algorithm for learning a cascade hierarchy to achieve the best speedup while meeting the performance requirements.

Performance constraint. Our cascade hierarchy must guarantee a specific detection rate d. One way to distribute the detection rate among nodes is to enforce that $d_{\mathcal{P}}$, detection rate for $\mathcal{P}$ is no less than d^{β} and $d_{\mathcal{T}(\mathcal{P}')}$ for the child nodes $\mathcal{P}'$ no less than $d^{1-\beta}$ for $\beta \in (0,1]$. Therefore $d_{\mathcal{T}(\mathcal{P})} = d_{\mathcal{P}}\mathbb{E}[d_{\mathcal{T}(\mathcal{P}')}] \geq d$.

Cost function. We would like to minimize the cost function $C_{\mathcal{P}}(d)$, the minimum expected evaluation cost of the subtree responsible for classifying the set of parts $\mathcal{P}$ at the target detection rate d.

Evaluation of the cost function requires considering how $\mathcal{P}$ is partitioned and assigned to the child nodes. Let μ denote one way in which $\mathcal{P}$ may be partitioned into distinct subsets, and let μ_j denote the j-th set in μ. For example, if parts were numbers, and $\mathcal{P} = \{1,3,6,7,12\}$, then a possible partition μ would be $\{\{1,12\},\{3,6\},\{7\}\}$ and $\mu_2 = \{3,6\}$.

The cost function is defined as:

$$C_{\mathcal{P}}(d) = 1 + \min\left(C^B(d), C^E(d)\right) \tag{2}$$

$$C^B(d) \triangleq R_{\mathcal{P}}(d)B|\mathcal{P}| \tag{3}$$

$$C^E(d) \triangleq \min_{\mu,\beta\in[0,1)} R_{\mathcal{P}}(d^{\beta}) \sum_j C_{\mu_j}(d^{1-\beta}) \tag{4}$$

where the cost function includes the cost of evaluating the current node classifier (which we consider one unit cost) and the minimum cost between two choices: *brute-force evaluation*, which incurs a cost of $C^B(d)$, and *node expansion*, which costs $C^E(d)$[1].

In case of brute-force evaluation the classifier throws away as many examples as possible before the detection rate drops below d, and fully evaluates all retained windows using all the $|\mathcal{P}|$ part classifiers. Here B is the average cost of fully evaluating a part classifier.

The other option is to expand $\mathcal{P}$ to have a subtree of children. We need to pick the optimal number of child nodes and a partitioning of the parts into those nodes, both encoded by μ. We also need to pick the optimal tradeoff β between using up some of the allowed detection rate on our own classifier, vs passing it down to the children[2] (see Fig. 2b).

Reducing the state space. The space of all possible trees defined in the previous section is intractably large. To reduce the state space we make the following assumptions:

- For distributing detection rate among a node and its children, we use the same β parameter for all nodes. β is learned using cross-validation.
- We restrict the node classifiers to be linear SVMs. The classifiers use only a small subset of the features from an example and are fast to evaluate (see Sec. 5).
- We restrict the set of possible partitions for μ. Specifically, we use a clustering algorithm that ensures that similar parts fall in the same cluster (see Sec. 5 for similarity measure and the clustering algorithm). The cluster sizes are balanced.
- We restrict the maximum number of children to 4.

These restrictions reduce the search space to manageable size. At each node the possible number of child nodes is bounded and small. Furthermore, the depth of the hierarchy is bounded, as we will see later.

[1] Both costs depend on $\mathcal{P}$ and other variables such as μ and β. These variables are omitted for notational clarity.

[2] For simplicity here we assume that the detection rate is evenly distributed among all children

Minimizing the cost function. To minimizes the cost function $C_{\mathcal{P}}(d)$ for a desired detection rate d, we use branch-and-bound to search for the optimal hierarchy. In addition, since each state in the branch-and-bound algorithm corresponds to a full hierarchy, we use dynamic programming to efficiently compute its cost. The algorithm (Alg. 1) is detailed below.

Algorithm 1 for finding $C_{\mathcal{P}}(d)$

▷ Find the min-cost tree for target detection rate d

function FINDOPTIMALTREE(d,β)
  $IsOptimal(root) \leftarrow false$
  ImproveEstimate($root, d, \infty, \beta$)
end function

▷ Increase $\hat{C}_{\mathcal{P}}$ until it exceeds C^{target} or the subtree at $\mathcal{P}$ is optimal

function IMPROVEESTIMATE($\mathcal{P}, d, C^{target}, \beta$)
  if called for the first time for node $\mathcal{P}$ **then**
    InitializeTree($\mathcal{P}, d, \beta$)
  end if
  $C^B \leftarrow R_{\mathcal{P}}(d)B|\mathcal{P}|$ ▷ Brute-force cost, Eq. 3
  while (true) **do**
    $\mu^* \leftarrow \arg\min_\mu \sum_j \hat{C}_{\mu_j}$ ▷ μ^* is the currently best split of the children of $\mathcal{P}$
    $\hat{C}^E \leftarrow R_{\mathcal{P}}(d^\beta)\sum_j \hat{C}_{\mu_j^*}$ ▷ Lower bounds of expand cost, Eq. 4
    if $C^B < \hat{C}^E$ **then**
      $\hat{C}_{\mathcal{P}} \leftarrow 1 + C^B$ ▷ Brute-force is better
      $IsOptimal(\mathcal{P}) \leftarrow true$
      $children(\mathcal{P}) \leftarrow \emptyset$
      return
    end if
    $\hat{C}_{\mathcal{P}} \leftarrow 1 + \hat{C}^E$ ▷ Expand is better (so far)
    if $\forall j, IsOptimal(\mu_j^*)$ **then**
      $IsOptimal(\mathcal{P}) \leftarrow true$
      $children(\mathcal{P}) \leftarrow \mu^*$
      return ▷ Fully explored $\mathcal{P}$
    end if
    if $\hat{C}_{\mathcal{P}} > C^{target}$ **then**
      return ▷ Met target detection rate
    end if
    $\mu_j^*, \mu_k^* \leftarrow$ the best and second-best child of μ^*
    ▷ Work on μ_j^* until no longer the best
    ImproveEstimate($\mu_j^*, d^{1-\beta}, \hat{C}_{\mu_k^*}, \beta$)
  end while
end function

function INITIALIZETREE($\mathcal{P}, d, \beta$)
  for μ, j **do**
    $\hat{C}_{\mu_j} \leftarrow 1$
    $IsOptimal(\mu_j) \leftarrow false$
  end for
  train SVM for $\mathcal{P}$ and compute $R_{\mathcal{P}}(d)$ and $R_{\mathcal{P}}(d^\beta)$
end function

At each node $\mathcal{P}$ we maintain a lower (optimistic) bound of the cost, $\hat{C}_{\mathcal{P}}$, which we initialize to 1, as each node must at the very least run its classifier. We also maintain an upper bound, which is the cost of brute-force evaluation C^B (Eq. 3). The core algorithm *ImproveEstimate* repeatedly improves the lower bound of the best subtree $\hat{C}_{\mu_j}$ until some other subtree $\hat{C}_{\mu_k}$ becomes more promising or the lower and upper bounds meet. Once the two bounds meet, the subtree $\hat{C}_{\mu_j}$ becomes "optimal" as any other partition is guaranteed to be no cheaper than brute-force evaluation. A subtree is also optimal when all of its possible partitions μ become optimal. Finally, the algorithm completes when the root node becomes optimal.

The algorithm requires repeatedly updating the cost estimate $\hat{C}_{\mathcal{P}}$ using Eq. 2 and 4. To avoid redundant computation, we cache the classifier and cost estimate at each node so that both the retention rate $R_{\mathcal{P}}$ and the C_{μ_j}'s in Eq. 2 can be evaluated by a simple lookup.

Analysis. Algorithm 1 converges and returns the optimal cascade hierarchy within the restricted state space. Optimality of our algorithm follows from the optimality of the branch-and-bound algorithm. Convergence of our algorithm is straightforward if the maximum depth among all cascade hierarchies is bounded, since the search space would also be bounded. The depth is bounded because every new layer either reduces the number of parts $|\mathcal{P}|$ by at least half, or raises the desired detection rate by a power of $1-\beta$. Each part classifier has a maximum detection rate d^* such that brute force evaluation is cheaper than a traditional cascade. Therefore the total depth is bounded by $O(\log|\mathcal{P}|\frac{\log(d^*)}{(1-\beta)\log d})$.

5 Cascade Hierarchy for Poselets

In this section we adapt the cascade hierarchy to accelerate the poselet detector.

Stripe classifiers The classifier at each node is a linear SVM with local features called "stripes". We split the classification window of an example into a series of horizontal stripes, and classify only based on HOG features from a single stripe. We pick the stripe that best distinguishes the set of poselet types from the background (see Fig. 5). Our choice is driven by implementation efficiency, as the horizontal stripes are consecutive in memory in the HOG feature space. Our model is general and can work with any other features. A systematic study of the effect of feature choices is interesting but outside of the scope of this paper.

Clustering of similar poselets Recall that one of our simplifying assumptions to ensure tractability in learning is to restrict the set of partitions μ to only one choice for a given number of children $K_{\mathcal{P}}$. We provide this partitioning by clustering the poselet types in $\mathcal{P}$ into $K_{\mathcal{P}}$ clusters. A good partitioning should cluster similar poselets together so that they can be easily separated from the background class. We cluster poselet types based on the Mahalanobis distance between their filters with respect to the best stripe they have in common.

We generate $K_{\mathcal{P}}$ clusters of equal size by repeatedly picking the best cluster of size $|\mathcal{P}|/K_{\mathcal{P}}$ along with the best stripe associated with the cluster. Fig. 5 shows the result of this clustering algorithm applied to a subtree.

6 Results and Discussion

We conduct experiments to examine our algorithm's ability to adapt to different accuracy requirements, to trade off efficiency with accuracy and to scale to large number of poselets.

6.1 Exploring the effect of detection rate and β

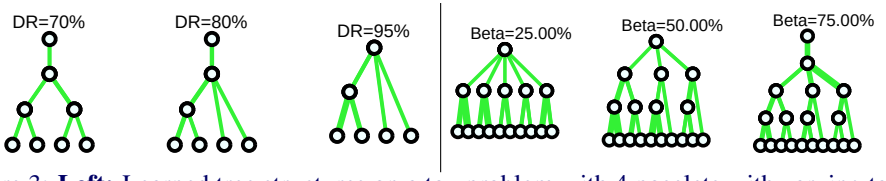

Figure 3: **Left:** Learned tree structures on a toy problem with 4 poselets with varying target detection rate. The depth of the tree shrinks monotonically as target detection rate grows from 70% to 95%. **Right:** Learned tree structures on a toy problem with 10 poselets with varying β for trading off current vs future detection rate. The depth of the tree grows as β grows from 25% to 75%.

Fig. 3-left shows the resulting tree of the toy problem of classifying four poselets as we vary the target detection rate. For the high accuracy rates our algorithm prefers to do multi-way splits and use the maximum number of splits. However, if it is allowed to have lower detection rate it determines that a single cascade step is best as it can filter out quickly a large number of the examples.

Fig. 3-right shows the resulting tree of the toy problem of classifying 10 poselets as we vary β, the tradeoff parameter between current and future detection rate. High β means the current node is myopic and can afford to evaluate a cascade. However when β is low, we need to maintain high detection rates ourselves and, if necessary, brute force evaluate everything, which results in a shallow tree.

6.2 AP as a function of speed

Fig. 4-left shows the average precision of our hierarchical cascade (denoted *MaxK*=4 since each node can have at most 4 children) as a function of evaluation speed as measured on the person category of PASCAL 2007 test set. We compare against the following baselines: *MaxK*=*k*: a cascade hierarchy where each node has at most *k* children. *Downsampling*: a C++ version of a traditional poselet classifier and adjusted the scanning step in X,Y and scale to achieve the same number of computational units (in our case the cost to compute the inner product along one horizontal stripe is one unit); *Cascades*: cascades trained independently on individual poselets using the same stripe classifiers; *Randcluster*: a cascade hierarchy where the partitions at each node are assigned randomly but the tree structure is optimized.

As the figure shows, we can achieve more than **10x** speedup at the cost of decreasing AP by just a single point, whereas simple adjustments to the sampling rate significantly reduce the accuracy. Grouping similar poselets together is congenial to trading off accuracy and speed, but of greater importance is having the optimal tree structure (see performance improvement from $MaxK = 2$, a binary tree, to $MaxK = 4$, a tree with a maximum width of 4). Traditional cascades perform well in high accuracy low speedup scenarios, but without sharing its performance does not translate to high speedup cases. Finally, our model starts to show signs of over-fitting when the speedup demand is too high (over 1500%). In such scenarios, random clustering may provide some robustness against prematurely killing an entire group of poselets with tightly clustered appearance.

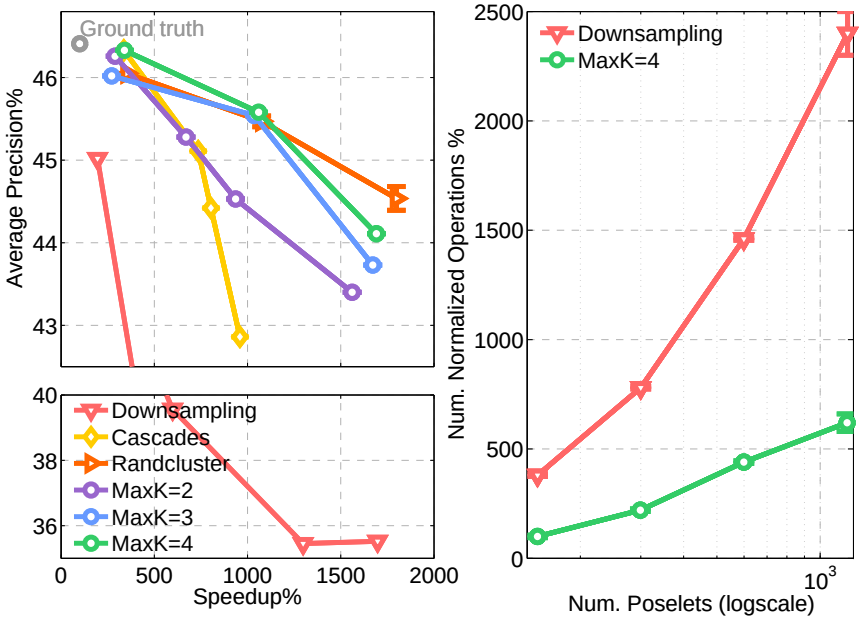

Figure 4: **Left:** Average precision of our classifier (*MaxK*=4) on the PASCAL 2007 set for the Person category as a function of evaluation speed. We compare against the AP of *Downsampling*: standard poselet detector with coarser sampling in space and scale; *Cascades*: independent cascades for each individual poselet; *Randcluster*: cascade hierarchy with random partition and *MaxK*=*k*: cascade hierarchy where each node can have at most *k* children. **Right:** Computation time for the same detection rate as a function of the number of poselets.

6.3 Speed as a function of the number of poselets

Fig. 4-right shows how our computation time scales as we increase the number of poselets but keep the same target detection rate. We tested with 150, 300, 600 and 1200 poselets [4]. As expected the traditional poselet classifier scales linearly with the number of poselets, but due to feature sharing our hierarchical cascade has logarithmic growth curve.

6.4 Performance in seconds

We timed our implementation in C++ of the baseline and one of our models that achieves 45.5AP on PASCAL person category and whose speedup (as determined by the number of operations) is 1145%. The numbers are in seconds per image on a single-threaded implementation running on Intel Xeon CPU at 2.67Ghz. The table below shows that the actual speedup time of around 1035% is in the same ballpark as estimated by the processing unit count. However, we are, of course bound by Amdahl's law as the overall total speed of detection depends on other factors (such as feature extraction) which do not improve. Note, however, that feature extraction time is amortized as we increase the number of categories

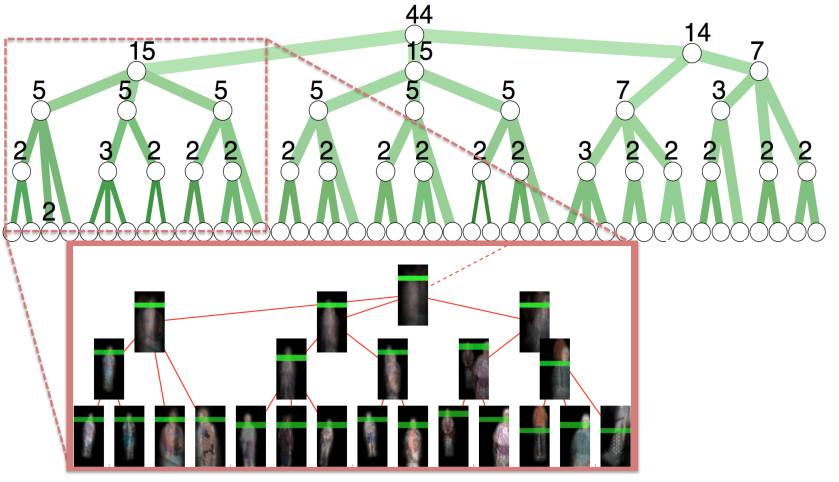

Figure 5: A classification tree generated by our algorithm for classifying 44 poselets at 90% target detection rate. The thickness of the edges denotes the retention rate of the classifiers. The number of poselet types classified by each node is indicated. **Left corner:** A zoom on part of the tree. At each node we show the average mask over all classifiers captured by the node, along with the horizontal stripe that was used to classify the node.

and poselets as they reuse the same HOG features. There are a number of engineering improvements that can be done to further improve the overall performance, such as a multi-threading it and using more efficient HOG feature generation. These are beyond the scope of this paper.

Module	Baseline	@45.5AP
Poselet Detection	11.7s	1.13s
Feature Extraction	2.0s	2.0s
Total Detection	13.8s	3.2s

7 Conclusion

In this paper we develop a technique for efficient joint evaluation of multiple part-based detectors. Our method is general as it does not depend on the specific choices of classifiers, features or parts. We provide a branch-and-bound algorithm that determines the most optimal tree (in our restricted search space) that satisfies a given detection rate.

We applied our algorithm on the task of improving the efficiency of poselet detection and demonstrated more than **an order of magnitude speedup** for a small controlled loss of accuracy, and a **logarithmic** runtime complexity with regard to the number of poselets. Taken to the extreme, variations of our method could be used to efficiently evaluate (albeit at a low detection rate) many thousands of part-based classifiers. This could lead to a largely unexplored research area in visual object recognition which could be interesting due to the heavy-tail nature of the patterns associated with visual object categories.

References

[1] Gilles Blanchard and Donald Geman. Hierarchical testing designs for pattern recognition. *The Annals of Statistics*, 2005.

[2] Lubomir Bourdev and Jonathan Brandt. Robust Object Detection via Soft Cascade. In *CVPR*, 2005.

[3] Lubomir Bourdev and Jitendra Malik. Poselets: Body part detectors trained using 3D human pose annotations. In *ICCV*, 2009.

[4] Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Describing people: A poselet-based approach to attribute classification. In *ICCV*, 2011.

[5] Lubomir Bourdev, Fei Yang, and Rob Fergus. Deep poselets for human detection. *arXiv preprint arXiv:1407.0717*, 2014.

[6] Xiaoyong Chai, Lin Deng, Qiang Yang, and Charles X Ling. Test-Cost Sensitive Naive Bayes Classification. In *ICDM*, 2004.

[7] Thomas Dean, Mark Ruzon, Mark Segal, Jonathon Shlens, Sudheendra Vijayanarasimhan, and Jay Yagnik. Fast , Accurate Detection of 100 , 000 Object Classes on a Single Machine. In *CVPR*, 2013.

[8] Jia Deng, Satheesh Sanjeev, Berg Alexander C., and Fei-Fei Li. Fast and Balanced: Efficient Label Tree Learning for Large Scale Object Recognition. In *NIPS*, 2011.

[9] P Dollár, R Appel, and W Kienzle. Crosstalk cascades for frame-rate pedestrian detection. In *ECCV*, 2012.

[10] Charles Dubout and F Fleuret. Exact acceleration of linear object detectors. In *ECCV*, 2012.

[11] Mark Everingham, Luc Van Gool, Chris K. I. Williams, John Winn, and Andrew Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results, 2007.

[12] Pedro F. Felzenszwalb, Ross B. Girshick, and David McAllester. Cascade object detection with deformable part models. In *CVPR*, 2010.

[13] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *PAMI*, 2010.

[14] Pascal Fua, Carlos Joaquin Becker, François Fleuret, Raphael Sznitman, et al. Fast object detection with entropy-driven evaluation. In *CVPR*, 2013.

[15] S. Gangaputra and D. Geman. A Design Principle for Coarse-to-Fine Classification. In *CVPR*, 2006.

[16] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[17] Chunhui Gu, Pablo Arbeláez, Yuanqing Lin, Kai Yu, and Jitendra Malik. Multi-component models for object detection. In *ECCV*, 2012.

[18] Sergey Karayev, Tobias Baumgartner, Mario Fritz, and Trevor Darrell. Timely object recognition. In *NIPS*, 2012.

[19] Daphne Koller and Tianshi Gao. Active Classification based on Value of Classifier. In *NIPS*, 2011.

[20] Christoph H Lampert, Matthew B Blaschko, and Thomas Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *CVPR*. IEEE, 2008.

[21] Subhransu Maji, Lubomir Bourdev, and Jitendra Malik. Action recognition from a distributed representation of pose and appearance. In *CVPR*, 2011.

[22] Marcin Marszałek and Cordelia Schmid. Constructing category hierarchies for visual recognition. *ECCV*, 2008.

[23] Pierre Moreels and Pietro Perona. A Probabilistic Cascade of Detectors for Individual Object Recognition. In *ECCV*, 2008.

[24] Marco Pedersoli, Andrea Vedaldi, and Jordi Gonzalez. A Coarse-to-fine approach for fast deformable object detection. In *CVPR*, 2011.

[25] John C Platt, Nello Cristianini, and John Shawe-Taylor. Large margin dags for multiclass classification. In *NIPS*, 1999.

[26] Esa Rahtu, Juho Kannala, and Matthew Blaschko. Learning a category independent object detection cascade. In *ICCV*, 2011.

[27] Michalis Raptis and Leonid Sigal. Poselet key-framing: A model for human activity recognition. In *CVPR*, 2013.

[28] Mohammad Javad Saberian and Nuno Vasconcelos. Learning optimal embedded cascades. *PAMI*, 2012.

[29] Mohammad Amin Sadeghi and David Forsyth. Fast template evaluation with vector quantization. In *NIPS*, 2013.

[30] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[31] Zhuowen Tu. Probabilistic boosting-tree: Learning discriminative models for classification, recognition, and clustering. In *ICCV*, 2005.

[32] Paul. Viola and Michael. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.

[33] Yang Wang, Duan Tran, and Zicheng Liao. Learning hierarchical poselets for human parsing. In *CVPR*, 2011.

[34] Bangpeng Yao, Xiaoye Jiang, Aditya Khosla, Andy Lai Lin, Leonidas J. Guibas, and Li Fei-Fei. Action recognition by learning bases of action attributes and parts. In *ICCV*, 2011.

[35] N Zhang, Ryan Farrell, and Trever Darrell. Pose Pooling Kernels for Sub-category Recognition. In *CVPR*, 2012.

[36] Ning Zhang, Manohar Paluri, Marc'Aurelio Ranzato, Trevor Darrell, and Lubomir Bourdev. PANDA: Pose Aligned Networks for Deep Attribute Modeling. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.