30Hz Object Detection with DPM V5

Mohammad Amin Sadeghi, David Forsyth

Computer Science Department, University of Illinois at Urbana-Champaign
{msadegh2,daf}@illinois.edu

Abstract. We describe an implementation of the Deformable Parts Model [1] that operates in a user-defined time-frame. Our implementation uses a variety of mechanism to trade-off speed against accuracy. Our implementation can detect all 20 PASCAL 2007 objects simultaneously at 30Hz with an mAP of 0.26. At 15Hz, its mAP is 0.30; and at 100Hz, its mAP is 0.16. By comparison the reference implementation of [1] runs at 0.07Hz and mAP of 0.33 and a fast GPU implementation runs at 1Hz. Our technique is over an order of magnitude faster than the previous fastest DPM implementation. Our implementation exploits a series of important speedup mechanisms. We use the cascade framework of [3] and the vector quantization technique of [2]. To speed up feature computation, we compute HOG features at few scales, and apply many interpolated templates. A hierarchical vector quantization method is used to compress HOG features for fast template evaluation. An object proposal step uses hash-table methods to identify locations where evaluating templates would be most useful; these locations are inserted into a priority queue, and processed in a detection phase. Both proposal and detection phases have an any-time property. Our method applies to legacy templates, and no retraining is required.

Keywords: Fast Object Detection, Real-time Object Detection, Fast Deformable Parts Model.

1 Introduction

A major burden in using object detectors in practice is speed. Except for certain objects including face, pedestrians and certain rigid objects, detectors do not currently run at video rate. We employ a series of techniques to detect several objects together at video rate. The architecture we present in this paper can detect the 20 PASCAL VOC categories simultaneously at 30Hz.

We focus on speeding up DPM [1] because it is a mature and stable technology. While other detection methods are more accurate, the full potential of these technologies has not yet been explored, and they will not take their final form for some time. We believe that our speed-up techniques exploit fundamental properties of templates and will apply to deep leaning methods.

We build up our detector based on Deformable Parts Model [1] and compare to its latest implementation [21]. The latest implementation detects 20 PASCAL VOC object categories in about 13 seconds per image from the PASCAL dataset. Several techniques have been developed to speed up DPMs [3][11][2].

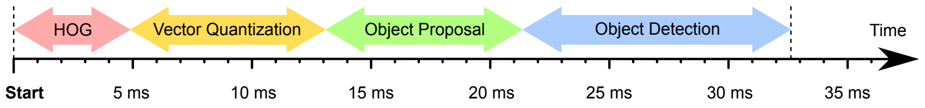

Fig. 1. Our fast implementation of Deformable Parts Model can jointly detect 20 PASCAL categories at 30fps or faster. The pipeline consists of four steps that together run at video rate speed. To achieve this speed we used optimized techniques for each step. Optimizations for HOG feature computation are discussed in Section 2; Fast Vector Quantization is discussed in Section 3; The object proposal technique is discussed in Section 4; and object scoring is discussed in Section 5. For details about the exact computation time of our implementation please refer to Section 6. Allocation of time between the proposal and the detection phase can be balanced according to the processor architecture, dataset properties and application requirements; time allocation is discussed in Section 7.

These techniques can speed up computation time to about 0.5 seconds (2Hz) with almost no loss of accuracy. Our techniques obtains a further speed-up to 30Hz with a minor loss of accuracy. Furthermore, our technique allow an explicit trade-off of accuracy for speed.

Furthermore, our technique can maintain a fixed frame rate at 30.0Hz that is essential in practical applications. Most speed-up techniques optimize average speed and cannot guarantee a fixed time per frame.

Our technique consists of four major steps that are illustrated in Figure 1. Given a query image, we first extract a lightweight version of HOG features from the image (details in Section 2). We then vector quantize HOG features according to Section 3. We use a data-structure to obtain object proposal to identify the promising locations (Section 4). We finally score the proposals by evaluating the corresponding templates (Section 5).

We employ separate optimization techniques to speed up each of the four main stages. We implemented a highly optimized code to extract HOG features very quickly. Our implementation utilize various low-level optimizations including vector operations, multiple cores and CPU cache management. We also use a lightweight version of the HOG pyramid that further speeds up the process. After the HOG features are computed we use a hierarchical clustering process to vector quantize the HOG features (Section 3).

We use a data-structure to provide cheap object-dependent proposals in Section 4. Our proposal stage uses a hashing scheme that allows us to process only a small fraction of templates at each location. Our object scoring stage (Section 5) can also operate in a user-defined time-frame. It processes as many locations as it can within the specified time-frame.

Our implementation is fast and light-weight. The code is implemented in C++ but can be called from MATLAB. Our implementation will be available online for public use upon paper acceptance. Our algorithm not only can process an image to detect 20 pascal categories simultaneously at 30fps, it can further trade off accuracy for time to achieve a detection rate of 100Hz.

1.1 Prior work

There is a rich literature of fast object detection built up on the original Deformable Parts Model [1] algorithm. Several successful speed-up techniques have been introduced in the last few years.

Cascades speed up evaluation by using rough tests to identify promising locations to further process using fine tests. For example, Felzenszwalb et al. [3] evaluate root models, and then evaluate the part scores only in promising locations. At each iteration their method evaluate the corresponding template only if the current score of the object is higher than a certain threshold. Sadeghi et al. [2] follow a similar approach but they use a fast vector quantization technique that is compatible with cascades to further boost the speed. Pedersoli et al. [12] estimate the score of a location using a lower resolution version of root templates and use higher resolution templates in high-scoring locations. Dollár et al. [10] enable neighbouring locations to communicate when a template is being evaluated. Cascade approach to object detection has been shown to be very successful for speed-ups.

Transform Methods evaluate templates at all locations simultaneously by exploiting properties of the Fast Fourier Transform. The advantage of these methods, pioneered by Dubout et al. [11] is that the computation is fast and exact at the same time. In comparison, most other techniques involve approximation. The disadvantage of this approach is that it is not *random-access*; a large chunk of the locations are processed in one pass making the algorithm incompatible with cascade techniques.

Hash Tables exploit locality sensitive hashing [15] to get a system that can detect many thousands of object categories in a matter of seconds [14]. This strategy appears effective and achieves a good speed-up with very large numbers of categories. Dean et al. [14] use a hash table at the core of their technique that allows them to spend computation for only the high-scoring locations. The advantage of this technique compared to cascades is that they don't require any computation for low chance locations whereas cascade algorithms examine every location at least once.

Vector Quantization is well-studied for data compression [18]. In the past few years several algorithms have used vector quantization to speed up computation. These techniques operate in situations where arithmetic accuracy is not crucial. Jégou et al. [17] successfully apply vector quantization to approximate nearest neighbour search. Kokkinos [13], Vedaldi et al. [16] and Sadeghi et al. [2] apply different variations of this approach to object detection and demonstrate significant speed-ups.

Hierarchical Classification techniques run detectors in a tree structure with a depth of $\Theta(\log C)$ to be able to cover C categories. Nistér et al. [8] clusters categories using hierarchical k-means. Bengio et al. [24] use detectors that are suitable to discriminate between groups of categories. Both techniques are scalable in terms of C.

Object Proposals are used in object detection techniques that need to avoid a dense sliding window search. Some object proposal algorithms produce category-independent proposals (e.g. Endres et al. [6] and Cheng et al. [7]) while others [14] provide category-dependent proposals. The main source of speed-up in these techniques is that they significantly limit the number of locations to evaluate detectors. Category-dependent proposals are preferred in speed-up applications as they need to be evaluated by fewer detectors.

GPU Implementation can be used to speed up object detectors as well. Vanilla DPM [22] is a version of DPM that can harness the power of GPU to speed-up object detectors.

These techniques have improved the object detection speed so much that the feature computation stage has became a major bottleneck. Dollár et al. [4] present elegant techniques to speed-up features computation. We use a version of [4] to speed up our feature computation (Section 2).

2 Pyramid of Features vs. Pyramid of Templates

Conventional object detectors operate at various scales to be able to detect objects with variable sizes. Template based object detectors extract local features at various scales (e.g. HOG pyramid [5] and histogram of sparse codes [19]) and evaluate a given template at all scales. In practice ten scales per octave is typical.

Feature computation is a major bottleneck for pedestrian detection. Dollár et al. [4] present an elegant technique to process features for certain key scales (one or two per octave) and interpolate for the rest of scales. Their experiments with pedestrian detection show that this leads to a significant speed-up for feature computation. Benenson et al. [25] interpolate templates for integral channel features. Our approach is similar to Dollár et al. [4]; however, instead of rescaling features we rescale templates (Figure 2). We rescale each template to several scales in order to make a *Pyramid of Templates*. Our experiments show that this works as accurate as rescaling features while being faster in practice. The pyramid of templates has two major advantages over the regular pyramid of features:

1. Because HOG templates are several times smaller in size than HOG features (2K cells per object category vs. 100K cells per image) processing and storing HOG features takes much more time than templates. Furthermore, categories are often fixed for several images while every new image comes with a new feature. As a result, reducing the number of feature levels per octave directly limits the space required to store features. In our experiments HOG features are compressed from 8MB to 1.6MB per image. The benefits include more efficient caching and more efficient mobile application.

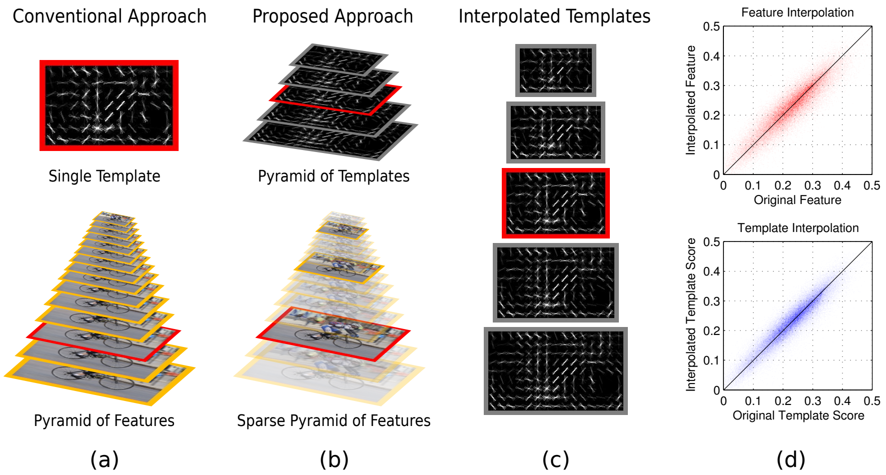

Fig. 2. a: Conventional object detectors run templates on a pyramid of features to capture a range of scales (ten scales per octave is typical). Dollár et al. [4] compute two scales per octave then interpolate the rest of the scales to considerably speed up feature computation at the cost of about 2% loss of average precision. **b**: Instead of interpolating features we interpolate templates. We show that interpolating templates is faster and leads to further speed-up techniques. **c**: We generate new templates by interpolating templates to different scales. **d**: This process introduces some error. The two scatter plots illustrate original template score versus the score produced by interpolated features/templates. **d**: **Top**: Features are interpolated according to [4]. **Bottom**: Templates are interpolated instead of features. Although interpolating templates is faster than interpolating feature pyramids, the errors are in the same range.

2. Several speed-up techniques are based on having a large number of templates (e.g. Pirsiavash et al. [9] and Dean et al. [14]). The computational complexity of [14] is claimed to be independent of C the number of templates. Their computational complexity depends only on L the number of locations to evaluate templates (whereas most algorithms have at least a $\Theta(CL)$ term in their complexity). Our technique uses $5C$ templates and $\frac{1}{5}L$ locations; therefore, it can directly benefit from speed-up techniques presented by Pirsiavash et al. [9] and Dean et al. [14].

A few technical issues arise when resizing templates for object detection. All part templates need to be resized as well as deformation costs and part locations. The interpolation method can affect the quality of the new template. In order to resize a HOG template we interpolate every layer separately. We compared bilinear and bicubic interpolations and bicubic interpolation appears to be the best. The interpolated weights are adjusted by a factor to maintain the mean and the standard deviation of the scores. Our experiments show that this optimization leads to an mAP loss of about 0.02, compatible to that of [4].

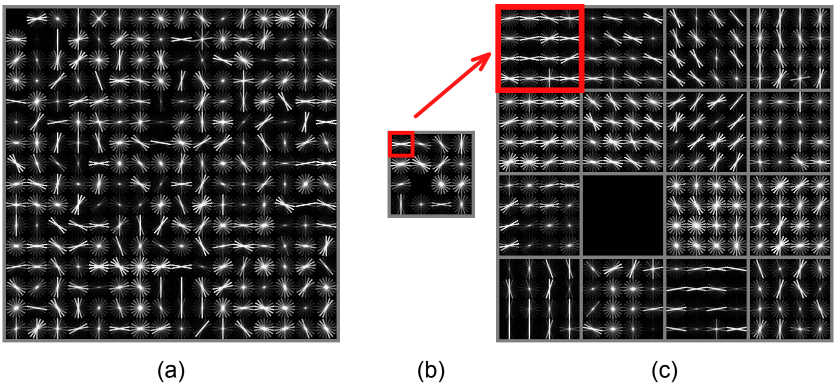

Fig. 3. a: The method proposed by Sadeghi and Forsyth [2] quantizes each cell into one of 256 pre-defined clusters. Nearest neighbour search is a significant bottleneck in their technique. In this paper we use hierarchical clustering instead of flat clustering. **b**: each cell is first quantized into one of the 16 clusters. **c**: Depending on the first level, the cell is clustered into one of 16 clusters in the respective group in c. Note that hierarchical clustering reduces the number of comparisons from 256 per cell to two stages of 16 comparisons per cell.

3 Hierarchical Vector Quantization

Several optimization techniques have been employed to speed up Deformable Parts Model object detectors. The fastest was proposed by Sadeghi and Forsyth [2]. This is nearly two orders of magnitude faster than the original implementation of [21]. The key to their success is a vector quantization technique that decreases the computation demand by a large factor. They vector quantize HOG features and compute template scores by indexing certain look-up tables and adding their scores.

We use vector quantization for the same purpose but with a slightly different approach. The main computation bottleneck in [2] is vector quantization. They need 70ms per image to quantize HOG features for one image. The high computational demand is due to the fact that each HOG cell needs to be compared against every one of 256 cluster centers. (Figure 3, a). We use a hierarchical clustering technique to speed up this process. We first cluster each cell into 16 clusters (Figure 3, b). Then according to the nearest cluster in the first step we compare against 16 other clusters to find the nearest cluster (Figure 3, c). We pre-compute clusters using k-means algorithm.

Our experiments show that the proposed hierarchical clustering technique leads to a negligible loss of 0.001 in mAP. In contrast, the speed-up gain is about 8-fold.

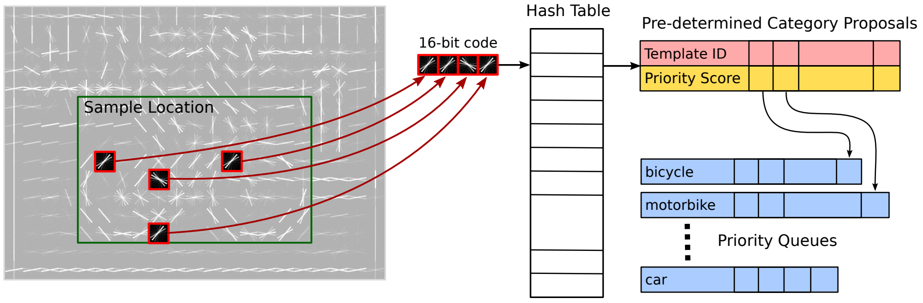

Fig. 4. Our proposal generation data-structure. We use a few look-up tables that are filled with pre-determined proposals. For each location we make a hash code by observing four pre-specified cells. We index the code into a hash table and obtain a list of pre-determined category proposals for each location. We store the proposals in category specific priority lists and later use them to evaluate the score of each location for each proposed category.

4 Object Proposal using Hash Table

We cannot evaluate all templates at all locations fast enough. Instead, we use a hashing technique to identify promising locations and insert them into priority queues (Figure 4). Proposals will then be processed in the object scoring stage (Section 5). This architecture means that both the proposal process and the template evaluation process can be terminated at any time allowing our method to operate at fixed frame rate and trade-off accuracy for speed.

The cascade framework applied to Deformable Parts Model first evaluates a rough version of a given root template and then evaluates the corresponding part scores and finally re-estimate the scores by using fine templates (e.g. Felzenszwalb et al. [3] and Sadeghi et al. [2]). Although cascade methods prune the majority of locations, they need to at least evaluate all root templates at all locations (they can prune part templates but not root templates). To process the 20 PASCAL categories this step takes about 400ms in [3] and about 90ms in [2]. The two techniques are both too slow for video rate speed.

Several algorithms are introduced to generate object proposals for object detection (e.g. Endres et al. [6] and Cheng et al. [7]). We use a proposal generation data-structure to limit template evaluation to a sparse set of proposals rather than dense sliding window search. Our data-structure uses a hash table similar to Dean et al. [14]. Our hash table is distinguished from [14] in three aspects:

1. Instead of Winner Takes All (WTA) hash we use a hashing scheme compatible to our hierarchical vector quantization (Figure 4).
2. The data-structure used by Dean et al. [14] proposes template ID's without proposed scores. Our data-structure provides a *priority score* with each

template ID. The priorities help us later choose which templates to process further.

3. The data-structure used by Dean et al. [14] uses hundreds or thousands of hash tables. We instead use 10 hash tables. The fact that we need fewer hash tables is partly due to our pre-stored priority scores.

4.1 Hash Codes

In order to generate proposals, we process all locations in an image (sliding window search) using our data-structure. Since different templates are different in size, we refer to each location using its top left co-ordinate (Figure 4). At each location we extract proposals using 10 separate hash tables.

Each hash table is indexed with a distinct 16-bit code. The 16-bit code is generated by observing four quantized cells and concatenating their corresponding quantization ID. We use a dictionary of 16 words (4-bits) for each cell that is equivalent to the first level of hierarchical quantization discussed in Section 3 (Figure 3). Reference cells are randomly determined for each hash table while initializing the data-structure.

Each cell of the hash tables is linked to a list of proposed templates and their corresponding priorities. A template can be determined by its category, root index and scale. For the PASCAL dataset and Deformable Parts Model version 5, there are 20 categories, 6 root templates per category and 5 scales (Figure 2): a total of 1200 root templates. We store 20 templates for each cell of the hash table. However, most of the proposals are not used in most cases.

4.2 Priority Lists

For each root template we store a separate priority list (Figure 4). Each list stores several proposal locations with their corresponding priority scores. The priority scores are used to determine the priorities between locations given a root template. Each root template has a limited budget of locations to examine that are chosen according to priority scores.

We use a simple array to store each priority list. After the lists are populated with proposals we keep a number of proposals that are expected to complete within the specified time-frame. We then evaluate root templates on remaining locations and update their priority score with the actual responses from the root templates.

The reason we store a separate priority list per template – as opposed to one joint priority list – is that the scores of different root templates are not directly comparable. Also the user may need to specify more process time on a certain template depending on the application. In our experiments we process equal number of locations per root template. Process allocation could be adjusted depending on the architecture of processor and the application. We discuss time allocation in more detail in Section 7.

4.3 Hash Table Initialization

We use 10 parallel hash tables each indexed with a separate 16-bit code. A hash code is generated by concatenating the quantization ID's of four cells. We randomly choose the cells in a 12×12 window and build the look up tables accordingly. For each possible hash code we compute a rough approximation score using the look-up tables used by Sadeghi and Forsyth [2]. We choose the 20 top categories according to the approximation scores. We perform a score adjustment process to make sure all templates are equally likely to be proposed.

We process the 10 hash tables in a sequence to balance proposals among all locations in case of early termination. If not enough time is available to go through all hash tables, the tables used will cover image locations fairly.

5 Object scoring

Our proposal generation process provides a separate priority queue for each template. Because the number of proposals is often more than what we afford to process, many proposals cannot be evaluated.

We use a version of Round-Robin algorithm to process priority queues corresponding to different templates. We process one location from each queue in a circular order, handling all queues with equal priority. As soon as one proposal from one queue is done we process an example from the next queue. We continue this process until time is out. Our algorithm is parallelized with OpenMP to harness the power of all processor cores. Each thread in our process is responsible for an equal number of templates. All threads stop when time is up and return their detections. The time required for Non-max suppression (NMS) is also negligible.

We follow the technique presented by Felzenszwalb et al. [3] to process each location. Given a proposal, we first approximate the score of the root template using FTVQ [2]. We then add the approximated score of the first part together with its deformation cost. We continue adding the score of all other parts in a sequence. After we evaluate a part score we may stop and reject the proposal according to a pre-trained threshold.

After computing the approximated scores using FTVQ, we replace the approximated score of the root template and the part templates with their exact score in the same order. Again we may stop the process in each step according to a threshold. If the proposal is able to pass all steps we may report it according to NMS results.

Felzenszwalb et al. [1] and Sadeghi et al. [2] cache part template scores in their implementation to avoid re-computation. Because we operate in a sparse set of locations, the chances that a part template score is re-used at a certain location is small. Therefore we don't cache any scores. We observed that not caching scores could improve speed in our implementation as we need to allocate lower memory so we can utilize hardware cache more effectively.

Method	Ours	Ours	Ours	FTVQ [2]	DPM V5 [21]
Frequency	**100Hz**	**30Hz**	**15Hz**	**2Hz**	**0.07Hz**
aeroplane	0.1630	0.2695	0.3029	0.3320	0.3318
bicycle	0.3563	0.5735	0.5946	0.5933	0.5878
bird	0.0021	0.0909	0.0909	0.1027	0.1019
boat	0.0303	0.0303	0.1141	0.1568	0.1801
bottle	0.0909	0.1938	0.2425	0.2664	0.2535
bus	0.2989	0.4130	0.4720	0.5129	0.5056
car	0.2505	0.4240	0.4996	0.5373	0.5271
cat	0.1368	0.1725	0.1931	0.2251	0.1904
chair	0.0909	0.0909	0.1053	0.2010	0.2046
cow	0.0909	0.1062	0.1994	0.2432	0.2444
diningtable	0.1743	0.2500	0.2510	0.2685	0.2750
dog	0.0507	0.1159	0.1159	0.1260	0.1238
horse	0.2724	0.4735	0.5539	0.5651	0.5709
motorbike	0.2019	0.3850	0.4399	0.4849	0.4838
person	0.1962	0.3736	0.3971	0.4322	0.4327
pottedplant	0.0909	0.1179	0.1129	0.1345	0.1366
sheep	0.0000	0.0909	0.1702	0.2085	0.2154
sofa	0.1208	0.2860	0.3497	0.3568	0.3633
train	0.2801	0.3962	0.4198	0.4520	0.4651
tvmonitor	0.3075	0.3703	0.3840	0.4216	0.3943
mean AP	0.1603	0.2612	0.3004	0.3310	0.3294

Table 1. Comparison of different frame rates of our method with two major implementations of Deformable Parts Model: Fast Template evaluation using Vector Quantization (FTVQ) [2] and Deformable Parts Model (DPM) Version 5 [21]. We report per category AP that is computed as the average of precisions at 11 recall rates. Frequency is computed as $\frac{1}{t}$ where t is the time to detect all the 20 PASCAL VOC categories in one image. This time includes features computation time but excludes the time to load the image. We compare the algorithms on PASCAL VOC 2007 challenge that is a standard for benchmarking detection performance. Precision-Recall curves are illustrated in Figure 5.

6 Experimental Results

To evaluate our algorithm we compare it to a set of algorithms that are all based on Deformable Parts Model [1]. We evaluate our algorithm with three frequency settings: 15fps, 30fps and 100fps. We compare the techniques on PASCAL VOC 2007 that is established as a standard baseline.

We evaluated our algorithm by looking at the detection time and average precision (AP) score with respect to our baseline. We use DPM V5 [21] as our Average Precision baseline that is the most recent and most accurate implementation of DPM. To evaluate the time we compare to [2] that runs nearly two orders of magnitude faster than [21] and is the fastest algorithm before this publication. Our algorithm run on a system with an Intel Xeon E5-1650 processor

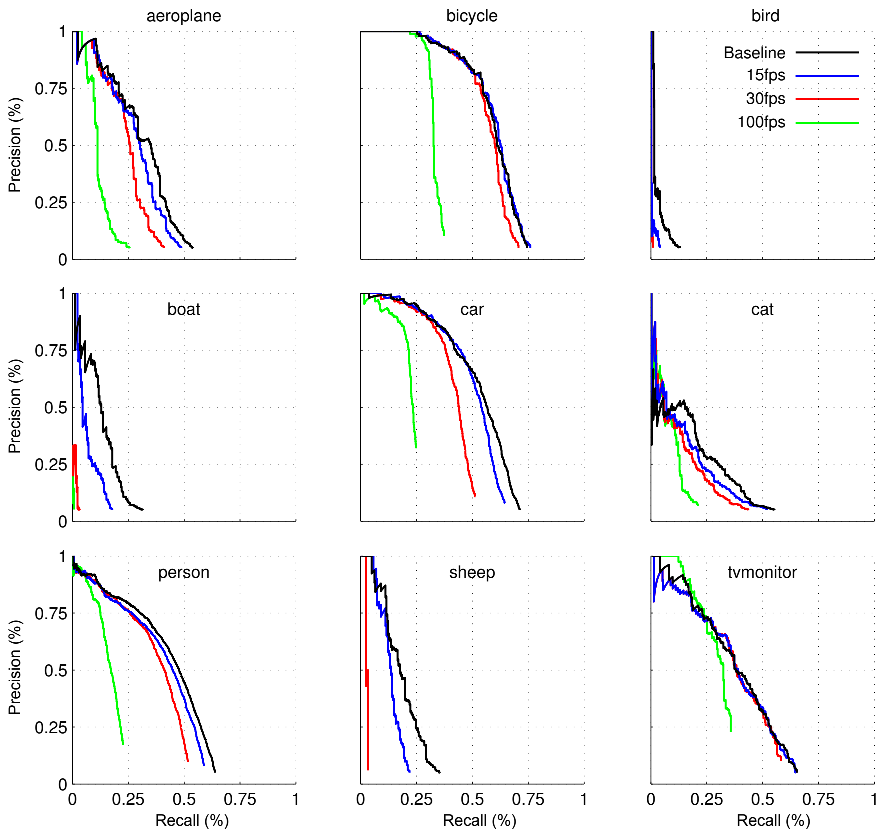

Fig. 5. Precision-Recall curves for 9 objects in PASCAL dataset comparing to the baseline. The black curve (above) corresponds to the accuracy of deformable parts model at regular speed (Table 1). In the blue curve all 20 PASCAL categories are detected at once in a time frame of 67ms (15fps). In the red curve all 20 PASCAL categories are detected at once in a time frame of 33ms (30fps). In the green curve all 20 PASCAL categories are detected at once in a time frame of 10ms (100fps). For all precision recall curves a threshold is chosen so each PR curve would cover precision > 0.05. In practical applications often one working point is chosen in the high precision area. Note that the gap between the curves in the high precision are tiny within the red, the blue and the black curves. This means in applications where a high precision working point is set, the loss is less noticeable. Note that the green curve fails to produce any detections for bird, boat and sheep categories. More information about APs can be found in Table 1.

and 32GB of RAM. Both our proposed algorithm and the baseline utilize all the 6 cores of the CPU at full load.

Method	mAP	time
HSC [19]	0.343	180s
WTA [14]	0.240	26s
DPM V5 [21]	0.330	13.3s
DPM V4 [20]	0.301	13.2s
DPM V3 [1]	0.268	11.6s
Vedaldi et al. [16]	0.277	7s

Method	mAP	time
FFLD [11]	0.323	1.8s
DPM Cascade [3]	0.331	1.7s
FTVQ [2]	0.331	0.53s
Ours at 15Hz	0.300	0.07s
Ours at 30Hz	0.261	0.03s
Ours at 100Hz	0.160	0.01s

Table 2. Comparison of various versions of DPM [1]. The reported time here is the time to complete the detection of 20 categories starting from raw image. Performance is computed on the PASCAL VOC 2007 dataset. Note that our method is three orders of magnitude faster than that of the original implementation. HSC [19] is slow because it uses an experimental set of features that is different than HOG. The method by Yan et al. [23] is not included in the table as its running time (0.22s per category) is reported on a single core. The methods in this table run 20 categories on six cores.

Our algorithm runs legacy models from DPM V5 [21] that are trained to have 6 root templates per category and 8 parts per root template. Our algorithm doesn't need to train a new model, we build up our model by processing the pre-trained detectors of DPM V5 [21].

We use a separate optimization techniques to speed up each of the stages of [2]. We implemented a highly optimized function to extract HOG features very quickly. Our implementation uses AVX vector operations and multiple cores. It is also optimized to utilizes CPU cache carefully. We also limit the number of layers to extract HOG features by a factor of 5. These optimizations together speed up HOG feature computation from 40ms in [2] to an average of 4ms.

We use a hierarchical clustering process to vector quantize HOG features (Section 3). Our hierarchical algorithm examines two sets of 16 clusters per HOG cell that is 8 times lower than that of [2] which examines 256 clusters in one layer. Since we process five times fewer feature layers (as mentioned in Section2), the average vector quantization load is further reduced. The total time required for our vector quantization technique is down from 70ms in [2] to about 5ms on average.

Our object proposal stage (Section 4) allows us to process only a small fraction of templates at each location. It can terminate early to acomodate time for other stages. Our object proposal process will terminate in 7ms if it is not terminated early. Our object scoring stage (Section 5) can also operate in a specified time-frame. On average it takes about 1.2μs to process one location for one category (including root and part scores). This algorithm processes as many locations as it affords in the specified time-frame. In the fastest case it can run at 100Hz. In this speed our algorithm affords to process only one location per root template (For PASCAL 2007 we have 1200 root templates that is 20 categories $\times 6$ components $\times 5$ scales).

Our implementation is very fast and light-weight. The code is implemented in C++ but can be called from MATLAB. Our algorithm can process an image

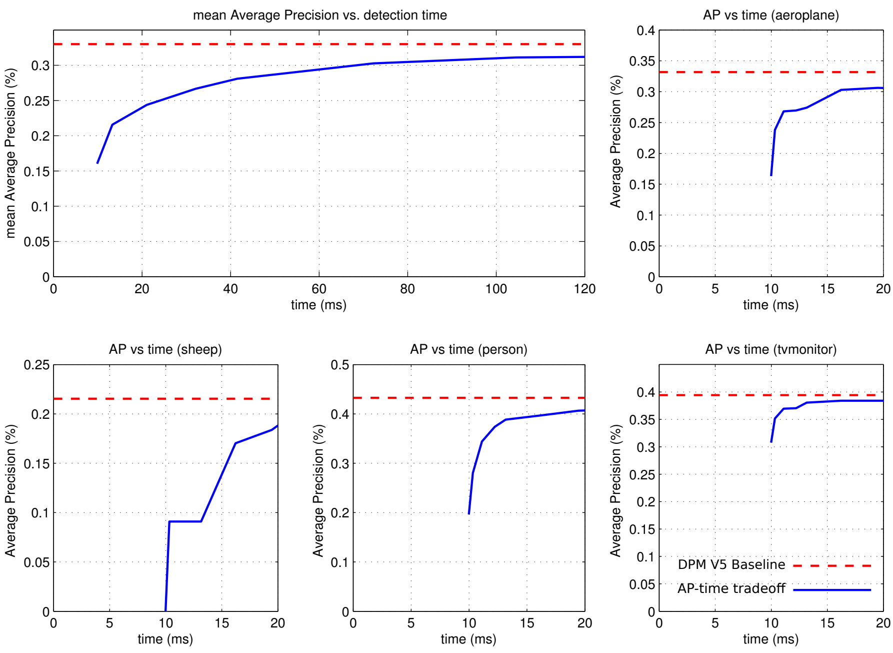

Fig. 6. Our method operates within a time limit specified by the user. It can jointly detect the entire set of PASCAL VOC challenge categories in about 10ms, that is about 0.5ms per category. The top-left plot shows the trade-off between operation time-frame and mean Average Precision (mAP) of the 20 PASCAL categories. In this setting all 20 objects are detected jointly within the time-frame. The rest of the plots show that this trade-off for detecting a single category. In this setting only one category is detected within the time-frame. Note that different categories respond differently to the time-limit. The Sheep detector fails at 100fps while the tvmonitor detector remains robust. The red dashed line shows DPM V5 [21] baseline while the solid blue curve shows Average Precision vs. time trade-off.

to detect 20 pascal categories simultaneously at 30fps or faster. It can further trade off accuracy for time; it can run at 100Hz while detecting 20 PASCAL categories in a time frame of 10ms. It also requires less than 10MB of memory at its peak demand to process an image for 20 categories (PASCAL Images are mostly 350×500 pixels large). This is three orders of magnitude faster than the original DPM V5 [21] implementation that itself is highly optimized.

Table 1 compares our algorithm with two established baselines. Our algorithm achieves 30Hz with an mAP of 0.26. At 15Hz, its mAP is 0.30; and at 100 Hz, its mAP is 0.16. Frequency is computed as $\frac{1}{t}$ where t is the time to detect all the 20 PASCAL VOC categories together in one image. This time includes features computation time but excludes the time to load image. We exclude the time to load the image because the time highly depends on the media.

Precision-Recall curves for our experiments are illustrated in Figure 5. Note that the gap between the curves in the high precision area in tiny between the red, the blue and the black curves. This is very important in practical applications as they often consider false positives costly and work in high precision regimes.

Table 2 compares our algorithm to several variations of DPM in terms of speed and accuracy. We report running time to detect all 20 PASCAL categories from raw image. We also compare our mean Average Precision to other techniques. In this table we compared to only algorithms that run on CPU. The fastest algorithm on GPU is Vanilla DPM [22] that runs at about 1Hz to detect the 20 PASCAL categories in a 640×480 image. It cannot sacrifice accuracy for speed.

Our algorithm can trade off accuracy for speed. Figure 6 illustrates the trade-off for both detecting all objects jointly and also detecting only a single object. This figure shows some detectors fail at 100fps while some others remain robust.

7 Discussion

We believe that there are further improvements available. We expect that speed could be improved by exploring our hashing process to: (a) interleave image loading and feature computation; and (b) avoid feature computation at some image blocks. We expect accuracy could be improved by careful tuning of time allocation (a) between proposal and detection process and (b) between templates.

The trade-offs in Figure 6 shows some detectors fail at 100fps while some others remain robust. This suggests the optimal time allocation is not to allocate equal time to each category; some categories need more time while some categories need less.

The optimal time allocation depends on several factors including: processor architecture, the global time limit, the demand by each category and the application defined priorities for detecting different categories. Feature extraction and quantization require a fixed processing budget. Our design allows the rest of the budget to be divided between proposal generation and object scoring. The optimal partition depends on the application.

Our experiments show objects that are harder to detect suffer more with a limited budget (see Figure 5, boat, bird) whereas categories with higher AP remain more robust. Furthermore, Certain objects are more likely to appear in groups (e.g. sheep, person) so they are more sensitive to limiting the number of locations to process. The study of optimal process allocation in different situations requires an extensive study that doesn't fit into the context of this paper.

Our trade-off allows for any speed improvement technique to directly result in accuracy improvement. The choice of working point in speed-accuracy trade-off allows for further data such as video or depth to be used for speed or accuracy.

References

1. P. F. Felzenszwalb and R. B. Girshick and D. McAllester and D. Ramanan. Object Detection with Discriminatively Trained Part Based Models. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.
2. M. A. Sadeghi and D. Forsyth Fast Template Evaluation with Vector Quantization In *Advances in Neural Information Processing Systems (NIPS)*, 2013.
3. P. F. Felzenszwalb and R. B. Girshick and D. McAllester. Cascade Object Detection with Deformable Part Models. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.
4. P. Dollár and R. Appel and S. Belongie and P. Perona Fast Feature Pyramids for Object Detection In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014.
5. N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.
6. I. Endres and D. Hoiem Category Independent Object Proposals In *European Conference on Computer Vision*, 2010.
7. Ming. Cheng and Z. Zhang and W. Lin and P. Torr BING: Binarized Normed Gradients for Objectness Estimation at 300fps In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.
8. D. Nister and H. Stewenius Scalable Recognition with a Vocabulary Tree In *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.
9. H. Pirsiavash and D. Ramanan Steerable part models In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.
10. P. Dollár and R. Appel and W. Kienzle Crosstalk Cascades for Frame-Rate Pedestrian Detection In *European Conference on Computer Vision*, 2012.
11. C. Dubout and F. Fleuret. Exact Acceleration of Linear Object Detectors. In *European Conference on Computer Vision*, 2012.
12. M. Pedersoli and J. Gonzalez and A. Bagdanov and and JJ. Villanueva. Recursive Coarse-to-Fine Localization for fast Object Detection. In *European Conference on Computer Vision*, 2010.
13. I. Kokkinos. Bounding Part Scores for Rapid Detection with Deformable Part Models In *2nd Parts and Attributes Workshop, in conjunction with ECCV*, 2012.
14. T. Dean and M. Ruzon and M. Segal and J. Shlens and S. Vijayanarasimhan and J. Yagnik. Fast, Accurate Detection of 100,000 Object Classes on a Single Machine. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.
15. P. Indyk and R. Motwani. Approximate nearest neighbours: Towards removing the curse of dimensionality. In *ACM Symposium on Theory of Computing*, 1998.
16. A. Vedaldi and A. Zisserman. Sparse Kernel Approximations for Efficient Classification and Detection In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.
17. Herv Jgou and Matthijs Douze and Cordelia Schmid. Product quantization for nearest neighbour search. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.
18. R. M. Gray and D. L. Neuhoff. Quantization. In *IEEE Transactions on Information Theory*, 1998.
19. X. Ren and D. Ramanan. Histograms of Sparse Codes for Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.
20. P. Felzenszwalb and R. Girshick and D. McAllester. Discriminatively Trained Deformable Part Models, Release 4. In *http://people.cs.uchicago.edu/ pff/latent-release4/.*

21. R. Girshick and P. Felzenszwalb and D. McAllester. Discriminatively Trained Deformable Part Models, Release 5. In *http://people.cs.uchicago.edu/ rbg/latent-release5/*.
22. H. Song and S. Zickler and T. Althoff and R. Girshick and M. Fritz and C. Geyer, P. Felzenszwalb, T. Darrell. Sparselet Models for Efficient Multiclass Object Detection In *European Conference on Computer Vision*, 2012.
23. J. Yan, Z. Lei and L. Wen and S. Z. Li. The Fastest Deformable Part Model for Object Detection In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.
24. S. Bengio and J. Weston and D. Grangier. Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems*, 2010.
25. R. Benenson and M. Mathias and R. Timofte and L. Van Gool. Pedestrian detection at 100 frames per second. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.