Computational Visual Media
DOI 10.1007/s41095-016-0051-7 Vol. 2, No. 3, September 2016, 245–256

Research Article

Parallelized deformable part models with effective hypothesis pruning

Zhi-Min Zhou[1], **Xu Zhao**[1] (✉)

Abstract As a typical machine-learning based detection technique, deformable part models (DPM) achieve great success in detecting complex object categories. The heavy computational burden of DPM, however, severely restricts their utilization in many real world applications. In this work, we accelerate DPM via parallelization and hypothesis pruning. Firstly, we implement the original DPM approach on a GPU platform and parallelize it, making it 136 times faster than DPM release 5 without loss of detection accuracy. Furthermore, we use a mixture root template as a pre-filter for hypothesis pruning, and achieve more than 200 times speedup over DPM release 5, apparently the fastest implementation of DPM yet. The performance of our method has been validated on the Pascal VOC 2007 and INRIA pedestrian datasets, and compared to other state-of-the-art techniques.

Keywords deformable part models (DPM); GPU; parallel computing; hypothesis pruning; visual detection

1 Introduction

Detecting objects in visual media is important for many computer vision tasks. To understand an image, object detection usually is the first step. It is still a challenging problem because the appearance of each object category varies greatly due to differences in illumination and viewpoint, as well as intra-class diversity of shape, colour, texture, and other visual properties for object instances.

Many techniques have been created to attack this fundamental challenge over past decades [1–4]. Amongst them, machine-learning based approaches have been very successful in capturing the invariance of specific object categories from large amounts of training data. In a machine learning framework, object detection is reduced to a binary classification problem, where a classifier is trained to determine whether or not an input image patch is an instance of a target category at the given image position and scale. This strategy is commonly called "sliding window" detection.

As a typical machine-learning based detection technique, in recent years, deformable part models (DPM) [4, 5] have achieved great success: they are very effective in detecting complex deformable objects. The method follows the general sliding window pipeline, but involves a more discriminative mechanism which models not only the global appearance of the whole sliding window, but also the appearance of local parts and their geometric relationships within the window. Deformable models can handle significant variability in object appearance because the parts making up an object have deformable configurations. This means that diverse appearance changes can be described and captured by the models. To further reinforce the capability of DPM to represent complex appearance variations, especially caused by the changes of view angle and pose, DPM mixture models were introduced in Ref. [4].

DPM is a widely applicable technique for general object detection. Recent works have extended DPM to more challenging tasks, such as face detection [6], pedestrian detection [7], pose estimation [8], and human motion recognition [9], and in doing so have achieved leading performance. DPM's computational cost, however, is a significant bottleneck, hindering its use in real world

1 Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: Z.-M. Zhou, 115236@sjtu.edu.cn; X. Zhao, zhaoxu@sjtu.edu.cn (✉).
Manuscript received: 2016-01-29; accepted: 2016-04-01

applications. For the Pascal VOC dataset, it takes more than 10 s to process each image; the time is mainly spent on feature computation and convolutional score calculation. DPM uses histogram of oriented gradient (HOG) features, which are representative but time consuming to calculate. Furthermore, after the templates (filters) have been trained, at detection time, the original DPM evaluates the appearance score densely at every image position and scale by calculating the correlation between filters and feature maps. These two factors result in a heavy computational burden for DPM, but potentially these can be largely relieved. To speed up DPM and make it more applicable to real world requirements, in this work, we accelerate DPM via the following contributions.

Parallelization. There are already several accelerated versions of DPM; they will be reviewed in detail in the next section. Most focus on algorithm redesign from a theoretical or technical viewpoint. Parallelization of DPM on GPUs, however, has not yet attracted much attention from the research community. Few publicly accessible works on GPU implementation of DPM can be found except for Ref. [10], which we will consider later. In fact, the DPM framework is very suitable for parallel GPU implementation. We have done this. We have conducted a comprehensive experimental evaluation on the Pascal VOC 2007 and INRIA pedestrian datasets. The performance of our GPU version of DPM, which we call DPM-GPU, significantly outperforms other accelerated DPM approaches, achieving over 136 times speedup compared to the original DPM release 5, without accuracy loss. This is apparently the most complete attempt to parallelize DPM release 5 on the GPU and evaluate it on challenging public datasets.

Hypothesis pruning. As DPM uses a standard sliding window procedure, every image position at every image pyramid level needs to be evaluated to determine whether or not an object instance is present. However, only a very sparse set contains true positives. Hypothesis pruning can therefore be used to largely reduce the unnecessary computing cost. In this work, we introduce a initial filter to prune object hypotheses. We apply a mixture root filter as a pre-filter for DPM. The motivations are twofold. First, a mixture root filter is capable of removing most of the negative hypotheses without lowering the average precision. We conduct a grid search to find the optimal threshold for mixture root filters, which is used to determine whether to keep a hypothesis or not. Secondly, we can repeatedly utilise the computed feature map and the mixture root filters obtained in the training stage but without extra training expense. The effectiveness of our strategy is validated on both benchmark datasets. It is over 200 times faster than DPM release 5 [4], averaged over all 20 categories of the Pascal VOC 2007 dataset, and without accuracy loss. As far as we know, this is the fastest implementation of DPM yet.

2 Related work

DPM [4] is based on pictorial structure [11, 12], extending it to a discriminative framework by introducing latent support vector machine (LSVM) to capture deformable part configurations, and mixture models to deal with more complex intra-class appearance variations. Although its detection performance is good, DPM is too slow for many real world applications.

To speed up DPM, several accelerated versions have been proposed. In terms of strategy, our work is most closely related to the cascade DPM [13], inspired by Ref. [14], which prunes partial hypotheses with a sequence of thresholds. The original DPM [4, 15] considers all possible locations in a given image when evaluating the score of a "root" part along with other constituent parts. Cascade DPM instead prunes sequentially hypotheses whose scores are below a series of thresholds. A part hierarchy is defined to determine the order of pruning. Cascade DPM is a serially executed algorithm, in which the hypotheses surviving after pruning step i are presented as candidates to pruning step $i + 1$. For this reason, cascade DPM is unsuited to GPU parallelization. In our work, all filters are computed in parallel independently, allowing full utilisation of GPU resources. Our experimental results reveal that for most object categories, a mixture of root filters is capable of pruning most false alarms without lowering the average precision. Hence, we select it but not the whole cascade DPM hierarchy as the pre-filter for hypothesis pruning.

In terms of GPU based parallelization of DPM, our work is most similar to Ref. [10], which parallelized DPM release 3. However, there are several significant differences and improvements in our work, comparing to Ref. [10], which are summarized as follows.

1. We not only accelerate DPM using GPU hardware, but also redesign the algorithm to include the pruning strategy; it plays an important role in the final speedup.
2. The mAP of 20 classes in the Pascal VOC 2007 dropped significantly in Ref. [10], while our work makes a small improvement. We adopt many measures to guarantee the detection performance of our parallelized DPM.
3. We have implemented most parts of DPM on the GPU, including the procedure for merging root filter scores and part filter scores, and finding candidate windows whose scores are bigger than a threshold. These steps are not considered in Ref. [10].
4. When computing convolutions between features and filters, the filters are put in constant memory to reduce global memory accesses in Ref. [10]. This limits the model size to be smaller than the constant memory size, which is restrictive. In our implementation, shared memory is used to store filters.

Further different acceleration strategies have been used to speed up DPM, which emphasise different aspects of detection costs using DPM. In Ref. [16], a coarse-to-fine inference procedure is proposed to minimize the cost of matching each part to the image using a multiresolution hierarchical part-based model. In Ref. [17], FFT is used to reduce the computational cost of convolution and accelerate it by one order of magnitude. To reduce the search space, motivated by Ref. [18], a branch-and-bound scheme [19] is introduced to compute bounds that accommodate hypothesis. Recently, Ref. [20] accelerated DPM by constraining 2D correlation as a linear combination of 1D correlations, using neighbourhood-aware cascades for part pruning, and building look-up tables for HOG extraction. It is claimed to be the fastest DPM implementation but still spends nearly 300 ms per frame.

We compare all of the methods mentioned above and ours, using benchmark datasets to evaluate what constitutes the state-of-the-art in DPM acceleration. The results demonstrate that our strategy achieves the best performance.

3 Parallel DPM and hypothesis pruning

3.1 DPM revisited

DPM uses a sliding window pipeline, in which a score $\beta \cdot \Phi(x)$ needs to be evaluated to classify an image point x as an object or not, where β contains model parameters and $\Phi(x)$ is a feature vector [5]. Taking into account the choice of mixture components and part deformation, the classifier computes the score in the form:

$$f_\beta(x) = \max_z \beta \cdot \Phi(x, z) \tag{1}$$

where z are latent values specifying a mixture component choice and the part configurations associated with that component [5]. Detection occurs when the score is above a threshold, and the inferred latent values $z^* = \arg\max_z \beta \cdot \Phi(x, z)$ are returned. The model parameters β are obtained by training an LSVM using a coordinate descent algorithm.

An n-part model for an object category is defined by a set of parameters $\beta = (F_0, (F_1, d_1), \cdots, (F_n, d_n), b)$, where F_0 is a root filter, F_i is a part filter, d_i is a deformation vector, and b is a scalar bias term. An object hypothesis at $z = (p_0, \cdots, p_n)$, where $p_i = (x_i, y_i, s_i)$ specifies the position and scale of the i-th filter, is given by

$$\text{score}(z) = \sum_{i=0}^{n} F_i \cdot \phi(I, p_i) - \sum_{i=1}^{n} d_i \cdot \psi(p_i, p_0) + b \tag{2}$$

where $\psi(p_i, p_0) = (\text{d}x_i, \text{d}y_i, \text{d}x_i^2, \text{d}y_i^2)$, with $\text{d}x_i = x_i - x_0$ and $\text{d}y_i = y_i - y_0$ and I is the given image.

To detect objects using a mixture model, the accumulated root scores must be computed independently for each component, and the component hypothesis with the highest score is selected.

3.2 Mixture root filters guided hypothesis pruning

What we call the *mixture root filters* here is the DPM model with no parts. There are three methods to determine a mixture model for hypothesis pruning.

1. The mixture root model is produced during training of the final DPM model. This gives the mixture root model directly without an extra

model training process. This is an advantage of our proposed hypothesis pruning method.

2. To get a better mixture root model to guide hypothesis pruning, we can increase the iteration time or enhance the termination conditions during mixture root model training.
3. We can set the number of parts in the DPM model to zero, so the final model after training is the mixture root model.

Generally speaking, the first method is the most convenient way to obtain mixture root filters, but the other two methods perform better at guiding hypothesis pruning. We have tried all three methods to select the best mixture root model for hypothesis pruning. The size of each root filter is the same in the final model and in the mixture root model. We emphasize that the root filters contained in the mixture root model differ from those in the final model. Root filters in the final model lose their detection ability when trained in combination with with part filters.

The exhaustive search over all image scales and positions is the most time-consuming part of DPM. So to speed it up, we wish to reduce the number of ineffective hypotheses. To this end, we have designed a two-stage detection pipeline. In the first stage, a coarse but dense search is conducted over all image points to filter out most futile hypotheses. We use the mixture of root filters, the intermediate result in DPM training, as the initial filter for coarse pruning. Thus, for the c-th component of the mixture root filters, the selected hypothesis set is

$$\mathcal{D}_{\mathrm{h}}^{(c)} = \{z | \mathrm{score}_{\mathrm{r}}^{(c)}(z) = F_0^{(c)} \cdot \phi(I, p_0) > T_{\mathrm{mix}}\} \quad (3)$$

where T_{mix} is a threshold, determined in the training stage by grid search (see later). In the second stage, we make a fine search over the selected hypotheses using the final full model: the part deformations are used to further refine the results of coarse search.

Using mixture root filters as a pre-filter can (i) repeatedly utilize the computed feature map and (ii) avoid training an extra filter. By using grid search, we can find the optimal threshold for the coarse search. This enables the computational cost to be largely reduced while the average precision is not lowered. The redesigned algorithm can be implemented not only on the CPU but also on the GPU. Furthermore, the pre-filtering step is also suitable for parallel computing. The acceleration scheme based on hypothesis pruning is described in Algorithm 1.

Note that after obtaining the detection set $\mathcal{D}^{(c)}$, we select the highest scoring component hypothesis among all the components in mixture models as the final detected object.

We now briefly analyze the time complexity of our hypothesis pruning algorithm. Suppose we have a feature pyramid with L valid positions to be evaluated and a DPM model containing M components. Take one component as an example (the whole model is the same), which contains one root filter and N part filters. Then the original algorithm needs

$$32L(H_{\mathrm{R}}W_{\mathrm{R}} + NH_{\mathrm{P}}W_{\mathrm{P}}) \quad (4)$$

multiplications, where 32 is the dimensionality of the DPM feature, H_{R}, W_{R} represent the height and width of the root filter, and H_{P}, W_{P} represent the height and width of the part filter, respectively. Suppose a fraction $\eta \in (0, 1)$ of hypotheses are rejected by the mixture model. Then our proposed method needs

$$32L\left[H_{\mathrm{R}}W_{\mathrm{R}} + (1-\eta)(H_{\mathrm{R}}W_{\mathrm{R}} + NH_{\mathrm{P}}W_{\mathrm{P}})\right] \quad (5)$$

multiplications. Thus, the rejection rate η is critical to our proposed hypothesis pruning algorithm, which

Algorithm 1: Hypothesis-pruning-based acceleration

Input: T_{mix}: pre-filter threshold, T_{f}: final threshold, I: input image, $M_{\mathrm{m}}^{(c)}$: c-th component of the mixture root model, $M_{\mathrm{f}}^{(c)}$: the c-th component of the final model.

Output: Detection set $\mathcal{D}^{(c)}$ from the c-th component.

```
Calculate feature pyramid F_pyra with levels 0 to l
for i = 0 to l do
    for all (x, y) in feature plane F^i_pyra do
        score^{il}_{m_c}(x, y) = convolve(M^{(c)}_m(x, y), F^i_pyra(x, y))
    end for
end for
for i = 0 to l do
    for all (x, y) in feature plane F^i_pyra do
        if score^i_{m_c}(x, y) < T_mix then
            break
        else
            score^i_f(x, y) = con(M^{(c)}_f(x, y), F^i_pyra(x, y))
            if score^i_f(x, y) >= T_f then
                D = D ∪ (i, x, y)
            end if
        end if
    end for
end for
D^{(c)} = non-maximum-suppression(D)
```

will be considered in Section 4. If η is big enough, then the speedup factor is

$$\frac{H_{\mathrm{R}}W_{\mathrm{R}} + NH_{\mathrm{P}}W_{\mathrm{P}}}{H_{\mathrm{R}}W_{\mathrm{R}}} = 1 + \frac{NH_{\mathrm{P}}W_{\mathrm{P}}}{H_{\mathrm{R}}W_{\mathrm{R}}} \tag{6}$$

3.3 GPU implementation

We parallelize the proposed DPM acceleration scheme on CUDA platform [21]. The complete procedure is shown in Fig. 1. The platform comprises a CPU host and GPU device. Most of the time consuming modules, such as feature extraction and convolution, are implemented on the GPU. The CPU hosts the computing flow and mainly performs initialisation and post-processing. The acceleration process involves the following steps.

Host initialization. In this step, image, model parameters, and the related thresholds are loaded. The CPU prepares data for the GPU. These data include filters, filter size, image pyramid size, feature pyramid size, etc.

GPU computing. Based on the input image, the image pyramid is first created. Next, starting from the top level of the pyramid, the feature map is computed. By convolving the feature map with the mixture root model, we get proposals according to the threshold set by grid search. Then the final model is applied to evaluate these proposals. When the current pyramid level is below the top 10 levels, the part filters' responses need to be computed, considering their deformation cost. The summation of the root and part responses forms the final response at this level. Once the response pyramids of every component have been formed, the GPU searches in parallel for points scoring above the final threshold, and computes their bounding boxes according to the size of the component and the corresponding pyramid level. The results are then transferred back to CPU.

Post-processing. In a given image, just a few object instances are typically detected. Post-processing of these detection results, including bounding box clipping and non-maximum suppression, is done on the CPU.

3.4 GPU optimization

Our optimization of the GPU implementation considers memory management and involves algorithm redesign. GPU optimization plays an important role in GPU parallelization, and can double the detection speed.

Memory management is critical to acceleration. In GPU parallel computing, CGMA (compute to global memory access) [22] is an important index to evaluate GPU code. The memory bandwidth is about 224 GB/s for the GTX970, but the theoretical computing ability is up to 2440 GFLOPs, more than 10 times the memory bandwidth. In order to make full use of GPU computing ability, it is critical to manage GPU memory. Different kinds of GPU memory have different characteristics [23]: a short summary is displayed in Table 1. In our implementation, in a large number of tests, we find that using following strategies for allocating GPU memory resources results in the best performance.

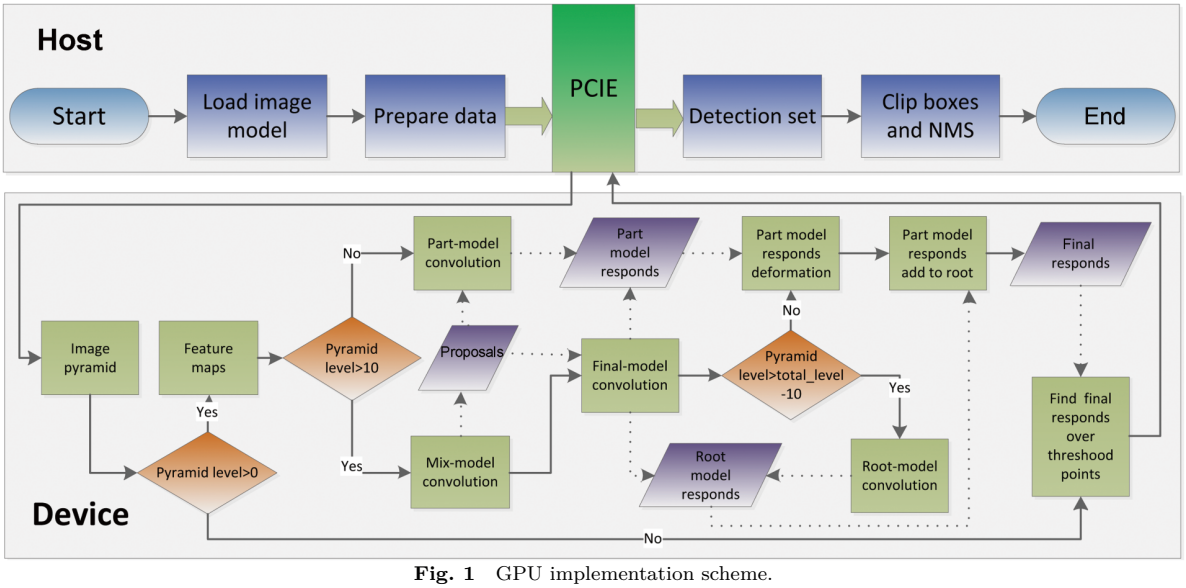


Fig. 1 GPU implementation scheme.

Table 1 Characteristics of different GPU memory

Memory	Location	Size	Access	Cache	Speed	Lifetime
Global memory	Onboard	1–8 GB	Device R/W; host R/W	Non	Slow	Program
Constant memory	Onboard	32/64 KB	Device R/W; host R/W	Yes	Quick	Program
Texture memory	Onboard	No more than global memory	Device R/W; host R/W	Yes	Quick	Program
Shared memory	Onchip	32–96 KB	Device R/W	N/A	Fast	Block
Register memory	Onchip	64 KB	Device R/W	N/A	Fastest	Thread

- Constant memory is used to store the pyramid size, filter size, and so on, as they are used repeatedly. Many threads access the same address in constant memory; constant memory broadcasts it instead of communicating with each thread in turn.
- Texture memory is used to band each level of the feature pyramid, as the data is relatively big and is frequently visited. Generally one level of the feature pyramid is beyond the size of shared memory, although shared memory is the best choice.
- Shared memory is used to save filter data. Each filter is convolved with each position in the feature pyramid. As the feature pyramid is banded with the texture memory, each thread in a CUDA block can load filters in shared memory together, which is much faster than each thread loading filters on its own.
- Global memory is used to save the results which will be used later or transferred back to the CPU. As constant memory and texture memory cannot be written at running time, shared memory is cleared after each GPU launch, and all computing results need to be written to global memory.

After GPU memory optimization, the speedup for the Pascal VOC 2007 dataset was increased from 80 to 160.

Algorithm optimization is another important part of GPU acceleration. During GPU implementation of the DPM algorithm, we found some redundant parts could be omitted, as follows:

- The top interval which is set to be 10 levels of the feature pyramid does not need to be convolved with part filters. Also, convolution of the bottom interval levels of the feature pyramid with root filters can be omitted, because part filters need to be placed at twice the spatial resolution of the placement of the root in the DPM algorithm.
- In order to deal with image border conservatively in the original DPM algorithm, each level in the feature pyramid is padded with a certain size feature, which is set according to the filter's size in the model. The artificially added feature is 32D data represented by F_{pad}; its first 31D are 0 and the last dimension is 1. In our GPU implementation we do not pad the border feature, but use B to represent the padded border area: if a position $P(x, y) \subseteq B$ then the dot product is

$$F_{\text{M}}(f_1, \cdots, f_{32}) \cdot F_{\text{pad}} = f_{32} \tag{7}$$

where F_{M} is a point in the model. There is no need to correlate a padded feature with filters, we can simply decide whether a point is in the padded border or not. This not only makes the feature pyramid smaller, but also cuts down the amount of calculation and especially the number of memory accesses.
- Bounding box prediction is dropped in our method, because it is not a fundamental part of the original DPM algorithm.

Optimizing the original algorithm achieves a 200-time speedup in our GPU implementation. We have profiled our final GPU implementation, as shown in Fig. 2. The model used had typical settings as in Ref. [24]. The size of the image analyzed was 640×425, as used in Ref. [4]. The penultimate row in Fig. 2 shows the GPU resource usage. At first we compute the image pyramid, corresponding to resizeXKernel and resizeYKernel in Fig. 2. As can be seen, the GPU resources are nearly fully used in this stage (the left red rectangle enclosed part). Since we extract features and compute correlations from top to bottom in the image pyramid, the GPU usage is low at first and gradually becomes higher with the increase in image size (the right red rectangle enclosed part represents the GPU usage of the computing process for the biggest image in the pyramid).

4 Empirical results

The proposed method has been evaluated on the

Fig. 2 CUDA Nsight profile of our GPU implementation.

Pascal VOC [25], INRIA pedestrian [3], and SJTUVehicle datasets. Each dataset contains thousands of real world images with bounding boxes marking ground-truth object instances. If a predicted bounding box overlaps more than 50% with a ground-truth bounding box, it is considered correct. Only one detection is considered correct when a system detects several bounding boxes which overlap a single ground-truth bounding box.

4.1 Experimental setting

In this work, all evaluated methods were built based on DPM release 5 [24], using 10 levels in an octave, an HOG bin size of 8, 6 components for each category (or 2 for the INRIA dataset), and 8 parts for each component. We implemented a C++ version of DPM release 5 along with its multicore version, a C++ GPU version, and a C++ GPU with hypothesis pruning version.

For fair evaluation, all programs were run on the same computer with an Intel i5 750 2.67 GHz CPU (with 4 cores) and an nVidia GTX-970 GPU.

4.2 Grid search for the optimal threshold

In our DPM acceleration scheme, to prune the hypotheses, we first needed to set a threshold for the pre-filter. The final model only evaluates the points whose score is over that threshold. To do so, we conducted grid search on training data to find the optimal threshold. For the mixture root filter, we sought an optimal recall rate threshold in the interval $[-0.15, 0.15]$ centred on 0.95 with step length 0.01. We thus needed to test over the entire dataset 31 times for each mixture root model. It was a time consuming procedure, for Pascal VOC 2007 taking 18 hours to test one threshold; it just cost 5 minutes using our method. The aim is to find a threshold giving a good balance between average precision (AP) and time cost.

Figure 3 shows the grid search results for 8 example Pascal VOC object categories. By observing the AP–time curves shown in the figures, we can find that when the threshold is relatively low, almost all the points can pass the filtering and the time cost is slightly higher than a pure GPU version but AP remains the same. With an increased threshold, some points are rejected by the pre-filter so that the computational burden is alleviated and the detection time falls. The AP increases a little at first as some false positives are filtered out. As the threshold is further increased, more and more hypotheses are rejected and therefore the detection time gradually levels out. At the same time, both recall and AP approach 0.

We expected that when the AP curve reached its highest peak, the detection time would be close to its lowest bound. For the Pascal VOC dataset, 15 out of 20 object categories were in line with our expectation, but the other 5 object categories did not agree with this rule. This is because for these categories, the discriminative power of the mixture root filter is not strong enough to clearly differentiate false-positives and true-positives. Recall would be lowered if the true-positives were also be filtered out, so that AP will reduce if the increase in precision

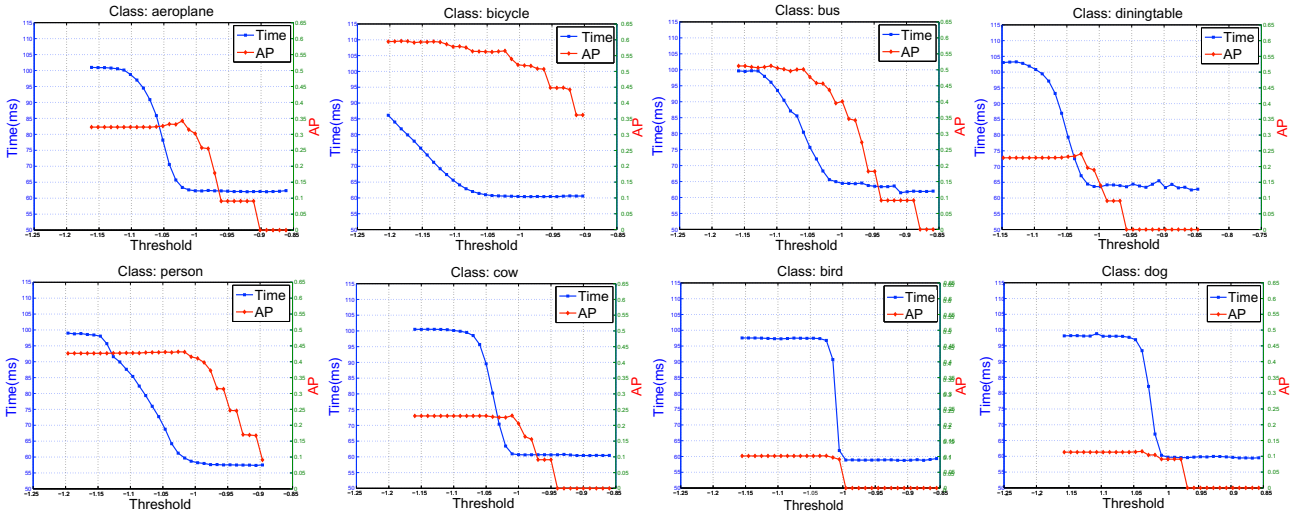


Fig. 3 Grid search for the optimal threshold for the pre-filter.

cannot make up for the loss in recall. Fortunately, even for the hard categories, we can still find a suitable threshold to strike a balance between time cost and AP.

4.3 Evaluation on Pascal VOC 2007

We evaluated a range of DPM acceleration methods on the Pascal VOC 2007 dataset: (1) DPM release 5 [24], (2) cascade DPM [13], (3) branch-bound DPM [19], (4) FFT DPM [17], (5) coarse-to-fine DPM [16], (6) fastest DPM [20], and our approach.

Firstly, we report the APs of all 20 object categories in Table 2, where DPM-GPU-P is the proposed method in this work and DPM-C++-omp is the multicore C++ version of DPM release 5. All methods achieve similar APs; nevertheless, our proposed method gets the highest AP in 13 object categories.

The critical performance indicator, time cost, of each compared method, is reported in Table 3. The speedup factors for different accelerated versions of DPM algorithm are presented in Table 4. In order to make a comparison with related work, we have included another 5 methods in this table: as the source code of those 5 methods has not been published, the speedup factors are computed based on the results in their papers. The C++ version of DPM release 5 is faster than the original DPM release 5, but has an acceleration factor of just 1.45 (see Table 4), and the speedup is not significant. This is because in DPM release 5, feature extraction and convolution operations, the most time-consuming parts, are also programmed in C. The multicore C++ version does not even reach 4 times faster than the C++ version, because only convolution operations are processed in parallel by multiple cores. Cascade-DPM, along with branch-bound-DPM, FFT-DPM, and coarse-to-fine-DPM attain one order of magnitude of acceleration over the baseline DPM release 5. The recently proposed fastest DPM [20] just runs over 40 times faster than the baseline. For the Pascal VOC 2007, the average detection time of another DPM GPU implementation [10] is 154 ms according to the author, giving a speedup factor of 88.5 over the baseline release 5. Our implementation of GPU parallelization achieves two orders of magnitude acceleration and can run up to 136 times faster than the baseline. By adding hypothesis pruning, our proposed acceleration scheme achieves 200-time speedup. It takes about 5–6 minutes to process the entire Pascal VOC test set once using our method, compared to 18 hours using release 5 DPM. This makes it possible to search for the optimal threshold over the entire training dataset, not just over an interval centered on 0.95 recall.

Note that for a fair comparison, the time spent on reading images is included in the detection time in our tests for all the compared methods. As the detection time of our method is just nearly 65 ms per frame, the performance gains would be more significant if image reading time costs were ignored.

To validate the effectiveness of our proposed pre-filter for hypothesis pruning, we tested how many hypotheses can be filtered out in the training data,

Table 2 Average precision (AP) of different methods on 20 categories of Pascal VOC 2007 test dataset

	Aero	Bike	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow	Table	Dog	Horse	Mbike	Person	Plant	Sheep	Sofa	Train	TV	Avg.
DPM-v5 [24]	32.2	59.3	**10.2**	15.2	26.1	50.9	53.7	20.5	20.2	22.9	23.1	11.3	56.5	48.3	**43.3**	13.3	20.2	30.4	45.0	40.3	32.1
DPM-cascade [13]	32.1	59.4	**10.2**	14.8	26.1	47.1	53.8	20.6	20.1	**23.4**	19.6	**11.4**	56.5	48.4	43.2	13.4	20.2	30.3	45.0	40.1	31.8
DPM-C++	31.6	59.5	9.8	15.1	26.2	50.6	53.0	15.5	20.6	22.1	20.6	10.4	56.4	**50.5**	40.2	12.6	20.6	30.5	**45.8**	41.5	31.7
DPM-C++-omp	31.5	59.5	9.8	15.1	26.2	50.6	53.0	15.5	20.6	22.1	20.5	10.3	56.4	**50.5**	40.2	12.6	20.6	30.5	**45.8**	41.5	31.6
DPM-GPU	32.3	59.9	**10.2**	15.1	26.2	**51.2**	54.2	**21.0**	20.2	23.0	22.7	11.3	57.3	48.7	42.8	13.2	20.2	30.4	45.4	41.0	32.3
DPM-GPU-P	**33.2**	**60.2**	9.7	**15.8**	**26.3**	49.7	**56.1**	19.7	**21.1**	22.6	**23.5**	10.9	**58.0**	48.8	43.2	**13.7**	**20.9**	**30.9**	45.5	**41.7**	**32.6**

Table 3 Average time of different methods on 20 categories of Pascal VOC 2007 test dataset (Unit: ms/frame)

	Aero	Bike	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow	Table	Dog	Horse	Mbike	Person	Plant	Sheep	Sofa	Train	TV	Avg.
DPM-v5 [24]	13648	14622	13496	14204	12995	14115	12745	15082	14612	14941	15623	14076	14030	13629	11971	11484	11189	13245	14369	12565	13632
DPM-cascade [13]	1287	920	1658	1454	1738	1015	1740	1269	2085	974	854	1281	981	1037	2268	1954	1169	974	914	1368	1347
DPM-C++	8533	9068	8436	9796	9099	10261	8615	10016	9746	10016	10608	10407	9409	9102	8325	8414	8232	9572	10618	9187	9373
DPM-C++-omp	3482	3557	3327	3465	3149	3281	2818	3225	3227	3217	3343	3311	3387	3222	2969	2864	2784	3274	3424	3048	3219
DPM-GPU	102.4	98.7	97.7	101.3	97.2	101.0	98.4	100.5	98.3	99.7	103.7	100.2	99.7	102.8	97.7	97.5	97.0	102.8	102.1	101.4	100.0
DPM-GPU-P	**63.5**	**68.8**	**59.5**	**61.4**	**68.1**	**75.7**	**62.1**	**80.5**	**59.1**	**58.7**	**67.1**	**67.1**	**63.1**	**70.6**	**62.5**	**56.8**	**60.5**	**66.4**	**63.2**	**58.7**	**64.9**

Table 4 Speedup factor of the compared methods on Pascal VOC 2007

	DPM-v5 [24]	Cascade [13]	C++	C++-omp	DPM-GPU	**D**PM-GPU-P	DPM-BB [19]	DPM-FFT [17]	DPM-CF [16]	Fastest-DPM [20]	DPM-gpu [10]
Speedup (×)	1.00	10.12	1.45	4.34	136.31	**203.21**	3.81	8.51	7.37	42.17	88.5

while at the same time, AP remains similar to its baseline value. The results can be found in Fig. 4. Our hypothesis pruning scheme removes over 98% of ineffective hypotheses in many classes of the Pascal VOC and only three classes (bottle, bus, and motorbike) lie below the average filtering rate which is up to 95% without effecting detection performance. This shows that mixture root model based pre-filtering works effectively in hypothesis pruning.

4.4 Evaluation on INRIA pedestrian dataset

We also evaluated the methods on the INRIA pedestrian dataset [3]. Detection performance is given in Fig. 5. Obviously, the GPU version and GPU-filter (the proposed pre-filter based) version have similar performance, which are much better than the baseline and cascade-DPM. The time costs of the compared methods are given in Table 5. Although there is just one component in the INRIA

Table 5 Average time of different methods on INRIA pedestrian dataset

	DPM-v5 [24]	Cascade [13]	DPM-GPU	DPM-GPU-P
Average time (ms/frame)	21500	1770	90.1	71.2

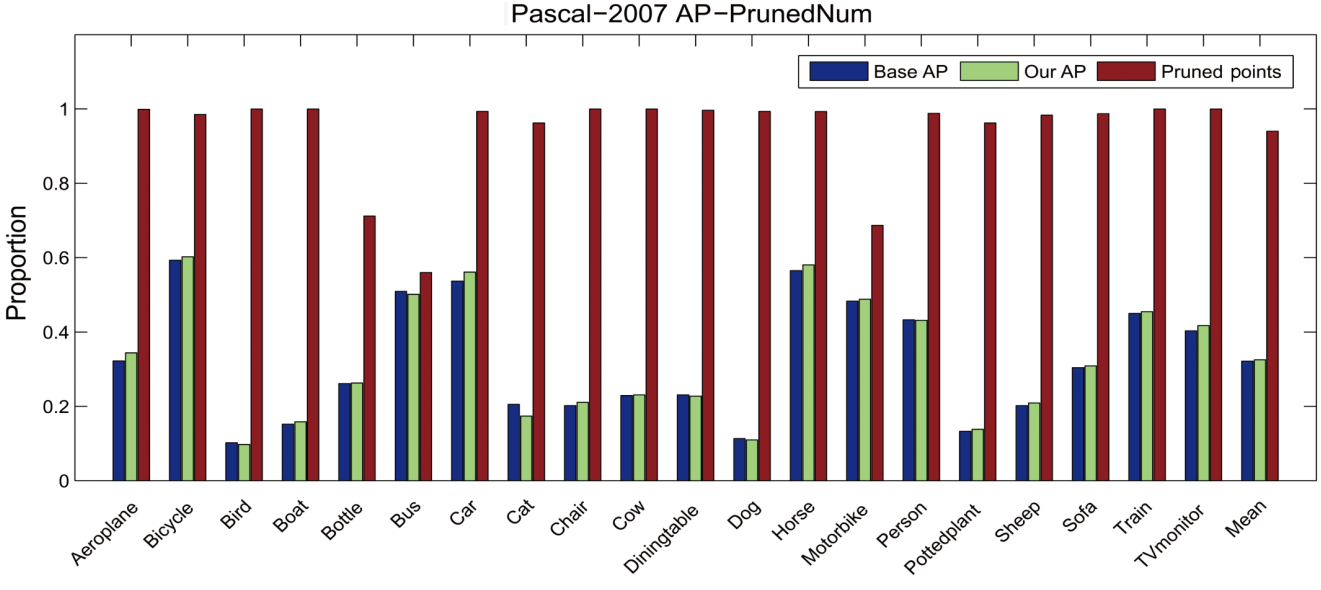


Fig. 4 Proportion of hypotheses pruned, along with the AP of the baseline and the proposed method.

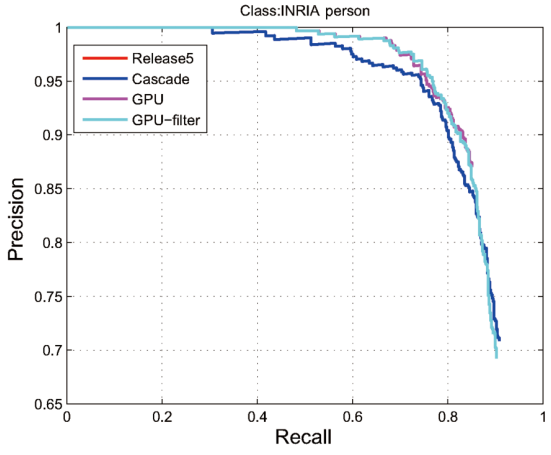


Fig. 5 Precision–recall curve comparison between different methods.

model, the speed is not as fast as expected. The main reason is that the average image size and filter size are much bigger than for the Pascal VOC as this dataset focuses on a standing person, which has a much bigger height to width aspect ratio. This directly increases the computational time of the convolution operation. Another reason is that a single component cannot fully utilize the GPU's parallelization capability. Even so, our method still achieves 300-time acceleration over the baseline.

4.5 Evaluation on plate localization dataset

We have also evaluated our parallelized DPM on the SJTUVehicle dataset, collected by SJTU visionlab. The SJTUVehicle dataset contains 270,000 vehicle images from highway monitoring. About 5500 frames were selected and labeled. As the dataset is not published at present, we chose 4 image sizes for evaluation. Our plate license DPM model contains 12 components, each of which consists of 1 root filter and 7 part filters. The part filter's aspect ratio is 6:3 to match characters in real Chinese license plates.

The DPM algorithm gets excellent detection performance on the SJTUVehicle dataset: both recall and precision are above 95%, and our method hardly loses any licence plates in the SJTUVehicle dataset. Our parallelized implementation provides an encouraging average detection speed of up to 10 FPS. The speedup increases sharply along as the image size changes from 300×300 to 600×600, as can be seen in Table 6. Small images cannot make full use of GPU computing resources; nevertheless, our method still achieves around 200-time acceleration. As the image size becomes big enough, our method runs nearly 300 times faster than the baseline.

5 Conclusions and discussion

In this paper, we have presented an effective DPM parallelization method. Firstly, we implemented the original DPM release 5 on a GPU platform and achieved 136-time acceleration. Furthermore, we introduced mixture root filters as a pre-filter to prune ineffective hypotheses, and achieved more than 200-time speedup over the baseline, without accuracy loss. A comprehensive evaluation on the Pascal VOC, INRIA pedestrian, and SJTUVehicle datasets demonstrates that our method is the fastest implementation of DPM amongst various compared methods.

In our method, learning an optimal threshold for the pre-filter is a big challenge. Our current grid search strategy considers AP as the criterion to measure detection performance. This gives precision and recall equal importance, but actually recall is more important than precision during training, because a few more false-positives have little impact on speed as long as true-positives are kept. Also, our current pre-filter still needs to be used to consider all points at every scale. Other simple cascade pre-filters could be explored to further reduce the computational cost of the current mixture root filter. The above two concerns will determine the direction

Table 6 Average time and speedup of different methods on the SJTUVehicle dataset

Method	Four kinds of image size							
	1034×834		698×728		579×591		282×273	
	Time (s/frame)	Speedup	Time (s/frame)	Speedup	Time (s/frame)	Speedup	Time (s/frame)	Speedup
DPM-v5	60.5458	1	38.8021	1	28.5938	1	9.1171	1
DPM-C++	36.5954	1.65	23.3871	1.66	16.9455	1.69	4.8539	1.88
DPM-C++-omp	14.4747	4.18	9.1869	4.22	6.6382	4.31	1.9472	4.68
DPM-GPU	0.3701	163.6	0.2285	169.86	0.1706	167.6	0.06186	147.4
DPM-GPU-P	**0.2052**	**295.1**	**0.1326**	**292.7**	**0.1052**	**271.8**	**0.05102**	**178.8**

of our next work on this topic.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Nos. 61273285, 61375019).

References

[1] Dalal, N.; Triggs, B. Histograms of oriented gradients for human detection. In: Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 1, 886–893, 2005.

[2] Viola, P.; Jones, M. Rapid object detection using a boosted cascade of simple features. In: Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 1, I-511–I-518, 2001.

[3] Lienhart, R.; Maydt, J. An extended set of Haar-like features for rapid object detection. In: Proceedings of International Conference on Image Processing, Vol.1, I-900–I-903, 2002.

[4] Felzenszwalb, P. F.; Girshick, R. B.; McAllester, D.; Ramanan, D. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence* Vol. 32, No. 9, 1627–1645, 2010.

[5] Felzenszwalb, P.; Girshick, R.; McAllester, D.; Ramanan, D. Visual object detection with deformable part models. *Communications of the ACM* Vol. 56, No. 9, 97–105, 2013.

[6] Zhu, X.; Ramanan, D. Face detection, pose estimation, and landmark localization in the wild. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2879–2886, 2012.

[7] Yan, J.; Zhang, X.; Lei, Z.; Liao, S.; Li, S. Z. Robust multi-resolution pedestrian detection in traffic scenes. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 3033–3040, 2013.

[8] Yang, Y.; Ramanan, D. Articulated pose estimation with flexible mixtures-of-parts. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 1385–1392, 2011.

[9] Wang, L.; Wu, J.; Zhou, Z.; Liu, Y.; Zhao, X. Recognition by detection: Perceiving human motion through part-configured feature maps. In: Proceedings of IEEE International Conference on Multimedia and Expo Workshops, 1–6, 2014.

[10] Gadeski, E.; Fard, H. O.; Borgne, H. L. GPU deformable part model for object recognition. *Journal of Real-Time Image Processing*, 1–13, 2014.

[11] Felzenszwalb, P. F.; Huttenlocher, D. P. Pictorial structures for object recognition. *International Journal of Computer Vision* Vol. 61, No. 1, 55–79, 2005.

[12] Fischler, M. A.; Elschlager, R. A. The representation and matching of pictorial structures. *IEEE Transactions on Computers* Vol. C-22, No. 1, 67–92, 1973.

[13] Felzenszwalb, P. F.; Girshick, R. B.; McAllester, D. Cascade object detection with deformable part models. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2241–2248, 2010.

[14] Amit, Y.; Geman, D. A computational model for visual selection. *Neural Computation* Vol. 11, No. 7, 1691–1715, 1999.

[15] Felzenszwalb, P. F.; McAllester, D. A.; Ramanan, D. A discriminatively trained, multiscale, deformable part model. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 1–8, 2008.

[16] Pedersoli, M.; Vedaldi, A.; Gonzàlez, J.; Roca, X. A coarse-to-fine approach for fast deformable object detection. *Pattern Recognition* Vol. 48, No. 5, 1844–1853, 2015.

[17] Dubout, C.; Fleuret, F. Exact acceleration of linear object detectors. In: *Lecture Notes in Computer Science, Vol. 7574*. Fitzgibbon, A.; Lazebnik, S.; Perona, P.; Sato, Y.; Schmid, C. Eds. Springer Berlin Heidelberg, 301–311, 2012.

[18] Lampert, C. H.; Blaschko, M. B.; Hofmann, T. Efficient subwindow search: A branch and bound framework for object localization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* Vol. 31, No. 12, 2129–2142, 2009.

[19] Kokkinos, I. Rapid deformable object detection using dual-tree branch-and-bound. In: Proceedings of Advances in Neural Information Processing Systems 24, 2681–2689, 2011.

[20] Yan, J.; Lei, Z.; Wen, L.; Li, S. Z. The fastest deformable part model for object detection. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2497–2504, 2014.

[21] Wilt, N. *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Pearson Education, 2013.

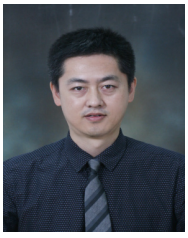
[22] Ryoo, S.; Rodrigues, C. I.; Baghsorkhi, S. S.; Stone, S. S.; Kirk, D. B; Hwu, W.-m. W. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 73–82, 2008.

[23] Sanders, J.; Kandrot, E. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 2010.

[24] Girshick, R. B.; Felzenszwalb, P. F.; McAllester, D. Discriminatively trained deformable part models (release 5). 2012. Available at https://people.eecs.berkeley.edu/∼rbg/latent/.

[25] Everingham, M.; Van Gool, L.; Williams, C. K. I.; Winn, J.; Zisserman, A. The PASCAL visual object classes (VOC) challenge. *International Journal of Computer Vision* Vol. 88, No. 2, 303–338, 2010.

Zhi-Min Zhou is an M.S. candidate in the Department of Automation, Shanghai Jiao Tong University, China. He received his B.S. degree from the University of Electronic Science and Technology of China in 2013. His research interests include image processing, GPU high performance computing, pattern recognition, and computer vision.



Xu Zhao is currently an associate professor in the Department of Automation, Shanghai Jiao Tong University, China. He received his Ph.D. degree in pattern recognition and intelligent systems from Shanghai Jiao Tong University in 2011. He was a visiting scholar in the Beckman Institute for Advanced Science and Technology, the University of Illinois at Urbana-Champaign from 2007 to 2008. He was a postdoctoral research fellow in Northeastern University from 2012 to 2013. His research interests include visual analysis of human motion, machine learning, and image and video processing.