

Efficient Histogram-Based Sliding Window

Yichen Wei
Microsoft Research Asia
yichenw@microsoft.com

Litian Tao
Beihang University
taolitian@cse.buaa.edu.cn

Abstract

Many computer vision problems rely on computing histogram-based objective functions with a sliding window. A main limiting factor is the high computational cost. Existing computational methods have a complexity linear in the histogram dimension. In this paper, we propose an efficient method that has a constant complexity in the histogram dimension and therefore scales well with high dimensional histograms. This is achieved by harnessing the spatial coherence of natural images and computing the objective function in an incremental manner. We demonstrate the significant performance enhancement by our method through important vision tasks including object detection, object tracking and image saliency analysis. Compared with state-of-the-art techniques, our method typically achieves from tens to hundreds of times speedup for those tasks.

1. Introduction

A histogram is a discretized distribution that measures the occurrence frequency of quantized visual features, *e.g.*, pixel intensity/color [6], gradient orientation [9, 17], texture patterns [1] or visual words [14]. Such a representation is a reasonable trade-off between computational simplicity, discriminability and robustness to geometric deformations.

Computing a histogram-based objective function with a sliding window is common in solving computer vision problems such as object detection [1, 17, 5], object tracking [16], and image filtering [22]. The objective function is evaluated on a window sliding over the entire image and this is computationally very intensive. Although there exists efficient local optimum search method [6] for tracking and global optimum search method [5] for detection, such a dense evaluation is often inevitable. For example, in object tracking, one often needs to conduct a full frame search for small, fast-moving objects [3, 16].

Computing a dense confidence map for object detection is of special importance. In spite of significant progresses in object detection recently [2], accurate detection of general objects still remains a challenging task. It is generally be-

lieved that contextual information plays an important role in image understanding. Individual tasks such as object detection, scene segmentation, and geometric inference, should be integrated and enhance each other to resolve the inherent ambiguity existing in each of them. Several computational frameworks [8, 12] have been developed towards this purpose, in which a dense confidence map for object detection is an indispensable step.

The main limitation of a histogram-based sliding window is its high computational cost. For an image of size $n \times n$, a window of size $r \times r$ and a histogram of dimension B , a straightforward method scans n^2 windows, scans r^2 pixels per window to construct the histogram and scans B bins of the histogram to evaluate the objective function¹. The overall complexity $O(n^2(r^2 + B))$ is prohibitive when either n , r or B is large. Several techniques have been proposed to remove the r^2 factor and reduce the complexity to $O(n^2B)$ [11, 22, 16]. When the histogram dimension B is large, such techniques do not scale well. Unfortunately, high dimensional histograms now are commonplace for solving many vision problems, *e.g.*, color histograms for object tracking [6] ($B = 32^3$), LBP [1] for face recognition/detection (B can be hundreds) and bag of visual words for image retrieval [14] or object detection [5, 2] (B is typically hundreds or thousands).

To alleviate the computational cost for high dimensional histograms, we propose an efficient method with a constant complexity in the histogram dimension. As demonstrated later in the paper, for object detection task using support vector machine with a non-linear kernel and bag of visual words model, which are leading techniques [5, 2], our method can achieve up to a hundred times speedup over existing techniques, reducing the computation time from minutes to seconds. In addition, it facilitates the subsequent contextual information fusion [8, 12] and makes such techniques much more practical.

Our method is also helpful for object tracking tasks, where real-time running performance is usually desired. To achieve good running performance, previous methods either

¹We assume a bin-additive objective function. Such functions are a large family and commonly used in vision problems.

use large size features but perform local search [6, 18], or perform full frame search but use small size features [11, 3, 25, 16]. Our method can achieve real-time performance using both high dimensional histogram features and full frame search. To our best knowledge, it is the first time that such a high performance is achieved when both n and B are large.

Our method is inspired by the image smoothness prior, *i.e.*, natural images usually possess smooth appearance. Consequently, the quantized visual features are also spatially coherent and a histogram of a sliding window typically changes slowly. As illustrated in Figure 1, when a window slides by one column, $2r$ pixels change [13] and the number of changed histogram bins, denoted as $2sr$, is usually much smaller. Therefore only a few bins of the histogram need to be updated. Here s is a factor between 0 and 1. Its value depends on the smoothness of the image and indicates how *sparsely* the histogram changes. In our experiments, we observe that for most natural images the range of this value is $s \in [0.03, 0.3]$.

Based on the above observation, *our method performs histogram construction and objective function evaluation in an incremental manner simultaneously, and resulting computational complexity is $O(n^2 \cdot \min(B, 2sr))$* . It subsumes the complexity $O(n^2B)$ of previous methods and will be much faster when $B \gg 2sr$.

2. Previous Fast Computational Methods

Fast histogram computation *Integral histogram* [11] approach computes a histogram over an arbitrary rectangle in $O(B)$ time. Each bin is computed in a constant time via a pre-computed integral image. When $B \ll r^2$, this is very efficient. Because the pre-computation of integral images has $O(n^2B)$ complexity in both time and memory, its usage is limited for large images and high dimensional histograms. For $n = 512, B = 1000$, 1G memory is required and pre-computation takes about one second, which cannot be simply ignored sometimes.

Distributive histogram approach [22, 16] utilizes the *distributivity* property that a histogram over two regions is the sum of two histograms on these regions. As the window slides by one column, the histogram is updated by adding a new column histogram and removing an old one in $O(B)$ time. When a row scanning is finished and the window slides downwards, all the column histograms are updated by removing the top pixel and adding the bottom pixel in $O(1)$ time.

Both above approaches have $O(B)$ complexity in histogram construction and function evaluation. Although histogram construction in [22, 16] can be accelerated several times via SIMD instructions, function evaluation could be much more complex and dominate the computation time, *e.g.*, an SVM classifier with a non-linear kernel. It is crucial to reduce $O(B)$ for such functions.

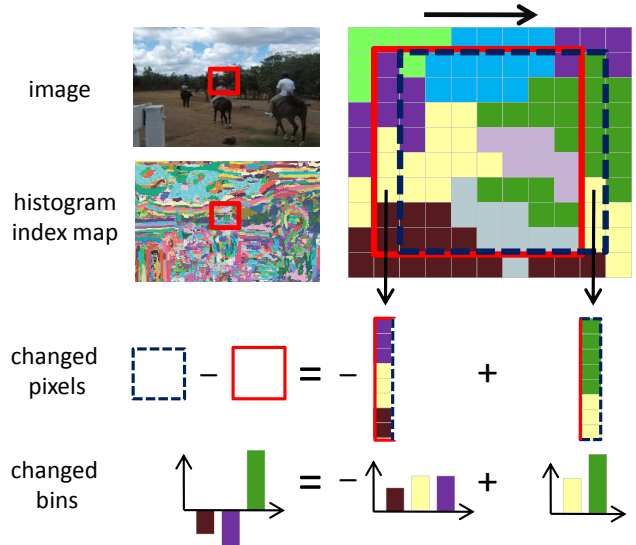


Figure 1. As the window slides by one column, 16 pixels changed but only 3 histogram bins changed. The histogram index map is generated with quantized visual words as explained in section 4.1.

Branch-and-bound Such methods [5, 19, 4] find the global optimal window in sub-linear time by iterating between: 1) branching the parameter space into sub-regions and; 2) bounding the objective function over those regions. The algorithm always splits the region with best bounds until the region contains only one point and can no longer be split, implying that the global optimum is found. A tight bounding function is required for fast convergence. Such bounds have been derived for histogram based functions such as linear SVM classifier and χ^2 similarity [5, 4], and the technique has shown excellent running performance in object detection and image retrieval.

Still, there are a few limitations. Because integral histogram is used to evaluate the bounding function and each bin needs two integral images (twice memory), the memory problem is more serious when B is large. It efficiently finds the global optimal window but does not evaluate other windows. This is appropriate for an accurate objective function with a clear and desirable peak, but hard for a function that is flat (uncertain) or multi-mode, *e.g.*, an image without or with multiple objects in object detection/tracking. In such cases its computational complexity increases and could be as bad as an exhaustive search in the worst case.²

Our method is complementary to the above techniques. It scales well with the histogram dimension in both memory and time. It performs dense function evaluation efficiently and can be applied to objective functions that branch-and-bound method is hard to apply due to the difficulty in obtaining a tight bound, *e.g.*, an SVM with a non-linear kernel.

²It has been shown in [19] that the worst complexity can be improved for a specific objective function

$\mathcal{F}(\mathbf{h})$	$f_b(h_b)$	time
dot product/linear SVM	$h_b \cdot m_b$	3.2
L_2 norm	$(h_b - m_b)^2$	3.5
histogram intersection	$\min(h_b, m_b)$	3.9
L_1 norm	$ h_b - m_b $	4.1
χ^2 distance	$\frac{(h_b - m_b)^2}{h_b + m_b}$	7.1
Bhattacharyya similarity	$\sqrt{h_b \cdot m_b}$	64.8
entropy	$h_b \log h_b$	93.9
SVM with χ^2 kernel	$\sum_{k=1}^{sv} w_k * \chi^2(h_b, m_b^k)$	460

Table 1. Several functions in form (1) and their relative computational costs ($sv = 50$ for SVM). The running time numbers are measured by running these functions millions of times on a modern CPU. $\{m_b\}_{b=1}^B$ is either a model histogram or a support vector to which the feature histogram \mathbf{h} is compared. $\{w_k\}_{k=1}^{sv}$ is the weight of support vectors in SVM.

3. Efficient Histogram-Based Sliding Window

Let $\mathbf{h} = \{h_1, \dots, h_B\}$ denote the feature histogram of the window and $\mathcal{F}(\mathbf{h})$ denote the objective function. As the window slides, it is unnecessary to evaluate \mathcal{F} across the entire histogram but sufficient to only update the affected part. This requires that \mathcal{F} is *incrementally computable*, i.e., there exists a function \mathcal{F}' simpler than \mathcal{F} , such that

$$\mathcal{F}(\mathbf{h} + \delta\mathbf{h}) = \mathcal{F}(\mathbf{h}) + \mathcal{F}'(\delta\mathbf{h}, \mathbf{h}).$$

Therefore incremental computation of $\mathcal{F}'(\delta\mathbf{h}, \mathbf{h})$ is more efficient than re-evaluating $\mathcal{F}(\mathbf{h} + \delta\mathbf{h})$.

In this paper, we study bin-additive functions \mathcal{F} , i.e., that can be expressed as summation³ of functions defined on individual bins,

$$\mathcal{F}(\mathbf{h}) = \sum_{b=1}^B f_b(h_b). \quad (1)$$

Table 1 summarizes several functions in this family that are commonly used in vision problems, e.g., various bin-to-bin histogram distances or so called *quasi-linear* histogram kernels [2].

It is easy to see that \mathcal{F} is incrementally computable,

$$\begin{aligned} \mathcal{F}(\mathbf{h} + \delta\mathbf{h}) &= \sum_{b=1}^B f_b(h_b + \delta h_b) \\ &= \underbrace{\sum_{b=1}^B f_b(h_b)}_{\mathcal{F}(\mathbf{h})} + \underbrace{\sum_{b, \delta h_b \neq 0} (f_b(h_b + \delta h_b) - f_b(h_b))}_{\mathcal{F}'(\delta\mathbf{h}, \mathbf{h})}. \end{aligned}$$

Let $|\delta\mathbf{h}|$ be the number of non-zero entries in $\delta\mathbf{h}$, evaluation of \mathcal{F}' requires $2|\delta\mathbf{h}|$ evaluations of $f(\cdot)$. By storing

³Strictly speaking, "summation" should be "function of summation". We use the former for simplicity.

```

1: initialize each column histogram  $\mathbf{c}^x$  with first  $r$  pixels
2: initialize histogram  $\mathbf{h} = \sum_{x=1}^r \mathbf{c}^x$ 
3: initialize function  $\{d_b = f_b(h_b)\}_{b=1}^B$  and  $\mathcal{F} = \sum_b d_b$ 
4: for  $y = 1$  to  $n - r$ 
5:   for  $x = 1$  to  $n - r$ 
6:     foreach  $b \in \delta\mathbf{h}(\delta\mathbf{h} = \mathbf{c}^{x+r-1} - \mathbf{c}^{x-1})$  that  $\delta h_b \neq 0$ 
7:        $h_b \leftarrow h_b + \delta h_b$ 
8:        $\mathcal{F} \leftarrow \mathcal{F} + f_b(h_b) - d_b$ 
9:        $d_b \leftarrow f_b(h_b)$ 
10:    end
11:   write  $\mathcal{F}$  to output image at  $(x, y)$ 
12:  end
13: update all column histograms  $\mathbf{c}^x$  by adding pixel
    at  $(x, y + r)$  and removing pixel at  $(x, y)$ 
14: end

```

Figure 2. Efficient histogram-based sliding window (EHSW-D).

and maintaining those values $\{d_b = f_b(h_b)\}_{b=1}^B$, only $|\delta\mathbf{h}|$ evaluations of f are needed.

EHSW-D Our first algorithm is denoted as EHSW-D (Efficient Histogram-based Sliding Window-Dense). Similarly as in [16], for each column $x \in [1..n]$, a column histogram \mathbf{c}^x of r pixels is maintained. When the window slides by one column, the increment $\delta\mathbf{h}$ is computed as

$$\delta\mathbf{h} = \mathbf{c}^+ - \mathbf{c}^-,$$

where \mathbf{c}^+ and \mathbf{c}^- are the new and old column histograms, respectively. The algorithm is summarized in Figure 2.

Let t_A be the computational cost of arithmetic operations (addition/subtraction/compare) and t_V the cost of evaluating f . The computational cost for each window is as follows: a column histogram has one pixel added and one pixel removed ($2t_A$) and \mathbf{h} is updated via adding and removing a column histogram ($2Bt_A$). In function evaluation, all bins are traversed (Bt_A) but f is evaluated only for non-zero bins $2sr$ times ($2srt_V$).

EHSW-S When $B \gg r$, most entries in \mathbf{c} are zero and traversing the array to find non-zero entries becomes very inefficient. It is better to use a sparse structure that retains only non-zero entries in \mathbf{c} .

The sparse structure should allow efficient insertion, removal and query of a bin entry (sorting is not needed). A natural choice is to use a hash table with bin as key and its value as content. As insertion/removal involves expensive memory operations, we implement a specialized hash table that avoids such operations. A list of $(bin, value)$ pairs is maintained for non-zero bins. B buckets are allocated in advance and each bucket holds a pointer pointing to the corresponding element in the list (the pointer corresponding to an empty bin is null). Bucket collision never happens and query of a bin is done in $O(1)$ time via its bucket pointer. As the list contains at most r pixels, we pre-allocate

Method	Construct \mathbf{h}	Evaluate \mathcal{F}
EHSW-D	$(2 + 2B)t_A$	$Bt_A + 2srt_V$
EHSW-S	$(2 + (r + 6)c)t_A$	$2srt_A + 2srt_V$
Integral	$3Bt_A$	Bt_V
Distributive	$(2 + 2B)t_A$	Bt_V

Table 2. Computational complexity (per window) of our methods, integral histogram [11] and distributive histogram [16]. t_A and t_V are the computational cost of arithmetic operation and $f(\cdot)$ evaluation, respectively. For different $f(\cdot)$, t_V varies a lot and could be much more expensive than t_A (see Table 1). s and c are image smoothness measurements, $s \leq 1, c \leq 1$.

and retain a buffer that holds r (*bin, value*) pairs. Consequently, insertion/removal directly fetches/returns memory units from/to the buffer ($rt_A/2$, the average time of linear scanning r memory units) and updates the list pointers ($2t_A$) and bucket pointer (t_A). Compared to a standard STL hash map, our implementation is much faster even with a large r .

The new algorithm using sparse representation is denoted as EHSW-S. It is slightly modified from EHSW-D in Figure 2 and its computational cost for each window is as follows: in histogram update (line 13), the pixel is firstly queried in the hash table ($2t_A$ for two pixels). Insertion or removal is invoked when the incremented bin does not exist or decremented bin becomes empty, *i.e.*, when the added or removed pixel is different from all the other pixels in the column. Let $c \in [0, 1]$ (coherence) denote such probability, the cost of insertion and removal is then $(r + 6)ct_A$ ($((r/2 + 3)ct_A$ for one pixel). Note that c is different from s but their values are usually similar as observed in our experiments. In function evaluation, the two sparse column histograms are traversed separately (line 6-10 is repeated for \mathbf{c}^{x+r} and \mathbf{c}^{x-1}). Update of h_b (line 7) and evaluation of f are performed $2sr$ times ($2srt_A$ and $2srt_V$, respectively).

Computational Complexity Table 2 summarizes the computational complexity of several methods. We focus on the per window cost as it is the dominant factor. Integral histogram approach [11] computes each bin of the histogram with three arithmetic operations on an integral image. Distributive approach [16] constructs the histogram in the same way as EHSW-D. Both of them need to evaluate the objective function over all the histogram bins.

To compare the running time of those methods, we perform synthetic experiments on a PC with a 2.83G Pentium 4 CPU and 2G memory. As performance of our methods depends on s, c , we fix $s = c = 0.25$ in the experiments. This is a reasonable setting as will be seen in real experiments. The image size is fixed as 512×512 . The computational time of different objective functions, histogram dimensions and window sizes, is summarized in figures 3 and 4.

From the computational complexity in Table 2 and running time in figures 3 and 4, we can draw several general

conclusions: 1) integral histogram is not advantageous for sliding window computation⁴; 2) EHSW becomes more advantageous as histogram dimension increases, *i.e.*, when B is very small, $\text{Dist.} > \text{EHSW-D} > \text{EHSW-S}$; when B is large, $\text{EHSW-S} > \text{EHSW-D} > \text{Dist.}$; 3) EHSW becomes more advantageous as the objective function complexity increases ($\text{L1Norm} \rightarrow \text{Entropy} \rightarrow \text{SVM Chi-square}$).

Note that the sparse representation used in EHSW-S could also be used in distributive approaches [22, 16]. However, it will not be as helpful as in our method because those approaches do not exploit the sparseness of histogram increment to update the objective function.

Memory Complexity Integral histogram stores an integral image for each bin and consumes n^2B memory. Distributive histogram approach stores a histogram for each column with totally nB memory. Compared with distributive approach, our method stores additional B values $\{d_b\}_{b=1}^B$. Therefore EHSW-D consumes $(n + 1)B$ memory. For EHSW-S, each column histogram stores B buckets and a list buffer of r entries, with each entry consisting of a pair of values and a pointer. Therefore, the total memory is $(n + 1)B + 3nr$.

3.1. Extensions

More window shapes For a non-square window of size $r_w \times r_h$, the complexity factor r in Table 2 is reduced to $\min(r_w, r_h)$ by sliding the window along the longer side, *i.e.*, horizontally when $r_w > r_h$ and vertically when $r_w < r_h$.

The idea of incremental histogram construction is not limited to rectangular windows but can also be applied to other shapes [16]. Similarly, our method can also be extended to such windows. Extension of our method from 2D image to 3D video is also straightforward.

Histograms on a spatial grid A histogram only loosely correlates the visual features with spatial coordinates. This loose relationship can be enhanced by splitting the window into a grid and computing a histogram for each cell [20]. The final objective function is the sum of functions defined on individual cells. This can be done by running multiple sliding windows separately for each cell and summing up the function values.

Assuming all cells are of the same size, a better approach is to run only one sliding window. In this case, the histogram is constructed only once but multiple functions of different cells are evaluated simultaneously (line 8,9 in Figure 2 is repeated for each function). Each result is written to an output image with an offset (line 11) determined by the cell locations.

⁴This is not a criticism as integral histogram is much more general than computing histogram of a sliding window. Also integral histogram naturally allows a parallel computation while others do not due to their incremental manner.

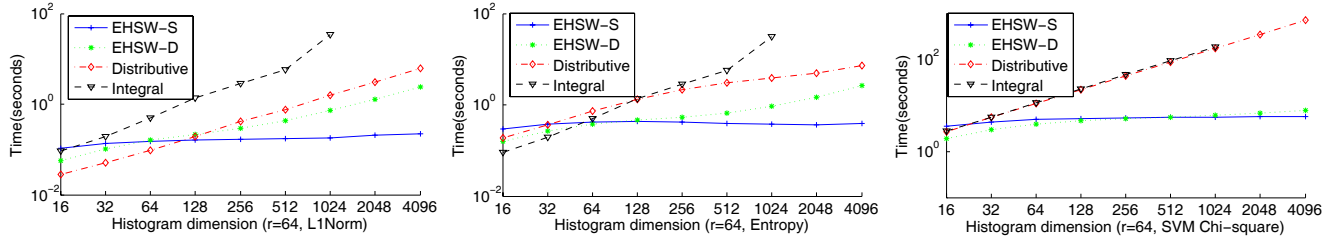


Figure 3. Running time with varying histogram dimension B and different objective functions. Note the logarithmic scale of both axes. Window size is fixed as $r = 64$. We do not test integral histogram approach when $B > 1024$ due to the physical memory limitation. EHSW-S is independent of B while other approaches have a linear dependency.

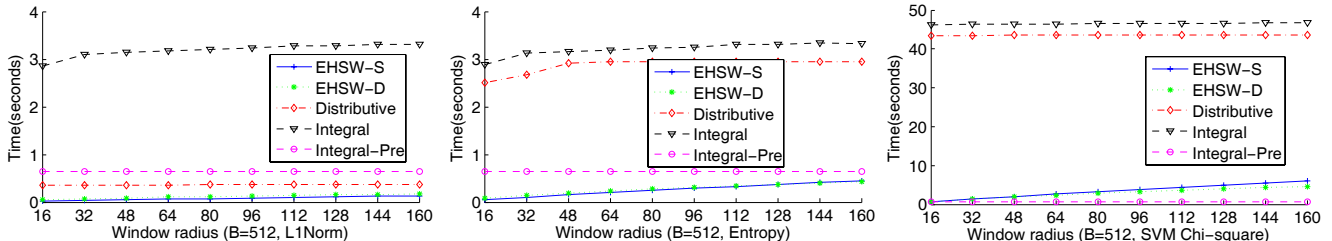


Figure 4. Running time with varying window size r and different objective functions. Histogram dimension is fixed as $B = 512$, a typical size of an 8^3 color histogram or a visual word code book. Our methods are linear in r but still outperform other approaches. Pre-computation time of integral histogram is also shown for a reference.

4. Applications

4.1. Object Detection

Computing a confidence map of object detection is important in computational models that combine multiple tasks and fuse contextual information [8, 12]. Hoiem et al.’s method [8] requires that each task outputs *intrinsic images*. In object detection task, those are the confidence maps of individual objects. Heitz et al.’s approach [12] learns cascaded classification models. A high level classifier computes features directly on the confidence map of a low level object detector.

Figure 5 shows exemplar results in our experiment. It is difficult to determine the accurate location/scale of individual objects from only local appearance. Exploiting contextual relationship is helpful to resolve such ambiguities.

Support vector machine classification and bag of visual words model are leading techniques in object detection [5, 2]. A large code book and a complex kernel function typically give rise to good performance. Our method can compute a confidence map much more efficiently than other techniques in this setting.

Experiment We use the challenging PASCAL VOC 2006 dataset [15] and test four object classes, *i.e.*, person, motorbike, bicycle and horse, as their contextual relationships can often be observed, *e.g.*, a person is above a motorbike or a horse.

In this experiment, we aim to verify the efficiency of our method, and our implementation uses standard and state-

of-the-art techniques. The object is represented as image patches densely sampled at regular grid points [17, 2]. Each patch is described by its SIFT descriptor [9] and average RGB colors. The patches are quantized into a code book of K visual words created by k-means clustering on 250,000 randomly selected patches. A 2×2 grid [20] is used to incorporate the spatial relation of the object parts and the object is represented as concatenation of histograms of visual words on the cells. An SVM classifier is trained using manually labeled object windows as positive examples and randomly sampled windows from negative images as negative examples.

Performance Detection performance is measured by *average precision (AP)*. It is the average of precisions at different levels of recall and used in PASCAL competition [15]. We tested different code book sizes and SVMs using a linear kernel and a non-linear χ^2 kernel [2]. Results are reported in Figure 6. We can draw two conclusions: 1) A χ^2 kernel significantly outperforms a linear kernel; 2) Increasing the code book size improves the performance as a small size code book has too coarse quantization and lacks discriminability. In our experiments, performance stops increasing after K reaches 1600.

Using $K = 1600$ and χ^2 kernel, our method (EHSW-S) is significantly faster than distributive approach [16]. The speedup factor depends on the image smoothness and is from tens to hundreds in our experiments. Statistics of smoothness values and speedup factors of 2686 test images, as well as several example images, are illustrated in Figure

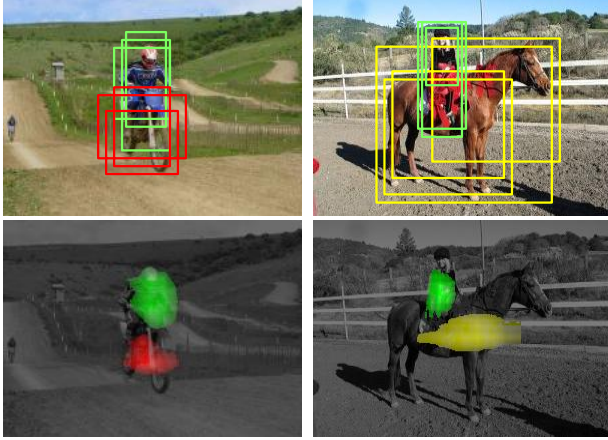


Figure 5. Top: detected objects of different locations/scales. Bottom: confidence maps of different objects.

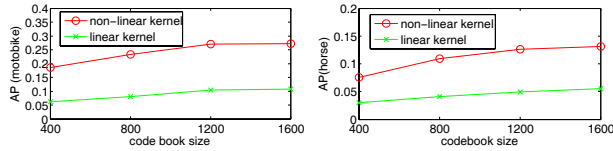


Figure 6. Average precisions of ‘motorbike’(left) and ‘horse’(right) classes. Class ‘bicycle’ has a similar performance as ‘motorbike’. For ‘person’ class, the best AP is 0.049 with $K = 1600$ and a non-linear kernel. Our implementation uses standard ingredients and achieves comparable performance in PASCAL 2006 detection competition [15].

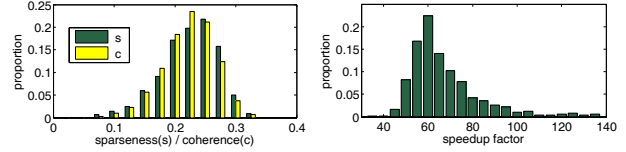
7. For an image size of about 300×300 , a window size of 140×60 and an SVM with 1000 support vectors, our method typically takes a few seconds to generate a confidence map.

The optimal code book size is affected by the patch descriptor. With high dimensional and more informative patch descriptors, a large code book is required, *e.g.*, the performance of object detection experiments in [10] keeps increasing even when K reaches 10,000. Our method would be more advantageous in such a setting.

The non-linear kernel (χ^2) evaluation has a complexity linear in the number of support vectors. Recently Maji et. al. [21] developed approximation technique that reduces this linearity to a constant. Therefore non-linear kernel classification becomes much faster at the cost of inexact evaluation. Their technique is complementary to our method and can be easily combined.

4.2. Object Tracking

Local tracking methods using color histogram [6, 18] have demonstrated good performance when the target has small frame-to-frame motion. Nevertheless, they are prone to failure in case of fast motion and distracting background



$s=0.102$, speedup = 96



$s=0.229$, speedup = 61



$s=0.327$, speedup = 47

Figure 7. Top left: distribution of smoothness measurements s and c . Top right: distribution of speedup factors of our method EHSW-S over distributive approach [16]. Bottom: three test images, their visual word maps and statistics.

clutter. Full frame tracking can solve these problems with exhaustive search in the entire frame. However, previous such approaches [11, 3, 25, 16] need to use small size features when a high running performance is required.

Our method can achieve high running performance using both full frame search and more effective high dimensional histogram features. This is illustrated in a challenging example shown in Figure 8. The three target skiers are similar, moving fast and occluding each other. The background also contains similar colors. There are a lot of local optima that will fail local tracking methods. In full frame tracking, we compare results using a 16^3 RGB histogram [6] and a 16 bins intensity histogram.⁵ The likelihood maps are displayed in the middle and bottom row in Figure 8, respectively, with the best 10 local optima (dashed rectangles) overlaid. To identify the effectiveness of using different features, each local optimum is labeled as correct (red rectangles) if its overlap with ground truth is more than 50%, or wrong (green rectangles) otherwise.

⁵We have also tried the 16 bins histogram in hue channel as used in [16], but the result is worse than that of using intensity.

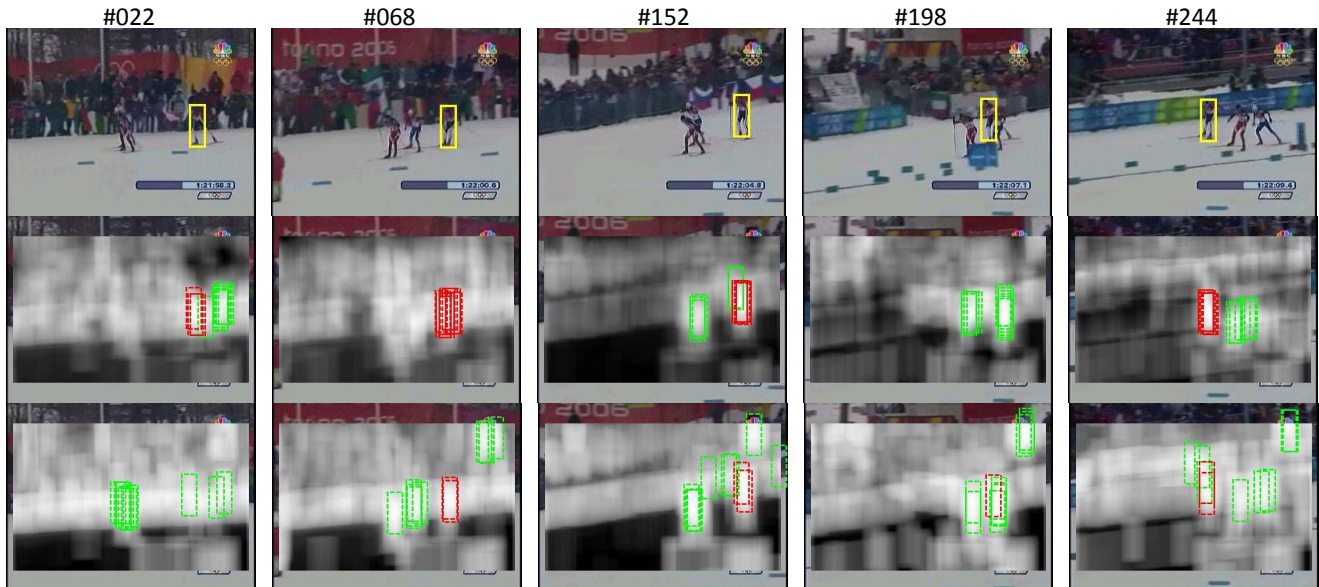


Figure 8. A tracking example of 250 frames. Top: ground truth tracking results of the right skier (yellow rectangles). Middle: likelihood maps using 16^3 bins RGB histogram. Bottom: likelihood maps using 16 bins intensity histogram. On each likelihood map, 10 best local optima are overlaid and labeled as correct (red dashed rectangles) or wrong (green dashed rectangles) based on their overlap with ground truth.

As can be clearly observed, the intensity histogram poorly discriminates the target (right skier) from the background and generates noisy likelihood maps with many false local optima. The color histogram generates cleaner likelihood maps with better local optima. The average numbers of correct local optima per frame using color and intensity histograms are 4.0 and 1.8, respectively.

Performance With intensity histogram, both distributive approach [16] and EHSW-S run at high frame rates, 60 and 50 fps (frames per second), respectively. With color histogram, distributive approach slows down to 0.3 fps while EHSW-S still runs in real time 25 fps, obtaining 83 times speedup.

Bhattacharyya similarity [6] is used as likelihood. We have also tested $L_{1(2)}$ norms but found they are worse than Bhattacharyya similarity. The image size is 320×240 , target size is 53×19 and average sparseness value is $s = 0.27$, $c = 0.29$ for color histogram.

Discussion It is worth noting that the global optima are wrong on some frames even using color histogram. This indicates the insufficiency of only searching the global optimum on each frame. A solution is to impose the temporal smoothness constraint and find an optimal object path across frames via dynamic programming [3, 25], using local optima on each frame as object candidates. With such global optimization technique, better local optima are more likely to generate correct results. Using color histograms, there are 14 frames where the correct object is missed, *i.e.*,

none of the 10 best local optima is correct, mostly due to occlusion (*e.g.*, frame 198 in Figure 8). While intensity histogram is used, this number is 35.

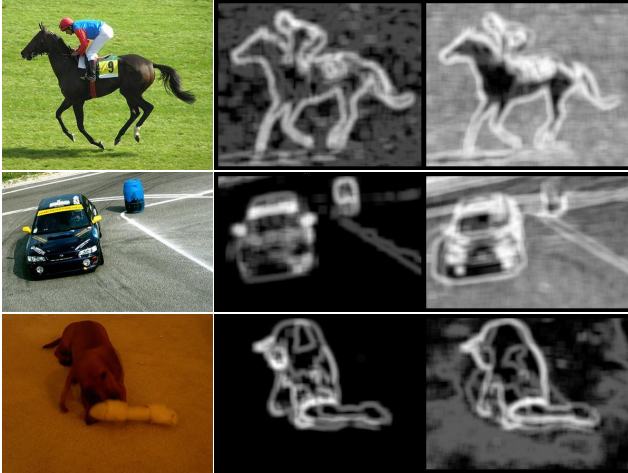
4.3. Feature Entropy for Image Saliency Analysis

Image saliency analysis problem is to find visually informative parts in images. It is important in object detection, recognition and image understanding. Many methods have been proposed in the literature (see [23] for a review) while the task still remains very challenging.

Entropy is the randomness of a distribution and can serve as an information measurement of visual features. It is directly used as image saliency in [7] and can be combined with other methods, *e.g.*, the multi-cue learning framework [24].

Our method can compute an entropy map efficiently. We tested a subset of images from the public data set [24] using 16^2 bins *ab* color histogram and 16 bins intensity histogram. We observed that the result using color histogram is clearly more visually informative than that using intensity histogram in most images.

A few examples are shown in Figure 4.3. It is worth noting that using color histogram is not only better, but also faster with our method as the image smoothness is stronger in the color space. Most of the test images have the sparseness value $s \in [0.03, 0.1]$ for color histogram and our method is much faster than distributive approach [16].



image(size)	feature(B)	s	EHSW-S	Dist. [16]
horse (400 × 322)	color(16 ²)	0.093	32 ms	158 ms
	intensity(16)	0.184	62 ms	54 ms
car (400 × 266)	color(16 ²)	0.044	15 ms	121 ms
	intensity(16)	0.166	54 ms	43 ms
dog (400 × 300)	color(16 ²)	0.022	11 ms	133 ms
	intensity(16)	0.060	20 ms	23 ms

Figure 9. Top: from left to right are images, entropy maps using 16^2 bins ab color histogram and entropy maps using 16 bins intensity histogram. Sliding window is 16×16 . Bottom: statistics. Note that EHSW-S with color histogram is even faster than that with intensity histogram, because the image appearance is smoother in ab color space than in intensity space, *i.e.*, sparseness s is smaller.

5. Conclusions

We present an efficient method for computing a histogram-based objective function with a sliding window. The high efficiency benefits from a natural prior on spatial coherence of image appearance. The efficiency of the proposed method has been demonstrated in several applications, where a significant speedup is achieved with comparison to state-of-the-art methods. Future work is to extend our method for more complex objective functions, such as earth mover’s distance, bilateral filters, etc.

References

- [1] A.Hadid, M.Pietikainen, and T.Ahonen. A discriminative feature space for detecting and recognizing faces. In *CVPR*, 2004.
- [2] A.Vedaldi, V.Gulshan, M.Varma, and A.Zisserman. Multiple kernels for object detection. In *ICCV*, 2009.
- [3] A. M. Buchanan and A. W. Fitzgibbon. Interactive feature tracking using k-d trees and dynamic programming. In *CVPR*, 2006.

- [4] C.H.Lampert. Detecting objects in large image collections and videos by efficient subimage retrieval. In *ICCV*, 2009.
- [5] C.H.Lampert, M.B.Blaschko, and T.Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *CVPR*, 2008.
- [6] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *CVPR*, 2000.
- [7] C.Rother, L.Bordeaux, Y.Hamadi, and A.Blake. Autocolage. *Proceedings of ACM SIGGRAPH*, 2006.
- [8] D.Hoiem, A.Efros, and M.Hebert. Closing the loop in scene interpretation. In *CVPR*, 2008.
- [9] D.Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.
- [10] F.Moosmann, B.Triggs, and F.Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *NIPS*, 2006.
- [11] F.Porikli. Integral histogram: A fast way to extract histograms in cartesian spaces. In *CVPR*, pages 829–836, 2005.
- [12] G.Heitz, S.Gould, A.Saxena, and D.Koller. Cascaded classification models: Combining models for holistic scene understanding. In *NIPS*, 2008.
- [13] T. Huang, G. Yang, and G. Tang. A fast two-dimensional median filtering algorithm. *IEEE Trans. Acoust., Speech, Signal Processing*, 27(1):13–18, 1979.
- [14] J.Sivic and A.Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, 2003.
- [15] M.Everingham, A.Zisserman, C.Williams, and L.Gool. The pascal visual object classes challenge 2006 results. Technical report, 2006.
- [16] M.Sizintsev, K.G.Derpanis, and A.Hogue. Histogram-based search: a comparative study. In *CVPR*, 2008.
- [17] N.Dalal and B.Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.
- [18] P. Pérez, C. Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking. In *ECCV*, 2002.
- [19] S.An, P.Peursum, W.Liu, and S.Venkatesh. Efficient algorithms for subwindow search in object detection and localization. In *CVPR*, 2009.
- [20] S.Lazebnik, C.Schmid, and J.Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006.
- [21] S.Maji, A.Berg, and J.Malik. Classification using intersection kernel support vector machine is efficient. In *CVPR*, 2008.
- [22] S.Perreault and P.Hebert. Median filtering in constant time. *Trans. Image Processing*, 16(9):2389–2394, 2007.
- [23] T.Huang, K.Cheng, and Y.Chuang. A collaborative benchmark for region of interest detection algorithms. In *CVPR*, 2009.
- [24] T.Liu, J.Sun, N.Zheng, X.Tang, and H.Shum. Learning to detect a salient object. In *CVPR*, 2007.
- [25] Y.Wei, J.Sun, X.Tang, and H.Shum. Interactive offline tracking for color objects. In *ICCV*, 2007.