A Square-Root Sampling Approach to Fast Histogram-Based Search

Huang-Wei Chang
Stanford University
huangwei@stanford.edu

Hwann-Tzong Chen
National Tsing Hua University, Taiwan
htchen@cs.nthu.edu.tw

Abstract

We present an efficient pixel-sampling technique for histogram-based search. Given a template image as a query, a typical histogram-based algorithm aims to find the location of the target in another large test image, by evaluating a similarity measure for comparing the feature histogram of the template with that of each possible subwindow in the test image. The computational cost would be high if each subwindow needs to compute its histogram and evaluate the similarity measure. In this paper, we adopt the probability-product kernels as the similarity measures, and show that the computation of histograms and the evaluation of the kernel-based similarities can be integrated through a sampling approach. Specifically, we present a square-root sampling method to avoid the computation of histograms, and meanwhile, reduce the number of pixels required for evaluating the similarity measure. The proposed approximation algorithm is time- and memory-efficient. The time complexity of computing the similarity for each subwindow is $O(1)$. The memory requirement of our algorithm is not in proportion to the size of the template or the number of histogram bins, which allows the use of more distinctive image representations for better search accuracy.

1. Introduction

Histogram-based image representations such as color histograms or histograms of oriented gradients are widely used in computer vision. The advantages of using histogram representations have been shown through various applications, *e.g.*, image retrieval [10, 11, 21], image segmentation [6, 16, 19], shape analysis [1], object detection [3, 5, 7], and tracking [4, 17]. For image data, a histogram can be considered as a discrete probability distribution that characterizes the local appearance by the low-level image features. Most popular image features for generating the histogram are color and gradient. To compute, for example, a color histogram of an image patch, a standard procedure would first convert the color information of each pixel into a quantized value, and then the quantized value is mapped to an index of a corresponding histogram bin. The number of pixels assigned to each bin is accumulated over the whole image patch. A color histogram is generally robust to rotation, deformation, and small variations in lighting condition.

In this work, we address the problem of searching a template in a test image using the histogram-based representations. The computational complexity of the search problem mainly depends on two issues: *i)* computing the histogram at a candidate subwindow, and *ii)* running through all possible subwindows to compare the histogram of each subwindow with the one of the template. Two commonly-used similarity measures for comparing histograms are the χ^2 test statistic, *e.g.* [1], and the Bhattacharyya coefficient, *e.g.* [4]. Both similarity measures involve some nonlinear terms associated with the histogram of the candidate subwindow, and both yield the global extrema when two histograms are identical.

Previous algorithms that attempt to speed up the histogram-based search have addressed the two aforementioned issues from several perspectives. Huang *et al*. [12] address the redundancy in the computation of histograms, and present an algorithm that updates the histograms of neighboring subwindows by adding the rightmost column-histogram and subtracting the leftmost one. The *integral histogram* presented by Porikli [18] generalizes the idea of *integral image* [22] to histogram computation. The computation of the integral histogram is very fast if the histogram has only a small number of bins. However, the computational cost and the requirement of memory for integral histogram are proportional to the number of histogram bins. This limitation makes the integral histogram hard to be applied to situations that require the number of bins to be large so that the histogram could yield a more distinctive representation. More recently, Sizintsev *et al*. [20] present the *distributive histogram* which is based on fast median filtering. The distributive histogram shows better performance than previous approaches, but the computational cost and memory requirement of the distributive histogram are still in proportion to the number of histogram bins. For a search task involving a color histogram of thousands of bins, the scale factor of bin number would override the factor of im-

age size, and thus the efficiency would be degraded.

Regarding the issue of comparing the histogram of the template with the ones of candidate subwindows, it would be time-consuming if an algorithm straightforwardly evaluates the similarity measure over all possible subwindows in the test image. If prior information about the probable target locations is available, tracking can be taken into consideration for reducing unnecessary evaluations of similarity measures, *e.g.*, [4, 17]. Only local search is needed if the target is assumed to be nearby. More recently , Lampert *et al*. present the *efficient subwindow search* (ESS) algorithm [15] that uses the branch-and-bound technique to evaluate the similarity measures on candidate locations according to their priorities. To gain the speedup from the ESS algorithm, one needs to find a good bound on the similarity measure; the bound should be easy to evaluate and also tight enough to ensure quick convergence. As a result, the choices of histogram representations and similarity measures may be restricted. For example, it is not trivial to find a good bound for the 'normalized' color histogram, with Bhattacharyya coefficient as the similarity measure. Without a good bound, the ESS algorithm will perform just as an exhaustive search. Standard multi-resolution, coarse-to-fine techniques such as pyramid methods [9] are also often used to speed up the search, and in general, all search algorithms may benefit from the use of multi-resolution techniques.

This paper presents a new sampling approach to the problem of histogram-based search. The key idea of the proposed approximate sampling algorithm is to avoid the direct computation of histograms in the test image, and also to reduce the number of pixel positions required for evaluating the similarity measure. Our approximation algorithm is time- and memory-efficient. Unlike the integral histogram, the storage requirement of our approach is not in proportion to the number of histogram bins. Hence our algorithm allows the approximate search of large templates and also the use of a normalized histogram with a sufficient number of bins, which would yield a more distinctive representation and in turn improve the accuracy of search. We show that, by using the probability product kernels [13] as the similarity measures for comparing normalized histograms, it is not necessary to compute the histogram of each subwindow if our goal is merely to evaluate the similarity measure. The objective of this paper is to provide an alternative perspective on histogram-based search, through the derivation of a square-root pixel-sampling procedure that integrates, and thus simplifies, the computation of histograms and the evaluation of similarity measures.

2. Notation and Problem Setting

Our goal is to do fast template matching by comparing two histograms. Suppose we are given a template $\mathcal{T}$ and we have computed its histogram representation from low-level image features, *e.g.*, color or gradient. The values of image features are quantized into discrete numbers corresponding to the bin indices of the histogram. The number of pixels having the same bin-index is accumulated over the whole template to build the histogram. With the histogram-based representation, we hope to search inside a test image and find the subwindows that have similar histograms to the template $\mathcal{T}$. We denote the histogram of $\mathcal{T}$ as $h_{\mathcal{T}}$, and the number of pixels inside $\mathcal{T}$ as $|\mathcal{T}|$, which is also equal to the sum over bins, $|\mathcal{T}| = \sum_k h_{\mathcal{T}}(k)$. Let p be the normalized version of $h_{\mathcal{T}}$ given by $p = h_{\mathcal{T}}/|\mathcal{T}|$, so we can consider p as a discrete distribution, with $\sum_k p(k) = 1$. Furthermore, let the histogram obtained at a candidate subwindow $\mathcal{R}$ in the test image be $h_{\mathcal{R}}$ and its normalized version be q.

Given two normalized histograms p and q, we need to use some similarity measure for comparing p and q. The similarity measure reflects how likely the subwindow $\mathcal{R}$ will contain the target we try to find. We discuss below the similarity measures that are used in this work.

3. Probability Product Kernels

We use the probability product kernels as the similarity measures for comparing two discrete distributions. Similar to [13] on the continuous distributions, we may define a family of kernels $K_\rho : \mathcal{P} \times \mathcal{P} \rightarrow \mathbb{R}$ on the space of normalized discrete distributions over some index set Ω. Specifically, the general *probability product kernel* between normalized histograms p and q is defined as

$$K_\rho(p, q) = \sum_k p(k)^\rho q(k)^\rho \,. \tag{1}$$

It is easy to show that such a similarity measure is a valid kernel, since for any $p_1, p_2, \ldots, p_n \in \mathcal{P}$, the Gram matrix $\mathbf{K}$ consisting of elements $K_{ij} = K_\rho(p_i, p_j)$ is positive semidefinite:

$$\sum_i \sum_j \alpha_i \alpha_j K_\rho(p_i, p_j) = \sum_k \left(\sum_i \alpha_i \, p_i(k)^\rho \right)^2 \geq 0 \,, \tag{2}$$

for $\alpha_i, \alpha_2, \ldots, \alpha_n \in \mathbb{R}$.

Different ρ values correspond to different types of probability product kernels. For $\rho = 1$, we have

$$K_1(p, q) = \sum_k p(k)\, q(k) = \mathbb{E}_p\left[q(k)\right] = \mathbb{E}_q\left[p(k)\right] \,, \tag{3}$$

which can be referred to as the *expected likelihood kernel* [13]. Furthermore, for $\rho = 1/2$, we obtain

$$K_{\frac{1}{2}}(p, q) = \sum_k \sqrt{p(k)\, q(k)} \,, \tag{4}$$

which is known as the Bhattacharyya coefficient [2], or can be referred to as the Bhattacharyya kernel [13].

Note that the Bhattacharyya kernel has values within $[0, 1]$. In particular, the Bhattacharyya kernel is at its maximum when p and q are identical, since by that we would get $\sum_k \sqrt{p(k)\,q(k)} = \sum_k \sqrt{p(k)\,p(k)} = \sum_k p(k) = 1$. The Bhattacharyya kernel has been shown to be a very good similarity measure for discrete distributions, and has been successfully used in several vision applications, *e.g.*, visual tracking [4], [17], and handwritten digits recognition [14]. On the other hand, the expected likelihood kernel is not as effective as the Bhattacharyya kernel in comparing distributions because the expected likelihood kernel could be maximized when one distribution is peaked at the mode of the other, although the two distributions are not identical. In this work we consider the Bhattacharyya kernel as the main similarity measure for comparing discrete distributions.

4. Fast Evaluation of the Bhattacharyya Kernel over an Image

The task of locating a given template inside a test image will be accomplished through evaluating the Bhattacharyya kernel over the template histogram and the histogram of each possible subwindow inside the test image. A bottleneck of the efficiency is due to the iterative computation of the histogram for each possible subwindow. The key idea of our approach is to avoid the computation of histograms, and to enable the direct approximation to the Bhattacharyya kernel over the pre-processed test image.

Before introducing our method of evaluating the Bhattacharyya kernel, for clarity, we begin with the simpler case of computing the expected likelihood kernel. As mentioned previously, the expected likelihood kernel is not as effective as the Bhattacharyya kernel. Nevertheless, the expected likelihood kernel is rather easy to compute using the technique of back-projection [21]. It is worth noting that we can obtain the exact rather than approximate results on the expected likelihood kernel using back-projection. Recall that the expected likelihood kernel is defined by $K_1(p, q) = \sum_k p(k)\,q(k)$. We may compute $h_{\mathcal{T}}$ and p in advance with the given template image. For the kth bin of $h_{\mathcal{T}}$, its value is obtained by counting the pixels that are mapped to the index k:

$$h_{\mathcal{T}}(k) = \sum_{\mathbf{x}\in\mathcal{T}} \delta[b(\mathbf{x}) - k]\,, \tag{5}$$

where $\delta[n]$ is the Kronecker delta, with $\delta[n] = 1$ if $n = 0$, and $\delta[n] = 0$ otherwise. The mapping function $b(\mathbf{x})$ maps a pixel $\mathbf{x}$ to its corresponding bin index.

The computation of the expected likelihood kernel can be expressed as

$$\begin{aligned} K_1(p, q) &= \sum_k p(k)\,q(k) \\ &= \sum_k p(k) \left(\frac{1}{|\mathcal{R}|} \sum_{\mathbf{x}\in\mathcal{R}} \delta[b(\mathbf{x}) - k] \right) \\ &= \frac{1}{|\mathcal{R}|} \sum_{\mathbf{x}\in\mathcal{R}} \sum_k p(k)\,\delta[b(\mathbf{x}) - k] \\ &= \frac{1}{|\mathcal{R}|} \sum_{\mathbf{x}\in\mathcal{R}} p\left(b(\mathbf{x})\right)\,. \end{aligned} \tag{6}$$

Therefore, the computation of the expected likelihood kernel can be done by taking the sum of values $p(b(\mathbf{x}))$ within subwindow $\mathcal{R}$. As a result, we are able to use the following algorithm to evaluate the expected likelihood kernel over the whole test image. The output of the following algorithm is a support map that reflects the similarity between the template and each subwindow.

Fast Evaluation of Expected Likelihood Kernel

1. Compute the normalized histogram p for the template.
2. Quantize the image features of each pixel $\mathbf{x}$ in the test image to obtain its bin index $b(\mathbf{x})$.
3. Create an auxiliary image as large as the test image. Assign the value $p(b(\mathbf{x}))$ to the pixel at the corresponding position in the auxiliary image.
4. Build the integral image [22] of the auxiliary image. The kernel value for each subwindow can be evaluated by adding and subtracting four values at the corner of the subwindow using the integral image technique. Create a support map consisting of the kernel values as the output.

In the following we introduce a square-root rejection sampling scheme for evaluating the expected likelihood kernel. Unlike the previous algorithm, the auxiliary image in this case contains only a small number of pixels whose values are nonzero. The algorithm can be considered as an approximation version of the previous algorithm.

Sparse Evaluation of Expected Likelihood Kernel via Sampling

1. Compute the normalized histogram p for the template image, and take the square root to get $\sqrt{p(k)}$ of each bin k.
2. Quantize the image features of each pixel $\mathbf{x}$ in the test image to obtain the bin index $b(\mathbf{x})$.

3. Create an auxiliary image of the test image. The default value for every pixel in the auxiliary image is set to zero. For each pixel $\mathbf{x}$ of the test image, apply rejection sampling, and accept $\mathbf{x}$ with probability $\sqrt{p(b(\mathbf{x}))}$. In addition, assign the value $\sqrt{p(b(\mathbf{x}))}$ to the accepted pixel in the auxiliary image.

4. Build an integral image or more efficient data structures for the sparse auxiliary image, and compute the sum (the approximate kernel value) over each subwindow. Create a support map consisting of the sums as the output.

Justification: For each subwindow $\mathcal{R}$, the probability of a pixel belonging to the kth bin and being accepted via the rejection sampling should be $\sqrt{p(k)}$. Therefore, the expected number of the sampled pixels belonging to the kth bin within the subwindow $\mathcal{R}$ is $\sqrt{p(k)}\, h_{\mathcal{R}}(k)$, where $\sqrt{p(k)}$ is the survival rate of the rejection sampling, and $h_{\mathcal{R}}(k)$ is the original number of pixels belonging to the kth bin in subwindow $\mathcal{R}$ of the test image. The sparse evaluation algorithm thus approximates the expected likelihood kernel by

$$\begin{aligned} &\sum_k \sqrt{p(k)} \cdot \sqrt{p(k)}\, h_{\mathcal{R}}(k) = \sum_k p(k)\; h_{\mathcal{R}}(k) \\ &= |\mathcal{R}| \sum_k p(k)\; q(k) = |\mathcal{R}|\, K_1(p,q)\,. \end{aligned} \tag{7}$$

That is, the sparse evaluation algorithm can produce, on average, approximately the same result as the original fast evaluation algorithm. The advantage of the sparse evaluation algorithm is that we have a sparse auxiliary image, and more efficient data structures may be used to substitute for the integral image if needed.

Now we go on describing the sampling-based evaluation algorithm of the Bhattacharyya kernel, which is the focus of this paper. In the light of back-projection, it would be helpful if we can derive a similar way of evaluating the Bhattacharyya kernel for each subwindow. We use a rejection sampling scheme to select a small number of pixels from the test image, and on those selected pixels we perform local density estimation in a neighborhood to achieve the effect of accepting only a square-root number of pixels within the subwindow.

The following procedure is used to select a subset of pixels from the test image. This procedure will transform the test image into a very sparse image.

Sparse Evaluation of Bhattacharyya Kernel via Square-Root Sampling

1. Compute the normalized histogram p for the template, and take the square root to get $\sqrt{p(k)}$ of each bin k.

2. Quantize the image features of each pixel $\mathbf{x}$ in the test image to obtain its bin index $b(\mathbf{x})$.

3. Create a binary auxiliary image as large as the test image. All bits of the auxiliary image are set to zero by default. For each pixel $\mathbf{x}$, apply rejection sampling, and accept $\mathbf{x}$ with probability $\sqrt{p(b(\mathbf{x}))}$. Change the bit to 1 for the accepted pixel in the auxiliary image. We will obtain a sparse auxiliary image after the rejection sampling.

4. For each accepted pixel $\hat{\mathbf{x}}$ in the sparse auxiliary image, we estimate the local density through sampling randomly N pixels from the test image inside a neighborhood $\mathcal{N}(\hat{\mathbf{x}})$ around $\hat{\mathbf{x}}$, and then count the number N' of pixels belonging to the $b(\hat{\mathbf{x}})$-th bin. The local density is thus estimated by N'/N and the expected value of $h_{\mathcal{N}(\hat{\mathbf{x}})}(b(\hat{\mathbf{x}}))$ can be approximated as $h_{\mathcal{N}(\hat{\mathbf{x}})}(b(\hat{\mathbf{x}})) \simeq |\mathcal{R}|\, \frac{N'}{N}$. (In practice we choose $\mathcal{N}(\hat{\mathbf{x}})$ with $|\mathcal{N}(\hat{\mathbf{x}})| \approx 0.7|\mathcal{R}|$, where $|\mathcal{N}(\hat{\mathbf{x}})|$ is the area of $\mathcal{N}(\hat{\mathbf{x}})$). Take the square root and obtain $\sqrt{h_{\mathcal{N}(\hat{\mathbf{x}})}(b(\hat{\mathbf{x}}))}$. Again, apply rejection sampling, and accept $\hat{\mathbf{x}}$ with probability $1/\sqrt{h_{\mathcal{N}(\hat{\mathbf{x}})}(b(\hat{\mathbf{x}}))}$. Pixels that do not survive during the rejection sampling are eliminated from the sparse auxiliary image.

5. Build an integral image of the sparse auxiliary image, and count the number of the 1-bits within each subwindow of the sparse auxiliary image. The output is a support map consisting of the counts (approximating to the kernel values).

Justification. We assume that, for each subwindow $\mathcal{R}$ in the test image, the values of $\sqrt{h_{\mathcal{N}(\hat{\mathbf{x}})}(k)}$ for $\hat{\mathbf{x}} \in \mathcal{R}$ with $b(\hat{\mathbf{x}}) = k$ uniformly approximate to $\sqrt{h_{\mathcal{R}}(k)}$, and we denote the uniform values by $\sqrt{\tilde{h}_{\mathcal{R}}(k)}$. Such an approximation would be reliable if the neighborhoods $\mathcal{N}(\hat{\mathbf{x}})$ sufficiently overlap $\mathcal{R}$. Based on the above assumption, we may find that in the sparse auxiliary image the expected number of the survived pixels belonging to the kth bin is approximated by $\sqrt{p(k)}\, h_{\mathcal{R}}(k) \,/ \sqrt{\tilde{h}_{\mathcal{R}}(k)}$, where $\sqrt{p(k)}$ is the survival rate of the first round of rejection sampling, $1/\sqrt{\tilde{h}_{\mathcal{R}}(k)}$ is the approximate survival rate of the second round of rejection sampling based on the local densities, and $h_{\mathcal{R}}(k)$ is the original number of pixels belonging to the kth bin in subwindow $\mathcal{R}$ of the test image. Accordingly, the expected number of 1-bits within a subwindow inside the sparse auxiliary image may be considered as an approximation to the Bhattacharyya kernel:

$$\begin{aligned} \sum_k 1 \cdot \frac{\sqrt{p(k)}\, h_{\mathcal{R}}(k)}{\sqrt{\tilde{h}_{\mathcal{R}}(k)}} &\simeq \sum_k \sqrt{p(k)}\; \sqrt{h_{\mathcal{R}}(k)} \\ &= \sqrt{|\mathcal{R}|}\; K_{\frac{1}{2}}(p,q). \end{aligned} \tag{8}$$

The reliability of the approximation depends on the assumption and the characteristics associated with the test image, and we will examine the proposed algorithm by assessing the quality of search results under various conditions. The computations of estimating local densities and counting the 1-bits over subwindows are very fast because only a small number of sampled pixels are involved. Note that the use of the integral image is not essential; more efficient data structures such as 2D range trees can be considered to take advantage of the sparsity of sampled pixels.

5. Algorithm Analysis

In this section we provide an analysis of the square-root sampling algorithm for Bhattacharyya kernel evaluation. To begin with, let B denote the number of histogram bins. The number of pixels in a test image $\mathcal{Y}$ is defined by $|\mathcal{Y}|$. We assume the number of pixels that survive the first round of square-root rejection sampling is M; this number is generally much smaller than $|\mathcal{Y}|$. Furthermore, as described in the algorithm, the number of pixels selected for local density estimation is N. Since the first step of the algorithm is performed only once for the template's histogram (in $O(|\mathcal{T}|)$ time) and can be done in advance, we ignore this step in the analysis of computational cost. The time complexity for the remaining steps is summarized as follows.

- In Step 2, obtaining the bin index for each pixel in the test image can be done in $O(|\mathcal{Y}|)$ time.
- In Step 3, rejection sampling for each pixel takes $O(|\mathcal{Y}|)$ time, and assigning values to accepted pixels takes $O(M)$ time.
- In Step 4, for each pixel accepted in Step 3, estimating the local density by sampling N pixels takes $O(N)$ time. We have M pixels being accepted, so in this part we need $O(MN)$ time. However, we may specifically choose N to ensure $O(MN) = O(|\mathcal{Y}|)$.
- In Step 5, the computation regarding the integral image can be done in less than $O(|\mathcal{Y}|)$ time.

Consequently, the total computational cost is $O(|\mathcal{Y}|)$, and the time complexity of evaluating the kernel value at each subwindow is thus only $O(1)$. Note that the two state-of-the-art histogram-based algorithms, the integral histogram [18] and the distributive histogram [20], both have $O(|\mathcal{Y}|\, B)$ time complexity. For histograms with many bins, (*e.g.*, the color histogram used in [4] has $4,096$ bins), the computation cost regarding the factor of B would be large. Finally, the memory requirement for the square-root sampling algorithm is $O(|\mathcal{Y}|)$, which is due to the use of the sparse auxiliary image.

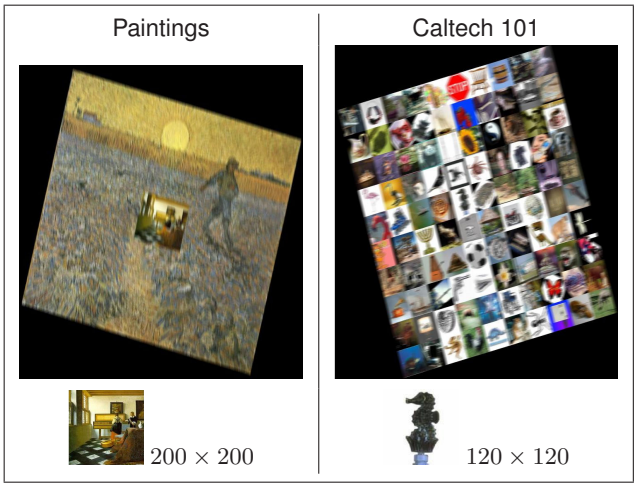


Figure 1. Examples of test images and query images for the two sets of experiments.

6. Experiments

We test the efficiency and reliability of the proposed sampling approach using different types of images with variations. In all experiments, we use color histograms to represent the template and the subwindows in the test image. For a color histogram of $4,096$ bins, each of the RGB channels is quantized to have 16 levels, and jointly the RGB values of a pixel decide a bin index ranging from 1 to 16^3. For each query, a bounding box corresponding to the subwindow having the largest kernel value (the maximum of the support map) is chosen as the search result. We compare five algorithms: *i)* fast evaluation of expected likelihood kernel (ELK), *ii)* sparse evaluation of expected likelihood kernel (SELK), *iii)* sparse evaluation of Bhattacharyya kernel via square-root sampling (SBTCY), *iv)* sum of squared difference between two normalized histograms (SSD$_{update}$) using the updating scheme [12] for fast computation, and *v)* exact evaluation of Bhattacharyya kernel using the updating scheme (BTCY$_{update}$) for efficient computation. The first three algorithms are presented in this paper, and the last two are standard approaches for comparison. All experiments are done on a Core2 Duo 1.6GHz laptop with 3GB RAM, and all algorithms are implemented in C.

In the first set of experiments we use 60 images of paintings containing a wide variety of styles and colors. At each run of the experiment, we randomly choose a painting as the background and another as the target. The background image is resized to 1000×1000 pixels and the target image is resized to 200×200 pixels. We paste the target image onto the background image to create a test image. The target image is used as the template to be searched inside the test image. To add variations, we first rotate the test image by a random angle within $[-15°, 15°]$, and then transform

the test image by a matrix $I + 0.1A$, where A is a 2-by-2 matrix with elements drown from a zero-mean unit-variance Gaussian, and finally, a motion-blur by 20 pixels in a random direction is applied to the test image. Examples of the resulting test image and the template for query are shown in the left column of Fig. 1. We perform the searches using the five algorithms, and compute the accuracy as the percentage of the search result overlapping the ground-truth location. We repeat 300 runs of experiments and the performances of the five algorithms are shown in Fig. 2. Three sizes of color histograms with 8^3, 16^3, and 32^3 bins have been tested. The proposed SBTCY algorithm achieves similar accuracy as the exact-evaluation algorithms SSD_{update} and BTCY_{update}. A query by SBTCY takes about 0.5s.

In the second set of experiments we use the object images from the Caltech 101 database [8]. We randomly choose one object image from each category, excluding the 'faces_easy' category. We resize each image to 120×120 pixels and arrange the 100 chosen images on a 10-by-10 grid to generate an image of size 1200×1200 pixels. We also apply random rotation, scaling, shearing, and motion-blur to the tiled image and obtain various test images, as shown in Fig. 1. We then pick each of the 100 object images in turn and use it as the template to perform the search on a random test image. We repeat the whole process three times, and thus perform, in total, 300 queries with 300 different templates on different test images. The tasks are more difficult since many object images are not colorful and include a large portion of uniform background. The performances of the five algorithms are shown in Fig. 3. The accuracy of SBTCY is comparable to SSD_{update}, but not as good as BTCY_{update} when the number of bins decreases.

7. Discussion

As shown in the experimental results, the square-root sampling approach can be applied to fast histogram-based search, and is able to yield reliable search results similar to exact evaluations. The proposed approximation algorithm is more suitable for colorful images like paintings and its gain in computation time is more significant when the size of templates and the number of bins are large. This type of sampling approach that integrates the computation of image representations with the evaluation of similarity measures might also be useful for other applications based on different image features. We are also seeking to get more insight into the theoretical validity of such approaches.

Acknowledgment. We thank the reviewers for their helpful comments. We are especially grateful to the anonymous Area Chairs for the valuable and insightful suggestions. This research was supported in part by grants 98-EC-17-A-19-S2-0052 and 98-2221-E-007-083-MY3.

References

[1] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *TPAMI*, 24(4):509–522, 2002.

[2] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc.*, 35:99–110, 1943.

[3] S. Birchfield. Elliptical head tracking using intensity gradients and color histograms. In *CVPR*, pages 232–237, 1998.

[4] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *TPAMI*, 25(5):564–575, 2003.

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR (1)*, pages 886–893, 2005.

[6] J. Delon, A. Desolneux, J. L. Lisani, and A. B. Petro. A nonparametric approach for histogram segmentation. *TIP*, 16(1):253–261, 2007.

[7] F. Ennesser and G. G. Medioni. Finding waldo, or focus of attention using local color information. *TPAMI*, 17(8):805–809, 1995.

[8] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *Workshop on Generative-Model Based Vision*, 2004.

[9] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice-Hall, Inc., 2001.

[10] J. L. Hafner, H. S. Sawhney, W. Equitz, M. Flickner, and W. Niblack. Efficient color histogram indexing for quadratic form distance functions. *TPAMI*, 17(7):729–736, 1995.

[11] J. Huang, R. Kumar, M. Mitra, W.-J. Zhu, and R. Zabih. Image indexing using color correlograms. In *CVPR*, pages 762–768, 1997.

[12] T. Huang, G. Yang, and G. Tang. A fast two-dimensional median filtering algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(1):13–18, 1979.

[13] T. Jebara and R. I. Kondor. Bhattacharyya and expected likelihood kernels. In *COLT*, pages 57–71, 2003.

[14] R. I. Kondor and T. Jebara. A kernel between sets of vectors. In *ICML*, pages 361–368, 2003.

[15] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *CVPR*, 2008.

[16] X. Liu and D. Wang. Image and texture segmentation using local spectral histograms. *TIP*, 15(10):3066–3077, 2006.

[17] P. Pérez, C. Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking. In *ECCV (1)*, pages 661–675, 2002.

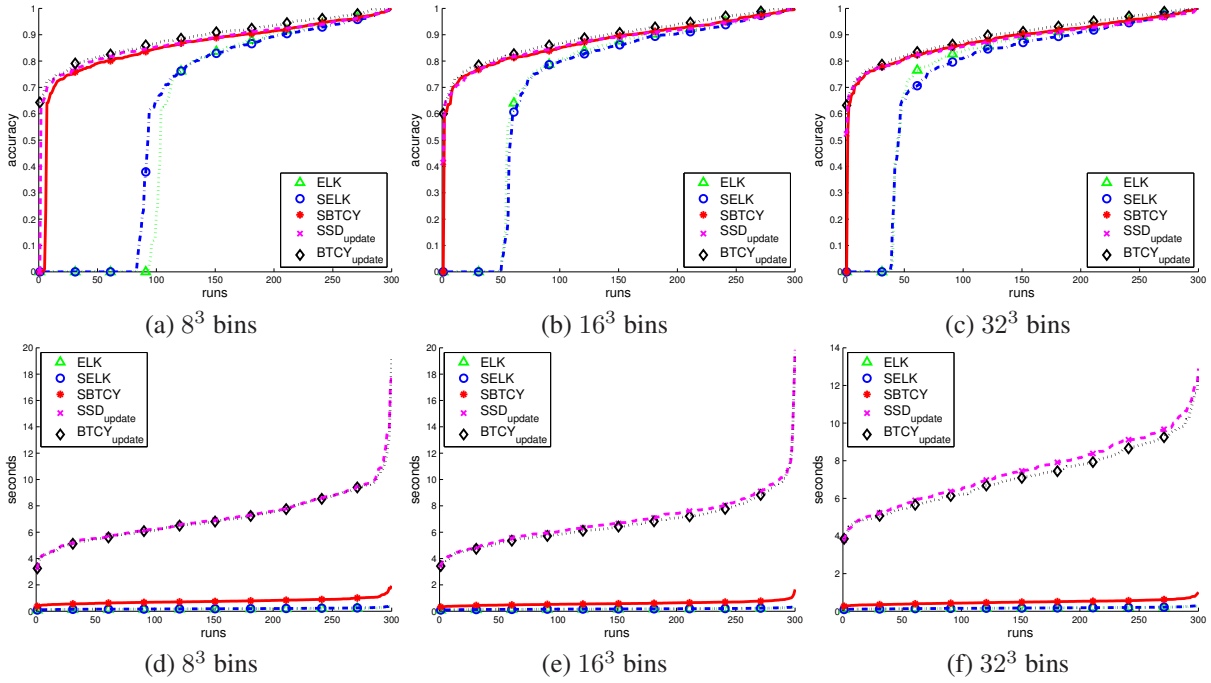
[18] F. M. Porikli. Integral histogram: A fast way to extract histograms in cartesian spaces. In *CVPR (1)*, pages 829–836, 2005.

[19] J. Puzicha, J. M. Buhmann, and T. Hofmann. Histogram clustering for unsupervised image segmentation. In *CVPR*, pages 2602–2608, 1999.

[20] M. Sizintsev, K. G. Derpanis, and A. Hogue. Histogram-based search: A comparative study. In *CVPR*, 2008.

[21] M. J. Swain and D. H. Ballard. Color indexing. *IJCV*, 7(1):11–32, 1991.

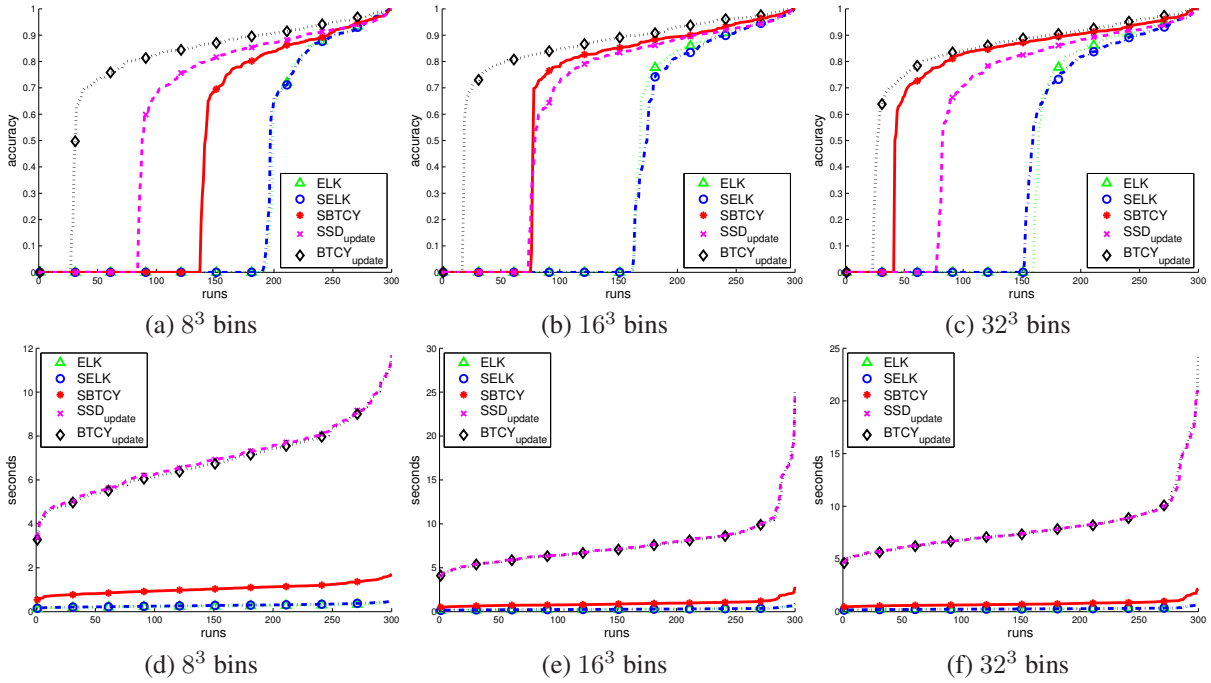
[22] P. A. Viola and M. J. Jones. Robust real-time face detection. *IJCV*, 57(2):137–154, 2004.



(a) 8^3 bins (b) 16^3 bins (c) 32^3 bins

(d) 8^3 bins (e) 16^3 bins (f) 32^3 bins

Figure 2. Experiment 1: Paintings. (a), (b), and (c) are the sorted accuracy of the five algorithms over the 300 runs of queries. (d), (e), and (f) are the sorted timing results. The size of test image is about 1000×1000 pixels and the size of query template is 200×200 pixels. The accuracy of the proposed approximation algorithm SBTCY is comparable to the accuracy of exact evaluations done by SSD_{update} and BTCY_{update}. Generally, SBTCY is 10 times faster than SSD_{update} and BTCY_{update}. Typically a query by SBTCY takes about 0.5s.



(a) 8^3 bins (b) 16^3 bins (c) 32^3 bins

(d) 8^3 bins (e) 16^3 bins (f) 32^3 bins

Figure 3. Experiment 2: Caltech 101. (a), (b), and (c) are the sorted accuracy of the five algorithms over the 300 runs of queries. (d), (e), and (f) are the sorted timing results. The size of test image is about 1200×1200 pixels and the size of query template is 120×120 pixels. The tasks are more difficult since many object images are not colorful and contain a large part of uniform background. The performances are not as good as those in Experiment 1. Note that the accuracy of SBTCY using 16^3 and 32^3 bins is better than SSD_{update}.