Beyond the Euclidean distance: Creating effective visual codebooks using the histogram intersection kernel

Jianxin Wu James M. Rehg
Center for Robotics and Intelligent Machines
School of Interactive Computing, Georgia Institute of Technology
wujx2001@gmail.com,rehg@cc.gatech.edu

Abstract

Common visual codebook generation methods used in a Bag of Visual words model, e.g. k-means or Gaussian Mixture Model, use the Euclidean distance to cluster features into visual code words. However, most popular visual descriptors are histograms of image measurements. It has been shown that the Histogram Intersection Kernel (HIK) is more effective than the Euclidean distance in supervised learning tasks with histogram features. In this paper, we demonstrate that HIK can also be used in an unsupervised manner to significantly improve the generation of visual codebooks. We propose a histogram kernel k-means algorithm which is easy to implement and runs almost as fast as k-means. The HIK codebook has consistently higher recognition accuracy over k-means codebooks by 2-4%. In addition, we propose a one-class SVM formulation to create more effective visual code words which can achieve even higher accuracy. The proposed method has established new state-of-the-art performance numbers for 3 popular benchmark datasets on object and scene recognition. In addition, we show that the standard k-median clustering method can be used for visual codebook generation and can act as a compromise between HIK and k-means approaches.

1. Introduction

Bag of Visual words (BOV) is currently a popular approach for object and scene recognition. Local features (either at interest points, or, densely or randomly sampled) are extracted and an image is then considered as a *bag of features*, i.e. completely ignoring the spatial relationship among features. Probably due to the lack of an effective mechanism to encode spatial information among features, BOV is widely adopted in vision tasks. A typical BOV-based method has the following steps:

- **Extract features.** Features descriptors (e.g. SIFT [17] or HOG [7]) are extracted from local sub-windows.
- **Generate a codebook and map features to visual code words.** A visual codebook is a method that divides the feature descriptor space into several regions. Features in one region correspond to the same *visual code word*, which is represented by an integer between 1 and size of the codebook. An image is then encoded as a histogram of visual code words.
- **Learn and test.** Various machine learning methods can be applied to the histogram representation of images, e.g. SVM is a frequently-used learner.

The quality of the codebook has an important impact on the success of BOV-based methods. Popular and successful methods for object and scene categorization typically employ unsupervised learning methods (e.g. k-means clustering or a Gaussian Mixture Model) to obtain a visual codebook. The most popular feature vectors are based on histograms of image measurements such as spatial gradients, optical flow, or color. For example, SIFT and HOG both use histograms of pixel intensity gradients in their descriptors. Currently, the Euclidean (l_2) distance measure is used in most codebook generation methods. However, for the case of supervised classification, it has been shown that the l_2 distance is not the most effective method for comparing two histograms [18]. In particular, the Histogram Intersection Kernel (HIK) was demonstrated to give significantly improved performance.

In this paper we demonstrate that HIK can be used to significantly improve the generation of visual codebooks with unsupervised learning. We describe three related techniques which differ in their computational and storage costs. These new methods are simple to implement, and our software implementations are freely available. We show that for essentially twice the computational cost of the standard k-means based method (which uses the l_2 distance), we can gain consistent accuracy improvement of 2-4% across a diverse collection of object and scene recognition datasets. Specifically, this paper makes four contributions:

2009 IEEE 12th International Conference on Computer Vision (ICCV)
978-1-4244-4419-9/09/$25.00 ©2009 IEEE

First, we show that HIK generates better codebooks and thus improves recognition accuracy. We generalize and speedup the method in [18], such that the generation and application of HIK codebook has the same complexity as standard k-means. Our proposed method achieves consistent performance improvements over the k-means codebook, and has established new state-of-the-art performance numbers for three benchmark object and scene recognition datasets. More generally, we suggest that *HIK should be used whenever two histograms are compared*.

Second, we show that a one-class SVM formulation can be used to improve upon the effectiveness of the HIK codebook, by providing well-separated, compact clusters in histogram feature space.

Third, we empirically show that k-median is a compromise between k-means and HIK codebooks. K-median's performance is consistently lower than the proposed HIK codebooks, but better than k-means in most cases. It runs as fast as the proposed method, but uses less storage.

Finally, we validate our method through experiments on standard datasets, utilizing both the SIFT feature and CENTRIST, a recently proposed feature based on CENsus TRansform hISTogram [35], which has been shown to offer performance advantages for scene classification.

2. Related Work

The main point of this paper is that when histogram features are employed, the histogram intersection kernel (HIK) should be used to compare them. HIK was introduced by Swain and Ballard [27] for color-based object recognition. [21] demonstrated that HIK forms a positive definite kernel, facilitating its use in SVM classifiers. Simultaneously, works such as [17, 7] demonstrated the value of histogram features for a variety of tasks. However, the high computational cost of HIK at run-time remained a barrier to its use in practice. This barrier was removed for the case of SVM classifiers by the work of Maji, Berg and Malik, who presented a technique to accelerate the kernel evaluations [18].

In this paper we extend the results of [18] in two ways: First, we demonstrate that the speedup of HIK can be extended to codebook generation (and unsupervised learning in general). Second, our Algorithm 2 provides an exact $O(d)$ method, which makes it possible to obtain the maximum efficiency without any potential loss of accuracy.

K-means is the most widely used method for codebook generation [26]. However, several alternative strategies have been explored. K-means usually positions its clusters almost exclusively around the densest regions. Jurie and Triggs used a mean-shift type clustering method to overcome this drawback [12]. There are also information theoretic methods that try to capture the "semantic" common visual components by minimizing information loss [16, 13]. An extreme method was presented in [28] that divided the space into regular lattice instead of learning a division from data. There are also efforts to build hash functions (multiple binary functions / hash bits) in order to accelerate distance computations [32]. In this work we propose a new alternative to k-means, based on the histogram intersection kernel.

In k-means based methods, a code word is represented by the cluster center (average of all features that belongs to this code word), which is simple and fast to compute. It was discovered that assigning a feature to multiple code words (soft-assignment) may improve codebook quality [23, 29]. Within a probabilistic framework, code words can be represented by the Gaussian Mixture Model (GMM) [22, 33]. GMM has better representation power than a single cluster center. However, it requires more computational power. Another interesting representation is hyperfeature [1], which considers the mapped code word indexes as a type of image feature and repeatedly generates new codebooks and code words into a hierarchy.

Methods have been proposed to accelerate the space division and code word mapping. [20] used a tree structure to divide the space hierarchically and [19] used ensembles of randomly created cluster trees. Both methods map visual features to code words much faster than k-means.

Some methods do not follow the *divide then represent* pattern. For example, [36] unified the codebook generation with classifier learning. In another interesting research Vogel and Schiele manually specified a few code words and used supervised learning to learn these concepts [31].

It is worth noting that all of these previous methods used the l_2 distance metric, and could therefore in principle be improved through use of HIK.

3. HIK Visual Codebook

3.1. The Histogram Intersection Kernel

Let $\boldsymbol{h} = (h_1, \ldots, h_d) \in R_+^d$ be a histogram. $\boldsymbol{h}$ could represent an image (e.g. histogram of code words) or a subwindow (e.g. SIFT feature descriptor). The histogram intersection kernel κ_{HI} is defined as follows

$$\kappa_{\mathrm{HI}}(\boldsymbol{h}_1, \boldsymbol{h}_2) = \sum_{i=1}^{d} \min(h_{1i}, h_{2i}). \qquad (1)$$

It is proved in [21] that κ_{HI} is a valid positive definite kernel. Thus there exists a mapping ϕ that maps any histogram $\boldsymbol{h}$ to a corresponding vector $\phi(\boldsymbol{h})$ in a high dimensional (possibly infinite dimensional) feature space Φ, such that $\kappa_{\mathrm{HI}}(\boldsymbol{h}_1, \boldsymbol{h}_2) = \phi(\boldsymbol{h}_1) \cdot \phi(\boldsymbol{h}_2)$. Through the nonlinear mapping ϕ, histogram similarity is equivalent to an inner product in the feature space Φ.

This kernel trick makes it possible to use the histogram intersection kernel in creating codebooks, while keeping the simplicity of k-means clustering. We propose to use a histogram kernel k-means algorithm to generate visual codebooks. In Algorithm 1, by kernel k-means in the feature

space spanned by ϕ, histograms are compared using HIK instead of the inappropriate Euclidean distance.

Algorithm 1 HIK Visual Codebook Generation

1: {Given n histograms $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n$ in R_+^d, m (size of the codebook), and ε (tolerance). }
2: {The output is a mapping from a histogram to a visual code word index, $w_1(\boldsymbol{h}_*) : R_+^d \to \{1, \ldots, m\}$.}
3: $t \leftarrow 0, e^0 \leftarrow \infty$
4: Use the *k-means++* method [2] to choose m histograms $\bar{\boldsymbol{h}}_1, \ldots, \bar{\boldsymbol{h}}_m$, and use $\boldsymbol{m}_i = \phi(\bar{\boldsymbol{h}}_i)$ as initial centers. ϕ is the mapping associated with κ_{HI}.
5: **repeat**
6: For all $1 \le i \le n$, $l_i \leftarrow \arg\min_{1 \le j \le m} \|\phi(\boldsymbol{h}_i) - \boldsymbol{m}_j\|^2$. (Set l_i to index of the center that is closest to the data point $\boldsymbol{h}_i$.)
7: For all $1 \le i \le m$, $\pi_i = \{j | l_j = i, 1 \le j \le n\}$. (Set π_i to the set of indexes that belong to the center $\boldsymbol{m}_i$.)
8: For all $1 \le i \le m$, $\boldsymbol{m}_i \leftarrow \frac{\sum_{j \in \pi_i} \phi(\boldsymbol{h}_j)}{|\pi_i|}$. (Update the centers.)
9: $t \leftarrow t + 1, e^t = \sum_{1 \le i \le n} \|\phi(\boldsymbol{h}_i) - \boldsymbol{m}_{l_i}\|^2$
10: **until** $e^{t-1} - e^t \le \varepsilon$.
11: **output**: For any histogram $\boldsymbol{h}_* \in R_+^d$,

$$w_1(\boldsymbol{h}_*) = \arg\min_{1 \le i \le m} \|\phi(\boldsymbol{h}_*) - \boldsymbol{m}_i\|^2. \quad (2)$$

Note that since *k-means++* used in Algorithm 1 is a randomized algorithm, two runs of Algorithm 1 with the same input will possibly generate different results.

3.2. Fast Evaluation

The major component of Algorithm 1 is a kernel k-means algorithm [25] using κ_{HI} (c.f. Eqn. 1). Since the centers $\boldsymbol{m}_i$ are vectors in the unrealized, high dimensional space Φ, the key computation is carried out in the following way (using the usual kernel trick $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y}) = \kappa_{\mathrm{HI}}(\boldsymbol{x}, \boldsymbol{y})$ such that $\boldsymbol{m}_i$ does not need to be explicitly generated):

$$\|\phi(\boldsymbol{h}_*) - \boldsymbol{m}_i\|^2 = \left\| \phi(\boldsymbol{h}_*) - \frac{\sum_{j \in \pi_i} \phi(\boldsymbol{h}_j)}{|\pi_i|} \right\|^2 \quad (3)$$

$$= \|\phi(\boldsymbol{h}_*)\|^2 + \frac{\sum\limits_{j,k \in \pi_i} \kappa_{\mathrm{HI}}(\boldsymbol{h}_j, \boldsymbol{h}_k)}{|\pi_i|^2} - \frac{2 \sum\limits_{j \in \pi_i} \kappa_{\mathrm{HI}}(\boldsymbol{h}_*, \boldsymbol{h}_j)}{|\pi_i|}. \quad (4)$$

The first term in Eqn. 4 does not affect the result in lines 6 and 11 of Algorithm 1, and the second term can be pre-computed. Thus most of the computations are spent in computing the last term $\sum_{j \in \pi_i} \kappa_{\mathrm{HI}}(\boldsymbol{h}_*, \boldsymbol{h}_j)$.

Note that this term is similar to the binary SVM classifier using HIK, which has the following form

$$\mathrm{sgn}\left(\sum\nolimits_{i \in \pi} \alpha_i y_i \kappa_{\mathrm{HI}}(\boldsymbol{h}_*, \boldsymbol{h}_i) + \rho \right) \quad (5)$$

where $\boldsymbol{h}_i$, α_i, and y_i are support vectors and their corresponding weights and labels.

A naive method will take $O(|\pi_i| d)$ steps to compute Eqn. 4. In [18], Maji, Berg and Malik proposed fast methods to compute Eqn. 5 (exact answer in $O(d \log |\pi|)$ steps and approximate answer in $O(d)$ steps.) In this paper we generalize their method and propose a variant that finds the exact answer for Eqn. 4 in $O(d)$ steps.

Note that both Eqn. 5 and the last term in Eqn. 4 are special forms of the following expression

$$f(\boldsymbol{h}_*) = \sum\nolimits_{i \in \pi} c_i \kappa_{\mathrm{HI}}(\boldsymbol{h}_*, \boldsymbol{h}_i), \quad (6)$$

where π indexes a set of histograms (support vectors) and c_i are constant coefficients.

A histogram of visual code word indexes has the property that every histogram component is a non-negative integer, i.e. it is a vector in N^d. Similarly, a feature descriptor histogram can usually be transformed into the space N^d. For example, the SIFT descriptors are stored as vectors in N^{128}. In general, a vector in R_+^d can be transformed into N^d by first multiplying an integer to the histogram and then rounding its components to nearest integers.

In the rest of this paper we assume that any histogram $\boldsymbol{h} = (h_1, \ldots, h_d)$ satisfies that $h_i \in N$ and $h_i \le h_{\max}$ for all i. Then the quantity $f(\boldsymbol{h}_*)$ can be computed as follows,

$$\begin{aligned} f(\boldsymbol{h}_*) &= \sum_{i \in \pi} c_i \kappa_{\mathrm{HI}}(\boldsymbol{h}_*, \boldsymbol{h}_i) \\ &= \sum_{i \in \pi} \sum_{1 \le j \le d} c_i \min(h_{*j}, h_{ij}) \\ &= \sum_{1 \le j \le d} \left(\sum_{i \in \pi} c_i \min(h_{*j}, h_{ij}) \right) \\ &= \sum_{1 \le j \le d} \left(\sum_{h_{*j} \ge h_{ij}} c_i h_{ij} + h_{*j} \sum_{h_{*j} < h_{ij}} c_i \right). \end{aligned} \quad (7)$$

Note that the two summands in Eqn. 7 can both be pre-computed. It is shown in [18] that Eqn. 7 is a piece-wise linear function of h_{*j}. Thus using a binary search for h_{*j}, Eqn. 7 can be computed in $O(d \log |\pi|)$ steps. However, since we can assume that h_{*j} is an integer in the range $0..h_{\max}$, we have an even faster computing method. Let T be a table of size $d \times (1 + h_{\max})$, with $\sum_{k \ge h_{ij}} c_i h_{ij} + k \sum_{k < h_{ij}} c_i$ being assigned to the (j,k)-th entry $T(j,k)$, $1 \le j \le d, 0 \le k \le h_{\max}$. Then it is clear that $f(\boldsymbol{h}_*) = \sum_{j=1}^d T(j, h_{*j})$. This method is summarized in Algorithm 2.

[18] also approximated histogram components by uniformly sampled points in the range of possible values in order to achieve the $O(d)$ speed. However, the uniform sampling strategy might be undesirable because histogram cell values usually concentrate in a small range.

It is obvious that $f(\boldsymbol{h}_*)$ can be evaluated in $O(d)$ steps after the table T is pre-computed. Because Algorithm 2 only involves table lookup and summation, it is faster (less

Algorithm 2 Fast Computing of HIK Sums

1: {Given n histograms $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n$ in N^d, with $0 \leq h_{ij} \leq h_{\max}$ for $1 \leq i \leq n, 1 \leq j \leq d$ }
2: {The output is a fast method to compute $f(\boldsymbol{h}_*) = \sum_{i=1}^{n} c_i \kappa_{\mathrm{HI}}(\boldsymbol{h}_*, \boldsymbol{h}_i)$, where $\boldsymbol{h}_* \in N^d$.}
3: Create T, a $d \times (1 + h_{\max})$ table.
4: $T(j,k) \leftarrow \sum_{k \geq h_{ij}} c_i h_{ij} + k \sum_{k < h_{ij}} c_i$, for $1 \leq j \leq d$, $0 \leq k \leq h_{\max}$.
5: **output**: $f(\boldsymbol{h}_*) = \sum_{j=1}^{d} T(j, h_{*j})$.

overhead) than the approximation scheme in [18], which is also $O(d)$. Depending on the relative size of $h_{\max}$ and the number of approximation bins used in [18], Algorithm 2's storage requirement ($O(h_{\max} d)$) could be larger or smaller than that of [18]. The pre-computation of T requires $O(dnh_{\max})$ steps. Although filling in the table T is computationally expensive, it is done only once.[1] It is also worth noting that under our assumption Algorithm 2's result is precise rather than approximate.

Both the pre-computation complexity and storage requirement are linear in $h_{\max}$, which is a parameter specified by users. Our experiments show that while a too small $h_{\max}$ usually produces inferior results, larger $h_{\max}$ does not necessarily improves system performance. In this paper, we choose $h_{\max} = 128$, which seems to give the best results in our experiments.

Our algorithms have the same complexity as a usual linear k-means when generating a visual codebook or mapping from histograms to visual code word indexes (Eqn. 2 or 4). In practice the proposed method takes about twice the time of k-means. In summary, the proposed method generates a visual codebook that can not only run almost as fast as the k-means method, but also can utilize the non-linear similarity measure κ_{HI} that is suitable for comparing histograms.

3.3. One-class SVM code word generation

A codebook generated by the k-means algorithm first divides the space R_+^d into m regions, then represents each code word (region) by the centroid of vectors (histogram, feature vectors, etc.) that fall into this region. This approach is optimal if we assume that vectors in all regions follow the Gaussian distributions with the same spherical covariance matrix (only differ in their means).

This assumption rarely holds. Different regions usually have very different densities and covariance structures. Simply dividing the space R_+^d into a Voronoi diagram from the set of region centers is, in many cases, misleading. However, further refinements are usually computationally prohibitive. For example, if we model regions as Gaussian distributions with distinct covariance matrices, the generation and mapping from visual features to code words will require much more storage and computational resources than we can afford.

We propose to use one-class SVM [24] to represent the divided regions in an effective and computationally efficient way. Given a set of histograms in a region $\boldsymbol{h}_\pi = \{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n\}$, we construct a one-class SVM with parameter $\nu \in (0, 1]$,

$$\mathrm{sgn}\left(\sum_{i \in \pi} \alpha_i \kappa_{\mathrm{HI}}(\boldsymbol{h}, \boldsymbol{h}_i) - \rho\right) \tag{8}$$

where α_i's are non-negative, sparse and $\sum_i \alpha_i = 1$. Intuitively, a one-class SVM classifier seeks a "simple" (compact) subset of $\boldsymbol{h}_\pi$ (or the divided region) that retains a large portion of the histograms (or densities). It is proved that ν is the upper bound on the fraction of outliers (i.e. on which Eqn. 8 are less than 0), and at the same time a lower bound on the fraction of support vectors (i.e. $\alpha_i \neq 0$) [24].

The one-class SVM (using HIK) summarizes the distribution of histograms inside a visual code word. It takes into consideration the shape and density of the histogram distribution. It seeks to include most of the histograms (at least $(1 - \nu)|\pi|$) in a compact hypersphere in the feature space, while paying less attention to those borderline cases (at most $\nu|\pi|$ examples). We believe that this compact hypersphere better summarizes a visual code word.

At the same time, these new code words can be computed very efficiently. Eqn. 8 is evaluated in $O(d)$ steps because it is again a special case of Algorithm 2. We propose Algorithm 3 to use the one-class SVM to generate visual code words.[2] In this paper, we set the parameter $\nu = 0.2$.

Algorithm 3 One-class SVM Code Word Generation

1: {Use Algorithm 1 to generate the divisions π_i ($i = 1, \ldots, m$) from n histogram $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n$ in R_+^d. }
2: For each division $1 \leq i \leq m$, train a one-class SVM from its data $\boldsymbol{h}_{\pi_i}$ with a parameter ν,

$$w_2^i(\boldsymbol{h}_*) = \sum_{j \in \pi_i} \alpha_j \kappa_{\mathrm{HI}}(\boldsymbol{h}_*, \boldsymbol{h}_j) - \rho_i \tag{9}$$

3: **output**: For any histogram $\boldsymbol{h}_* \in R_+^d$,

$$w_2(\boldsymbol{h}_*) = \arg\max_{i=1}^{m} w_2^i(\boldsymbol{h}_*). \tag{10}$$

In many applications a histograms $\boldsymbol{h} = (h_1, \ldots, h_d)$ satisfy that $\|\boldsymbol{h}\|_1 = \sum_{i=1}^{d} h_i = N$ is a constant. Under this condition, Eqn. 9 is equivalent to

$$w_2^i(\boldsymbol{h}_*) = r_i^2 - \|\phi(\boldsymbol{h}_*) - \boldsymbol{m}_i\|^2$$

where $\boldsymbol{m}_i = \sum_{j \in \pi_i} \alpha_j \boldsymbol{h}_j$ and $r_i^2 = N + \|\boldsymbol{m}_i\|^2 - 2\rho_i$. In other words, a histogram is considered as belonging to the i-th visual word if it is inside the sphere (in the feature space Φ) centered at $\boldsymbol{m}_i$ with radius r_i. A sphere in Φ is different

[1] A better method is to first bucket sort each dimension of the histograms then fill in the values of T sequentially, which takes only $O\left(d(n + h_{\max})\right)$ steps.

[2] Note that we use the space R_+^d because Algorithm 3 is not restricted to N^d.

from the a usual k-means sphere (in R_+^d) because it respects the similarity measure κ_{HI}, and its radius r_i automatically adapts with the distribution of histograms in a visual word.

Although the proposed algorithms are described using the histogram intersection kernel κ_{HI}, they are readily applied to other kernel types. For example, if we use the linear kernel $\kappa_{\mathrm{LIN}}(\boldsymbol{h}_1, \boldsymbol{h}_2) = \boldsymbol{h}_1 \cdot \boldsymbol{h}_2$, Algorithm 1 will reduce to a usual k-means visual codebook generation method, and Algorithm 3 will perform one-class SVM in the space R_+^d.

3.4. K-median codebook generation

Although k-means (or, equivalently κ_{LIN} or l_2 distance) is the most popular codebook generation method, the histogram intersection kernel has a closer connection to the l_1 distance. For two numbers a and b, it is easy to show that $2\min(a, b) + |a - b| = a + b$. As a consequence, we have

$$2\kappa_{\mathrm{HI}}(\boldsymbol{h}_1, \boldsymbol{h}_2) + \|\boldsymbol{h}_1 - \boldsymbol{h}_2\|_1 = \|\boldsymbol{h}_1\|_1 + \|\boldsymbol{h}_2\|_1 . \quad (11)$$

In cases when $\|\boldsymbol{h}\|_1$ is constant for any histogram $\boldsymbol{h}$, κ_{HI} and the l_1 distance are linearly correlated.

For an array $x_1, \ldots, x_n$, it is well known that the value that minimizes the l_1 error ($x_* = \arg\min_x \sum_{i=1}^n |x - x_i|$) equals the median value of the array. Thus k-median is a natural alternative for codebook generation.[3]

K-median has been less popular than k-means for the creation of visual codebooks. An online k-median algorithm has been used by Larlus and Jurie to create visual codebooks in the Pascal challenge [9]. In Table 2 (Section 4.2) we will compare the batch version of k-median to k-means and the proposed HIK method.

4. Experiments

4.1. Datasets and Setup

We validate the proposed methods using three datasets: the Caltech 101 object recognition dataset [10], the 15 class scene recognition dataset [14], and the 8 class sports events dataset [15]. In each dataset, the available data are randomly split into a training set and a testing set based on published protocols on these datasets. The random splitting is repeated 5 times, and the average accuracy is reported. In each train/test splitting, a visual codebook is generated using the training images, and both training and testing images are transformed into histograms of code words. Accuracy is computed as the mean accuracy of all categories (i.e. average of the diagonal entries in the confusion matrix).

The proposed algorithms can efficiently process a huge number of histogram features, e.g. approximately 200k to 320k histograms are clustered across the training sets in the three datasets in about less than 6 minute.

[3]The only difference between k-median and k-means is that k-median uses l_1 instead of l_2 as the distance metric.

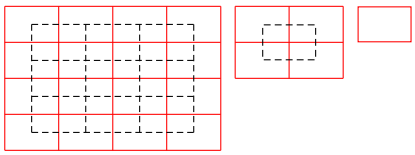

Figure 1: Illustration of the level 2, 1, and 0 split of an image.

In the BOV model, we use 16x16 image patches and densely sample features over a grid with a spacing of 2, 4, or 8 pixels. We use two types of feature descriptors: SIFT for Caltech 101, and CENTRIST (CENsus TRansform hISTogram, refer to [35]) for the other two datasets.[4] All feature vectors are scaled and rounded such that a histogram only contains non-negative integers that approximately sum to 128 (thus $h_{\max} = 128$ is always valid.)

The first step is to use feature descriptors from the training images to form a visual codebook, in which we use $m = 200$ to generate 200 visual code words. Next every feature is mapped to an integer (code word index) between 1 and m. Thus an image or image sub-window is represented by a histogram of code words in the specified image region. In order to incorporate spatial information, we use the spatial hierarchy in [35], as illustrated in Fig. 1. An image is represented by the concatenation of histograms from all the 31 sub-windows in Fig. 1, which is a 6200 dimensional histogram. To capture the edge information, we sometimes use Sobel gradients of an input image as an additional input, and concatenate histograms from the original input and Sobel gradient image (which is 12400 dimensional). Following [3], we also sample features at 5 scales.[5]

SVM is used for classification. LIBSVM [6] is used for the scene and sports dataset. It uses the 1-vs-1 strategy, which will produce too many classifiers for the Caltech 101 dataset (more than 5000). Instead we use the Crammer & Singer formulation in BSVM [11]. Since we are classifying histograms, we modified both LIBSVM and BSVM so that they are able to utilize the histogram intersection kernel.

4.2. Main Results

We conducted several sets of experiments to validate the proposed algorithms. Experimental results are organized using the following rule: the texts in italic type summarize findings from one set of experiments and details are described after the italic texts. We first present the main results. Mean / standard deviation values and paired t-tests are used to show the benefit of HIK codebooks (Algorithm 1), while the Wilcoxon test is used for Algorithm 3.

[4]We will also evaluate the effect when these two feature types are switched in these datasets.

[5]For more details, please refer to the source code of our libHIK package, which is available at `http://www.cc.gatech.edu/cpl/projects/libHIK`.

Table 1: Results of HIK, k-median and k-means codebooks and one-class SVM code words. (a), (b), and(c) are results for the Caltech 101, 15 class scene, and 8 class sports datasets, respectively. κ_{HI} and κ_{LIN} means that a histogram intersection or linear kernel is used, respectively. oc_{svm} and $\neg oc_{\mathrm{svm}}$ indicate whether one-class SVM is used in generating code words. B and $\neg B$ indicate whether Sobel images are concatenated or not. And $s = 4$ or $s = 8$ is the grid step size when densely sampling features. The number of training/testing images in each category are indicated in the sub-table captions. The best result in each *column* is shown in **boldface**.

	$B, s = 4$	$B, s = 8$	$\neg B, s = 8$
$\kappa_{\mathrm{HI}}, oc_{\mathrm{svm}}$	**67.44±0.95%**	**65.20±0.91%**	**61.00±0.90%**
$\kappa_{\mathrm{HI}}, \neg oc_{\mathrm{svm}}$	66.54±0.58%	64.11±0.84%	60.33±0.95%
k-median	66.38±0.79%	63.65±0.94%	59.64±1.03%
$\kappa_{\mathrm{LIN}}, oc_{\mathrm{svm}}$	62.69±0.80%	60.09±0.92%	56.31±1.13%
$\kappa_{\mathrm{LIN}}, \neg oc_{\mathrm{svm}}$	64.39±0.92%	61.20±0.95%	57.74±0.70%

(a) Caltech 101, 15 train, 20 test

	$B, s = 4$	$B, s = 8$	$\neg B, s = 8$
$\kappa_{\mathrm{HI}}, oc_{\mathrm{svm}}$	**84.12±0.52%**	**84.00±0.46%**	**82.02±0.54%**
$\kappa_{\mathrm{HI}}, \neg oc_{\mathrm{svm}}$	83.59±0.45%	83.74±0.42%	81.77±0.49%
k-median	83.04±0.61%	82.70±0.42%	80.98±0.50%
$\kappa_{\mathrm{LIN}}, oc_{\mathrm{svm}}$	79.84±0.78%	79.88±0.41%	77.00±0.80%
$\kappa_{\mathrm{LIN}}, \neg oc_{\mathrm{svm}}$	82.41±0.59%	82.31±0.60%	80.02±0.58%

(b) 15 class scene, 100 train, rest test

	$B, s = 4$	$B, s = 8$	$\neg B, s = 8$
$\kappa_{\mathrm{HI}}, oc_{\mathrm{svm}}$	**84.21±0.99%**	**83.54±1.13%**	81.33±1.56%
$\kappa_{\mathrm{HI}}, \neg oc_{\mathrm{svm}}$	83.17±1.01%	83.13±0.85%	**81.87±1.14%**
k-median	82.13±1.30%	81.71±1.30%	80.25±1.12%
$\kappa_{\mathrm{LIN}}, oc_{\mathrm{svm}}$	80.42±1.44%	79.42±1.51%	77.46±0.83%
$\kappa_{\mathrm{LIN}}, \neg oc_{\mathrm{svm}}$	82.54±0.86%	82.29±1.38%	81.42±0.76%

(c) 8 class sports, 70 train, 60 test

Table 2: Comparison of three codebook generation methods. The average accuracy on three datasets are shown in the first row. In the last 2 rows, K-means is used as a baseline, i.e. a value 2 means 200% of that of k-means.

	HIK	k-median	k-means
Accuracy	78.59%	77.18%	76.45%
Computation cost	2	2	1
Storage requirement	$h_{\max}$	1	1

Histogram Intersection Kernel Visual Codebook (Algorithm 1) greatly improves classification accuracy. We compare the classification accuracies of systems that use Algorithm 1 (i.e. using κ_{HI}) and the usual k-means algorithm (i.e. using κ_{LIN}) and k-median. From the experimental results in Table 1, it is obvious that in all three datasets, the classification accuracy with a κ_{HI}-based codebook is consistently higher than that with a k-means or k-median codebook. Using a paired t-test with significance level 0.05, the differences are statistically significant in 16 out the 18 cases in Table 1, when comparing κ_{HI} and κ_{LIN} based codebooks.

K-median is a compromise between k-means and HIK codebooks. As shown in Table 1 and 2, HIK codebooks consistently outperformed k-median codebooks. However, k-median generally outperformed the popular k-means codebooks. Furthermore, k-median requires less memory than the proposed method.

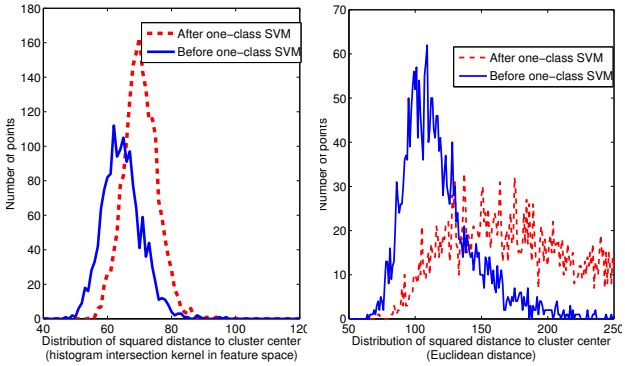

Figure 2: Effects of one-class SVM.

HIK codebook can be computed efficiently (Algorithm 2). We have shown that Algorithm 2 evaluates in $O(d)$ steps, in the same order as k-means. Empirically, the κ_{HI}-based method spent less than 2 times CPU cycles than that of k-means. For example, the proposed method took 105 seconds to generate a codebook for the Caltech 101 dataset, while k-means used 56 seconds in our experiments.

One-class SVM improves histogram intersection kernel code words (Algorithm 3). The t-test is not powerful enough here because we have only 5 paired samples and they are not necessarily normally distributed. The Wilcoxon signed-rank test is more appropriate [8] to show the effect of Algorithm 3. Algorithm 3 improved classification accuracy of the κ_{HI}-based method in 8 out of 9 cases in Table 1. The Wilcoxon test shows that the difference is significant at significance level 0.02.

In short, using HIK codebooks and one-class SVM altogether generated the best results in almost all cases (best results are shown in boldface within each column of Table 1).

One-Class SVM degrades usual k-means code words. It is interesting to observe a completely reversed trend when κ_{LIN} is used with one-class SVM. Applying Algorithm 3 in a usual k-means method reduced accuracy in all cases. Since a vector in R^d is not an appropriate understanding of a d-dimensional histogram, we conjecture that Algorithm 3 with κ_{LIN} produced a better division of the space R^d, but probably a worse one in the space of histograms.

Fig. 2 shows the effect of applying Algorithm 3 to example code words. The distribution of squared distance to cluster center becomes more compact in case of κ_{HI} with a minor increase in the average error. However, in the k-means case, the distances spread to larger values.

There is not an obvious kernel for the l_1 distance, and so we did not use one-class SVM for codebooks generated by k-median.

4.3. Effects of Information Content

Next we study the effects of using different type and amount of information, e.g. different type of base features

Table 3: Results when features are sampled in only 1 image scale, using $\neg B$ and $s = 8$.

	Caltech101	15 scene	8 sports
$\kappa_{\mathrm{HI}}, oc_{\mathrm{svm}}$	**57.70±0.59%**	**82.30±0.49%**	**77.54±1.49%**
$\kappa_{\mathrm{HI}}, \neg oc_{\mathrm{svm}}$	57.13±0.57%	82.07±0.53%	77.33±1.01%
$\kappa_{\mathrm{LIN}}, oc_{\mathrm{svm}}$	53.18±0.98%	78.70±0.51%	73.42±1.41%
$\kappa_{\mathrm{LIN}}, \neg oc_{\mathrm{svm}}$	54.06±0.24%	80.48±0.83%	76.29±0.60%

Table 4: Results when feature type was switched for object and scene recognition. CENTRIST was used in Caltech 101 and SIFT in the other two in the first row, and features are switched in the second row. We use $\neg B$, $s = 8$, κ_{HI}, and oc_{SVM}. Note that the second row contains the top right corner numbers in Table 1.

Caltech101	15 scene	8 sports
$53.25 \pm 0.80\%$	$78.54 \pm 0.22\%$	$81.17 \pm 0.65\%$
$61.00 \pm 0.90\%$	$82.02 \pm 0.54\%$	$81.33 \pm 1.56\%$

and step size in dense feature sampling.

Sampling features at 5 scales improved accuracy. It is advantageous to sample features from multiple scaled versions of the input image. Comparing Table 3 and the last columns of Table 1, we find that sampling more features improves object recognition by a large margin while the improvement to scene recognition is relatively small. Also, Table 3 reinforced all the conclusions from Section 4.2.

Smaller step size is better. Similarly, a smaller step size means that more features are sampled. Table 1 shows that when other conditions were the same, $s = 4$ outperformed $s = 8$ in general. However, we again observe differences between object and scene recognition. The accuracy difference in Caltech 101 is significant. In the sports dataset, $s = 4$ slightly outperformed $s = 8$ and they are indistinguishable in the 15 class scene dataset. Thus it is not necessary to compute $s = 2$ results for the two scene recognition datasets. In Caltech 101, however, $s = 2$ further improved recognition accuracy to $67.82 \pm 0.59\%$ (using κ_{HI}, oc_{SVM}, and B.)

Use the right feature for different tasks. SIFT is widely used in object recognition for its performance. CENTRIST has been shown as a suitable feature for place and scene recognition [35]. As shown in Table 4, if we use SIFT for scene recognition and CENTRIST for object recognition, the recognition accuracies are significantly reduced.

More code words are (sometimes) better. We also experimented with different number of code words. In the scene recognition tasks, we did not observe significant changes in recognition accuracies. In the Caltech 101 dataset, however, a higher accuracy $70.74 \pm 0.69\%$ was achieved using 1000 code words (with κ_{HI}, oc_{SVM}, B, and $s = 2$).

In summary, we need to choose the appropriate feature for a specific task (CENTRIST for scene recognition and SIFT for object recognition), and incorporate as much information as possible.

What's more interesting is the different behaviors of object and scene recognition problems exhibited in our experiments. Scene recognition requires different type of features (CENTRIST instead of SIFT) and less information (performance almost stabilize when step size is 8 and codebook size is 200). We strongly recommend the CENTRIST descriptor (or variants like PACT [34]) and the proposed algorithms for recognizing place and scene categories.

4.4. Comparison with previously published results

In this section we will compare our methods with previously published results. The parameters we use in the proposed methods are κ_{HI}, oc_{SVM}, and B.

In the scene recognition tasks, the proposed method achieved the highest accuracy in the literature. In the 15 class scene recognition task, the proposed method has an accuracy of $84.12 \pm 0.52\%$. The Spatial Pyramid Matching method achieved $81.4 \pm 0.5\%$. SP-pLSA [5], a method combining spatial pyramids and pLSA (probabilistic Latent Semantic Analysis), had 83.7% correct recognitions. The sports dataset was first published in [15], which achieved a 73.4% accuracy. The method in [15] used manual segmentation and object labels as additional inputs for their training method. Spatial PACT [34], using a spatial hierarchy of PACT (which is simply PCA of CENTRIST), achieved 78.50% correct category predictions, which is still inferior to the proposed methods ($84.21 \pm 0.99\%$) by a large margin.

Results for the Caltech 101 dataset are usually divided into two types: methods that use a single type of feature and methods that integrate multiple cues (e.g. color, texture, shape, etc). Several methods that use multiple cues outperformed our method, for example [4, 3, 30]. The proposed method (which uses only a single type of feature), however, has much higher accuracy than published single cue methods. With $m = 1000$ and $s = 2$ and 15 training examples per category, its accuracy is $70.74 \pm 0.69\%$. This accuracy is higher than methods such as NBNN (Naive-Bayes Nearest-Neighbor) [3] ($65.0 \pm 1.14\%$). For more single cue results, please refer to [3]. It is expected that when the κ_{HI}-based codebook and one-class SVM code words are used, the proposed method will be integrated into and further improve the multiple cue methods.

5. Conclusions

In this paper we show that when the Histogram Intersection Kernel (HIK) is used as the similarity measure in clustering feature descriptors that are histograms, the generated visual codebooks produce better code words and as a consequence, improved Bag of Visual words classifiers. We propose a HIK based codebook generation method which runs almost as fast as k-means and has consistently higher accuracy than k-means codebooks by 2-4%. Unlike k-means in which cluster centroids are used to represent code words,

we proposed a one-class SVM formulation (using HIK) to generate better code words. The proposed algorithms achieve state-of-the-art accuracy on three benchmark object and scene recognition datasets. Although k-median is rarely used to generate codebooks, we empirically evaluated k-median codebooks and recommend it as a compromise between the proposed method and k-means. K-median codebooks have lower accuracy than HIK codebooks but usually have higher accuracy than k-means codebooks. It also requires less memory than HIK codebooks.

We also provide a software package, named libHIK, which contains implementations of the methods proposed in this paper. The software is available at `http://www.cc.gatech.edu/cpl/projects/libHIK`.

This research was supported in part by a grant from the Google Research Awards Program.

References

[1] A. Agarwal and B. Triggs. Multilevel image coding with hyperfeatures. *IJCV*, 78(1):15–27, 2008.

[2] D. Arthur and S. Vassilvitskii. **k-means++**: the advantage of careful seeding. In *18th Symposium on Discrete Algorithms (SODA)*, pages 1027–1035, 2007.

[3] O. Boiman, E. Shechtman, and M. Irani. In defense of nearest-neighbor based image classification. In *CVPR*, 2008.

[4] A. Bosch, X. Muñoz, and A. Zisserman. Image classification using random forests and ferns. In *ICCV*, 2007.

[5] A. Bosch, A. Zisserman, and X. Muñoz. Scene classification using a hybrid generative/discriminative approach. *IEEE TPAMI*, 30(4):712–727, 2008.

[6] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

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

[8] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *JMLR*, 7:1–30, 2006.

[9] M. Everingham, A. Zisserman, C. Williams, and L. V. Gool. The PASCAL visual object classes challenge 2006 (VOC 2006) results. Technical report, 2006.

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

[11] C.-W. Hsu and C.-J. Lin. *BSVM*, 2006. Software available at `http://www.csie.ntu.edu.tw/~cjlin/bsvm`.

[12] F. Jurie and B. Triggs. Creating efficient codebooks for visual recognition. In *ICCV*, volume 1, pages 604–610, 2005.

[13] S. Lazebnik and M. Raginsky. Supervised learning of quantizer codebooks by information loss minimization. *IEEE TPAMI*, 31(7):1294–1309, 2009.

[14] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, volume II, pages 2169–2178, 2006.

[15] L.-J. Li and L. Fei-Fei. What, where and who? Classifying events by scene and object recognition. In *ICCV*, 2007.

[16] J. Liu and M. Shah. Scene modeling using Co-Clustering. In *ICCV*, 2007.

[17] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[18] S. Maji, A. C. Berg, and J. Malik. Classification using intersection kernel support vector machines is efficient. In *CVPR*, 2008.

[19] F. Moosmann, E. Nowak, and F. Jurie. Randomized clustering forests for image classification. *IEEE TPAMI*, 30(9):1632–1646, 2008.

[20] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *CVPR*, volume 2, pages 2161–2168, 2006.

[21] F. Odone, A. Barla, and A. Verri. Building kernels from binary strings for image matching. *IEEE Trans. Image Processing*, 14(2):169–180, 2005.

[22] F. Perronnin. Universal and adapted vocabularies for generic visual categorization. *IEEE TPAMI*, 30(7):1243–1256, 2008.

[23] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *CVPR*, 2008.

[24] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[25] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[26] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *ICCV*, volume 2, pages 1470–1477, 2003.

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

[28] T. Tuytelaars and C. Schmid. Vector quantizing feature space with a regular lattice. In *ICCV*, 2007.

[29] J. C. van Gemert, J.-M. Geusebroek, C. J. Veenman, and A. W. Smeulders. Kernel codebooks for scene categorization. In *ECCV*, 2008.

[30] M. Varma and D. Ray. Learning the discriminative power-invariance trade-off. In *ICCV*, 2007.

[31] J. Vogel and B. Schiele. Semantic modeling of natural scenes for content-based image retrieval. *IJCV*, 72(2):133–157, 2007.

[32] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS 21*, pages 1753–1760, 2009.

[33] J. Winn, A. Criminisi, and T. Minka. Object categorization by learned universal visual dictionary. In *ICCV*, volume 2, pages 1800–1807, 2005.

[34] J. Wu and J. M. Rehg. Where am I: Place instance and category recognition using spatial PACT. In *CVPR*, 2008.

[35] J. Wu and J. M. Rehg. CENTRIST: A visual descriptor for scene categorization. Technical Report GIT-GVU-09-05, GVU Center, Georgia Institute of Technology, 2009.

[36] L. Yang, R. Jin, R. Sukthankar, and F. Jurie. Unifying discriminative visual codebook generation with classifier training for object category recognition. In *CVPR*, 2008.