Journal of Machine Learning Research 12 (2011) 3097-3118 Submitted 10/10; Revised 7/11; Published 11/11

Efficient and Effective Visual Codebook Generation Using Additive Kernels

Jianxin Wu JXWU@NTU.EDU.SG
Wei-Chian Tan SGWEICHIAN@GMAIL.COM
School of Computer Engineering
Nanyang Technological University
Singapore, 639798, Singapore

James M. Rehg REHG@CC.GATECH.EDU
Center for Behavior Imaging
School of Interactive Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA

Editor: Ben Taskar

Abstract

Common visual codebook generation methods used in a bag of visual words model, for example, 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 with histogram features, the Histogram Intersection Kernel (HIK) is more effective than the Euclidean distance in supervised learning tasks. In this paper, we demonstrate that HIK can 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 the standard k-means. The HIK codebooks have consistently higher recognition accuracy over k-means codebooks by 2–4% in several benchmark object and scene recognition data sets. The algorithm is also generalized to arbitrary additive kernels. Its speed is thousands of times faster than a naive implementation of the kernel k-means algorithm. In addition, we propose a one-class SVM formulation to create more effective visual code words. Finally, we show that the standard k-median clustering method can be used for visual codebook generation and can act as a compromise between the HIK / additive kernel and the k-means approaches.

Keywords: visual codebook, additive kernel, histogram intersection kernel

1. Introduction

Bag of visual words (BOV) is currently a popular approach to object and scene recognition in computer vision. Local features are extracted from an image, and the image is then considered as a *bag of features*, that is, completely ignoring the spatial relationship among features. Probably due to the lack of an efficient and effective mechanism to encode spatial information among features, BOV is widely adopted in vision tasks. A typical BOV-based method consists of the following stages:

- **Extract features.** Visual features and their corresponding descriptors are extracted from local image patches. Two typical visual descriptors are SIFT by Lowe (2004) and HOG by Dalal and Triggs (2005). Usually two ways are used to determine where to extract local features. Some methods extract features at certain detected interest points. Other methods

densely sample local features in a regular grid of pixel locations, for example, in Lazebnik et al. (2006). Visual descriptors extracted from these local patches are considered as feature vectors that describe these local regions.

- **Generate a codebook and map features to visual code words.** A visual codebook is a method that divides the space of visual descriptors into several regions. Features in one region correspond to the same *visual code word*, which is represented by an integer between 1 and the 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. For example, SVM is a frequently used learner in BOV models for object and scene recognition.

The quality of the visual codebook has a significant impact on the success of BOV-based methods. Popular and successful methods for object and scene categorization typically employ unsupervised learning methods (for example, k-means clustering or Gaussian Mixture Model) to obtain a visual codebook. When there is a need to compute the dissimilarity of two feature vectors, the Euclidean distance is the most frequently used metric.

However, in spite of its mathematical simplicity and efficacy in many other applications, we find that the Euclidean distance is not the most suitable similarity (or dissimilarity) measure for creating visual codebooks.

Two observations support our argument. First, most of the popular visual descriptors are based on histograms of various image measurements such as spatial gradients, optical flow, or color. For example, both SIFT and HOG use histograms of pixel intensity gradients in their descriptors. Second, for the case of supervised classification, it has been shown that the ℓ_2 distance is not the most effective metric for comparing two histograms, for example, in Maji et al. (2008). In particular, the Histogram Intersection Kernel (HIK) was demonstrated to give significantly improved results. Other similarity measures designed for comparing histograms, for example, the χ^2 measure, have also exhibited higher accuracy in SVM classification than the dot-product kernel (which corresponds to the Euclidean distance). One common characteristic of HIK and χ^2 is that they are instances in a family of kernels called the additive kernel (Maji and Berg, 2009; Vedaldi and Zisserman, 2010).

In this paper we demonstrate that HIK and other additive kernels can be used to generate better visual codebooks with unsupervised learning, in comparison to the popular k-means clustering method. The proposed methods are simple to implement, and our software implementations are freely available. We show that using roughly twice the computational time of the standard k-means based method (which uses the ℓ_2 distance), we can gain consistent accuracy improvements of 2–4% across a diverse collection of object and scene recognition problems. Specifically, this paper makes four contributions:

First, we show that HIK generates better codebooks and thus improves recognition accuracy. We generalize and speedup the method in Maji et al. (2008), such that the generation and application of HIK codebook has the same theoretical complexity as the standard k-means. Our proposed method achieves consistent performance improvements over k-means codebooks, and has established state-of-the-art performance numbers for four benchmark object and scene recognition data sets. We also show that a one-class SVM formulation can be used to improve the effectiveness of HIK codebooks, by providing well-separated, compact clusters in the histogram feature space.

Second, we show that all additive kernels can be used to efficiently generate visual codebooks. The HIK visual codebook creation method is also generalized to be compatible with any additive

kernel, while the learning cost remains unchanged as $O(nmd)$, where n, m, and d are the number of features to be clustered, the codebook size, and the feature dimension, respectively. In contrast, a naive implementation has complexity $O(n^2d)$. Since $n \gg m$, in practice the speedup is more than three orders of magnitude. Similar recognition accuracies are observed for different additive kernels, including HIK and χ^2. More generally, we suggest that *HIK (or other additive kernel such as χ^2) should be used whenever two histograms are compared.*

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

Finally, we validate our method through experiments on standard data sets, using both the SIFT feature and CENTRIST, a recently proposed feature based on CENsus TRansform hISTogram (Wu and Rehg, 2011), which has been shown to offer performance advantages for scene classification.

The rest of the paper is organized as follows.[1] Related methods are discussed in Section 2. Section 3 introduces the histogram intersection kernel and various other additive kernels, and presents a general kernel k-means based method for visual codebook generation. In Section 4 we propose methods to efficiently generate visual codebooks for HIK and other additive kernels. Experiments are shown in Section 5, and Section 6 concludes this paper.

2. Related Works

In this section we will briefly review two categories of related works: different kernels for comparing histograms, and various visual codebook generation methods.

The main point of this paper is that when histogram features are employed, the histogram intersection kernel or another additive kernel should be used to compare them. HIK was introduced by Swain and Ballard (1991) for color-based object recognition. Odone et al. (2005) demonstrated that HIK forms a positive definite kernel when feature values are non-negative integers, facilitating its use in SVM classifiers. Simultaneously, works such as Lowe (2004) and Dalal and Triggs (2005) 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 various recent research works (Maji et al., 2008; Wu, 2010; Vedaldi and Zisserman, 2010), based on techniques to accelerate the kernel evaluations. Additive kernels, which include HIK as one of its instances, have also shown excellent performance in SVM classification of histograms (Vedaldi and Zisserman, 2010).

In this paper, we extend the results of Maji et al. (2008) 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 the loss of accuracy.

On the visual codebook side, k-means is the most widely used method for visual codebook generation (Sivic and Zisserman, 2003). However, several alternative strategies have been explored. K-means usually positions its clusters almost exclusively around the densest regions. A mean-shift type clustering method was used to overcome this drawback in Jurie and Triggs (2005). There are also information theoretic methods that try to capture the "semantic" common visual components by minimizing information loss (Liu and Shah, 2007; Lazebnik and Raginsky, 2009). An extreme

1. A preliminary version of portions of this work has been published in Wu and Rehg (2009).

method was presented in Tuytelaars and Schmid (2007), which divided the space of visual descriptors into regular lattice instead of learning a division of the space from training data. There are also efforts to build hash functions (that is, multiple binary functions / hash bits) in order to accelerate distance computations (Weiss et al., 2009). Recently, sparse coding is also used to vector quantize visual descriptors, for example, in Yang et al. (2009) and Gao et al. (2010). In this work, we propose a new alternative to k-means, based on the histogram intersection kernel and other additive kernels.

In k-means based methods, a visual code word is usually represented by the cluster center (that is, the average of all features that belong to this code word), which is simple and fast to compute. It was discovered that assigning a feature to multiple code words (which is also termed as soft-assignment) may improve the codebook quality (Philbin et al., 2008; van Gemert et al., 2008). Within a probabilistic framework, code words can be represented by the Gaussian Mixture Model (GMM) (Perronnin, 2008; Winn et al., 2005). GMM has better representation power than a single cluster center. However, it requires more computational power. Another interesting representation is the hyperfeature in Agarwal and Triggs (2008), 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. Nistér and Stewénius (2006) used a tree structure to divide the space of visual descriptors hierarchically and Moosmann et al. (2008) 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, Yang et al. (2008) unified the codebook generation step with the classifier learning step seamlessly. In another interesting research work, Vogel and Schiele (2007) manually specified a few code words and used supervised learning to learn these concepts from manually labeled examples.

It is worth noting that all of these previous methods used the ℓ_2 distance metric (except Gao et al., 2010 which followed our previous work Wu and Rehg, 2009, and used HIK). They could therefore in principle be improved through the use of HIK or other additive kernels.

3. Visual Codebook for Additive Kernels

In this section we will first introduce the Histogram Intersection Kernel (HIK), and then its generalization to the additive kernel case. In order to make our presentation clearer, we will use boldface characters (for example, $\mathbf{x}$) to represent vectors. The scalar x_j is the j-th dimension of $\mathbf{x}$.

3.1 Histogram Intersection Kernel

Let $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}_+^d$ be a histogram of non-negative real values with d histogram bins, where $\mathbb{R}_+$ is the set of non-negative real numbers. $\mathbf{x}$ could represent an image (for example, a histogram of visual code words in the bag of visual words model) or an image patch (for example, a SIFT visual descriptor). The histogram intersection kernel κ_{HI} is defined as follows (Swain and Ballard, 1991):

$$\kappa_{\mathrm{HI}}(\mathbf{x}_1, \mathbf{x}_2) = \sum_{j=1}^{d} \min(x_{1,j}, x_{2,j}) \ . \tag{1}$$

It is proved in Wu (2010) that HIK is a valid positive definite kernel when the data $\mathbf{x} \in \mathbb{R}_+^d$. Thus there exists a mapping ϕ that maps any histogram $\mathbf{x}$ to a corresponding vector $\phi(\mathbf{x})$ in a high dimensional (possibly infinite dimensional) feature space Φ, such that $\kappa_{\mathrm{HI}}(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_2)$.

Through the nonlinear mapping ϕ, histogram similarity is equivalent to a dot product in the feature space Φ. Furthermore, when the feature values are non-negative integers, that is, $\mathbf{x} \in \mathbb{N}^d$, we can explicitly find the mapping $\phi(\cdot)$. If we constrain the feature values to be bounded from above by $\bar{v}$, that is, $0 \leq x_j \leq \bar{v}$ for all $\mathbf{x}$ and $1 \leq j \leq d$, the mapping $\phi(\cdot)$ for HIK is then the following unary representation $B(\cdot)$ of an integer (Odone et al., 2005):

$$B(x) : x \mapsto \left[\underbrace{1 \cdots 1}_{x\ 1's} \underbrace{0 \cdots 0}_{\bar{v}-x\ 0's} \right] .$$

It is easy to verify that $\kappa_{\mathrm{HI}}(\mathbf{x}_1, \mathbf{x}_2) = B(\mathbf{x}_1)^T B(\mathbf{x}_2)$, in which $B(\mathbf{x})$ is the concatenation of $B(x_1), B(x_2), \ldots, B(x_d)$. Note that $B(x_j) \in \mathbb{R}^{\bar{v}}$ and $B(\mathbf{x}) \in \mathbb{R}^{d\bar{v}}$.

This kernel trick makes it possible to use HIK in creating codebooks, while keeping the simplicity of k-means clustering. That is, we may use a kernel k-means algorithm (Schölkopf et al., 1998) to generate visual codebooks. In Algorithm 1, histograms are compared using HIK instead of the inappropriate Euclidean distance if we set $\phi(\cdot) = B(\cdot)$.[2]

When the data points $\mathbf{x} \in \mathbb{R}^d$, that is, allowing negative feature values, HIK is not a positive definite kernel. And it can not be used in Algorithm 1 to generate visual codebooks.[3]

3.2 Additive Kernels

Algorithm 1 is not restricted to work only with the histogram intersection kernel. It is a general kernel k-means algorithm which can be used together with any positive definite kernel. In particular, we are interested in a family of kernels called the *additive kernels* (Maji and Berg, 2009). Algorithm 1 instantiated with an additive kernel can be greatly accelerated, for which we will present in Section 4.

An additive kernel is a positive definite kernel that can be expressed in the following form

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = \sum_{j=1}^{d} \hat{\kappa}(x_{1,j}, x_{2,j}) .$$

A positive semidefinite function $\hat{\kappa}(\cdot,\cdot)$ is used to compute the similarity of two scalar values. An additive kernel κ then compares two vectors by comparing and summing up every dimension of these two vectors using $\hat{\kappa}$.

It is obvious that the histogram intersection kernel is an instance of the additive kernels. In fact, a family of additive kernels can be derived from HIK. If $g(\cdot)$ is a non-negative and non-decreasing function, then the generalized histogram intersection kernel,

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = \sum_{j=1}^{d} g(\min(x_{1,j}, x_{2,j})) ,$$

is a valid additive kernel. HIK corresponds to $g(x) = x, x \geq 0$. The GHI kernel proposed in Boughorbel et al. (2005) is also an instance of this family with $g(x) = x^\beta$ for $\beta > 0$ and $x \geq 0$. In this paper

2. Note that since *k-means++* is used in Algorithm 1 and it is a randomized algorithm, two runs of Algorithm 1 with the same input will possibly generate different results.
3. HIK is a conditionally positive definite kernel when $\mathbf{x} \in \mathbb{R}^d$ (Maji and Berg, 2009). It can still be used in some SVM solvers.

Algorithm 1 Codebook Generation Using Kernel k-means

1: **Input**: $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}_+^d$ (n input histograms), m (size of the codebook), and ε (tolerance).

2: {The output is a function that maps a histogram to its visual code word index, $w_1(\mathbf{x}_*) : \mathbb{R}_+^d \to \{1, \ldots, m\}$.}

3: $t \leftarrow 0$, {Initialize t, the iteration counter, to 0.}
$\varepsilon^t \leftarrow \infty$. {Initialize the current clustering error to ∞.}

4: Initialize the visual codebook. First, use the *k-means++* method (Arthur and Vassilvitskii, 2007) to choose m distinct examples from the input set $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. We denote these examples as $\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_m$. Second, use $\mathbf{m}_i = \phi(\bar{\mathbf{x}}_i)$, $i = 1, 2, \ldots, m$, as the initial visual code words. $\phi(\cdot)$ is the mapping associated with a positive definite kernel.

5: **repeat**

6: For every input histogram $\mathbf{x}_i$, find the visual code word that $\mathbf{x}_i$ belongs to, and denote the index of this visual code word as l_i:

$$l_i \leftarrow \arg\min_{1 \le j \le m} \|\phi(\mathbf{x}_i) - \mathbf{m}_j\|^2, \quad 1 \le i \le n.$$

7: For every visual code word $\mathbf{m}_i$, find the set of the input histograms that belong to this visual code word, and denote this set as π_i:

$$\pi_i = \{j | l_j = i, 1 \le j \le n\}, \quad 1 \le i \le m.$$

8: For every visual code word $\mathbf{m}_i$, update it to be the centroid of input histograms that belong to this visual code word:

$$\mathbf{m}_i \leftarrow \frac{\sum_{j \in \pi_i} \phi(\mathbf{x}_j)}{|\pi_i|}, \quad 1 \le i \le m.$$

9: Update the iteration counter and compute the current clustering error:

$$t \leftarrow t + 1,$$

$$\varepsilon^t = \frac{1}{n} \sum_{i=1}^{n} \|\phi(\mathbf{x}_i) - \mathbf{m}_{l_i}\|^2.$$

10: **until** $\varepsilon^{t-1} - \varepsilon^t \le \varepsilon$.

11: **Output**: For any histogram $\mathbf{x}_* \in \mathbb{R}_+^d$, its corresponding visual code word index is:

$$w_1(\mathbf{x}_*) = \arg\min_{1 \le i \le m} \|\phi(\mathbf{x}_*) - \mathbf{m}_i\|^2. \tag{2}$$

we will explore one specific instance from this family, which we call exponential HIK (or eHIK), defined as

$$\kappa_{\text{eHI}}(\mathbf{x}_1, \mathbf{x}_2) = \sum_{j=1}^{d} \min(e^{\gamma x_{1,j}}, e^{\gamma x_{2,j}}), \quad \gamma > 0 \ .$$

χ^2 is another additive kernel that has been used for comparing histograms. The original χ^2 measure is defined as $\chi^2(x_1, x_2) = \frac{(x_1 - x_2)^2}{x_1 + x_2}$ for $x_1, x_2 \in \mathbb{R}_+$. Alternatively, a variant of χ^2 is explored

in Vedaldi and Zisserman (2010) for SVM classification when the feature values are positive:

$$\kappa_{\chi^2}(\mathbf{x}_1, \mathbf{x}_2) = \sum_{j=1}^{d} \frac{2x_{1,j}x_{2,j}}{x_{1,j} + x_{2,j}} .$$

We will adopt this definition in our experiments.

3.3 K-median Codebook Generation

Although k-means (or equivalently, using the 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

$$2\kappa_{\mathrm{HI}}(\mathbf{x}_1, \mathbf{x}_2) + \|\mathbf{x}_1 - \mathbf{x}_2\|_1 = \|\mathbf{x}_1\|_1 + \|\mathbf{x}_2\|_1 ,$$

in which $\|\mathbf{x}\|_1$ is the ℓ_1 norm of $\mathbf{x}$. In cases when $\|\mathbf{x}\|_1$ is constant for any histogram $\mathbf{x}$, 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 which 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. The only difference between k-median and k-means is that k-median uses ℓ_1 instead of ℓ_2 as the distance metric.

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 (Everingham et al., 2006). In Section 5, we empirically compare visual codebooks generated by the k-median algorithm to those generated by both the k-means algorithm and the proposed additive kernel k-means method.

4. The Efficient Additive Kernel k-means Clustering Method

As mentioned in Section 3.2, additive kernels are attractive for kernel k-means because very fast clustering is possible for these kernels. In this section, we first propose an efficient kernel k-means algorithm for the histogram intersection kernel, and then generalize the algorithm to all additive kernels.

4.1 Common Computation Bottleneck

Given n examples in $\mathbb{R}^d$, the standard k-means clustering method (that is, $\phi(\mathbf{x}) = \mathbf{x}$ in Algorithm 1) requires $O(nmd)$ steps in one iteration (from line 5 to line 10). Similarly, the k-median algorithm also requires $O(nmd)$ steps in one iteration.

When $\phi(\mathbf{x}) \neq \mathbf{x}$, the centers $\mathbf{m}_i$ are vectors in the unrealized, high dimensional space Φ. $\mathbf{m}_i$ might even be infinite dimensional for some kernels (for example, the RBF kernel). The computations are

then carried out in the following way (using the usual kernel trick $\phi(\mathbf{x}_1)^T\phi(\mathbf{x}_2) = \kappa(\mathbf{x}_1, \mathbf{x}_2)$ such that $\mathbf{m}_i$ does not need to be explicitly generated):

$$\begin{aligned}
&\|\phi(\mathbf{x}_*) - \mathbf{m}_i\|^2 \\
=&\left\|\phi(\mathbf{x}_*) - \frac{\sum_{j\in\pi_i}\phi(\mathbf{x}_j)}{|\pi_i|}\right\|^2 \\
=&\|\phi(\mathbf{x}_*)\|^2 + \frac{1}{|\pi_i|^2}\sum_{j,k\in\pi_i}\kappa(\mathbf{x}_j,\mathbf{x}_k) - \frac{2}{|\pi_i|}\sum_{j\in\pi_i}\kappa(\mathbf{x}_*,\mathbf{x}_j). \qquad (3)
\end{aligned}$$

The first term in Equation 3 does not affect the result in lines 6 and 11 of Algorithm 1. The second term does not change within a specific iteration of Algorithm 1. Thus, we need to compute this term only once for every visual code word in each iteration. Most of the computations are then spent in computing the last term $\sum_{j\in\pi_i}\kappa(\mathbf{x}_*,\mathbf{x}_j)$.

A naive implementation to compute this term will be costly. For example, if we use the histogram intersection kernel and compute this term literally using Equation 1, the complexity is $O(|\pi_i|d)$. The complexity of line 6 in Algorithm 1 will be on the order

$$\sum_{i=1}^{n} m|\pi_{l_i}|d = \sum_{i=1}^{m}\left(\sum_{j:l_j=i} m|\pi_i|d\right) = \sum_{i=1}^{m} m|\pi_i|^2 d\ ,$$

since there are $|\pi_i|$ input histograms $\mathbf{x}_j$ satisfying $l_j = i$. Using the Cauchy-Schwarz inequality, it is clear that

$$\sum_{i=1}^{m}|\pi_i|^2 \geq \frac{1}{m}\left(\sum_{i=1}^{m}|\pi_i|\right)^2 = \frac{n^2}{m}\ ,$$

because $\sum_{i=1}^{m}|\pi_i| = n$. In practice, the sizes of π_i are usually similar for different i, and one iteration of this naive implementation will have complexity $O(n^2 d)$. We generally have $n \gg m$, thus a kernel k-means will be much more expensive than the standard k-means. In summary, the last term in Equation 3 is the bottleneck in the computations.

The form of this term, $\sum_{j\in\pi_i}\kappa(\mathbf{x}_*,\mathbf{x}_j)$, is similar to the binary SVM classifier, which has the following form:

$$\text{sign}\left(\sum_{i\in\pi}\alpha_i y_i \kappa(\mathbf{x}_*,\mathbf{x}_i) + \rho\right)\ , \qquad (4)$$

where $\mathbf{x}_i$, α_i, and y_i are, respectively, the support vectors, and their corresponding weights and labels.

Based on these observations, we propose a more general objective,

$$f(\mathbf{x}_*) = \sum_{i\in\pi} c_i \kappa(\mathbf{x}_*,\mathbf{x}_i)\ , \qquad (5)$$

where π indexes a set of histograms (data points to be clustered, or support vectors) and c_i are constant coefficients. Note that both Equation 4 and the last term in Equation 3 are special forms of Equation 5, with $c_i = \alpha_i y_i$ and $c_i = 1$, respectively.

Our goal is then to reduce the complexity of Equation 5 to $O(d)$ (the same complexity as that of standard k-means when $\phi(\mathbf{x}) = \mathbf{x}$), which will in turn yield efficient kernel k-means clustering and SVM testing methods. We will first present the algorithm for HIK, and then its generalization to arbitrary additive kernels.

4.2 Efficient Computation of HIK

Maji et al. (2008) proposed fast methods to compute Equation 4 for the histogram intersection kernel to improve the testing speed of HIK SVM classifiers, which achieved an exact answer of Equation 4 in $O(d\log_2|\pi|)$ steps and an approximate answer in $O(d)$ steps. In this paper we propose a variant that finds the exact answer for Equation 5 in $O(d)$ steps when the feature values are non-negative integers.

A histogram of visual code word indexes has the property that every histogram component is a non-negative integer, that is, it is a vector in $\mathbb{N}^d$. Similarly, a visual descriptor histogram can usually be transformed into the space $\mathbb{N}^d$. For example, the SIFT descriptors are stored as vectors in $\mathbb{N}^{128}$. In general, a vector in $\mathbb{R}^d_+$ can be transformed into $\mathbb{N}^d$ by a linear transformation followed by quantization.

In the rest of this paper, we assume that any histogram $\mathbf{x} = (x_1, \ldots, x_d)$ satisfies that $x_i \in \mathbb{N}$ and $0 \le x_i \le \bar{v}$ for all i. Then the quantity $f(\mathbf{x}_*)$ can be computed as follows:

$$\begin{aligned} f(\mathbf{x}_*) &= \sum_{i\in\pi} c_i \kappa_{\mathrm{HI}}(\mathbf{x}_*, \mathbf{x}_i) \\ &= \sum_{i\in\pi} \sum_{1\le j\le d} c_i \min(x_{*,j}, x_{i,j}) \\ &= \sum_{1\le j\le d} \left(\sum_{i\in\pi} c_i \min(x_{*,j}, x_{i,j}) \right) \\ &= \sum_{1\le j\le d} \left(\sum_{i:x_{*,j}\ge x_{i,j}} c_i x_{i,j} + x_{*,j} \sum_{i:x_{*,j}<x_{i,j}} c_i \right). \end{aligned} \tag{6}$$

Note that the two summands in Equation 6 can both be pre-computed. It is shown in Maji et al. (2008) that Equation 6 is a piece-wise linear function of $x_{*,j}$. Thus using a binary search for $x_{*,j}$, Equation 6 can be computed in $O(d\log|\pi|)$ steps in Maji et al. (2008).

However, since we assume that $x_{*,j}$ is an integer in the range $[0\ \bar{v}]$, we have an even faster method. Different dimensions of $\mathbf{x}_*$ make independent contributions to $f(\mathbf{x}_*)$ in Equation 6, because of the additive property. Thus it is sufficient to solve the problem for one single feature dimension at a time. And because there are only $\bar{v}+1$ possibilities for $x_{*,j}$ given a fixed j, we just need to pre-compute the solutions for these $\bar{v}+1$ values. Let T be a table of size $d\bar{v}$, with

$$T(j,k) \leftarrow \sum_{i:k\ge x_{i,j}} c_i x_{i,j} + k \sum_{i:k<x_{i,j}} c_i$$

for all $1 \le j \le d$ and $1 \le k \le \bar{v}$. Then it is clear that

$$f(\mathbf{x}_*) = \sum_{j=1}^{d} T(j, x_{*,j}) \,. \tag{7}$$

This method is summarized in Algorithm 2. Note that since $T(j,0) = 0$ for all j, there is no need to store it.

It is obvious that $f(\mathbf{x}_*)$ can be evaluated in $O(d)$ steps after the table T is pre-computed. And because Algorithm 2 only involves table lookup and summation, it is faster (that is, has less overhead) than the approximation scheme in Maji et al. (2008), which is also $O(d)$. Depending on the

Algorithm 2 Fast Computation of HIK Sums

1: **Input**: n histograms $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in $\mathbb{N}^d$, with $0 \leq x_{i,j} \leq \overline{v}$ for $1 \leq i \leq n$ and $1 \leq j \leq d$.
2: {The output is a fast method to compute

$$f(\mathbf{x}_*) = \sum_{i=1}^{n} c_i \kappa_{\mathrm{HI}}(\mathbf{x}_*, \mathbf{x}_i) \ ,$$

where $\mathbf{x}_* \in \mathbb{N}^d$ and $0 \leq x_{*,j} \leq \overline{v}, \forall\ 1 \leq j \leq d$.}
3: Create T, a $d \times \overline{v}$ table.
4: For $1 \leq j \leq d$, $1 \leq k \leq \overline{v}$,

$$T(j,k) \leftarrow \sum_{i:k \geq x_{i,j}} c_i x_{i,j} + k \sum_{i:k < x_{i,j}} c_i \ .$$

5: **Output**:

$$f(\mathbf{x}_*) = \sum_{j=1}^{d} T(j, x_{*j}) \ .$$

relative size of $\overline{v}$ and the number of approximation bins used in Maji et al. (2008), Algorithm 2's storage requirement, $O(\overline{v}d)$, could be larger or smaller than that of Maji et al. (2008). It is also worth noting that under our assumptions, Algorithm 2's result is precise rather than approximate.

Both the complexity of the pre-computation and the storage requirement are linear in $\overline{v}$, which is a parameter specified by users.[4] Our experiments show that while too small a $\overline{v}$ usually produces inferior results, a large $\overline{v}$ does not necessarily improve performance. In this paper, we choose $\overline{v} = 128$, which seems to give the best results in our experiments.

Our algorithm has the same computational complexity as the standard k-means when generating a visual codebook or mapping histograms to visual code word indexes (that is, Equation 2 or Equation 3). 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 use the non-linear similarity measure κ_{HI} that is most suitable for comparing histograms.

4.3 Generalization to Additive Kernels

Algorithm 2 can be generalized from HIK to arbitrary additive kernels. The following two conditions are also satisfied by all additive kernels: different dimensions of $\mathbf{x}_*$ make independent contributions to $f(\mathbf{x}_*)$; and there are only $\overline{v}+1$ possibilities for $x_{*,j}$ when j is fixed. We just need to find an appropriate value for $T(j,k)$, and Equation 7 is then valid for all additive kernels. Of course, we assume that the feature values are natural numbers bounded from above by $\overline{v}$.

For an additive kernel

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = \sum_{j=1}^{d} \hat{\kappa}(x_{1,j}, x_{2,j}) \ ,$$

4. A simple implementation to pre-compute the table T takes $O(nd\overline{v})$ steps. We will present a $O(d(n+\overline{v}))$ implementation of Algorithm 2 in Section 4.3.

we propose Algorithm 3 to efficiently assign values to the table T. Considering a fixed dimension j, by the independence property, we have

$$T(j,k) = \sum_{i=1}^{n} c_i \hat{\kappa}(x_{i,j}, k) \; . \tag{8}$$

In general, it takes $O(nd\bar{v})$ steps to fill the table T for an additive kernel if we literally translate Equation 8. However, for additive kernels, Algorithm 3 uses a sequential update strategy whose complexity is only $O(d(n+\bar{v}^2))$.

Algorithm 3 Assign values to T for an arbitrary additive kernel

Input: n histograms $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in $\mathbb{N}^d$, with $0 \le x_{i,j} \le \bar{v}$, for all $1 \le i \le n$ and $1 \le j \le d$; and an additive kernel $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \sum_{j=1}^{d} \hat{\kappa}(x_{1,j}, x_{2,j})$.
{The output is a table $T \in \mathbb{R}^{d\bar{v}}$ for fast computation of Equation 5, where $\mathbf{x}_* \in \mathbb{N}^d$ and $0 \le x_{*,j} \le \bar{v}, \forall\, 1 \le j \le d$.}
for $j = 1, \ldots, d$ **do**
  Create a vector $\mathbf{h} \in \mathbb{R}^{\bar{v}}$, and $\mathbf{h} \leftarrow \mathbf{0}$.
  for $i = 1, \ldots, n$ **do**
    $h_{x_{i,j}} \leftarrow h_{x_{i,j}} + c_i$.
  end for
  $T(j,0) = \sum_{i=1}^{n} \hat{\kappa}(x_{i,j}, 0)$.
  for $k = 1, \ldots, \bar{v}$ **do**
    $T(j,k) \leftarrow T(j,k-1) + \sum_{v=0}^{\bar{v}} h_v \left(\hat{\kappa}(v,k) - \hat{\kappa}(v,k-1) \right)$.
  end for
end for
Output: A table T such that

$$f(\mathbf{x}_*) = \sum_{j=1}^{d} T(j, x_{*j}) \; .$$

For a fixed feature dimension j,

$$\begin{aligned}
& T(j,k) - T(j,k-1) \\
=& \sum_{i=1}^{n} c_i \left(\hat{\kappa}(x_{i,j}, k) - \hat{\kappa}(x_{i,j}, k-1) \right) \\
=& \sum_{v=0}^{\bar{v}} \left(\sum_{i: x_{i,j} = v} c_i \left(\hat{\kappa}(v,k) - \hat{\kappa}(v,k-1) \right) \right) \\
=& \sum_{v=0}^{\bar{v}} \left(\left(\sum_{i: x_{i,j} = v} c_i \right) \left(\hat{\kappa}(v,k) - \hat{\kappa}(v,k-1) \right) \right) .
\end{aligned}$$

In Algorithm 3, we make a weighted histogram $\mathbf{h}$ for the j-th dimension such that $h_v = \sum_{i: x_{i,j} = v} c_i$. This is the first inner-loop, and its complexity is $O(n)$. It then takes $O(\bar{v}^2)$ steps to sequentially update the j-th row of the table T. In total, Algorithm 3 takes $O(d(n+\bar{v}^2))$ steps. Since in general $n \gg \bar{v}$, Algorithm 3 is more efficient than a $O(nd\bar{v})$ method.

	k-means	Kernel k-means (naive)	Kernel k-means (proposed)
Storage	md	nd	$md\bar{v}$
Running time	$O(nmd)$	$O(n^2d)$	$O(nmd)$

Table 1: Space and running time requirements of the standard k-means, a naive implementation of kernel k-means, and the proposed method. The space requirement does not include the memory required to store the input histograms. The running time requirement shows the complexity of one kernel k-means iteration.

One difference between Algorithms 2 and 3 is that $T(j,0)$ may not be equal to 0 in Algorithm 3. A more important difference is that Algorithm 2 can be further improved to $O(d(n+\bar{v}))$. Note that in Algorithm 2, $\hat{\kappa}(x,y) = \min(x,y)$. We then have $\hat{\kappa}(v,k) - \hat{\kappa}(v,k-1)$ equals 1 if $v > k-1$ and 0 if otherwise. In consequence, $T(j,k) - T(j,k-1) = \sum_{v=k}^{\bar{v}} h_v$, which can in turn be sequentially updated and takes only $O(1)$ steps to compute for every k value. The complexity of Algorithm 2 is then $O(d(n+\bar{v}))$.

In practice, we generally have $n \gg m$, $n \gg d$, and $n \gg \bar{v}$. Typical values in our experiments are $n = 300,000$, $d = 128$ or $d = 256$, $m = 200$, and $\bar{v} = 128$. The complexity of kernel k-means is then dominated by the line 6 of Algorithm 1. Space and running time requirements of various algorithms are summarized in Table 1. The naive implementation does not need additional storage during the visual codebook generation step. However, it needs to keep all input histograms (nd numbers) for the quantization step. The other two methods do not need to keep input histograms for quantization.

The theoretical complexities in Table 1 match the empirical running time in our experiments. For example, in one experiment using the Caltech 101 data set (refer to Section 5), the naive implementation and the proposed method took 2403 and 1.2 seconds, respectively. The empirical speedup ratio is 2000. In this experiment, the theoretical speedup ratio is $O(n/m)$, and $n/m \approx 1200$. Since the naive implementation is impractical for large-scale problems, we will not provide empirical results of this method in our experiments.[5]

4.4 One Class SVM Codebook Representation

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

This assumption rarely holds. Different regions usually have very different densities and covariance structures. Simply dividing the space $\mathbb{R}^d$ 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 of codebooks and mapping from visual features to code words will require much more storage and computational resources than we can afford.

5. Since one kernel k-means iteration takes more than 2400 seconds, it will take months to finish running all the experiments in Section 5.

We propose to use one-class SVM (Schölkopf et al., 2001) to represent the divided regions in an effective and computationally-efficient way. Given a set of histograms in a region $\mathbf{x}_\pi = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, we construct a one-class SVM with parameter $\nu \in (0,1]$,

$$\text{sign}\left(\sum_{i\in\pi} \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i) + \rho\right), \tag{9}$$

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 $\mathbf{x}_\pi$ (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 (that is, on which Equation 9 are less than 0), and at the same time a lower bound on the fraction of support vectors (that is, $\alpha_i \neq 0$) (Schölkopf et al., 2001).

The one-class SVM 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. Equation 9 is evaluated in $O(d)$ steps because it is again a special case of Algorithm 2. We propose Algorithm 4 to use one-class SVM to generate visual code words. Note that we use the space $\mathbb{R}^d$ because Algorithm 4 is not restricted to $\mathbb{N}^d$. In this paper, we set the parameter $\nu = 0.2$.

Algorithm 4 One-class SVM Code Word Generation

1: **Input**: Same as that of Algorithm 1.

2: Use Algorithm 1 to generate the divisions π_i ($i = 1, \ldots, m$) from the input histograms $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in $\mathbb{R}^d_+$.

3: For each division $1 \leq i \leq m$, train a one-class SVM from its data $\mathbf{x}_{\pi_i}$ with a parameter ν,

$$w_2^i(\mathbf{x}_*) = \sum_{j\in\pi_i} \alpha_j \kappa(\mathbf{x}_*, \mathbf{x}_j) + \rho_i \,. \tag{10}$$

4: **Output**: For any histogram $\mathbf{x}_* \in \mathbb{R}^d_+$,

$$w_2(\mathbf{x}_*) = \arg\max_{1\leq i\leq m} w_2^i(\mathbf{x}_*) \,.$$

In many applications, a histogram $\mathbf{x} = (x_1, \ldots, x_d)$ satisfies the condition that $\|\mathbf{x}\|_1 = \sum_{j=1}^d x_j = N$ is a constant. Under this condition, Equation 10 is equivalent to

$$w_2^i(\mathbf{x}_*) = r_i^2 - \|\phi(\mathbf{x}_*) - \mathbf{m}_i\|^2 \,,$$

where $\mathbf{m}_i = \sum_{j\in\pi_i} \alpha_j \mathbf{x}_j$ and $r_i^2 = N + \|\mathbf{m}_i\|^2 - 2\rho_i$. 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 $\mathbf{m}_i$ with radius r_i. A sphere in Φ is different from a usual k-means sphere because it respects the similarity measure κ, and its radius r_i automatically adapts to the distribution of histograms in a visual word. Note that different kernels such as the dot-product kernel or κ_{HI} can be used in Algorithm 4.

5. Experiments

We validate the proposed methods using four benchmark data sets in computer vision: the Caltech 101 object recognition data set (Fei-Fei et al., 2004), the 15 class scene recognition data set (Lazebnik et al., 2006), the 8 class sports events data set (Li and Fei-Fei, 2007), and the 67 class indoor data set (Quattoni and Torralba, 2009).

5.1 Setup

In each data set, the available data is randomly split into a training set and a testing set based on published protocols on these data sets. 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 (that is, the average of diagonal entries in the confusion matrix).

The proposed algorithms can efficiently process a huge number of histogram features, for example, approximately 200k to 320k histograms are clustered across the first three data sets in less than 6 minute. In the 67 class indoor data set, more than 1 million histograms are clustered.

In the BOV model, we use 16×16 image patches and densely sample features over a grid with a spacing of 2, 4, or 8 pixels. We use two types of visual descriptors: SIFT for Caltech 101, CENTRIST (CENsus TRansform hISTogram, refer to Wu and Rehg (2011) for more details) for the scene, event, and indoor data sets.[6] All feature vectors are scaled and rounded such that a histogram only contains non-negative integers that approximately sum to 128 (thus $\overline{v} = 128$ is always valid.)

The first step is to use visual 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 Wu and Rehg (2008). An image is represented by the concatenation of histograms from all the 31 sub-windows, 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 the Sobel gradient image (which is 12400 dimensional). Following Boiman et al. (2008), we also sample features at 5 scales.

SVM is used for classification. LIBSVM (Chang and Lin, 2001) is used for the scene and sports data set. Since LIBSVM uses the 1-vs-1 strategy, it will produce too many classifiers for the Caltech 101 and indoor data set (more than 5000 and 2200 respectively). Therefore we instead use the Crammer & Singer formulation in BSVM (Hsu and Lin, 2006) for these two data sets. Since we are classifying histograms, we modified both LIBSVM and BSVM so that they are able to use the histogram intersection kernel.[7] It is observed that HIK is robust to the C parameter in SVM. For example, using the LIBSVM solver, classification accuracy remains almost unchanged after $C > 0.001$, as empirically showed in Wu (2010). Thus we do not use cross-validation to choose a different C value for every different training set. Instead, we use cross-validation to find $C = 2$ and

6. We will also evaluate the effect when these two feature types are switched in these data sets.

7. The methods proposed in this paper are publicly available in the libHIK package, which can be downloaded from `http://c2inet.sce.ntu.edu.sg/Jianxin/projects/libHIK/libHIK.htm`. The modified version of LIBSVM and BSVM are also included in libHIK.

$C = 0.03125$ for LIBSVM and BSVM respectively on a sample training set. These C values are then used on all data sets for LIBSVM and BSVM, respectively.

5.2 Main Results

We conducted several sets of experiments to validate the proposed algorithms. Experimental results are organized using the following rule: texts in the italic type summarize findings from one set of experiments and details are described after the italic texts. Mean / standard deviation values and paired t-tests are used to show the benefit of histogram kernel codebooks (Algorithm 1), while the Wilcoxon test is used for evaluating the one-class SVM code word generation method (Algorithm 4). We first present the main results, which are based on the experimental results summarized in Table 2.

In Table 2, sub-tables (a), (b), (c), and (d) are results for the Caltech 101, 15 class scene, 8 class sports, and the 67 class indoor data sets, respectively. κ_{HI} and κ_{LIN} means that a histogram intersection or a linear kernel visual codebook 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, which follows the protocol of previously published results on these data sets.

Histogram Intersection Kernel Visual Codebook (Algorithm 1) greatly improves classification accuracy. We compare the classification accuracies of systems that use Algorithm 1 with κ_{HI}, the standard k-means algorithm (that is, using κ_{LIN}), and k-median. From the experimental results in Table 2, it is obvious that in all four data sets, the classification accuracy with a κ_{HI}-based codebook is consistently higher than that with a k-means codebook. Using a paired t-test with significance level 0.05, the differences are statistically significant in 21 out of the 24 cases in Table 2, when comparing κ_{HI} and κ_{LIN} based codebooks. The three exceptions all come from the 8 class sports event data set, when one-class SVM is not used (that is, comparing the second row to the fifth row in Table 2c). HIK codebooks also have advantages over k-median codebooks in most cases.

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 data set, while k-means used 56 seconds in our experiments.

One-class SVM improves histogram intersection kernel code words (Algorithm 4). 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 (Demšar, 2006) to show the effect of Algorithm 4. Algorithm 4 improved the classification accuracy of the κ_{HI}-based method in 11 out of 12 cases in Table 2. The Wilcoxon test shows that the difference is significant at significance level 0.01.

In summary, using HIK codebooks and one-class SVM together generated the best results in almost all cases (best results are shown in boldface within each column of Table 2).

One-Class SVM degrades the standard k-means code words. It is interesting to observe a completely reversed trend when κ_{LIN} is used with one-class SVM. Applying Algorithm 4 in the standard k-means method reduced accuracy in all cases. Since a vector in $\mathbb{R}^d$ is not an appropriate understanding of a histogram with d bins, we conjecture that Algorithm 4 with κ_{LIN} produced a better division of the space $\mathbb{R}^d$, but probably a worse one in the space of histograms.

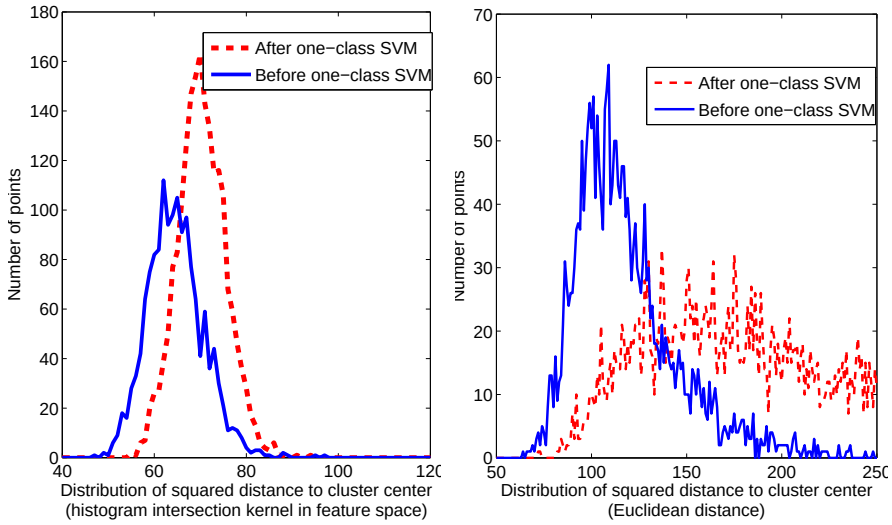

Figure 1: Effects of one-class SVM.

Figure 1 shows the effect of applying Algorithm 4 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.

K-median is a compromise between k-means and HIK codebooks. As shown in Table 2, HIK codebooks outperformed k-median codebooks in most cases.[8] However, k-median generally outperformed the popular k-means codebooks. Furthermore, k-median requires less memory than the proposed method. Qualitative comparisons of these methods are summarized in Table 3.

5.3 Experimental Results for Additive Kernels

Experiments with codebooks generated using the other two additive kernels (χ^2 and exponential HIK) are shown in Table 4. For ease of comparison, results of HIK (without one-class SVM) codebooks are also shown in Table 4.

HIK and χ^2 based codebooks have very similar accuracies, and both outperform the exponential HIK codebooks. However, all three additive kernel based codebooks generally have higher accuracies than the standard k-means codebook generation method. Since the time complexity of additive kernel based codebooks is the same as that of the k-means method, it is advantageous to apply such kernels in generating visual codebooks. For example, the χ^2 kernel in some cases leads to higher accuracies than the histogram intersection kernel.

5.4 Effects of Information Content

Next we study the effects of using different types and amounts of information, for example, different types of base features and step sizes in dense feature sampling.

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

	$B, s = 4$	$B, s = 8$	$\neg B, s = 8$
$\kappa_{\mathrm{HI}}, oc_{\mathrm{svm}}$	**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

	$B, s = 4$	$B, s = 8$	$\neg B, s = 8$
$\kappa_{\mathrm{HI}}, oc_{\mathrm{svm}}$	**43.01±0.81%**	**41.75±0.94%**	**35.09±1.04%**
$\kappa_{\mathrm{HI}}, \neg oc_{\mathrm{svm}}$	41.73±0.80%	40.07±0.27%	33.55±0.26%
k-median	41.81±1.11%	40.22±1.07%	34.04±1.56%
$\kappa_{\mathrm{LIN}}, oc_{\mathrm{svm}}$	35.94±1.14%	34.63±1.24%	28.69±1.04%
$\kappa_{\mathrm{LIN}}, \neg oc_{\mathrm{svm}}$	39.79±0.47%	38.28±0.39%	32.49±0.72%

(d) 67 class indoor, 80 train, 20 test

Table 2: Results of HIK, k-median and k-means codebooks and one-class SVM code words. The best result in each column is shown in **boldface**.

	HIK	k-median	k-means
Computation time	2	2	1
Codebook storage size	$\overline{v}$	1	1

Table 3: Comparison of three codebook generation methods. k-means is used as a baseline, that is, a value '2' means approximately 200% of that of k-means.

	$B, s = 4$	$B, s = 8$	$\neg B, s = 8$
κ_{HI}	66.54±0.58%	64.11±0.84%	60.33±0.95%
κ_{χ^2}	**67.35±0.77%**	**64.31±1.28%**	**60.63±0.85%**
κ_{eHI}	66.18±0.72%	63.71±0.58%	57.13±0.89%

(a) Caltech 101, 15 train, 20 test

	$B, s = 4$	$B, s = 8$	$\neg B, s = 8$
κ_{HI}	83.59±0.45%	**83.74±0.42%**	**81.77±0.49%**
κ_{χ^2}	**83.67±0.42%**	83.56±0.51%	81.60±0.46%
κ_{eHI}	83.17±0.52%	82.78±0.51%	80.79±0.71%

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

	$B, s = 4$	$B, s = 8$	$\neg B, s = 8$
κ_{HI}	83.17±1.01%	83.13±0.85%	**81.87±1.14%**
κ_{χ^2}	**83.54±1.01%**	**83.21±1.31%**	81.75±0.65%
κ_{eHI}	80.71±1.60%	80.67±1.81%	78.46±1.05%

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

	$B, s = 4$	$B, s = 8$	$\neg B, s = 8$
κ_{HI}	41.73±0.80%	40.07±0.27%	**33.55±0.26%**
κ_{χ^2}	**41.85±1.03%**	**40.22±1.01%**	33.51±0.99%
κ_{eHI}	38.97±0.93%	37.04±1.12%	31.72±0.84%

(d) 67 class indoor, 80 train, 20 test

Table 4: Results of HIK, χ^2 and exponential HIK codebooks. One-class SVM code word generation is *not* used. The best result in each column is shown in **boldface**.

Caltech 101	15 scene	8 sports	67 indoor
53.25±0.80%	78.54±0.22%	81.17±0.65%	33.48±0.59%
61.00±0.90%	82.02±0.54%	81.33±1.56%	35.09±1.04%

Table 5: Results when features are sampled in only 1 image scale and 5 scales, respectively. HIK codebooks are used, with oc_{svm}, $\neg B$ and $s = 8$.

Sampling features at 5 scales improves accuracy. It is advantageous to sample features from multiple scaled versions of the input image. Also, Table 5 reinforces the conclusions from Section 5.2.

Smaller step size is better. Similarly, a smaller step size means that more features are sampled. Table 2 shows that when other conditions were the same, $s = 4$ outperformed $s = 8$ in general. We observed differences between object and scene recognition. The accuracy difference in Caltech 101 is significant. In the sports and indoor data set, $s = 4$ slightly outperformed $s = 8$ and they are indistinguishable in the 15 class scene data set. Thus it is not necessary to compute $s = 2$ results for

Caltech 101	15 scene	8 sports	67 indoor
60.99±0.67%	79.86±0.30%	82.33±0.74%	38.04±1.24%
65.20±0.91%	84.00±0.46%	83.54±1.13%	41.75±0.94%

Table 6: Results when feature type is switched. We use B, $s = 8$, and oc_{SVM}. The second row contains numbers extracted from Table 2, and the first row are results when feature type is switched.

the two scene recognition data sets. 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. And CENTRIST has been shown as a suitable feature for place and scene recognition in Wu and Rehg (2011). As shown in Table 6, if we use SIFT for scene recognition and CENTRIST for object recognition, the recognition accuracies are reduced.

More code words are (sometimes) better. We also experimented with different numbers of code words. In the scene recognition tasks, we did not observe significant changes in recognition accuracies. In the Caltech 101 data set, 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 comparison, using standard k-means with 1000 code words (together with B, $s = 2$, and $\neg oc_{\text{SVM}}$ which is the better choice for κ_{LIN}), the accuracy is $67.89 \pm 1.11\%$. The proposed method is significantly better than standard k-means codebooks with more visual code words.

In summary, we need to choose the appropriate feature for a specific task (CENTRIST for scene recognition and SIFT for object recognition), and to 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 stabilized when step size is 8 and codebook size is 200). We strongly recommend the CENTRIST descriptor, or its variant like PACT (Wu and Rehg, 2008), and the proposed algorithms for recognizing place and scene categories.

6. Conclusion

In this article, we show that when the histogram intersection kernel is used as the similarity measure in clustering visual descriptors that are histograms, the generated visual codebooks produce better code words and as a consequence, improve the bag of visual words model. We propose a HIK based codebook generation method which runs almost as fast as k-means and has consistently higher accuracies than k-means codebooks by 2–4% in several benchmark object and scene recognition data sets. As an alternative to k-means, in which cluster centroids are used to represent code words, we proposed a one-class SVM formulation to generate better visual code words. We also generalize the proposed visual codebook generation method to arbitrary additive kernels. In particular, this extends our speedup results to the popular χ^2 kernel. The proposed algorithms achieve state-of-the-art accuracies on four benchmark object and scene recognition data sets.

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 accuracies than HIK codebooks but usually have higher accuracy than k-means codebooks. They also require less memory than HIK codebooks.

We provide a software package, named libHIK, which contains implementation of the methods proposed in this paper. The software is available at `http://c2inet.sce.ntu.edu.sg/Jianxin/projects/libHIK/libHIK.htm`.

Acknowledgments

J. Wu is supported by the NTU startup grant and the Singapore MoE AcRF Tier 1 project RG 34/09. This research was supported in part by a grant from the Google Research Awards Program. We thank the anonymous reviewers, whose comments have helped improving this paper.

References

Ankur Agarwal and Bill Triggs. Multilevel image coding with hyperfeatures. *International Journal of Computer Vision*, 78(1):15–27, 2008.

David Arthur and Sergei Vassilvitskii. **k-means++**: the advantage of careful seeding. In *18th Symposium on Discrete Algorithms*, pages 1027–1035, 2007.

Oren Boiman, Eli Shechtman, and Michal Irani. In defense of nearest-neighbor based image classification. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2008.

Sabri Boughorbel, Jean-Philippe Tarel, and Nozha Boujemaa. Generalized histogram intersection kernel for image recognition. In *Proc. Int'l Conf. on Image Processing*, 2005.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, 2005.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

Mark Everingham, Andrew Zisserman, Christopher Williams, and Luc Van Gool. The PASCAL visual object classes challenge 2006 (VOC 2006) results, 2006.

Li Fei-Fei, Rob Fergus, and Pietro 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.

Shenghua Gao, Ivor Wai-Hung Tsang, Liang-Tien Chia, and Peilin Zhao. Local features are not lonely – Laplacian sparse coding for image classification. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2010.

Chih-Wei Hsu and Chih-Jen Lin. BSVM, 2006. Software available at `http://www.csie.ntu.edu.tw/~cjlin/bsvm`.

Frédéric Jurie and Bill Triggs. Creating efficient codebooks for visual recognition. In *The IEEE Conf. on Computer Vision*, volume 1, pages 604–610, 2005.

Svetlana Lazebnik and Maxim Raginsky. Supervised learning of quantizer codebooks by information loss minimization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 31(7): 1294–1309, 2009.

Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, volume II, pages 2169–2178, 2006.

Li-Jia Li and Li Fei-Fei. What, where and who? Classifying events by scene and object recognition. In *The IEEE Conf. on Computer Vision*, 2007.

Jingen Liu and Mubarak Shah. Scene modeling using Co-Clustering. In *The IEEE Conf. on Computer Vision*, 2007.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

Subhransu Maji and Alexander C. Berg. Max-margin additive classifiers for detection. In *The IEEE Conf. on Computer Vision*, 2009.

Subhransu Maji, Alexander C. Berg, and Jitendra Malik. Classification using intersection kernel support vector machines is efficient. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2008.

Frank Moosmann, Eric Nowak, and Frederic Jurie. Randomized clustering forests for image classification. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 30(9):1632–1646, 2008.

David Nistér and Henrik Stewénius. Scalable recognition with a vocabulary tree. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, pages 2161–2168, 2006.

Francesca Odone, Annalisa Barla, and Alessandro Verri. Building kernels from binary strings for image matching. *IEEE Trans. Image Processing*, 14(2):169–180, 2005.

Florent Perronnin. Universal and adapted vocabularies for generic visual categorization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 30(7):1243–1256, 2008.

James Philbin, Ondřej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2008.

Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2009.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

Josef Sivic and Andrew Zisserman. Video Google: A text retrieval approach to object matching in videos. In *The IEEE Conf. on Computer Vision*, volume 2, pages 1470–1477, 2003.

Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

Tinne Tuytelaars and Cordelia Schmid. Vector quantizing feature space with a regular lattice. In *The IEEE Conf. on Computer Vision*, 2007.

Jan C. van Gemert, Jan-Mark Geusebroek, Cor J. Veenman, and Arnold W.M. Smeulders. Kernel codebooks for scene categorization. In *European Conf. Computer Vision*, 2008.

Andrea Vedaldi and Andrew Zisserman. Efficient additive kernels via explicit feature maps. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2010.

Julia Vogel and Bernt Schiele. Semantic modeling of natural scenes for content-based image retrieval. *International Journal of Computer Vision*, 72(2):133–157, 2007.

Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems 21*, pages 1753–1760, 2009.

John Winn, Antonio Criminisi, and Thomas Minka. Object categorization by learned universal visual dictionary. In *The IEEE Conf. on Computer Vision*, volume 2, pages 1800–1807, 2005.

Jianxin Wu. A fast dual method for HIK SVM learning. In *European Conf. Computer Vision*, LNCS 6312, pages 552–565, 2010.

Jianxin Wu and James M. Rehg. Where am I: Place instance and category recognition using spatial PACT. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

Jianxin Wu and James M. Rehg. Beyond the Euclidean distance: Creating effective visual codebooks using the histogram intersection kernel. In *The IEEE Conf. on Computer Vision*, pages 630–637, 2009.

Jianxin Wu and James M. Rehg. CENTRIST: A visual descriptor for scene categorization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 33(8):1489–1501, 2011.

Jianchao Yang, Kai Yu, Yihong Gong, and Thomas Huang. Linear spatial pyramid matching using sparse coding for image classification. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2009.

Liu Yang, Rong Jin, Rahul Sukthankar, and Frederic Jurie. Unifying discriminative visual codebook generation with classifier training for object category recognition. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2008.