Kernelized Locality-Sensitive Hashing

Brian Kulis, *Member*, *IEEE*, and Kristen Grauman, *Member*, *IEEE*

Abstract—Fast retrieval methods are critical for many large-scale and data-driven vision applications. Recent work has explored ways to embed high-dimensional features or complex distance functions into a low-dimensional Hamming space where items can be efficiently searched. However, existing methods do not apply for high-dimensional kernelized data when the underlying feature embedding for the kernel is unknown. We show how to generalize locality-sensitive hashing to accommodate arbitrary kernel functions, making it possible to preserve the algorithm's sublinear time similarity search guarantees for a wide class of useful similarity functions. Since a number of successful image-based kernels have unknown or incomputable embeddings, this is especially valuable for image retrieval tasks. We validate our technique on several data sets, and show that it enables accurate and fast performance for several vision problems, including example-based object classification, local feature matching, and content-based retrieval.

Index Terms—Similarity search, locality-sensitive hashing, central limit theorem, Kernel methods, image search.

1 Introduction

Fast indexing and search for large databases is critical to content-based image and video retrieval—particularly given the ever-increasing availability of visual data in a variety of interesting domains, such as scientific image data, community photo collections on the web, news photo collections, or surveillance archives. The most basic but essential task in image search is the "nearest neighbor (NN)" problem: to take a query image and accurately find the examples that are most similar to it within a large database. A naive solution to finding neighbors entails searching over all n database items and sorting them according to their similarity to the query, but this becomes prohibitively expensive when n is large or when the individual similarity function evaluations are expensive to compute. For vision applications, this complexity is amplified by the fact that often the most effective representations are high dimensional or structured, and the best-known distance functions can require considerable computation to compare a single pair of objects.

To make large-scale search practical, vision researchers have recently explored *approximate* similarity search techniques, where a predictable loss in accuracy is sacrificed in order to allow fast queries even for high-dimensional inputs [1], [2], [3], [4], [5]. Methods for this problem, most notably *locality-sensitive hashing* (LSH) [6], [7], offer probabilistic guarantees of retrieving items within $(1+\epsilon)$ times the optimal similarity, with query times that are sublinear with respect to n. The basic idea is to compute randomized hash functions that guarantee a high probability of collision for similar examples. In a similar spirit, a number of methods show how to form low-dimensional binary embeddings that can capture more expensive distance functions [8], [9], [10], [11]. This line of work has shown considerable promise for a variety of image search tasks such as near-duplicate retrieval, example-based object recognition, pose estimation, and feature matching.

In spite of hashing's success for visual similarity search tasks, existing techniques have some important restrictions. Current methods generally assume that the data to be hashed come from a multidimensional vector space, and require that the underlying embedding of the data be explicitly known and computable. For example, many LSH methods rely on random projections with input vectors. Spectral hashing [11], another recent hashing technique, also requires the inputs to be vectors and assumes that such vectors are drawn from a uniform distribution, an assumption that is clearly violated in practice.

This is a problematic limitation, given that many recent successful vision results employ *kernel functions* for which the underlying embedding is known only *implicitly* (i.e., only the kernel function is computable). It is thus far impossible to apply LSH and its variants to search data with a number of powerful kernels—including many kernels designed specifically for image comparisons (e.g., [12], [13], [14]), as well as some basic well-used functions like a Gaussian RBF kernel. Further, since visual representations are often most naturally encoded with structured inputs (e.g., sets, graphs, trees), the lack of fast search methods with performance guarantees for flexible kernels is detrimental.

In this paper, we present an LSH-based technique for performing fast similarity searches over *arbitrary* kernel functions. The problem is as follows: Given a kernel function $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ and a database of n objects, how can we quickly find the most similar item to a query object $\boldsymbol{q}$ in terms of the kernel function, that is, $\text{argmax}_i \kappa(\boldsymbol{q}, \boldsymbol{x}_i)$? Like standard LSH, our hash functions involve computing random projections; however, unlike standard LSH, these random projections are constructed using only the kernel function and a sparse set of examples from the database itself. Our main technical contribution is

- *B. Kulis is with the the Computer Science and Engineering Department, Ohio State University, 395 Dreese Labs, Columbus, OH 43210. E-mail: kulis@cse.ohio-state.edu.*
- *K. Grauman is with the Department of Computer Sciences, University of Texas at Austin, 1 University Station C0500, Austin, TX 78712. E-mail: grauman@cs.utexas.edu.*

Manuscript received 18 Oct. 2010; revised 23 June 2011; accepted 6 Sept. 2011; published online 1 Nov. 2011.
Recommended for acceptance by V. Pavlovic.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-2010-10-0803.
Digital Object Identifier no. 10.1109/TPAMI.2011.219.

to formulate the random projections necessary for LSH in kernel space. Our construction relies on an appropriate use of the central limit theorem (CLT) [15], which allows us to approximate a random vector using items from our database. The resulting scheme, which we call kernelized LSH (KLSH), generalizes LSH to scenarios when the feature space embeddings ($\phi(\boldsymbol{x})$, $\phi(\boldsymbol{y})$) are either unknown or incomputable.

We empirically validate our scheme with several visual search tasks. For object recognition, we present results on the Caltech-101 [16] and a data set of Flickr images, and show that our hashing scheme outperforms existing hashing methods on these data sets since it can compute hash functions over arbitrary kernels. For feature indexing with a larger database, we provide results on the Photo Tourism data set of local patches [17], [18]. We experiment with the Tiny Image data set of 80 million images [19] in order to show our technique's ability to scale to very large databases. Because our algorithm enables fast approximate search for arbitrary kernels, we can now access a much wider class of similarity functions needed for many content-based retrieval applications. We also present experiments to quantify how well our method approximates ideal random projections as well as comparisons to an existing hashing technique for embedding shift-invariant kernels via random Fourier features [20].

2 Related Work

In this section, we review related work in fast search algorithms and their application for visual search problems.

Data structures using spatial partitions and recursive hyperplane decomposition (e.g., $k-d$ trees [21]) provide an efficient means to search low-dimensional vector data exactly; however, they are known to break down in practice for high-dimensional data, and cannot provide better than a worst case linear query time guarantee. Since high-dimensional image descriptors are commonly used in object recognition, various methods to mitigate these factors have been explored, such as hierarchical feature quantization [22], decision trees [23], and priority queues [24].

Tree-based search structures that can operate with arbitrary metrics [25], [26] remove the assumption of a vector space by exploiting the triangle inequality. However, in practice, selecting useful partitioning strategies requires good heuristics, and, in spite of logarithmic query times in the expectation, metric-tree methods can also degenerate to a linear time scan of all items depending on the distribution of distances for the data set. Our results confirm this failure mode for several data sources and metrics of interest for image search.

Randomized approximate similarity search algorithms have been designed to preserve query time guarantees, even for high-dimensional inputs. Locality-sensitive hashing [6], [7] offers sublinear time search by hashing highly similar examples together in a hash table. Prior to our work, LSH functions that accommodate Hamming distance [27], inner products [7], ℓ_p norms [28], normalized partial matching [3], normalized set intersection [5], learned Mahalanobis metrics [4], and particular kernel functions such as the Gaussian kernel [20], [29] have all been developed. Vision researchers have shown the effectiveness of this class of methods for various image search applications, including shape matching, pose inference and bag-of-words indexing [1], [2], [3], [4], [5]. However, thus far arbitrary kernel functions remain off limits for LSH.

Embedding functions offer another useful way to map expensive distance functions into something more manageable computationally. Recent work has considered how to construct or learn an embedding that will preserve the desired distance function, typically with the intention of mapping to a very low-dimensional space that is more easily searchable with known techniques [30], [8], [31], [9], [10], [11]. These methods are related to LSH in the sense that both seek small "keys" that can be used to encode similar inputs, and often these keys exist in Hamming space. While most work with vector inputs, the technique in [8] accepts generic distance functions, though its boosting-based training process is fairly expensive, and search is done with a linear scan. The recent "spectral hashing" algorithm [11] requires that data be from a euclidean space and uniformly distributed, in contrast to our technique, which assumes that the data are from an arbitrary kernel space and makes no strong assumptions about the underlying distribution of the data. Another related embedding technique builds a low-dimensional embedding given data and an arbitrary kernel function [32]. Whenever the data are linearly separable by a large margin in the input kernel space, the resulting low-dimensional embedding is guaranteed to be approximately separable. As with our technique, this embedding can be applied to arbitrary kernels, but the aim of the approach is different in that the goal is to maintain separability of the embedded data rather than perform fast similarity searches.

Our main contribution is a general algorithm to draw hash functions that are locality sensitive for arbitrary normalized kernel functions, thereby permitting sublinear time approximate similarity search. This significantly widens the accessibility of LSH to generic kernel functions, whether or not their underlying feature space is known. Since our method does not require assumptions about the data distribution or input, it is directly applicable to many existing useful measures that have been studied for image search.

3 Background: Locality-Sensitive Hashing

We begin by briefly reviewing Locality-Sensitive Hashing, following [7]. Assume that our database is a set of data objects $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. Given an input query $\boldsymbol{q}$, we are interested in finding those items in the database that are most similar to the query.

The basic idea behind LSH is to project the data into a low-dimensional binary (Hamming) space; that is, each data point is mapped to a b-bit vector, called the *hash key*. If this projection is performed appropriately, we can find approximate nearest neighbors in time sublinear in n. The hash keys are constructed by applying b binary-valued hash functions $h_1, \ldots, h_b$ to each of the database objects. In order to be valid, each hash function h must satisfy the locality-sensitive hashing property:

$$\Pr[h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j)] = \mathrm{sim}(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{1}$$

where $\mathrm{sim}(\boldsymbol{x}_i, \boldsymbol{x}_j) \in [0, 1]$ is the similarity function of interest.[1] The intuition is as follows: If we can rely on only highly similar examples colliding together in the hash table (i.e., being assigned the same hash key), then at query time, directly hashing to a stored bucket will reveal the most similar examples, and only those need to be searched. Given valid LSH functions, the query time for retrieving $(1+\epsilon)$-near neighbors is bounded by $O(n^{1/(1+\epsilon)})$ for the Hamming distance [6]. One can therefore trade off the accuracy of the search with the query time required.

As an example, consider the well-known cosine similarity: $\mathrm{sim}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^T \boldsymbol{x}_j)/(\|\boldsymbol{x}_i\|_2 \|\boldsymbol{x}_j\|_2)$. In [7], Charikar proposed a hash function for this similarity function based on rounding the output of a product with a random hyperplane:

$$h_{\boldsymbol{r}}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T \boldsymbol{x} \geq 0 \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

where $\boldsymbol{r}$ is a random hyperplane from a zero-mean multivariate Gaussian $\mathcal{N}(0, I)$ of the same dimensionality as the input $\boldsymbol{x}$. The fact that this hash function obeys the locality-sensitive hash property follows from a result from Goemans and Williamson [33], who showed for such a random $\boldsymbol{r}$ that

$$\Pr\left[\mathrm{sign}(\boldsymbol{x}_i^T \boldsymbol{r}) = \mathrm{sign}(\boldsymbol{x}_j^T \boldsymbol{r})\right] = 1 - \frac{1}{\pi} \cos^{-1}\left(\frac{\boldsymbol{x}_i^T \boldsymbol{x}_j}{\|\boldsymbol{x}_i\| \|\boldsymbol{x}_j\|}\right). \tag{3}$$

Procedurally, one chooses a random vector $\boldsymbol{r}$ from $\mathcal{N}(0, I)$, then computes the sign of $\boldsymbol{r}^T \boldsymbol{x}$ for each $\boldsymbol{x}$ in the database, then repeats this over the b random vectors for a total of b hash functions. The hash table then consists of the hash keys and their pointers to data items. Given a query vector $\boldsymbol{q}$, one computes its hash key by applying the same b hash functions. A query hashes to certain buckets in the hash table, where it collides with some small portion of the stored examples. Only these examples are searched and returned in rank order as the output nearest neighbors for the query.

To perform the approximate similarity searches, we use the method in [7], which requires searching $O(n^{1/(1+\epsilon)})$ examples for the $k = 1$ approximate nearest neighbor. Given the list of n database hash keys, $M = 2n^{1/(1+\epsilon)}$ random permutations of the bits are formed, and each list of permuted hash keys is sorted lexicographically to form M sorted orders. A query hash key indexes into each sorted order with a binary search, and the $2M$ nearest examples found are the approximate nearest neighbors. Additionally, we introduce a parameter B that is the number of neighboring bins to consider as potential nearest neighbors when searching through the sorted permutations (unless stated otherwise, this is set to 0 by default). See [7] for more details.

Previously, hash functions have been designed for cases where the similarity function "sim" refers to an ℓ_p norm, Mahalanobis metric, or inner product [28], [4], [7]. In this work, the similarity function of interest is an arbitrary (normalized) kernel function:

$$\begin{aligned} \mathrm{sim}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) \Big/ \left(\sqrt{\kappa(\boldsymbol{x}_i, \boldsymbol{x}_i)\kappa(\boldsymbol{x}_j, \boldsymbol{x}_j)}\right) \\ &= (\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j))/(\|\phi(\boldsymbol{x}_i)\|_2 \|\phi(\boldsymbol{x}_j)\|_2), \end{aligned}$$

for some (possibly unknown) embedding function $\phi(\cdot)$. In the next section, we will present our algorithm for drawing hash functions that will satisfy (1) for any kernel.

Finally, we note that random projections arise in formulations of LSH beyond the approach of Charikar for cosine similarity. For instance, random projections may be used for approximating the ℓ_2 norm, as shown by [28], so our approach should easily carry over to this scenario. Beyond LSH, there are several other dimensionality reduction techniques based on the Johnson-Lindenstrauss lemma that involve computing random projections, and our results may additionally be of interest to those applications.

4 Kernelized Locality-Sensitive Hashing

The random hyperplane hashing method proposed by Charikar assumes that the vectors are represented explicitly, so that the sign of $\boldsymbol{r}^T \boldsymbol{x}$ can easily be computed.[2] We now consider the case when the data are kernelized. We denote the inputs as $\phi(\boldsymbol{x})$ and assume that the underlying embeddings may be unknown or very expensive to compute. Since we have access to the data only through the kernel function $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$, it is not clear how to compute the hash functions. This is because to use the hash function in (2) we need to reference a random hyperplane *in the kernel-induced feature space*. For example, the RBF kernel has an infinite-dimensional embedding, making it seemingly impossible to construct $\boldsymbol{r}$. Thus, the key challenge in applying LSH to this scenario is in constructing a vector $\boldsymbol{r}$ from $\mathcal{N}(0, I)$ such that $\boldsymbol{r}^T \phi(\boldsymbol{x})$ can be computed via the kernel function.

The main idea of our approach is to construct $\boldsymbol{r}$ as a weighted sum of a subset of the database items. An appropriate construction will allow the random hyperplane hash function to be computed purely via kernel function evaluations, but will also ensure that $\boldsymbol{r}$ is approximately Gaussian. Consider each data point $\phi(\boldsymbol{x}_i)$ from the database as a vector from some underlying distribution $\mathcal{D}$ with mean μ and covariance Σ, which are generally unknown. Given a natural number t, we define

$$\boldsymbol{z}_t = \frac{1}{t} \sum_{i \in S} \phi(\boldsymbol{x}_i), \tag{4}$$

where S is a set of t database objects chosen i.i.d. from $\mathcal{D}$. The central limit theorem [15] tells us that, for sufficiently large t, the random vector

$$\tilde{\boldsymbol{z}}_t = \sqrt{t}(\boldsymbol{z}_t - \mu) \tag{5}$$

is distributed according to the multivariate Gaussian $\mathcal{N}(0, \Sigma)$. By applying a whitening transform, the vector $\Sigma^{-1/2} \tilde{\boldsymbol{z}}_t$ will be distributed according to $\mathcal{N}(0, I)$, precisely the distribution required in (2).

Therefore, we denote our random vector as $\boldsymbol{r} = \Sigma^{-1/2} \tilde{\boldsymbol{z}}_t$, and the desired hash function $h(\phi(\boldsymbol{x}))$ is given by

1. LSH has been formulated in two related contexts—one in which the likelihood of collision is guaranteed relative to a threshold on the radius surrounding a query point [27], and another where collision probabilities are equated with a similarity score [7]. We use the latter definition here.

2. Note that other LSH functions exist, but all involve an explicit representation of the input, and are not amenable to the general kernelized case.

$$h(\phi(\boldsymbol{x})) = \begin{cases} 1, & \text{if } \phi(\boldsymbol{x})^T \Sigma^{-1/2} \tilde{\boldsymbol{z}}_t \geq 0 \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

Both the covariance matrix Σ and the mean μ of the data are unknown, and so they must be approximated via a sample of the data. We choose a set of p database objects, which we denote without loss of generality as the first p items $\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_p)$ of the database (we will discuss the choice of the value of p later). Now, we may (implicitly) estimate the mean as $\mu = \frac{1}{p}\sum_{i=1}^{p} \phi(\boldsymbol{x}_i)$; similarly, we can consider the covariance matrix Σ over the p samples, though it too cannot be stored explicitly.

Define a kernel matrix over the p sampled points, denote it as K. For the derivation below, we assume that the data points used to generate the kernel matrix are zero centered; this may be achieved implicitly (in kernel space) via the transformation

$$K \leftarrow K - \frac{1}{p} K \boldsymbol{e}\boldsymbol{e}^T - \frac{1}{p} \boldsymbol{e}\boldsymbol{e}^T + \frac{\boldsymbol{e}^T K \boldsymbol{e}}{p^2} \boldsymbol{e}\boldsymbol{e}^T.$$

If Φ is the matrix of the p database objects, we can obtain the above expression by forming a kernel matrix over the data after transforming the data by subtracting the mean: $\Phi \leftarrow \Phi - \frac{1}{p}\Phi \boldsymbol{e}\boldsymbol{e}^T$.

In order to compute $h(\phi(\boldsymbol{x}))$, we will use a technique similar to that used in kernel Principal Component Analysis (kPCA) [34] to project onto the eigenvectors of the covariance matrix. Let the eigendecomposition of the kernel matrix defined above be $K = U\Theta U^T$. If the eigendecomposition of Σ is $V\Lambda V^T$, then $\Sigma^{-1/2} = V\Lambda^{-1/2}V^T$. Therefore, we can rewrite the hash function as follows:

$$h(\phi(\boldsymbol{x})) = \text{sign}(\phi(\boldsymbol{x})^T V \Lambda^{-1/2} V^T \tilde{\boldsymbol{z}}_t).$$

Note that the nonzero eigenvalues of Λ are equal to the nonzero eigenvalues of Θ. Further, denote the kth eigenvector of the covariance matrix as $\boldsymbol{v}_k$ and the kth eigenvector of the kernel matrix as $\boldsymbol{u}_k$. According to the derivation of kernel PCA, when the data is zero centered, we can compute the projection

$$\boldsymbol{v}_k^T \phi(\boldsymbol{x}) = \sum_{i=1}^{p} \frac{1}{\sqrt{\theta_k}} \boldsymbol{u}_k(i) \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}), \quad (7)$$

where the $\phi(\boldsymbol{x}_i)$ are the sampled p data points.[3]

We complete the computation of $h(\phi(\boldsymbol{x}))$ by performing this computation over all k eigenvectors, resulting in the following expression:

$$\phi(\boldsymbol{x})^T V \Lambda^{-1/2} V^T \tilde{\boldsymbol{z}}_t = \sum_{k=1}^{p} \frac{1}{\sqrt{\theta_k}} \boldsymbol{v}_k^T \phi(\boldsymbol{x}) \boldsymbol{v}_k^T \tilde{\boldsymbol{z}}_t.$$

We substitute (7) for each of the eigenvector inner products and expand the resulting expression: $\phi(\boldsymbol{x})^T V \Lambda^{-1/2} V^T \tilde{\boldsymbol{z}}_t =$

$$\sum_{k=1}^{p} \frac{1}{\sqrt{\theta_k}} \left(\sum_{i=1}^{p} \frac{1}{\sqrt{\theta_k}} \boldsymbol{u}_k(i) \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}) \right) \left(\sum_{i=1}^{p} \frac{1}{\sqrt{\theta_k}} \boldsymbol{u}_k(i) \phi(\boldsymbol{x}_i)^T \tilde{\boldsymbol{z}}_t \right).$$

Now, we reorder the summations and reorganize terms

$$= \sum_{k=1}^{p} \sum_{i=1}^{p} \sum_{j=1}^{p} \frac{1}{\theta_k^{3/2}} \boldsymbol{u}_k(i) \boldsymbol{u}_k(j) \big(\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x})\big) \big(\phi(\boldsymbol{x}_j)^T \tilde{\boldsymbol{z}}_t\big)$$

$$= \sum_{i,j=1}^{p} \big(\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x})\big) \big(\phi(\boldsymbol{x}_j)^T \tilde{\boldsymbol{z}}_t\big) \left(\sum_{k=1}^{p} \frac{1}{\theta_k^{3/2}} \boldsymbol{u}_k(i) \boldsymbol{u}_k(j) \right).$$

Finally, we use the fact that $K_{ij}^{-3/2} = \sum_{k=1}^{p} \frac{1}{\theta_k^{3/2}} \boldsymbol{u}_k(i) \boldsymbol{u}_k(j)$ and simplify further to obtain

$$= \sum_{i=1}^{p} \sum_{j=1}^{p} K_{ij}^{-3/2} \big(\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x})\big) \big(\phi(\boldsymbol{x}_j)^T \tilde{\boldsymbol{z}}_t\big)$$

$$= \sum_{i=1}^{p} \boldsymbol{w}(i) \big(\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x})\big),$$

where $\boldsymbol{w}(i) = \sum_{j=1}^{p} K_{ij}^{-3/2} \phi(\boldsymbol{x}_j)^T \tilde{\boldsymbol{z}}_t$.[4]

This means that the Gaussian random vector can be expressed as $\boldsymbol{r} = \sum_{i=1}^{p} \boldsymbol{w}(i) \phi(\boldsymbol{x}_i)$—that is, as a weighted sum over the feature vectors chosen from the set of p sampled database items. We now expand $\tilde{\boldsymbol{z}}_t$. Recall that $\tilde{\boldsymbol{z}}_t = \sqrt{t}(\frac{1}{t}\sum_{i \in S} \phi(\boldsymbol{x}_i) - \mu) = \frac{1}{\sqrt{t}} \sum_{i \in S} \phi(\boldsymbol{x}_i)$. Substituting this into $\boldsymbol{w}(i)$ yields

$$\boldsymbol{w}(i) = \frac{1}{\sqrt{t}} \sum_{j=1}^{p} \sum_{\ell \in S} K_{ij}^{-3/2} K_{j\ell}.$$

We assume that the t points selected for S are a subset of the p sampled points (i.e., the t points are sampled from the set of points used to sample the mean and covariance); this makes computation simpler since we have implicitly centered the data. In that case, if $\boldsymbol{e}$ is a vector of all ones and $\boldsymbol{e}_S$ is a vector with ones in the entries corresponding to the indices of S, then the computation of $\boldsymbol{w}$ simplifies as follows:

$$\boldsymbol{w} = K^{-1/2} \boldsymbol{e}_S. \quad (8)$$

Note that we are ignoring the factor of t since it does not affect the sign of the hash function.

To compute the hash function, we therefore use

$$h(\phi(\boldsymbol{x})) = \text{sign}\left(\sum_{i=1}^{p} \boldsymbol{w}(i) \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}) \right). \quad (9)$$

One subtle issue remains: We assumed that the $\phi(\boldsymbol{x}_i)$ vectors had been implicitly zero centered. That is, each of the sampled $\phi(\boldsymbol{x}_i)$ is replaced by $\phi(\boldsymbol{x}_i) - \frac{1}{p}\sum_{j=1}^{p} \phi(\boldsymbol{x}_j)$. Therefore, the hash function should be written in terms of the original data points as

$$h(\phi(\boldsymbol{x})) = \text{sign}\left(\sum_{i=1}^{p} \boldsymbol{w}(i) \left(\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}) - \frac{1}{p} \sum_{j=1}^{p} \phi(\boldsymbol{x}_j)^T \phi(\boldsymbol{x}) \right) \right).$$

However, the second part of the sum simplifies to zero since $\sum_{i=1}^{p} \boldsymbol{w}(i) = 0$, and so the final hash function can indeed be written as given above in (9).

3. For centered data, the covariance is denoted as $\Sigma = \frac{1}{p}\sum_{i=1}^{p} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T$. To simplify the discussion, we are ignoring the $\frac{1}{p}$ term in our analysis, which does not affect the resulting hash function computation.

4. We can alternatively reach this conclusion by noting that $\phi(\boldsymbol{x})^T \Sigma^{-1/2} \tilde{\boldsymbol{z}}_t = \phi(\boldsymbol{x})^T \Phi K^{-3/2} \Phi^T \tilde{\boldsymbol{z}}_t$ when the data are centered, where $K = \Phi^T \Phi$. This follows by computing the singular value decomposition of Φ and simplifying.

Putting everything together, the resulting method is surprisingly simple. To summarize our kernelized locality-sensitive hashing algorithm:

- Select p data points and form a kernel matrix K over this data.
- Center the kernel matrix.
- Form the hash table over database items: For each hash function $h(\phi(\boldsymbol{x}))$, form $\boldsymbol{e}_S$ by selecting t indices at random from $[1, \ldots, p]$, then form $\boldsymbol{w} = K^{-1/2}\boldsymbol{e}_S$, and assign bits according to $h(\phi(\boldsymbol{x})) = \text{sign}(\sum_i \boldsymbol{w}(i)\kappa(\boldsymbol{x}, \boldsymbol{x}_i))$.
- For each query, form its hash key using these hash functions and employ existing LSH methods to find the approximate nearest neighbors.

Computationally, the most expensive step is in the single offline computation of $K^{-1/2}$, which takes time $O(p^3)$. Once this matrix has been computed, each individual hash function requires $O(p^2)$ kernel function evaluations to compute its corresponding $\boldsymbol{w}$ vector (also done offline). Once $\boldsymbol{w}$ has been computed for a given hash function, the computation of the hash function can be computed with p evaluations of the kernel function. In order to maintain efficiency and to maintain sublinear time searches, we want p to be much smaller than n—for example, $p = O(\sqrt{n})$ would guarantee that the algorithm maintains sublinear search times.

5 Discussion

Some additional care must be taken to verify that the analysis for KLSH holds when the underlying embeddings are infinite dimensional (for example, with the Gaussian kernel), but in fact the general case does hold. To summarize the main details associated with arbitrary reproducing kernel Hilbert spaces (RKHSs): First, the central limit theorem holds in general Banach spaces (for which RKHSs are a special case) under certain conditions—see [35] for a discussion. Second, in the infinite-dimensional case, we whiten the data via the covariance operator; this has been studied for the kernel ICA problem [36]. Finally, projecting onto eigenvectors of the covariance is performed as in kernel PCA, which holds for infinite-dimensional embeddings. We stress that the KLSH algorithm is unchanged for such embeddings.

Another potential concern is in the case when Σ is not full rank, that is, when the underlying data distribution lies in a subspace of the full kernel space. In this case, the resulting approximately random vector $\boldsymbol{r}$ will lie in a subspace of the full kernel space. However, this is not problematic since, for a true random vector $\boldsymbol{r}$, we can decompose it into a component lying in the subspace and a component orthogonal to the subspace of the data. The projection of the orthogonal component will always equal 0, and so the fact that our random projections only lie on the subspace of the data does not cause any difficulties. Since the algorithm does not compute $\Sigma^{-1/2}$ explicitly, we will not require any further change to the algorithm (as long as the kernel matrix is full rank).

Additionally, the random vector $\boldsymbol{r}$ constructed during the KLSH routine is only *approximately* distributed according to $\mathcal{N}(0, I)$—the central limit theorem assumes that the mean and covariance of the data are known exactly, whereas we employ an approximation using a sample of p points. Furthermore, it is possible that the resulting $\boldsymbol{r}$ vectors do not approximate a Gaussian well unless p and t are extremely large. This may be the case if the underlying embeddings of the kernel function are very high dimensional. As a result, it may be difficult or even impossible to formally prove that such hash functions will produce truly random projections in general when p and t are sublinear in n; we leave such analysis as an open problem for future work. The good news is that empirical evidence suggests that we do not need very many samples to compute a satisfactory random vector for kernelized hashing; as we will see in the experimental results, with $p = 300$ and $t = 30$ we obtain good hashing results with several different kernels and over very large data sets.

We would also like to stress the general applicability and simplicity of our approach. Even if the underlying embedding for a particular kernel function is known, our technique may still be desirable due to its relative simplicity. For instance, kernels such as the pyramid match kernel [37] or the proximity distribution kernel [38] have known sparse, high-dimensional embeddings [3], [39] for which standard LSH methods can be applied; however, in such scenarios the computation of the hash functions is dependent on the kernel embeddings, requiring separate (and sometimes intricate) hashing implementations for each particular kernel function. In contrast, our approach is general and only requires knowledge of the kernel function. As a result, the KLSH scheme may be preferable even in these cases.

6 Experimental Results

To empirically validate the effectiveness of the proposed hashing scheme, we provide results on several data sets. Our primary goal is to verify that the proposed KLSH method can accommodate kernels with incomputable feature embeddings, and use them to perform searches that are fast but still reliable relative to a linear scan. We demonstrate the algorithm's flexibility with respect to kernel choice by including a number of useful kernel functions, including the χ^2-kernel, Gaussian RBF, and several learned kernels.[5]

To summarize our empirical findings: We will show that our method can be applied effectively on a number of data sets and using a number of common image kernels, with results that are comparable to a linear scan (Sections 6.2-6.4). We will also demonstrate that the method can be applied on a database of 80 million images, demonstrating the potential of the method to be applied in large-scale scenarios (Section 6.5). Further, we will quantify how well our method approximates true random vectors using a common normality test (Section 6.6). Finally, we will compare our method to the random Fourier features method for hashing developed in [20] for the class of shift-invariant kernels, and show that our approach is favorable on several data sets (Section 6.7).

6.1 Evaluation Criteria and Computational Considerations

Throughout, we present results showing the percentage of database items searched with hashing as opposed to the

5. Our KLSH code is available at http://vision.cs.utexas.edu/projects/klsh.

linear scan baseline; this format is more direct than reporting raw runtime results, which are dependent on the particular optimizations of the code. In terms of additional overhead, finding the approximate nearest neighbors given the query hash key is very fast, particularly if the computation can be distributed across several machines (since the random permutations of the hash bits are independent of one another). The main computational cost is in taking the list of examples that collided and sorting them by their similarity to the query; this running time is primarily controlled by ϵ. Specifically, since we use Charikar's search procedure for the hashing stage [7], at query time we need to perform a binary search on each of $M = 2n^{1/(1+\epsilon)}$ sorted orders. Some specific timing and overhead results are discussed for the Tiny Image data set, where we see that searching the approximate nearest neighbors is indeed the most time-consuming step of the procedure. This holds in general in our experience.

In terms of memory usage, the value of ϵ directly determines the number of hash permutations required. For large data sets, such as Tiny Images, a large number of hash permutations can result in substantial memory overhead for storage of the hash tables. To counter this, there are various techniques that have been explored for reducing the memory overhead. One is to use a linear scan in the Hamming space and thus only store a single hash permutation. This can be implemented quickly since the Hamming distance between two binary vectors is simply an XOR followed by a bit count; for example, on the Tiny Image data, nearest neighbors can be retrieved in just a few seconds. Another option is to reduce the number of hash permutations stored. We explore one such option for the Tiny Images where we use a larger ϵ and then increase the B parameter, which we discuss in further detail below.

6.2 Example-Based Object Recognition

Our first experiment uses the Caltech 101 data set, a standard benchmark for object recognition. We use our technique to perform nearest neighbor classification, categorizing each novel image according to which of the 101 categories it belongs. This data set is fairly small (~ 9 K total images); we use it because recent impressive results for this data have applied specialized image kernels, including some with no known embedding function. The goal is therefore to show that our hashing scheme is useful in a domain where such kernel functions are typically employed, and that nearest neighbor accuracy does not significantly degrade with the use of hashing. We also use this data set to examine how changes in the parameters affect accuracy.

We employ the correspondence-based local feature kernel (CORR) designed in [12], and learn a kernel on top of it using the metric learning algorithm given in [40]. The learned kernel basically specializes the original kernel to be more accurate for the classification task, and was shown in [4] to provide the best reported single kernel results on the Caltech-101 (when using linear scan search). We train the metric with 15 images per class. To compute accuracy, we use a simple k-nearest neighbor classifier ($k = 1$) using both a linear scan baseline and KLSH.

Fig. 1 compares our accuracy using hashing versus that of an exhaustive linear scan, for varying parameter settings. The parameters of interest are the number of bits used for the hash keys b, the value of ϵ (from standard LSH), and the values of t and p (from our KLSH algorithm). When varying one of the parameters, the others remain fixed at the following: $b = 300$, $\epsilon = 0.5$, $t = 30$, and $p = 300$. Since the algorithm is randomized, we ran KLSH 10 times and averaged the results over the 10 runs. The results of changing ϵ and the number of hash bits are consistent with the behavior seen for standard LSH (see, for example, [3], [4]). From the plots, it appears that the KLSH parameters (t, p), are reasonably robust (see Figs. 1c and 1d).

The parameter ϵ trades off accuracy for speed, and thus has a more significant impact on final classifier performance (see Fig. 1a). Our best result of 59 percent hashing accuracy, with $\epsilon = 0.2$, is significantly better than the best previous hashing-based result on this data set: 48 percent with the pyramid match, as reported in [4]. Note that hashing with the pyramid match was possible in the past only because that kernel has a known explicit embedding function [3]; with our method, we can hash with matching kernels for which the embedding function is unknown (e.g., CORR). This 11-point accuracy improvement illustrates the importance of being able to choose the best suited kernel function for the task—which KLSH now makes possible.

In terms of the speed, the percentage of database items searched was uniform as t and p changed. On average, KLSH searched 17.4 percent of the database for $\epsilon = 0.2$; 6.7 percent for $\epsilon = 0.5$, and 1.2 percent for $\epsilon = 1.5$. As we will see in subsequent sections, even lower percentages of the database are searched once we move on to much larger data sets.

As a baseline, we also ran this experiment using the metric-tree (M-tree) approach developed in [26], using the implementation provided online by the authors. This is a well-known exact search algorithm that accepts arbitrary metrics as input. To map the CORR kernel values $\kappa(x, y)$ to distance values, we compute: $D(\boldsymbol{x}, \boldsymbol{y}) = (\kappa(\boldsymbol{x}, \boldsymbol{x}) + \kappa(\boldsymbol{y}, \boldsymbol{y}) - 2\kappa(\boldsymbol{x}, \boldsymbol{y}))^{\frac{1}{2}}$.

While accuracy with this method is consistent with a linear scan, its search time was quite poor; of the n database items, the M-tree required searching $n \pm 30$ examples to return the first NN. The speed was unchanged when we varied the splitting function (between the generalized hyperplane and balanced strategies), the promotion method used to promote objects in the parent role (random, maximum upper bound on distances, or minimum maximum radius policies), or the minimum node utilization parameter (which we tested for values from 0 to 0.5 in increments of 0.1). The likely problem is that the distribution of distances between the indexed objects has relatively low variance, making the tree-based measure less effective. Thus, even though the M-tree can operate with arbitrary metrics, KLSH has a clear performance advantage for this data.

6.3 Indexing Local Patches for Efficient Correspondences

Our second experiment uses the patch data set [18] associated with the Photo Tourism project [17]. It consists of local image patches of Flickr photos of various landmarks. The goal is to compute correspondences between local

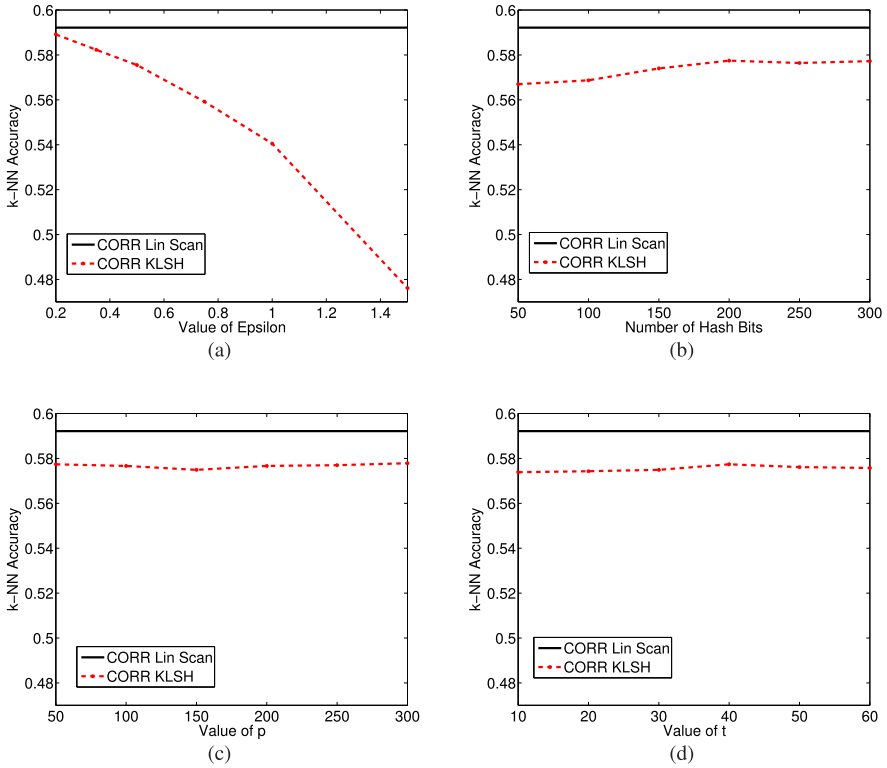

Fig. 1. Results on the Caltech-101 data set. The plots illustrate the effect of parameter settings on classification accuracy when using KLSH with a learned CORR kernel. When varying a parameter, the others are fixed at $\epsilon = 0.5$, $t = 30$, $p = 300$, and $b = 300$. The value of ϵ controls the accuracy versus speed tradeoff, and most influences results; the more examples we are willing to search, the more closely we can approximate the linear scan. When searching only 6.7 percent of the data ($\epsilon = 0.5$), accuracy is nearly 58 percent, versus 59 percent with a linear scan. Overall, accuracy is quite stable with respect to the number of bits and our algorithm's p and t parameters.

features across multiple images, which can then be provided to a structure-from-motion algorithm to generate 3D reconstructions of the photographed landmark [17]. Thus, one critical subtask is to take an input patch and retrieve its corresponding patches within any other images in the database—another large-scale similarity search problem.

We use the $n = 100{,}000$ image patches provided for the Notre Dame Cathedral. Since the goal is to extract ideally *all* relevant patches, we measure accuracy in terms of the recall rate. We consider two measures of similarity between the patches' SIFT descriptors: euclidean distance (L_2), and a Gaussian RBF kernel computed with a metric learned on top of L_2 (again using [40] to learn the parameters). For the first measure, standard LSH can be applied. For the second, only KLSH is applicable since the underlying embedding of the data is not easily computable (in fact, its dimension is technically infinite).

Fig. 2 shows the results. The two lower curves (nearly overlapping) show the recall results when using either a linear scan or LSH with the SIFT vectors and euclidean

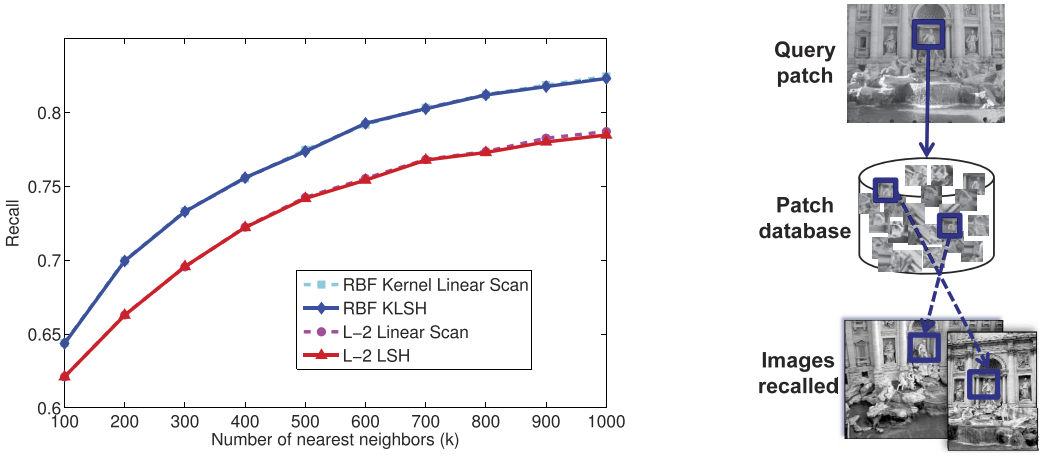

Fig. 2. Results on the Photo Tourism data set. The lower two curves show LSH or linear scan applied for L_2 search on SIFT descriptors; the upper two curves show KLSH or linear scan applied for a Gaussian RBF kernel after metric learning is applied to the SIFT vectors. For both, recall rates using hashing are nearly equivalent to those using linear scan search. However, now that our KLSH algorithm enables hashing with the RBF kernel, more accurate retrieval is possible in sublinear time.

distance. The two higher curves show recall using a Gaussian RBF kernel on top of the learned SIFT features. In both cases, the recall rate relative to a linear scan is hardly affected by the hashing approximation. However, with KLSH, a stronger kernel can be used with the hashing (the RBF), which improves the accuracy of the results. For this data set, KLSH requires searching only 0.26 percent of the database on average, a significant improvement.

Again, the M-tree baseline offered no better performance than a linear scan on this data set; as earlier, we suspect this is because of the high dimensionality of the features, and the low variance in the interfeature distances.

6.4 Example-Based Scene Recognition

To further demonstrate the applicability and flexibility of KLSH, we next consider hashing with the χ^2-kernel. The χ^2-kernel is defined as a generalized Gaussian kernel parameterized by the χ^2 distance between two histograms h_i and h_j:

$$K_{\chi^2}(h_i, h_j) = \exp\left(-\frac{1}{\gamma}\sum_{k=1}^{H}\frac{(h_i(k) - h_j(k))^2}{h_i(k) + h_j(k)}\right), \qquad (10)$$

where H denotes the number of histogram bins and γ is a scaling parameter. This kernel is a favored similarity measure for object recognition and image and video retrieval. In particular, it is repeatedly found to be very effective for the popular bag-of-visual-words representation, which is used to encode the distribution of local spatial or spatiotemporal appearance patterns (e.g., [41], [42], [43]). As with the experiments above, to our knowledge no previous hash functions can support this specialized kernel with guaranteed sublinear time retrieval.

To test this kernel with bag-of-words descriptors, we perform experiments with a data set previously studied in [39]. It contains 5,400 images of 18 different tourist attractions from the photo-sharing site Flickr. The data set was composed by taking three cities in Europe that have major tourist attractions: Rome, London, and Paris. The tourist sites for each city were taken from the top attractions in www.TripAdvisor.com under the headings Religious site, Architectural building, Historic site, Opera, Museum, and Theater.[6] We downloaded the first 300 images returned from each search query to represent the data for each class. We manually fixed the ground truth labels.

All images were scaled down to have moderate width (320 pixels). To extract local image features at interest points, we detect corner and blob-like regions using the Harris-affine [44] and Maximally Stable Extremal Regions (MSER) [45] detectors, and represent all regions with SIFT descriptors [46]. Following standard procedures, we use k-means to form the visual word codebook, and set the number of visual words to be $k = 200$.

We again pose a nearest neighbor classification task, and compare results of a linear scan with the KLSH algorithm. We randomly split the data into database and test queries via a 50/50 split. As with the Caltech experiment, we choose $p = 300$, $t = 30$, and 300 hash bits, and vary ϵ. We compared a k-nearest neighbor classifier for the test queries ($k = 5$) using a linear scan with the kernelized hashing method.

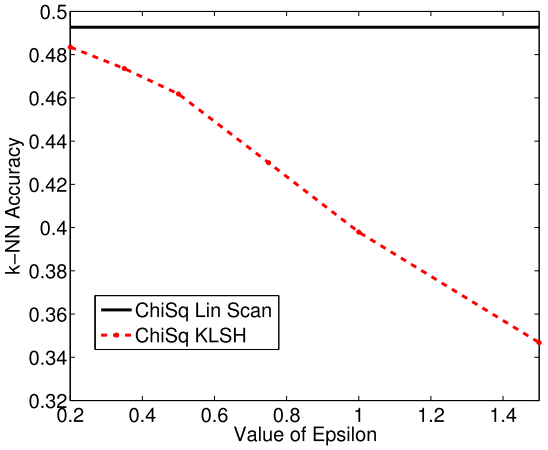

Fig. 3. Results on the Flickr data set. The plot shows k-nearest neighbor accuracy of a linear scan and the KLSH algorithm as a function of ϵ. Here, $p = 300$, $t = 30$, and $b = 300$ hash bits are used. We see that the accuracy of hashing approaches the accuracy of a linear scan as ϵ decreases. Note that chance accuracy on this 18-way classification problem would be only 0.05.

Fig. 3 shows the results. As expected, the accuracy of the k-nearest neighbor classifier approaches the results of the linear scan as ϵ goes to 0. These results share the same characteristics as the results over the Caltech data set, and demonstrate another valuable kernel that was previously inaccessible with hashing algorithms. Fig. 4 shows some example retrievals to qualitatively compare KLSH's results to those of a linear scan; we see that the results of KLSH typically match well with the results of the linear scan.

6.5 Large-Scale Image Search with Tiny Images

Next, we provide results using a very large-scale data set of 80 million images, provided by the authors of [19]. Here, the task is content-based image retrieval. The images are "tiny": 32×32 pixels each. Following [11], we use a global Gist descriptor for each image, which is a 384-dimensional vector describing the texture within localized grid cells. We apply KLSH to a Gaussian RBF kernel over the Gist features. This experiment clearly demonstrates the practical scalability of our hashing approach for very large image databases.

Given that the images were collected with keyword-based web crawlers, the data set does not have definitive ground truth to categorize the images. However, we can use the data to qualitatively show the kinds of images that are retrieved, to quantitatively show how well KLSH approximates a linear scan, and to confirm that our algorithm is amenable to rapidly searching very large image collections.

Fig. 5 shows example image retrieval results. As in the Flickr image figure above, the leftmost image in each set is the query, the top row shows the linear scan results, and the row below it shows our KLSH results. Ideally, linear scan and KLSH would return the same images, and indeed they are often close and overlap. The Gist descriptors appear to perform best on images with clearly discernable shape structure—sometimes the descriptors are insufficient, and so both the linear scan and hashing results do not appear to match the query.

Fig. 6 quantifies the relative accuracy for the images retrieved using our KLSH keys with a linear scan with the

6. Overall, the list yielded 18 classes: Arc de Triomphe, Basilica San Pietro, Castel SantAngelo, Colosseum, Eiffel Tower, Globe Theatre, Hotel des Invalides, House of Parliament, Louvre, Notre Dame Cathedral, Pantheon, Piazza Campidoglio, Roman Forum, Santa Maria Maggiore, Spanish Steps, St. Paul's Cathedral, Tower Bridge, and Westminister Abbey.

Fig. 4. Example queries and retrieved neighbors for the Flickr data set. Each column shows eight queries; the leftmost image in each group is the query image and to its immediate right are the five nearest examples according to a linear scan with the χ^2 kernel (the ideal result). Just under each of those are the neighbors retrieved with KLSH. Note the similarity between those found exhaustively and those found with our algorithm, and the strength of the χ^2-kernel for this scene matching task.

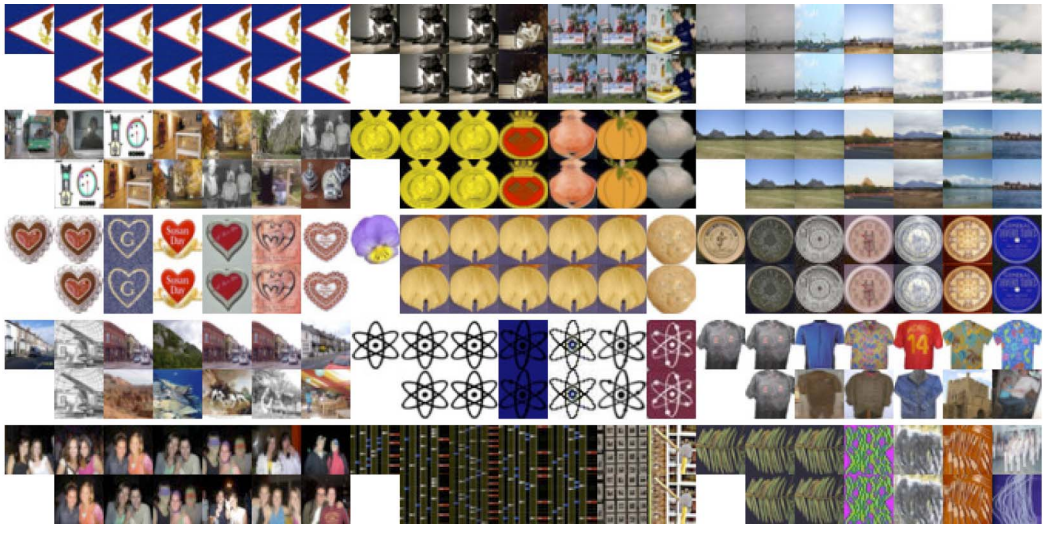

Fig. 5. Example results for the Tiny Image data set. For each example, the query appears as the leftmost image. The top row corresponds to results of a linear scan with the Gist vectors; the second row corresponds to the KLSH results. Our method often retrieves neighbors very similar to those of the linear scan, but does so by searching only 0.98 percent of the 80 million images.

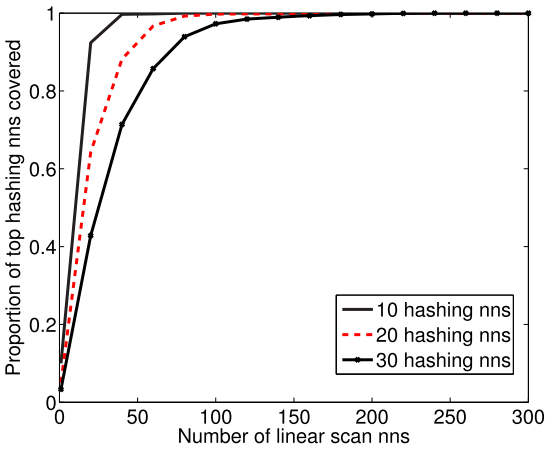

Fig. 6. Results on the Tiny Images data set. The plot shows how many linear scan neighbors are needed to cover the first 10, 20, or 30 hashing neighbors. The ideal curve would reach the top left corner of the plot. The 10-hashing nns curve shows, for example, that 100 percent of the neighbors in KLSH's top 10 are within the top 50 returned with an exhaustive linear scan.

Hamming distance versus an exhaustive linear scan in the original Gist feature space. The curves show how many of the top-ranked exhaustive linear scan neighbors must be included before the top 10, 20, or 30 KLSH results are all accounted for. For example, they show that for the top 10 KLSH neighbors, 100 percent will be within the top 50 linear scan neighbors on average. Similarly, for the top 30 KLSH neighbors, 80 percent will be within the top 50 linear scan neighbors on average.

We have also evaluated the Tiny Images search using Charikar's LSH hashing algorithm with the KLSH keys for this large-scale database, as reported in [47]. We use only 130 hash key permutations (which corresponds to $\epsilon \approx 2.74$) in order to restrict the memory overhead when storing the sorted orders. To counter the accuracy loss that a higher value of ϵ may entail, we increase the B parameter to 100 so as to search nearby hash buckets and thereby explore more hashing matches. We use $b = 300$ bits per image and randomly select 100 images as queries. The KLSH parameters are fixed at $t = 30$ and $p = 300$, as before.

In this setting, KLSH searches only 0.98 percent of the entire database on average in order to produce the top 10 approximate nearest neighbors per query. (We can easily decrease this percentage even further by using a smaller value of B; our choice may have been too liberal.) With hashing, the average running times for a NN query with our Matlab code fully optimized are: 0.001 s for hash key construction and permutation (overhead), 0.13 s for binary search to find approximate NNs (overhead), and 0.44 s for searching the approximate NNs. In contrast, a linear scan takes 45 seconds per query on average, assuming that all images are stored in memory.

6.6 Normality Test for the Kernelized Hyperplanes

The proposed KLSH algorithm relies on generating Gaussian-distributed hyperplanes in the kernel-induced feature space. How Gaussian are the hyperplanes that the algorithm implicitly generates? Due to the central limit theorem, we can expect that the larger the number of sampled points used to generate the hyperplanes (i.e., the larger p and t are), the more closely the functions will resemble samples from the desired distribution. While by definition we cannot empirically test the normality of the hyperplanes for an arbitrary kernel, we can examine their normality for the special case of a linear kernel for which we do know the kernel-induced feature space mapping (the identity).

Thus, in this experiment, we evaluate the *Gaussianity* of KLSH hyperplanes computed using our algorithm's learned weights on selected training points. We use a data set of 90,000 128-dimensional SIFT descriptors taken from the Notre Dame images in the Photo Tourism image patch collection. For increasing values of p, we use KLSH to generate the weights on selected points among the p training instances, and then explicitly compute the KLSH hyperplane dictated by those learned weights: $\boldsymbol{r} = \sum_{i=1}^{p} \boldsymbol{w}(i)\phi(\boldsymbol{x}_i)$, which is simply $\boldsymbol{r} = \sum_{i=1}^{p} \boldsymbol{w}(i)\boldsymbol{x}_i$ for the linear kernel. We do this for 1,000 trials for each value of p, each time selecting a random subset of p indices from the data, and setting $t = \lfloor 0.2 \times p \rfloor$.

To score the degree of normality, we use the Anderson-Darling statistic [48]. The Anderson-Darling method is a statistical test for the hypothesis that n independent, identically distributed random variables have a specified continuous distribution, and is often used to test whether a sample of data departs from normality. Lower values of the statistic indicate greater evidence for a Gaussian fit, and the test is declared successful (Gaussian hypothesis confirmed) if the associated probability exceeds the critical value for a given confidence level α.

For reference, we compare the Gaussianity results of KLSH to two methods: 1) hyperplanes sampled directly from the desired $\mathcal{N}(0, I)$ 128-dimensional distribution, and 2) hyperplanes computed by taking a randomly weighted combination of points from the same training pool, with weights sampled from $\mathcal{N}(0, 1)$. The former serves as the "ideal" upper bound since those are the true hyperplanes one would hash with in this special case of no kernel mapping; the latter serves as a sanity check baseline to verify the meaningfulness of the KLSH-derived weights.

Fig. 7 shows the results in terms of the raw distribution of Anderson-Darling scores (left) and the corresponding success rates (right) as a function of p. These results clearly demonstrate that KLSH generates hyperplanes from the desired distribution, nearly matching the normality statistics of the upper bound, even when using rather few training points (e.g., for $p = 150$). This is a good sign for hashing complexity since our algorithm's query time is linearly dependent on the value of p. In contrast to KLSH, the baseline that takes a random combination of training points yields hyperplanes unlikely to be normal. Interestingly, the success rate of the random baseline increases with p, perhaps a side effect of the CLT. Nonetheless, as expected, KLSH is significantly better, and makes it possible to generate Gaussian-distributed hyperplanes with very few sampled training points for this data set.

6.7 Comparison to Random Fourier Features

We now focus on a comparison of KLSH applied to the Gaussian kernel with an existing technique based on random Fourier features. In [31], it was shown that any shift-invariant kernel (such as the Gaussian kernel) can be approximated by a low-dimensional embedding via random Fourier features.

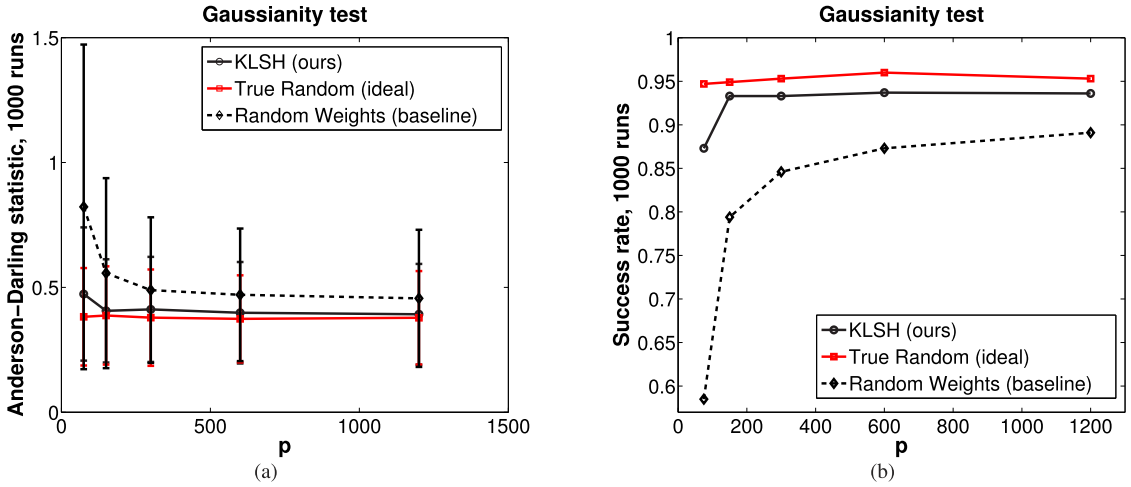

Fig. 7. Empirical verification of the "Gaussianity" of the random hyperplanes generated by KLSH, for a linear kernel with image patch data. We compare KLSH's hyperplanes to both those generated by directly sampling from a Gaussian distribution ("True Random"), as well as those generated by taking a random weighted combination of training points from the same pool ("Random Weights"). (a) Distribution of Anderson-Darling normality statistics computed for 1,000 trials for each value of p; lower values indicate greater normality. (b) The corresponding success rates across all 1,000 trials for each approach ($\alpha = 0.05$). These results demonstrate that KLSH generates hyperplanes from the desired distribution, nearly matching the normality statistics of the upper bound (hyperplanes sampled from the Gaussian directly), even for rather low values of p. In contrast, the baseline that takes a random combination of training points (dotted curves) yields hyperplanes unlikely to be normal. See text for more details.

For example, to construct a d-dimensional embedding for the Gaussian kernel, each dimension i is parameterized by a Gaussian vector $\boldsymbol{\omega}_i$ and a value b_i chosen uniformly at random in $[0, 2\pi]$, and then the ith dimension of the embedding for an arbitrary data point $\boldsymbol{x}$ is given by $\sqrt{2/d}\cos(\boldsymbol{\omega}_i^T \boldsymbol{x} + b_i)$. In follow-up work, Raginsky and Lazebnik [20] showed how to extend this embedding for locality-sensitive hashing by appropriately binarizing the random Fourier embedding. For both the binarized and nonbinarized embeddings, error bounds were proven for approximations to the corresponding shift-invariant kernels.

We compare the hash embeddings obtained by KLSH (using the Gaussian kernel) with those of the binarized random Fourier features. We evaluate over the five vector-based data sets used in [49] for testing the ability of a hashing method to approximate the true nearest neighbors in the input (kernel) space, with the evaluation criterion adopted from [11]. These data sets are: the Photo Tourism data; Peekaboom and LabelMe, two image data sets on top of which global Gist descriptors have been extracted; MNIST, the standard handwritten digits data set; and Nursery, one of the UCI data sets. The x-axis in the plots in Fig. 8 corresponds to the number of hash bits used, while the y-axis corresponds to how well the nearest neighbors in the Hamming space match the nearest neighbors in the original space. Specifically, for each data set, we randomly choose 1,000 points as our "database" and 3,000 points as our set of queries. Among database points, we find the average distance between every point and its 50th nearest neighbor (using the Gaussian kernel), and use this as a threshold for determining true nearest neighbors. Then, for each query, we determine which query-database pairs have Hamming distance less than or equal to 3, and plot the fraction of these that are true nearest neighbors, which we denote as the overlap score. We would expect that the fraction should approach 1 as the number of bits increases.

The results of the evaluation over the five data sets are shown in Fig. 8. Since the binarized Fourier embeddings are optimized for the Gaussian kernel, we expected them to outperform our more general technique. Surprisingly, this does not seem to be the case, and our method appears to be a better match to the Gaussian kernel under our evaluation criterion on each of the benchmark data sets.

One possible explanation for the poorer performance of the Fourier features is that the analysis of the binarized embeddings given in [20] does not explicitly bound any measure between nearest neighbors in the binary space and nearest neighbors in the original Gaussian kernel space. Because our method is additionally not restricted to shift-invariant kernels, the results suggest that our approach may be more desirable in general for hashing as compared to using Fourier features.

7 Conclusions

We presented a general algorithm to draw hash functions that are locality sensitive for arbitrary kernel functions, thereby permitting sublinear time approximate similarity search. This significantly widens the accessibility of LSH to generic normalized kernel functions, whether or not their underlying feature space is known. While our experiments focus on visual data, there is nothing about our approach that is specific to image features. Furthermore, since our method does not require assumptions about the data distribution or input, it is directly applicable to many existing useful measures that have been studied for image search and other domains.

Acknowledgments

The authors thank the anonymous reviewers for helpful comments. They thank Rob Fergus, Antonio Torralba, and Bill Freeman for the Tiny Image data, and Yong Jae Lee for the Flickr data. This research was supported in part by US National Science Foundation (NSF) CAREER award IIS-0747356 and the Henry Luce Foundation.

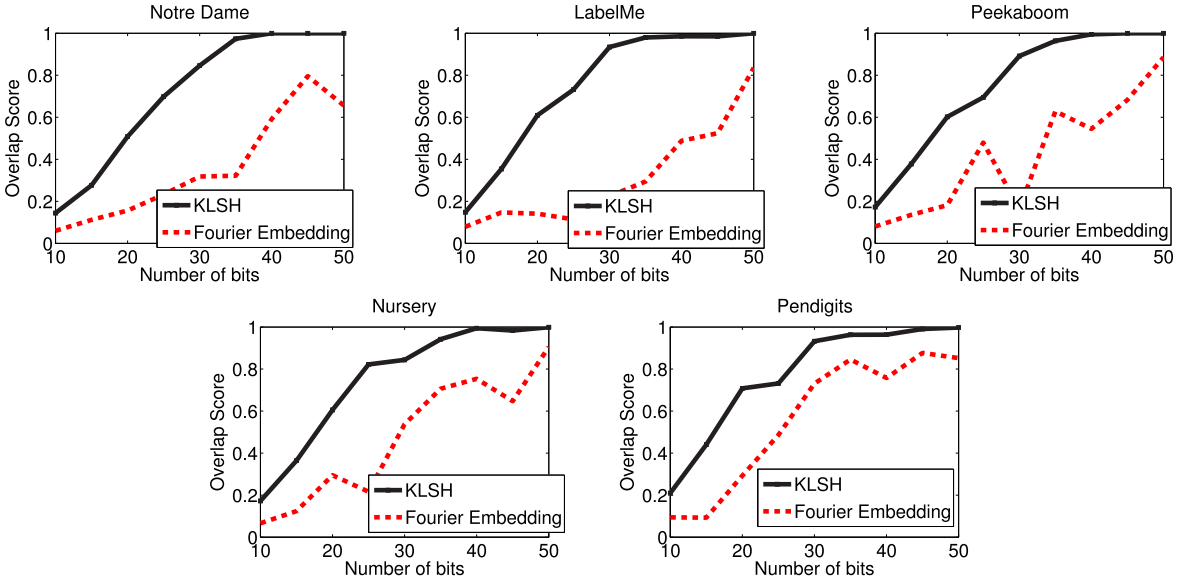

Fig. 8. Results comparing the binarized random Fourier features of [20] with KLSH. Both methods are specialized to the Gaussian kernel, and are applied to the data sets from [49]—Notre Dame, Labelme, Peekaboom, Nursery, and Pendigits. The y-axis corresponds to the fraction of points that have Hamming distance less than or equal to 3 which are true nearest neighbors under the input Gaussian kernel (higher is better). See text for details.

REFERENCES

[1] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast Pose Estimation with Parameter-Sensitive Hashing," *Proc. IEEE Int'l Conf. Computer Vision,* 2003.

[2] *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice,* G. Shakhnarovich, T. Darrell, and P. Indyk, eds. The MIT Press, 2006.

[3] K. Grauman and T. Darrell, "Pyramid Match Hashing: Sub-Linear Time Indexing over Partial Correspondences," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2007.

[4] P. Jain, B. Kulis, and K. Grauman, "Fast Image Search for Learned Metrics," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2008.

[5] O. Chum, J. Philbin, and A. Zisserman, "Near Duplicate Image Detection: Min-Hash and tf-idf Weighting," *Proc. British Machine Vision Conf.,* 2008.

[6] A. Gionis, P. Indyk, and R. Motwani, "Similarity Search in High Dimensions via Hashing," *Proc. 25th Int'l Conf. Very Large Data Bases,* 1999.

[7] M. Charikar, "Similarity Estimation Techniques from Rounding Algorithms," *Proc. ACM Symp. Theory of Computing,* 2002.

[8] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios, "BoostMap: A Method for Efficient Approximate Similarity Rankings," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2004.

[9] A. Torralba, R. Fergus, and Y. Weiss, "Small Codes and Large Image Databases for Recognition," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2008.

[10] R. Salakhutdinov and G. Hinton, "Semantic Hashing," *Proc. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval,* 2007.

[11] Y. Weiss, A. Torralba, and R. Fergus, "Spectral Hashing," *Proc. Advances in Neural Information Processing Systems,* 2009.

[12] H. Zhang, A. Berg, M. Maire, and J. Malik, "SVM-KNN: Discriminative Nearest Neighbor Classification for Visual Category Recognition," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2006.

[13] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid, "Local Features and Kernels for Classification of Texture and Object Categories: A Comprehensive Study," *Int'l J. Computer Vision,* vol. 73, no. 2, pp. 213-238, 2007.

[14] M. Varma and D. Ray, "Learning the Discriminative Power-Invariance Trade-Off," *Proc. IEEE Int'l Conf. Computer Vision,* 2007.

[15] J. Rice, *Mathematical Statistics and Data Analysis.* Duxbury Press, 2001.

[16] L. Fei-Fei, R. Fergus, and P. Perona, "Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories," *Proc. Workshop Generative Model Based Vision,* June 2004.

[17] N. Snavely, S. Seitz, and R. Szeliski, "Photo Tourism: Exploring Photo Collections in 3D," *Proc. Siggraph,* 2006.

[18] G. Hua, M. Brown, and S. Winder, "Discriminant Embedding for Local Image Descriptors," *Proc. IEEE Int'l Conf. Computer Vision,* 2007.

[19] A. Torralba, R. Fergus, and W.T. Freeman, "80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 30, no. 11, pp. 1958-1970, Nov. 2008.

[20] M. Raginsky and S. Lazebnik, "Locality Sensitive Binary Codes from Shift-Invariant Kernels," *Proc. Advances in Neural Information Processing Systems,* 2009.

[21] J. Freidman, J. Bentley, and A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Trans. Math. Software,* vol. 3, no. 3, pp. 209-226, Sept. 1977.

[22] D. Nister and H. Stewenius, "Scalable Recognition with a Vocabulary Tree," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2006.

[23] S. Obdrzalek and J. Matas, "Sub-Linear Indexing for Large Scale Object Recognition," *Proc. British Machine Vision Conf.,* 2005.

[24] J. Beis and D. Lowe, "Shape Indexing Using Approximate Nearest-Neighbour Search in High Dimensional Spaces," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 1997.

[25] J. Uhlmann, "Satisfying General Proximity/Similarity Queries with Metric Trees," *Information Processing Letters,* vol. 40, pp. 175-179, 1991.

[26] P. Ciaccia, M. Patella, and P. Zezula, "M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces," *Proc. Int'l Conf. Very Large Data Bases,* Aug. 1997.

[27] P. Indyk and R. Motwani, "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality," *Proc. 30th Symp. Theory of Computing,* 1998.

[28] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-Sensitive Hashing Scheme Based on p-Stable Distributions," *Proc. Symp. Computational Geometry,* 2004.

[29] A. Andoni, "Nearest Neighbor Search: The Old, the New, and the Impossible," PhD dissertation, MIT, 2009.

[30] A. Broder, "On the Resemblance and Containment of Documents," *Proc. Compression and Complexity of Sequences,* 1997.

[31] A. Rahimi and B. Recht, "Random Features for Large-Scale Kernel Machines," *Proc. Advances in Neural Information Processing Systems,* 2007.

[32] M. Balcan, A. Blum, and S. Vempala, "Kernels as Features: On Kernels, Margins, and Low-Dimensional Mappings," *Machine Learning,* vol. 65, no. 1, pp. 79-94, 2006.
[33] M. Goemans and D. Williamson, "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming," *J. ACM,* vol. 42, no. 6, pp. 1115-1145, 1995.
[34] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear Component Analysis as a Kernel Eigenvalue Problem," *Neural Computation,* vol. 10, pp. 1299-1319, 1998.
[35] J. Hoffmann-Jorgensen and G. Pisier, "The Law of Large Numbers and the Central Limit Theorem in Banach Spaces," *Ann. Probability,* vol. 4, pp. 587-599, 1976.
[36] J. Yang, X. Gao, D. Zhang, and J. Yang, "Kernel ICA: An Alternative Formulation and Its Application to Face Recognition," *Pattern Recognition,* vol. 38, no. 10, pp. 1784-1787, Oct. 2005.
[37] K. Grauman and T. Darrell, "The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features," *Proc. IEEE Int'l Conf. Computer Vision,* 2005.
[38] H. Ling and S. Soatto, "Proximity Distribution Kernels for Geometric Context in Category Recognition," *Proc. IEEE Int'l Conf. Computer Vision,* 2007.
[39] B. Kulis, P. Jain, and K. Grauman, "Fast Similarity Search for Learned Metrics," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 31, no. 12, pp. 2143-2157, Dec. 2009.
[40] J. Davis, B. Kulis, P. Jain, S. Sra, and I. Dhillon, "Information-Theoretic Metric Learning," *Proc. Int'l Conf. Machine Learning,* 2007.
[41] L.W. Renninger and J. Malik, "When Is Scene Recognition Just Texture Recognition?" *Vision Research,* vol. 44, pp. 2301-2311, 2004.
[42] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid, "Local Features and Kernels for Classification of Texture and Object Categories: A Comprehensive Study," *Int'l J. Computer Vision,* vol. 73, no. 2, pp. 213-238, 2007.
[43] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning Realistic Human Actions from Movies," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2008.
[44] K. Mikolajczyk and C. Schmid, "Scale and Affine Invariant Interest Point Detectors," *Int'l J. Computer Vision,* vol. 1, no. 60, pp. 63-86, Oct. 2004.
[45] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust Wide Baseline Stereo from Maximally Stable Extremal Regions," *Proc. British Machine Vision Conf.,* 2002.
[46] D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int'l J. Computer Vision,* vol. 60, no. 2, pp. 91-110, 2004.
[47] B. Kulis and K. Grauman, "Kernelized Locality-Sensitive Hashing for Scalable Image Search," *Proc. IEEE Int'l Conf. Computer Vision,* 2009.
[48] T. Anderson and D. Darling, "Asymptotic Theory of Certain "Goodness-of-Fit" Criteria Based on Stochastic Processes," *Annals of Math. Statistics,* vol. 23, no. 2, pp. 193-212, 1952.
[49] B. Kulis and T. Darrell, "Learning to Hash with Binary Reconstructive Embeddings," *Proc. Advances in Neural Information Processing Systems,* 2009.

Brian Kulis received the BA degree from Cornell University in computer science and mathematics in 2003 and the PhD degree in computer science from the University of Texas in 2008. He is an assistant professor of computer science at Ohio State University. His research focuses on machine learning, data mining, and large-scale optimization. Previously, he was a postdoctoral fellow in the Electrical Engineering and Computer Science Department at the University of California Berkeley and was also affiliated with the International Computer Science Institute. For his research, he has won three best student paper awards at top-tier conferences—two at the International Conference on Machine Learning (in 2005 and 2007) and one at the IEEE Conference on Computer Vision and Pattern Recognition (in 2008). He was also the recipient of an MCD graduate fellowship from the University of Texas (2003-2007) and an Award of Excellence from the College of Natural Sciences at the University of Texas. In the fall of 2007, he was a research fellow at the Institute for Pure and Applied Mathematics at the University of California Los Angeles. He is a member of the IEEE.

Kristen Grauman received the BA degree in computer science from Boston College in 2001 and the SM and PhD degrees in computer science from the Massachusetts Institute of Technology (MIT) in 2003 and 2006, respectively. She is the Clare Boothe Luce Assistant Professor in the Department of Computer Science at the University of Texas at Austin. She joined UT-Austin in 2007. Her research focuses on object recognition and visual search. She is a Microsoft Research New Faculty fellow, and a recipient of the US National Science Foundation (NSF) CAREER award and the Howes Scholar Award in computational science. She and her collaborators were awarded the CVPR Best Student Paper Award in 2008 for work on hashing algorithms for large-scale image retrieval and the Marr Prize at ICCV in 2011 for work on modeling relative visual attributes. She serves regularly on the program committees for the major computer vision conferences and is a member of the editorial board for the *International Journal of Computer Vision*. She is a member of the IEEE and IEEE Computer Society.