Revisiting Kernelized Locality-Sensitive Hashing for Improved Large-Scale Image Retrieval

Ke Jiang, Qichao Que, Brian Kulis
Department of Computer Science and Engineering
The Ohio State University
{jiangk, que, kulis}@cse.ohio-state.edu

Abstract

We present a simple but powerful reinterpretation of kernelized locality-sensitive hashing (KLSH), a general and popular method developed in the vision community for performing approximate nearest-neighbor searches in an arbitrary reproducing kernel Hilbert space (RKHS). Our new perspective is based on viewing the steps of the KLSH algorithm in an appropriately projected space, and has several key theoretical and practical benefits. First, it eliminates the problematic conceptual difficulties that are present in the existing motivation of KLSH. Second, it yields the first formal retrieval performance bounds for KLSH. Third, our analysis reveals two techniques for boosting the empirical performance of KLSH. We evaluate these extensions on several large-scale benchmark image retrieval data sets, and show that our analysis leads to improved recall performance of at least 12%, and sometimes much higher, over the standard KLSH method.

1. Introduction

Similarity search (or nearest neighbor (NN) search) for large databases plays a critical role in a number of important vision applications including content-based image and video retrieval. Usually, the data are represented in a high-dimensional feature space, and the number of objects in the database can scale to the billions in modern applications. As such, fast indexing and search is a vital component to many large-scale retrieval systems.

A key theoretical and practical breakthrough for the similarity search problem was the development of locality-sensitive hashing (LSH) [7, 3, 2], which relies on Gaussian random projections for Euclidean distance and can provably retrieve *approximate* nearest neighbors in time that grows sublinearly in the number of database items. In the vision community, LSH has long been employed as one of the core methods for large-scale retrieval [10, 21, 12, 19, 8, 24, 16, 1]. Unfortunately, in some cases, image comparison criteria are based on functions other than the simple Euclidean distance between corresponding image feature vectors, which makes LSH inapplicable in several settings. Some foundational work has been done to extend LSH to kernels satisfying particular conditions, such as hashing for shift-invariant kernels in [21] based on random Fourier features [22]. More generally, Kulis and Grauman [12] proposed a technique called kernelized LSH (KLSH) for approximate nearest neighbor searches with arbitrary kernels, thus extending LSH to situations where only kernel function evaluations are possible. The main idea behind KLSH is to approximate the necessary Gaussian random projections in the kernel space using an appropriate random combination of items from the database, based on an application of the central limit theorem.

Since its publication, KLSH has been used extensively in the computer vision community, and related hashing methods have been built from the KLSH foundations [19, 8, 24, 16]; however, KLSH still suffers from some important drawbacks. First, while Kulis and Grauman show that the central limit theorem ensures that the approximate random projections constructed become true Gaussian random projections as the number of sampled database items gets larger, no bounds are explicitly given to clarify the tradeoff between accuracy and runtime. Even worse, the approach that KLSH uses–that of applying a random projection with a $\mathcal{N}(0, I)$ vector–is conceptually inappropriate in an infinite-dimensional kernel space since, as we will discuss later, no such canonical Gaussian distribution even exists.

In this paper, we present a simple yet powerful reinterpretation of KLSH, which we describe in Section 3. This new perspective gracefully resolves the "infinite Gaussian" issue and provides us with the first explicit performance bounds to clearly demonstrate tradeoffs between runtime and retrieval accuracy. Crucially, this tradeoff also reveals two potential techniques which boost the empirical performance of vanilla KLSH. In particular, we show how to modify KLSH to obtain improvements in recall performance of

at least 12%, and sometimes much higher, on all the benchmarks examined in Section 5.

1.1. Related work and our contributions

There is limited existing theoretical analysis [24] of KLSH based on Nyström approximation bounds [5]. However, this analysis only examines the average error between the original kernel function values and the approximations made by KLSH, and does not provide any bounds on retrieval performance. Moreover, as we will discuss in Section 3, there is a subtle difference between KLSH and the Nyström method, rendering the aforementioned analysis problematic. Further, we will demonstrate in Section 5 that KLSH bears advantages over the Nyström method when the number of database items selected to approximate kernel functions is relatively small.

There have been conflicting views about the comparison of KLSH and LSH after applying kernel PCA [23] to the data. For example, some work [12] has concluded that KLSH has a clear performance edge over KPCA+LSH, while these results are contradicted by the empirical analysis in [1, 4] which demonstrated that LSH after a KPCA projection step shows a significant improvement over KLSH. We will see in Section 3 that these two seemingly disparate methods are *equivalent* (up to how the random vectors are drawn in the two approaches), and the performance gap observed in practice is largely *only* due to the choice of parameters. Although [1] gives some error analysis for the LSH after a PCA projection step using the Cauchy-Schwarz inequality, no explicit performance bounds are proved. Thus, it fails to show the interesting tradeoffs and retrieval bounds that we derive in Section 4.1.

Recently, there has been work on kernel approximation-based visual search methods. Asymmetric sparse kernel approximations [4] aim to approximate the nearest neighbor search with an asymmetric similarity score computed from m randomly selected landmarks. It has shown excellent empirical performance with $m = 8192$. Kernelized random subspace hashing (KRSH) [18] attempts to randomly generate the orthogonal bases for an m-dimensional subspace in kernel space. Then classical hashing schemes are employed on the projection to this subspace. These approaches may be viewed as variants of the Nyström method; we note that the authors of [18] were able to provide a preservation bound on inter-vector angles and showed better angle preservation than KLSH.

Our main contribution can be summarized as threefold. First, we provide a new interpretation of KLSH, which not only provides a firmer theoretical footing but also resolves issues revolving around comparisons between KLSH and LSH after projection via kernel PCA. Second, we are able to derive the first formal retrieval bounds for KLSH, demonstrating a tradeoff similar to the classic bias-variance tradeoff in statistics. Lastly and most importantly, our analysis reveals two potential techniques for boosting the performance of standard KLSH. We successfully validate these techniques on large-scale benchmark image retrieval datasets, showing at least a 12% improvement in recall performance across all benchmarks.

2. Background: LSH for Similarities and KLSH

Assume that the database is a set of n samples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \in \mathbb{R}^d$. Given a query $\boldsymbol{q} \in \mathbb{R}^d$ and a user-defined kernel function $\kappa(\cdot, \cdot) = \langle \Phi(\cdot), \Phi(\cdot) \rangle$ with the feature map $\Phi : \mathbb{R}^d \to \mathcal{H}$, where $\mathcal{H}$ is the implicit reproducing kernel Hilbert space (RKHS), we are interested in finding the most similar item in the database to the query $\boldsymbol{q}$ with respect to $\kappa(\cdot, \cdot)$, i.e., $\text{argmax}_i \kappa(\boldsymbol{q}, \boldsymbol{x}_i)$.

LSH is a general technique for constructing and applying hash functions to the data such that two similar objects are more likely to be hashed together [7, 3]. When the hash functions are binary, and b hash functions are employed, this results in a projection of the data into a b-dimensional binary (Hamming) space. Note that there are several possible LSH schemes, including non-binary hashes, but we will focus mainly on binary hashing in this paper. One advantage to binary hashing is that nearest neighbor queries in the Hamming space can be implemented very quickly; tree-based data structures can be used to find approximate nearest neighbors in the Hamming space in time sub-linear in the number of data points [2], and even an exact nearest neighbor computation can be performed extremely quickly in the Hamming space.

In order to meet the locality-sensitive requirement for similarity functions, each hash function h should satisfy [2]:

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

Here, we only consider normalized kernel functions $\kappa(\cdot, \cdot) \in [0, 1]$; for un-normalized kernels, our results can be applied after normalization via $\kappa(\boldsymbol{x}, \boldsymbol{y}) / \sqrt{\kappa(\boldsymbol{x}, \boldsymbol{x}) \cdot \kappa(\boldsymbol{y}, \boldsymbol{y})}$. Given valid hash families, the query time for retrieving $(1 + \epsilon)$-nearest neighbors is bounded by $O(n^{1/(1+\epsilon)})$ for the Hamming distance [7, 2]. For the linear kernel $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j$ (i.e. $\Phi(\boldsymbol{x}) = \boldsymbol{x}$) for normalized histograms, Charikar [2] showed that a hash family can be constructed by rounding the output of the product with a random hyperplane:

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

where $\boldsymbol{r} \in \mathbb{R}^d$ is a random vector sampled from the standard multivariate Gaussian distribution (i.e., from $\mathcal{N}(0, I)$). This can be directly extended to kernels having known explicit representations with dimension $d_\Phi < \infty$. However, this

is not the case for many commonly-used kernels in vision applications.

In order to deal with arbitrary kernels, KLSH attempts to mimic this technique by drawing approximate Gaussian random vectors in the RKHS via the central-limit theorem (CLT). The key advantage to this approach is that the resulting hash function computation can be accomplished solely using kernel function evaluations.

Considering $\{\Phi(\boldsymbol{x}_1), \ldots, \Phi(\boldsymbol{x}_t)\}$ as t realizations of random variable $\Phi(X)$ with known mean μ and covariance operator C, the classical CLT [6] ensures that the random vector $C^{-1/2}\tilde{z}_t = C^{-1/2}[\sqrt{t}(\frac{1}{t}\sum_{i=1}^{t}\Phi(\boldsymbol{x}_i) - \mu)]$ converges to a standard Gaussian random vector as $t \to \infty$. Therefore, the hash family (2) can be approximated by:

$$h(\Phi(x)) = \begin{cases} 1, & \text{if } \Phi(x)^T C^{-1/2}\tilde{\boldsymbol{z}}_t \geq 0 \\ 0, & \text{otherwise} \end{cases}. \quad (3)$$

In practice, the mean μ and the covariance matrix C of the data are not known and must be estimated through a random set $S = \{\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_m\}$ from the database. Choosing the t random samples used in the CLT approximation from S ($t < m$), [12] showed that (3) has the convenient form

$$h(\Phi(\boldsymbol{x})) = \text{sign}(\sum_{i=1}^{m} \boldsymbol{w}(i)\kappa(\boldsymbol{x}, \hat{\boldsymbol{x}}_i)), \quad (4)$$

where $\boldsymbol{w} = \bar{K}^{-1/2}\boldsymbol{e}_{S_t}$ with $\bar{K}$ the $m \times m$ centered kernel matrix formed by $\{\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_m\}$ and $\boldsymbol{e}_{S_t}$ an $m \times 1$ vector with ones at the entries corresponding to the t samples. Note that some constant scaling terms have been dropped without changing the hash function evaluation.

The validity of KLSH relies heavily on the central limit theorem. One crucial question that is not addressed in [12] is the existence of $\mathcal{N}(0, I_\infty)$ in the case where the kernel function is based on an infinite-dimensional embedding (such as the Gaussian kernel). Unfortunately, there is no such canonical Gaussian distribution in an RKHS, as given by the following lemma.

Lemma 1. *[15] A Gaussian distribution with covariance operator C in a Hilbert space exists if and only if, in an appropriate base, C has a diagonal form with non-negative eigenvalues and the sum of these eigenvalues is finite.*

As implied by Lemma 1, the convergence to the standard Gaussian in an infinite-dimensional Hilbert space is not grounded, as the eigenvalues of the covariance operator sum to infinity.[1] As such, the motivation for KLSH is problematic at best and, at worst, could render KLSH inappropriate for many of the retrieval settings for which it was specifically designed. At the same time, KLSH has shown solid empirical performance on kernels associated with infinite-dimensional $\mathcal{H}$ [12]. How can we explain the discrepancy between the empirical performance and the lack of a solid theoretical motivation? We resolve these issues in the next section.

[1]Note that the central limit theorem does still apply in Hilbert spaces, but the covariance operators must always have finite trace.

3. A New Interpretation of KLSH

In the following, we will provide a simple but powerful reinterpretation of KLSH, which will allow us to circumvent the aforementioned issues with infinite-dimensional $\mathcal{H}$. In particular, we will show that KLSH may be viewed precisely as KPCA+LSH, except that the Gaussian vectors drawn for LSH are drawn via the CLT in the KPCA projected space.

KLSH as Explicit Embedding. Let us take a deeper look at the hash function (3). Utilizing the eigen-decomposition of the covariance C, we can write

$$g(\Phi(\boldsymbol{x})) = \sum_{i=1}^{d_\Phi} \frac{1}{\sqrt{\lambda_i}}(\boldsymbol{v}_i^T\Phi(\boldsymbol{x})) \cdot (\boldsymbol{v}_i^T\tilde{\boldsymbol{z}}_t) \quad (5)$$

$$= \sum_{i=1}^{d_\Phi}(\boldsymbol{v}_i^T\Phi(\boldsymbol{x})) \cdot \left(\frac{1}{\sqrt{\lambda_i}}\boldsymbol{v}_i^T\tilde{\boldsymbol{z}}_t\right), \quad (6)$$

where $h(\Phi(\boldsymbol{x})) = \text{sign}[g(\Phi(\boldsymbol{x}))]$, $\lambda_1 \geq \cdots \geq \lambda_m \geq \cdots \geq 0$ are the eigenvalues of C with $\boldsymbol{v}_i$ the corresponding eigenvectors. In many situations, the dimension d_Φ of Φ is infinite.

If we perform a truncation at k, we obtain a finite-dimensional representation for Φ with $\sum_{i=k+1}^{d_\Phi}\lambda_i$ as the expected approximation error. The resulting sum in (6) after this truncation can be viewed as an inner product between two k-dimensional vectors. Specifically, the first vector is $(\boldsymbol{v}_1^T\Phi(\boldsymbol{x}), \boldsymbol{v}_2^T\Phi(\boldsymbol{x}), \ldots, \boldsymbol{v}_k^T\Phi(\boldsymbol{x}))$, which is simply the projection of $\Phi(\boldsymbol{x})$ onto the subspace spanned by the top k principal components. For the other k-dimensional vector $(\frac{1}{\sqrt{\lambda_1}}\boldsymbol{v}_1^T\tilde{\boldsymbol{z}}_t, \ldots, \frac{1}{\sqrt{\lambda_k}}\boldsymbol{v}_k^T\tilde{\boldsymbol{z}}_t)$, we plug in the definition of $\tilde{\boldsymbol{z}}_t$ to obtain for each i:

$$\frac{1}{\sqrt{\lambda_i}}\boldsymbol{v}_i^T\tilde{\boldsymbol{z}}_t = \frac{\sqrt{t}}{\sqrt{\lambda_i}}\left(\frac{1}{t}\sum_{j\in S}\boldsymbol{v}_i^T\Phi(\boldsymbol{x}_j) - \boldsymbol{v}_i^T\mu\right) \quad (7)$$

$$= \frac{\sqrt{t}}{\sqrt{\lambda_i}}(\bar{y}_{i,t} - \mu_i) \quad (8)$$

$$\overset{appr}{\sim} \mathcal{N}(0,1), \quad (9)$$

where $\bar{y}_{i,t} = \frac{1}{t}\sum_{j\in S}\boldsymbol{v}_i^T\Phi(\boldsymbol{x}_j)$ (i.e., the t sample average of the projection of Φ onto the $\boldsymbol{v}_i$-direction), μ_i is the projection of μ onto the $\boldsymbol{v}_i$-direction, and the last approximation comes from the central limit theorem. Since we do not know μ and C explicitly, we can use plug-in sample estimates.

Under the above interpretation, we see that (6) after truncation may be viewed as computing an inner product between two k-dimensional vectors: the first vector is the data point $\Phi(\boldsymbol{x})$ after projecting via KPCA, and the second vector is a Gaussian random vector. In other words, this sum may be interpreted as first computing KPCA, and then using the CLT to draw random vectors for LSH in the projected space. Since KLSH uses a sample of m data points from the data set to estimate the covariance and mean, we automatically obtain truncation of the sum as the estimated covariance is at most (m-1)-dimensional (it is (m-1)-dimensional because of the centering operation). We can therefore see that KLSH performs LSH after projecting to an (m-1)-dimensional space via principal component analysis in $\mathcal{H}$. Thus, KLSH is conceptually equivalent to applying LSH after a PCA projection in $\mathcal{H}$. The only difference is that KLSH uses the central limit theorem to approximately draw Gaussian vectors in the projected space, whereas standard KPCA+LSH draws the random vectors directly. Note that the central limit theorem is known to converge at a rate of $O(t^{-1/2})$, and in practice obtains very good approximations when $t \geq 30$ [6, 12]; as such, results of the two algorithms are within small random variations of each other.

In summary, we are now able to explain the empirical performance of [12] and also avoid the technical issues with the non-existence of standard Gaussian distributions in infinite-dimensional Hilbert spaces. As we will see, this perspective will lead to a performance bound for KLSH in Section 4.1, which also sheds light on simple techniques which could potentially improve the retrieval performance.

Comparison with the Nyström method. As arguably the most popular kernel matrix approximation method, the Nyström method [5] uses a low-rank approximation to the original kernel matrix in order to reduce the complexity of computing inverses. Although KLSH bears some similarities with the Nyström method as pointed out in [24], here we briefly clarify the differences between these approaches.

Given m anchor points, $\{\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_m\}$, the Nyström method constructs a rank-r approximation to the kernel matrix over all the database items as

$$\tilde{K}_r = K_{nm} \hat{K}_r^{\dagger} K_{nm}^T, \tag{10}$$

where $\tilde{K}_r$ is the rank-k approximation to the original kernel matrix K, K_{nm} is the $n \times m$ kernel matrix with the (i,j)-th entry as $\kappa(\boldsymbol{x}_i, \hat{\boldsymbol{x}}_j), i = 1, \ldots, n, j = 1, \ldots, m$, and $\hat{K}_r^{\dagger}$ is the pseudo-inverse of $\hat{K}_r$. which is the best rank-k approximation to the $m \times m$ kernel matrix $\hat{K}$ formed by the selected anchor points. We can write the best rank-r approximation as $\hat{K}_r = \hat{U}_r \hat{D}_r \hat{U}_r^T$, where $\hat{D}_r$ is the $r \times r$ diagonal matrix whose diagonal entries are the leading r eigenvalues of $\hat{K}$, and $\hat{U}_r$ is the $m \times r$ matrix with each column as the corresponding eigenvector of $\hat{K}$. Now, we can derive a vector representation for each data as

$$\psi(\boldsymbol{x}) = \hat{D}_r^{-1/2} \hat{U}_r^T (\kappa(\boldsymbol{x}, \hat{\boldsymbol{x}}_1), \ldots, \kappa(\boldsymbol{x}, \hat{\boldsymbol{x}}_m))^T. \tag{11}$$

This is very similar to the representation used in equation (4), where we can write the vector representation (considering also a rank-r approximation) as

$$\phi(\boldsymbol{x}) = \bar{D}_r^{-1/2} \bar{U}_r^T (\kappa(\boldsymbol{x}, \hat{\boldsymbol{x}}_1), \ldots, \kappa(\boldsymbol{x}, \hat{\boldsymbol{x}}_m))^T, \tag{12}$$

where $\bar{D}_r$ and $\bar{U}_r$ are the diagonal matrix with leading r eigenvalues and the $m \times r$ matrix with each column the corresponding eigenvectors of $\bar{K}$, respectively. $\bar{K}$ is the centered version of $\hat{K}$.

Although they look similar in format, the two representations turn out to yield very different hashing performance. Even though the Nyström method aims to approximate the whole kernel matrix, we point out that the "centering" operation used by KLSH is essential to give strong performance, especially when m is relatively small. We will empirically explore this issue further in Section 5.

4. Theoretical Analysis

In this section, we present our main theoretical results, which consist of a performance bound for the KLSH method analogous to results for standard LSH. Perhaps more importantly, our analysis suggests two simple techniques to improve the empirical performance of the KLSH method.

4.1. A performance bound

We present a theoretical analysis of KLSH via the "KPCA+LSH" perspective. We make the assumption that KLSH draws truly random Gaussian vectors in the projected subspace, for simplicity of presentation. A more refined analysis would also include the error included by approximating the Gaussian random vectors via the central limit theorem. Such analysis should be possible to incorporate via the Berry-Esseen theorem [6]; however, as discussed earlier, in practice the CLT-based random vectors provide sufficient approximations to Gaussian vectors (see also Section 6.6 of [13] for a more in-depth empirical comparison).

We first formulate the setting and assumptions for our main results. Suppose $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ are n i.i.d. random samples drawn from a probability measure p. Given a query $\boldsymbol{q}$ and a similarity function κ, often referred to as a kernel function, we want to find $\boldsymbol{y}_q^* = \text{argmax}_{i \in S} \kappa(\boldsymbol{q}, \boldsymbol{x}_i)$. As is standard in the machine learning literature, when the kernel κ is positive semi-definite, it can be (implicitly) associated with a feature map Φ that maps original data points $\boldsymbol{x}$ into a high dimensional or even infinite dimensional feature space $\mathcal{H}$. The kernel function $\kappa(\boldsymbol{x}, \boldsymbol{y})$ gives the inner product $\langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}) \rangle$ in the feature space. The feature map and

p together induce a distribution on $\mathcal{H}$, which has a covariance operator, denoted by C. Finally, let $\lambda_1 \geq \lambda_2 \geq \cdots$ and $\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots$ be the corresponding eigenvalues and eigenvectors of C, respectively.

As we assume that we do not have the explicit formula for the feature map, we project $\Phi(\boldsymbol{x})$ onto a k-dimensional subspace V_k, which is spanned by the first k principal components of C. Empirically, we use a sample of m points to estimate C and V_k. Note that after projection, the new kernel function becomes

$$\hat{\kappa}(\boldsymbol{x}, \boldsymbol{y}) = \langle P_{\hat{V}_k}(\Phi(\boldsymbol{x})), P_{\hat{V}_k}(\Phi(\boldsymbol{y}))\rangle, \tag{13}$$

which is no longer normalized. Here, $\hat{V}_k$ is the sample estimator of V_k. To fit into the standard LSH framework, we normalize this kernel by

$$\hat{\kappa}_n(\boldsymbol{x}, \boldsymbol{y}) = \frac{\hat{\kappa}(\boldsymbol{x}, \boldsymbol{y})}{N(\boldsymbol{x})N(\boldsymbol{y})}, \tag{14}$$

where $N(\boldsymbol{x}) = \sqrt{\hat{\kappa}(\boldsymbol{x}, \boldsymbol{x})} = \|P_{\hat{V}_k}(\Phi(\boldsymbol{x}))\| \leq 1$. As points with small $N(\boldsymbol{x})$ may cause instability issues, we simply remove them and only keep points with $N(\boldsymbol{x}) \geq 1 - \sqrt{\lambda_k} - \eta$ for proper choice of η. Note that this step should not affect the search result very much; in fact, we prove in Lemma 2 below that the optimal points $\boldsymbol{y}_q^*$ will not be eliminated with high probability. See the supplementary material for a proof.

Lemma 2. *Let* $\delta_k = \frac{\lambda_k - \lambda_{k+1}}{2}$ *and* $\eta = \frac{2}{\delta_k\sqrt{m}}\left(1 + \sqrt{\frac{\xi}{2}}\right)$, *where* k *is the total number of chosen principal components and* m *is the number of points for estimating the eigenspace. With probability at least* $1 - e^{-\xi}$, *for any point* $\boldsymbol{x} \in \mathcal{X}$, *we have*

$$N(\boldsymbol{x}) > 1 - \sqrt{\lambda_k} - \eta. \tag{15}$$

We can see that the probability of eliminating $\boldsymbol{y}_q^*$ is small given the choice of η. Now we can give our main result in the following theorem.

Theorem 3. *Consider an* n*-sample database* $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ *and a query point* $\boldsymbol{q}$. *For any* $\epsilon, \xi > 0$, *with success probability at least* $(1 - e^{-\xi})/2$ *and query time dominated by* $O(n^{\frac{1}{1+\epsilon}})$ *kernel evaluations, the KLSH algorithm retrieves a nearest neighbor* $\hat{\boldsymbol{y}}_{\boldsymbol{q},k}$ *with corresponding bound*

$$\kappa(\boldsymbol{q}, \hat{\boldsymbol{y}}_{q,k}) \geq (1+\epsilon)(1 - \sqrt{\lambda_k} - \eta)\kappa(\boldsymbol{q}, \boldsymbol{y}_q^*) - \epsilon - (2+\epsilon)\left(\sqrt{\lambda_k} + \eta\right)^2, \tag{16}$$

if we only keep those points with $N(\boldsymbol{x}) > 1 - \sqrt{\lambda_k} - \eta$ *for consideration, where* $\eta = \frac{2}{\delta_k\sqrt{m}}\left(1 + \sqrt{\frac{\xi}{2}}\right)$ *and* $0 < \eta < 1 - \sqrt{\lambda_k}$.

Our proof is given in the supplementary material, which utilizes existing bounds for kernel PCA [14] and the standard LSH performance bound [7, 3, 2].

4.2. Discussion and Extensions

Understanding the Bound. The key ingredient of the bound (16) is the error term $\sqrt{\lambda_k} + \frac{2}{\delta_k\sqrt{m}}\left(1 + \sqrt{\xi/2}\right)$. Observe that, as k and m become larger at appropriate rates, both $\sqrt{\lambda_k}$ and η will go to zero. Therefore, as the number of chosen data points and KPCA projections gets large, the bound approaches

$$\kappa(\boldsymbol{q}, \hat{\boldsymbol{y}}_{q,k}) \geq (1+\epsilon)\kappa(\boldsymbol{q}, \boldsymbol{y}_q^*) - \epsilon. \tag{17}$$

Further, as the parameter ϵ from LSH decreases to zero, this bound guarantees us that the true nearest neighbors will be retrieved. Also observe that, with a fixed k, increasing m will always improve the bound, but the $\sqrt{\lambda_k}$ term will be non-zero and will likely yield retrieval errors. This has been empirically shown in [1], namely that the performance of KPCA+LSH saturates when m is large enough, usually in the thousands.

Low-Rank Extension. On the other hand, with a fixed m, there is a trade-off between decreasing λ_k and increasing $\frac{1}{\delta_k}$, similar to the classic bias-variance trade-off in statistics; we expect $\frac{1}{\delta_k}$ to increase as k increases, whereas λ_k will decrease for larger k. As a result, for a fixed m, the best choice of k may actually not be $k = m-1$, as in the original KLSH algorithm, but could be smaller. In light of this, we introduce a *low-rank extension of the standard KLSH*: instead of performing LSH in the $(m$-1)-dimensional subspace of $\mathcal{H}$ as in KLSH, we may actually achieve better results by only projecting into a smaller r-dimensional subspace obtained by the top r principal components. Specifically, we replace $\boldsymbol{w}$ in equation (4) with

$$\boldsymbol{w}_r = \bar{K}_r^{-1/2}\boldsymbol{e}_{S_t}, \tag{18}$$

where $\bar{K}_r$ is the best rank-r approximation to $\bar{K}$. In [1], the authors recommend to use the same number of principal components as the number of hash bits when applying KPCA+LSH, at least for the χ^2 kernel. We will show in Section 5 that in fact the optimal choice of rank is dependent on the dataset and the kernel, not the number of hash bits. Moreover, the performance of KLSH can be quite sensitive on the choice of r.

Extension via Monotone Transformation. Another relevant factor is the decaying property of the eigenvalues of the covariance operator C in the induced RKHS $\mathcal{H}$, which not only affects the λ_k and δ_k, but also the constant which corresponds to the estimation error. Unlike kernel methods for classification and detection, there is a unique property regarding the use of kernel functions for retrieval: applying

a monotone increasing transformation to the original kernel function does not change the ranking provided by the original kernel.

Thus, we can explore popular transformations that can reward us better retrieval performance. For example, given a kernel $\kappa(\boldsymbol{x}, \boldsymbol{y})$, consider an exponential transformation,

$$\kappa_s(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(s * (\kappa(\boldsymbol{x}, \boldsymbol{y}) - 1)\right), \tag{19}$$

where $s > 0$ is the scale parameter and $\kappa_s(\boldsymbol{x}, \boldsymbol{y}) \in (0, 1]$. We can see that the ranking of the nearest neighbors stays the same no matter what value we choose for s as long as $s > 0$. However, changing the scaling impacts the eigenvalues of the covariance operator. In particular, it often slows down the decay with $s > 1$ and will eliminate the decay entirely when $s \to \infty$. In the context of our bound, that means the $1/\delta_k$ term will increase more slowly. Moreover, this also reduces the estimation error of the eigen-space. However, with a value of s too large, we also need a very large k in order to keep the $\sqrt{\lambda_k}$ term small. Thus the scaling must be carefully tuned to balance the trade-off.

5. Empirical Analysis

We now empirically validate the techniques proposed in Section 4.2. A comparison with the Nyström method is also reported.

5.1. Datasets and evaluation protocols

We perform our experiments on three benchmark datasets commonly used in large-scale image search comparisons: MIRFLICKR [9], which consists of 1 million randomly sampled Flickr images represented using 150-dimensional edge histogram descriptors; SIFT1M [11], which has 1 million 128-dimensional local SIFT [17] descriptors; and GIST1M [11], which is comprised of 1 million 960-dimensional GIST [20] descriptors. The query size for all three datasets is 10,000. For MIRFLICKR and GIST1M, all descriptors whose values are all zero were removed from the database.

Throughout the comparisons, we set the parameters of KLSH as follows: we generate a 256-bit hash code, and set $m = 1000, t = 50$ to form the "random" projection matrix, which is equivalent to performing kernel PCA with a sample set of size $m = 1000$. The choice of the number of bits is to largely suppress the performance variation due to randomness, but the conclusions made here are consistent among different choices of number of bits.

We consider two popular kernels[2] from the vision community, namely the χ^2 and intersection kernels for histograms:

$$(\chi^2) : \kappa(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{d} \frac{2\boldsymbol{x}[i]\boldsymbol{y}[i]}{\boldsymbol{x}[i] + \boldsymbol{y}[i]} \tag{20}$$

$$(\text{Intersection}) : \kappa(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{d} \min(\boldsymbol{x}[i], \boldsymbol{y}[i]), \tag{21}$$

where $\boldsymbol{x}[i]$ is the i-th entry of $\boldsymbol{x}$. We perform exhaustive nearest neighbors search and evaluate the quality of retrieval using the Recall@R measure, which is the proportion of query vectors for which the nearest neighbor is ranked in the first R positions. This measure indicates the fraction of queries for which the nearest neighbor is retrieved correctly if a short list of R is verified in the original space. Note here, $R = 100$ only represents 0.01% of the database items. We focus on this measure since it gives a direct measure of the nearest neighbor search performance.

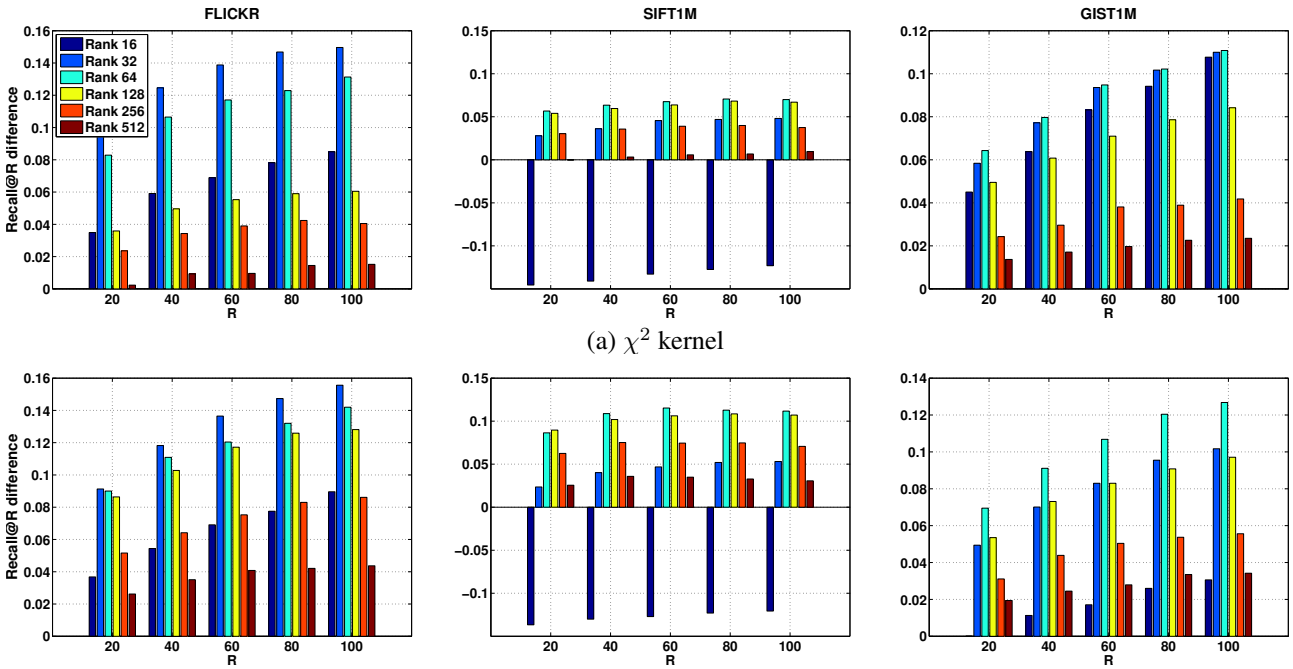
Note that we are deliberately not comparing to other hashing schemes such as semi-supervised hashing methods or optimization-based methods; our goal is to demonstrate how our analysis can be used to improve results of KLSH. Existing work on KLSH and variants has considered comparisons between KLSH and other techniques [12] and, given space restrictions, we do not focus on such comparisons here.

5.2. Effect of rank

Figure 1 shows the effect of the rank r (with all other parameters fixed): all but the smallest rank performed better or at least comparable with the vanilla KLSH. This further confirms the empirical results shown in [1, 4] that KPCA+LSH with smaller number of principal components beats KLSH in retrieval performance. However, this is not the entire story. We can clearly see the performance trade-off as discussed in Section 4.2. Initially, the retrieval performance improves with an increasing number of principal components used, which corresponds to decreasing λ_r. However, at some point, performance drops corresponding to the increase of δ_r. For MIRFLICKR and GIST1M, the difference among ranks can be dramatic, showing the sensitivity of the choice of rank. In addition, the best-performing rank is not only dependent on the kernel chosen but also critically dependent on the dataset examined. Nonetheless, we can still obtain at least 7% in absolute improvement for Recall@100 for the SIFT1M data which is least affected by this trade-off. Here, the performance for different kernels is quite similar, but is divergent for different datasets. For instance, the optimal rank for the MIRFLICKR data is much smaller than that of the SIFT1M and GIST1M. However, we observe no relationship between rank and the number of bits used, making the recommendations of [1] questionable.

Comparison with the Nyström method. Figure 2a shows

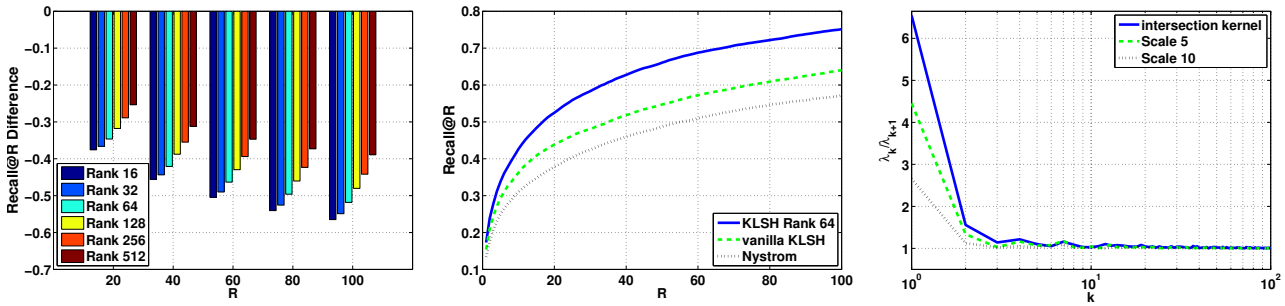
[2]Here, we do not consider fixed-dimensional kernels such as the Hellinger kernel, which are uninteresting for our setting.



(a) χ^2 kernel

(b) Intersection kernel

Figure 1: Recall@R improvement over the vanilla KLSH with $m = 1000$ for rank $\in \{16, 32, 64, 128, 256, 512\}$. (Best viewed in color.)



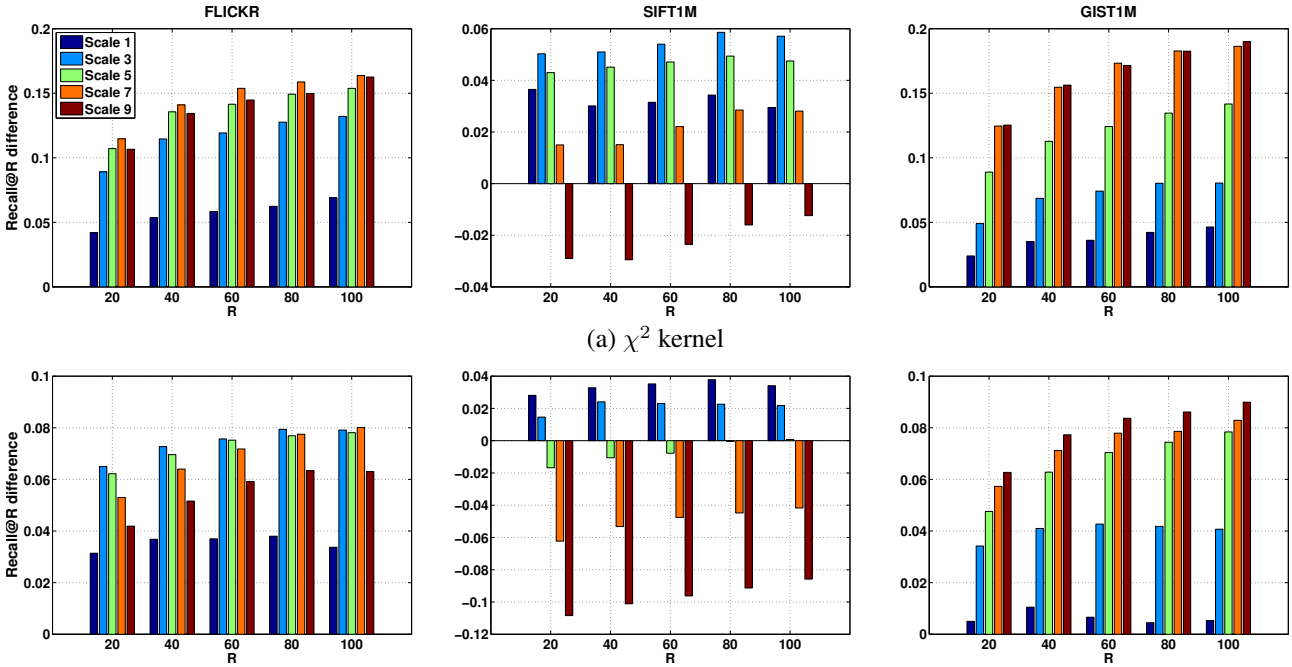
(a) (b) (c)

Figure 2: SIFT1M intersection kernel. (a) Recall@R improvement over the full-rank Nyström method with $m = 1000$ for rank $\in \{16, 32, 64, 128, 256, 512\}$; (b) Recall@$R$ results for KLSH with rank 64, vanilla KLSH and the full-rank Nyström method for $m = 1000$; (c) Change of decaying properties for the exponential transformation with scale parameter $\in \{5, 10\}$. (Best viewed in color.)

the effect of the rank r for the Nyström method: the performance is monotonically increasing with the rank. Moreover, it shows unacceptable retrieval performance even with rank 512. Contrasting with the obvious tradeoff with the choice of ranks in KLSH, these results corroborate our earlier observation that KLSH is indeed different from the Nyström method. Regarding the performance comparison, we can see from Figure 2b that the Nyström method performs worse than both the low-rank version of KLSH and the standard"full-rank" KLSH by a large margin.

5.3. Effect of monotone transformation

Here, we show the effect of the transformation introduced in (19). Note that we are free to choose any possible transformation as long as the chosen transformation is increasingly monotonic; our choice of (19) is simply for illustration. We can see from Figure 2c how changing the scale parameter affects the decay of the eigenvalues. In particular, we see that increasing the scaling slows down the decay and will continue to decrease the decay as s gets larger.



(a) χ^2 kernel

(b) Intersection kernel

Figure 3: Recall@R result improvement over the low-rank KLSH with $m = 1000$ for scale $\in \{1, 3, 5, 7, 9\}$. Here, we use rank 32 for FLICKR, 100 for SIFT1M, and 64 for GIST1M. (Best viewed in color.)

Recall@100	Dataset	KLSH	low-rank	low-rank+transformation	total improvement
χ^2 kernel	Flickr	0.3629	0.5125	0.6762	**+0.3133**
	SIFT1M	0.6942	0.7642	0.8213	**+0.1271**
	GIST1M	0.2252	0.3360	0.5260	**+0.3008**
Intersection kernel	Flickr	0.4182	0.5738	0.6529	**+0.2347**
	SIFT1M	0.6397	0.7468	0.7844	**+0.1447**
	GIST1M	0.2746	0.4014	0.4913	**+0.2167**

Table 1: Summary of absolute improvement for Recall@100.

Figure 3 demonstrates the power of the transformation (with all other parameters fixed): the Recall@R steadily increases as we slow down the decaying speed until the decaying is too gradual. And too large a s may drop the performance significantly. The choice of s and its usefulness also largely depends on both the kernel function and the dataset in consideration: it has more effect on the χ^2 kernel than the intersection kernel. On the other hand, it is more effective in the GIST1M dataset than in SIFT1M. Note here, we are comparing with the original kernel with a fixed rank which favors the original kernel. Thus, there is room for further improvement by choosing a larger rank.

Table 1 summaries the total absolute improvement combining the two techniques together. We can see that the retrieval improvement is at least 12%, sometimes much higher, among all benchmarks.This again validates the merit of our analysis in Section 4.2 regarding the interesting trade-offs shown in our performance bound (16) and demonstrates the power of these simple techniques.

6. Conclusion

We introduced a new interpretation of the kernelized locality-sensitive hashing technique. Our perspective makes it possible to circumvent the conceptual issues of the original algorithm and provides firmer theoretical ground by viewing KLSH as applying LSH on appropriately projected data. This new view of KLSH enables us to prove the first formal retrieval performance bounds, which further suggests two simple techniques for boosting the retrieval performance. We have successfully validated these results empirically on large-scale datasets and showed that the choice of the rank and the monotone transformation are vital to achieve better performance.

Acknowledgement

This work was supported in part by NSF awards IIS-1217433 and RI-1117707.

References

[1] A. Bourrier, F. Perronnin, R. Gribonval, P. Perez, and H. Jegou. Explicit embeddings for nearest neighbor search with Mercer kernels. *Journal of Mathematical Imaging and Vision*, 2015. 1, 2, 5, 6

[2] M. Charikar. Similarity estimation techniques for rounding algorithms. In *ACM Symposium on Theory of Computing*, 2002. 1, 2, 5

[3] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Symposium on Computational Geometry*, 2004. 1, 2, 5

[4] D. Davis, J. Balzer, and S. Soatto. Asymmetric sparse kernel approximations for large-scale visual search. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2014. 2, 6

[5] P. Drineas and M. Mahoney. On the Nyström method for approximating a gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–2175, 2005. 2, 4

[6] W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, 1972. 3, 4

[7] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimension via hashing. In *Proceedings of the International Conference on Very Large DataBases*, 1999. 1, 2, 5

[8] J. He, W. Liu, and S.-F. Chang. Scalable similarity search with optimized kernel hashing. In *Proceedings of the 16th ACM SIGKDD International conference on Knowledge Discovery and Data Mining*, 2010. 1

[9] M. Huiskes, B. Thomee, and M. Lew. New trends and ideas in visual concept detection. In *ACM International Conference on Multimedia Information Retrieval*, 2010. 6

[10] P. Indyk and N. Thaper. Fast image retrieval via embeddings. In *3rd International Workshop on Statistical and Computational Theories of Vision*, 2003. 1

[11] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:117–128, 2011. 6

[12] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2009. 1, 2, 3, 4, 6

[13] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(6):1092–1104, 2012. 4

[14] Z. Laurent and G. Blanchard. On the convergence of eigenspaces in kernel principal component analysis. In *Advances in Neural Information Processing Systems*, 2005. 5

[15] M. Lifshits. *Lectures on Gaussian Processes*. Springer, 2012. 3

[16] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2012. 1

[17] D. Lowe. Distinctive image features from scale invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004. 6

[18] Y. Mu, G. Hua, W. Fan, and S.-F. Chang. Hash-SVM: scalable kernel machines for large-scale visual classification. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2014. 2

[19] Y. Mu, J. Shen, and S. Yan. Weakly-supervised hashing in kernel space. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2010. 1

[20] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelop. *International Journal of Computer Vision*, 42:145–175, 2001. 6

[21] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Advances in Neural Information Processing Systems*, 2010. 1

[22] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2007. 1

[23] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998. 2

[24] H. Xia, P. Wu, S. Hoi, and R. Jin. Boosting multi-kernel locality-sensitive hashing for scalable image retrieval. In *the 35th Annual ACM SIGIR Conference*, 2012. 1, 2, 4