Fast Similarity Search for Learned Metrics

Brian Kulis, *Member*, *IEEE*, Prateek Jain, *Student Member*, *IEEE*, and Kristen Grauman, *Member*, *IEEE*

Abstract—We introduce a method that enables scalable similarity search for learned metrics. Given pairwise similarity and dissimilarity constraints between some examples, we learn a Mahalanobis distance function that captures the examples' underlying relationships well. To allow sublinear time similarity search under the learned metric, we show how to encode the learned metric parameterization into randomized locality-sensitive hash functions. We further formulate an indirect solution that enables metric learning and hashing for vector spaces whose high dimensionality makes it infeasible to learn an explicit transformation over the feature dimensions. We demonstrate the approach applied to a variety of image data sets, as well as a systems data set. The learned metrics improve accuracy relative to commonly used metric baselines, while our hashing construction enables efficient indexing with learned distances and very large databases.

Index Terms—Metric learning, similarity search, locality-sensitive hashing, LogDet divergence, kernel learning, image search.

1 Introduction

As the world's store of digital images and documents continues to grow exponentially, many interesting problems demand fast techniques capable of accurately searching very large databases. This is particularly true as novel data-rich approaches to computer vision begin to emerge. For instance, local feature-based recognition methods require searching huge databases of patch descriptors [1], as do new methods for computing 3D models from multiuser photo databases [2]. Similarly, image or video-based data mining [3], [4] and example-based approaches to pose estimation [5], [6] seek to leverage extremely large image collections, while nearest neighbor classifiers are frequently employed for recognition and shape matching [7], [8]. For most such tasks, the quality of the results relies heavily on the chosen image representation and the distance metric used to compare examples.

Unfortunately, preferred representations tend to be high dimensional [1], [4], and often the best distance metric is one specialized (or learned) for the task at hand [8], [7], [9], rather than, say, a generic euclidean norm or Gaussian kernel. Neither factor bodes well for large-scale search: known data structures for efficient exact search are ineffective for high-dimensional spaces, while existing methods for approximate sublinear time search are defined only for certain standard metrics. Thus, there is a tension when choosing a representation and metric, where one must find a fine balance between the suitability for the problem and the convenience of the computation. We are interested in reducing this tension; to that end, in this work, we develop a general algorithm that enables fast approximate search for a family of learned metrics and kernel functions.

The success of any distance-based indexing, clustering, or classification scheme depends critically on the quality of the chosen distance metric and the extent to which it accurately reflects the data's true underlying relationships, e.g., the category labels or other hidden parameters. A good metric would report small distances for examples that are similar in the parameter space of interest (or that share a class label), and large distances for examples that are unrelated. General-purpose measures, such as L_p norms, are not necessarily well-suited for all learning problems with a given data representation.

Recent advances in metric learning make it possible to learn distance (or kernel) functions that are more effective for a given problem, provided some partially labeled data or constraints are available [10], [11], [12], [13], [14], [15], [8]. By taking advantage of the prior information, these techniques offer improved accuracy when indexing or classifying examples. However, thus far, they have limited applicability to very large data sets since specialized learned distance functions preclude the direct use of known efficient search techniques. Data structures for efficient exact search are known to be ineffective for high-dimensional spaces and can (depending on the data distribution) degenerate to brute-force search [16], [17]; approximate search methods can guarantee sublinear time performance, but are defined only for certain generic metrics. As such, searching for similar examples according to a learned metric currently requires an exhaustive (linear) scan of all previously seen examples, in the worst case. This is a limiting factor that, thus far, deters the use of metric learning with very large databases.

In this work, we introduce a method for fast approximate similarity search with learned Mahalanobis metrics. We formulate randomized hash functions that incorporate side information from partially labeled data or paired constraints so that examples may be efficiently indexed according to the

- *B. Kulis is with the Electrical Engineering and Computer Science Department and the International Computer Science Institute, University of California at Berkeley, 387 Soda Hall, Berkeley, CA 94720. E-mail: kulis@eecs.berkeley.edu.*
- *P. Jain and K. Grauman are with the Department of Computer Sciences, University of Texas at Austin, 1 University Station C0500, Austin, TX 78712. E-mail: {pjain, grauman}@cs.utexas.edu.*

Manuscript received 2 Dec. 2008; revised 22 Apr. 2009; accepted 5 July 2009; published online 5 Aug. 2009.
Recommended for acceptance by K. Boyer, M. Shah, and T. Syeda-Mahmood. For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMISI-2008-12-0827.
Digital Object Identifier no. 10.1109/TPAMI.2009.151.

learned metric without resorting to a naive exhaustive scan of all items. We present a straightforward solution for the case of relatively low-dimensional input vector spaces, and further derive a solution to accommodate very high-dimensional data for which explicit input space computations are infeasible. The former contribution makes fast indexing accessible for numerous existing metric learning methods (e.g., [10], [11], [14], [13], [15]), while the latter is of particular interest for commonly used image and text representations, such as bags of words, multidimensional multiresolution histograms, and other high-dimensional features.

We demonstrate the generality of our approach by applying it to four distinct large-scale search problems: indexing bug reports for a software support system, exemplar-based visual recognition, human body pose estimation, and local image feature indexing. Our method allows rapid and accurate retrieval and improves over relevant state-of-the-art techniques.

2 Related Work

Recent work has yielded various approaches to metric learning, including several techniques to learn a combination of existing kernels [19], [20], as well as methods to learn a Mahalanobis metric [10], [13], [21], [11], [15] and methods to learn example-specific local distance functions [8]. In particular, Xing et al. learn a Mahalanobis metric for k-means clustering by using semidefinite programming to minimize the sum of squared distances between similarly labeled examples, while requiring a certain lower bound on the distances between examples with different labels [10]. In related techniques, Globerson and Roweis [14] constrain within-class distances to be zero and maximize between-class distances, Weinberger et al. formulate the problem in a large-margin k-nearest-neighbors (k-NN) setting [13], while Goldberger et al. maximize a stochastic variant of the leave-one-out k-NN score on the training set [21]. In addition to using labeled data, research has shown how metric learning can proceed with weaker supervisory information, such as equivalence constraints or relative constraints. For example, equivalence constraints are exploited in the Relevant Component Analysis method of Bar-Hillel et al. [11]; the Dimensionality Reduction by Learning an Invariant Mapping (DrLIM) method developed by Hadsell et al. [22] learns a global nonlinear mapping of the input data; the Support Vector Machine-based approach of Schultz and Joachims [23] incorporates relative constraints over triples of examples and is extended by Frome et al. to learn example-specific local distance functions [24]. Davis et al. develop an information-theoretic approach that accommodates any linear constraint on pairs of examples and provide an efficient optimization solution that forgoes expensive eigenvalue decomposition [15].

To compute the optimal weighted combination of a collection of kernels or kernel matrices given some training labels, Lanckriet et al. propose a semidefinite programming solution [19], while Crammer et al. [25] and Hertz et al. [26] develop boosting-based approaches. Multiple kernel learning has also received attention in the vision community recently, with work showing the value of combining multiple types of image features with appropriate weights, whether through kernel alignment [27], [20] or cross validation [28].

Embedding functions can be useful both to capture (as closely as possible) a desired set of provided distances between points, as well as to provide an efficient approximation for a known but computationally expensive distance function of interest [29], [9]. Multidimensional scaling [30], Locally Linear Embeddings [31], and IsoMap [32] provide ways to capture known distances in a low-dimensional space, and provably low-distortion geometric embeddings have also been explored (e.g., [33]). The BoostMap approach of Athitsos et al. learns efficient euclidean-space embeddings that preserve proximity as dictated by useful but expensive distance measures [29]. More recently, Torralba et al. have shown that several learning methods can produce effective low-dimensional embedding functions for image comparisons using the global Gist image feature [34].

In order to efficiently index multidimensional data, data structures based on spatial partitioning and recursive hyperplane decomposition have been developed, e.g., $k-d$-trees [16] and metric trees [17]. Due to the particular importance of indexing local patch features, several tree-based strategies have also been proposed [35], [36] in the vision community. Some such data structures support the use of arbitrary metrics. However, while their expected query-time requirement may be logarithmic in the database size, selecting useful partitions can be expensive and requires good heuristics; worse, in high-dimensional spaces, all exact search methods are known to provide little improvement over a naive linear scan [37].

As such, researchers have considered the problem of *approximate* similarity search, where a user is afforded explicit trade-offs between the guaranteed accuracy versus speed of a search. Several randomized approximate search algorithms have been developed that allow high-dimensional data to be searched in time sublinear in the size of the database. Notably, Indyk and Motwani [37] and Charikar [38] proposed locality-sensitive hashing (LSH) techniques to index examples in Hamming space in sublinear time, and Datar et al. extended LSH for L_p norms in [39].

Data-dependent variants of LSH have been proposed: Georgescu et al. select partitions based on where data points are concentrated [40], while Shakhnarovich et al. use boosting to select feature dimensions that are most indicative of similarity in the parameter space [5]. This tunes the hash functions according to the estimation problem of interest; however, indexed examples are sorted according to the input space (nonlearned) distance. Embedding functions that map a specialized distance into a generic space (e.g., euclidean) have also been developed to exploit fast approximate search methods for particular metrics of interest, such as a bijective match distance or partial match kernel [41], [42]; however, such embeddings cannot accommodate constrained distances and are not instances of learned metrics. Muja and Lowe [43] empirically compare several of the above discussed methods for fast approximate nearest neighbor search.

We address the problem of sublinear time approximate similarity search for a class of *learned* metrics. While randomized algorithms such as LSH have been employed extensively to mitigate the time complexity of identifying

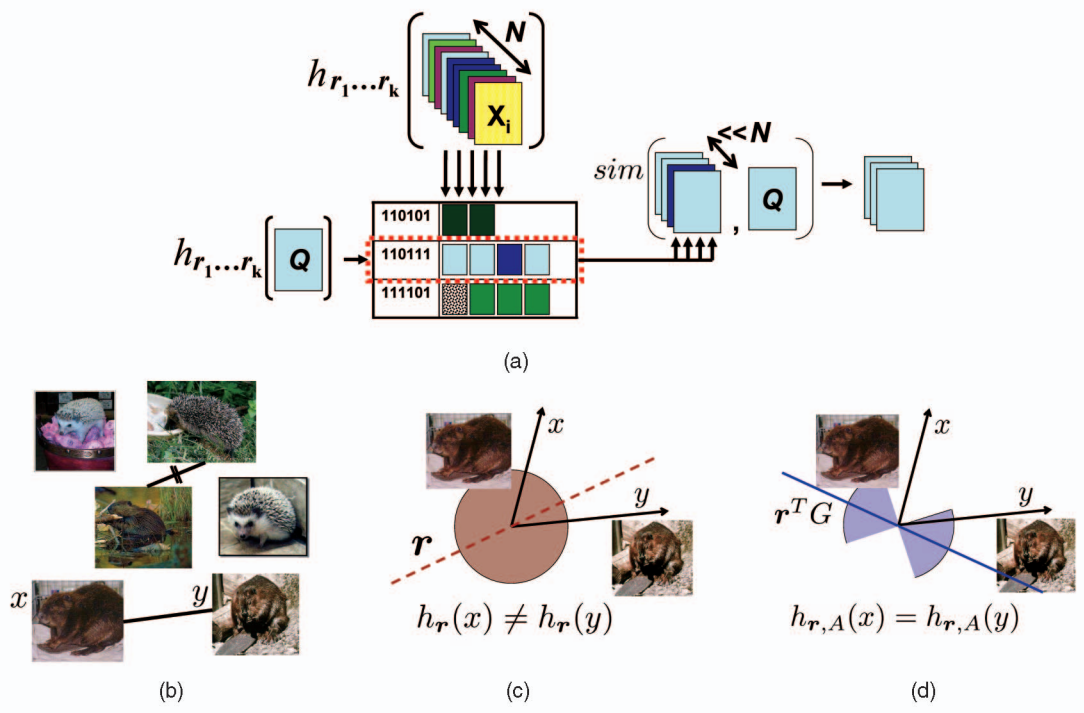

Fig. 1. (a) Overview of locality-sensitive hashing. A list of k hash functions $h_{r_1}, \ldots, h_{r_k}$ is applied to map N database images to a hash table, where similar items are likely to share a bucket. After hashing a query $\mathbf{Q}$, one must only evaluate the similarity between $\mathbf{Q}$ and the database examples with which it collides to obtain the approximate near-neighbors [37], [38]. (b) When learning a metric, some paired constraints can be obtained for a portion of the image database, specifying some examples that ought to be treated as similar (straight line) or dissimilar (crossed out line). (c) Whereas existing randomized LSH functions hash examples similar under the original distance together, (d) our semi-supervised hash functions incorporate the learned constraints so that examples constrained to be similar—or other pairs like them—will with high probability hash together. The circular red region in (c) denotes that the existing LSH functions generate a hyperplane uniformly at random to separate images. In contrast, as indicated by the blue "hourglass" region in (d), our hash functions bias the selection of the random hyperplane to reflect the specified (dis)similarity constraints. In this example, even though the measured angle between x and y is wide, our semi-supervised hash functions are unlikely to split them into different buckets since the constraints indicate that they should be treated as similar.

similar examples, particularly in vision [44], their use has been restricted to generic measures for which the appropriate hash functions are already defined; that is, direct application to learned metrics was not possible. We instead devise a method that allows knowledge attained from partially labeled data or paired constraints to be incorporated into the hash functions (see Fig. 1). Our algorithm is theoretically sound: There is provably no additional loss in accuracy relative to the learned metric beyond the quantifiable loss induced by the approximate search technique. In addition, the proposed method stands to benefit several existing methods for metric learning, since much research has been devoted to the class of Mahalanobis metrics in particular. This paper expands upon our previous conference publication [18].

3 Approach

The main idea of our approach is to learn a parameterization of a Mahalanobis metric based on provided labels or paired constraints for some training examples, while simultaneously encoding the learned information into randomized hash functions. These functions will guarantee that the more similar inputs are under the learned metric, the more likely they are to collide in a hash table. After constructing hash tables containing all of the database examples, those examples similar to a new instance are found in sublinear time in the size of the database by evaluating the learned metric between the new example and any example with which it shares a hash bucket.

3.1 Parameterized Mahalanobis Metrics

Given n points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, with all $\boldsymbol{x}_i \in \Re^d$, we wish to compute a positive-definite (p.d.) $d \times d$ matrix A to parameterize the squared Mahalanobis distance:

$$d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i - \boldsymbol{x}_j)^T A (\boldsymbol{x}_i - \boldsymbol{x}_j), \tag{1}$$

for all $i, j = 1, \ldots, n$. Note that a generalized inner product (kernel) measures the pairwise similarity associated with that distance:

$$s_A(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T A \boldsymbol{x}_j. \tag{2}$$

The Mahalanobis distance is often used with A as the inverse of the sample covariance when data are assumed to be Gaussian, or with A as the identity matrix if the squared euclidean distance is suitable. In general, the Mahalanobis distance may be viewed as measuring the squared euclidean distance after applying a linear transformation (if $A = G^T G$, then the Mahalanobis distance is equivalently $(G\boldsymbol{x}_i - G\boldsymbol{x}_j)^T (G\boldsymbol{x}_i - G\boldsymbol{x}_j)$). Given a set of interpoint distance constraints, one can directly learn a matrix A to yield a measure that is more accurate for a given classification or clustering problem. Many methods have been proposed for

Mahalanobis metric learning [10], [13], [14], [11], [15]; we consider the information-theoretic metric learning method of [15] because it is kernelizable and, hence, can efficiently handle data with high-dimensional feature spaces. Since below we will derive a new algorithm to systematically update semi-supervised hash functions in concert with this metric learner, we next briefly overview the necessary background and equations from [15].

3.2 Information-Theoretic Metric Learning

Given an initial $d \times d$ p.d. matrix A_0 specifying prior knowledge about interpoint distances, the learning task is posed as an optimization problem that minimizes the LogDet loss between matrices A and A_0, subject to a set of constraints specifying pairs of examples that are similar or dissimilar. In semi-supervised multiclass settings, the constraints are taken directly from the provided labels: points in the same class are constrained to be similar, and points in different classes are constrained to be dissimilar.

To compute A, the LogDet loss $D_{\ell\mathrm{d}}(A, A_0)$ is minimized while enforcing the desired constraints:

$$\begin{aligned} \min_{A \succeq 0} \quad & D_{\ell\mathrm{d}}(A, A_0) \\ \text{s.t.} \quad & d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq u \quad (i,j) \in \mathcal{S}, \\ & d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq \ell \quad (i,j) \in \mathcal{D}, \end{aligned} \tag{3}$$

where $D_{\ell\mathrm{d}}(A, A_0) = \mathrm{tr}(AA_0^{-1}) - \log\det(AA_0^{-1}) - d$, d is the dimensionality of the data points, $d_A(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the Mahalanobis distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ as defined in (1), $\mathcal{S}$ and $\mathcal{D}$ are sets containing pairs of points constrained to be similar and dissimilar, respectively, and ℓ and u are large and small values, respectively (and will be defined below).[1] The optimization is "information-theoretic" in that it corresponds to minimizing the relative entropy between the associated Gaussians whose covariance matrices are parameterized according to A and A_0.

The LogDet loss is amenable for metric learning in part because there is a very simple algorithm for optimizing (3), which involves repeatedly projecting the current solution onto a single constraint, via the update [15]:

$$\begin{aligned} A_{t+1} &= A_t + \beta_t A_t (\boldsymbol{x}_{i_t} - \boldsymbol{x}_{j_t})(\boldsymbol{x}_{i_t} - \boldsymbol{x}_{j_t})^T A_t, \\ \alpha_t &= \begin{cases} \min\left(\lambda_{i_t j_t}, \dfrac{1}{d_{A_t}(\boldsymbol{x}_{i_t}, \boldsymbol{x}_{j_t})} - \dfrac{1}{u}\right) & \text{if } (i_t, j_t) \in \mathcal{S}, \\ \min\left(\lambda_{i_t j_t}, \dfrac{1}{l} - \dfrac{1}{d_{A_t}(\boldsymbol{x}_{i_t}, \boldsymbol{x}_{j_t})}\right) & \text{if } (i_t, j_t) \in \mathcal{D}, \end{cases} \\ \lambda_{i_t j_t} &= \lambda_{i_t j_t} - \alpha_t, \quad \beta_t = \frac{\alpha_t}{1 - \alpha_t d_{A_t}(\boldsymbol{x}_{i_t}, \boldsymbol{x}_{j_t})}, \end{aligned} \tag{4}$$

where $\boldsymbol{x}_{i_t}$ and $\boldsymbol{x}_{j_t}$ are the constrained data points for iteration t, $\lambda_{i_t j_t}$ is the corresponding dual variable, and β_t is a projection parameter computed by the algorithm. Thus, we begin at time step 0 with the provided initial A_0 as our Mahalanobis matrix, and at each time step, we choose a single constraint and project onto that constraint. Under the assumption that A_0 is a full rank matrix and there is a feasible solution, the algorithm will converge to the optimal solution A. Note that the cost of a projection is $O(d^2)$ and the projection parameter β_t is computed in closed form. See [15] for further details.

Another advantage of using the LogDet loss for metric learning, and the primary reason we employ it in this work, is that one can efficiently *kernelize* the algorithm. When the dimensionality of the data is very high, one cannot explicitly work with A, and so the update in (4) cannot be performed. However, one may still implicitly update the Mahalanobis matrix A via updates in kernel space for an equivalent kernel learning problem in which $K = X^T A X$, for $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$. If K_0 is the input kernel matrix for the data ($K_0 = X^T A_0 X$), then the appropriate update is

$$K_{t+1} = K_t + \beta_t K_t (\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T K_t, \tag{5}$$

where the vectors $\boldsymbol{e}_{i_t}$ and $\boldsymbol{e}_{j_t}$ refer to the i_tth and j_tth standard basis vectors, respectively, and the projection parameter β_t is the same as in the last equation of (4) (see [15]). This update is derived by multiplying (4) on the left by X^T and on the right by X. If $A_0 = I$, then the initial kernel matrix is $K_0 = X^T X$; this may be formed using any valid kernel function, and the result of the algorithm is to learn a distance metric on top of this input kernel.

For low-dimensional data, once the optimal Malahanobis metric A is learned, generalizing to new points is easy—we simply apply the definition of the Mahalanobis distance directly. In the kernel space case, the optimal A has the form (Davis et al. [15, Theorem 1]):

$$A = I + XMX^T, \tag{6}$$

where $M = K_0^{-1}(K - K_0)K_0^{-1}$ and K is the optimal learned kernel matrix. By expanding $(\boldsymbol{x}_a - \boldsymbol{x}_b)^T A (\boldsymbol{x}_a - \boldsymbol{x}_b)$ using (6), the learned distances between *any* pair of points $(\boldsymbol{x}_a, \boldsymbol{x}_b)$ can be computed using only inner products of the data vectors, and hence, can be applied in kernel space:

$$\begin{aligned} &(\boldsymbol{x}_a - \boldsymbol{x}_b)^T A (\boldsymbol{x}_a - \boldsymbol{x}_b) \\ &\quad = (\boldsymbol{x}_a - \boldsymbol{x}_b)^T (I + XMX^T)(\boldsymbol{x}_a - \boldsymbol{x}_b) \\ &\quad = (\boldsymbol{x}_a - \boldsymbol{x}_b)^T \left(I + XK_0^{-1}(K - K_0)K_0^{-1}X^T\right)(\boldsymbol{x}_a - \boldsymbol{x}_b). \end{aligned}$$

By performing the updates in kernel space, the storage requirements change from $O(d^2)$ to $O(n^2)$. In many cases, both the dimensionality *and* the number of data points are large, and so both updates are infeasible. However, as given above, the learned distances are linear in K and, as a result, we can constrain distances between *any* pair of points. This makes it possible to limit the size of our base kernel matrix: We choose a *small* set of c *basis points*, form K_0 over these basis points and then constrain distances between any pair of points in terms of the initial kernel function $K_0(\boldsymbol{x}_a, \boldsymbol{x}_b)$. Because the learned distances are still linear in K, the resulting optimization problem may still be efficiently solved, allowing us to scale to very large data sets with very high dimensionality. While the kernelization approach was given in [15], the use of constraints outside of the input kernel matrix (i.e., the one defined over the basis points) has not previously been explored.

In the next section, we show how to constrain the distribution of randomized hash functions according to a learned parameterization, in the event that A_t can be

1. Note that, alternatively, the constraints may also be specified in terms of relative distances, i.e., $d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) < d_A(\boldsymbol{x}_i, \boldsymbol{x}_k)$.

manipulated directly. Then we derive an implicit formulation that enables information-theoretic learning with high-dimensional inputs for which A_t cannot explicitly be represented.

3.3 Hashing for Semi-Supervised Similarity Search

A family of locality-sensitive hash (LSH) functions $\mathcal{F}$ is a distribution of functions where the following holds: For any two objects $\boldsymbol{x}$ and $\boldsymbol{y}$,

$$\Pr_{h\in\mathcal{F}}[h(\boldsymbol{x}) = h(\boldsymbol{y})] = sim(\boldsymbol{x}, \boldsymbol{y}), \tag{7}$$

where $sim(\boldsymbol{x}, \mathbf{y})$ is some similarity function defined on the collection of objects, and $h(\boldsymbol{x})$ is a hash function drawn from $\mathcal{F}$ that returns a single bit [38]. Note that there is a related definition of LSH functions proposed in [37], however, in this paper, we use the one proposed in [38] only. Concatenating a series of b hash functions drawn from $\mathcal{F}$ yields b-dimensional hash keys (bit strings). When $h(\boldsymbol{x}) = h(\boldsymbol{y})$, $\boldsymbol{x}$ and $\boldsymbol{y}$ collide in the hash table. Because the probability that two inputs collide is equal to the similarity between them, highly similar objects are indexed together in the hash table with high probability. On the other hand, if two objects are very dissimilar, they are unlikely to share a hash key. Note that (7) is essentially the "gateway" to locality-sensitive hashing: If one can provide a distribution of hash functions guaranteed to preserve this equality *for the similarity function of interest*, then approximate nearest neighbor indexing may be performed in sublinear time. Existing LSH functions can accommodate the Hamming distance [37], L_p norms [39], and inner products [38], and such functions have been explored previously in the vision community [44], [5], [42].

In the following, we present new algorithms to construct LSH functions for learned metrics. Specifically, we introduce a family of hash functions that accommodates learned Mahalanobis distances, where we want to retrieve examples $\boldsymbol{x}_i$ for an input $\boldsymbol{x}_q$ for which the value $d_A(\boldsymbol{x}_i, \boldsymbol{x}_q)$ resulting from (1) is small, or, in terms of the kernel form, for which the value of $s_A(\boldsymbol{x}_i, \boldsymbol{x}_q) = \boldsymbol{x}_q^T A \boldsymbol{x}_i$ is high.

3.4 Explicit Formulation

For the explicit case, we show how to adapt the randomized hyperplane hashing approach of [38], which further follows from results in [45]. In the process of designing a randomized algorithm for the MAX-CUT problem, Goemans and Williamson demonstrated the following: Given a collection of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ on the unit sphere, and a randomly generated vector $\boldsymbol{r}$, the following relationship holds:

$$\Pr\big[\text{sign}\big(\boldsymbol{x}_i^T \boldsymbol{r}\big) \neq \text{sign}\big(\boldsymbol{x}_j^T \boldsymbol{r}\big)\big] = \frac{1}{\pi}\cos^{-1}\big(\boldsymbol{x}_i^T \boldsymbol{x}_j\big).$$

In other words, the sign of a vector's inner product with a random hyperplane will likely be the same for vectors having a small angle between them. In [38], Charikar uses this result to design hash functions for the case where the inner product is the similarity function of interest. In particular, the following hash function is locality-sensitive when sim is the dot product:

$$h_{\boldsymbol{r}}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T \boldsymbol{x} \geq 0, \\ 0, & \text{otherwise}, \end{cases} \tag{8}$$

$$\Pr[h_{\boldsymbol{r}}(\boldsymbol{x_i}) = h_{\boldsymbol{r}}(\boldsymbol{x_j})] = 1 - \frac{1}{\pi}\cos^{-1}\big(\boldsymbol{x}_i^T \boldsymbol{x}_j\big), \tag{9}$$

where again $\boldsymbol{r}$ is a random hyperplane of the same dimensionality as input $\boldsymbol{x}$.

We extend this to accommodate learned Mahalanobis distances. Given the matrix A for a metric learned as above,[2] such that $A = G^T G$, we generate the following randomized hash functions $h_{\boldsymbol{r},A}$, which accept an input point and return a hash key bit:

$$h_{\boldsymbol{r},A}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T G \boldsymbol{x} \geq 0, \\ 0, & \text{otherwise}, \end{cases} \tag{10}$$

where the vector $\boldsymbol{r}$ is chosen at random from a d-dimensional Gaussian distribution with zero mean and unit variance. Thus, by parameterizing the hash functions instead by G (which is computable since A is p.d.), we obtain the following relationship:

$$\Pr\big[h_{\boldsymbol{r},A}(\boldsymbol{x}_i) = h_{\boldsymbol{r},A}(\boldsymbol{x}_j)\big] = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{\boldsymbol{x}_i^T A \boldsymbol{x}_j}{\sqrt{|G\boldsymbol{x}_i||G\boldsymbol{x}_j|}}\right),$$

which sustains the LSH requirement of (7) for a learned Mahalanobis metric, whether A is computed using the method of [15] or otherwise [10], [13], [11], [14]. Essentially, we have shifted the random hyperplane $\boldsymbol{r}$ according to A and, by factoring it by G, we allow the random hash function itself to "carry" the information about the learned metric. The denominator in the cosine term normalizes the learned kernel values.

In this case, we could equivalently transform all the data according to A *prior* to hashing; however, the choice of presentation here helps set up the more complex formulation we derive below. Note that (10) requires that the input dimension d be low enough that the $d \times d$ matrix A can be explicitly handled in memory, allowing the updates in (4).

3.5 Implicit Formulation

We are particularly interested in the case where the dimensionality d may be very high—say, on the order of 10^4 to 10^6—but the examples are sparse and therefore can be stored efficiently (e.g., bags of words or histogram pyramids [4], [9]). Even though the examples are each sparse, *the matrix A can be dense*, with values for each dimension. In this case, the kernelized metric learning updates in (5) are necessary. However, this complicates the computation of hash functions as they can no longer be computed directly as in (10) above. Thus, we next derive a new algorithm to make simultaneous implicit updates to both the hash functions and the metric. The idea is to use the same hash functions as in (10), but to express G in a form that is amenable to computing the hash bit over high-dimensional input data.

We denote high-dimensional inputs by $\phi(\boldsymbol{x})$ to mark their distinction from the dense inputs $\boldsymbol{x}$ handled earlier. We are initially given c examples that participate in similarity or dissimilarity constraints. Let $\Phi = [\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_c)]$ be the $d \times c$ matrix of those initial c data points, and let $\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ be the initial (nonlearned) kernel value between example $\boldsymbol{x}_i$ and the input $\boldsymbol{x}_j$. Initially, $K_0 = \Phi^T \Phi$, and so, implicitly,

2. A variable without a subscript t denotes its value after convergence.

$A_0 = I$. As in the explicit formulation above, the goal is to wrap G into the hash function, i.e., to compute $\boldsymbol{r}^T G\phi(\boldsymbol{x})$, but now we must do so without working directly with G.

In the following, we will show that an appropriate hash function $h_{\boldsymbol{r},A}$ for inputs $\phi(\boldsymbol{x})$ can be defined as:

$$h_{\boldsymbol{r},A}(\phi(\boldsymbol{x})) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T\phi(\boldsymbol{x}) + \sum_{i=1}^{c} \gamma_i^r \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}) \geq 0, \\ 0, & \text{otherwise,} \end{cases} \tag{11}$$

where $\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x})$ is the original kernel value between a constrained point $\boldsymbol{x}_i$ and the query $\boldsymbol{x}$, and γ_i^r are coefficients computed once (offline) during metric learning (and will be defined below). Note that, while G is dense and therefore not manageable, computing $\boldsymbol{r}^T\phi(\boldsymbol{x})$ is computationally inexpensive as only the entries of $\boldsymbol{r}$ corresponding to nonzero entries in $\phi(\boldsymbol{x})$ need to be generated. Should the inputs be high-dimensional but dense, our implicit form is still valuable, as we bypass computing $O(d^2)$ products with G and require only $O(d)$ inner products for $\boldsymbol{r}^T\phi(\boldsymbol{x})$.

Next, we present a construction to express G in terms of the initially chosen c data points, and thus, a method to compute (11) efficiently. Our construction relies on two technical lemmas, which are given in the Appendix of this paper.

Recall the update rule for A from (4): $A_{t+1} = A_t + \beta_t A_t \boldsymbol{v}_t \boldsymbol{v}_t^T A_t$, where $\boldsymbol{v}_t = \phi(\boldsymbol{y}_t) - \phi(\boldsymbol{z}_t)$, if points $\boldsymbol{y}_t$ and $\boldsymbol{z}_t$ are involved in the constraint under consideration at iteration t. We emphasize that just as this update must be implemented implicitly via (5), so too must we derive an *implicit* update for the G_t matrix required by our hash functions.

Since A_t is p.d., we can factorize it as $A_t = G_t^T G_t$, which allows us to rewrite the update as

$$A_{t+1} = G_t^T\big(I + \beta_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T\big)G_t.$$

As a result, if we factorize $I + \beta_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T$, we can derive an update for G_{t+1}:

$$\begin{aligned} G_{t+1} &= \big(I + \beta_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T\big)^{1/2} G_t \\ &= \big(I + \alpha_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T\big)G_t, \end{aligned} \tag{12}$$

where the second equality follows from Lemma 1 using $\boldsymbol{y} = G_t\boldsymbol{v}_t$, and α_t is defined accordingly (see the Appendix).

Using (12) and Lemma 2, G_t can be expressed as $G_t = I + \Phi S_t \Phi^T$, where S_t is a $c \times c$ matrix of coefficients that determines the contribution of each of the c points to G. Initially, S_0 is set to be the zero matrix, and from there, every S_{t+1} is iteratively updated in $O(c^2)$ time via

$$\begin{aligned} S_{t+1} = S_t + \alpha_t (I + S_t K_0)(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T (I + K_0 S_t^T) \\ (I + K_0 S_t). \end{aligned}$$

Using this result, at convergence of the metric learning algorithm, we can compute $G\phi(\boldsymbol{x})$ in terms of the c^2 input pairs $(\phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j))$ as follows:

$$\begin{aligned} G\phi(\boldsymbol{x}) &= \phi(\boldsymbol{x}) + \Phi S \Phi^T \phi(\boldsymbol{x}) \\ &= \phi(\boldsymbol{x}) + \sum_{i=1}^{c}\sum_{j=1}^{c} S_{ij}\phi(\boldsymbol{x}_j)\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}). \end{aligned}$$

Therefore, we have

$$\begin{aligned} \boldsymbol{r}^T G\phi(\boldsymbol{x}) &= \boldsymbol{r}^T\phi(\boldsymbol{x}) + \sum_{i=1}^{c}\sum_{j=1}^{c} S_{ij}\boldsymbol{r}^T\phi(\boldsymbol{x}_j)\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}) \\ &= \boldsymbol{r}^T\phi(\boldsymbol{x}) + \sum_{i=1}^{c} \gamma_i^r \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}), \end{aligned}$$

where $\gamma_i^r = \sum_j S_{ij}\boldsymbol{r}^T\phi(\boldsymbol{x}_j)$ and is a notation substitution for the first equality. This notation reflects that the values of each γ_i^r rely only on the basis points, and thus can efficiently be computed in the training phase, prior to hashing anything into the database. Critically, the number of terms in this sum is the number of basis constrained points c—not the total number of constraints used during metric learning, nor the number of total database points. Finally, having determined the expression for $\boldsymbol{r}^T G\phi(\boldsymbol{x})$, we arrive at our hash function definition in (11). Note the analogy between the use of $\boldsymbol{r}^T G\boldsymbol{x}$ and $\boldsymbol{r}^T G\phi(\boldsymbol{x})$ in (10) and (11), respectively. Also note that our approach avoids computing the eigenvalue/Cholesky decomposition of the high-dimensional matrix A_t and enables efficient updates to G_t.

This section entails one of the two main technical contributions of this work: explicit and implicit methods to construct semi-supervised hash functions. We emphasize that our formulation is theoretically sound and in itself is novel; what is accomplished would not be possible with simple merging of the metrics in [15] with LSH.

3.6 Searching Hashed Examples

Having defined an algorithm for drawing locality-sensitive hash functions for learned metrics, we can apply existing methods [37], [38] to perform sublinear time approximate similarity search. Given N database points and an input query $\boldsymbol{x}_q$, approximate near-neighbor techniques guarantee retrieval of example(s) within the radius $(1+\epsilon)D$ from $\boldsymbol{x}_q$ in $O(N^{1/(1+\epsilon)})$ time, where the true nearest neighbor is at a distance of D from $\boldsymbol{x}_q$.

To generate a b-bit hash key for every example, we select b random vectors $[\boldsymbol{r}_1, \ldots, \boldsymbol{r}_b]$ to form b hash functions. The hash key for an input $\boldsymbol{x}$ is then the concatenation of the outputs of (10) (or, similarly, the outputs of (11) for an input $\phi(\boldsymbol{x})$). The trade-off in the selection of b is as follows: Larger values will increase the accuracy of how well the keys themselves reflect the metric of interest, but will also increase computation time and can lead to too few collisions in the hash tables. On the other hand, if b is lower, hashing will be faster, but the key will only coarsely reflect our metric, and too many collisions may result.

The problem of indexing into the database with $\boldsymbol{x}_q$ is then reduced to hashing with these same b functions and retrieving items corresponding to database bit vectors having minimal distance to the query. A query hashes to certain buckets in the hash table, where it collides with some small portion of the stored examples. For this step, we employ the technique for approximate search in Hamming space developed by Charikar [38], which requires searching $O(N^{1/(1+\epsilon)})$ examples for the $k = 1$ approximate-NN. Given the list of 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 is indexed into each sorted order with a binary search, and the $2M$ nearest examples found this way are the approximate nearest neighbors. See [38] for details.

Having identified these nearest bit vectors, we then compute the actual learned kernel values between their associated original data points and the original query. The hashed neighbors are ranked according to these scores, and this ranked list is used for k-NN classification, clustering, etc., depending on the application at hand.

3.7 Sparse High-Dimensional Embedding Functions

In Section 3.5, we formulated a method to implicitly update the distribution from which hash functions are drawn alongside the kernelized metric learner. This formulation accommodates high-dimensional inputs, and is well-suited to a number of existing common representations, including sparse bag-of-words vectors (text or visual). In addition, in this section, we describe embedding functions that will enable kernel learning for two useful matching kernels. These embeddings make it possible to express the desired base kernel as a dot product, which can then be improved with the Mahalanobis-based metric learning. The first matching considered is the pyramid match kernel [9] and is a review of the embedding function presented by Grauman and Darrell in [42]. We include it here to help elucidate a result in which we apply our implicit formulation in Section 4. The second matching is the proximity distribution kernel [46], and the embedding function we provide is a novel contribution of this work.

For both embeddings, the raw inputs are unordered sets of points, where each set $\mathbf{X}_i$ contains some number m_i of d-dimensional points: $\mathbf{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{m_i}\}$, with $\boldsymbol{x}_j \in \Re^d$, for all $j = 1, \ldots, m_i$. (Note that this is a reuse of the variable d: The dimension of these points is unrelated to the dimensionality of the A matrices described above.) In image matching applications, for example, these points are typically local image features that each describes some region in an image (e.g., SIFT [1]), and each image yields one set of points. The two kernels for which we develop embedding functions offer two different ways to compare the point sets based on feature correspondences. In the following, we first define the embedding function for each of the two kernels and then describe how both embeddings relate to the randomized hash functions.

3.7.1 Pyramid Match Embedding

The pyramid match is a low-distortion approximation[3] for the least-cost partial correspondence between two sets of feature vectors and requires only linear time in the number of points per set to compute. The least-cost partial matching between sets of features is the one-to-one assignment mapping the smaller set to the larger one in such a way that the summed cost between matched points is minimized. The matching cost essentially indicates how well the parts in two sets correspond. The main idea of the pyramid match approximation is to decompose the feature space into a multiresolution hierarchy and then for each point set to collect histograms at each resolution. Intersecting two histograms representing two different point sets implicitly counts the number of point matches at each possible distance (bin resolution), and a weighted combination of these changing intersection counts provides the total matching score [9].

A point set $\mathbf{X}$ is first converted to a multiresolution histogram (pyramid):

$$\Psi(\mathbf{X}) = [H_0(\mathbf{X}), \ldots, H_{L-1}(\mathbf{X})], \tag{13}$$

where $L = \lceil \log_2 B \rceil$ refers to the number of pyramid levels, B is the feature value range, and $H_i(\mathbf{X})$ is a histogram vector formed over points in $\mathbf{X}$ using d-dimensional bins of side length 2^i. Nonuniformly shaped bins are also possible and may be formed by hierarchical clustering on a corpus of features [48]. The pyramids are represented sparsely, with up to only $m = |\mathbf{X}|$ nonzero entries per level.

The (unnormalized) pyramid match kernel (PMK) value for sets $\mathbf{Y}$ and $\mathbf{Z}$ is defined as

$$\tilde{K}_{PMK}(\mathbf{Y}, \mathbf{Z}) = w_{L-1}\mathcal{I}_{L-1} + \sum_{i=0}^{L-2}(w_i - w_{i+1})\mathcal{I}_i, \tag{14}$$

where $\mathcal{I}_i = \sum_j \min(H_i^{(j)}(\mathbf{Y}),\ H_i^{(j)}(\mathbf{Z}))$ is the intersection between the ith histogram in $\Psi(\mathbf{Y})$ and $\Psi(\mathbf{Z})$, respectively, and $H_i^{(j)}$ is the count in bin j of H_i. To measure matching similarity, the weights w_i are set to be inversely proportional to the size of the histogram bins at level i, with the constraint that $w_i \geq w_{i+1}$ (e.g., $w_i = \frac{1}{2^i}$ is a valid option); alternatively, the weights may be learned in a discriminative fashion via multiple kernel learning [27].

Since histogram intersection can be mapped to a dot product by representing the bin counts with a unary-style encoding [49] and since a weighted intersection value is equivalent to the intersection of weighted counts, one can map this kernel to a simple dot product. In [42], an embedding function is designed that maps each histogram pyramid to a single vector in such a way that the inner product between any two such mapped outputs will give the pyramid match kernel value between the original point sets.

Let $\mathcal{U}([wH])$ denote the following (padded) unary encoding of the histogram H weighted by w:

$$\mathcal{U}([wH]) = \left(\underbrace{\overbrace{1, \ldots, 1}^{wH^{(1)}}, \overbrace{0, \ldots, 0}^{P - wH^{(1)}}}_{\text{first bin}}, \ldots, \underbrace{\overbrace{1, \ldots, 1}^{wH^{(r)}}, \overbrace{0, \ldots, 0}^{P - wH^{(r)}}}_{\text{last bin}} \right), \tag{15}$$

where P is the maximum possible weighted count in any histogram bin, and $H^{(j)}$ is the count in bin j of H. If weighted counts are real-valued, this process can in theory proceed by scaling to a given precision and truncating to integers. With the normalization factor also scaled, the output remains equivalent. However, the unary encoding need not be explicitly computed in practice. Let $\mathbf{v}_i(\mathbf{X})$ refer to the histogram for set $\mathbf{X}$ at pyramid level i, weighted by $w = w_i - w_{i+1}$: $\mathbf{v}_i(\mathbf{X}) = [(w_i - w_{i+1})H_i(\mathbf{X})]$.

3. "Low-distortion" refers to the bounded error between the approximate match cost given by the pyramid match and the true optimal least-cost partial match. Error bounds for the PMK are shown in [47].

The following embedding f_{PMK} serves to map the set of vectors $\mathbf{X}$ to a single vector [42]:

$$f_{PMK}(\mathbf{X}) = [\mathcal{U}(\mathbf{v}_0(\mathbf{X})), \mathcal{U}(\mathbf{v}_1(\mathbf{X})), \mathcal{U}(\mathbf{v}_2(\mathbf{X})), \ldots, \mathcal{U}(\mathbf{v}_{L-2}(\mathbf{X})), \mathcal{U}([w_{L-1}H_{L-1}(\mathbf{X})])],$$

and the dot product between two such encodings for sets $\mathbf{Y}$ and $\mathbf{Z}$ yields the unnormalized pyramid match score from (14) above: $f_{PMK}(\mathbf{Y}) \cdot f_{PMK}(\mathbf{Z}) = \tilde{K}_{PMK}(\mathbf{Y}, \mathbf{Z})$.

The length $|f_{PMK}(\mathbf{Y})|$ of an encoding vector $f_{PMK}(\mathbf{Y})$ is simply the sum of its total number of nonzero (one) entries. Since self-intersection of a histogram returns the number of total points in the histogram ($\mathcal{I}(H(\mathbf{Y}), H(\mathbf{Y})) = |\mathbf{Y}|$), the length of an embedding vector will be equivalent to the original set's self-similarity score under the pyramid match:

$$\begin{aligned} |f_{PMK}(\mathbf{Y})| &= w_{L-1}|\mathbf{Y}| + \sum_{i=0}^{L-2}(w_i - w_{i+1})|\mathbf{Y}| \\ &= \tilde{K}_{PMK}(\mathbf{Y}, \mathbf{Y}). \end{aligned} \tag{16}$$

We can compute similar embeddings and hash functions for the "vocabulary-guided" pyramid match given in [48], or the spatial pyramid match defined in [50], since the intersected pyramids there too can be written as a dot product between weighted histograms. Because a vocabulary-guided pyramid uses irregularly shaped histogram bins, for that embedding, the weights must be applied at the level of the bins instead of at the level of the pyramid resolutions.

3.7.2 Proximity Distribution Embedding

Next we show how to embed a related kernel developed by Ling and Soatto in [46]. This kernel matches distributions of co-occurring local descriptor types as they appear at increasing distances from one another in an image. In this case, each feature point $\mathbf{x}_j$ is first mapped to one of k discrete "codewords" or "visual words," which are the prototypical local feature types identified via clustering on some previous corpus of descriptors. Each point set is converted to a $k \times k \times R$-dimensional histogram that counts the number of times each visual word co-occurs within the $r = 1, \ldots, R$ spatially nearest neighbor features of any other visual word. Specifically, for a given image input described with point set $\mathbf{X}$, each histogram entry $H^{\mathbf{X}}(i, j, r)$ is defined as the number of times visual word type j occurs within the r spatially nearest neighbors of a visual word of type i. Since $r = 1, \ldots, R$, this is a cumulative distribution of the co-occurring pairs of words.

The (unnormalized) proximity distribution kernel (PDK) value between two images with feature sets $\mathbf{Y}$ and $\mathbf{Z}$ is then

$$K_{PDK}(\mathbf{Y}, \mathbf{Z}) = \sum_{i=1}^{k}\sum_{j=1}^{k}\sum_{r=1}^{R} \min\big(H^{\mathbf{Y}}(i, j, r), H^{\mathbf{Z}}(i, j, r)\big), \tag{17}$$

where $H^{\mathbf{Y}}$ and $H^{\mathbf{Z}}$ are the associated arrays of histograms computed for the two input feature sets [46].

To write our embedding function, we simply need to flatten the three-dimensional array of cumulative histograms and write it as an implicit unary encoding. Let $v_x(r)$ refer to the co-occurrence counts at neighborhood rank r for a given word x against every other word: $v_x(r) = [H(x, 1, r), \ldots, H(x, k, r)]$ and let $v(r)$ refer to all co-occurrence counts at neighborhood rank r: $v(r) = [v_1(r), \ldots, v_k(r)]$. Then, using the same notation $\mathcal{U}$ as above, the following embedding function maps the proximity distribution kernel to an inner product:

$$\begin{aligned} &f_{PDK}(\mathbf{X}) = [\mathcal{U}(v(1)), \ldots, \mathcal{U}(v(R))] \text{ and} \\ &f_{PDK}(\mathbf{Y}) \cdot f_{PDK}(\mathbf{Z}) = K_{PDK}(\mathbf{Y}, \mathbf{Z}). \end{aligned} \tag{18}$$

3.7.3 Hashing with the Embedded Kernels

The embedding functions for these kernels allow us to perform sublinear time similarity search with random hyperplane hash functions, whether according to their "raw" definitions, or according to their learned variants. For both kernels' original definitions, we can now hash using the inner product LSH function from (10). Specifically, we have

$$\Pr[h_{\vec{r}}(f(\mathbf{Y})) = h_{\vec{r}}(f(\mathbf{Z}))] = 1 - \frac{\theta(f(\mathbf{Y}), f(\mathbf{Z}))}{\pi}, \text{ where}$$

$$\theta(f(\mathbf{Y}), f(\mathbf{Z})) = \cos^{-1}\left(\frac{f(\mathbf{Y}) \cdot f(\mathbf{Z})}{\sqrt{|f(\mathbf{Y})||f(\mathbf{Z})|}}\right) = \cos^{-1}(K(\mathbf{Y}, \mathbf{Z})),$$

where K and f refer to the appropriate version of K_{PMK} or K_{PDK} and f_{PMK} or f_{PDK}, respectively. The last term is the similarity value normalized according to the product of the self-similarity scores. For both kernels and their learned variants, we can now perform hashing using the implicitly updated hash function we defined in (11), with $\phi(\boldsymbol{x}) = f(\mathbf{X})$.

Even forgoing the explicit unary encoding, the dimensionality of the embedding space is very high, with an exponential dependence on the dimensionality of the feature points (in the uniform-bin PMK case), an exponential dependence on the number of levels (in the nonuniform bin PMK case), and with $d = k \times k \times R$ (in the PDK case). However, the histograms are very sparsely populated, and only nonzero entries in the embedding output need ever be touched when computing a dot product (or hash function output). Likewise, with our implicit learned kernel hashing, these correspondence-based measures will be improved in kernel space according to the provided paired constraints, and then hashing requires only dot products between a new input and some constrained inputs and the random hyperplanes (see Section 3.5).

4 Results

Our approach is in general applicable for content-based search of large databases according to learned Mahalanobis metrics. In this section, we provide results with a variety of data sets to illustrate its flexibility with respect to representations, constraints, and base metric/kernel functions. First, we demonstrate our algorithm in the low-dimensional setting applied to a nearest neighbor classification problem for software support. Then, we evaluate our algorithm in the implicit setting for image search in three distinct domains: exemplar-based recognition, pose estimation, and feature indexing. In all cases, our experimental goal is twofold: 1) to evaluate the impact on accuracy a learned metric has relative to both standard baseline metrics and state-of-the-art methods, and 2) to test how reliably our semi-supervised hash functions preserve the learned metrics in practice when

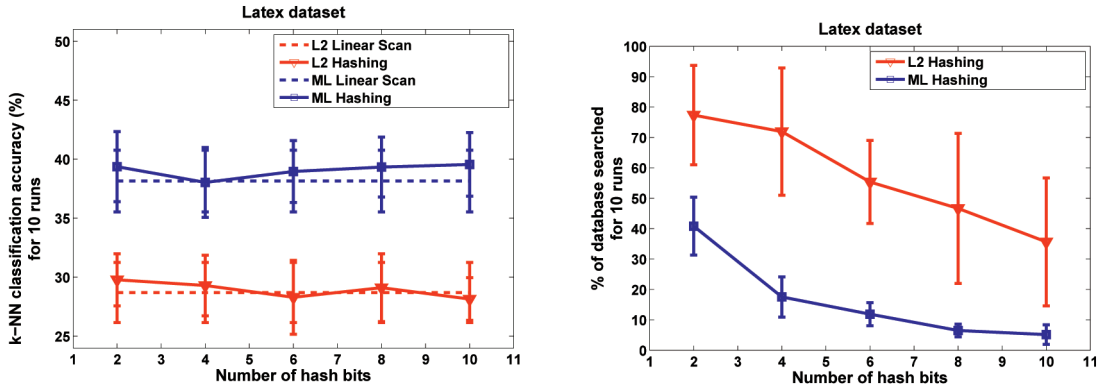

Fig. 2. Comparison of the accuracy (left) and time requirements (right) when hashing with the original L_2 and learned ("ML") metrics for the Latex data set. The left plot shows k-NN classification accuracy. The right plot shows the search time in terms of the percentage of database items searched per query, as a function of the number of hash bits b. Results are from 10 runs with random query/database partitions, with $\epsilon = 1.5$. Our learned hash functions reduce the search to about 5 percent of the database, with virtually no loss in accuracy over the exhaustive linear scan.

performing sublinear time database searches. We therefore report results in terms of both accuracy improvements as well as speedups realized.

Throughout, we select examples for (dis)similarity constraints randomly from a pool of examples. For categorical data, (dis)similarity constraints are associated with points having different (same) labels; for data with parameter vectors, constraints are determined based on examples' nearness in the parameter space. We compute the distance between all pairs of a subset ($\approx$100) of the database examples according to the nonlearned metric, and then let the distance constraints' lower ℓ and upper u limits be the first and 99th percentile of those values, respectively. We measure accuracy in terms of the error of the retrieved nearest neighbors' labels, which is either a parameter vector (in the case of the pose data) or a class label (in the case of the systems data, object images, and patches).

4.1 Software Support for Latex Compilation Errors

We first evaluate our method for learned metric hashing on the nearest neighbor (NN) classification problem using data from the CLARIFY system of Ha et al. [51]. CLARIFY assists a programmer in diagnosing errors by identifying previously seen abnormal termination reports with similar program features, and pointing the programmer to other users who have had similar problems. We experiment with a database of $N = 3{,}825$ such examples collected from the Latex typesetting program. The features are $d = 20$-dimensional, and so our explicit formulation for learning hash functions is most appropriate. Paired similarity constraints are generated using 20 labeled examples from each class. For 10 random partitions of the data, we extract 30 examples for each of its nine classes, and treat the remainder as database examples. We measure the $k = 4$ nearest neighbor classification accuracy and search times over all 270 queries per run, under four settings: the original euclidean distance metric and a linear scan, the original distance with LSH, the learned metric with a linear scan, and the learned metric with LSH. For both hashing cases, we fix $\epsilon = 1.5$. Recall that this parameter controls the trade-off between speed and accuracy for the approximate search, so a value of $\epsilon = 1.5$ means searching $2N^{(1/(1+\epsilon))} = 35$ examples in this case.

Fig. 2 shows the resulting accuracy and complexity gains. (Throughout, our approach is denoted by "ML.") By incorporating the paired constraints, the learned metric shows clear accuracy gains over the unconstrained euclidean distance, yielding about 10 percent higher correct classification rates. The k-NN rates for both associated hashed results are, on average, as good as the linear scan results, and in this case, have little dependence on the number of hash functions used. As b increases, however, the hash keys become more specific and allow larger amounts of the database to be ignored for any given query (right-hand plot). When searching only 5 percent of the database, our learned hash functions suffer no loss in accuracy yet enable an average $13x$ speedup (maximum speedup $34x$) relative to an exhaustive scan with the learned metric (including the overhead cost of computing the hash keys). Interestingly, for the same values of ϵ and b, the number of examples searched with the learned hash functions is noticeably lower than that of the generic hash functions and has a tighter distribution. While the indexing guarantees remain the same, we infer that, just as the learned metric adjusts the feature space so that in-class examples are more closely clustered, the learned hash functions better map them to distinct keys.

4.2 Human Body Pose Estimation

In this section, we demonstrate our method applied to single-frame human body pose estimation. Example-based techniques to infer pose (e.g., [6], [5]) store a large database of image examples that are labeled with their true pose (i.e., 3D joint positions or angles). A query image is indexed into the database according to image similarity and the query's pose is estimated based on the pose parameters attached to those nearest neighbors. Thus, our objective for this task is to learn a metric for the image features that reports small distances for examples that are close in pose space, and to make the search scalable by hashing according to a learned metric. This is similar to the goals of the parameter-sensitive hashing (PSH) method of [5]. However, our approach is distinct from [5] in that it allows one to seamlessly both hash and search according to the learned metric. As a result, it may provide more accurate retrievals, as we show

TABLE 1
Mean Pose Error (in Centimeter) Obtained with Each Method

Method	d	k=1	k=7	k=50
L_2 linear scan	24K	8.9	12.0	15.1
L_2 hashing	24K	9.4	12.8	15.6
PSH, linear scan	1.5K	9.4	12.2	15.9
PCA, linear scan	60	13.5	14.0	16.8
ML PCA, linear scan	60	13.1	13.8	16.2
ML linear scan	24K	8.4	11.5	14.1
ML hashing	24K	8.8	12.1	14.9

Our approach (denoted by ML) outperforms both the L_2 baseline as well as the PSH method of Shakhnarovich et al. [5].

empirically below. More generally, in contrast to [5], the proposed approach enables fast search for a class of metrics and kernels, which provides flexibility regarding both the user-selected base metric as well as the learning algorithm and types of constraints used to construct the metric.

We use a database of examples provided by the authors of [52], where PSH is employed within a pose tracker. The images were generated with Poser graphics software: Human figures in a variety of clothes are rendered in many realistic poses drawn from mocap data. Our main motivation for working with this data set is its sheer scale: It has half a million *labeled* examples, which means that we will be able to quantify the quality of any retrieval. Each image is represented by a $d = 24{,}016$-dimensional multiscale edge detection histogram (EDH). The vectors' high dimensionality requires our implicit formulation for semi-supervised hash functions. We use a linear kernel over $c = 50$ randomly selected examples as the initial kernel (K_0). We hold out 1,000 test queries examples and generate 1,000,000 similarity constraints among 50,000 of the remaining training examples. For each, we constrain the distance of the 10 nearest exemplars (in terms of pose parameters) to be less than ℓ. Similarly, of all the examples with a pose distance greater than a threshold t, 10 are randomly picked and their distance to the example is constrained to be greater than u. The values of t and c are selected with cross validation. Note that whereas the previous data set illustrated the use of label-based constraints, which the learned metric uses to map same-class examples close to one another, here we have examples "labeled" with real-valued parameter vectors, so the learned metric will adjust the feature space distances to be more like the desired parameter space distances.

As baselines, we compute results for NN search with both the euclidean distance (L_2) on the EDHs and the Hamming distance on the PSH embeddings provided by the authors of [52]. To hash with the L_2 baseline, we simply apply [38]. We also use principal components analysis (PCA) to reduce the dimensionality of the EDH vectors in order to apply our explicit formulation for comparison. We measure the error for a query by the mean distance of its true joint positions to the poses in the k-NN. To give a sense of the variety of the data, a random database example is on average at a distance of 34.5 cm from a query.

Table 1 shows the overall errors for each method. With a linear scan, ML yields the most accurate retrievals of all methods, and with hashing it outperforms all the hashing-based techniques. The PCA-based results are relatively poor, indicating the need to use the full high-dimensional features, and thus our implicit formulation. A paired-error T-test reveals that our improvements over PSH and L_2 are statistically significant, with 99.95 percent confidence.

Fig. 4 shows the NN retrieved by each method for five typical queries. In most examples, L_2 and PSH estimate the overall pose reasonably well, but suffer on one or more limbs, whereas our approach more precisely matches all limbs and yields a lower total error. While PSH does not improve over the L_2 baseline for this data set (as it did for data in [5]), it does do nearly as well as L_2 when using about $16x$ fewer dimensions; it appears its main advantage here is the ability to significantly reduce the dimensionality.

Our semi-supervised hash functions maintain the accuracy of the learned metric, but for substantially less search time than the linear scan. With our Matlab implementation, a linear scan requires 433.25 seconds per query, while our hashing technique requires just 1.39 seconds. On average, metric learning with hashing searches just 0.5 percent of the database. Fig. 3 compares the error obtained by ML+ hashing and L_2+ hashing when varying the number of hash bits (left plot) and the search time allowed (right plot). For a large number of bits, the hash keys are more precise, and hence the error drops (although hashing overhead increases).

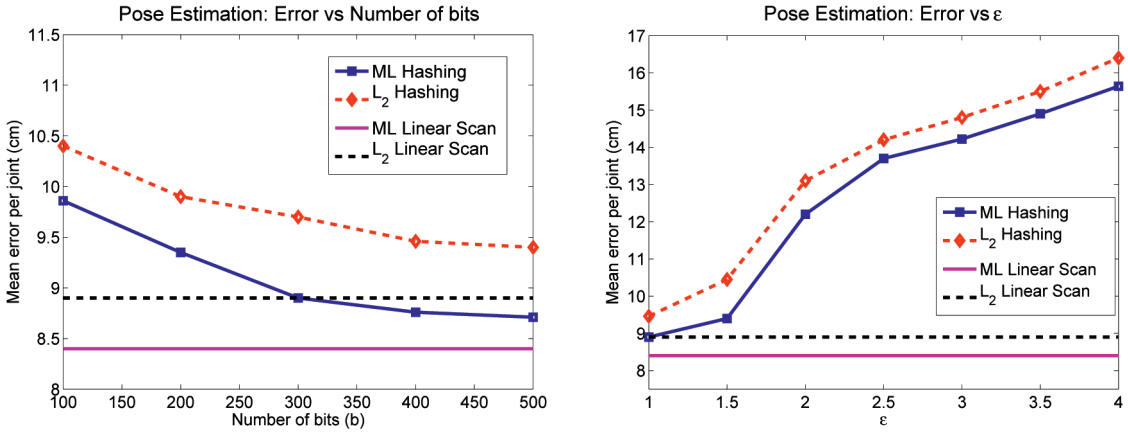

Fig. 3. **Left**: Error as a function of the number of hash bits. Fast search with the learned metric is more accurate than the L_2 baseline. For both, the error converges around $b = 500$ bits. **Right**: Hashing error relative to an exhaustive linear scan as a function of ϵ, which controls the search time required.

Fig. 4. Examples of pose estimates. Each column contains a different pose. The top row contains query images and the remaining rows show the best pose retrieved by each method. The second row shows the best pose obtained by our method (labeled ML-HASH). The coarse pose estimates obtained by each method are somewhat similar, but with the learned metric, our approach is able to more accurately obtain matches for the fine details of the joint positions.

Similarly, since $M = 2N^{1/(1+\epsilon)}$, for high values of ϵ we must search fewer examples, but accuracy guarantees decrease.

4.3 Exemplar-Based Object and Scene Categorization

In this section, we evaluate our method applied for exemplar-based object and scene recognition with the Caltech-101, a common benchmark, and a data set of scene images downloaded from Flickr. The goal is to predict the object or scene class of a test example by finding the most visually similar examples in the labeled database, and then allowing those neighbors to cast votes on the label. In this set of experiments, we demonstrate the flexibility of our approach relative to the choice of a base metric, with results using three different kernels defined in the vision literature [9], [46], [7].

4.3.1 Caltech-101 Database

To compare the Caltech-101 images, we consider learning kernels on top of Grauman and Darrell's PMK [9] applied to SIFT features, and the kernel designed by Zhang et al. [7] applied to geometric blur features. As described above, the PMK uses multiresolution histograms to estimate the correspondence between two sets of local image features. To hash with the nonlearned PMK, the pyramids are embedded as described in Section 3.7.1. The pyramid inputs are sparse but extremely high-dimensional ($d = O(10^6)$); thus, explicitly representing A is infeasible and the implicit form of our technique is necessary. The kernel in [7] also measures the correspondences between local features, but by averaging over the minimum distance to matching features in terms of the descriptors and their position in the image; we will refer to it as CORR. Note that we can learn kernels for both the PMK and CORR using the kernel learning formulation from [15], but can only hash with the learned PMK, since an explicit vector space representation ($\phi(\boldsymbol{x})$) for the CORR kernel is unknown.

We first evaluate the effectiveness of metric learning itself on this data set. We pose a k-NN classification task,

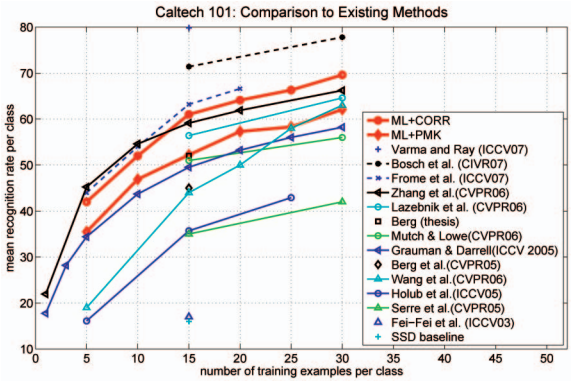

Fig. 5. Comparison against existing techniques on the Caltech-101. Our method outperforms all other single metric/kernel approaches. ML+PMK refers to our learned kernel when the pyramid match [9] is the base kernel; ML+CORR refers to our learned kernel when the correspondence kernel of [7] is the base kernel. The base kernel curves are the ones labeled Grauman & Darrell (ICCV '05) and Zhang et al. (CVPR '06), respectively, in the legend. (This figure is best viewed in color.)

and evaluate both the original (PMK or CORR) and learned kernels when used in a linear scan mode. We set $k = 1$ for our experiments; this value was chosen arbitrarily. We vary the number of training examples T per class for the database, using the remainder as test examples, and measure accuracy in terms of the mean recognition rate per class, as is standard practice for this data set.

Fig. 5 shows our results relative to all other existing techniques that have been applied to this data set. Our approach outperforms all existing single-kernel classifier methods when using the learned CORR kernel: We achieve 61.0 percent accuracy for $T = 15$ and 69.6 percent accuracy for $T = 30$. Our learned PMK achieves 52.2 percent accuracy for $T = 15$ and 62.1 percent accuracy for $T = 30$. Fig. 6 shows specifically the comparison of the original baseline kernels for NN classification. The plot on the left reveals gains in NN retrieval accuracy; notably, our learned kernels with simple NN classification also outperform the baseline kernels when used with SVMs [7], [9]. Only the results of recent multiple-metric approaches [8], [20], [28] (shown with dashed lines in Fig. 5) are more accurate, though they also incur the greater cost of applying each of the base kernels in sequence to all examples, while our method requires only one comparison to be computed per example.

Now we consider hashing over the learned PMK. For $T = 15$, our learned hash functions achieve 47 percent accuracy and require about $10x$ less computation time than a linear scan when accounting for the hash key computation (here, $N = 1{,}515$, which is modest compared to the pose data evaluated above). The right-hand plot in Fig. 6 shows the error of our learned PMK-based hashing compared to the baseline [42] as a function of ϵ. For these data, the value of b had little effect on accuracy. As with the linear scan search, we still realize significant accuracy improvements, but now with a guaranteed sublinear time search.

4.3.2 Flickr Scene Database

To evaluate our learned hash functions when the base kernel is the PDK, we performed experiments with a data set of 5,400 images of 18 different tourist attractions from the photo sharing site Flickr. We took three cities in Europe that have major tourist attractions: Rome, London, and Paris. The

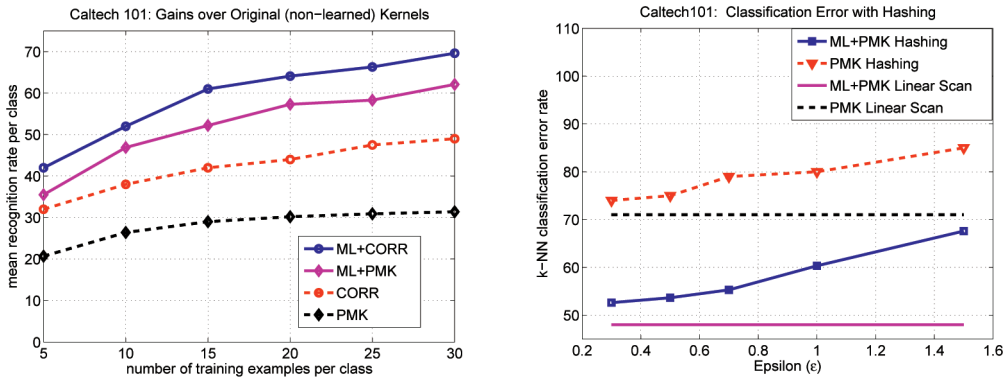

Fig. 6. Object recognition on the Caltech-101 data set. **Left**: Our learned kernels significantly improve NN recognition accuracy relative to their nonlearned counterparts, the CORR and PMK kernels. **Right**: Comparison of the k-NN classification error when hashing with the original and learned PMK. This plot shows the accuracy-search time trade-off when using the original or learned hashing functions. CORR refers to the correspondence kernel proposed by Zhang et al. [7], and PMK refers to the pyramid match kernel [9].

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. Overall, the list yielded 18 classes: 8 from Rome, 5 from London, and 5 from Paris. The classes are: 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 Westminster Abbey. We downloaded the first 300 images returned from each search query to represent the data for each class. Since not all images downloaded for a given tag actually contain the proper scene, we manually added ground truth labels. About 90 percent of the initial tags on the downloaded images were accurate.

Duplicate images and images that had no response from the interest point detectors were removed and then replaced with lower ranked images so that the number of images per category remained at 300. All images were scaled down to have moderate width (320 pixels).

Note that the regular viewpoints and scales in the Caltech-101 images above make it possible to improve the unordered set representation using simply image coordinate positions, which means that the PMK with features including spatial position are adequate. For the Flickr data, however, the viewpoint and scale vary significantly across instances of the same scene, so the loose configurations of features captured by the PDK provide a better way to preserve semi-local geometry without being overly restrictive. We detect corner and blob-like regions in the images using the Harris-affine [53] and Maximally Stable Extremal Regions (MSER) [54] detectors, and represent all regions with SIFT descriptors [1]. The PDK requires a codebook to quantize features; following [46], we set the number of visual words to be $k = 200$ and include neighbors up to rank $R = 64$. We use the embedding provided in Section 3.7.2 for hashing with PDK.

We again pose a nearest neighbor classification task, and compare results when using either a linear scan or hashing, with the original base PDK or the learned PDK. We randomly select 275 images per class for training and the remaining images for testing. Fig. 7 shows the results. The linear scan error for either hashing method shows the best possible performance, and again as we decrease the ϵ parameter, we can expect more accurate results at the cost of longer query times. The learned PDK achieves a significantly higher accuracy than the original base PDK while searching a smaller amount of the Flickr database.

4.4 Indexing Local Patch Descriptors

In this section, we evaluate our approach on a patch matching task using data provided from the Photo Tourism

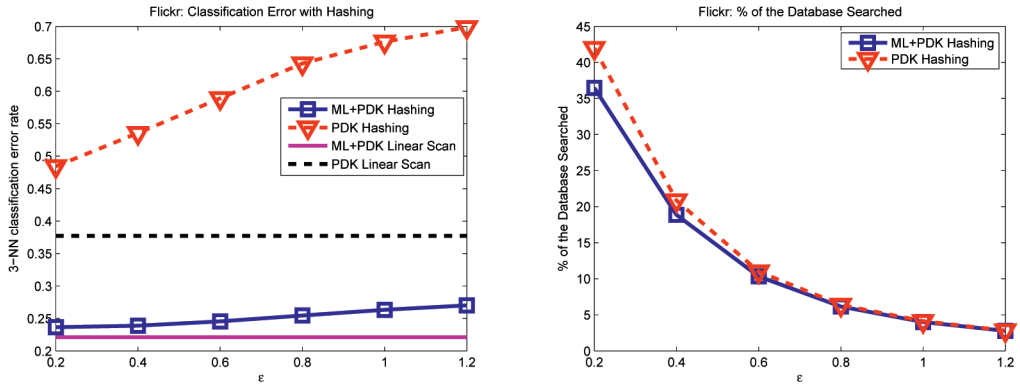

Fig. 7. Scene classification on the Flickr data set. **Left**: Hashing error relative to an exhaustive linear scan as a function of ϵ, which controls the search time required. **Right**: Amount of the database searched as a function of ϵ. Clearly, our method ML+PDK Hashing achieves significantly higher accuracy while searching a smaller portion of the database.

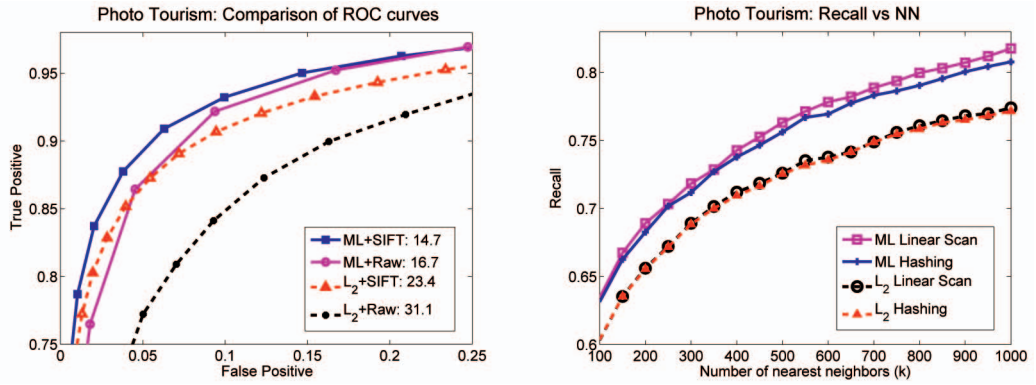

Fig. 8. **Left**: This plot illustrates accuracy improvements of the learned metric (ML) relative to L_2 baselines, for both raw patches (dimensionality $d = 4{,}096$) and SIFT descriptors (dimensionality $d = 128$). (Note that the axes are zoomed into the upper left-hand corner of the curves.) **Right**: This plot shows the recall as a function of the number of SIFT patches retrieved, for our method and the L_2 baseline. Our semi-supervised hash functions maintain accuracy close to a linear scan (as seen by the overlapping ML Linear Scan and ML Hashing curves), while requiring much less search time.

project [2], [55]. The data set contains about 300,000 local patches extracted from interest points in multiple users' photos of scenes from different viewpoints. The objective is to be able to rapidly identify any matching patches from the same 3D scene point in order to provide correspondences to a structure from motion algorithm. For this application, *classifying* patches is not so useful; rather, one wants to find all relevant patches. Thus, we measure accuracy in terms of precision and recall.

We add random jitter (scaling, rotations, and shifts) to all patches as prescribed in [55], extract both the raw patch intensities and SIFT descriptors, and then pose the retrieval task to the L_2 baseline and our learned metrics for each representation. To learn metrics, we gather constraints from 10,000 matching and nonmatching patch pairs, with a 50-50 mix taken from the Trevi and Halfdome portions of the data. All methods are tested on 100,000 pairs from the Notre Dame portion. The left plot in Fig. 8 compares their accuracy via ROC curves for each feature and metric combination; the numbers in the legend summarize the error in terms of the false positive rate once 95 percent of the true positives are retrieved. We see that $\mathrm{ML}{+}\mathrm{raw}$ intensities yields a significant gain over $L_2{+}\mathrm{raw}$, while ML+SIFT also gives some improvement.[4]

Finally, we consider our ML-hashing algorithm for the SIFT patches. We measure accuracy by the relevance of the NN ranking: for increasing values of k, we compute the recall rate within the top k-NN. We calculate this score with and without hashing, and before and after metric learning. In order to control k for the hashing, we consider as many nearby hash bins as necessary. In the right plot in Fig. 8, we see that the learned metric outperforms the L_2 baseline and hashing does not noticeably degrade accuracy. When $k = 1{,}000$, we search only 16.1 percent of the database when hashing over the learned metric and, when $k = 1$, we search only 0.8 percent, leading to substantial gains in retrieval time (about a factor of 80 versus linear scan).

4. In this experiment, we were able to reproduce the baseline for L_2 given in [55]; however, we were unable to do so for their SIFT baseline for which 6 percent error is obtained. We suspect this is due to our unoptimized SIFT extraction and that ML would continue to yield similar improvements as above if provided better descriptors.

5 Conclusions

We have introduced a method to enable efficient approximate similarity search for learned metrics, and experiments show good results for a variety of data sets, representations, and base metrics. Our main contribution is a new algorithm to construct theoretically sound locality-sensitive hash functions—for both implicit and explicit parameterizations of a Mahalanobis distance. For high-dimensional data, we derive simultaneous implicit updates for both the hash function and the learned metric. Experiments demonstrate our technique's accuracy and flexibility for a number of large-scale search tasks.

In future work, we intend to explore online extensions to our algorithm that will allow similarity constraints to be processed in an incremental fashion, while still allowing intermittent queries. We are also interested in considering generalizations of our implicit hashing formulation to accommodate alternative kernelized metric learning algorithms, and in pursuing active constraint selection methods within our framework.

Appendix

We prove the following lemmas to aid in our construction in Section 3.

Lemma 1. *Let $B = I + \beta \boldsymbol{y}\boldsymbol{y}^T$ be positive semidefinite. Then, $B^{1/2} = I + \alpha \boldsymbol{y}\boldsymbol{y}^T$, with $\alpha = (\pm\sqrt{1 + \boldsymbol{y}^T\boldsymbol{y}\beta} - 1)/\boldsymbol{y}^T\boldsymbol{y}$.*

Proof. Let $B^{1/2} = (I + \alpha \boldsymbol{y}\boldsymbol{y}^T)$. Thus, $B = (I + \alpha \boldsymbol{y}\boldsymbol{y}^T)^2$. Expanding yields $I + 2\alpha \boldsymbol{y}\boldsymbol{y}^T + \alpha^2(\boldsymbol{y}^T\boldsymbol{y})\boldsymbol{y}\boldsymbol{y}^T = I + (2\alpha + \alpha^2\boldsymbol{y}^T\boldsymbol{y})\boldsymbol{y}\boldsymbol{y}^T$. For the lemma to hold, we require that $(I + \alpha \boldsymbol{y}\boldsymbol{y}^T)^2 = I + \beta \boldsymbol{y}\boldsymbol{y}^T$ and this holds when $2\alpha + \alpha^2\boldsymbol{y}^T\boldsymbol{y} = \beta$. Solving this quadratic equation for α, we obtain the desired result. The eigenvalues of B are 1 and $1 + \beta\boldsymbol{y}^T\boldsymbol{y}$, which is greater than or equal to 0 since B is positive semidefinite. Thus, $\sqrt{1 + \beta\boldsymbol{y}^T\boldsymbol{y}}$ is real, so, α is real. □

Lemma 2. *For all t, if $G_0 = I$ and $S_0 = 0$, then*

$G_{t+1} = I + \Phi S_{t+1} \Phi^T$ and
$S_{t+1} = S_t$
$+ \alpha_t (I + S_t K_0)(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T (I + K_0 S_t^T)(I + K_0 S_t)$.

Proof. We prove this lemma using induction. In the base case, $S_0 = 0$, implying $G_0 = I$ and $G_0^T G_0 = A_0 = I$. Now, let the hypothesis holds for step t, i.e., $G_t = I + \Phi S_t \Phi^T$. Note that this form for G_t is analogous to the form for A as given in (6) (however, the matrices S and M are not equivalent). The update for matrix G at step $t+1$ is given by

$$\begin{aligned} G_{t+1} &= \left(I + \beta_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T\right)^{1/2}, \\ G_t &= \left(I + \alpha_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T\right) G_t, \end{aligned} \tag{19}$$

where $\boldsymbol{v}_t = \phi(\boldsymbol{y}_t) - \phi(\boldsymbol{z}_t)$ and α is given by Lemma 1. Now, substituting for G_t, we get

$$G_{t+1} = I + \Phi S_t \Phi^T + \alpha_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T G_t. \tag{20}$$

Now, $\boldsymbol{v}_t = \Phi(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})$. Thus,

$$\begin{aligned} G_t \boldsymbol{v}_t &= (I + \Phi S_t \Phi^T)\Phi(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t}) \\ &= (\Phi + \Phi S_t \Phi^T \Phi)(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t}) \\ &= \Phi(I + S_t K_0)(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t}), \end{aligned} \tag{21}$$

where the last equality follows from $\Phi^T \Phi = K_0$. Similarly,

$$\Phi G_t = \Phi^T (I + \Phi S_t \Phi) = (\Phi + \Phi^T \Phi S_t \Phi) = (I + K_0 S_t)\Phi. \tag{22}$$

Using (20), (21), and (22),

$$G_{t+1} = I + \Phi S_t \Phi^T + \alpha_t \Phi (I + S_t K_0)(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T \left(I + K_0 S_t^T\right)(I + K_0 S_t)\Phi.$$

Thus, substituting,

$$S_{t+1} = S_t + \alpha_t (I + S_t K_0)(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T \left(I + K_0 S_t^T\right)(I + K_0 S_t)$$

proves the lemma. □

ACKNOWLEDGMENTS

The authors would like to thank Yong Jae Lee and Greg Shakhnarovich for helpful discussions and for sharing the Flickr and Poser data, and Simon Winder, Matthew Brown, and Gang Hua for making the patch image data available. This research was supported in part by the US National Science Foundation (NSF) CAREER 0747356, Microsoft Research, US Defense Advanced Research Projects Agency (DARPA) VIRAT, NSF EIA-0303609, and the Henry Luce Foundation.

REFERENCES

[1] D. Lowe, "Distinctive Image Features from Scale Invariant Keypoints," *Int'l J. Computer Vision,* vol. 60, no. 2, 2004.
[2] N. Snavely, S. Seitz, and R. Szeliski, "Photo Tourism: Exploring Photo Collections in 3D," *Proc. ACM SIGGRAPH,* pp. 835-846, 2006.
[3] A. Torralba, R. Fergus, and W.T. Freeman, "80 Million Tiny Images: A Large Database for Non-Parametric Object and Scene Recognition." *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 30, no. 11, pp. 1958-1970, Nov. 2008.
[4] J. Sivic and A. Zisserman, "Video Data Mining Using Configurations of Viewpoint Invariant Regions," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2004.
[5] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast Pose Estimation with Parameter-Sensitive Hashing," *Proc. IEEE Int'l Conf. Computer Vision,* 2003.
[6] V. Athitsos and S. Sclaroff, "Estimating 3D Hand Pose from a Cluttered Image," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2003.
[7] 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.
[8] A. Frome, Y. Singer, F. Sha, and J. Malik, "Learning Globally-Consistent Local Distance Functions for Shape-Based Image Retrieval and Classification," *Proc. IEEE Int'l Conf. Computer Vision,* 2007.
[9] K. Grauman and T. Darrell, "The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features," *Proc. IEEE Int'l Conf. Computer Vision,* 2005.
[10] E. Xing, A. Ng, M. Jordan, and S. Russell, "Distance Metric Learning, with Application to Clustering with Side Information," *Advances in Neural Information Processing Systems,* 2002.
[11] A. Bar-Hillel, T. Hertz, N. Shental, and D. Weinshall, "Learning a Mahalanobis Metric from Equivalence Constraints," *J. Machine Learning Research,* vol. 6, pp. 937-965, June 2005.
[12] T. Hertz, A. Bar-Hillel, and D. Weinshall, "Learning Distance Functions for Image Retrieval," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2004.
[13] K. Weinberger, J. Blitzer, and L. Saul, "Distance Metric Learning for Large Margin Nearest Neighbor Classification," *Advances in Neural Information Processing Systems,* 2006.
[14] A. Globerson and S. Roweis, "Metric Learning by Collapsing Classes," *Advances in Neural Information Processing Systems,* 2005.
[15] J. Davis, B. Kulis, P. Jain, S. Sra, and I. Dhillon, "Information-Theoretic Metric Learning," *Proc. Int'l Conf. Machine Learning,* 2007.
[16] 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.
[17] J. Uhlmann, "Satisfying General Proximity/Similarity Queries with Metric Trees," *Information Processing Letters,* vol. 40, pp. 175-179, 1991.
[18] P. Jain, B. Kulis, and K. Grauman, "Fast Image Search for Learned Metrics," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2008.
[19] G. Lanckriet, N. Cristinanini, P. Bartlett, L. Ghaoui, and M. Jordan, "Learning the Kernel Matrix with Semidefinite Programming," *J. Machine Learning Research,* vol. 5, pp. 27-72, 2004.
[20] M. Varma and D. Ray, "Learning the Discriminative Power Invariance Trade Off," *Proc. IEEE Int'l Conf. Computer Vision,* 2007.
[21] J. Goldberger, S.T. Roweis, G.E. Hinton, and R. Salakhutdinov, "Neighbourhood Components Analysis," *Advances in Neural Information Processing Systems,* 2004.
[22] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality Reduction by Learning an Invariant Mapping," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 1735-1742, 2006.
[23] M. Schultz and T. Joachims, "Learning a Distance Metric from Relative Comparisons," *Advances in Neural Information Processing Systems,* 2003.
[24] A. Frome, Y. Singer, and J. Malik, "Image Retrieval and Classification Using Local Distance Functions," *Advances in Neural Information Processing Systems 19,* B. Scholkopf, J. Platt, and T. Hofmann, eds., MIT Press, 2007.
[25] K. Crammer, J. Keshet, and Y. Singer, "Kernel Design Using Boosting," *Advances in Neural Information Processing Systems,* 2002.
[26] T. Hertz, A. Bar-Hillel, and D. Weinshall, "Learning a Kernel Function for Classification with Small Training Samples," *Proc. Int'l Conf. Machine Learning,* 2006.
[27] P. Jain, T. Huynh, and K. Grauman, "Learning Discriminative Matching Functions for Local Image Features," technical report, Univ. of Texas at Austin, Apr. 2007.

[28] A. Bosch, A. Zisserman, and X. Munoz, "Representing Shape with a Spatial Pyramid Kernel," *Proc. Int'l Conf. Image and Video Retrieval,* 2007.
[29] 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.
[30] R. Duda, P. Hart, and D. Stork, *Pattern Classification,* second ed., chap. 10. John Wiley and Sons, Inc., 2001.
[31] S. Roweis and L. Saul, "Nonlinear Dimensionality Reduction by Locally Linear Embedding," *Science,* vol. 290, no. 5500, pp. 2323-2326, 2000.
[32] J. Tenenbaum, V. de Silva, and J. Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction," *Science,* vol. 290, no. 5500, pp. 2319-2323, Dec. 2000.
[33] M. Badoiu, E. Demaine, M. Hajiaghayi, and P. Indyk, "Low-Dimensional Embedding with Extra Information," *Proc. 20th Symp. Computational Geometry,* 2004.
[34] A. Torralba, R. Fergus, and Y. Weiss, "Small Codes and Large Image Databases for Recognition," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2008.
[35] 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.
[36] D. Nister and H. Stewenius, "Scalable Recognition with a Vocabulary Tree," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2006.
[37] P. Indyk and R. Motwani, "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality," *Proc. 30th Ann. Symp. Theory of Computing,* 1998.
[38] M. Charikar, "Similarity Estimation Techniques from Rounding Algorithms," *Proc. ACM Ann. Symp. Theory of Computing,* 2002.
[39] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-Sensitive Hashing Scheme Based on p-Stable Distributions," *Proc. Ann. Symp. Computational Geometry,* 2004.
[40] B. Georgescu, I. Shimshoni, and P. Meer, "Mean Shift Based Clustering in High Dimensions: A Texture Classification Example," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2003.
[41] P. Indyk and N. Thaper, "Fast Image Retrieval via Embeddings," *Proc. Int'l Workshop Statistical and Computational Theories of Vision,* 2003.
[42] K. Grauman and T. Darrell, "Pyramid Match Hashing: Sub-Linear Time Indexing over Partial Correspondences," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2007.
[43] M. Muja and D.G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration," *Proc. Int'l Conf. Computer Vision Theory and Applications,* 2009.
[44] *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice,* G. Shakhnarovich, T. Darrell, and P. Indyk, eds. The MIT Press, 2006.
[45] 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.
[46] H. Ling and S. Soatto, "Proximity Distribution Kernels for Geometric Context in Category Recognition," *Proc. IEEE Int'l Conf. Computer Vision,* 2007.
[47] K. Grauman and T. Darrell, "The Pyramid Match Kernel: Efficient Learning with Sets of Features," *J. Machine Learning Research,* vol. 8, pp. 725-760, Apr. 2007.
[48] K. Grauman and T. Darrell, "Approximate Correspondences in High Dimensions," *Advances in Neural Information Processing Systems,* 2007.
[49] F. Odone, A. Barla, and A. Verri, "Building Kernels from Binary Strings for Image Matching," *IEEE Trans. Image Processing,* vol. 14, no. 2, pp. 169-180, Feb. 2005.
[50] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2006.
[51] J. Ha, C. Rossbach, J. Davis, I. Roy, H. Ramadan, D. Porter, D. Chen, and E. Witchel, "Improved Error Reporting for Software That Uses Black-Box Components," *Proc. Conf. Programming Language Design and Implementation,* 2007.
[52] L. Taycher, G. Shakhnarovich, D. Demirdjian, and T. Darrell, "Conditional Random People: Tracking Humans with CRFs and Grid Filters," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2006.
[53] K. Mikolajczyk and C. Schmid, "Scale and Affine Invariant Interest Point Detectors," *Int'l J. Computer Vision,* vol. 60, no. 1, pp. 63-86, Oct. 2004.
[54] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust Wide Baseline Stereo from Maximally Stable Extremal Regions," *Proc. British Machine Vision Conf.,* 2002.
[55] G. Hua, M. Brown, and S. Winder, "Discriminant Embedding for Local Image Descriptors," *Proc. IEEE Int'l Conf. Computer Vision,* 2007.

Brian Kulis received the BA degree in computer science and mathematics from Cornell University in 2003 and the PhD degree in computer science from the University of Texas in 2008. He is currently a postdoctoral fellow in the Electrical Engineering and Computer Science Department at the University of California at Berkeley and the International Computer Science Institute. His research interests focus on machine learning, data mining, and large-scale optimization. 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 is 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. He is a member of the IEEE.

Prateek Jain received the BTech degree in computer science and engineering from the Indian Institute of Technology (IIT), Kanpur. He is currently working toward the PhD degree in the Computer Science Department at the University of Texas at Austin. His research interests are in machine learning, large-scale optimization, and computer vision. He is a recipient of an MCD fellowship from the University of Texas (2005-2009) and a student scholarship award from the International Conference on Machine Learning (ICML '09). He is a student member of the IEEE.

Kristen Grauman received the BA degree from Boston College in 2001 and the SM and PhD degrees from the Massachusetts Institute of Technology, in the Electrical Engineering and Computer Science Department's Computer Science and Artificial Intelligence Laboratory in 2003 and 2006, respectively. She is the Clare Boothe Luce Assistant Professor in the Department of Computer Sciences at the University of Texas at Austin (UT Austin). She is a recipient of a 2008 US National Science Foundation (NSF) CAREER Award, the Microsoft Research New Faculty Fellowship, and the Frederick A. Howes Scholar Award in computational science. Her group's research in computer vision and machine learning focuses on visual search and category recognition. She is a member of the IEEE.