Fast Image Search for Learned Metrics

Prateek Jain Brian Kulis Kristen Grauman
Department of Computer Sciences
University of Texas at Austin
{pjain, kulis, grauman}@cs.utexas.edu

Abstract

We introduce a method that enables scalable image search for learned metrics. Given pairwise similarity and dissimilarity constraints between some images, we learn a Mahalanobis distance function that captures the images' underlying relationships well. To allow sub-linear 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 make it infeasible to learn an explicit weighting over the feature dimensions. We demonstrate the approach applied to a variety of image datasets. Our learned metrics improve accuracy relative to commonly-used metric baselines, while our hashing construction enables efficient indexing with learned distances and very large databases.

1. Introduction

As the world's store of digital images continues to grow exponentially, and as novel data-rich approaches to computer vision begin to emerge, many interesting problems demand fast techniques capable of accurately searching very large databases of images or image features. For instance, local feature-based recognition methods require searching huge databases of patch descriptors [20], as do new methods for computing 3D models from multi-user photo databases [25]. Similarly, image- or video-based data mining [27, 24] and example-based approaches to pose estimation [23, 2] seek to leverage extremely large image collections, while nearest neighbor classifiers are frequently employed for recognition and shape matching [31, 10]. For most such tasks, the quality of the results relies heavily on the chosen image representation and the distance metric used to compare examples.

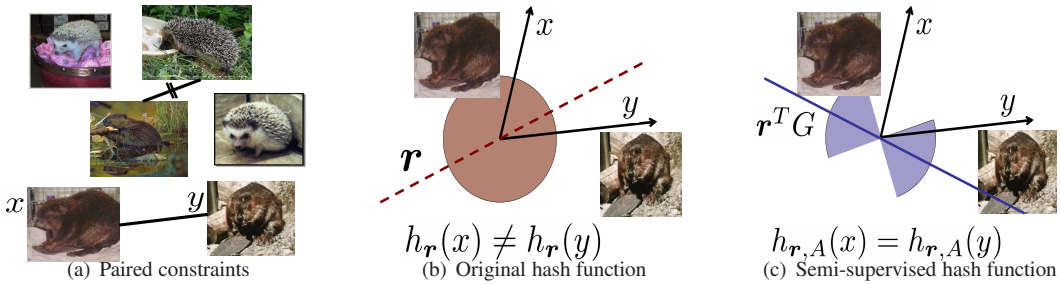
Unfortunately, preferred representations tend to be high-dimensional [20, 24], and often the best distance metric is one specialized (or learned) for the task at hand [10, 31, 13], rather than, say, a generic Euclidean norm or Gaussian kernel. Neither factor bodes well for large-scale image search: known data structures for efficient exact search are ineffective for high-dimensional spaces, while existing methods for approximate sub-linear time search are defined only for certain standard metrics. Thus, there is a tension when choosing an image 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.

A good distance metric between images accurately reflects the true underlying relationships, e.g., the category labels or other hidden parameters. It should 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 [30, 3, 15, 8, 10]. 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 datasets, 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 [9, 28]; approximate search methods can guarantee sub-linear 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 image databases.

In this work we introduce a method for fast approxi-

978-1-4244-2243-2/08/$25.00 ©2008 IEEE



(a) Paired constraints (b) Original hash function (c) Semi-supervised hash function

Figure 1. (a) 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). (b) Whereas existing randomized LSH functions hash examples similar under the original distance together, (c) 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 (b) denotes that existing LSH functions generate a hyperplane uniformly at random to separate images, in contrast, as indicated by the blue "hourglass" region in (c), our hash functions bias the selection of random hyperplanes to reflect the specified (dis)similarity constraints.

mate 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 learned metric without resorting to a naive linear 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., [30, 3, 8]), while the latter is of particular interest for commonly used image representations, such as bags-of-words, multi-dimensional multi-resolution histograms, and other high-dimensional features.

We demonstrate the generality of our approach by applying it to three distinct large-scale image search problems: exemplar-based recognition, pose estimation, and feature indexing. Our method allows rapid and accurate retrieval, and gains 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, 29], as well as methods to learn a Mahalanobis metric [30, 3, 8], and methods to learn example-specific local distance functions [10]. 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 [1, 13]. In contrast to learned metrics, such geometric embeddings are meant to mirror a fixed distance function and do not adapt to reflect supervised constraints.

In order to efficiently index multi-dimensional data, data structures based on spatial partitioning and recursive hyperplane decomposition have been developed, e.g. $k-d$-trees [9] and metric trees [28]. Due to the particular importance of indexing local patch features, several tree-based strategies have also been proposed [4, 21] 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 query time improvement over a naive linear scan [17].

As such, researchers have considered the problem of *approximate* similarity search, where a user is afforded explicit tradeoffs 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 sub-linear in the size of the database, notably the locality-sensitive hashing (LSH) methods of [17, 6]. Data-dependent variants of LSH have been proposed: the authors of [11] select partitions based on where data points are concentrated, while in [23] boosting is used to select feature dimensions that are most indicative of similarity in the parameter space. This tunes the hash functions according to the estimation problem of interest; however, indexed examples must be sorted according to the input space (non-learned) distance.

We address the problem of sub-linear time approximate similarity search for a class of *learned* metrics. While randomized algorithms such as LSH have been employed heavily in vision to mitigate the time complexity of identifying similar examples [22], 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 Figure 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.

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.

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:

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

for all $i, j = 1, \ldots, n$. Note that a generalized inner product (kernel) measures the pairwise similarity associated with that distance: $s_A(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T A \boldsymbol{x}_j$. Given a set of inter-point 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 [30, 3, 8]; we consider the information-theoretic metric learning method of [8] because it is kernelizable. 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 [8].

3.2. Information-Theoretic Metric Learning

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

To compute A, the LogDet divergence is minimized while enforcing desired constraints:

$$\begin{aligned} \min_{A \succeq 0} \quad & D_{\ell \mathrm{d}}(A, A_0) \\ \text{s. t.} \quad & d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq u \quad (i,j) \in \mathcal{S}, \\ & d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq \ell \quad (i,j) \in \mathcal{D}, \end{aligned} \quad (2)$$

where $D_{\ell \mathrm{d}}(A, A_0) = \mathrm{tr}(AA_0^{-1}) - \log\det(AA_0^{-1}) - d$, $\mathcal{S}$ 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 (defined below).

Computing the optimal solution to (2) involves repeatedly projecting the current solution onto a single constraint, via the update:

$$A_{t+1} = A_t + \beta_t A_t (\boldsymbol{x}_{i_t} - \boldsymbol{x}_{j_t})(\boldsymbol{x}_{i_t} - \boldsymbol{x}_{j_t})^T A_t, \quad (3)$$

where $\boldsymbol{x}_{i_t}$ and $\boldsymbol{x}_{j_t}$ are the constrained data points for iteration t, and β_t is a projection parameter computed by the algorithm.

When the dimensionality of the data is very high, one cannot explicitly work with A, and so the update in (3) 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 an input kernel matrix for the data, the appropriate update is:

$$K_{t+1} = K_t + \beta_t K_t (\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T K_t, \quad (4)$$

where the vectors $\boldsymbol{e}_{i_t}$ and $\boldsymbol{e}_{j_t}$ refer to the i_t-th and j_t-th standard basis vectors, respectively, and the projection parameter β_t is the same as in (3) (see [8]). Note that it is possible for the set of examples involved in constraints to be a superset of the set of examples in the input kernel.

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

3.3. Hashing for Semi-Supervised Similarity Search

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

$$\Pr_{h \in \mathcal{F}}[h(\boldsymbol{x}) = h(\boldsymbol{y})] = sim(\boldsymbol{x}, \boldsymbol{y}), \quad (5)$$

where $sim(\boldsymbol{x}, \boldsymbol{y})$ is some similarity function defined on the collection of objects [6, 17]. 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. Existing LSH functions can accommodate the Hamming distance [17], L_p norms [7], and inner products [6], and such functions have been explored previously in the vision community [22, 23, 14].

In the following we present new algorithms to construct LSH functions for learned metrics. Specifically, we introduce a family of hash functions that accommodate 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.

Explicit Formulation. Given the matrix A for a metric learned as above[1], such that $A = G^T G$, we generate the following randomized hash functions $h_{\boldsymbol{r},A}$, which accept

[1] A variable without a subscript t denotes its value after convergence.

an input point and return a hash key bit:

$$h_{\boldsymbol{r},A}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T G \boldsymbol{x} \geq 0 \\ 0, & \text{otherwise} \end{cases}, \quad (6)$$

where the vector $\boldsymbol{r}$ is chosen at random from a d-dimensional Gaussian distribution with zero mean and unit variance. This construction leverages earlier results showing that (i) the probability of two unit vectors having a dot product with random vector $\boldsymbol{r}$ that are opposite in sign is proportional to the angle between them [12], and (ii) the sign of $\boldsymbol{r}^T \boldsymbol{x}_i$ is therefore a locality-sensitive function for the inner product of any two inputs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ [6].

Thus by parameterizing the hash functions instead by G (which is computable since A is p.d.), we obtain the following relationship:

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

which sustains the LSH requirement of (5) for a learned Mahalanobis metric, whether A is computed using the method of [8] or otherwise [30, 3]. 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 (6) requires that the input dimension d be low enough that A can be explicitly handled in memory, allowing the updates in (3).

Implicit Formulation. We are also 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 representable (e.g., bags of words or histogram pyramids [24, 13]). 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 (4) are necessary. However, this complicates the computation of hash functions, as they can no longer be computed directly as in (6) above. Thus, in this section we derive a new algorithm to make simultaneous implicit updates to both the hash functions and the metric.

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 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 (non-learned) kernel value between example $\boldsymbol{x}_i$ and the input $\boldsymbol{x}_j$. Initially, $K_0 = \Phi^T \Phi$, and so, implicitly, $A_0 = I$. 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:

$$h_{\boldsymbol{r},A}(\phi(\boldsymbol{x})) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T \phi(\boldsymbol{x}) + \sum_{i=1}^{c} \gamma_i^r \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}) \geq 0 \\ 0, & \text{otherwise} \end{cases}, \quad (7)$$

where $\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x})$ is the original kernel value between $\boldsymbol{x}_i$ 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 non-zero 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 (7) efficiently. Our construction relies on two technical lemmas, which we list in the appendix. Recall the update rule for A from (3): $A_{t+1} = A_t + \beta_t A_t \boldsymbol{v}_t \boldsymbol{v}_t^T A_t$, where $\boldsymbol{v}_t = \phi(\boldsymbol{y}_t) - \phi(\boldsymbol{z}_t)$, if points y_t and z_t are involved in the constraint under consideration at iteration t. We emphasize that just as this update must be implemented implicity via (4), so too we must 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:

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

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}:

$$G_{t+1} = (I + \beta_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T)^{1/2} G_t = (I + \alpha_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T) G_t, \quad (8)$$

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

Using (8) 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 zero matrix, and from there every S_{t+1} is iteratively updated in $O(c^2)$ time via $S_{t+1} = S_t + \alpha_t (I + S_t K_0)(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T (I + K_0 S_t^T)(I + K_0 S_t)$.

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

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

Step	Explicit	Implicit
Metric learning projection (offline)	$O(d^2)$	$O(c^2)$
Hashing: compute $h_{\boldsymbol{r},A}(\boldsymbol{x})$	$O(d)$	$O(z)$
Search: identify the query's ANNs	$O(Md)$	$O(Mz)$

Table 1. Computational complexity for the proposed method, using variables defined in the text.

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 known constrained points, and thus can be efficiently computed in the training phase, prior to hashing anything into the database. Finally, having determined the expression for $\boldsymbol{r}^T G \phi(\boldsymbol{x})$, we arrive at our hash function definition in (7). Note the analogy between the use of $\boldsymbol{r}^T G \boldsymbol{x}$ and $\boldsymbol{r}^T G \phi(\boldsymbol{x})$ in (6) and (7), respectively.

In this section we presented our main technical contribution: 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 a simple merging of the metrics in [8] with LSH.

3.4. Searching Hashed Examples

Having constructed LSH functions for learned metrics, we can apply existing methods [17, 6] to perform sub-linear time approximate similarity search. Given N data points in a Hamming space and an input $\boldsymbol{x}_q$, approximate near-neighbor (ANN) 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 $\boldsymbol{x}_q$. We employ the method of [6], which requires searching $M = 2N^{1/(1+\epsilon)}$ examples to obtain the first ANN. (Note that $M << N$ for large databases.) After hashing, we only need to compute the learned kernel values between the query and the examples with which it collided. 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.

To generate b-bit hash keys, we select b random vectors $[\boldsymbol{r}_1, \ldots, \boldsymbol{r}_b]$ to form b hash functions and concatenate the resulting bits from (6) or (7). There is a tradeoff in the selection of b: larger values will increase the accuracy of how well the keys themselves reflect the learned metric, but will increase computation time and can lead to too few collisions in the hash tables. On the other hand, lower values of b make hashing faster, but the key will only coarsely reflect our metric, and too many collisions may result.

Table 1 summarizes the computational complexity for the main steps of our algorithm: projections during offline metric learning, computing each hash bit for a given point, and computing the ANNs for a hashed query. z is the number of non-zero entries in the query, $z \leq d$. See [18] for more details on the complexity.

4. Results

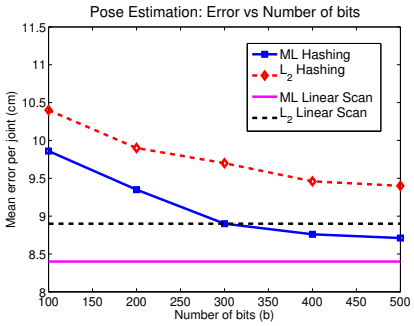
The need to search for images or local image descriptors within very large databases arises frequently. In the following we apply our algorithm for image search in three distinct domains: exemplar-based recognition, pose estimation, and feature indexing. In all cases, our experimental goal is twofold: 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 performing sub-linear 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 among 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 (≈ 100) of the database examples according to the non-learned metric, and then let the distance constraints' lower ℓ and upper u limits be the 1-st and 99-th 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 patches and object images).

Human Body Pose Estimation. First we demonstrate our method applied to single-frame human body pose estimation. Example-based techniques to infer pose (e.g. [2, 23]) 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 (NN). 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 the learned metric. This is similar to the goals of the parameter-sensitive hashing (PSH) method of [23]. However our approach is distinct from [23] 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 empirically below.

We use a database of half a million examples provided by the authors of [26], 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. Each image is represented by a $d = 24,016$-dimensional multi-scale edge detection histogram (EDH). The vectors' high dimension requires our implicit formulation for semi-supervised

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, lin. 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



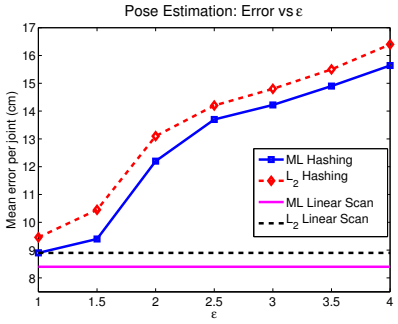


Figure 2. Pose results. **Left**: Mean pose error (in cm) obtained with each method. Our approach (denoted ML) outperforms the L_2 baseline and PSH [23]. **Middle**: Error as a function of the number of hash bits. Fast search with the learned metric is more accurate than the L_2 baseline. For both, the 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.

Figure 3. Examples of pose estimates. Each column contains a different pose. Top row contains query images, remaining rows show the best pose retrieved by each method. Second row shows best pose obtained by our method

hash functions. We use a linear kernel over $c = 50$ randomly selected examples as the initial kernel (K_0). We hold out 1000 test query examples, and generate $1,000,000$ similarity constraints among $50K$ 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 τ, 10 are randomly picked and their distance to the example is constrained to be greater than u. The values of tau and c are selected with cross-validation.

As baselines, we compute results for NN search with both the Euclidean distance (L_2) on the EDH's, and the Hamming distance on the PSH embeddings provided by the authors of [26]. To hash with the L_2 baseline we simply apply [6]. We also use PCA to reduce the dimensionality of the EDH vectors in order to apply our explicit formulation. 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.

The table in Figure 2 shows the overall errors for each method. (Throughout our approach is denoted by 'ML'.) 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-d 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% confidence.

Figure 3 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 dataset (as it did for data in [23]), 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 dimension.

Our semi-supervised hash functions maintain the accuracy of the learned metric, but for orders of magnitude less search time than the linear scan. With our Matlab implementation, a linear scan requires 433.25 s per query, while our hashing technique requires just 1.39 s. On average, metric learning with hashing searches just 0.5% of the database. Figure 2 compares the error obtained by ML+hashing and L_2+hashing when varying the number of hash bits (middle 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). Similarly, since $M = 2N^{1/(1+\epsilon)}$, for higher values of ϵ we must search fewer examples, but accuracy guarantees decrease.

Exemplar-based Object Categorization. Next we evaluate our method applied for NN object recognition with the Caltech-101, a now common benchmark. To compare these images we consider learning kernels on top of the pyramid match kernel (PMK) [13] applied to SIFT features, and the kernel designed in [31] applied to geometric blur features.

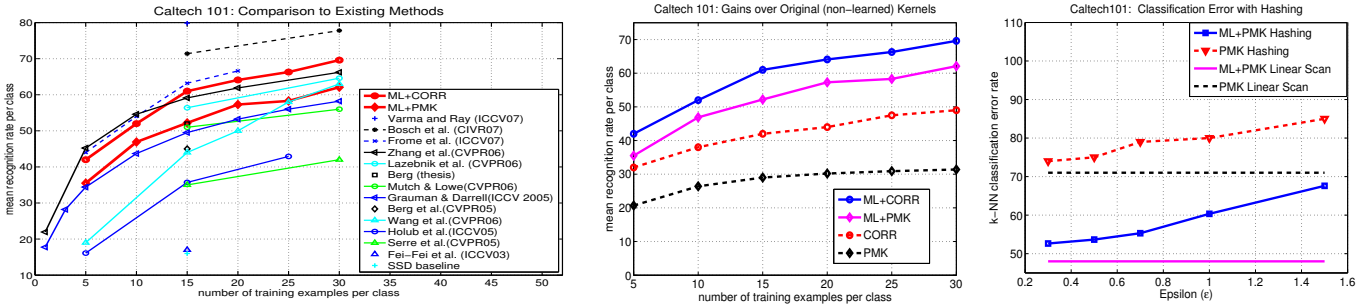


Figure 4. Caltech-101 results. **Left**: Comparison against existing techniques. Our method outperforms all other single metric/kernel approaches. **Middle**: Our learned kernels significantly improve NN search accuracy relative to their non-learned counterparts, the CORR and PMK kernels. **Right**: Comparison of the k-NN classification error when hashing with the original and learned PMK. This plot shows the accuracy-search time tradeoff when using the original or learned hashing functions. CORR refers to the kernel proposed by Zhang et al. [31]. (Best viewed in color.)

The PMK uses multi-resolution histograms to estimate the correspondence between two sets of local image features. To hash with the non-learned PMK, the pyramids can be embedded in such a way that standard inner product LSH functions are applicable [14]. 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 [31] 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 our implicit formulation, but can only hash with the learned PMK, since explicit vector space representation ($\phi(\boldsymbol{x})$) for the CORR is unknown.

We first evaluate the effectiveness of metric learning itself on this dataset. We pose a k-NN classification task, and evaluate both the original (PMK or CORR) and learned kernels when used in a linear scan mode. 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 dataset.

Figure 4 shows our results relative to all other existing techniques (left) and specifically against the original baseline kernels for NN (middle). Our approach outperforms all existing single-kernel classifier methods when using the learned CORR kernel: we achieve 61.0% accuracy for $T = 15$ and 69.6% accuracy for $T = 30$. Our learned PMK achieves 52.2% accuracy for $T = 15$ and 62.1% accuracy for $T = 30$. The middle plot in Figure 4 reveals gains in NN retrieval accuracy; notably, our learned kernels with simple NN classification also outperform the baseline kernels when used with SVMs [31, 13]. Only the results of recent multiple-metric approaches [10, 29, 5] (shown with dashed lines in the left plot) 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. We hypothesize that using the kernels learned with our method along with the ones used in [29, 5] would further boost the accuracy; this remains as the subject of future experiments.

Now we consider hashing over the learned PMK. For $T = 15$, our learned hash functions achieve 47% accuracy, and require about $10x$ less computation time than a linear scan when accounting for the hash key computation (here $N = 1515$, which is modest compared to the pose data). The rightmost plot in Figure 4 shows the error of our learned PMK-based hashing compared to the baseline [14] 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 sub-linear time search.

Indexing Local Patch Descriptors. Finally, we evaluate our approach on a patch matching task using data provided from the Photo Tourism project [25] and [16]. The dataset contains about 300K 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 [16], 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 non-matching patch pairs, with a 50-50 mix taken from the Trevi and Halfdome portions of the data. All methods are tested on 100K pairs from the Notre Dame portion. The left plot in Figure 5 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% of the true positives are retrieved. ML+raw intensities yields a significant gain over

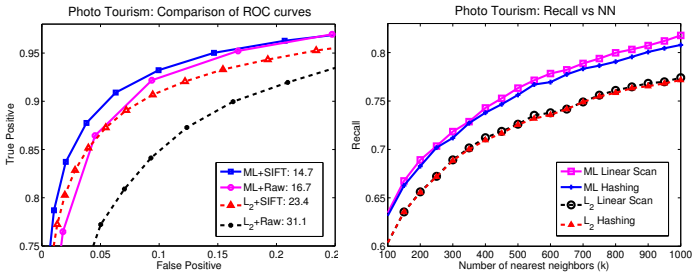


Figure 5. Patch results. **Left**: This plot illustrates accuracy improvements of the learned metric (ML) relative to L_2 baselines, for both raw patches (dimensionality $d = 4096$) and SIFT descriptors (dimensionality $d = 128$). **Right**: This plot shows the recall as a function of the number of SIFT patches retrieved, for our method and the L_2 baseline. Our semi-supervised hash functions maintain accuracy close to a linear scan, while requiring much less search time.

L_2+raw, while ML+SIFT also gives some improvement.[2]

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 Figure 5, we see that the learned metric outperforms the L_2 baseline, and that hashing does not noticeably degrade accuracy. When $k = 1000$, we search only 16.1% of the database when hashing over the learned metric, and when $k = 1$, we search only 0.8%, leading to substantial gains in retrieval time (about a factor of 80 vs. linear scan).

Conclusions: We have introduced a method to enable efficient approximate similarity search for learned metrics, and experiments show good results for a variety of datasets. 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 with a variety of datasets clearly demonstrate our technique's accuracy and flexibility for large-scale image search tasks.

Appendix

Proofs are provided in [18].

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

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

$$G_{t+1} = I + \Phi S_{t+1} \Phi^T$$

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

[2] In this experiment we were able to reproduce the baseline for L_2 given in [16], however we were unable to do so for their SIFT baseline, for which 6% error is obtained. We suspect this is due to our un-optimized SIFT extraction, and that ML would continue to yield similar improvements as above if provided better descriptors.

Acknowledgements: We thank Greg Shakhnarovich for sharing the Poser data. This research was supported in part by grants from ORAU and Microsoft Research.

References

[1] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. BoostMap: A Method for Efficient Approximate Similarity Rankings. In *CVPR*, 2004.
[2] V. Athitsos and S. Sclaroff. Estimating 3D Hand Pose from a Cluttered Image. In *CVPR*, June 2003.
[3] A. Bar-Hillel, T. Hertz, N. Shental, and D. Weinshall. Learning a Mahalanobis Metric from Equivalence Constraints. *Journal of Machine Learning Research*, 6:937–965, June 2005.
[4] J. Beis and D. Lowe. Shape Indexing Using Approximate Nearest-Neighbour Search in High Dimensional Spaces. In *CVPR*, 1997.
[5] A. Bosch, A. Zisserman, and X. Munoz. Representing shape with a spatial pyramid kernel. In *CIVR*, 2007.
[6] M. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *ACM Symp. on Theory of Computing*, 2002.
[7] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In *SOCG*, 2004.
[8] J. Davis, B. Kulis, P. Jain, S. Sra, and I. Dhillon. Information-Theoretic Metric Learning. In *ICML*, 2007.
[9] J. Freidman, J. Bentley, and A. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.
[10] A. Frome, Y. Singer, F. Sha, and J. Malik. Learning globally-consistent local distance functions for shape-based image retrieval and classification. In *ICCV*, 2007.
[11] B. Georgescu, I. Shimshoni, and P. Meer. Mean Shift Based Clustering in High Dimensions: A Texture Classification Example. In *CVPR*, 2003.
[12] M. Goemans and D. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *JACM*, 42(6):1115–1145, 1995.
[13] K. Grauman and T. Darrell. The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features. In *ICCV*, 2005.
[14] K. Grauman and T. Darrell. Pyramid Match Hashing: Sub-Linear Time Indexing Over Partial Correspondences. In *CVPR*, 2007.
[15] T. Hertz, A. Bar-Hillel, and D. Weinshall. Learning distance functions for image retrieval. In *CVPR*, Washington, DC, June 2004.
[16] G. Hua, M. Brown, and S. Winder. Discriminant embedding for local image descriptors. In *ICCV*, Rio de Janeiro, October 2007.
[17] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *30th STOC*, 1998.
[18] P. Jain, B. Kulis, and K. Grauman. Fast similarity search for learned metrics. Technical Report TR-07-48, University of Texas at Austin, 2007.
[19] G. Lanckriet, N. Cristinanini, P. Bartlett, L. Ghaoui, and M. Jordan. Learning the kernel matrix with semidefinite programming. In *Journal of Machine Learning Research*, volume 5, pages 27–72, 2004.
[20] D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *IJCV*, 60(2), 2004.
[21] D. Nister and H. Stewenius. Scalable Recognition with a Vocabulary Tree. In *CVPR*, 2006.
[22] G. Shakhnarovich, T. Darrell, and P. Indyk, editors. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. 2006.
[23] G. Shakhnarovich, P. Viola, and T. Darrell. Fast Pose Estimation with Parameter-Sensitive Hashing. In *ICCV*, 2003.
[24] J. Sivic and A. Zisserman. Video Data Mining Using Configurations of Viewpoint Ivariant Regions. In *CVPR*, Washington, D.C., June 2004.
[25] N. Snavely, S. Seitz, and R. Szeliski. Photo Tourism: Exploring Photo Collections in 3D. In *SIGGRAPH Conference Proceedings*, pages 835–846, New York, NY, USA, 2006. ACM Press.
[26] L. Taycher, G. Shakhnarovich, D. Demirdjian, and T. Darrell. Conditional random people:tracking humans with crfs and grid filters. In *CVPR*, 2006.
[27] A. Torralba, R. Fergus, and W. T. Freeman. Tiny images. Technical Report MIT-CSAIL-TR-2007-024, CSAIL, MIT, 2007.
[28] J. Uhlmann. Satisfying General Proximity / Similarity Queries with Metric Trees. *Information Processing Letters*, 40:175–179, 1991.
[29] M. Varma and D. Ray. Learning the discriminative power-invariance trade-off. In *ICCV*, 2007.
[30] E. Xing, A. Ng, M. Jordan, and S. Russell. Distance Metric Learning, with Application to Clustering with Side-Information. In *NIPS*, 2002.
[31] H. Zhang, A. Berg, M. Maire, and J. Malik. SVM-KNN: Discriminative Nearest Neighbor Classification for Visual Category Recognition. In *CVPR*, 2006.