Pyramid Match Hashing: Sub-Linear Time Indexing Over Partial Correspondences

Kristen Grauman
University of Texas at Austin
Austin, TX 78712 USA
grauman@cs.utexas.edu

Trevor Darrell
Massachusetts Institute of Technology
Cambridge, MA 02139 USA
trevor@csail.mit.edu

Abstract

Matching local features across images is often useful when comparing or recognizing objects or scenes, and efficient techniques for obtaining image-to-image correspondences have been developed [4, 11, 16]. However, given a query image, searching a very large image database with such measures remains impractical. We introduce a sub-linear time randomized hashing algorithm for indexing sets of feature vectors under their partial correspondences. We develop an efficient embedding function for the normalized partial matching similarity between sets, and show how to exploit random hyperplane properties to construct hash functions that satisfy locality-sensitive constraints. The result is a bounded approximate similarity search algorithm that finds $(1+\epsilon)$-approximate nearest neighbor images in $O(N^{1/(1+\epsilon)})$ time for a database containing N images represented by (varying numbers of) local features. We demonstrate our approach applied to image retrieval for images represented by sets of local appearance features, and show that searching over correspondences is now scalable to large image databases.

1. Introduction

Representations that decompose images into local patches or regions have proven to be very useful, in large part due to their tendency to be preserved under a variety of imaging conditions and transformations. To leverage local representations when performing image-to-image comparisons, many effective retrieval and object recognition algorithms evaluate similarity by establishing correspondences (or a matching) between sets of local parts, e.g., [6, 7, 3, 11, 4].

As advances are made in terms of powerful representations and sophisticated matching techniques, it is critical to consider how they might scale to accommodate image retrieval with very large databases and recognition with a very large number of categories or exemplars. If a retrieval system is to index all of the images on the web by their visual content, it cannot conceivably operate with a naive linear scan, where a matching is computed between a query and every image in the database. Likewise, if a recognition engine based on correspondences is to ever cope with the thousands of categories humans easily recognize, it must not require that a novel input be matched against every stored exemplar for all categories.

Although researchers have developed the means to perform each individual matching efficiently [6, 11, 4, 16], indexing *over* those correspondences remains a significant computational challenge. While various tree data structures have been explored to efficiently index features or keypoints themselves [20, 18, 17, 22, 19], existing methods are limited to handling only these single vector inputs, and because they index features independently, do not allow us to evaluate one-to-one matchings.

In this work we present a sub-linear time randomized hashing algorithm for indexing sets of feature vectors according to their partial correspondences. We construct an embedding and locality-sensitive hash functions under which feature sets can be efficiently indexed, with guarantees on the expected error induced by the approximation relative to the significant gains in query speed we achieve. Specifically, we find the $(1+\epsilon)$-approximate nearest neighbor (NN) images in $O(N^{1/(1+\epsilon)})$ time for a database containing N images, each of which is represented by a set of local features. The matching effected is partial and robust: images may be described by varying numbers of features, and the presence of very distant ("outlier") features in an image cannot significantly skew the correspondence similarity that is measured for an otherwise good match.

We demonstrate our approach for image retrieval tasks with large image databases, and show that for very little loss in accuracy over a brute force linear scan, we obtain significant computational advantages—typically, only 1-3% of a database needs to be searched. In our experiments we have focused on image matching with local appearance features; however, the approach is general and applies to

1-4244-1180-7/07/$25.00 ©2007 IEEE

any set-based representation where correspondences are a meaningful comparison measure. Beyond content-based image retrieval itself, the sub-linear time search tool we provide has potential applications to recognition with exemplars and other example-based learning problems where a large amount of training data is valuable.

2. Related Work

Vision researchers have previously explored ways to mitigate the cost of establishing correspondences between sets of local image features, either by exploiting structural constraints to more efficiently compute optimal solutions [6] or by designing approximations [4, 11, 16]. In [6], Felzenszwalb and Huttenlocher use dynamic programming and distance transforms to detect a part-based object. To approximate the least-cost correspondences between pairs of local features, Berg et al. optimize a linear bounding problem [4], while Leordeanu and Hebert propose an efficient spectral method [16]. We have developed a linear-time approximation to the partial matching between sets of features, and have demonstrated its use as a kernel for discriminative classification of categories [11]. In [15], Lazebnik et al. develop a variant that operates over spatial features.

While these methods offer good complexity improvements for the image-to-image matching problem, none addresses the problem of how to scale the correspondence measure to index very large databases; despite the fast matchings, a linear scan mode of computation is assumed.

Recent progress has been made using geometric embeddings and randomized algorithms to reduce computation requirements for vision tasks. In the BoostMap method of Athitsos et al., a learned embedding for the Chamfer distance maps data into a Euclidean space, where comparisons are less expensive; however, retrievals are performed with a linear scan [1]. In [14], Indyk and Thaper develop a metric embedding for weighted bipartite graph matching between equally-sized sets and apply it to global color histogram matching with locality-sensitive hashing [13] (LSH), a sub-linear time approach to approximate similarity search. We further explore the embedding and LSH for comparing sets of local shape features in [10], and Shakhnarovich et al. design a variant of LSH that is tuned to retrieve examples that are similar in some parameter space and apply it to index global descriptions of body pose [21].

In this work we also develop a form of locality-sensitive hashing for sub-linear time search. However, in contrast to previous techniques [14, 10, 21], our embedding allows input feature sets to have varying cardinalities, and provides for hashing over a normalized partial match. This is an important advantage for handling outlier "unmatchable" features, as we will demonstrate in Section 4. In addition, unlike [14, 10], with our hashing algorithm it is possible to perform the feature space decomposition according to its underlying structure, which means indexing can remain accurate even for sets with high-dimensional features.

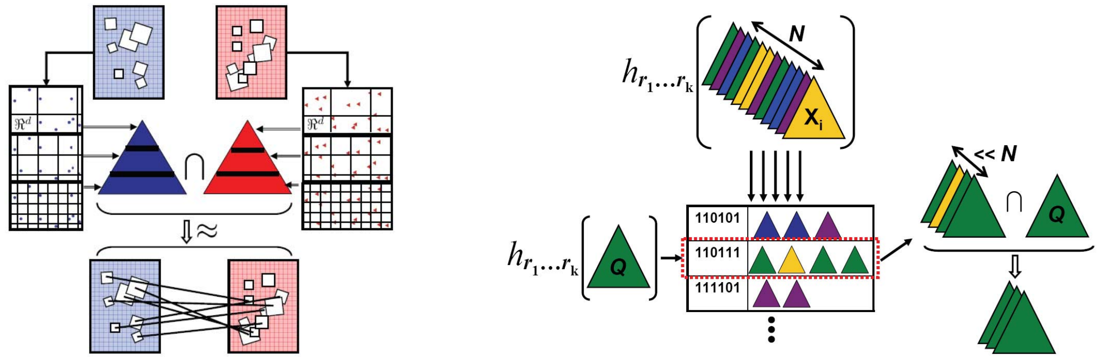
Several researchers have considered special tree data structures to organize image feature vectors for fast access [20, 2, 17, 22, 19]. Beis and Lowe develop the approximate Best-Bin-First technique, a variant of $k-d$ trees, to speed search for individual keypoints [2]. Shao et al. employ a Vantage Point tree to organize feature vectors [22], while Lepetit et al. [17] and Obdrzalek and Matas [20] have shown novel methods based on decision trees for efficiently indexing features. In [19], Nister and Stewenius introduce a "vocabulary-tree", a hierarchical partition that reduces the search time for similar features and allows bag-of-words distances with very large vocabularies.

These approaches share our goal of realizing rapid image-based search. However, they address the problem of how, given a feature vector, to efficiently retrieve the most similar feature vectors among a pool of feature vectors, with similarity defined in terms of Euclidean distance. In contrast, we are concerned with the problem of how, given a *set* of feature vectors, to efficiently retrieve the most similar sets from a database of sets, with similarity defined in terms of one-to-one correspondences (a matching). While the bag-of-words representation in [19] describes quantized features jointly, unlike our approach it does not allow a partial match and cannot formally guarantee sub-linear time image search without assumptions about the frequency with which features will occur in query images.

In addition, the approaches above are intended for accessing images that contain *instances* of the same object, a scenario where identifying a few very similar features has been shown to be sufficient to reach stored images of the same object. Our framework applies to general matchings not only between object instances, but also between textures or categories, which often exhibit wider appearance variation and may not be isolated from a database on the basis of a few discriminative features alone. Instead, the joint matching of all component features may be preferable; such matchings have been shown to yield good category-level comparisons (e.g. [11, 15, 4, 3]).

3. Approach

The main contribution of this work is a novel embedding for a set of vectors that enables sub-linear time approximate similarity search over partial correspondences with random hyperplane hash functions. The idea is to encode a point set with a weighted multi-resolution histogram in such a way that a dot product between any two such encodings will reflect the similarity of the original point sets according to an approximate, normalized partial matching between their component feature vectors. Then, by drawing on a property of random hyperplanes, we designate randomized hash functions which guarantee that examples with strong match-



(a) Pyramid match embedding

(b) Pyramid match hashing

Figure 1. Overview of the approach. The pyramid match (a) takes two sets of feature vectors as input (for instance, two sets of image patch descriptors), maps the vectors to multiresolution histograms, and intersects them to efficiently approximate the optimal partial matching (correspondence) between the original feature sets. Our novel embedding of the pyramid match and associated random hash functions allow sub-linear time indexing over correspondences (b); the pyramid match is applied only to a small portion of the database examples, but we still guarantee a specified retrieval accuracy with high probability.

ing similarity will (with high probability) hash into the same buckets. Approximate similarity search in the Hamming space of the hash keys then identifies the approximate nearest neighbors according to the approximate matching score, in time sub-linear in the number of database examples.

In image retrieval terms, this means we first take a collection of images, each one of which is represented in some fashion by a set of feature vectors. For example, each could be described by a set of SIFT [18] descriptors extracted at salient points, or a set of shape context [3] histograms or geometric blur descriptors [4] extracted at edge points, or a set of color distributions, etc. The database items are prepared by mapping every set of vectors to a single high-dimensional vector via the embedding function. After this embedding, the dot product between any two examples would reflect the partial matching similarity between the original feature sets, that is, the strength of the correspondence between their local parts. All embedded database examples are next encoded as binary hash key strings, with each bit determined with a random hash function designed to probabilistically give similar responses for examples with similar dot products. These hash keys are stored in such a way that they are accessible in sub-linear time.

Given a query image, local features of the chosen type are extracted, and the embedding function is applied to form the vector encoding for the query set. Then, rather than compute the dot product between the embedded query and every embedded database item, we apply the same randomized hash functions used for the database items to index into the stored database hash keys, thereby (with high probability) obtaining in sub-linear time the most similar database neighbors in terms of the normalized partial match between the original local image features. See Figure 1 for a schematic overview of our approach.

We consider point sets from the input space S, which contains sets of vectors drawn from feature space F: $S = \left\{ \mathbf{X} | \mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \right\}$, where each feature is a d-dimensional vector, $\mathbf{x}_i \in F \subseteq \Re^d$, and $m = |\mathbf{X}|$.

A partial matching between two point sets is an assignment that maps all points in the smaller set to some subset of the points in the larger (or equally-sized) set. Given point sets $\mathbf{X}$ and $\mathbf{Y}$, where $|\mathbf{X}| \leq |\mathbf{Y}|$, a partial matching $\mathcal{M}(\mathbf{X}, \mathbf{Y}; \pi)$ pairs each point in $\mathbf{X}$ to some unique point in $\mathbf{Y}$ according to the permutation of indices given by $\pi = \left[\pi_1, \ldots, \pi_{|\mathbf{X}|}\right]$, $1 \leq \pi_i \leq |\mathbf{Y}|$, where π_i specifies which point $\mathbf{y}_{\pi_i} \in \mathbf{Y}$ is matched to $\mathbf{x}_i \in \mathbf{X}$, for $1 \leq i \leq |\mathbf{X}|$. The cost of a partial matching is the sum of the distances between matched points: $\mathcal{C}(\mathcal{M}(\mathbf{X}, \mathbf{Y}; \pi)) = \sum_{\mathbf{x}_i \in \mathbf{X}} ||\mathbf{x}_i - \mathbf{y}_{\pi_i}||_1$. The optimal partial matching uses the assignment π^* that minimizes the matching cost:

$$\pi^* = \underset{\pi}{\operatorname{argmin}}\ \mathcal{C}(\mathcal{M}(\mathbf{X}, \mathbf{Y}; \pi)). \tag{1}$$

Given a database of feature sets $D = \{\mathbf{X}_1, \ldots, \mathbf{X}_N\} \subseteq S$, and a query set of features $\mathbf{Q} \in S$, the nearest neighbor in D in terms of correspondences is the set $\mathbf{R}^*$ that has the minimal partial matching cost to $\mathbf{Q}$:

$$\mathbf{R}^* = \underset{\mathbf{X}_i \in D}{\operatorname{argmin}}\ \mathcal{C}(\mathcal{M}(\mathbf{Q}, \mathbf{X}_i; \pi^*)). \tag{2}$$

Let $C = \mathcal{C}(\mathcal{M}(\mathbf{Q}, \mathbf{R}^*; \hat{\pi}^*))$, where $\hat{\pi}^*$ refers to a bounded approximation for π^*. In this work we develop a sub-linear time hashing algorithm that guarantees retrieval in $O(N^{(1/1+\epsilon)})$ time of an approximate nearest-neighbor $\hat{\mathbf{R}}$ for $\mathbf{Q}$ such that $\mathcal{C}\left(\mathcal{M}(\mathbf{Q}, \hat{\mathbf{R}}; \hat{\pi}^*)\right) \leq (1 + \epsilon)C$.

3.1. Approximate Partial Correspondences

To construct our embedding, we build upon the matching technique called the *pyramid match*, which we introduced in [11]. The pyramid match is a low-distortion approximation for the least-cost correspondence between two sets of vectors that requires only linear time in the number of vectors per set to compute. We will briefly summarize the relevant math of the pyramid match algorithm here, but see [11] for details and intuition for why this approximation works.

Point sets are converted to multi-resolution histograms (pyramids): $\Psi(\mathbf{X}) = [H_0(\mathbf{X}), \ldots, H_{L-1}(\mathbf{X})]$, where $\mathbf{X} \in S$, $L = \lceil \log_2 A \rceil$, A is the feature value range, $H_i(\mathbf{X})$ is a histogram vector formed over points in $\mathbf{X}$ using d-dimensional bins of side length 2^i.[1] These pyramids are represented sparsely, with up to $m = |\mathbf{X}|$ nonzero entries per level.

The (un-normalized) pyramid match score is defined as:

$$\tilde{\mathcal{P}}_\Delta\left(\Psi(\mathbf{Y}), \Psi(\mathbf{Z})\right) = w_{L-1}\mathcal{I}_{L-1} + \sum_{i=0}^{L-2}(w_i - w_{i+1})\,\mathcal{I}_i, \quad (3)$$

where $\mathbf{Y}, \mathbf{Z} \in S$, and $\mathcal{I}_i$ is the intersection between the i^{th} histogram in $\Psi(\mathbf{Y})$ and $\Psi(\mathbf{Z})$, respectively [11]. 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).

To avoid favoring large sets and to form a measure that respects the triangle inequality, we will consider the pyramid match value normalized by the product of each input's self-similarity:

$$\mathcal{P}_\Delta\left(\Psi(\mathbf{Y}), \Psi(\mathbf{Z})\right) = \frac{\tilde{\mathcal{P}}_\Delta\left(\Psi(\mathbf{Y}), \Psi(\mathbf{Z})\right)}{\sqrt{\tilde{\mathcal{P}}_\Delta\left(\Psi(\mathbf{Y}), \Psi(\mathbf{Y})\right) \times \tilde{\mathcal{P}}_\Delta\left(\Psi(\mathbf{Z}), \Psi(\mathbf{Z})\right)}}. \quad (4)$$

The pyramid match will serve as our approximation to the optimal partial matching ($\hat{\pi}^*$). Below we develop an embedding for the pyramid match and the locality-sensitive hashing functions that will allow sub-linear time nearest neighbor search on top of it.

3.2. Locality Sensitive Hashing

A locality sensitive hashing scheme is a distribution on a family $\mathcal{F}$ of hash functions operating on a collection of objects, such that for two objects x, y,

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

where $sim(x, y)$ is some similarity function defined on the collection of objects [5]. In other words, the probability that two inputs collide in the hash table is equal to the similarity between them, and so highly similar objects will be indexed by the hash table with high probability. Such a hashing scheme has been shown to support efficient data structures for performing approximate nearest-neighbor queries on a database of objects, when hash functions that are appropriate for both the data objects and similarity function of interest can be defined [13, 5].

[1] Data-dependent, non-uniformly shaped bins are also possible, and may be formed by hierarchical clustering on a corpus of features [12].

3.3. Random Hyperplane Hash Functions

In [8], Goemans and Williamson provide a randomized algorithm for the MAX-CUT problem using semidefinite programming. As part of this work, they prove that given a collection of vectors $\{\vec{v}_1, \ldots, \vec{v}_n\}$ belonging to the unit sphere, and a randomly generated vector $\vec{r}$, the probability that any two vectors $\vec{v}_i$ and $\vec{v}_j$ each has a dot product with $\vec{r}$ having an opposite sign is related to the vectors as follows:

$$\Pr\left[\operatorname{sgn}(\vec{v}_i \cdot \vec{r}) \neq \operatorname{sgn}(\vec{v}_j \cdot \vec{r})\right] = \frac{1}{\pi}\cos^{-1}(\vec{v}_i \cdot \vec{v}_j). \quad (6)$$

That is, the probability a random hyperplane separates two vectors is directly proportional to the angle $\cos^{-1}(\vec{v}_i \cdot \vec{v}_j)$.

In [5], Charikar considers how this property may be exploited for locality sensitive hashing. Given a database of vectors in $\Re^d$, a vector $\vec{r}$ is chosen at random from the d-dimensional Gaussian distribution with zero mean and unit variance. The corresponding hash function $h_{\vec{r}}$ accepts a vector $\vec{u} \in \Re^d$, and is defined as:

$$h_{\vec{r}}(\vec{u}) = \begin{cases} 1, & \text{if } \vec{r} \cdot \vec{u} \geq 0 \\ 0, & \text{if } \vec{r} \cdot \vec{u} < 0 \end{cases}. \quad (7)$$

Then, drawing on the relationship in Eqn. 6, a valid locality sensitive hashing scheme is:

$$\begin{aligned} \Pr\left[h_{\vec{r}}(\vec{v}_i) = h_{\vec{r}}(\vec{v}_j)\right] &= 1 - \frac{\theta(\vec{v}_i, \vec{v}_j)}{\pi}, \text{ where} \quad (8) \\ \theta(\vec{v}_i, \vec{v}_j) &= \cos^{-1}\left(\frac{\vec{v}_i \cdot \vec{v}_j}{\sqrt{|\vec{v}_i|\;|\vec{v}_j|}}\right). \end{aligned}$$

In the following, we show that we can achieve hashing over the pyramid match kernel with this hash function family. We develop an embedding function for the pyramid mapping $\Psi(\mathbf{X})$ of point set $\mathbf{X}$ that incorporates the weights and computation of the pyramid matching $\mathcal{P}_\Delta$. When considered as a type of unary encoding, we have an embedding for each point set that under a dot product yields the un-normalized pyramid match similarity value.

Given a histogram H that contains r bins, and a weight w, let $[wH]$ denote an r-dimensional vector giving the counts in each bin of the histogram, with each count scaled by w. Note that this weighting is distributive over histogram intersection; that is, a weighted histogram intersection value is equivalent to the intersection of the weighted histograms, or $w\,\mathcal{I}\left(H(\mathbf{Y}), H(\mathbf{Z})\right) = \mathcal{I}\left([wH(\mathbf{Y})], [wH(\mathbf{Z})]\right)$.

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

$$\mathcal{U}\left([wH]\right) = \Bigg(\underbrace{\overbrace{1, \ldots, 1}^{wH^{(1)}},\ \overbrace{0, \ldots, 0}^{P - wH^{(1)}}}_{\text{first bin}}, \ldots, \underbrace{\overbrace{1, \ldots, 1}^{wH^{(r)}},\ \overbrace{0, \ldots, 0}^{P - wH^{(r)}}}_{\text{last bin}}\Bigg), \quad (9)$$

where P is the maximum possible weighted count in any histogram bin, and $H^{(j)}$ is the count in bin j of

H.[2] Let $\mathbf{v}_i(\mathbf{X})$ refer to the histogram for set $\mathbf{X}$ at pyramid level i, weighted by $w = w_i - w_{i+1}$: $\mathbf{v}_i(\mathbf{X}) = [(w_i - w_{i+1})\, H_i(\mathbf{X})]$.

The following embedding f serves to map the set of vectors $\mathbf{X}$ to a single vector:

$$f(\mathbf{X}) = \Big[\mathcal{U}(\mathbf{v}_0(\mathbf{X})),\, \mathcal{U}(\mathbf{v}_1(\mathbf{X})),\, \mathcal{U}(\mathbf{v}_2(\mathbf{X})), \ldots, \\ \mathcal{U}(\mathbf{v}_{L-2}(\mathbf{X})),\, \mathcal{U}([w_{L-1} H_{L-1}(\mathbf{X})])\Big]. \tag{10}$$

The dot product between two such encodings for sets $\mathbf{Y}$ and $\mathbf{Z}$ yields the un-normalized pyramid match score from Eqn. 3 above:

$$f(\mathbf{Y}) \cdot f(\mathbf{Z}) = \tilde{\mathcal{P}}_\Delta\left(\Psi(\mathbf{Y}), \Psi(\mathbf{Z})\right). \tag{11}$$

The length $|f(\mathbf{Y})|$ of an encoding vector $f(\mathbf{Y})$ 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:

$$\begin{aligned} |f(\mathbf{Y})| &= w_{L-1}\,|\mathbf{Y}| + \sum_{i=0}^{L-2} (w_i - w_{i+1})\,|\mathbf{Y}| \\ &= \tilde{\mathcal{P}}_\Delta\left(\Psi(\mathbf{Y}), \Psi(\mathbf{Y})\right). \end{aligned} \tag{12}$$

Putting these pieces together, we have an embedding of the pyramid match kernel that allows us to perform sub-linear time similarity search with random hyperplane hash functions. With the embedding in Eqn. 10 and the guarantee from Eqn. 8, we have:

$$\begin{aligned} \Pr\left[h_{\vec{r}}(f(\mathbf{Y})) = h_{\vec{r}}(f(\mathbf{Z}))\right] &= 1 - \frac{\theta(f(\mathbf{Y}), f(\mathbf{Z}))}{\pi}, \text{ where} \\ \theta(f(\mathbf{Y}), f(\mathbf{Z})) &= \cos^{-1}\left(\frac{f(\mathbf{Y}) \cdot f(\mathbf{Z})}{\sqrt{|f(\mathbf{Y})|\,|f(\mathbf{Z})|}}\right) \\ &= \cos^{-1}\left(\mathcal{P}_\Delta\left(\Psi(\mathbf{Y}), \Psi(\mathbf{Z})\right)\right). \end{aligned}$$

Notice that this last term is the normalized pyramid match similarity value, where normalization is done according to the product of the self-similarity scores.[3]

We do not need to explicitly expand the components $\mathbf{v}_i(\mathbf{X})$ into their unary encodings. Likewise, we do not need to generate an entry for every dimension of the random vector $\vec{r}$ in Eqn. 7 to compute a hash bit from $f(\mathbf{X})$. Instead, the counts in $H_i(\mathbf{X})$ indicate which entries in $\vec{r}$ will result in a nonzero contribution to $\langle f(\mathbf{X}) \cdot \vec{r} \rangle$, that is, those entries where the encoding for $\mathbf{v}_i(\mathbf{X})$ would be 1, not 0. For those required entries only, we generate values in $\vec{r}$ on demand: we seed a random number generator relative to the index of the particular nonzero entry in $f(\mathbf{X})$, obtain two uniform random numbers in $[0, 1]$, and then convert those to a normally distributed random number from $N(0, 1)$ using the Box-Muller transformation. The inner product between the random vector and the embedding is then the sum of those particular entries in $\vec{r}$, and the sign of this sum determines the hash key bit $h_{\vec{r}}(f(\mathbf{X}))$.

To further improve the efficiency of computing hash key bits, rather than sample random Gaussian values for $\vec{r}$ repeatedly for each unit of a total weighted bin count V (i.e., each of V 1-bits), we draw directly from the sum of V Gaussians, which is normally distributed with mean $\sum_{i=1}^{V} \mu_i = 0$ and variance $\sum_{i=1}^{V} \sigma_i^2 = V$ (hence the $\sqrt{V_l}$ term in step 5 of Algorithm 1).

3.4. Indexing in Hamming Space

Using k random hash functions (that is, k independent instances of the vector $\vec{r}$ above), for each database set $\mathbf{X}_j$ we generate a k-dimensional binary hash key string that is the concatenation of the hash key bits that result from Eqn. 7 with input $\vec{u} = f(\mathbf{X})$. Now the problem of indexing into the database with query set $\mathbf{Q}$ is reduced to hashing $f(\mathbf{Q})$ with these same k functions and retrieving items corresponding to database bit vectors having minimal Hamming distances to the query bit vector.

For this step, we employ the technique for approximate search in Hamming space developed by Charikar [5], which guarantees that at most $O(N^{1/(1+\epsilon)})$ of the N bit vectors must be examined to retrieve the $(1+\epsilon)$-approximate nearest neighbors. Given the list of database hash keys, $M = O(N^{1/(1+\epsilon)})$ 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 [5] for details.

Having pulled up these nearest bit vectors, we then compute the actual pyramid match similarity values between their associated database pyramids and the query's pyramid. The hashed neighbors are ranked according to these scores, and this ranked list is the final output of the algorithm. A useful property of our indexing approach is that adding to the database does not require recomputing the preprocessing steps; to add a new example, its hash key is computed, permuted, and then inserted into the existing sorted orders. See Algorithm 1 for a summary of our approach.

3.5. Normalized Partial Matches

To achieve a complete partial matching—where no penalty whatsoever is accumulated for unmatched features

[2] 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, as described below, the unary encoding is never explicitly computed.

[3] Similar embeddings and hash functions are possible with the "vocabulary-guided" pyramid match given in [12], 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 weights must be applied at the level of the bins instead of at the level of the pyramid resolutions.

Algorithm 1 Pyramid match hashing algorithm.

Given: Database of images $\{\mathbf{X}_1, \ldots, \mathbf{X}_N\}$ each with feature vectors $\mathbf{X}_j = \{\mathbf{x}_1, \ldots, \mathbf{x}_{m_j}\}, \mathbf{x}_i \in \Re^d$:

1: **for all** sets $\mathbf{X}_j, j = 1, \ldots, N$ **do**

2: **Compute embedding:** Compute the L-level multi-resolution histogram $\Psi(\mathbf{X}_j)$ and then weighted vector $f(\mathbf{X}_j)$, represented sparsely as $\{\langle I_l, V_l \rangle\}_{l=1}^{Z}$, a list of d-dim. nonzero indices I_l and their associated weighted counts V_l, with $Z = O(m_j L)$.

3: **Compute hash key:**

4: **for all** Hash functions $\vec{r}_i, i = 1, \ldots, k$ **do**

5: Generate next hash key bit:

$$h_{\vec{r}_i}(f(\mathbf{X}_j)) = \begin{cases} 1, & \text{if } \sum_{l=1}^{Z} \vec{r}_i^{(l)} \sqrt{V_l} \geq 0 \\ 0, & \text{otherwise} \end{cases},$$

where $\vec{r}_i^{(l)} \sim N(0,1)$ is the I_l^{th} entry in random vector $\vec{r}_i$, generated via seeds relative to i and I_l.

6: **end for**

7: Concatenate k bits to form binary hash key: $\left[h_{\vec{r}_1}(f(\mathbf{X}_j)), \ldots, h_{\vec{r}_k}(f(\mathbf{X}_j))\right]$

8: **end for**

9: **Process hash keys** for Hamming space approximate-NN search according to [5]: generate $M = O(N^{1/(1+\epsilon)})$ random k-dimensional permutations, permute all database hash keys by each one, and sort each list of permuted keys.

Given: Query image represented by set of features $\mathbf{Q}$,

10: Compute embedding $f(\mathbf{Q})$ and hash key $\left[h_{\vec{r}_1}(f(\mathbf{Q})), \ldots, h_{\vec{r}_k}(f(\mathbf{Q}))\right]$ as in 2 and 3 above.

11: Apply each permutation to query hash key bits.

12: Perform binary search on each sorted, permuted order of database hash keys, and collect the indices $[t_1, \ldots, t_{2M}]$ corresponding to the database items' hash keys that are indexed in each.

13: Sort hashed examples according to $\mathcal{P}_\Delta\left(\Psi(\mathbf{Q}), \Psi(\mathbf{X}_{t_i})\right)$, for $i = 1, \ldots, 2M$.

in a larger input set—it is necessary to normalize the matching cost only according to the size of the smaller set. However, the hashing described above makes use of a normalization factor that includes the sizes of *both* input sets. This yields a correspondence measure between two variably-sized sets that does include some penalty for the unmatched points in the larger set, but remains robust to increasingly distant outlier features.

For example, consider two sets; with the minimum cardinality normalization, their pyramid match score would remain constant if we were to add more and more features to one of the sets. In contrast, with the product normalization, the pyramid match value would slowly decrease as we added those features. When is this a desired property for image matching? If there is expected to be an unknown amount of clutter, background, or unmatched features in both of the images being matched, this normalization is reasonable. The best matching will be the one that can find good matches for all the features in both sets. An image matching with more clutter (unmatchable features) will receive a lower similarity weight than an image matching with fewer unmatched features. However, pyramid match hashing will not care *how different* the unmatched features are to any features in the other set; that is, the penalty is only relative to *how many* unmatched features there are. We verify this property experimentally in Section 4.

Which normalization approach is most suitable may depend on the application. We have shown how to perform sub-linear time hashing with the product normalization, and in Chapter 5 of [9] we prove that it is not possible to do locality sensitive hashing with the alternative minimum cardinality normalization.

4. Results

In this section we evaluate our indexing technique in several ways. We first systematically test the pyramid match's robustness to outlier features, and compare it against an alternate approximate matching approach. Then we demonstrate pyramid match hashing applied to image retrieval for two different data sets.

Robust Matching: The degree to which the unmatchable (or "outlier") features differ from the matched features will not affect our matching scores, meaning that pyramid match hashing is robust to increasingly distant outlier features. In contrast, bijective matchings as in [14, 10] computed over sets that have been re-weighted to achieve total equal masses are not robust to increasingly distant outliers. These properties hold by definition (see [9], Ch. 5 for an illustrative figure), and we verify them empirically here.

In order to work with realistic data but still have control over the amount of clutter features, we established synthetic class models. Each model is comprised of some fixed number m' of parts, and each part has a Gaussian model that generates its d-dimensional appearance vector (in the spirit of the "constellation model" used by Fergus et al. [7] and others). Given these category models, we can then add clutter features and noise, simulating in a controlled manner the variations that occur with the patches extracted from real images. The appearance of the clutter features is determined by selecting a random vector from a uniform distribution whose range is set to be a function of the model feature values' range (see below).

We generated 50 examples for two synthetic category models, each of which was defined by a set of $m' = 35$ features with $d = 2$, for a total of 100 point sets. We computed pairwise similarities using the pyramid match normalized by the product of the input sets' cardinalities, pairwise similarities using the optimal partial matching and the same normalization, and pairwise distances based on a bijective matching approximation. To apply the bijective matching to unequally-sized sets, points in a set were weighted so that all weights summed to one.

Then we added to every set up to 100 clutter features having a value range bounded by a percentage Q of the inlier features' value range, and re-computed the resulting pairwise matching scores. We tested for values of Q ranging from 100% to 900%, in increments of 200. (When

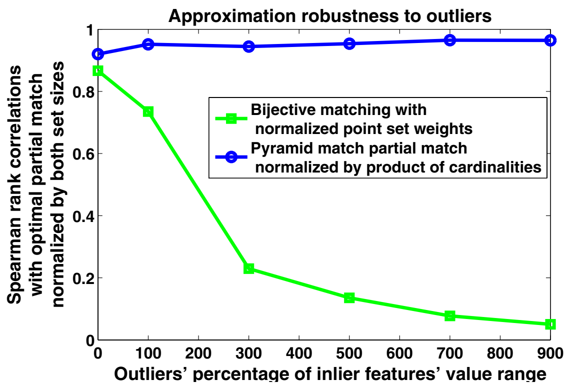


Figure 2. A partial match normalized by the product of the sizes of both input sets will remain robust to distant outlier clutter features (blue circles). This property cannot be simulated by re-weighting point sets and computing a bijective matching (green squares).

$Q = 100\%$, the inlier and outlier features have the same value range.) Figure 2 shows the results, with the approximations' ranking quality quantified by the Spearman correlation coefficient. The two left-most points on the plot correspond to matchings with equally-sized sets and no clutter. The remaining points correspond to matchings with increasingly more distant clutter or outlier features. The match scores normalized by the sizes of both input sets remain robust to the addition of stronger outlier features (blue circles), whereas the bijective matching must incorporate the distance of the outlier features in its matching and suffers as that distance increases (green squares).

Image Retrieval Experiments: Next we demonstrate pyramid match hashing applied to content-based image retrieval where images are represented by sets of local SIFT [18] image features. We consider two different data sets: the Caltech-4 database and the Caltech-101 database. In all experiments, we set $\epsilon = 1.0$, which means that our query times are bounded by $O(\sqrt{N})$ for N images.

We measure our performance with several metrics: (1) the observed accuracy of the approximate-NN indexing, (2) the extent to which our hash functions are in practice locality-sensitive to the pyramid match, (3) the ranking of hashed database neighbors relative to the ranking we would obtain with a linear scan of all items, and (4) the relevance of examples retrieved via hashing, again relative to the results of a linear scan. For metrics (1) and (3) we display results with 'box and whisker plots': each box has lines at the lower quartile, median value (red line), and upper quartile values, whiskers extend from each end of the box to show the extent of the rest of the data, and outliers are denoted with pluses. For metrics (2) and (4) we summarize the error/accuracy distributions in terms of their means and standard deviations (denoted by σ).

To measure the approximate-NN indexing accuracy (1), we measure how frequently we obtain some $(1 + \epsilon)$-

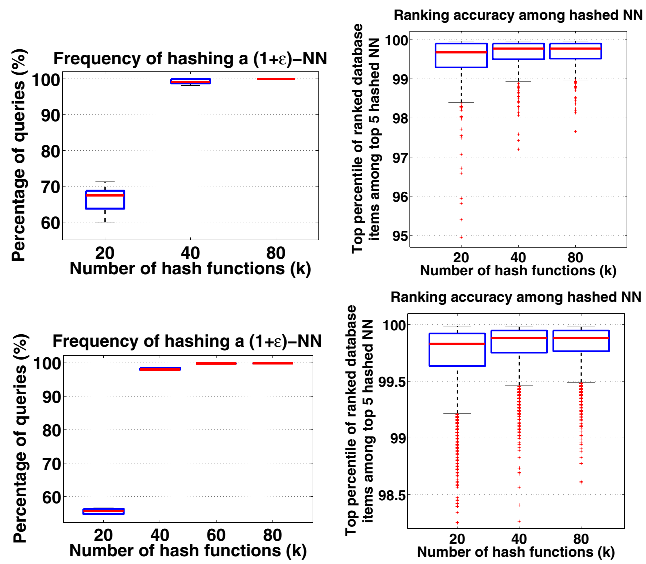


Figure 3. Image retrieval results for Caltech-4 (top row) and Caltech-101 (bottom row) databases. Left plots measure extent to which approximate-NN guarantee is realized; right plots show ranking quality of hashed NN relative to a linear scan of the entire database.

neighbor for each query. That is, we count how often we hash to one (or more) database hash keys that are within $(1 + \epsilon)C$ of the query, if the true nearest item is at distance C from it. To measure the hash function accuracy (2), we compute the error $Pr(h_{\vec{r}}(f(\mathbf{X})) = h_{\vec{r}}(f(\mathbf{Y}))) - \left(1 - \frac{\cos^{-1}(\mathcal{P}_{\Delta}(\Psi(\mathbf{X}), \Psi(\mathbf{Y})))}{\pi}\right)$, for all queries $\mathbf{X}$ and all database examples $\mathbf{Y}$. The probability of two sets having equal hash key bits is estimated by averaging over 80 random hash functions. The ranking quality (3) is computed in terms of the top percentile among the top K hashed NN (according to the ranking a linear scan would provide). The relevance of hashed NN (4) is measured by the ratio of the number of top K hashed NN having the same label as the query divided by the same count for the NN according to a linear scan; a ratio of 1 means the hashing retrieves at least as many relevant examples as the linear scan. We set $K = 5$. All results are collected for five repeated runs, due to the random elements of the algorithm. Figure 3 displays results for both databases using metrics (1) and (3).

Note that for these last two metrics, exhaustive search using the pyramid match is our baseline because the method we have proposed is meant to approximate the quality of such a search at a small fraction of the cost. Our implementation of the pyramid match requires on average 0.1 ms to compare two sets averaging 1400 features each, on a machine with a 2.4 GHz processor and 2 GB of memory. We have previously proven bounds on the pyramid match's error relative to the optimal partial match [9], and have shown that its accuracy in practice is very close to optimal [11, 9], and so we do not focus our results on this aspect.

The Caltech-4 database contains 3,188 total images spanning four different categories of objects. We withheld 20 images from each category to query the remaining images. The approximate-NN accuracy (top left plot) is very strong in practice here, with nearly a 100% chance of fulfilling the $(1 + \epsilon)$ guarantee when $k \geq 40$. As expected, a larger number of hash functions provides better accuracy. The distribution of errors between the hash function bit agreement and the pyramid match scores (metric 2) has a mean of -0.01 ($\sigma = 0.04$), again verifying our method's theoretical properties for this data. The top right plot demonstrates that the hashed NNs match the quality of the NNs obtained with a linear scan very well, with ranking percentiles at median values of 99.8. The mean relevance ratio is 0.97 ($\sigma = 0.12$) and the median ratio is 1.0 for the closest 5 neighbors. On average, a query with pyramid match hashing required searching only 79 images, or 2.5% of the database. Thus, our pyramid match hashing algorithm greatly improves the efficiency of partial match search with very little loss in accuracy.

The Caltech-101 database contains 8,677 images spanning 101 object categories. Because there are only 30 images in some categories, we withheld 10 images from each class to use as queries on the rest of the database. For this data, an average query required searching only 115 images, or 1.5% of the database. The realized approximate-NN accuracy follows a similar trend as above, with nearly perfect satisfaction of the indexing guarantee for 40 hash bits or more (bottom left plot). The mean hash function error (metric 2) is 0 ($\sigma = 0.03$); this again is evidence that the relationship between the pyramid match and our hash functions holds in practice. The ranking quality of the pyramid match hashing relative to the linear scan is high on this data, with median percentiles of 99.9 for 20 to 100 hash functions (bottom right plot). The mean ratio of relevant examples retrieved with hashing versus a linear scan is 0.76 ($\sigma = 0.4$), and the median value is 1.0 for this data. This distribution is wider than it was for the Caltech-4 data, suggesting that the large number of categories makes the retrieval of all relevant examples more challenging. Still, on average 76% of relevant examples found in the top 5 NN with a linear scan are also found by the hashing retrieval.

For both data sets, using more hash functions improves the indexing accuracy because it increases the probability that similar examples collide; however this accuracy comes at the cost of a linear increase in hash key computation time. Overall, we lose very little accuracy (Figure 3) but achieve average speedup factors of 11 (for N=3,188) and 20 (for N=8,677) per query, even including the overhead required to compute hash keys. This empirical evidence together with our theoretical bounds indicate the trend in speedups that will continue for even larger databases.

5. Conclusions

We have developed a sub-linear time randomized hashing method that enables scalable search over a normalized partial matching for very large databases. We have demonstrated our approach on retrieval tasks for images represented by sets of local appearance features, and we have analyzed its accuracy and theoretical guarantees in various ways. Nothing about the method is specific to a given representation; it can be applied in any case where it is useful to index sets of feature vectors based on their correspondence. In the future we are interested in exploring how pyramid match hashing might have impact on large-scale recognition or clustering problems where a traditional linear processing of the training data would not be feasible.

References

[1] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. BoostMap: A Method for Efficient Approximate Similarity Rankings. In *CVPR*, 2004.

[2] J. Beis and D. Lowe. Shape Indexing Using Approximate Nearest-Neighbour Search in High Dimensional Spaces. In *CVPR*, 1997.

[3] S. Belongie, J. Malik, and J. Puzicha. Shape Matching and Object Recognition Using Shape Contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(24):509–522, 2002.

[4] A. Berg, T. Berg, and J. Malik. Shape Matching and Object Recognition Low Distortion Correspondences. In *CVPR*, 2005.

[5] M. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *ACM Symp. on Theory of Computing*, 2002.

[6] P. Felzenszwalb and D. Huttenlocher. Pictorial Structures for Object Recognition. *IJCV*, 61(1), 2005.

[7] R. Fergus, P. Perona, and A. Zisserman. Object Class Recognition by Unsupervised Scale-Invariant Learning. In *CVPR*, 2003.

[8] M. Goemans and D. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *JACM*, 42(6):1115–1145, 1995.

[9] K. Grauman. *Matching Sets of Features for Efficient Retrieval and Recognition*. PhD thesis, MIT, 2006.

[10] K. Grauman and T. Darrell. Fast Contour Matching Using Approximate Earth Mover's Distance. In *CVPR*, 2004.

[11] K. Grauman and T. Darrell. The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features. In *ICCV*, 2005.

[12] K. Grauman and T. Darrell. Approximate Correspondences in High Dimensions. In *NIPS 19*. 2007.

[13] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *30th Symposium on Theory of Computing*, 1998.

[14] P. Indyk and N. Thaper. Fast Image Retrieval via Embeddings. In *Intl Wkshp on Stat. and Comp. Theories of Vision*, 2003.

[15] S. Lazebnik, C. Schmid, and J. Ponce. Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In *CVPR*, 2006.

[16] M. Leordeanu and M. Hebert. A Spectral Technique for Correspondence Problems using Pairwise Constraints. In *ICCV*, 2005.

[17] V. Lepetit, P. Lagger, and P. Fua. Randomized Trees for Real-Time Keypoint Recognition. In *CVPR*, 2005.

[18] D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *IJCV*, 60(2), 2004.

[19] D. Nister and H. Stewenius. Scalable Recognition with a Vocabulary Tree. In *CVPR*, 2006.

[20] S. Obdrzalek and J. Matas. Sub-linear Indeing for Large Scale Object Recognition. In *BMVC*, 2005.

[21] G. Shakhnarovich, P. Viola, and T. Darrell. Fast Pose Estimation with Parameter-Sensitive Hashing. In *ICCV*, 2003.

[22] H. Shao, T. Svoboda, V. Ferrari, T. Tuytelaars, and L. V. Gool. Fast Indexing for Image Retrieval Based on Local Appearance with Reranking. In *ICIP*, 2003.