QsRank: Query-sensitive Hash Code Ranking for Efficient ϵ-neighbor Search

Xiao Zhang
Tsinghua University
andypassion@gmail.com

Lei Zhang
Microsoft Research Asia
leizhang@microsoft.com

Heung-Yeung Shum
Microsoft Corporation
hshum@microsoft.com

Abstract

Although binary hash code-based image indexing methods have been recently developed for large-scale applications, the problem of ranking *such hash codes has been barely studied. In this paper, we propose a query sensitive ranking algorithm (*QsRank*) to rank PCA-based hash codes for the ϵ-neighbor search problem. The QsRank algorithm takes the target neighborhood radius ϵ and the raw feature of a given query as input, and models the statistical properties of the target ϵ-neighbors in the space of hash codes. Unlike the Hamming distance, the proposed algorithm does* not *compress query points to hash codes. Therefore, it suffers less information loss and is more effective than Hamming distance-based approaches. Based on the QsRank method, we developed an efficient indexing structure and retrieval algorithm for large-scale ϵ-neighbor search. Evaluations on two datasets of 10 million web images and 10 million SIFT descriptors demonstrate that the proposed retrieval system achieves higher accuracy with less memory cost and faster speed.*

1. Introduction

In this paper, we study the problem of large-scale approximate ϵ-neighbor search with Euclidean distance. The problem is defined as follows: given a set of data points $\{\mathbf{x}_i | i = 1, \ldots, n\}$, and a query $\mathbf{q}$, retrieve all points $\mathbf{x}_i$ such that $||\mathbf{q} - \mathbf{x}_i||_2 < \epsilon$. The ϵ-neighbor search continues to prove itself as an fundamental problem in many fields of computer vision such as matching image patches[2], object recognition[9, 10], near-duplicate image search[16] and data-driven image understanding[18, 14] where semantic of an image is inferred from its ϵ-neighbors. Besides, ϵ-neighbor search is also an important problem in many other areas of computer science such as graphics, geographic information system, CAD, database and etc..

To efficiently perform ϵ-neighbor search in a large-scale dataset, a number of methods have been proposed such as KD-Tree[3] and locality sensitive hashing (LSH)[1]. Recently, a new group of approaches[19, 21, 4, 15, 13, 17, 5, 7] have emerged which embed input data to the binary hash code space while preserving a given similarity metric of interest. These approaches have received increasingly more attention because they work well in high dimensional spaces and enable large efficiency gains in terms of storage cost and computation speed.

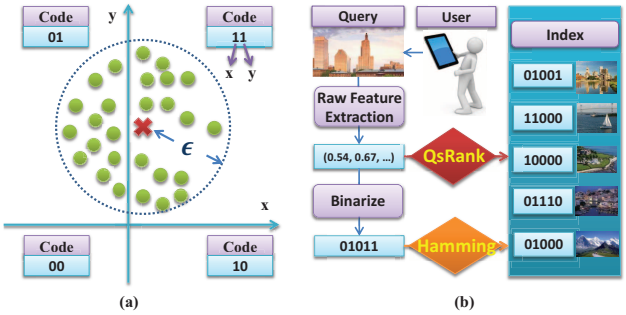


Figure 1. Figure (a) provides an example where Hamming distance fails to generate the best hash code ranking. Figure (b) illustrates that the proposed QsRank does not convert query to hash code when compute the ranking score, which is a key difference between QsRank and Hamming distance.

Currently, the majority of research efforts have been devoted to the construction of compact binary hash codes. However, few works have addressed the problem of ranking such hash codes. Most existing algorithms rank hash codes by their hamming distance to the queries. That is, the data points within the Hamming-ball of a given query in the binary hash code space will be returned. Although this approach is widely used, it is incapable of accurately ranking the hash codes because of the information loss during the binarization process, especially the ignorance of the query raw feature. Consider a simple scenario where the raw features of the data points are two dimensional vectors and the binarization is done by thresholding each dimension around zero, as illustrated in Fig. 1(a). In this figure, the query point marked by a red X is mapped to "11" and the green circles are the target ϵ-neighbors to retrieve. The data points with both hash code "01" and "10" have the same hamming distance to the query point. However, it is straightforward to see that the hash code "01" should be ranked much high-

er because there are many target neighbors with hash code "01" but *no* target neighbors mapped to "10". Therefore the hamming distance fails to give us a reasonable hash code ranking in this example.

In this paper, we propose a novel query sensitive ranking method (**QsRank**) for PCA-based hash codes[16, 4]. The QsRank algorithm takes the target neighborhood radius ϵ and the raw feature of a given query as input parameters and models the statistical properties of the target ϵ-neighbors in the hash code space. Unlike Hamming distance , the QsRank algorithm *does not convert queries to binary hash codes* when computing the ranking score, as illustrated in Fig. 1(b). Thus the resulted ranking score is more effective than hamming distance and provides better flexibility to adjust the balance between accuracy and efficiency.

To prove its effectiveness, we implemented a ϵ-neighbor search system with the QsRank algorithm. To achieve a faster retrieval speed, we adopted a hash table-based index structure. In the index, data points are grouped into hash buckets by short binary codes (16 bits). Given a query, the QsRank algorithm is leveraged to determine the probing sequence of the hash buckets. Only the data points located in the hash buckets with highest QsRank scores will be retrieved. Then reranking is performed to rank the retrieved data points using longer binary codes. In the evaluation on two datasets of 10 million images and SIFT descriptors, the proposed system achieves a superior accuracy than state-of-the-art indexing systems with less memory cost and faster retrieval speed.

The paper is organized as follows. The related work is discussed in Section 2. The generation of binary hash codes is discussed in Section 3. The QsRank algorithm is discussed in Section 4. The discussion on the proposed image search system using QsRank is provided in Section 5.The evaluation of the image search system on two large-scale datasets is discussed in Section 6. We conclude the paper and discuss future work in Section 7.

2. Related Work

Our work is motivated by the recent success of data-driven image understanding approaches. As shown in the Tiny Image project [14], accurate image recognition can be achieved by simple nearest-neighbor search in a database of 80 million tiny images. More recently, the Arista system [18] demonstrated that accurate and diverse tags can be labeled to images by mining the surrounding texts of near-duplicate images in a 2 billion image database.

To efficiently perform ϵ-neighbor search in such large-scale datasets, a number of methods have been proposed with compact data representation and fast matching. The earliest such methods are tree based spatial partition algorithms, such as KD-Tree[3]. A recent solution with wide popularity is locality sensitive hashing (LSH)[1]. LSH first projects data items by random vectors sampled from p-stable distributions and then discretize the projected value into a number of bins. Theoretical analysis guarantees that ϵ-neighbors have higher probability to be hashsed into the same bin with query than other items in the database.

More recently, a new group of approaches have emerged which embed input data to the binary hash code space while preserving a given similarity metric of interest. Torralba et al.[15] studied the binary coding problem and compared several methods to generate distance-preserving binary codes based on boosting, restricted Boltzmann machines , and locality sensitive hashing (LSH). To further improve the performance and scalability, Weiss et. al. have proposed Spectral Hashing (SH)[19], a method motivated by spectral graph partitioning. Kerui et al[8] presented a simple algorithm for generating compact binary representations (CP) of imagery data based on random projections. Their analysis gave the explicit bound on the number of bits needed to effectively solve the indexing problem. Gong et al.[4] proposed to generate binary hash codes by finding a rotation of zero-centered data so as to minimize the quantization error of mapping the data to the vertices of binary hypercube (ITQ). Wang et al.[17] have proposed a semi-supervised hashing method (SSH) that incorporates pairwise semantic similarity and dissimilarity constraints from labeled data.

To rank the generated binary hash codes given a query, most research works just use Hamming distance, which is fast but unable to distinguish between the relative importance of different bits. Recently, Jiang et al.[6] proposed a query-adaptive Hamming distance which assigns dynamic weighting for Hamming distance such that each bit is treated differently. There are two key difference between their method and QsRank. First, the query-adaptive Hamming distance still converts queries to hash codes during the ranking process while QsRank does not. Second, to compute the dynamic weighting, the query-adaptive Hamming distance assumes that classifiers can be trained to map queries into a set of pre-defined semantic classes, while QsRank does not make this assumption.

3. Generating Hash Codes by PCA

3.1. Computation

In this section, we introduce how we generate hash codes based on PCA. Denote the input data points as $\{\mathbf{x}_i \in \mathbb{R}^d, i = 1, \ldots, n\}$ that forms the rows of the data matrix $X \in \mathbb{R}^{n \times d}$. We assume the points are zero-centered, i.e., $\sum_{i=1}^{n} \mathbf{x}_i = 0$. Our goal is to generate a binary hash code matrix $H \in \{-1, 1\}^{n \times d}$. For each bit $j = 1, \ldots, d$, the hash encoding function is defined by

$$\begin{aligned} y_{ij} &= \mathbf{x}_i \mathbf{w}_j \\ H_{ij} &= sgn(y_{ij}) \end{aligned} \tag{1}$$

, where $\mathbf{w}_j \in \mathbb{R}^d$ is a projection vector, y_{ij} is the projected value and H_{ij} is the resulted hash code. Here, $sgn(y_{ij}) = 1$ if $y_{ij} \geq 0$ or -1 otherwise. In this way, we can formulate the encoding process as $H = sgn(XW)$, where $W \in \mathbb{R}^{d*d}$. The columns of W correspond to the eigenvectors of the data co-variance matrix $X^T X$. In our implementation, only *the first k columns* of W are used corresponding to the largest eigenvalues of $X^T X$.

3.2. Benefit

In this work, the main reasons for choosing PCA to generate hash code are three-folds.

- Firstly, the PCA compress most of the information in the top k dimensions, meaning that excellent retrieval results can be achieved with shorter hash codes and enable us to build an efficient hash table-based index structure.
- Secondly, the PCA projection is orthogonal and preserves the L2 distance. Therefore the ϵ-neighbors of a given query remain the same after the PCA projection. Furthermore, PCA projection minimizes the reconstruction error with the top k dimensions. Therefore the ϵ-neighborhood of a given query in the top k dimensions of PCA projected space provides a good approximation to the target ϵ-neighborhood in the raw feature space.
- Finally, the PCA projected values of a data point are uncorrelated. This enable us to develop an efficient algorithm to compute QsRank, which will be detailed in the next section.

4. Ranking Hash Codes by QsRank

In this section, we discuss the QsRank algorithm. First, we make a number of notations to facilitate the discussion.

4.1. Notations

Denote a PCA projected data point as $\mathbf{y} \in \mathbb{R}^d$, the PCA projected query as $\mathbf{q} \in \mathbb{R}^d$, its ϵ-neighborhood in the PCA projected space as

$$NN(\mathbf{q}, \epsilon) = \{\mathbf{y} | ||\mathbf{q} - \mathbf{y}||_2 < \epsilon\} \quad (2)$$

Since PCA projection preserves L2 distance, $NN(\mathbf{q}, \epsilon)$ is the same as the ϵ-neighborhood of the query before PCA projection. Denote $\mathbf{y}^k$ as a vector of the top k dimensions of $\mathbf{y}$ and similarly define $\mathbf{q}^k$ for $\mathbf{q}$. Finally, denote a hash code as:

$$\mathbf{h} = (h_1, \ldots, h_d), h_j \in \{-1, 1\}, j = 1, \ldots, d \quad (3)$$

Denote the set of data points mapped to $\mathbf{h}$ as

$$S(\mathbf{h}) = \{\mathbf{y} | h_j y_j > 0, j = 1, \ldots, d\} \quad (4)$$

With the notations, we proceed to introduce the QsRank algorithm.

4.2. QsRank Definition

Assume that $\mathbf{y}$ is generated from a distribution with pdf $p(\mathbf{y})$. Then given a query $\mathbf{q}$, the QsRank algorithm assigns a ranking score to a hash code $\mathbf{h}$ as follows:

$$QsRank(\mathbf{q}, \mathbf{h}, \epsilon) = \frac{\int_{\mathbf{y} \in \{S(\mathbf{h}) \cap NN(\mathbf{q}, \epsilon)\}} p(\mathbf{y}) d\mathbf{y}}{\int_{\mathbf{y} \in NN(\mathbf{q}, \epsilon)} p(\mathbf{y}) d\mathbf{y}} \quad (5)$$

One can see that the denominator of Eq. 5 equals the probability of $\mathbf{y}$ being an ϵ-neighbor of a query $\mathbf{q}$, and that the numerator equals the joint probability of $\mathbf{y}$ being mapped to a hash code $\mathbf{h}$ and $\mathbf{y}$ being an ϵ-neighbor of the query $\mathbf{q}$. Based on the Bayes rule, QsRank equals the conditional probability of $\mathbf{y}$ being mapped to the hash code $\mathbf{h}$ given that $\mathbf{y}$ is an ϵ-neighbor of the query $\mathbf{q}$. The probabilistic interpretation of QsRank definition can be formulated as follows:

$$\begin{aligned} QsRank(\mathbf{q}, \mathbf{h}, \epsilon) &= \frac{P(\mathbf{y} \in S(\mathbf{h}), \mathbf{y} \in NN(\mathbf{q}, \epsilon))}{P(\mathbf{y} \in NN(\mathbf{q}, \epsilon))} \\ &= P(\mathbf{y} \in S(\mathbf{h}) | \mathbf{y} \in NN(\mathbf{q}, \epsilon)) \quad (6) \end{aligned}$$

Based on Eq. 6, QsRank equals the (expected) percentage of the ϵ-neighbors of the query $\mathbf{q}$ that are mapped to the hash code $\mathbf{h}$. Therefore, QsRank measures the *expected recall loss* the search system suffers if data points mapped to $\mathbf{h}$ are not retrieved.

Here, any probability density function can be used for $p(\mathbf{y})$. The choice should be made based on the problem to solve. Next, we discuss an approximation algorithm to compute QsRank efficiently.

4.3. Approximation Algorithm

To efficiently compute QsRank, we first replace $NN(\mathbf{q}, \epsilon)$ by $NN(\mathbf{q}^k, \epsilon)$, which is the ϵ-neighborhood of $\mathbf{q}$ in the top k dimensions. This is a reasonable approximation because PCA projection minimize the reconstruction error with the top k projections. Then, we replace the ϵ-ball by a hypercube(HC) centered on $\mathbf{q}^k$, which is:

$$HC(\mathbf{q}^k, \epsilon) = \{\mathbf{y}^k | |q_j - y_j| < \epsilon, j = 1, \ldots, k\} \quad (7)$$

Although the overlap of $HC(\mathbf{q}^k, \epsilon)$ and $NN(\mathbf{q}^k, \epsilon)$ is much smaller in high dimensional space, we find this approximation acceptable in our application when k is small. Finally, since PCA projections are uncorrelated, we assume that each dimension of $p(\mathbf{y})$ in Eq.5 is independent, i.e. $p(\mathbf{y}) = \prod_{j=1}^{d} p_j(y_j)$. With above approximations, we are able to derive the following formula to compute QsRank:

$$QsRank(\mathbf{q}, \mathbf{h}, \epsilon) \approx \prod_{j=1}^{k} w_j(q_j, h_j, \epsilon) \quad (8)$$

N^ϵ	$N^\epsilon_{x>0}$	$N^\epsilon_{x<0}$	$N^\epsilon_{y>0}$	$N^\epsilon_{y<0}$
27	15	12	27	0

Table 1. Number of ϵ-neighbors (green circles) in Fig. 1(a).

Here, $w_j(q_j, h_j, \epsilon), j = 1, \ldots, k$ is the *single-bit output* of QsRank, and is computed as follows:

$$\begin{aligned} w_j(q_j, h_j, \epsilon) &= \frac{P(h_j y_j > 0, |q_j - y_j| < \epsilon)}{P(|q_j - y_j| < \epsilon)} \\ &= \frac{\int_{h_j y_j > 0, |q_j - y_j| < \epsilon} p_j(y_j) dy_j}{\int_{|q_j - y_j| < \epsilon} p_j(y_j) dy_j} \end{aligned} \quad (9)$$

Here, y_j is a random number generated from $p_j(y_j)$. In our experiments, we found that a uniform distribution assumption for $p_j(y_j)$ works quite well for efficiently computing $w_j(q_j, h_j, \epsilon), j = 1, \ldots, k$. Finally, we provide an example of computing $w_j(q_j, h_j, \epsilon)$ when $-\epsilon < q_j < \epsilon$ and $h_j = +1$:

$$w_j(q_j, +1, \epsilon) = \frac{\int_0^{q_j+\epsilon} p_j(y_j) dy_j}{\int_{q_j-\epsilon}^{q_j+\epsilon} p_j(y_j) dy_j} \quad (10)$$

4.4. Compare with Hamming Distance

A closer look at Eq.9 reveals four differences between QsRank and Hamming distance.

1. QsRank does not binarize the query when computing the ranking score.

2. Hamming distance does not take the target neighborhood radius into consideration, whereas QsRank explicitly considers ϵ as a parameter.

3. For the j-th bit, Hamming distance uses a binary value to represent the difference between a query and a data point. However, QsRank considers the unbinarized q_j and computes a real number $w_j(q_j, h_j, \epsilon) \in [0, 1]$ to get the final ranking score.

4. The final results of Hamming distance is a *summation* of every single-bit output, while QsRank is the *product* of every single-bit output. It means QsRank is zero if one of its single-bit output is zero. This is reasonable because $\forall j, |q_j - y_j| > \epsilon$ is a sufficient condition for $||\mathbf{q} - \mathbf{y}||_2 > \epsilon$. Therefore, this property makes QsRank a more appropriate ranking algorithm for the ϵ-neighbor problem.

Previously we provide an example of the failure of Hamming distance in Fig. 1(a). We use this example again in this section to further compare QsRank and Hamming distance. Recall that the green circles in Fig. 1(a) are the ϵ-neighbors we want to retrieve. Denote N^ϵ as the number of all ϵ-neighbors, $N^\epsilon_{x>0}$ as the number of ϵ-neighbors with positive value along the X axis, and similarly denote $N^\epsilon_{x<0}, N^\epsilon_{y>0}, N^\epsilon_{y<0}$. A simple counting in Fig. 1(a) leads to the result in Tab. 1. Denote the query point as $\mathbf{q} = (q_x, q_y)$. Then based on Eq. 6 and Eq. 9, we have:

Code	h_x	h_y	**QsRank**	**Hamming**
11	+1	+1	$0.556 \times 1 = \mathbf{0.556}$	$0 + 0 = \mathbf{0}$
01	-1	+1	$0.444 \times 1 = \mathbf{0.444}$	$1 + 0 = \mathbf{1}$
10	+1	-1	$0.556 \times 0 = \mathbf{0}$	$0 + 1 = \mathbf{1}$
00	-1	-1	$0.444 \times 0 = \mathbf{0}$	$1 + 1 = \mathbf{2}$

Table 2. The computation of QsRank and Hamming distance for the four hash codes in Fig. 1(a). h_x and h_y are the single-bit hash codes along the X axis and Y axis respectively. The single-bit output of QsRank comes from Eq. 11.

$$\begin{aligned} w_x(q_x, +1, \epsilon) &= N^\epsilon_{x>0}/N^\epsilon \approx 0.556 \\ w_x(q_x, -1, \epsilon) &= N^\epsilon_{x<0}/N^\epsilon \approx 0.444 \\ w_y(q_y, +1, \epsilon) &= N^\epsilon_{y>0}/N^\epsilon = 1 \\ w_y(q_y, -1, \epsilon) &= N^\epsilon_{y<0}/N^\epsilon = 0 \end{aligned} \quad (11)$$

The single-bit output of QsRank in the above equation is further used in Tab. 2 to compute the final QsRank score for the four hash codes in Fig. 1(a).

Several observations can be obtained from Fig. 1(a), Eq. 11 and Tab. 2. First, one can see that in the X axis of Fig. 1(a), the query is very close to the binarization boundary (i.e. q_x is very close to zero), thus its ϵ-neighbors are almost equally probable to be mapped to hash code $+1$ or -1 in this dimension. As a result, the values of $w_x(q_x, h_x, \epsilon)$ in Eq. 11 for $h_x = +1$ and $h_x = -1$ are very similar, which is more reasonable than the Hamming distance(since $q_x > 0$, the hamming distance along the X axis equals 0 between query and $h_x = +1$, and 1 between query and $h_x = -1$). Second, it can be observed that the distance between query and the binarization boundary in Y axis is larger than ϵ. Therefore any data point with a negative value on Y axis *can not* be an ϵ-neighbor of the query. As shown in Tab. 2, the overall QsRank for such data points is zero, meaning that they will not be retrieved. This demonstrates the reason why QsRank is equal to the "product" rather than the "summation" of every single-bit result.

4.5. Efficient Computation

To rank each data point in a more efficient way, we compute the log QsRank as follows:

$$\log(QsRank(\mathbf{q}, \mathbf{h}, \epsilon)) = \sum_{j=1}^{k} \log(w_j(q_j, h_j, \epsilon)) \quad (12)$$

Given a query $\mathbf{q}$, a two-step approach is adopted to compute the log QsRank for the data points in a database. The first step is a *precomputation* step. In this step, two vectors

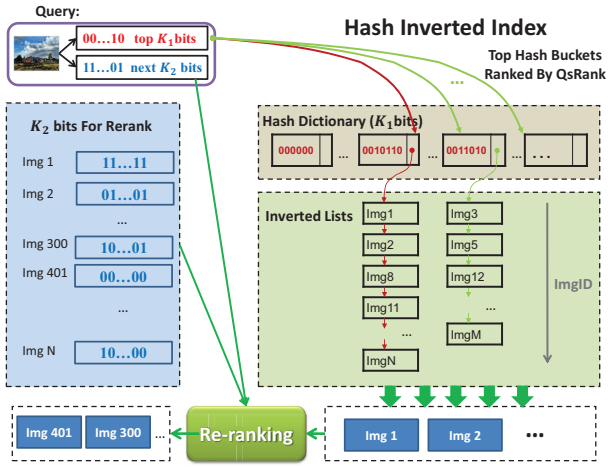


Figure 2. Index Structure with PCA Hash Code and QsRank.

$\mathbf{w}^+(q_j,\epsilon)$ and $\mathbf{w}^-(q_j,\epsilon)$ are computed:

$$w_j^+(q_j,\epsilon) = \log(w_j(q_j,+1,\epsilon)) \quad (13)$$

$$w_j^-(q_j,\epsilon) = \log(w_j(q_j,-1,\epsilon)) \quad (14)$$

Here, $w_j(\cdot)$ is defined in Eq.9. In the second step, the log QsRank of a data point with hash code $\mathbf{h}$ is computed according to Eq.12. Since we have already computed all the possible values for $log(w_j(q_j,h_j,\epsilon))$, we only need to perform a look-up operation as follows:

$$\log(w_j(q_j,h_j,\epsilon)) = \begin{cases} w_j^+(q_j,\epsilon) & \text{if } h_j = +1 \\ w_j^-(q_j,\epsilon) & \text{if } h_j = -1 \end{cases} \quad (15)$$

and then get the sum of $\log(w_j(q_j,h_j,\epsilon)), j = 1,\ldots,k$. Note that if any of the $w_j(q_j,h_j,\epsilon))$ is zero, then $QsRank(\mathbf{q},\mathbf{h},\epsilon)$ is zero and the data points with the hash code $\mathbf{h}$ will not be retrieved.

The time complexity of the precomputation step is $O(k)$ and independent of database size. And since we compute the log QsRank, only additions need to be performed in the second step, instead of multiplications. Furthermore, the second step just needs to be performed for hash codes with non-zero $w_j(q_j,h_j,\epsilon)), j = 1,\ldots,k$, which accounts for only 15% of the total data points in our experiments.

5. Scalable Indexing with QsRank

As shown in Fig. 2, we adopt a hash-based index structure. For each data point, we generate $K_1 + K_2$ bits binary code. The top K_1 bits are used to group the data points with the same binary code into a hash bucket. Therefore, each hash bucket in the index is associated with a binary hash code of K_1 bits, and the total number of hash buckets is 2^{K_1}. The next K_2 bits are also stored for reranking.

Given a query, we first compute QsRank for the hash codes associated with the hash buckets. In this way, a rank list of all the hash buckets is generated, based on which the top L hash buckets are probed. Then all the data points corresponding to the top ranked hash buckets are retrieved which comprise a candidate set. Next we represent the data points in the candidate set using binary codes of $K_1 + K_2$ bits and generate the final rank list by QsRank. Denote the candidate set size as M. The time complexity for retrieval is $O(2^{K_1}) + O(M)$, which is much smaller than linear scan based methods as demonstrated in the experiments.

5.1. The flexibility of choosing L

In the proposed index system, L (i.e. the number of hash buckets to probe) is an important parameter which balances the recall and the retrieval speed. As discussed previously, for a Hamming distance-based system, data points within the Hamming-ball of a given query in the hash code space will be returned. Therefore, the setting of L is limited by the radius R of the target Hamming-ball. For example, for $R = 0$, L needs to be set to $\binom{K_1}{0} = 1$ meaning that only the hash bucket that query mapped to will be probed. Here, K_1 is the code length of a hash code associated with a hash bucket. For $R = 1$, L needs to be set to $\binom{K_1}{0} + \binom{K_1}{1} = 1 + K_1$ meaning that only the hash buckets with hash codes whose Hamming distance to the query are no more than 1 will be probed. It is difficult for Hamming distance-based methods to set L between 1 and $1 + K_1$ because the K_1 hash codes whose Hamming distance to the query is 1 cannot be further distinguished. Since the possible Hamming distance between hash codes of K_1 bits is limited to the set of $\{0, 1, \ldots, K_1\}$, there are only $K_1 + 1$ possible values for L.

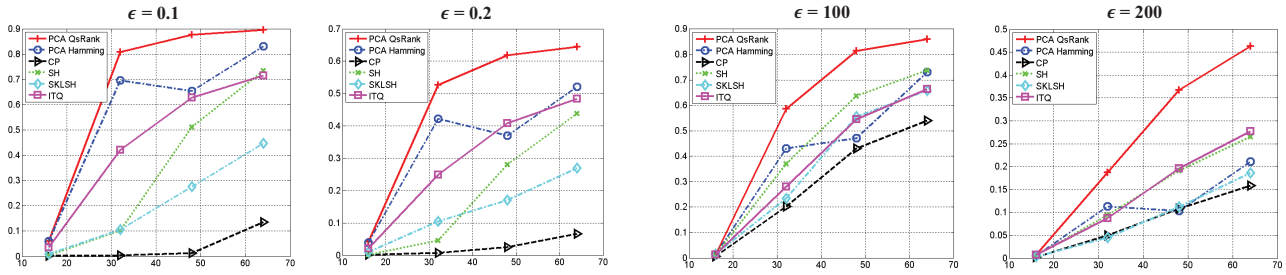
However, this is not a problem for QsRank-based system because QsRank computes a real number to rank a hash code. Therefore it is much easier to distinguish different hash codes thus provides much more flexibility to set L.

6. Evaluation

6.1. Experiment Setup

Datasets: the experiments were carried out on two datasets. The first dataset contains 10 million SIFT descriptors extracted from Oxford5K dataset [11]. Each descriptor is a vector of 128 dimensions, each of which is between 0 and 255. The second dataset comprises of $10,485,760$ images crawled from Internet. Block-based intensity and edge orientation information was used to describe each image. In total, each feature vector comprises of 116 dimensions. All dimensions were normalized to be between 0 and 1. For both datasets, 10K queries were randomly selected and their groundtruth neighbors in the ϵ neighborhood were generated by linear scan using L2 distance.

Baseline Methods: we compare the proposed indexing algorithm (**PCA-QsRank**) with six hamming distance-



(a) MAP on web images

(b) MAP on sift descriptors

Figure 3. Compare QsRank with Hamming distance based methods. The X axis corresponds to the hash code length.

based methods:

1. **PCA-Hamming** [16]: this is a PCA-based hash code indexing method which uses Hamming distance to rank hash code.

2. Iterative Quantization (**ITQ**) [4]: ITQ generates binary codes by finding a rotation of zero-centered data so as to minimize the quantization error of mapping the data to the vertices of binary hypercube.

3. Spectral Hashing (**SH**) [19]: SH is based on spectral graph partitioning and uses Laplace-Beltrami eigenfunctions for out-of-sample extension.

4. Locality Sensitive Hashing (**LSH**) [1]: LSH uses locality sensitive hashing functions to map close data points to the same bin and uses multiple hash table for indexing.

5. Compact Projection (**CP**) [8]: CP is a variant of LSH for generating binary codes. It projects a data point by a random matrix whose entries are sampled independently from a standard normal distribution.

6. **SKLSH** [12]: SKLSH is based on random features mapping for approximating shift-invariant kernels. In [12], this method is reported to outperform SH for code size larger than 64 bits.

For the implementation of the baseline methods, we used the matlab tools published on the authors' websites for ITQ, SH and SKLSH. For ITQ, the mean of the data was removed as suggested in [4]. For SKLSH, we followed the parameter setting in [4], which used a Gaussian kernel with bandwidth set to the average distance to the 50th nearest neighbor.

Evaluation Protocol: Retrieval effectiveness on both datasets was measured by mean average precision (MAP), computed as the area under the precision-recall curve and widely used for large-scale retrieval [11, 4, 20]. Besides, we used recall to evaluate the quality of the candidate set generated by PCA-QsRank with short binary hash codes for the proposed indexing system.

6.2. Experiment Results

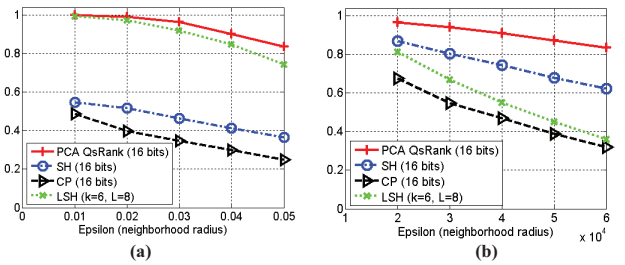
In this section, we present evaluation results to compare QsRank with Hamming distance and compare the proposed index systems with other state-of-the-art index structures. Throughout the evaluation, we set the $p_j(y_j)$ in Eq.9 to be a uniform distribution. We found this simple setting struck a good balance between retrieval accuracy and efficiency.

6.2.1 Linear Scan with QsRank and Hamming Distance

In this section, we compare the effectiveness of QsRank with Hamming distance based methods including PCA-Hamming[16], ITQ[4], CP[8], SH[19] and SKLSH[12]. The results are shown in Fig. 3. From the figure, one can see that QsRank consistently outperforms the baseline methods. In addition, the advantage of QsRank is more significant when ϵ is larger and the retrieval problem becomes more challenging.

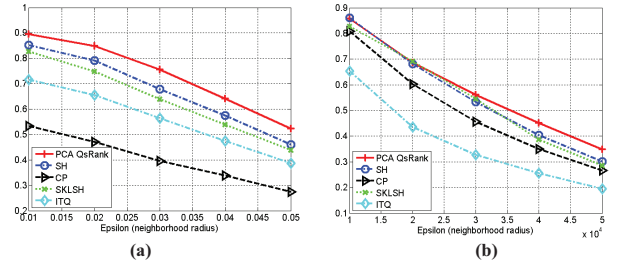
Another observation worth noting is that the performance of PCA-Hamming can actually *decrease* as the number of bits increases. In contrast, ITQ[4] also uses PCA as a preprocessing step for generating binary hash code, but its performance keeps improving when more bits are used. This demonstrates the effect of optimized rotation used in ITQ, and this result is consistent with the observation in [4]. It is interesting to see that the retrieval accuracy of QsRank also improves consistently when more bits are used, although no rotation is performed on top of the PCA generated projections. We think this is because QsRank assigns real numbers for single-bit ranking scores as defined in Eq.9 and thus it suffers less from the quantization error due to the binarization.

Note that one limitation of PCA-QsRank is that its code length cannot exceed the number of dimensions of the input data points. However, we demonstrate in the next section that even with relatively short binary code (64 bits), the retrieval accuracy of PCA-QsRank is higher than other baseline methods with binary codes of larger size (128 bits).



(a) (b)

Figure 4. Recall comparison for candidate set generation with different ϵ (neighborhood radius). Figure (a) and (b) shows results on 10 million web images and 10 million SIFT descriptors respectively.



(a) (b)

Figure 5. MAP comparison for the retrieval system with different ϵ (neighborhood radius). Figure (a) and (b) shows results on 10 million web images and 10 million SIFT descriptors respectively.

Methods	Candidate Set Size	
	Web Images	SIFT
PCA-QsRank	35669	49850
LSH[1]	35776	707823
CP[8]	50000	50000
SH[19]	50000	50000

Table 3. Candidate set size for recall comparison with other index structures.

Methods	Mem Cost (Bytes)	Retrieval Speed (Sec)	
		Web Images	SIFT
PCA-QsRank	10	0.026	0.029
ITQ[4]	10	0.027	0.031
CP[8]	16	0.219	0.352
SH[19]	16	0.238	0.281
SKLSH[12]	16	0.226	0.346

Table 4. Memory cost per image and retrieval speed comparison with state-of-the-art index methods on two datasets.

6.2.2 Index System Comparison

The proposed index system in Sec.5 have two components: a hash table to generate a candidate set of data points, and a rerank algorithm to provide the final rank list. In this section, we evaluate both of the components. Throughout the evaluation, we set K_1 to 16, K_2 to 48 and L to 50 for the proposed index system as discussed in Sec.5.

Recall comparison for candidate set generation: Here, we evaluate the percentage of true ϵ-neighbors in the candidate set generated by PCA-QsRank with hash code of 16 bits, and compare it to CP[8], SH[19] and LSH[1]. In the evaluation, 16 bits are used for CP and SH, the same number of bits as the PCA-QsRank. To generate a candidate set, linear scan is performed for CP and SH, based on which the top 50000 data points ranked by Hamming distance are used as candidate set. For LSH, we set K to 6 and L to 8. Note that this means LSH uses 8 hash tables for candidate set generation thus consumes 8 times more memory than other methods. If a data point is located in the same bucket as the query in at least one hash table, it is included in the candidate set of LSH. The average size of the candidate set for all the methods are summarized in Tab. 3.

The comparison results are shown in Fig. 4. From the results, one can see that PCA-QsRank achieves the best recall for different ϵ in both datasets. Besides, as shown in Tab. 3, the candidate set size for QsRank is relatively smaller than other methods, which further demonstrates the quality of the candidate set generated by PCA-QsRank. The excellent performance of PCA-QsRank with only 16 bits is because the PCA algorithm compresses most of the information in the top few bits. Therefore the PCA-QsRank algorithm is more suitable than SH and CP for the hash table based index structure.

Retrieval accuracy and cost comparison: Here, we evaluate the retrieval accuracy of the proposed index system measured by MAP. We compare with CP[8], SH[19] and SKLSH[12] with 128 bits binary code. We also compare with ITQ[4] combined with our proposed index in Section5, which uses 16 bits for candidate set generation and 64 bits for reranking. Since ITQ uses Hamming distance for ranking, we set maximum hamming distance to 2 when generating the candidate set. As discussed in the recall comparison experiments, CP and SH is not suitable for hash table-based index system. Also, SKLSH is reported to exhibit more advantage for larger code sizes. Therefore we perform linear scan for CP, SH and SKLSH to generate the retrieval results. The results are summarized in Fig. 5. In the figure, it is observed that PCA-QsRank with 64 bit binary code ($K_1 + K_2 = 64$) achieves superior results on both datasets compared with ITQ[4] and CP[8]. Besides, PCA-QsRank outperforms SH[19] and SKLSH[12] on the web image dataset. On the SIFT descriptor dataset, PCA-QsRank is comparable with SH and SKLSH when ϵ is small and superior to them as ϵ increases.

In terms of the retrieval cost, PCA-QsRank consumes 10 bytes per image, out of which 4 bytes is used for an image ID stored in the hash table-based index and 6 bytes for the 48 bit binary code for reranking. This is the same as ITQ[4], since both methods are combined with the same index structure proposed in Section5. The memory cost is larger for

CP, SH and SKLSH, which consumes 16 bytes memory per image for storing the 128 bit binary code. Furthermore, the retrieval speed of CP, SH and SKLSH is much slower than PCA-QsRank and ITQ, as shown in Tab. 4. This is because they need to scan all the data points in the database.

7. Conclusion

In this paper, we have presented a query sensitive hash code ranking algorithm (QsRank) for large-scale ϵ-neighbor search. The QsRank algorithm considers ϵ and the query raw feature when computing the ranking score and models the statistical properties of the target ϵ-neighbors in the binary hash code space. Unlike Hamming distance, the QsRank algorithm *does not convert queries to hash codes* when computing the ranking score. Therefore the resulted ranking score is more discriminative. We evaluate the QsRank algorithm on two datasets of 10 million web images and 10 million SIFT descriptors. There are several observations. First, combined with short PCA-based hash code, the QsRank can propose a small candidate set with high recall. Therefore it is suitable for a hash table-based index structure. Second, with code length of 64 bits, QsRank can achieve superior retrieval accuracy than state-of-the-art methods with much longer hash code (i.e. 128 bits). Furthermore, combined with the proposed hash table-based index with reranking, QsRank actually consumes less memory and achieves faster retrieval speed. Finally, QsRank provides more flexibility to balance accuracy and efficiency, as illustrated in Sec.5.

One limitation of the proposed QsRank algorithm is that it is based on PCA-based hash code which assigns one bit to each PCA projection. Therefore QsRank cannot use hash code with more bits than the number of dimensions of input features. We plan to study better quantization scheme for PCA-based hash codes and improve QsRank in the future work.

Acknowledgements

We would like to thank Yunchao Gong, Svetlana Lazebnik, Yair Weiss, Antonio Torralba and Rob Fergus for making their codes publicly available.

References

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[2] A. Auclair, L. Cohen, and N. Vincent. How to use sift vectors to analyze an image with database templates. *Adaptive Multimedial Retrieval: Retrieval, User, and Semantics*, pages 224–236, 2008.

[3] J. L. Bentley. K-d trees for semidynamic point sets. In *Symposium on Computational Geometry*, pages 187–197, 1990.

[4] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, pages 817–824, 2011.

[5] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. In *CVPR*, 2008.

[6] Y.-G. Jiang, J. Wang, and S.-F. Chang. Lost in binarization: query-adaptive ranking for similar image search with compact codes. In *ICMR*, page 16, 2011.

[7] R. Lin, D. Ross, and J. Yagnik. Spec hashing: Similarity preserving algorithm for entropy-based coding. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 848–854. IEEE, 2010.

[8] K. Min, L. Yang, J. Wright, L. Wu, X.-S. Hua, and Y. Ma. Compact projection: Simple and efficient near neighbor search with practical memory requirements. In *CVPR*, pages 3477–3484, 2010.

[9] S. Nayar, S. Nene, and H. Murase. Real-time 100 object recognition system. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 3, pages 2321–2325. IEEE, 1996.

[10] S. Nene and S. Nayar. Closest point search in high dimensions. In *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on*, pages 859–865. IEEE, 1996.

[11] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, pages 1–8. Ieee, 2007.

[12] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. *NIPS*, 22, 2009.

[13] R. Salakhutdinov and G. E. Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, 2009.

[14] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(11):1958–1970, 2008.

[15] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *CVPR*, 2008.

[16] B. Wang, Z. Li, M. Li, and W.-Y. Ma. Large-scale duplicate detection for web image search. In *ICME*, pages 353–356, 2006.

[17] J. Wang, O. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, pages 3424–3431, 2010.

[18] X.-J. Wang, L. Zhang, M. Liu, Y. Li, and W.-Y. Ma. Arista - image search to annotation on billions of web photos. In *CVPR*, pages 2987–2994, 2010.

[19] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.

[20] X. Zhang, Z. Li, L. Zhang, W. Ma, and H. Shum. Efficient indexing for large scale visual search. In *CVPR*, pages 1103–1110. IEEE, 2009.

[21] X. Zhang, L. Liang, and H.-Y. Shum. Spectral error correcting output codes for efficient multiclass recognition. In *ICCV*, pages 1111–1118, 2009.