Sensors **2010**, *10*, 11259-11273; doi:10.3390/s101211259

OPEN ACCESS

sensors

ISSN 1424-8220

www.mdpi.com/journal/sensors

Article

Approximate Nearest Neighbor Search by Residual Vector Quantization

Yongjian Chen [1], Tao Guan [2,*] and Cheng Wang [1]

[1] Digital Engineering & Simulation Research Center, Huazhong University of Science and Technology, Wuhan 430074, China; E-Mails: wangch@hhu.edu.cn (Y.C.); chyojn@gmail.com (C.W.)

[2] School of Computer Science & Technology, Huazhong University of Science and Technology, No.1037 Luoyu Road, Wuhan 430074, China

* Author to whom correspondence should be addressed; E-Mail: qd_gt@126.com.

Received: 9 October 2010; in revised form: 20 November 2010 / Accepted: 7 December 2010 / Published: 8 December 2010

Abstract: A recently proposed product quantization method is efficient for large scale approximate nearest neighbor search, however, its performance on unstructured vectors is limited. This paper introduces residual vector quantization based approaches that are appropriate for unstructured vectors. Database vectors are quantized by residual vector quantizer. The reproductions are represented by short codes composed of their quantization indices. Euclidean distance between query vector and database vector is approximated by asymmetric distance, *i.e.*, the distance between the query vector and the reproduction of the database vector. An efficient exhaustive search approach is proposed by fast computing the asymmetric distance. A straight forward non-exhaustive search approach is proposed for large scale search. Our approaches are compared to two state-of-the-art methods, spectral hashing and product quantization, on both structured and unstructured datasets. Results show that our approaches obtain the best results in terms of the trade-off between search quality and memory usage.

Keywords: approximate nearest neighbor search; high-dimensional indexing; residual vector quantization

1. Introduction

Approximate nearest neighbor search (ANN) is proposed to tackle the curse of the dimensionality problem [1,2] in exact nearest neighbor (NN) searching. The key idea is to find the nearest neighbor with high probability. ANN is a fundamental primitive in computer vision applications such as keypoint matching, object retrieval, image classification and scene recognition [3]. In many computer vision applications, the data-points are high-dimensional vectors that are embedded in Euclidean space, and the memory usage for storing and searching high-dimensional vectors is a key criterion for problems involving large amount of data.

The state-of-the-art approaches such as tree-based methods (e.g., KD-tree [4], hierarchical k-means (HKM) [5], FLANN [6]) and hash-based methods (e.g., Exact Euclidean Locality-Sensitive Hashing (E2LSH) [7,8]) involve indexing structures to improve the performance. The memory usage of indexing structure may even be higher than the original data when processing large scale data. Moreover, FLANN and E2LSH need a final re-ranking based on exact Euclidean distance, which means the original vector should be stored in main memory, this requirement seriously limits the databases' scale. Binary index methods such as [9-11] simplify the indexing structure by using binary code to index the space partitions. However, these methods also need the original vector for final re-ranking.

Recently proposed hamming embedding methods compress the vectors into short codes and approximate the Euclidiean distance between two vectors by the hamming distance between their codes. These methods include hamming embedding [12], miniBOF [13], small hashing code [14], small binary code [15] and spectral hashing [16]. These methods make it possible to store large scale data in main memory. One weakness of these methods is the discrimination limitation of hamming distance as the total number of possible hamming distance is limited by code length. [17] introduced product quantization to compress the vector into several bytes and proposed a more accurate distance approximation. However, its search quality is limited on unstructured vector data.

Objectives of the paper are comparable to those of [16,17]: (1) storing millions of high-dimensional vectors in memory and (2) quickly finding similar vectors to a target vector. In contrast with product quantization, we focus on the performance for unstructured vector data. We introduce residual vector quantization, which is appropriate for unstructured data, for the vector encoding. An efficient exhaustive search method is proposed based on fast distance computing. A non-exhaustive search method is proposed to improve the efficiency for large scale search. Our approaches are compared to two state-of-the-art methods, spectral hashing and product quantization, on both structured and unstructured datasets. Results show that our approaches obtain the best results in terms of accuracy and speed.

Our paper is organized as follows: Section 2 presents the residual vector quantization and Section 3 introduces our exhaustive and non-exhaustive search methods that are based on the residual vector quantization. Section 4 evaluates the search performance and compares our approaches with two state-of-the-art methods. Section 5 discusses the results and Section 6 is the conclusion.

2. Residual Vector Quantization

A K-point vector quantizer$Q$ maps a vector $x \in R^D$ into its nearest centroidin codebook $C = \{c_i, i = 1..K\} \subset R^D$:

$$\tilde{x} = Q(x) = \underset{c_i \in C}{\arg\min}\, d(x, c_i) \tag{1}$$

where $d(x, c_i)$ is the exact Euclidean distance between x and c_i. This destructive process can be interpreted as approximatingthe$x$by one of centroids in R^D space [18], and the residual vector is:

$$\varepsilon = x - \tilde{x} = x - Q(x) \tag{2}$$

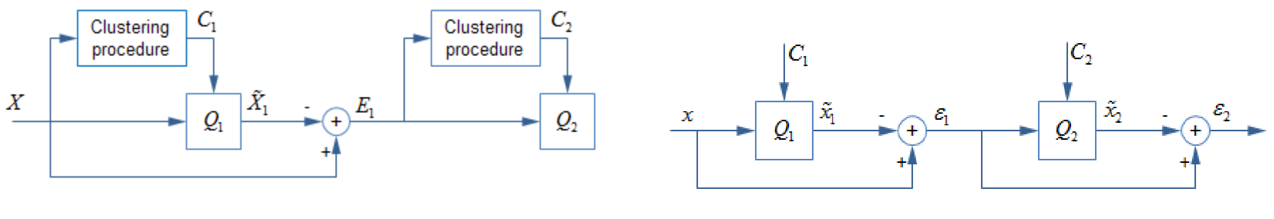
The performance of quantizer Q is measured by mean squared error (MSE):

$$MSE(Q) = EX\left[d(x, Q(x))^2 \right] \tag{3}$$

Residual vector quantization [19,20] is a common technique to reduce the quantization error with several low complexity quantizers. Residual vector quantization approximate the quantization error by another quantizer instead of discard it. Several stage-quantizers, each has its corresponding stage-codebook, are connected sequentially. Each stage-quantizer approximates preceding stage's residual vector by one of centroids in the stage-codebook and generates a new residual vector for succeeding quantization stage. Block diagrams of a two stages residual vector quantization are shown in Figure 1. In the learning phase (Figure 1(a)), a training vector set X is provided and the first stage-codebook C_1 is generated by k-means clustering method. The entire training set is then quantized by the first stage-quantizer Q_1 which is defined by C_1. The difference between X and its first stage quantization outputs , which is the first residual vector set E_1, is used for learning the second stage-codebook C_2. In quantizing phase (Figure 1(b)), the input vector x is quantized by first stage-quantizer Q_1, which is defined by first stage-codebook C_1. The difference between x and its first stage quantization output , which is the first residual vector ε_1, is quantized by second stage-quantizer Q_2. The second residual vector ε_2 is discarded. The first two quantization outputs are used to approximate the input vector:

$$x = \tilde{x}_1 + \varepsilon_1 = \tilde{x}_1 + \tilde{x}_2 + \varepsilon_2 \approx \tilde{x}_1 + \tilde{x}_2 = \tilde{x} \tag{4}$$

Figure 1. Block diagrams of two-stages residual vector quantization. **(a)** Learning codebooks; **(b)** Quantizing a vector.



(a) (b)

For L stages residual vector quantization, a vector x is approximated by the sum of its L stages' quantization outputs while the last stage's quantization error is discarded:

$$x = \sum_{i=1}^{L} \tilde{x}_i + \varepsilon_L \approx \sum_{i=1}^{L} \tilde{x}_i = \tilde{x} \tag{5}$$

For transformation or storage, indices of quantization outputs are used. For L stage residual vector quantization, which is constructed by K-point vector quantizers, the bit rate is $L \log_2 K$ per vector.

The quantization performance of ith stage-quantizer is:

$$MSE(Q_i) = \frac{1}{N} \sum_{\varepsilon \in \mathrm{E}_i} \varepsilon^T \varepsilon = \frac{1}{N} \sum_{j=1}^{K} \sum_{x \in V_j} \left\| x - c_{i,j} \right\|^2 \tag{6}$$

where E_i is the new residual vector set generated by Q_i, V_j is the jth cluster and $c_{i,j}$ is V_j's centroid. Considering the optimization problem of finding a vector y to minimize the objection function:

$$J = \sum_{x \in V_j} \left\| x - y \right\|^2 \tag{7}$$

By differentiating the objection function J with respect to y and setting derivative equal to zero, it is easy to obtain the minimizing$y$:

$$y = \frac{1}{N_j} \sum_{x \in V_j} x \tag{8}$$

where N_j is the number of vectors in jth cluster. This means the centroid of cluster minimizes the objection function:

$$\sum_{x \in V_j} \left\| x - c_{i,j} \right\|^2 = \min_y \sum_{x \in V_j} \left\| x - y \right\|^2 \le \sum_{x \in V_j} \left\| x - y \right\|^2 \Bigg|_{y=0} = \sum_{x \in V_j} \left\| x \right\|^2 \tag{9}$$

With the observations that $\sum_{\varepsilon \in \mathrm{E}_i} \varepsilon^T \varepsilon = \sum_{j=1}^{K} \sum_{x \in V_j} \left\| x - c_{i,j} \right\|^2$ and $\sum_{x \in \mathrm{E}_{i-1}} x^T x = \sum_{j=1}^{K} \sum_{x \in V_j} \left\| x \right\|^2$, we obtain the inequality:

$$MSE(Q_i) \le MSE(Q_{i-1}) \tag{10}$$

which means the k-means clustering method guarantee the MSE of stage-quantizers are decreasing monotonically.

3. Using Residual Vector Quantization for ANN

3.1. Exhaustive Search by Fast Distance Computation

In [17] the exact Euclidean distance between two vectors is approximated by asymmetric distance, *i.e.*, the distance between a vector and a reproduction of another vector:

$$d(x, y) \approx \tilde{d}(x, y) = d(x, Q(y)) \tag{11}$$

Asymmetric distance reduces the quantization noise and improves the search quality [17]. We have proposed fast asymmetric distance computation based on residual vector quantization. Suppose a database vector y is quantized by $L \times K$ residual vector quantizer, its indices of quantization output are $\{u_j, 1 \le u_j \le K, j = 1..L\}$, and the reproduction of y is constructed by the sum of corresponding centroids:

$$\tilde{y} = \sum_{i=1}^{L} \tilde{y}_i = \sum_{i=1}^{L} c_{i,u_i}, c_{i,u_i} \in C_i, 1 \le u_i \le K \tag{12}$$

where c_{i,u_i} is the u_ith centroid of codebook C_i. The squared asymmetric distance between y and the target vector x is the exact squared distance between x and $\tilde{y}$:

$$\begin{aligned} \tilde{d}(x,y)^2 &= d(x,\tilde{y})^2 = \|x-\tilde{y}\|^2 = \|x\|^2 + \|\tilde{y}\|^2 - 2\langle x,\tilde{y}\rangle \\ &= \|x\|^2 + \|\tilde{y}\|^2 - 2\left\langle x, \sum_{i=1}^{L} c_{i,u_i} \right\rangle = \|x\|^2 + \|\tilde{y}\|^2 - 2\sum_{i=1}^{L} \langle x, c_{i,u_i} \rangle \end{aligned} \tag{13}$$

where $\langle x,y\rangle$ is dot product. $\|\tilde{y}\|$ is pre-computed off-line when the database vector is quantized. The dot products of codebooks' centroids and target vector $x$are computed and stored in a look-up tablewhen x is submitted:

$$T = \{t_{i,j}\}, t_{i,j} = \langle x, c_{i,j} \rangle, c_{i,j} \in C_i, 1 \le i \le L, 1 \le j \le K \tag{14}$$

The squared asymmetric distance can then be efficiently estimated by several table lookups:

$$\tilde{d}(x,y)^2 = \|x\|^2 + \|\tilde{y}\|^2 - 2\sum_{i=1}^{L} t_{i,u_i} \tag{15}$$

If we only consider the order of distance, term $\|x\|$ is a constant for all database vector and can be ignored in asymmetric distance computation. R nearest neighbors are selected based on the estimatedsquared asymmetric distances.

3.2. Non-Exhaustive Search by Rough Approximation

Exhaustive search has to scan quantization codes of all database vectors. In problems such as bag-of-features-based large scale image retrieval, billions of images are represented by hundreds of local feature vectors per image, and it is prohibitive to scan the feature vector database, even with fast asymmetric distance computation.

In [17] the authors proposed a non-exhaustive search method for large scale datasets. A coarse quantizer is involved to filter out farther database vectors, and then a product quantizer is used for fine search. In contrast with using an external coarse quantizer, we propose a straight forward non-exhaustive search approach based on the approximating sequence of database vector y that is generated by residual vector quantization:

$$\{\tilde{y}^{(l)}\}, \tilde{y}^{(l)} = \sum_{i=1}^{l} \tilde{y}_i, 1 \le l \le L \tag{16}$$

Our exhaustive search approach uses only the most accurate item $\tilde{y}^{(L)}$ to approximate the y. In non-exhaustive search, the first L_1 quantization outputs generate a rough approximation:

$$\tilde{y}^{(L_1)} = \sum_{i=1}^{L_1} \tilde{y}_i, L_1 < L \tag{17}$$

The rough asymmetric distances between database vectors and the target vector are then evaluated by table lookups for coarse search:

$$d\left(x, \tilde{y}^{(L_1)}\right)^2 = \|x\|^2 + \left\|\tilde{y}^{(L_1)}\right\|^2 - 2\sum_{i=1}^{L_1} t_{i,u_i} \tag{18}$$

The database vectors which have large rough distances are pruned and the remaining database vectors are used to evaluate more accurate distances to the target vector by their most accurate approximations as in Equation (13).

The total number of possible rough approximations is K^{L_1}, thus an inverted file system is used to improve the search performance. Each inverted list corresponds to a possible rough approximation. When encoding database vectors by $L \times K$ residual vector quantization, each vector's first L_1 indices are used to determine which inverted list it should be inserted in, then the L_1 indices are discarded and only the last $L_2 = L - L_1$ indices and its vector id are stored in the inverted list. A query vector first evaluated its distances to the K^{L_1} possible rough approximations by Equation (18). The W nearest rough approximations are selected and corresponding W inverted lists are scanned to evaluate more accurate distance to query vector:

$$\begin{aligned} d\left(x, \tilde{y}^{(L)}\right)^2 &= \|x\|^2 + \left\|\tilde{y}^{(L)}\right\|^2 - 2\left\langle x, \tilde{y}^{(L)}\right\rangle = \|x\|^2 + \left\|\tilde{y}^{(L)}\right\|^2 - 2\sum_{i=1}^{L}\left\langle x, c_i^{(u_i)}\right\rangle \\ &= d\left(x, \tilde{y}^{(L_1)}\right)^2 + \left\|\tilde{y}^{(L)}\right\|^2 - \left\|\tilde{y}^{(L_1)}\right\|^2 - 2\sum_{i=L_1-1}^{L}\left\langle x, c_i^{(u_i)}\right\rangle \end{aligned} \tag{19}$$

Equation (19) shows the squared asymmetric distances which are computed in fine search can be updated by squared rough distance in the coarse search and only L_2 table lookups per vector are involved. The term $\left\|\tilde{\boldsymbol{y}}^{(\boldsymbol{L})}\right\|^2 - \left\|\tilde{\boldsymbol{y}}^{(\boldsymbol{L_1})}\right\|^2$ is pre-calculated and stored in offline quantization stage. By fast table lookups and distance update scheme, both coarse and fine search are efficient. R nearest neighbors are selected based on the squared asymmetric distances that are estimated in fine search.

4. Experiments and Results

4.1. Dataset

Three public available datasets were used to evaluate the performances of ANN methods: the structured SIFT descriptor dataset [21], semi-structured GIST descriptor dataset [21] and unstructured VLAD descriptor dataset [22]. SIFT descriptor codes small image patch while GIST descriptor and VLAD descriptor code entire image. SIFT descriptor is a histogram of oriented gradients that extracted from gray image patch. GIST descriptor is similar to SIFT applied to the entire image. It applies an oriented Gabor filter over different scales and averages the filter energy in each bin. The VLAD descriptor is constructed by first aggregating images' SIFT descriptors' quantization residual vectors locally and then reducing their dimensions by PCA.

The SIFT dataset and GIST dataset have three subsets: learning set, database set, and query set. The learning set is used for learning the model and evaluating quantization performance, the database and query sets are used for evaluating ANN search performance. For the SIFT dataset, the learning set is extracted from Flicker images [12] and the database and query descriptors are from INRIA Holidays images [23]. For GIST, the learning set consists of a subset of the tiny image set of [24]. The database set is the Holidays image set combined with Flicker1M used in [12]. The query vectors are from the Holidays image queries [23]. VLAD dataset is generated by public package and public local image descriptors [22] which are extracted from Holiday image dataset [23]. The dataset has 1,491 128-dimensional vectors and was divided into 500 groups. The first descriptor of each group is the query image and the correct retrieval results are the other images of the group. Total vectors in dataset are used as training set and database set. All these descriptors are high-dimensional float vectors. Scales of these datasets are summarized in Table 1.

Table 1. Dataset information.

Dataset	**SIFT**	**GIST**	**VLAD**
Dimension of descriptor	128	960	128
Size of learning set	100,000	500,000	1,491
Size of database set	1,000,000	1,000,000	1,491
Size of query set	10,000	1,000	500

4.2. Quantization Performance

This section investigates the quantization performance of our approach by evaluating the influence of parameters over quantization error. K is the number of centroids of stage-quantizer, L is the total number of stage-quantizers. The code length, *i.e.*, $L\log_2 K$, is regarded as a metric of storage.

Figure 2. Quantization error associated with K and L. (**left**) SIFT dataset; (**right**) GIST dataset.

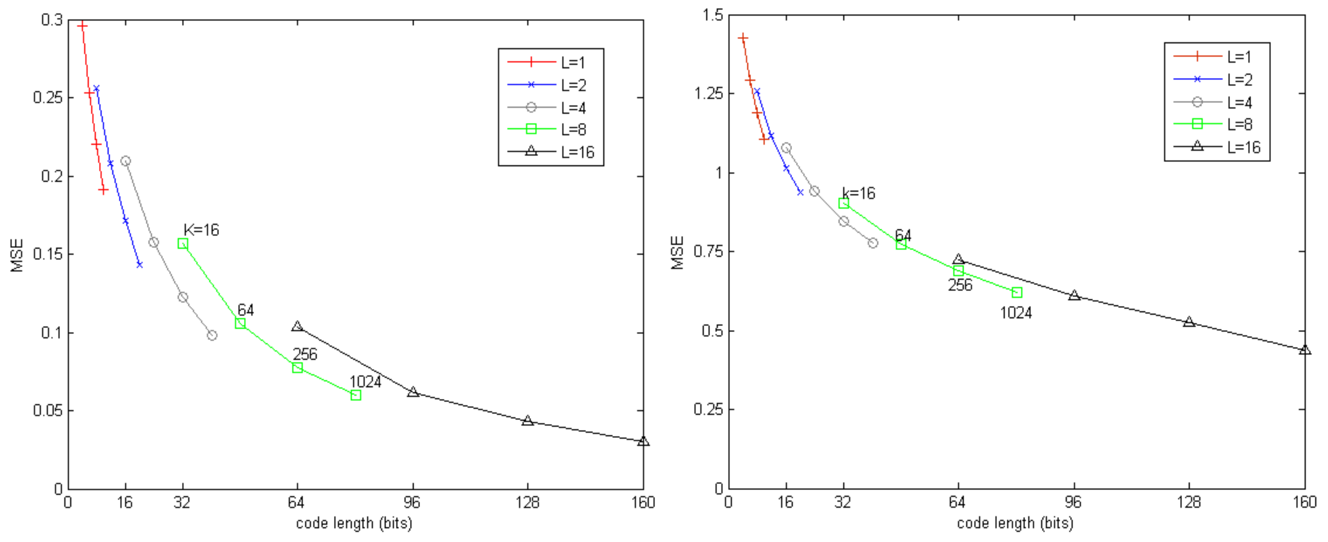


Figure 2 shows the trade-offs between quantization accuracy and memory. It is clear that the quantization error is reduced by increase either *K* or *L*. For a fixed number of bits, the residual vector quantizer which has fewer stage-codebooks and more centroids in each stage-codebook is more accurate than the residual vector quantizer which has more stage-codebooks and fewer centroids in each stage-codebook.

4.3. Parameters' Influences on Search Accuracy

The performances of our approaches are measured by two metrics: recall@R and ratio of distance errors (RDE). Recall@R is defined in [17] as the proportion of query vectors for which the nearest neighbor is randked in the first R positions. Values of recall@R close to 1 indicate high quality of search results. RDE [11] is defined as:

$$RDE = 1 - \frac{\sum_{i=1}^{k} d(NN_i, x)}{\sum_{i=1}^{k} d(ANN_i, x)} \tag{20}$$

where NN_i is the *i*th exact nearest neighbor of query x and ANN_i is x's *i*th approximate nearest neighbor. Values of RDE close to 0 indicate high quality of results. Mean and standard variance of RDE is used to measure the average search quality.

Figure 3 and 4 show the performance of our exhaustive search method. Figure 3 shows the trade-off between recall@R and code length for SIFT and GIST datasets. When the code length is fixed, the residual vector quantizer which has fewer codebooks and more centroids in each codebook is more accurate than the residual vector quantizer which has more codebooks and fewer centroids in each codebook. It seems a good choice to use 8 × 256 residual vector quantization for SIFT descriptor and 16 × 256 residual vector quantization for GIST descriptor.

Figure 3. Exhaustive search accuracy. (**left**) SIFT dataset. (**right**) GIST dataset.

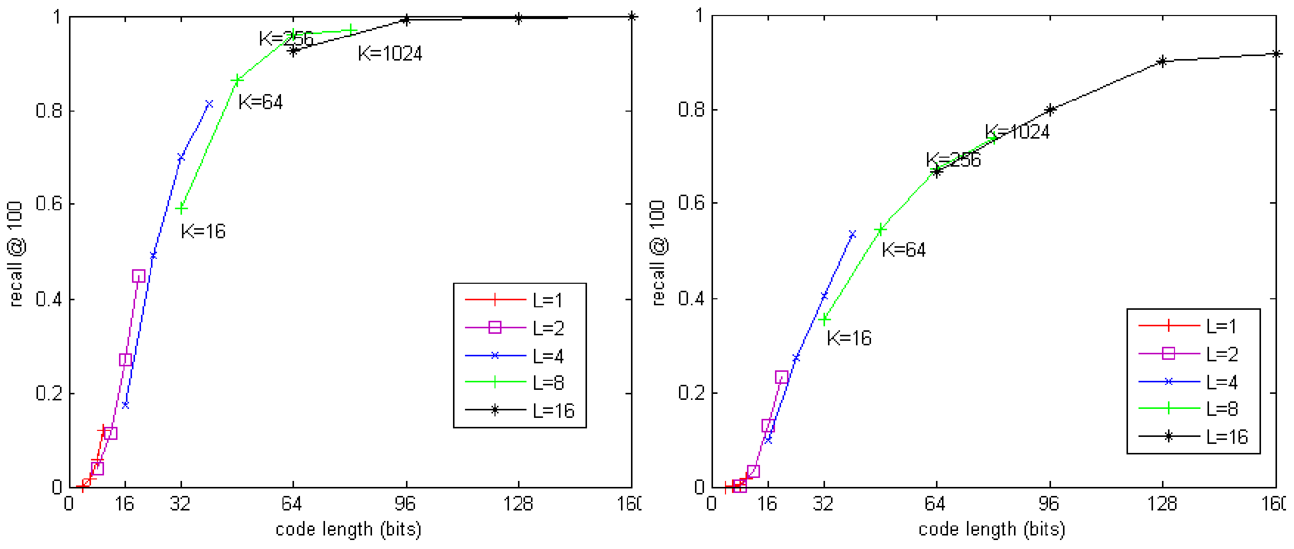


Figure 4 shows the RDE for SIFT dataset. The mean of RDE is tending to 0 when increasing code length. The standard variance of RDE is also significant reduced when increasing code length, which means the query results are more stable when more bits are used to encode the vectors.

Figure 4. RDE for SIFT dataset, exhaustive search method. (**left**) mean of RDE. (**right**) standard variance of RDE.

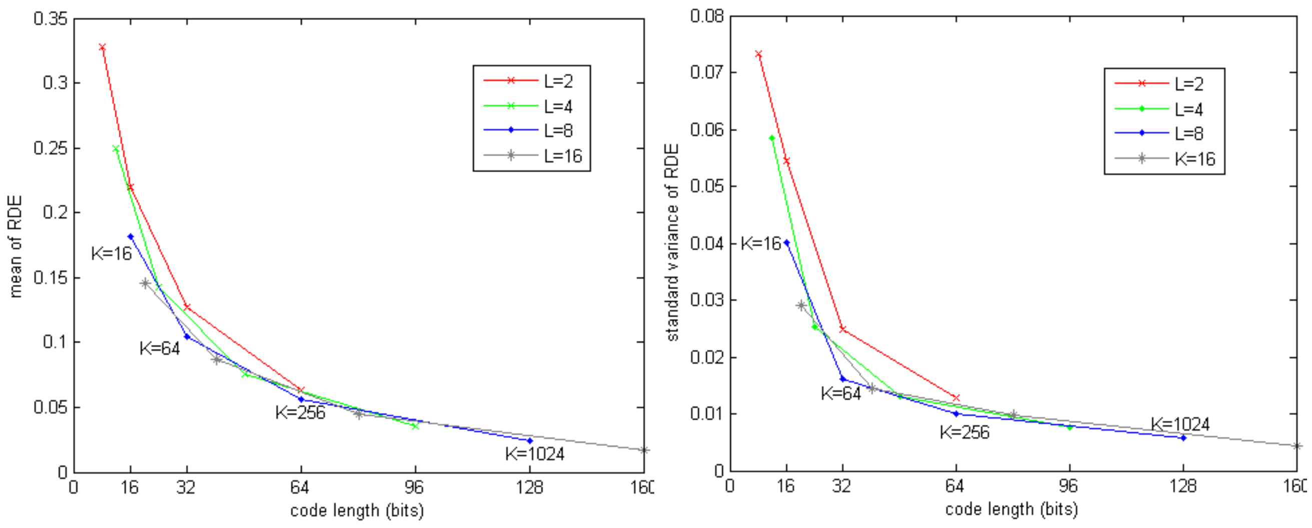
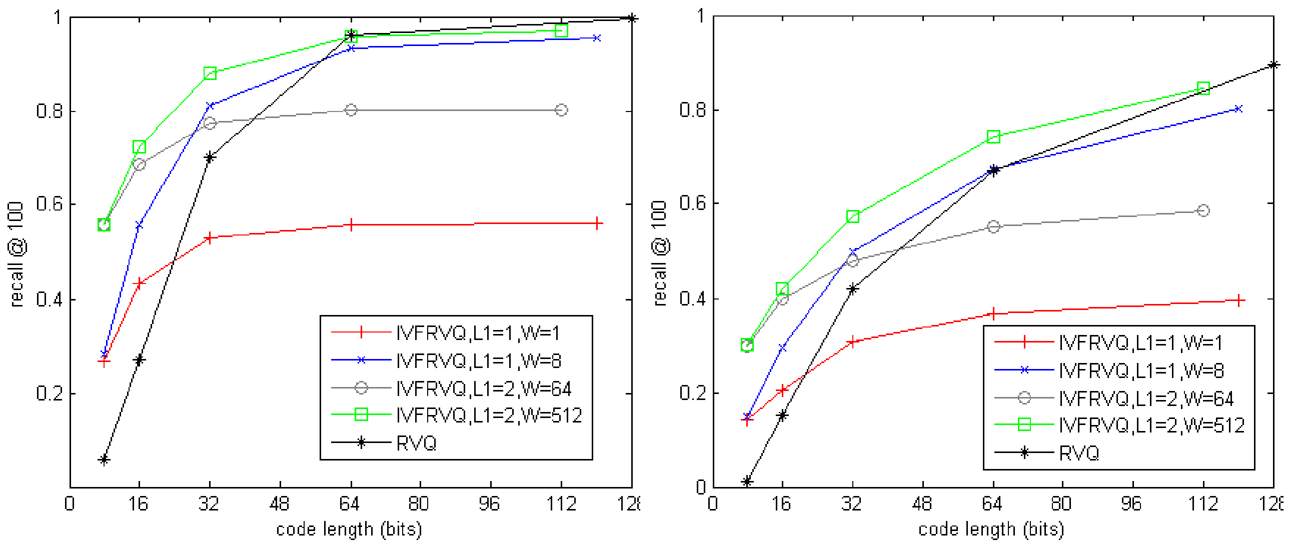


Figure 5 shows impact of the parameters for our non-exhaustive search method. $K = 256$, $L_1 \in \{1,2\}$ and $L_2 \in \{1,2,4,8,16\}$ are the numbers of stage-quantizers used for coarse search and fine search, W is the number of candidate inverted lists for fine search.The total number of inverted lists is K^{L_1}. The code length $L_2 \log_2 K$ is regarded as a metric of storage. Results of our exhaustive search method are also plotted in dash line for comparison. For simplicity, our exhaustive search and non-exhaustive search methods are respectively denoted as RVQ and IVFRVQ. We observed that the performance of IVFRVQ strongly depends on W which determines the fraction of inverted lists that are scanned. When a small fraction of inverted lists are scanned, increasing the code length is useless for improving the performance. When sufficient inverted lists are scanned, performance of IVFRVQ is comparable to even better than RVQ.

Figure 5. Search accuracy of non-exhaustive search. (**left**) SIFT dataset. (**right**) GIST dataset.



Tables 2 and 3 show comparisons of search efficiency. Both RVQ and IVFRVQ encode the vector into 64-bit code. It is clear that the pruning strategy significantly reduces the search time. It is noticed that it has to increase the *W* for search accuracy when $L_1 = 2$, but the frequent inverted lists access reduces the search performance.

Table 2. Comparison of RVQ and IVFRVQ on SIFT dataset.

Method	**Parameters**	**Search time(msec)**	**Average number of scanned codes**	**Recall @100**
RVQ	$L = 8, K = 256$	34	1,000,000	0.96
IVFRVQ	$L_1 = 1, L_2 = 8, K = 256, W = 1$	0.65	4,261	0.56
	$L_1 = 1, L_2 = 8, K = 256, W = 8$	**2.6**	**33,602**	**0.93**
	$L_1 = 2, L_2 = 8, K = 256, W = 64$	3.2	1,682	0.80
	$L_1 = 2, L_2 = 8, K = 256, W = 512$	15.1	9,692	0.96

Table 3. Comparison of RVQ and IVFRVQ on GIST dataset.

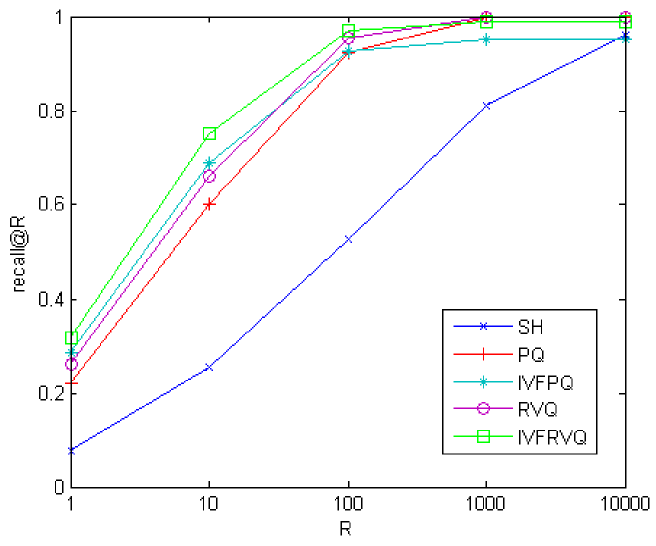
Method	**Parameters**	**Search time(msec)**	**Average number of scanned codes**	**Recall @100**
RVQ	$L = 8, K = 256$	36.1	1,000,000	0.67
IVFRVQ	$L_1 = 1, L_2 = 8, K = 256, W = 1$	2.9	5,205	0.36
	$L_1 = 1, L_2 = 8, K = 256, W = 8$	**4.6**	**42,699**	**0.67**
	$L_1 = 2, L_2 = 8, K = 256, W = 64$	5.7	2,423	0.55
	$L_1 = 2, L_2 = 8, K = 256, W = 512$	20.5	16,512	0.74

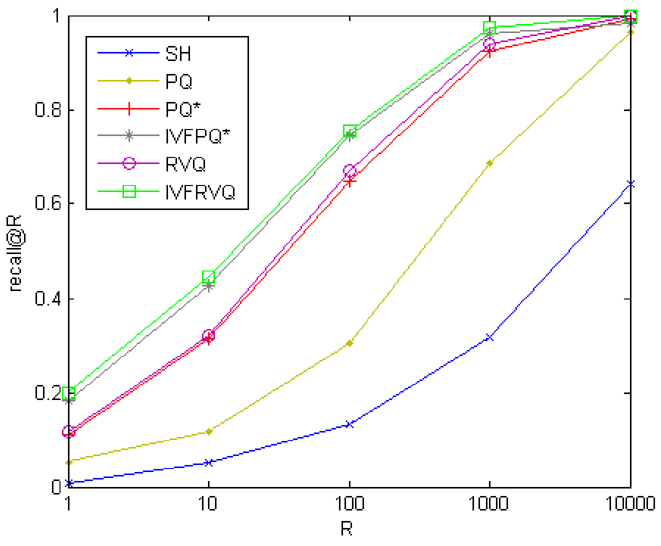
4.4. Compared with the State of the Art

In this section we compare our approach with two state-of-the-art methods: spectral hashing (SH) and product quantization. The performance of product quantization is sensitive to the grouping order of vector components. The natural product quantization groups the consecutive components while the structured product quantization groups related components together based on the prior knowledge of vector's structure. Experimental results in [17] show that the natural product quantization is appropriate for SIFT descriptor while the structured product quantization is appropriate for GIST descriptor. For simplicity, the natural product quantization method is denoted as PQ while the structured product quantization method is denoted as PQ*, their non-exhaustive version are denoted as IVFPQ and IVFPQ* respectively. Vectors are compressed into 64-bit binary codes. Eight 256-point quantizers are used for PQ and a 1024-point quantizer is used as the coarse quantizer for IVFPQ. We use $L = 8$, $K = 256$ for RVQ and $L_1 = 1$, $L_2 = 8$, $K = 256$ for IVFRVQ.

Figure 6 compares the search qualities on SIFT and GIST datasets. On the benchmark SIFT, our approaches significantly outperform spectral hashing and are slightly better than product quantization methods. On the benchmark GIST, our approaches significantly outperform spectral hashing and natural product quantization methods and are comparable to structured product quantization methods.

Figure 6. Comparison of search accuracies obtained by spectral hashing, product quantization methods and our approaches. (**left**) SIFT dataset, 64-bit codes. (**right**) GIST dataset, 64-bit codes.





The VLAD dataset is used for evaluating the accuracy of ANN methods on unstructured vectors. The performance is measured by mean average precision (mAP) [22] which is defined as the area of recall-precision curve, a larger value of mAP indicate a better retrieval performance. Table 4 shows the accuracies obtained by different methods (spectral hashing, product quantization and our approach) and different code length configurations (32 bits, 64 bits, 128 bits). Both product quantizer and our residual vector quantizer are constructed by 256-point vector quantizer. The code length of spectral hashing is directly assigned while those of product quantization and our approach are controlled by the number of quantizers. We use a 1024-point quantizer as the coarse quantizer for IVFPQ. We only test the 32-bit and 64-bit configurations for our approaches because the stage-quantization errors are too small to be handled by our single precision implementation when 16 stage-quantizers are used. It is clear that our approach is significant outperform spectral hashing and product quantization. Equivalently, our method obtains a comparable search quality with only half the code length of product quantization.

Table 4. Comparison with state of the art on VLAD dataset.

	32 bits	**64 bits**	**128 bits**
SH	0.255	0.349	0.397
PQ	0.337	0.409	0.457
RVQ	**0.407**	**0.510**	

4.5. Speed Comparison

Table 5 compares the search time of different methods on the SIFT dataset. Spectral hashing and product quantization use the public available Matlab packages. Our approaches are implemented in Matlab. Both the hamming distance computation for spectral hashing and the asymmetric distance computation for product quantization and our approaches are optimized by C. All methods compress

SIFT descriptors 64-bit binary code. The time is measured on a 2.2 GHz CPU laptop with 3 GB of RAM. The approaches RVQ, PQ and SH have similar rum times because they all scan the whole database and compute the distances by table lookups. Non-exhaustive search methods significant improve the performance. IVFRVQ is more efficient than IVFPQ for equal search accuracy because IVFPQ calculates W look-up tables for individual candidate inverted list while IVFRVQ only calculates one look-up table.

Table 5. Search speed for 64-bit code and different methods (SIFT dataset).

Method	Parameters	Search time(msec)	Average number of scanned codes	Recall @100
RVQ	$L=8, K=256$	34	1,000,000	0.96
IVFRVQ	$L_1=1, L_2=8, K=256, W=8$	***2.6***	33,602	***0.93***
PQ	$L=8, K=256$	33.7	1,000,000	0.93
IVFPQ	$K'=1024, L=8, K=256, W=8$	3	9,102	0.87
IVFPQ	$K'=1024, L=8, K=256, W=16$	7.3	17,621	0.93
SH	$nbit=64$	35.3	1,000,000	0.53

5. Discussion

5.1. Advantages of Residual Vector Quantization

The advantage of residual vector quantization is quantizing the whole vector in original space. Product quantization is based on the assumption that the subspaces are statistically mutual independent such that the original space can be represented by the production of these subspaces. But vectors in real data do not all meet that assumption. Moreover, the vector's structure determines the quantization parameters and makes product quantization inflexible. In contrast, residual vector quantization processes the whole vector in original space, and the parametersare not limited by the structure of vector.

5.2. Link between Residual Vector Quantization and Hierarchical k-means

Residual vector quantization can be regarded as a simplified hierarchical k-means (HKM). When generating a new quantization level, HKM performs k-means clustering in each previous level's cluster and generate a new partition for each previous level's cluster. In contrast, residual vector quantization generates a global partition and then embeds it into each previous level's cluster. It is similar to the hamming embedding (HE) method, while HE involves two levels and uses the orthogonal partition in each cluster. The simplified structure makes it possible to have more quantization levels and each level have more centroids for fine division of space. The method that transforming tree-like structure to flat structure, which has been used in ferns classifier [25], significant reduces the complexity of index structure while maintaining a fine-grained division of space.

5.3. Complexity

Processing vectors in original high dimensional space causes negative implications for complexity. Operations such as finding the nearest centroid or generating residual vectors are performed in high dimensional space while product quantization process subvectors in the low dimensional subspace. The memory usage of codebook is negligible when compared to the memory occupied by a codeddatabase. The complexity of look-up table computation is also negligible when compared with the complexity of scanning the database's codes. The drawback is the computational complexities of learning and quantization stage of residual vector quantization are linear times of the complexities of product quantization. Our feature work will focus on reducing the complexities of learning and quantization stage.

6. Conclusions

We have introduced residual vector quantization for approximate nearest neighbor search. Two efficient search approaches are proposed based on residual vector quantization. The non-exhaustive search method significantly improves the performance. We evaluate the performance on two structured datasets and one unstructured dataset, and compare our approaches with spectral hashing and product quantization. Our approaches obtain the best results in terms of the trade-off between accuracy, speed and memory usage. Results on structured datasets show our approaches slightly outperform product quantization. For unstructured data, our approaches significant outperform the product quantization.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments. This research is supported by the National Natural Science Foundation of China (NSFC) under Grant No. 60903095 and by the Postdoctoral Science Foundation Funded Project of China under Grant No. 20080440941.

References

1. Beyer, K.; Goldstein, J.; Ramakrishnan, R.; Shaft, U. When is "nearest neighbor" meaningful? In *Proceedings of Database Theory—ICDT'99*, Jerusalem, Israel, January 1999; pp. 217-235.
2. Böhm, C.; Berchtold, S.; Keim, D. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.* (*CSUR*) **2001**, *33*, 322-373.
3. Duan, L.Y.; Guan, T.; Yang, B. Registration combining wide and narrow baseline feature tracking techniques for markerless AR systems. *Sensors* **2009**, *9*, 10097-10116.
4. Silpa-Anan, C.; Hartley, R.; Machines, S.; Canberra, A. Optimised KD-trees for fast image descriptor matching. In *Proceedings of IEEE CVPR 2008*, Anchorage, AK, USA, 24–26 June 2008; pp. 1-8.
5. Nister, D.; Stewenius, H. Scalable recognition with a vocabulary tree. In *Proceedings of IEEE CVPR 2006*, New York, NY, USA, 17–22 June 2006; pp. 2161-2168.

6. Muja, M.; Lowe, D.G. *Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration*; Insticc-Inst Syst Technologies Information Control & Communication: Setubal, Portugal, 2009; pp. 331-340.
7. Datar, M.; Immorlica, N.; Indyk, P.; Mirrokni, V. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of 20th Annual ACM Symposium on Computational Geometery*, New York, NY, USA, June 2004; pp. 253-262.
8. Shakhnarovich, G.; Darrell, T.; Indyk, P. *Nearest-Neighbor Methods in Learning and Vision*: *Theory and Practice*; MIT Press: Cambridge, MA, USA, 2005.
9. Weber, R.; Schek, H.; Blott, S. A quantitative analysis and performance study for similarity—search methods in high-dimensional spaces. In *Proceedings of 24th VLDB Conference*, New York, NY, USA, 24–27 August 1998; pp. 194-205.
10. Koudas, N.; Ooi, B.; Shen, H.; Tung, A. LDC: Enabling search by partial distance in a hyper-dimensional space. In *Proceedings of ICDE 2004*, Boston, MA, USA, 30 March–2 April 2004; pp. 6-17.
11. Cui, B.; Shen, H.; Shen, J.; Tan, K. Exploring bit-difference for approximate KNN search in high-dimensional databases. In *Proceedings of* ADC 2005, Newcastle, Australia, 31 January–3 February 2005; pp. 165-174.
12. Jegou, H.; Douze, M.; Schmid, C. Hamming embedding and weak geometric consistency for large scale image search. In *Proceedings of Computer Vision—ECCV 2008*, Marseille, France, 12–18 October 2008; pp. 304-317.
13. Jégou, H.; Douze, M.; Schmid, C. Packing bag-of-feature. In *Proceedings of ICCV'09*, Kyoto, Japan, 29 September–2 October 2009; pp. 2357-2364.
14. Wang, B.; Li, Z.; Li, M.; Ma, W. Large-scale duplicate detection for web image search. In *Proceedings of IEEE ICME 2006*, Toronto, Canada, 9–12 July 2006; pp. 353-356.
15. Torralba, A.; Fergus, R.; Weiss, Y. Small codes and large image databases for recognition. In *Proceedings of IEEE CVPR 2008*, Anchorage, AK, USA, 24–26 June 2008; pp. 1-8.
16. Weiss, Y.; Torralba, A.; Fergus, R. Spectral hashing. *Adv. Neural Inf. Process. Syst.* **2009**, *21*, 1753-1760.
17. Jégou, H.; Douze, M.; Schmid, C. Product quantization for nearest neighbor search. *IEEE Trans. Patt. Anal. Mach. Int.*, in press.
18. Jegou, H.; Douze, M.; Schmid, C.; Perez, P. Aggregating local descriptors into a compact image representation. In *Proceedings of IEEE Conference on Computer Vision & Pattern Recognition 2010*, San Francisco, CA, USA, 13–18 June 2010.
19. Juang, B.H.; Gray, A.H. Multiple stage vector quantization for speech coding. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Singal Processing*, Paris, France, April 1982; pp. 597-600.
20. Gray, R.; Neuhoff, D. Quantization. *IEEE Trans. Inform.Theory* **1998**, *44*, 2325-2383.
21. *The ANN Evaluation Dataset*; Available online: http://www.irisa.fr/texmex/people/jegou/ann.php (accessed on 19 June 2010).
22. *Matlab Package of Aggregating Local Descriptors into a Compact Representation*; Available online: http://www.irisa.fr/texmex/people/jegou/src/compactimgcodes/index.php (accessed on 25 August 2010).

23. *The INRIA Holidays Dataset*; Available online: http://lear.inrialpes.fr/people/jegou/data.php#holidays (accessed on 27 July 2010).
24. Torralba, A.; Fergus, F.; Freeman, W.T. 80 million tiny images: A large database for non-parametric object and scene recognition. *IEEE Trans. Patt. Anal. Mach. Int.* **2008**, *30*, 1958-1970,.
25. Ozuysal, M.; Fua, P.; Lepetit, V. Fast keypoint recognition in ten lines of code. In *Proceedings of CVPR 2007*, Minneapolis, MN, USA, 18–23 June 2007.