Multimedia Systems
DOI 10.1007/s00530-015-0470-9

REGULAR PAPER

Optimized residual vector quantization for efficient approximate nearest neighbor search

Liefu Ai[1,2] · Junqing Yu[1,3] · Zebin Wu[1] · Yunfeng He[1] · Tao Guan[1]

Received: 24 April 2014 / Accepted: 19 May 2015
© Springer-Verlag Berlin Heidelberg 2015

Abstract In this paper, an optimized residual vector quantization-based approach is presented for improving the quality of vector quantization and approximate nearest neighbor search. The main contributions are as follows. Based on residual vector quantization (RVQ), a joint optimization process called enhanced RVQ (ERVQ) is introduced. Each stage codebook is iteratively optimized by the others aiming at minimizing the overall quantization errors. Thus, an input vector is approximated by its quantization outputs more accurately. Consequently, the precision of approximate nearest neighbor search is improved. To efficiently find nearest centroids when quantizing vectors, a non-linear vector quantization method is proposed. The vectors are embedded into 2-dimensional space where the lower bounds of Euclidean distances between the vectors and centroids are calculated. The lower bound is used to filter non-nearest centroids for the purpose of reducing computational costs. ERVQ is noticeably optimized in terms of time efficiency on quantizing vectors when combining with this method. To evaluate the accuracy that vectors are approximated by their quantization outputs, an ERVQ-based exhaustive method for approximate nearest neighbor search is implemented. Experimental results on three datasets demonstrate that our approaches outperform the state-of-the-art methods over vector quantization and approximate nearest neighbor search.

Keywords Approximate nearest neighbor search · Vector quantization · Codebook optimization · Filtration

Communicated by T. Mei.

✉ Junqing Yu
yjqing@hust.edu.cn

Liefu Ai
ailiefuhu@gmail.com

Zebin Wu
749978334@qq.com

Yunfeng He
yfhe@hust.edu.cn

Tao Guan
qd_gt@126.com

[1] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

[2] School of Computer and Information, Anqing Normal University, Anqing 246133, China

[3] Center of Network and Computation, Huazhong University of Science and Technology, Wuhan 430074, China

1 Introduction

Nearest neighbor (NN) search is very important to many computer vision applications, such as content-based image retrieval [1] and feature matching [2]. However, for high-dimensional vector, exact nearest neighbor search is inherently expensive due to the "curse of dimensionality" [3]. This difficulty has led to the development of the solutions to approximate nearest neighbor (ANN) search. The key idea shared by ANN methods is to find the NN with high probability "only," instead of probability 1 [4].

As the typical partition trees, KD trees [5] perform ANN search by using the best-first search scheme. Besides, researchers have proposed a lot of other tree-based methods, such as randomized KD tree [6–8], approximate *k*-means tree [9], and hierarchical *k*-means tree [10]. FLANN [11] is proposed to automatically determine the best configuration of the hierarchical *k*-means trees and randomized KD trees.

Exact Euclidean Locality Sensitive Hashing (E2LSH) [12] and its variants [13–16] have shown the effectiveness on dense vectors. E2LSH can provide theoretical guarantees on the search accuracy with limited assumptions. However, E2LSH-related methods need to store the original vectors in computer memory to perform final re-ranking use, which seriously limits the scale of the database.

For large amount of data, besides retrieval accuracy, memory usage for storing high-dimensional vectors also is a key criterion for problems involving those data. Recently, proposed hashing coding methods and hamming embedding methods map vectors in Euclidean space into short codes in Hamming space. Then, the vectors are represented by the short codes. The similarity between vectors is measured by the hamming distance between the codes. Such methods include small binary code [17], spectral hashing [18], spherical hashing [19], hamming embedding [20], mini-BOF [21], K-means Hashing [22], etc. These methods make it possible to store large scale vectors in computer memory. While promising, when the number of bits used for encoding vectors is fixed, the possible number of hamming distance is consequently fixed. Therefore, the discrimination of hamming distance is restricted by the length of code.

Recently, neighborhood graph search [23–26] has been attracting a lot of interests in the communities of ANN search. The data structure is a directed graph where each vector is connected with the nearest neighbors. The ANN search generally starts by finding several seed vectors in neighborhood graph and pushes them into a priority queue. Then, the nearest vector in the priority queue is popped, and its nearest neighbors which have not been visited are pushed into the priority queue. The search process terminated until the fixed number of result vectors is obtained. The keys lie in efficiently constructing k-NN graph [23, 24] and effectively ANN search among neighborhood graph [23, 25, 26]. These methods think that the similarity can propagate among vectors. For example, if one vector is close to the query, its nearest neighbor vectors in neighborhood graph are also thought to be close to the query. Because the exact Euclidean distance is generally adopted during ANN search, then the raw vectors should be stored.

Several ANN search methods [4, 27, 28] approach the Euclidean nearest neighbor problem as one of the efficient quantizations. Vectors are represented by their quantization indices; thus, the raw vectors need not be stored. Jegou et al. [4] propose product quantization (PQ)-based ANN search method, where PQ decomposes the vector space into a Cartesian product of low-dimensional sub-spaces. A vector is represented by a short code composed of its sub-space quantization indices. Besides, an asymmetric distance is designed to approximate the distance between two vectors, which outperforms hamming distance in terms of the trade-off between accuracy and search efficiency. Based on PQ, Babenko et al. [29] proposed an inverted multi-index where a vector is inserted into the inverted list according to its PQ codes. Jegou et al. [30] proposed a re-ranking mechanism where a vector is quantized by PQ of two levels and reconstructed by its two PQ codes. Based on PQ, Ge et al. [31] propose an optimized product quantization by minimizing quantization distortions, in which two optimizing methods: a non-parametric method and a parametric method are designed. PQ is based on the assumption that all the components of vectors are statistically independent of each other, while this is not applicable enough for all real data. To resolve this problem, Brandt [27] proposes a quantization technique which combines transform coding (TC) with PQ. The number of bits for encoding each component of vectors is allocated adaptively. To further reduce quantization error, Chen et al. [28] design an ANN search method by using residual vector quantization (RVQ) [32], which reduces the quantization errors with several low-complex stage quantizers. There are also another two similar works [41, 42] recently and introduce method to optimize codebooks and codes of vectors. Extended from k-means, orthogonal k-means and Cartesian k-means are proposed to reduce the encoding costs [33]. Orthogonal k-means can be considered as a generalization of the Iterative Quantization [34]. Built on orthogonal k-means, the Cartesian k-means can be viewed as a generalization of PQ [4]. While promising, most vector quantization approaches, i.e., PQ, TC, and RVQ, adopt the traditional brute-force method to find the nearest centroid when quantizing a vector. Consequently, the time costs on quantizing a vector are linearly dependent on the number of centroids. Although the size of codebook is not too large usually, the time costs would be inevitably enormous for large scale database and high-dimensional vectors.

Extended to our previous work [43], this paper emphasizes on designing accurate codebook for approximately representing the vectors in the database; thus, the accuracy of ANN search can be further improved. Besides, how to efficiently find the nearest centroids in the procedure of quantizing vectors is another focus. The main contributions are summarized in the following.

- Built on RVQ, ERVQ is proposed to train stage codebooks with joint optimization, where each stage codebook is trained for the purpose of minimizing overall quantization errors. Then, more accurate stage codebooks are generated, and the vectors can be approximated by their quantization outputs more accurately. Consequently, the accuracy of ANN search will be further improved.
- A non-linear vector quantization method is presented by filtering non-nearest centroids in 2-dimensional embed-

ded space. When combining with this method, ERVQ is significantly optimized over time efficiency on quantizing vectors compared to traditional method.

This paper is organized as follows: Sect. 2 reviews RVQ briefly and presents ERVQ; Sect. 3 introduces the filtration-based vector quantization method; an exhaustive ANN search method based on ERVQ is designed in Sect. 4; the performance of our approaches and the comparisons with the state-of-the-arts are reported in Sect. 5; and discussion and conclusions are reported in Sect. 6.

2 Enhanced residual vector quantization

2.1 Residual vector quantization

RVQ [28, 32] is a sequential multi-stage quantization technique which consists of several stage quantizers. Except the first stage, the data used to train the stage codebook is the residual vectors generated from the preceding stage quantizer.

The training process of a two-stage RVQ is shown in Fig. 1a. A training vector set $X = \{x_1, x_2, \ldots, x_n\}$ is provided and the k-means clustering algorithm is performed to generate k centroids as the first-stage codebook $C_1 = \{c_{1,1}, c_{1,2}, \ldots, c_{1,k}\}$. Each training vector $\mathrm{x_i}$ then is quantized according to the following formula:

$$\hat{x}_{i,1} = \arg\min_{c_{1,j} \in C_1} d(x_i, c_{1,j})\ (j = 1, 2, \ldots, k), \tag{1}$$

where $\hat{x}_{i,1}$ denotes the first-stage quantization output of x_i; $d(x_i, c_{1,j})$ denotes the Euclidean distance between x_i and the centroid $c_{1,j}$.

To train the second-stage codebook, firstly, the difference between X and the first-stage quantization outputs, called residual vector set $E_1 = \{\varepsilon_{1,1}, \varepsilon_{2,1}, \ldots, \varepsilon_{n,1}\}$, is calculated by the following formula:

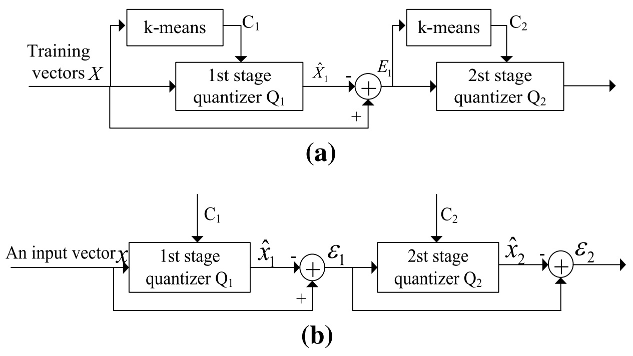


Fig. 1 Illustration of RVQ. **a** Training stage codebook. **b** Quantizing an input vector

$$\varepsilon_{i,1} = x_i - \hat{x}_{i,1} \tag{2}$$

Then, the second-stage codebook $C_2 = \{c_{2,1}, c_{2,2}, \ldots, c_{2,k}\}$ is generated by performing k-means clustering algorithm on E_1. To train a RVQ consisting of L stage codebooks, the above training process needs to repeat L times.

Figure 1b shows the quantizing phase of a two-stage RVQ. An input vector x is first quantized by first-stage codebook C_1. Then, the residual vector between x and its first-stage quantization output $\hat{x}_1$, denoted as ε_1, is quantized by second-stage codebook C_2 to obtain $\hat{x}_2$. This process is sequentially repeated until the last stage for L stages RVQ if $L > 2$. The input vector x can be approximated by its quantization outputs:

$$x = \sum_{l=1}^{L} \hat{x}_l + \varepsilon_L \approx \sum_{l=1}^{L} \hat{x}_l = \hat{x}, \tag{3}$$

where ε_L denotes the last-stage residual vector which is discarded. The quantization output of x is $QO(x) = \{\hat{x}_1, \ldots \hat{x}_l, \ldots, \hat{x}_L\}$. The indices corresponding to $QO(x)$ is represented with $ID(x) = \{ID(x, 1), \ldots, ID(x, l), \ldots, ID(x, L)\}$, where $ID(x, l)$ indicates the indices of centroid corresponding to $\hat{x}_l$.

The overall quantization error of x is computed by formula (4) and the quantization performance of RVQ is measured by mean squared error (MSE) which is calculated by formula (5). It can be seen that the last-stage quantization error just is the overall quantization error. In other words, except the last-stage quantizer of RVQ, the others only consider the quantization error incurred in quantizing vectors applied to that stage quantizer, not the overall quantization error.

$$\begin{aligned} d(x, \hat{x})^2 = \left\| x - \hat{x} \right\|^2 &= \left\| \sum_{l=1}^{L} \hat{x}_l + \varepsilon_L - \hat{x} \right\|^2 \\ &= \left\| \hat{x} + \varepsilon_L - \hat{x} \right\|^2 = \| \varepsilon_L \|^2 \end{aligned} \tag{4}$$

$$MSE = E\left[d(x, \hat{x})^2 \right]. \tag{5}$$

2.2 Enhanced residual vector quantization

A joint optimization process based on RVQ is proposed so that each stage quantizer is designed for the purpose of minimizing the overall distortion. Given stage codebooks trained by RVQ, when optimizing a stage codebook, the principle of optimization process, called joint optimization, is that ERVQ treats the others as fixed ones and the current as the last one which needs to be re-computed.

An optimization process for a stage codebook mainly includes two steps. Firstly, a new codebook is re-computed with the others by minimizing quantization error; then, the new generated stage codebook replaces the old one and is used to optimize the next stage codebook.

Algorithm 1 details the joint optimization of ERVQ. There are six inputs: X,$C = \{C_l\}$, $QO(x_i)$, $R = \{R_l\} = \{\{R_{l,j}\}\}$, ($l$ = 1,…, L, j = 1,…, k), MSE and threshold τ, and the output is the optimized codebook C^*, where the L is the number of stage codebooks; the k is the number of centroids in each stage codebook.

Algorithm 1 Joint optimizing stage-codebooks for ERVQ

```
Inputs: X , $C^{(0)} = \{C_l^{(0)}\}, R^{(0)} = \{R_l^{(0)}\} = \{\{R_{l,j}^{(0)}\}\}$ , $MSE^{(0)}$ , $QO(x_i)^{(0)} = \{\hat{x}_{i,1}^{(0)},...,\hat{x}_{i,L}^{(0)}\}$ , threshold $\tau$,
Output: optimized stage codebooks $C^*$
1   $iter \leftarrow 1$
2   while(TRUE) do
3   for $l = 1 \to L$ do
4       $C^{(iter)} \leftarrow C^{(iter-1)}$
5       $R^{(iter)} \leftarrow R^{(iter-1)}$
6       $QO(x_i)^{(iter)} = QO(x_i)^{(iter-1)}$ (i=1,2,...,N)
7       for $j = 1 \to k$ do
8         computing new centroids by:
              $c_{l,j}^{(iter)} = E[x_{R_{l,j}^{(iter)}} - \sum_{p=1\&\&p\neq l}^{L} \hat{x}_{R_{l,j}^{(iter)},p}^{(iter)}]$
9       end for
10      for $n = 1 \to N$ do
11        $temp = x_n$
12        for $m = 1 \to L$ do
13          if $m < l$ then
14            $\hat{x}_{n,m}^{(iter)} \leftarrow \hat{x}_{n,m}^{(iter-1)}$
15          else
16            $\hat{x}_{n,m}^{(iter)} = \arg\min_{c_{m,w}^{(iter)} \in C_m^{(iter)}} d(temp, c_{m,w}^{(iter)})$ (w=1,2,...,k)
17          end if
18          $temp \leftarrow temp - \hat{x}_{n,m}^{(iter)}$
19          adding identification of $x_n$ into $R_{m,ID(x_n,m)}^{(iter)}$ and
            updating $R_m^{(iter)}$
20        end for
21      end for
22  end for
23  calculating $MSE^{(iter)}$ according to formula (4) and (5)
24  if $\frac{MSE^{(iter-1)} - MSE^{(iter)}}{MSE^{(iter-1)}} < \tau$ then
25    break
26  else
27    $iter \leftarrow iter + 1$
28    continue
29  end if
30  End while
31  $C^* \leftarrow C^{(iter)}$
```

The $X = \{x_1, \ldots, x_i, \ldots, x_N\}$ is the training set of N vectors and N is the size of training set. The C represents initial L stage codebooks trained by RVQ where the C_l denotes the lth stage codebook, and the jth centroid in the lth stage codebook is denoted as $c_{l,j}$. The $QO(x_i)$ represents the quantization outputs of vector x_i. The $R_{l,j}$ denotes the vectors whose quantization outputs in lth stage codebook is the jth centroid $c_{l,j}$. The $R_{l,j}$ of lth stage codebook are united together and denoted as the set $R_l = \{R_{l,j}\}$. All the R_l of L stage codebooks are united together and denoted as the set $R = \{R_l\}$. The MSE is computed by formula (4) and (5). The threshold τ represents the convergence condition to decide whether the optimization process continues.

For simplicity, the parenthesized superscript (iter) is used to indicate the variables in the optimization process of nth iteration, where the initial value $iter = 0$. Correspondingly, all above initial variables are denoted as $C_l^{(0)}$, $R_{l,j}^{(0)}$, $QO(x_i)^{(0)}$, and $MSE^{(0)}$. The joint optimization of ERVQ is detailed in Algorithm 1.

The joint optimization is an iteration process. At each iteration, each stage codebook $\{c_{l,j}^{(iter)}\}$ is re-computed sequentially from stage 1 to stage L (line 7 to 9). To re-compute the jth centroid in the lth stage codebook, firstly, the residual vectors between the vectors indicated by $R_{l,j}^{(iter)}$ and their quantization outputs associated to the other $L - 1$ stage codebooks are computed; then, according to the formula showed at line 8 in algorithm 1, the mean of these residual vectors are computed as the new centroid. When all the k centroids of lth codebook are re-computed, computing the quantization outputs $QO(x_i)^{(iter)}$ for all x_i and updating the $R_{l,j}^{(iter)}$ associated to all stage codebooks (line 10 to 21).

When one iterative optimization is done, the rate $\frac{MSE^{(iter-1)} - MSE^{(iter)}}{MSE^{(iter-1)}}$ is calculated. If it is smaller than the presetting threshold τ, the optimization process is terminated, and the optimized stage codebooks C^* are output. Otherwise, the optimizing process continues. Also, instead of using a threshold, other stopping criteria can be applied to algorithm 1, such as maximum number of iterations.

We found a similar work [35] on optimizing stage codebooks which needs to update the quantization outputs associated to all stage codebooks when a new stage codebook is re-computed completely. Actually, since the new stage codebook $C_l^{(iter)}$ does not influence the preceding $l - 1$ stage quantization outputs of x_i, only the quantization outputs from the current to the last stage need to be updated. Thus, comparing to updating quantization outputs associated to all the stage quantizers, the computing costs can be significantly reduced if only updating the quantization outputs from the current to the last stage.

In practice, when training the initial L stage codebooks for ERVQ, the errors, computed by formula (4) and (5), become smaller and smaller with increasing l, and the difference of errors between lth stage and $(l+1)$th stage also tends to be smaller and smaller. Then, we can conclude that the codebook of RVQ is more important than its succeed one. We consider that this conclusion can be applied to ERVQ since ERVQ also quantizes a vector from stage 1 to stage L sequentially. Therefore, at each iteration during optimizing codebooks for ERVQ, we optimize the codebook sequentially from stage 1 and stage L according to their importance.

Given a vector x, the quantization procedure is similar to that of RVQ shown in Fig. 1b. The vector x is first quantized with the first-stage codebook; then, the residual vector between x and its quantization output is used as the input vector for the succeeding quantizer and another quantization output is obtained. This process is repeated until the last-stage quantizer. Consequently, the L quantization outputs $\{\hat{x}_1, \ldots \hat{x}_i, \ldots, \hat{x}_L\}$ of x are obtained.

3 Non-linear vector quantization

The core of vector quantization is to find the exact nearest centroid among a given codebook, while current quantization methods, such as PQ, RVQ, etc., adopt the brute-force match. The quantization efficiency is an important criterion for quantization methods. Inspired from [36] where an efficient method of finding exact nearest neighbor with sub-linear complexity is proposed for image classification, this paper presents a non-linear vector quantization method aiming at efficiently finding the nearest centroid. Vectors are embedded into a low-dimensional space and the non-nearest centroids are filtered in the embedded space. By combining with this method, ERVQ will gain better time efficiency on quantizing vectors.

3.1 Lower bound of Euclidean distance between vectors

Given a d-dimensional vector $x = (x_1, x_2, \ldots, x_d)$, the mean and standard deviation of components in x can be, respectively, calculated by $\mu_x = \frac{1}{d}\sum_{i=1}^{d} x_i$ and $\sigma_x = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu_x)^2}$. Given another d-dimensional vector y, the lower bound of Euclidean distance $d(x, y)$ between x and y is estimated by $\mu_x, \mu_y, \sigma_x, \sigma_y$, where the theorem is shown as the following:

Theorem 1

$$d(x,y)^2 \geq \|x\|^2 + \|y\|^2 - 2d\left(\mu_x\mu_y + \sigma_x\sigma_y\right)$$

Proof

$$d(x,y)^2 = \|x-y\|^2 = \|x\|^2 + \|y\|^2 - 2\langle x, y\rangle \tag{6}$$

$$\begin{aligned}\langle x, y\rangle &= \sum_{i=1}^{d} x_i y_i = \sum_{i=1}^{d}(x_i - \mu_x + \mu_x)\left(y_i - \mu_y + \mu_y\right)\\ &= \sum_{i=1}^{d}(x_i - \mu_x)\left(y_i - \mu_y\right) + \sum_{i=1}^{d}(x_i - \mu_x)\mu_y\\ &\quad + \sum_{i=1}^{d}\left(y_i - \mu_y\right)\mu_x + d\mu_x\mu_y\end{aligned} \tag{7}$$

Due to $\sum_{i=1}^{d}(x_i - \mu_x)\mu_y = 0$ and $\sum_{i=1}^{d}(y_i - \mu_y)\mu_x = 0$, the Eq. (7) can be transformed as

$$\langle x, y\rangle = \sum_{i=1}^{d}(x_i - \mu_x)\left(y_i - \mu_y\right) + d\mu_x\mu_y. \tag{8}$$

According to Cauchy–Schwarz inequality, the first term of the Eq. (8) in the right can be further transformed as

$$\begin{aligned}\left|\sum_{i=1}^{d}(x_i - \mu_x)(y_i - \mu_y)\right| &\leq \sqrt{\sum_{i=1}^{d}(x_i - \mu_x)^2 \sum_{i=1}^{d}(y_i - \mu_y)^2}.\\ &= d\sigma_x\sigma_y\end{aligned} \tag{9}$$

This leads to

$$d(x,y)^2 \geq \|x\|^2 + \|y\|^2 - 2d\left(\mu_x\mu_y + \sigma_x\sigma_y\right). \tag{10}$$

The lower bound of $d(x, y)^2$ involves $\|x\|^2$, $\|y\|^2$ and the inner product between (μ_x, σ_x) and (μ_y, σ_y).

3.2 Finding the nearest centroid in vector quantization

When quantizing a vector x, the key lies in finding the nearest centroid in the given codebook. If only considering the order of distance, the value $\|x\|^2$ is a constant for all centroids in a codebook. Thus, whether computing $\|x\|^2$ does not influence its quantization output, so it is not necessary to compute $\|x\|^2$. Therefore, the lower bound of Euclidean distance between x and a centroid y, $lb(x, y)$, can be defined as

$$lb(x,y) = \|y\|^2 - 2d\left(\mu_x\mu_y + \sigma_x\sigma_y\right). \tag{11}$$

Corresponding to $lb(x, y)$, the square Euclidean distance between x and a centroid y, $\tilde{d}(x, y)^2$, is defined as

$$\tilde{d}(x,y)^2 = d(x,y)^2 - \|x\|^2 = \|y\|^2 - 2\langle x, y\rangle. \tag{12}$$

Consequently, the Theorem 1 can be transferred as Theorem 2 in the following:

Theorem 2

$$\tilde{d}(x,y)^2 \geq lb(x,y)$$

For all the vectors to be quantized, the length of centroid, $\|y\|^2$, needs to be computed in order to find the nearest centroids. So, the $\|y\|^2$ can be calculated off-line when the codebook is trained completely, and stored as a look-up table. Thus, the value $\|y\|^2$ can be obtained by accessing the look-up table quickly if needed.

Consequently, in the phase of quantizing vector x, the on-line time cost on computing lower bound $lb(x, y)$ mainly involves in calculating the inner product between two 2-dimensional vectors: (μ_x, σ_x) and (μ_y, σ_y). Also, the on-line computation costs of $\tilde{d}(x, y)^2$ mainly lie in computing the inner product between vector x and y.

On the basis of Theorem 2, Algorithm 2 details the process of quantizing a vector with lower bound. There are six inputs: $C = \{c_i\}, \mu = \{\mu_{c_i}\}, \sigma = \{\sigma_{c_i}\}, i = 1, \ldots, k,\ x,\ \mu_x,\ \sigma_x$. The c_i indicates the ith centroid; the μ_{c_i} and σ_{c_i}, respectively, indicate the mean and the standard deviation of c_i; the x, μ_x, and σ_x, respectively, represent the input vector, the mean, and the standard deviation of x.

At the beginning, a centroid is randomly selected from the codebook C as initial nearest centroid, $c^{nearest}$, and the distance $\tilde{d}(x, c^{nearest})^2$ to the input vector is set as the minimum distance.

At each iteration, the nearest centroid $c^{nearest}$ that has been computed so far is maintained. During this process, the lower bound $lb(x, c_i)$ can efficiently filter the centroids c_i when $lb(x, c_i) \geq \tilde{d}(x, c^{nearest})^2$. Comparing to computing Euclidean distance in original space, computing inner product in embedded 2-dimensional space can significantly reduce the time costs. Therefore, it would be a huge reduction in computational cost especially when the vectors are high-dimensional. Comparing with brute-force method which needs to compute distances between x and all centroids, Algorithm 2 obtains the exact nearest centroid by only computing part of those. So it is non-linearly dependent on the scale of codebook.

When applying Algorithm 2 to ERVQ, it describes the process that an input vector is quantized by a stage quantizer. Of course, Algorithm 2 also can be used to optimize other quantization methods conveniently, such as RVQ and PQ.

On the other hand, when $lb(x, c_i) < \tilde{d}(x, c^{nearest})^2$, both $lb(x, c_i)$ and $\tilde{d}(x, c_i)^2$ need to be computed. In the best case, the lower bounds $lb(x, c_i)$ are always larger than the minimum distance min_d, and the distance $\tilde{d}(x, c_i)^2$ needs to be computed once to find the nearest centroid. On the contrary, both $lb(x, c_i)$ and $\tilde{d}(x, c_i)^2$ need to be computed k times. Both of such extreme cases seem unlikely to happen on real data.

Algorithm 2 Vector Quantization with Lower Bound

```
Inputs:  codebook $C=\{c_i\},\mu=\{\mu_{c_i}\},\sigma=\{\sigma_{c_i}\},i=1,...,k$ ,
         input vector x, $\mu_x$, $\sigma_x$
Output: nearest centroid $c^{nearest}$
1   Selecting a centroid $c'$ from C as the seed $c^{nearest}$
2   Calculating the distance $\min\_d \leftarrow \tilde{d}(x,c^{nearest})^2$
3   for $i=1 \rightarrow k$
4       if $lb(x,c_i) \geq \min\_d$ then
5           continue
6       else
7           Computing $\tilde{d}(x,c_i)^2$
8           if $\tilde{d}(x,c_i)^2 \geq \min\_d$ then
9               continue
10          else
11              $c^{nearest} \leftarrow c_i$
12              $\min\_d \leftarrow \tilde{d}(x,c_i)^2$
13          end if
14      end if
15  end for
16  return $c^{nearest}$
```

4 Exhaustive ANN search by ERVQ

To evaluate the accuracy that vectors are approximate by their quantization outputs, exhaustive search for ANN is implemented with fast distance computation.

In [4, 27, 28], the exact Euclidean distance between two vectors x and y is approximated by the asymmetric distance which indicates the distance between x with the quantization output of y:

$$d(x, y) \approx \hat{d}(x, y) = d(x, \hat{y}). \tag{13}$$

Suppose vector y in the database is quantized by L stages ERVQ with the process shown in Fig. 1b and Algorithm 2, the quantization outputs of y are $\{\hat{y}_1, \ldots, \hat{y}_L\}$. Then, y is approximate by the sum $\hat{y}$ of its quantization outputs according to formula (3).

Given a query vector x, similar to [17], the following formula is used to compute the square asymmetric distance from query vector x to vector y in the database:

$$\begin{aligned} d(x, \hat{y})^2 &= \|x - \hat{y}\|^2 = \|x\|^2 + \|\hat{y}\|^2 - 2\langle x, \hat{y}\rangle \\ &= \|x\|^2 + \|\hat{y}\|^2 - 2\left\langle x, \sum_{i=1}^{L} \hat{y}_i \right\rangle \\ &= \|x\|^2 + \|\hat{y}\|^2 - \sum_{i=1}^{L} 2\langle x, \hat{y}_i\rangle \end{aligned}, \tag{14}$$

where $\langle x, \hat{y}_i\rangle$ is the inner product; $\|\hat{y}\|^2$ is pre-computed when the database vector is quantized completely. $2\langle x, \hat{y}_i\rangle$ can be computed and stored in a look-up table when x is submitted. The output $\hat{y}_i$ is the centroids in the ith stage codebook, which is nearest to the corresponding input residual vector. The look-up table only needs to be constructed once for the query vector x.

If only considering the order of distance, the term $\|x\|^2$ in formula (14) is a constant for all database vectors and can be ignored when computing $d(x, \hat{y})^2$. Then, the computation of $d(x, \hat{y})^2$ mainly involves in accessing the look-up table. *knn* search results are selected based on the estimated $d(x, \hat{y})^2$.

5 Experiments

All the experiments are measured on a machine with Xeon 16 cores 2.4GHZ CPU and 128 GB of RAM.

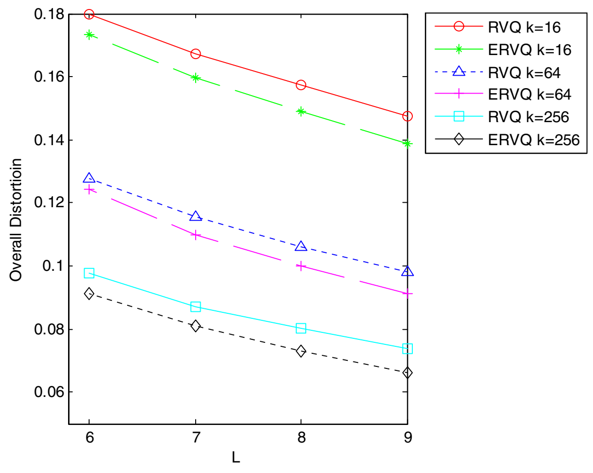
5.1 Datasets

Three publicly available datasets, GIST vector dataset, SIFT-1 M vector dataset [37], and SIFT-1B vector dataset [30], are used to evaluate the performance. SIFT descriptor codes small image patch, while GIST descriptor codes entire image. SIFT descriptor is a histogram of oriented gradients that extracted from gray image patch. GIST descriptor is similar to sift applied to entire image. It applies an oriented Gabor filter over different scales and averages the filter energy in each bin.

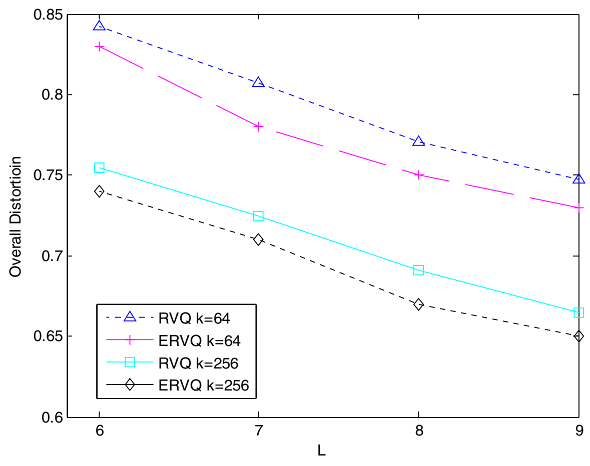
All those datasets have three subsets: learning set, database set, and query set. The learning set is used to train stage codebooks; the database and query sets are used for evaluating quantization performance and ANN search performance. For SIFT-1 M dataset, the learning set is extracted from Flicker images [38] and the database and query vectors are extracted from INRIA holidays images [39]. For GIST dataset, the learning set consists of the tiny image set of [40]. The database set is holidays image set combined with Flicker 1 M [38]. The query vectors are extracted from the holidays image queries [39]. SIFT-1B is release in [30]. All the descriptors are high-dimensional float vectors. The details of datasets are given in Table 1.

Table 1 Summary of the used datasets

Information	SIFT-1 M	GIST	SIFT-1B
Dimension	128	960	128
Size of learning set	100,000	500,000	100,000,000
Size of database set	1,000,000	1,000,000	1,000,000,000
Size of query set	10,000	1,000	10,000



(a)



(b)

Fig. 2 The comparison of overall quantization errors. **a** The overall quantization errors on SIFT-1 M database. **b** The overall quantization errors on GIST database

5.2 Comparison of quantization performance

To implement easily, when training the codebook of ERVQ, we set the total number of iteration (the number is set as 30) as the convergence condition instead of using a pre-set threshold.

Figure 2 compares the overall quantization errors produced by RVQ and ERVQ when quantizing vectors over SIFT-1 M and GIST. The horizontal axis represents the number of stage codebooks, $L\{6, 7, 8, 9\}$. The vertical axis represents the overall distortion which indicates the overall quantization errors. The k denotes the number of centroids in each stage codebook of RVQ or ERVQ. Then, the length of code that memory is required to store a vector when quantizing it can be computed correspondingly, $n\text{bits} = L \log_2 k$.

As expected, with increasing L or k, both the overall quantization errors produced by ERVQ and RVQ reduces correspondingly. On both SIFT-1 M and GIST database, on the condition of the same k and L, ERVQ has lower overall quantization error than RVQ. Take SIFT-1 M database as example, when $k = 256, L = 8$, the overall quantization error produced by RVQ and ERVQ are 0.0803, 0.0730, respectively, and the reduction rate of overall distortion reaches $0.0803 - 0.0730/0.0803 \approx 9.1\%$. It can be concluded that ERVQ produces lower quantization error than RVQ when quantizing vectors with the same length of code.

When using the same k, the overall quantization error produced by ERVQ with L is close to even on par with that of RVQ with $L + 1$. For instance, on SIFT-1 M database, overall quantization errors produced by ERVQ with $L = 8$ and RVQ with $L = 9$ are 0.0730 and 0.0736; on 1 M GIST database, overall quantization errors produced by ERVQ with $L = 8$ and RVQ with $L = 9$ are 0.67 and 0.666.

5.3 Comparison of time efficiency on vector quantization

To compare with the state-of-the-arts over time efficiency, the nearest neighbor search method in [36] is implemented and applied into ERVQ. This algorithm is implemented by two realizing ways. In the first way, we realize this algorithm by matlab script, while matlab is inefficient on loop code at the runtime. Thus, in the second way, MEX tool is used to realize this algorithm by combining matlab script and C programming, which combines the advantage of MATLAB on matrix computation and C programming on loop code at the runtime. To distinguish these two realized ways, these two realized ways are named with ERVQ with FNN and ERVQ with FNN-M, respectively. They are two realizing ways for the same algorithm, such as the loop code in C programming can be realized by the code of ‘For’ or ‘While.’

ERVQ with FNN-M is used as the baseline

For the sake of clarity, ERVQ with our vector quantization method is indicated as **ERVQ with no-linear NN** and ERVQ with traditional brute-force method is indicated as **ERVQ**.

Figure 3 shows the time costs on quantizing vectors by above methods with various k over SIFT-1 M and GIST databases. The k denotes the number of centroids in each stage codebook, which ranges in {16, 64, 256}. The number of stage codebooks of ERVQ, L, is set as 8.

It can be seen that ERVQ with non-linear NN spends less time than ERVQ, ERVQ with FNN-M, and ERVQ with FNN obviously. With efficiently implementation, ERVQ with FNN-M can significantly reduce time costs compared to ERVQ with FNN. Besides, although the quantization time increases correspondingly with increasing k or increasing dimensionality, the time costs of ERVQ with non-linear NN and ERVQ with FNN-M grow more slowly.

Therefore, It can be drawn to the conclusion that, compared to ERVQ, ERVQ with non-linear NN can significantly improve time efficiency on vector quantization by reducing the computation costs on finding nearest centroids, especially for large k and high-dimensional vectors.

By pre-computing the squared length ($\|c\|^2$) of each centroid c in L stage codebooks and stored in a look-up table when training process is completed, ERVQ with non-linear NN only needs to compute the inner product when computing Euclidean distance and lower bound between vectors and centroids when quantizing vectors. Given a D-dimensional vector and a D-dimensional centroid, the complexities of ERVQ with FNN-M and ERVQ with non-linear NN are both O(D) on computing Euclidean distance and inner product and both O(2) on computing lower bound and inner product for it. However, in practice, at the runtime, since computing Euclidean distance needs more CPU operations, it takes more CPU computation costs on calculating Euclidean distance than inner product. Under the same experimental conditions, taking SIFT-1 M database with $k = 256$ as example, ERVQ with FNN-M takes 94.52 s to quantizing the database, while ERVQ with non-linear NN takes 62.41 s. For ERVQ with non-linear NN, the time cost on computing the length of centroids in L stage codebooks is 0.0703 s. Then, the total cost of ERVQ with non-linear is still less than that of ERVQ with FNN-M. Therefore, we can conclude that ERVQ with non-linear can further improve the quantization efficiency with small amount of extra memory compared to ERVQ with FNN-M.

5.4 Comparison of exhaustive ANN search

Exhaustive ANN search is exploited to estimate the accuracy that vectors are approximated by their quantization outputs. Recall@R is used to measure the search accuracy, where the R represents the number of retrieved vectors. Recall@R is defined in [4] as the proportion of query vectors for which the nearest neighbor is ranked in the first R position. The larger the recall@R is, the better the search accuracy is.

We compare our approach with four state-of-the-art ANN search methods: RVQ-based exhaustive ANN search [28], PQ-based exhaustive ANN search [4], spectral hashing [18], and Cartesian k-means, which are, respectively, indicated as RVQ, PQ, SH, and ck-means. Correspondingly, our ERVQ-based exhaustive ANN search method is indicated as ERVQ.

In the exhaustive ANN search experiments, for PQ and ck-means, we use $m = 8$ and $k = 256$. Hence, $n\text{bits} = m \log_2 k = 64$-bits are used as database indices. For RVQ and

ERVQ, we both use $k = 256$, $L = 8$ and $k = 256$, $L = 9$. Hence, according to $n\text{bits} = L\log_2 k$, 64-bits and 72-bits are used as database indices. For SH, we use nbits = 64-bits to encode vectors in database. Given a query vector, PQ, ck-means, RVQ, and ERVQ use asymmetric distance to measure the similarity between query and database vectors, while SH uses the hamming distance.

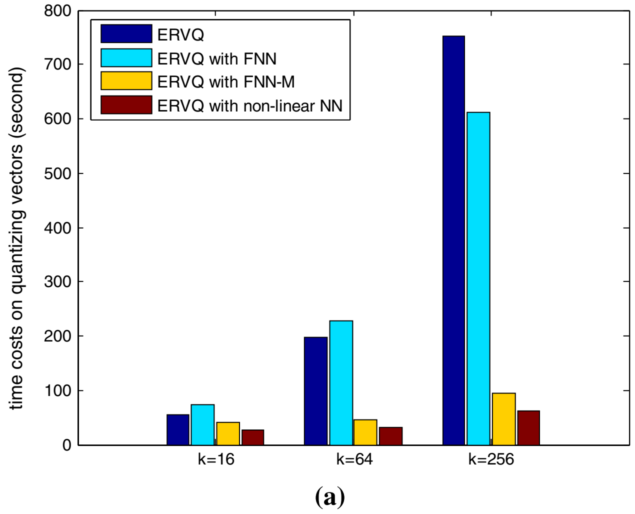
Figure 4 shows Recall@R plots for those five exhaustive ANN search methods on SIFT-1 M and GIST. Tables 2 and 3 detail the corresponding exhaustive ANN search performance.

On both SIFT-1 M and GIST, ERVQ outperforms the RVQ, PQ, and SH over search accuracy. ERVQ also is superior to ck-means on SIFT-1 M database, while ERVQ is comparable to ck-means on 1 M GIST dataset. Moreover, even ERVQ with 64-bits performs on par with RVQ with 72-bits on both SIFT-1 M and GIST database, which can be easily observed in Fig. 4 and Tables 2 and 3.

Over time efficiency, ERVQ outperforms RVQ and SH in terms of search time under the same length of code. Compared to ck-means, ERVQ has similar time efficiency on ANN search.

The search efficiency of PQ is slightly higher than that of RVQ and ERVQ. The reason lies in that PQ uses the sub-vector to construct the look-up table, while RVQ and ERVQ use the whole vector to build the look-up table.

During iteratively optimizing stage codebooks for ERVQ, the overall distortion turns to be smaller and smaller with the increasing stage. Therefore, after re-quantizing the training vectors, there are possibilities that no residual vector is allocated to some centroids in last several stage codebooks. Then, these centroids are considered as outliers and discarded. Consequently, when the training process



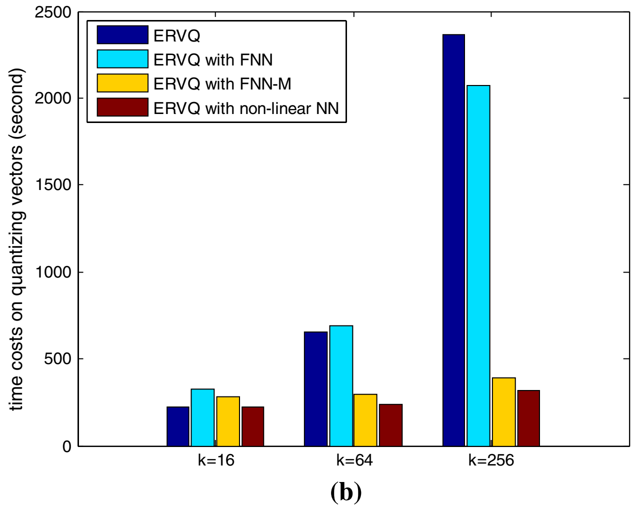
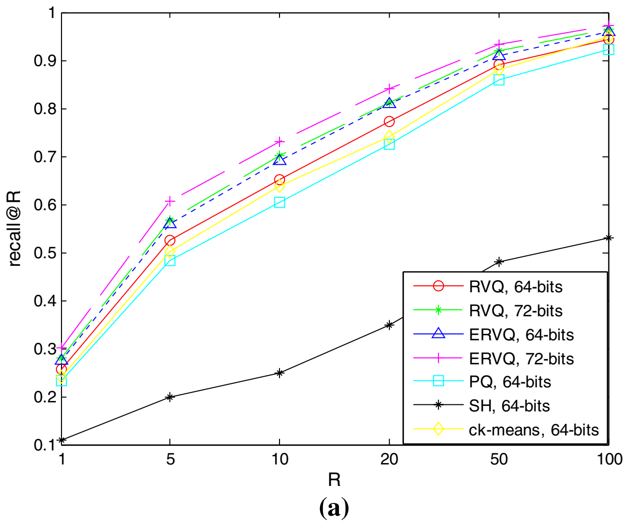


Fig. 3 The comparison of time costs on quantizing vectors. **a** SIFT-1 M database. **b** GIST database



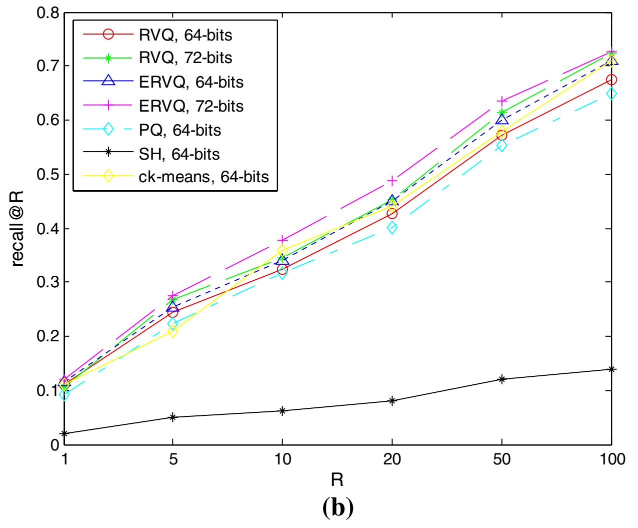


Fig. 4 The comparison of exhaustive search for ANN. **a** Recall@R with varying R on SIFT-1 M dataset. **b** Recall@R with varying R on GIST dataset

Table 2 Ann search performance on SIFT-1 M

Methods	Parameters	Search time (ms)	Recall@100
RVQ	64-bits, $k = 256$, $L = 8$	33.9	0.94
RVQ	72-bits, $k = 256$, $L = 9$	35.7	0.96
ERVQ	64-bits, $k = 256$, $L = 8$	31.9	0.96
ERVQ	72-bits, $k = 256$, $L = 9$	33.8	0.97
PQ	64-bits	29.0	0.92
ck-means	64-bits	30.9	0.95
SH	64-bits	35.1	0.53

Table 3 Ann search performance on GIST

Methods	Parameters	Search time (ms)	Recall@100
RVQ	64-bits, $k = 256$, $L = 8$	37.0	0.68
RVQ	72-bits, $k = 256$, $L = 9$	39.9	0.72
ERVQ	64-bits, $k = 256$, $L = 8$	36.2	0.71
ERVQ	72-bits, $k = 256$, $L = 9$	38.5	0.73
PQ	64-bits	34.5	0.65
ck-means	64-bits	35.6	0.71
SH	64-bits	38.3	0.14

is completed, the number of centroids in some stage codebooks may be smaller than the initial number k. Therefore, compared with RVQ, ERVQ takes fewer time costs on building look-up table for a query vector and spends less the search time.

Figure 5 shows the Recall@R plots for four exhaustive ANN search methods on SIFT-1B dataset. All the methods use 64-bits to encode vectors in database. The experimental results are consistent with that on SIFT-1 M and GIST, which favors ERVQ.

In the above ANN search experiments, we use linear scan to find the nearest items according to asymmetric distances. Search can be sped up significantly by using a coarse initial quantization and an inverted file structure, as suggested by [4, 28, 29]. In this paper, we did not implement these efficiencies as we focus primarily on training more accurate codebooks and improving the accuracy of approximating vectors.

5.5 Time costs on training codebooks and corresponding ANN search performance

Tables 4 and 5 show the time costs on training codebooks of ERVQ and RVQ on SIFT-1 M and GIST. The parameter that denotes the number of centroids in each RVQ's codebook and ERVQ's initial codebook is both set as $k = 256$. The parameter L that denotes the number of stage codebooks of RVQ and ERVQ is set as $L = 8$. For RVQ, the iteration number *niter* of k-means algorithm ranges in {100, 130, 150, 200}. For ERVQ, the iteration number *niter1* of k-means algorithm in training initial codebook for ERVQ is set as 100, and the iteration number *niter2* of optimizing codebooks is set as 30.

The time costs of training ERVQ involves in two aspects: time costs on training initial codebooks and time costs on optimizing the codebooks. When updating the quantization outputs of training vectors during optimization process, the method presented in Sect. 3 is used to find the nearest centroids. Compared to training initial codebooks, optimizing codebooks for obtain more accurate codebooks takes more time.

For RVQ, k-means algorithm is used to train codebook and the parameter *niter* is used as the convergent criterion. If *niter* reaches the pre-set value, k-means is considered to be convergence and stops. Actually, once the iteration of k-means algorithm converges, the centroids will not change if the iterative process continues (by increasing *niter* value).

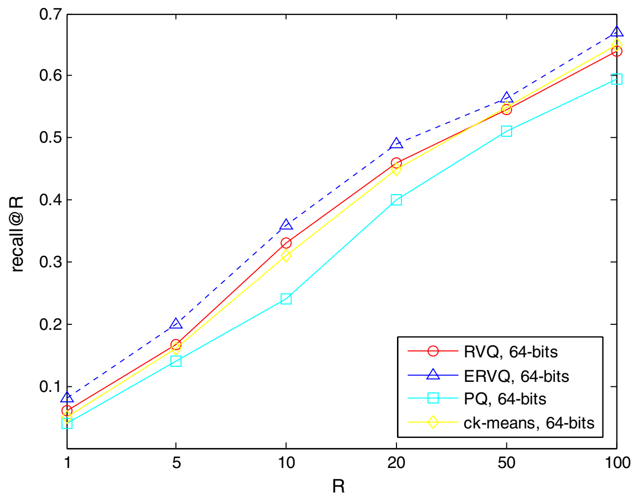


Fig. 5 Recall@R for exhaustive ANN search on SIFT-1B

Table 4 Time costs and distortion on training codebooks over SIFT-1 M

Methods	Parameters	Overall distortion	Time costs (s)
RVQ	niter = 100	0.803	169.75
RVQ	niter = 130	0.801	210.06
RVQ	niter = 150	0.803	245.85
RVQ	niter = 200	0.802	302.28
ERVQ	niter1 = 100, niter2 = 30	0.730	876.82

Table 5 Time costs and distortion on training codebooks over GIST

Methods	Parameters	Overall distortion	Time costs (s)
RVQ	niter = 100	0.698	3816.43
RVQ	niter = 130	0.70	4905.12
RVQ	niter = 150	0.697	5654.35
RVQ	niter = 200	0.695	7179.76
ERVQ	niter1 = 100, niter2 = 30	0.670	16125.13

Table 6 Ann search performance on SIFT-1 M under different *niter*

Methods	Parameters	Search time (ms)	Recall@100
RVQ	64-bits, niter = 100	33.9	0.94
RVQ	64-bits, niter = 130	33.6	0.96
RVQ	64-bits, niter = 150	33.7	0.94
RVQ	64-bits, niter = 200	33.3	0.94
ERVQ	64-bits, niter1 = 100, niter2 = 30	31.9	0.96

Table 7 Ann search performance on GIST under different *niter*

Methods	Parameters	Search time (ms)	Recall@100
RVQ	64-bits, niter = 100	37.0	0.68
RVQ	64-bits, niter = 130	37.2	0.68
RVQ	64-bits, niter = 150	36.7	0.68
RVQ	64-bits, niter = 200	36.8	0.68
ERVQ	64-bits, niter1 = 100, niter2 = 30	36.2	0.71

Tables 6 and 7 are the ANN search performance corresponding to the codebooks of the RVQ and ERVQ displayed in Tables 4 and 5.

From the empirical results shown in Tables 6 and 7, we can see that the recall@100 of RVQ ANN search with *niter* $\in \{130, 150, 200\}$ is the same as that of niter = 100. That means the k-means converges when niter = 100, and the centroids will not change with increasing niter (niter > 100). Consequently, the RVQ ANN search will not be influenced. Thus, we can conclude that RVQ ANN search cannot be improved by increasing the iteration number once k-means algorithm has converged. On the other hand, the performance of RVQ ANN search will not be improved by increasing training costs if the training process has converged.

6 Discussion and conclusions

In this paper, we presented an optimized residual vector quantization for ANN search. Unlike training stage codebooks sequentially in RVQ, ERVQ iteratively optimizes one-stage codebook by the others, so that every stage codebook is trained for minimizing the overall quantization error. Consequently, the accuracy of ANN search is improved. Besides, aiming at efficiently finding nearest centroids, a non-linear vector quantization method is proposed. By filtering non-nearest centroids in 2-dimensional space, this method can reduce the time costs on vector quantization significantly and be applied to ERVQ conveniently. Comparing with brute-force method and the state-of-the-art, experiments show that our method noticeably increases the time efficiency on quantizing vectors.

Relation to additive quantization [41] Both Additive Quantization and ERVQ encode vectors with several different codebooks and approximate vectors with the sum of centroids selected from several codebooks.

However, there are still differences. Given trained codebooks, Additive Quantization encodes the vectors in database by Beam Search algorithm. In this process, some candidates are maintained until to the last quantizer. Then, the best quantization outputs are selected from these candidates. On the contrary, our approach is completely different. ERVQ quantized a vector from the first quantizer to the last quantizer sequentially, and the best result associated to each quantizer constitutes the vector's total quantization outputs. In view of training codebooks, to our knowledge, Additive Quantization begins with random assignment and generates codebooks by combining with the used encoding mechanism and its overstrained system of linear equation. ERVQ begins codebooks training with several stage codebook obtained with RVQ, and then adopts joint optimization to generate new stage codebooks.

Relation to composite quantization [42] Similarity, both Composite Quantization and ERVQ encode vectors with several different codebooks and approximate vectors with the sum of centroids selected from several codebooks.

However, there are still differences. To reduce the computation cost on building the look-up table which is needed in the Euclidean distance computation equation, Composite Quantization makes an extra constraint, while ERVQ avoids this constraint by computing the squared length of vectors in the database at encoding procedure. To our knowledge, Composite Quantization iteratively updates codebooks and codes of vectors over the whole database, while ERVQ only optimizes codebooks over training set, and adopts sequential quantization mechanism to encode vectors in database. Also, the methods used for optimization are different.

While promising, some issues that should be further studied to improve our methods are discussed as follows:

1. Exhaustive ANN search has to scan all the vectors in database. When involving in the problems such as

bag-of-visual-words-based large scale image retrieval, billions of image are represented by hundreds even thousands of local feature vectors per image. It is prohibitive to scan the whole database. In the future work, we will focus on efficiently building inverted indexing structure with stage codebooks and evaluating non-exhaustive ANN search performance.

2. In this paper, ERVQ is presented for the purpose of improving the accuracy of stage codebooks. Due to the iterative optimization process, ERVQ takes longer training time than RVQ does. In the future, we will try to combine ERVQ with PCA by considering both the error of dimension reduction and overall quantization error in the process of optimizing codebook. Accordingly, the stage codebooks are trained in a low-dimensional space with less training costs and unchanged search accuracy.

Acknowledgments This paper is financially supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61173114, 61202300, and 61272202.

References

1. Sivic, J., Zisserman, A.: Video Google: a text retrieval approach to object matching in video. In: ICCV, pp. 1470–1477 (2003)
2. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. IJCV **60**(2), 91–100 (2004)
3. Bohm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. ACM Comput. Surv. **33**(3), 322–373 (2001)
4. Jegou, H., Matthijs, D., Cordelia, S.: Product quantization for nearest neighbor search. IEEE Trans. PAMI **33**(1), 117–128 (2011)
5. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**(9), 509–517 (1975)
6. Silpa-Anan C., Hartley R.: Optimised kd-trees for fast image descriptor matching. In: CVPR, pp. 1–8 (2008)
7. Jia Y., Wang J., Zeng G., Zha H., Hua X. S.: Optimizing kd-trees for scalable visual descriptor indexing. In: CVPR, pp. 3392–3399 (2010)
8. Wang, J., Wang, N., Jia, Y., Li, J., Zeng, G., Zha, H., Hua, X.S.: Triary-projection trees for approximate nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell. **36**(2), 388–403 (2014)
9. Philbin J., Chum O., Isard M., Sivic J., Zisserman A.: Object retrieval with large vocabularies and fast spatial matching. In: CVPR, pp. 1–8 (2007)
10. Nister D., Stewenius H.: Scalable recognition with a vocabulary tree. In: CVPR, pp. 2161–2168 (2006)
11. Muja M., Lowe D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. In: VISSSAPP, pp. 331–340 (2009)
12. Data, M., Immorlica, N., Indyk, P., Mirrokni, D.V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Symposium on Computational geometry, pp. 253–262 (2004)
13. Panigrahy, R.: Entropy based nearest neighbor search in high dimensions. In: ACM-SIAM SODA, pp. 1186–1195 (2006)
14. Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li, K.: Multi-Probe LSH: Efficient indexing for high-dimensional similarity search. In: VLDB, pp. 950–961 (2007)
15. Kuo, Y. H., Chen, K.T.C., Chiang, H., Hsu, W.H.: Query expansion for hash-based image object retrieval. In: ACM Conference on Multimedia, pp. 65–74 (2009)
16. Jegou, H., Amsaleg, L., Schmid, C., Gros, P.: Query-adaptative locality sensitive hashing. In: Conference on ICASSP, pp. 825–828 (2008)
17. Torralba, A., Fergus, R., Weiss, Y.: Small codes and large image databases for recognition. In: International conference on CVPR, pp. 1–8 (2008)
18. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS, pp. 1753–1760 (2009)
19. Heo, J.P., Lee, Y., He, J., Chang, S.F., Yoon, S.E.: Spherical hashing. In: International conference on CVPR, pp. 2957–2964 (2012)
20. Jegou, H., Douze, M., Schmid, C.: Improving Bag-of-Features for Large Scale Image Search. Int J Comput Vision **87**(3), 316–336 (2010)
21. Jegou, H., Douze, M., Schmid, C.: Packing bag-of-features. In: International Conference on Computer Vision (ICCV), pp. 2357–2364 (2009)
22. He, K., Wen, F., Sun, J.: K-means Hashing: an affinity-preserving quantization method for learning Binary Compact Codes. In: International Conference on CVPR, pp. 2938–2945 (2013)
23. Hajebi, K., Yadkori, Y.A., Shahbazi H., Zhang H.: Fast approximate nearest-neighbor search with *k*-nearest neighbor graph. In: IJCAI, pp. 1312–1317 (2011)
24. Wang, J., Wang, J., Zeng, G., Tu, Z., Gan, R., Li, S.: Scalable *k*-nn graph construction for visual search. In: CVPR, pp. 1106–1113 (2012)
25. Wang, J., Li, S.: Query-Driven Iterated Neighborhood Graph Search for Large Scale Indexing. In: ACM Multimedia, pp. 179–188 (2012)
26. Wang, J., Wang, J., Zeng, G., Gan, R., Li, S., Guo, B.: Fast Neighborhood Graph Search using Cartesian Concatenation., In: ICCV, pp. 2128–2135 (2013)
27. Brandt, J.: Transform coding for fast approximate nearest neighbor search in high dimensions. In: International conference on CVPR, pp. 1815–1822 (2010)
28. Chen, Y., Guan, T., Wang, C.: Approximate nearest neighbor search by residual vector quantization. Sensors **10**, 11259–11273 (2010)
29. Babenko, A., Lempitsky, V.: The Inverted multi-index. In: International Conference on CVPR, pp. 3069–3076 (2012)
30. Jegou, H., Tavenard, R., Douze, M., Amsaleg, L.: Search in one Billion Vectors: re-rank with Source Coding. In: IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP), pp. 861–864 (2011)
31. Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization for approximate nearest neighbor search. In: International Conference on CVPR, pp. 2946–2953 (2013)
32. Gray, R., Neuhoff, D.: Quantization. IEEE Trans. Inf. Theory **44**(6), 2325–2383 (1998)
33. Norouzi, M., Fleet, D.J.: Cartesian k-means. In: CVPR, pp. 3017–3024 (2013)
34. Gong, Y., Lazebnik, S.: Iterative quantization: a procrustean approach to learning binary codes. In: CVPR, pp. 817–824 (2011)
35. Chan, W., Gupta, S., Gersho, A.: Enhanced multistage vector quantization by joint codebook design. IEEE Trans Commun **40**(11), 1693–1697 (1992)
36. Hwang, Y., Han, B., Ahn, H.: A fast nearest neighbor search algorithm by nonlinear embedding. In: International Conference on CVPR, pp. 2053–306 (2012)
37. The ANN Evaluation Dataset. http://www.irisa.fr/texmex/people/jegou/ann.php

38. Jegou, H., Douze, M., Schmid, C.: Hamming embedding and weak geometric consistency for large scale image search. In: ECCV, pp. 304–317 (2008)
39. The INRIA Holidays Dataset. http://lear.inrialpes.fr/people/jegou/data.php#holidays
40. Torralba, A., Fergus, F., Freeman, W.T.: 80 million tiny images: a large database for non-parametric object and scene recognition. IEEE Trans. Pattern Anal. Mach. Intell. **30**(11), 1958–1970 (2008)
41. Babenko, A., Lempitsky V.: Additive quantization for extreme vector compression. In: CVPR, pp. 931–938 (2014)
42. Zhang T., Du C., Wang J.: Composite quantization for approximate nearest neighbor search. In: ICML, pp. 1–9 (2014)
43. Ai L., Yu J., Guan T., He Y.: Efficient approximate nearest neighbor search by optimized residual vector quantization. In: CBMI, pp. 1–4 (2014)