

Fast additive quantization for vector compression in nearest neighbor search

Jin Li¹ · Xuguang Lan¹ · Jiang Wang² · Meng Yang¹ · Nanning Zheng¹

Received: 15 May 2016 / Revised: 8 September 2016 / Accepted: 27 September 2016 /
Published online: 16 November 2016
© Springer Science+Business Media New York 2016

Abstract Vector quantization has been widely employed in nearest neighbor search because it can approximate the Euclidean distance of two vectors with the table look-up way that can be precomputed. Additive quantization (AQ) algorithm validated that low approximation error can be achieved by representing each input vector with a sum of dependent codewords, each of which is from its own codebook. However, the AQ algorithm relies on computational expensive beam search algorithm to encode each vector, which is prohibitive for the efficiency of the approximate nearest neighbor search. In this paper, we propose a fast AQ algorithm that significantly accelerates the encoding phase. We formulate the beam search algorithm as an optimization of codebook selection orders. According to the optimal order, we learn the codebooks with hierarchical construction, in which the search width can be set very small. Specifically, the codewords are firstly exchanged into proper codebooks by the indexed frequency in each step. Then the codebooks are updated successively to adapt the quantization residual of previous quantization level. In coding phase, the vectors are compressed with learned codebooks via the best order, where the search range is considerably reduced. The proposed method achieves almost the same performance as AQ, while the speed for the vector encoding phase can be accelerated dozens of times. The experiments are implemented on two benchmark datasets and the results verify our conclusion.

Keywords Additive quantization · Beam search · Vector compression · Nearest neighbor search

1 Introduction

Nearest neighbor (NN) search is a fundamental technique for many computer vision tasks, such as classification [3], recognition [21, 27], matching [5, 26] and retrieval [8, 10, 15, 23,

✉ Xuguang Lan
xglan@mail.xjtu.edu.cn

¹ Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University, Xi'an, P.R., China

² Institute for Deep Learning, Sunnyvale, USA

28]. The traditional NN search needs to calculate and store Euclidean distance between the high dimension vectors, thus it costs large amount of disk space and computation to deal with large scale data. To achieve higher efficiency, compact codes are widely used to speed up NN search for several reasons. First, representing high dimensional vectors with a small number of bits takes much less space while maintaining almost the same information (e.g., [27] shows that compressing images into binary codes still preserves the information required for recognition). Thus, compact codes of large dataset can be stored in memories, which are much faster than hard drives. Second, compact codes allow fast query because computing distance with the compressed data is much more efficient.

Recent work on learning compact codes falls into two categories. The first one is the hashing method that compresses vectors into binary codes using two stages [5, 9, 10, 16, 21, 26–29]. The first stage is to find a low-dimensional embedding and the second stage is to binarize the low-dimensional vectors. The other category is quantizing the input data with learned codebooks [1, 2, 4, 6, 8, 12, 15, 20, 30]. Both of the two categories of method can be used for the approximate nearest neighbor (ANN) search, in which the Euclidean distance can be replaced with the Hamming distance or the table look-up way like asymmetric distance computation strategy [15]. However, there are several differences between hashing and quantization method for ANN search. The most different impact is the way of computing the approximate distance. To approximate the Euclidean distances between query and database, hashing method computes Hamming distance between the binary codes of query and database, while quantization store the Euclidean distances between codewords and inquires them by the codes. Therefore the range of distance of quantization is much larger than hashing. Moreover, since the codewords and query are in the same Euclidean space, quantization methods can extend to asymmetric distance search strategy, which is proved better than the symmetric distance [15] like the Hamming distance. Since the proposed method follows the idea of the second category, we explicitly introduce the recent works about quantization as following.

Vector quantization (VQ) [11] implements Lloyd's method [17] to train a codebook where the nearest codeword is selected to represent an input vector. Product Quantization (PQ) [15] splits one input vector into several subvectors where each subvector is quantized independently by their own vector quantizers. The compact code of PQ is the concatenation of the indices of all quantizers. This quantization technique is efficient for high dimensional vectors because the size of each codebook can be considerably reduced compared to standard VQ techniques. However, PQ algorithm treats the subvectors as being independent, which does not consider the relationships between subvectors. To remedy this problem, transform coding (TC) [4] uses a set of scalar quantizers to compress each subvector, which employs principle components analysis (PCA) and adaptive bit allocation to reduce the information redundancy in each dimension. Optimized Product Quantization (OPQ) [8] learns an orthogonal projection matrix for input data to minimize quantization distortion by the space decomposition. Another quantization framework uses the addition of codewords instead of the Cartesian product to reconstruct the input vectors. Hence, it has no independence assumption on the codebooks. Residual vector quantization (RVQ) recursively implements the VQ method to quantize the residual of the previous quantization level [2, 6]. Additive quantization (AQ) [1] and Composition quantization (CQ) [30] trains a set of codebooks at the same time, where one and only one codeword from each codebook is selected to reconstruct the input.

AQ algorithm represents each input vector with a sum of codewords, where each codeword is from its own codebook. Since the independence assumption of PQ and OPQ is eliminated,

AQ has considerable improvement compared to the above two state-of-the-art methods. However, AQ relies on a heuristic search strategy called beam search [19, 25] to encode the input vectors. Beam search needs to exhaustively search over all available codebooks during the vector encoding, which causes high computational complexity that may not proper to dispose large-scale dataset. In practical applications, the efficiency of quantization method influences two aspects of the applied technique. On the service side, ANN search methods need to compress the data off-line and store the codes before a new query is given. Since amount of data is generated every day, fast encoding strategy can significantly save time so that the servers can compress the new data in time. On the client side, the query needs to be compressed to compact code for ANN search. A fast encoding strategy will reduce the retrieval time Therefore, an efficient encoding algorithm is significantly important.

The rest of the paper is organized as follows. In Section 2, the additive quantization is reviewed. In Section 3, the beam search problem is formulated and the proposed solution is introduced. In Section 4, we show the experimental results. The paper is concluded in Section 5.

2 Additive quantization

We first introduce the additive quantization which is the basis of our work. Figure 1 shows its whole framework, which contains the *training phase* and the *compressing phase*. In the training phase, the codebooks and the code of data are updated in *codebook learning stage* and *encoding stage* iteratively. Then the learned codebooks are used to compress the large-scale database in the compressing phase.

The process of the training phase can be described as follows. With N input vectors $\mathcal{X} = \{\mathbf{x}_n\}_{n=1}^N$ in training phase, the additive quantization represents one vector \mathbf{x} as

$$\mathbf{x} \approx \sum_{m=1}^M c_m(b_m), \quad b_m \in 1, \dots, K \tag{1}$$

where $c_m(b_m)$ denotes the b_m th codeword in codebook C_m , K is the number of codewords in each codebook, and M means the number of codebooks. We further denote the set of M codebooks $\mathcal{C}^{(M)} = \bigcup_{m=1}^M C_m$ and an M -tuple code $\mathbf{b} = [b_1, \dots, b_M]$. The training phase aims to find the proper codebooks and codewords representing training data with minimum approximation error. i.e.,

$$\min_{\mathcal{C}^{(M)}, \mathbf{b}} \sum_{\mathbf{x} \in \mathcal{X}} \left\| \mathbf{x} - \sum_{m=1}^M c_m(b_m) \right\|^2 \tag{2}$$

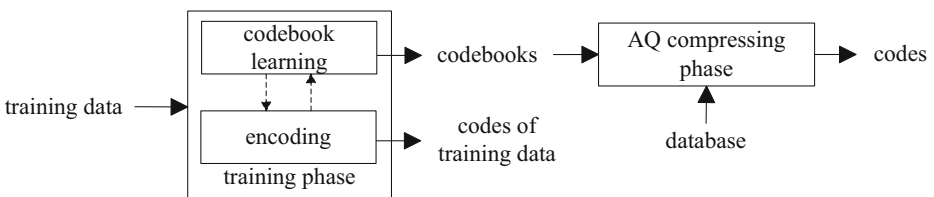


Fig. 1 The framework of Additive Quantization

Equation (2) is often solved via iteratively optimizing \mathbf{b} and $\mathbf{C}^{(M)}$. In the codebook learning stage, with the given code $\mathbf{b} \in \mathcal{B}$ (where \mathcal{B} is the set of codes for χ), optimization of $\mathbf{C}^{(M)}$ can be simply solved as an unconstrained nonlinear optimization problem [1]. In the encoding stage, $\mathbf{C}^{(M)}$ is fixed and the best code \mathbf{b} for each \mathbf{x} is updated. However, the best combination of codewords from each codebook is difficult to find in finite search loops, and the encoding stage usually occupies most of the time consumption in training phase. Therefore the key point of additive quantization is to find an efficient search strategy for vector encoding. The approximation error of \mathbf{x} is defined using Euclidean distance

Algorithm 1 beam search in AQ

Input: one vector \mathbf{x} , the set of codebooks $\mathcal{C} = \{C_m\}$, width of beam search T

Output: code \mathbf{b} , codebook selecting order \mathbf{m}

1. Initialize $\mathbf{K} = \mathbf{0}^{T \times M}$, $\mathbf{m} = \mathbf{0}^{T \times M}$.
 2. $\mathbf{K}(:, 1) \leftarrow$ indices of T -NN of \mathbf{x} from \mathcal{C}
 3. $\mathbf{p}(:, 1) \leftarrow$ indices of codebooks of $\mathbf{K}(:, 1)$
 4. **for** $i = 2$ to M
 5. Initialize $\mathbf{K}^* = \mathbf{0}^{T^2 \times i}$, $\mathbf{m}^* = \mathbf{0}^{T^2 \times i}$
 6. **for** $t = 1$ to T
 7. $\mathbf{K}^*((t-1)T+1:tT, 1:i-1) \leftarrow \mathbf{K}(t, 1:i-1)$
 8. $\mathbf{p}^*((t-1)T+1:tT, 1:i-1) \leftarrow \mathbf{p}(t, 1:i-1)$
 9. $\mathbf{r} = \mathbf{x} - \sum_{j=1}^i C_{m_{tj}}(\mathbf{K}_{tj})$
 10. $\mathbf{K}^*((t-1)T+1:tT, m) \leftarrow$ indices of T -NN of \mathbf{r} from $\mathcal{C} \setminus \bigcup_{j=1}^{i-1} C_{m_{tj}}$
 11. $\mathbf{m}^*((t-1)T+1:tT, 1:i-1) \leftarrow$ indices of codebooks of $\mathbf{K}^*((t-1)T+1:tT, i)$
 12. **end for** t
 13. $id \leftarrow$ indices of T smallest $\|\mathbf{x} - \sum_{j=1}^i C_{m_{tj}}^*(\mathbf{K}_{tj}^*)\|$
 14. $\mathbf{K}(:, 1:i) \leftarrow \mathbf{K}^*(id, 1:i)$
 15. $\mathbf{m}(:, 1:i) \leftarrow \mathbf{m}^*(id, 1:i)$
 16. **end for** i
 17. $id \leftarrow$ index of smallest $\|\mathbf{x} - \sum_{j=1}^M C_{m_{tj}}(\mathbf{K}_{tj})\|$
 18. $\mathbf{b} \leftarrow \mathbf{K}(id, :)$
-

$$E(\mathbf{b}) \triangleq \left\| \mathbf{x} - \sum_{m=1}^M c_m(b_m) \right\|^2 \quad (3)$$

With the input \mathbf{x} , the minimization of Eq. (3) is essentially a high-order Markov random field problem where each b_m in \mathbf{b} is a discrete random variable, and the state of node b_m can be updated iteratively to find the minimal error. Many methods can be used to solve this problem, such as alternative optimization technique [30] and Loopy Belief Propagation [24].

Discussed in [1], these optimal methods may cost very large amount of time to find the best combination even for small codebooks. AQ adopts the beam search [25] which constructs the output codewords successively. Specifically, beam search takes M steps to choose the M codewords instead of alternatively updating. With an input vector \mathbf{x} , the available set of codewords is initialized as $\mathcal{C}^{(M)}$. In the first slice, the top T nearest neighbors (T -NN) for \mathbf{x}

are found in $\mathcal{C}^{(M)}$ to be T candidates. In the next slice, for each candidate, an available set of codewords is updated as $\mathcal{C}^{(M-1)} = \mathcal{C}^{(M)} \setminus C_m$. Then the T -nearest neighbors for $x - c_m(b_m)$ are found in $\mathcal{C}^{(M-1)}$, therefore we obtain T^2 tuples of codewords for the previous T candidates. Then T unique tuples of codewords with less approximation (i.e., $\|x - c_m(b_m)\|^2$) are picked to be the candidates for next step. After M slices, the T unique tuples of codewords are selected, and one tuple with the smallest approximation error is returned as the code of x . An explicit description can be summarized in Algorithm 1 for further discussion. The beam search needs to seek all available set of codebooks, that means it searches the remaining $M - m + 1$ codebooks $\mathcal{C}^{(M-m+1)}$ in m th step, therefore the computational complexity referring to M is $O(M(M-1)/2) = O(M^2)$.

After the training phase, the beam search is used to compress the database with the codebooks learned in the compressing phase. However, we think the time consumption is still unacceptable in practical application, where the number of databases is usually very large. There are two shortages in AQ that can be considered to improve the efficiency of compressing phase. First, in the compressing phase the learned codes of training data are useless. Second, the prior of the input data is not used to contribute to encode the database after the training phase, therefore the compressing phase of AQ is the same as the encoding stage in the training phase, which still needs to search all possible set $\mathcal{C}^{(M-m+1)}$ in m th step.

In this paper, we propose two modifications to reduce the search range. First, we to take advantages of the codes of training data to calculate a codebook selecting order, in which we can search the nearest neighbors in one certain codebook $C_m \in \mathcal{C}^{(M)}$ instead of all possible codebooks $\mathcal{C}^{(M-m+1)}$ in each step, and then the computational complexity can be reduced to $O(M)$ instead of $O(M^2)$. Second, we re-train the codebook where the codewords in C_m tends to be selected in the m -th slice. Thus the search width can be narrowed. As the hierarchical codebooks are trained, the performance can be similar or even better than beam search in AQ.

3 Fast additive quantization

In this section, a formulation of optimization problem is established to analyze beam search which is the key method in the encoding phase in AQ. According to the formulation, an optimal order of codebook selecting is calculated. Then we learn the codebooks to obtain a hierarchical structure where we can reduce the search width in compressing phase.

3.1 Formulation of beam search

We first formulate the beam search as an optimization problem of codebook selecting order which is called the *path* in following discussion. Without loss of generality, we let the beam search width T equal to 1 to obtain a concise expression. Then according to the Algorithm 1, we define the path m as the M -tuple $[m^{(1)}, \dots, m^{(M)}]$ where the i -th component $m^{(i)} \in \{1, M\}$ represents the i th selected codebook, we also denote $r^{(i)}$ as the residual of the vector x subtracting the assigned i codewords in codebook $C_{m^{(1)}}, \dots, C_{m^{(i)}}$. The algorithm shows that all codewords in $\mathcal{C}^{(M-m+1)}$ are searched with the calculated residual $r^{(i-1)}$ in i -th iteration ($r^{(0)} = x$), then the nearest codeword

$b_{m^{(i)}}$ and the number of codebooks $m^{(i)}$ is returned, which can be formulated as

$$\begin{aligned}
 [m^{(i)}, b_{m^{(i)}}] = \arg \min_m & \left\{ \min_b \left| |r^{(i-1)} - c_m(b)| \right|^2 \right\} \\
 \text{s.t. } i = 1, \dots, M & \\
 b = 1, \dots, K & \\
 m \notin \left\{ m^{(j)} \right\}_{j=1}^{i-1} &
 \end{aligned} \tag{4}$$

According to our formulation, the beam search in i -th step can be decomposed into two minimization problems: one is the minimization with respect to m , which is called the *path optimization* because it needs to find a proper order for codebook selection. The other is the minimization with respect to codeword b , which is called *inner optimization* since it searches the nearest neighbor of $r^{(i-1)}$ within one codebook.

For the i -th step, we could obtain the i -th codeword $c_{m^{(i)}}(b_{m^{(i)}})$. Then $r^{(i)}$ is updated with all the obtained codewords $c_{m^{(1)}}(b_{m^{(1)}}), \dots, c_{m^{(i)}}(b_{m^{(i)}})$ as follows

$$r^{(i)} = x - \sum_{m=m^{(1)}}^{m^{(i)}} c_m(b_m) \tag{5}$$

After M steps, each component of the best code b is obtained iteratively. Eq. (4) shows that the beam search uses the enumeration method to solve the two-variable optimization problem in each step, which takes large time consumption. We decompose Eq. (4) to two individual optimization problems to approximate the exhaustive search. Specifically, the set of best codes B for χ is obtained after training phase, therefore we can fix the codes and solve the order optimization problem to calculate the best path m . Then for a large-scale database, we only need to solve the inner optimization problem under the best path in the compressing phase, which has lower computational complexity compared to the original compressing phase in AQ.

3.2 Optimal order of codebook selecting

After beam search, the best assignments B of training data can be obtained while the current codebooks C . Then we calculate the best path as the prior information using the best assignments. As $b = [b_1, \dots, b_M]$ is fixed, the inner optimization of Eq. (4) is eliminated, and then it can be simplified as

$$\begin{aligned}
 m^{(i)} = \arg \min_m & \left| |r^{(i-1)} - c_m(b_m)| \right|^2 \\
 \text{s.t. } i = 1, \dots, M & \\
 m \notin \left\{ m^{(j)} \right\}_{j=1}^{i-1} &
 \end{aligned} \tag{6}$$

This optimization problem can be easily solved by exhaustively searching all codebooks, which takes little time compared to the beam search. The Eq. (6) shows that the best path achieves the minimal residual value in each step comparing to other

paths. When we search the nearest neighbor in codebooks in this order, the approximation error is exactly the same as the result of beam search. It is distinct that one fixed path is not unified for all input data, therefore, we consider the residual value to be a random variable and then define the optimal path with minimal expectation of residual value in each step. Specifically, each component $m_o^{(i)}$ for the *optimal path* \mathbf{m}_o can be defined as

$$\begin{aligned}
 m_o^{(i)} \triangleq & \underset{m}{\operatorname{argmin}} \mathbb{E} \left(\|\mathbf{r}^{(i-1)} - c_m(b_m)\|^2 \right) \\
 \text{s.t. } & i = 1, \dots, M \\
 & m \notin \left\{ p_o^{(j)} \right\}_{j=1}^{i-1}
 \end{aligned} \tag{7}$$

And the residual $\mathbf{r}^{(i)}$ for each input \mathbf{x} can be calculated as

$$\begin{aligned}
 \mathbf{r}^{(i)} = \mathbf{x} - \sum_{m=m_o^{(1)}}^{m_o^{(i)}} c_m(b_m) \\
 \text{s.t. } i = 1, \dots, M
 \end{aligned} \tag{8}$$

We calculate \mathbf{m}_o by alternatively optimizing Eq. (7) and (8) after the training phase. Compared to other path, the optimal path has the smallest expectation of residual value in each step, which is defined as

$$\bar{R}^{(i)} = \mathbb{E} \left(\left\| \mathbf{x} - \sum_{m=m_o^{(1)}}^{m_o^{(i)}} c_m(b_m) \right\|^2 \right) = \mathbb{E} \left(\|\mathbf{r}^{(i)}\|^2 \right) \tag{9}$$

If the vectors in the database follow the same distribution of the training set, we think \mathbf{m}_o is also the optimal path for the database because of its similar statistical characteristic. And the optimal path \mathbf{m}_o , which will lead to find smaller final approximation error than other paths, tends to achieve smaller mean residual value in each step than other fixed path, with the conclusion verified by the experimental result.

3.3 Hierarchical codebook updating

Given the optimal path \mathbf{m}_o , we retrain the codebooks which have the hierarchical structure for the input. Specifically, $C_{m_o^{(i)}}$ are updated to minimize the quantization distortion of $\mathbf{r}^{(i-1)}$. A two-step process is proposed to adjust the codewords in codebooks for the residuals in M slices.

3.3.1 Codewords exchanging

As the beam search is used to find the combination of codewords from each codebooks, the optimal path for each input \mathbf{x} can be calculated at the same time. To let the codewords in $C_{m_o^{(i)}}$ can reduce the quantization distortion of $\mathbf{r}^{(i-1)}$ at the most extent, we exchange the top K codewords which are selected most frequently in step $m_o^{(i)}$ into the codebook $C_{m_o^{(i)}}$. Then the path for each input tends to be the same as \mathbf{m}_o .

3.3.2 Codewords updating

After the frequently selected codewords are exchanged to proper codebooks, we can fast re-quantize the input data using the residual quantization strategy [2] via the optimal path \mathbf{m}_o . With the code \mathbf{b} calculated by the residual quantier, the codebook $C_{m_o^{(i)}}$ is optimized as

$$C^* = \operatorname{argmin}_C \mathbb{E} \left(\|\mathbf{r}^{(i-1)} - C(b_{m_o^{(i)}})\|^2 \right) \quad (10)$$

Algorithm 2 hierarchical codebooks learning

Input: training set $\mathcal{X}=[\mathbf{x}_1, \dots, \mathbf{x}_N]$, number of codebooks M

Output: codebooks $\mathbf{C}=[C_1, \dots, C_M]$

1. Initialize \mathbf{C} using residual quantization.
 2. **for** round = 1 to 10
 3. $\mathcal{R} = \mathcal{X}, \mathbf{C}_{\text{in}} = \mathbf{C}$
 4. **for** $i = 1$ to M
 5. Calculate codes and paths for \mathcal{R} by **Algorithm 1**.
 6. Calculate \mathbf{m}_o by Eq. (7) and Eq. (8).
 7. $\mathbf{C}_{\text{in}} \leftarrow$ sort codewords in \mathbf{C}_{in} by the frequency indexed in $m_o^{(1)}$ slice.
 8. $C_i \leftarrow \mathbf{C}_{\text{in } m_o^{(1)}}$
 9. $\mathcal{R} \leftarrow$ residual of \mathcal{R} quantized by C_i
 10. $\mathbf{C}_{\text{in}} = \mathbf{C}_{\text{in}} \setminus \mathbf{C}_{\text{in } m_o^{(1)}}$
 11. **end for** i
 12. $\mathcal{B} \leftarrow$ quantize \mathcal{X} by residual quantization with \mathbf{C}
 13. $\mathbf{C} \leftarrow$ solve Eq. (10) with inverted order.
 14. **end for** round
-

Equation (10) can be solved using the least-quadratic problem, which is same as the codebook learning step in AQ [1]. Since the code $b_{m_o^{(i)}}$ may change if we firstly update the $(i-1)$ th codebook $C_{m_o^{(i-1)}}$, we update the codebooks with the inverted order $[m_o^{(M)}, m_o^{(M-1)}, \dots, m_o^{(1)}]$.

3.3.3 Adaptive codebook learning

We propose an adaptive learning strategy to construct the hierarchical codebooks using the two-step process. In the first iteration, the 1-width beam search ($T = 1$) is implemented to calculate the code and path from the M codebooks for each input and \mathbf{m}_o is calculated by Eq. (7) and (8). Second, the most frequently indexed codewords in 1-st step are moved into $C_{m_o^{(1)}}$, and then the input \mathbf{x} is quantized by the new codebook $C_{m_o^{(1)}}^*$, where $b_{m_o^{(1)}}$ is the code indexing the nearest codeword. In the next $m-1$ iteration, we repeat the above steps with the remaining $M-m+1$ codebooks for the residual $\mathbf{r}^{(m-1)}$. After the M iterations, the codewords are exchanged to the proper codebooks and new codes are calculated. Then we solve the Eq. (10) to update the each codebook. The Algorithm 2 summarizes the codebook learning process.

We implement the codebook learning step several loops to train the hierarchical codebooks. The computational complexity for each loop is $O(M^3KDT)$ while that for AQ is $O(M^2KDT)$. However, since we iteratively train the ordered codebooks, our method can only search a local optimal combination of codewords in once iteration. Therefore we let the width of beam search $T=1$ which can significantly reduce the cost in the beam search in training phase (AQ let $T=1$ 6 for codebook learning). In practical, the number of codebooks M is not large (generally equal to 4 in AQ), therefore the total cost of codebook learning is much less in our method than AQ.

3.4 Fast AQ compression

After we obtain the best path m_o and train the hierarchical codebooks, we can quantize the input with the fixed order of codebooks and a small search width T . The complexity of encoding a vector is $O(MKDT)$ instead of $O(M^2KDT)$ in AQ encoding phase. Moreover, as we exchange the most frequently indexed codewords into the proper codebooks, the search width T does not required to be very large to find the global minimal combination of codewords. In experiments, we find that the performance when $T=4$ is similar to that of AQ. Compare to the AQ where $T=64$ in compression phase, our method is dozen of times faster than AQ. Figure 2 illustrates the framework of our approach, where the highlights show the differences from AQ.

4 Experimental results

In this section, we present two experiments that validate the effectiveness of FAQ. The first is the comparison of the approximation error between the proposed approach and three baselines. The second experiment is comparison of the performance of nearest neighbor search. As our main propose is to reduce the computational complexity of AQ, we evaluate the time consumption of compressing phase in FAQ and AQ to show our improvement of computation when encoding large testing dataset.

The experimental settings and standard datasets are introduced in Section 4.1 while experimental results are shown in Section 4.2 and Section 4.3 respectively.

4.1 Dataset and baseline

SIFT-1 M [7]: this dataset is a collection of 128-dimensional SIFT descriptors [18]. It contains one million base vectors, 100,000 training vectors and 10,000 queries with known true Euclidean nearest. GIST-1 M [7]: GIST descriptors are 960-dimensional global features extracted from images [22]. This dataset consists of one million base vectors, 500,000 training vectors and 1000 queries.

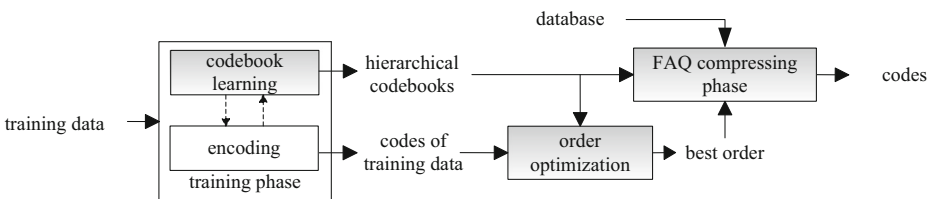


Fig. 2 The framework of Fast Additive Quantization

Several methods achieving state-of-the-art performance are used as the baselines. The first category of methods is based on addition quantization framework, including RVQ [2] and AQ [1]. For RVQ, the K-means method is used to train the codebooks where the number of iteration is 100. For AQ, the width T is set to 16 in encoding stage of the training phase and 64 for the compressing phase.

The second category is product quantization framework, which includes PQ [15], TC [4] and OPQ [8]. For PQ, we use the code written by Hervé Jégou [14] and the number of iteration is set to 100. For TC, which is a special case of PQ, we allocate and bits and implement the matrix transform, and then use [14] to train scalar quantizer for each dimension. For OPQ, we use the code published by Kaiming He [13]. The codebooks are initialized by 100 iterations PQ, and the non-parametric OPQ is used in 50 iterations to train the final codebooks and rotation matrix.

As our approach is modified based on AQ, we use 20-iteration RVQ to initial the codebooks. Then we apply fast AQ to encode the training data where search width is set to be 1. We find 10 iterations with training phase are enough to train the hierarchical codebooks. In compressing phase, the performance is similar to AQ when search width $T = 4$.

We also compare our method with some hashing methods. Since the most existing quantization methods are for unsupervised, we choose two well-known unsupervised hashing methods, LSH [9] and ITQ [10]. Specifically, we use training data to learn the hashing function, and then find the nearest neighbors of queries from the base vectors. Moreover, different from quantization methods, hashing methods aim to compress data only for search without data reconstruction, thus we only compare the hashing methods in the search performance without quantization distortion.

All measurements are taken on a machine with i7-4790 CPU, with 3.60GHz processor and 4 cores.

4.2 Data representation

The main measurement for vector quantization is the approximation error which reflects the loss of information. Figure 3 shows performance and the time consumption of our approach with different codebook size for SIFT-1 M dataset. Since the size of codebooks for TC is

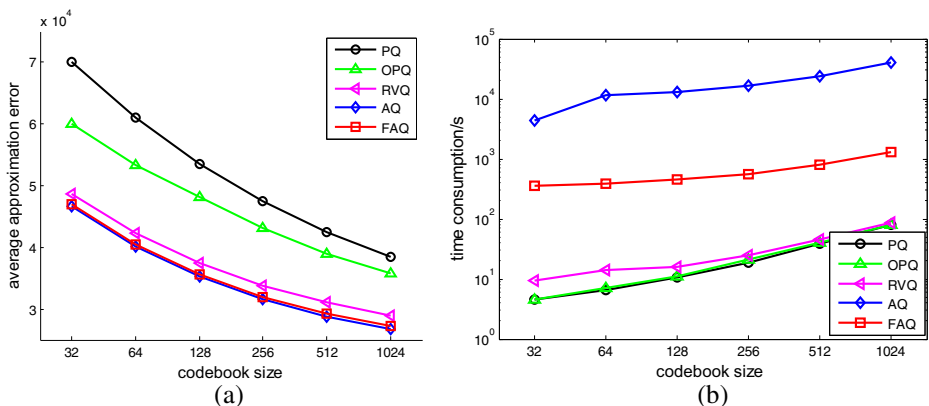


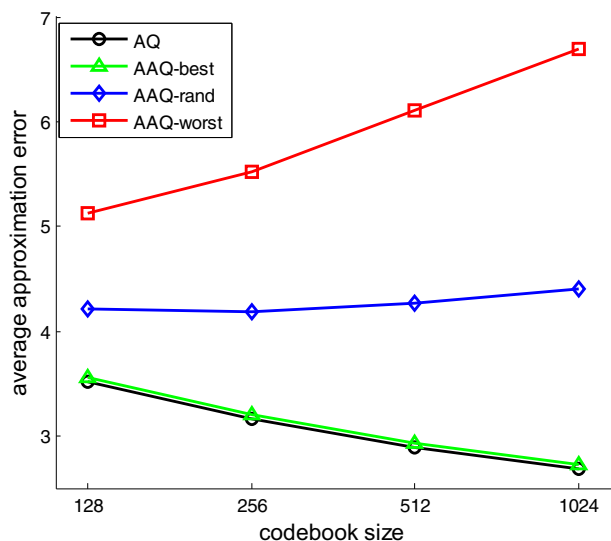
Fig. 3 Comparisons of approaches on dataset SIFT-1 M with various codebook sizes ($M = 4$): **a** approximation error; **b** time consumption

adaptive, we only compare FAQ with other four baselines in this experiment. Figure 3a presents that the error of FAQ is nearly the same as AQ, which much smaller than PQ and OPQ. In details, the error of PQ and OPQ ranges from 7×10^4 and 6×10^4 to 3.9×10^4 and 3.6×10^4 respectively, which is about 40 % and 23 % larger than the error of AQ. The performance of RVQ and FAQ are better than the product methods. However, the average performance gap between RVQ and AQ is about 6 %, which may not be ignored in many cases. On contrast, FAQ is about 1 % larger than AQ, which is smaller enough to be ignored. The comparison of time consumption is shown in Fig. 3b. Since PQ and RVQ can compress batches of vectors at the same time, their speed is much faster than AQ. As the proposed method aims to improve the efficiency of AQ, FAQ is about 30 times faster than the AQ for compressing vectors. Therefore FAQ can be implemented in applications instead of AQ since the little performance losses.

For further discussion, two other paths are used to show the improvement of hierarchical structure in codebooks. First is a random order, which randomly selects each codebook to search the nearest neighbor. Second is the largest path that achieves maximum residuals in each step. Figure 4 presents the approximation errors of the different three paths, in which the AQ is used as the baseline to reveal the performance gaps. The result shows that the performance gap between AQ and FAQ using optimal path is smaller than 2 %, while the gap of other two paths are quite large. Specifically, the distortion of the largest order is about 20 % to 150 % larger than AQ, and that of random order is 8 % to 65 %. The comparison verifies that the codebooks are hierarchical for different level of residual. That means the best combination tends to be the same as the solution of residual quantization via the optimal path. Moreover, we find the approximate error is positively correlates with the residual value in each step, and the theoretical analysis of their relationship will be researched as a future work.

Then we discuss the influence of different code lengths ($L = M \log_2 K$), where all four baselines are compared. To compare the performance of AQ, the Fast Additive Product

Fig. 4 Comparisons of different paths on 10,000 SIFT descriptors with various codebook sizes. “FAQ-best” stands for the result under optimal path, while “FAQ-rand” and “FAQ-largest” correspond to random path and largest path respectively



Quantization (FAPQ), which combines the FAQ and OPQ, is used for 64(128) bits code length to compare with the Additive Product Quantization (APQ). In more details, the vector is decomposed to $M_1 = 8(16)$ orthogonal components as OPQ first, and each component is encoded by FAQ with $M_2 = 4$ codebooks. Figure 5 shows that the FAQ achieves the similar performance and has advantage of time consumption in different code length. Figure 5a presents that the performance gap between AQ and FAQ is quite small compared to other baselines. When compressing millions vectors such SIFT descriptors, the time consumption for FAQ is about several minutes while AQ require hours. Thus our fast approach has advantage after considering the representational power and time consumption synthetically.

4.3 Nearest neighbor search

Approximate nearest neighbor (ANN) search is a widely used technology in image processing and computer vision. In this experiment, the datasets are encoded using the four methods for ANN search while the true Euclidean nearest neighbor given by dataset is regarded as the ground truth.

The asymmetric distance computation (ADC) strategy [15] is used for PQ and OPQ to approximate the Euclidean distance. Specifically, the database is encoded off-line first, then the Euclidean distance between query and each codeword is computed, which are further stored in a table, finally the approximate distance between query and database are computed by looking up the table, where the code are used for indexing the Euclidean distance of respective codewords. Different from ADC strategy, our approach use the summation of the module value of query and scalar product to approximate the Euclidean distance between query and database, what's more, the scalar product between codewords can be precomputed and stored in tables. The details are shown in [1].

Figure 6 shows the Recall@ R measures for the query [15] as the search accuracy, which is a generally used measure that calculates the accuracy of the top R nearest neighbors compared to the ground truth. For the SIFT dataset, the Recall@ R has similar accuracy between AQ and FAQ, which are better than PQ, TC, OPQ and RVQ. For GIST dataset, the performance of FAQ is similar to or even better than AQ,

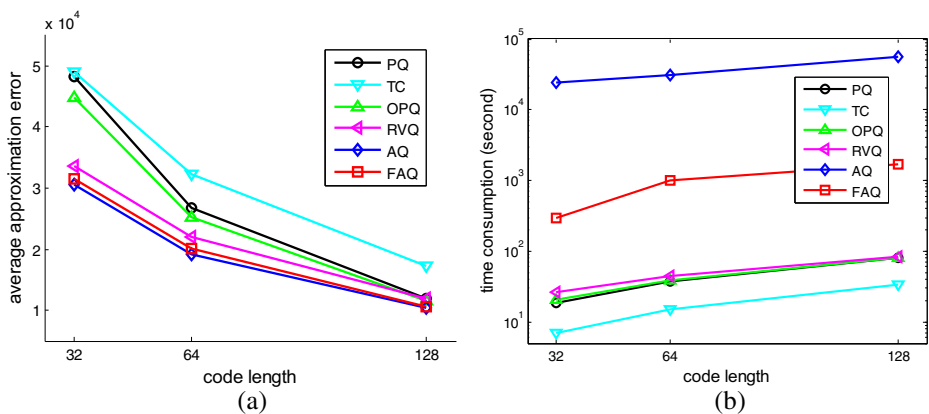


Fig. 5 Comparisons of approaches on SIFT-1 M dataset with different code length: **a** approximation error; **b** time consumption. AQ is used for 32 bits while hybrid algorithm (APQ) is used for 64(128) bits, so does our approach

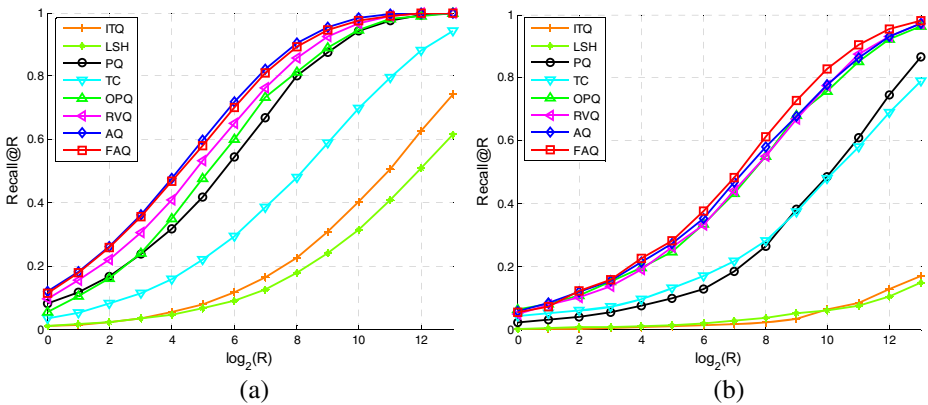


Fig. 6 Comparison of approaches with *recall* at R top ranked samples on different dataset: **a** SIFT-1 M dataset; **b** GIST-1 M dataset

OPQ and RVQ, which are much better than PQ and TC. Considering the structure of two dataset, we think the methods perform differently in two dataset for following reason. The SIFT descriptors have symmetric structure which contains 8 directions in 128 dimension and it tends to be independent when product quantization is used directly, therefore, performance between PQ and OPQ are similar. When it comes to GIST descriptor which has asymmetric structure, the independence assumption in PQ may cause more problems, therefore the AQ, RVQ, FAQ and OPQ have large improvement compared to PQ. Moreover, [15] mentioned that when input vector is divided into shorter subvectors, the quantization distortion using same bits becomes larger, therefore the TC which splits the vector into scalars achieves the worst performance. In fact, the performance of TC will improve significantly when the coding length rises. What's more, the results show that the search performance of quantization is much better than the hashing method, which is agreed with the discussion in the introduction.

5 Conclusion

In this paper we present the fast additive quantization (FAQ) approach, which trains the hierarchical codebooks via the optimal path to reduce the search range. To find the best search order, we first formulate the beam search as an order optimization problem of codebook selection, which contains the path optimization and inner optimization problem. Then we update the codebooks to minimize the quantization distortion of the residual of each quantization level. After the codebooks contain the hierarchical structure, the search width can be significantly reduced. In compressing phase, we search the nearest codewords in optimal order of codebooks with small width. The experimental results show that our method has the similar performance of vector quantization as AQ while the time consumption is considerable reduced in the vector encoding phase.

For future works, we will analyze the correlation between the approximation error and the expectation of residual values in each step theoretically.

Acknowledgments This work was supported in part by the National Key Research and Development Program of China under grant No. 2016YFB1000903, and NSFC No. 61573268.

References

1. Babenko A and Lempitsky V (2014) Additive Quantization for Extreme Vector Compression, in: 2014 I.E. Conference on Computer Vision and Pattern Recognition (CVPR), IEEE
2. Barnes C, Rizvi S, Nasrabadi N (1996) Advances in residual vector quantization: a review. *IEEE Trans Image Process* 2:226–262
3. Boiman O, Shechtman E, and Irani M (2008) In defense of nearest-neighbor based image classification, in: 2008 I.E. Conference on Computer Vision and Pattern Recognition (CVPR), IEEE
4. Brandt J (2010) “Transform Coding for Fast Approximate Nearest Neighbor Search in High Dimensions,” *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 1815–1822
5. Calonder M, Lepetit V, Strecha Christoph, and Fua Pascal (2010) BRIEF: Binary Robust Independent Elementary Features, In: European Conference on Computer Vision (ECCV), Springer
6. Chen Y, Guan T, Wang C (2010) Approximate nearest neighbor search by residual vector quantization. *Sensors* 10(12):11259–11273
7. Douze M, Jégou H, Singh H, Amsaleg L, and Schmid C (2009) Evaluation of GIST descriptors for web-scale image search, in: International Conference on Image and Video Retrieval
8. Ge T, He K, Ke Q, and Sun J (2013) Optimized product quantization for approximate nearest neighbor search, in: 2013 I.E. Conference on Computer Vision and Pattern Recognition (CVPR), IEEE
9. Gionis A, Indyk P, R (1999) Motwani, Similarity search in high dimensions via hashing, *VLDB*. pp:518–529
10. Gong Y, Lazebnik S (2013) Iterative quantization: a procrustean approach to learning binary codes, *trans. Pattern analysis and machine. Intelligence* 35(12):2916–2929
11. Gray R, Neuhoff D (1998) Quantization, *IEEE Trans. Information. Theory* 44:2325–2383
12. Guo Q, Zeng Z, Zhang S, Zhang G, Zhang Y (2016) Adaptive bit allocation product quantization. *Neurocomputing* 171:866–877
13. He K. the source code of optimization product quantization, <https://research.microsoft.com/en-us/um/people/kahe/cvpr13>
14. Jégou H, the source code of product quantization, <https://gforge.inria.fr/frs/download.php/33241>
15. Jégou H, Douze M, Schmid C (2011) Product quantization for nearest neighbor search, *trans. Pattern analysis and machine. Intelligence* 33(1)
16. Kulis B and Grauman K. Kernelized (2009) Locality-sensitive hashing for scalable image search, in: IEEE International Conference on Computer Vision, IEEE
17. Lloyd S (1982) Least squares quantization in PCM,” *IEEE Trans. Information. Theory* 28(2)
18. Lowe D (2004) Distinctive image features from scale-invariant keypoints. *Int J Comput Vis* 60(2)
19. Mallat S, Zhang Z (1993) Matching pursuit in a time frequency dictionary. *IEEE Trans Signal Process* 41(12)
20. Martinez J, Hoos H and Little J (2014) Stacked Quantizers for Compositional Vector Compression, in: arXiv preprint arXiv: 1411.2173
21. Obdrzalek S and Matas J (2005) Sub-linear indexing for large scale object recognition, in: 2005 British Machine Vision Conference (BMVC), BMVA press
22. Oliva A, Torralba A (2001) Modeling the shape of the scene: a holistic representation of the spatial envelope. *Int J Comput Vis* 42(3)
23. Pauleve L, Jegou H, Amsaleg L. (2010) Locality sensitive hashing: a comparison of hash function types and querying mechanisms. *Pattern Recogn Lett* 31
24. Pearl J (1988) Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, 3
25. Shapiro S (1987) Encyclopedia of Artificial Intelligence
26. Strecha C, Bronstein A, Bronstein M, Fua P (2010) Ldhash: improved matching with smaller descriptors, *trans. Pattern analysis and machine. Intelligence* 34
27. Torralba A, Fergus R, and Weiss Y (2008) Small codes and large image databases for recognition, in: 2008 I.E. Conference on Computer Vision and Pattern Recognition (CVPR), IEEE

28. Wang J, Kumar S, and Chang S (2010) Semi-supervised hashing for scalable image retrieval, in: 2010 I.E. Conference on Computer Vision and Pattern Recognition (CVPR), IEEE
29. Yu G and Yuan J (2014) Scalable forest hashing for fast similarity search, in: IEEE International Conference on Multimedia and Expo (ICME), IEEE
30. Zhang T, Du C, and Wang J (2014) Composite quantization for approximate nearest neighbor search, in: International Conference on Machine Learning (ICML)



Jin Li received the B.S. degrees in Automation from Xi'an Jiaotong University, Xi'an, China in 2013. Currently, he is a Ph.D. candidate in the School of Electronic and Information Engineering. His research interests including quantization technique for approximate nearest neighbor search and large-scale retrieval.



Xuguang Lan received the Ph.D. degree in pattern recognition and intelligent systems from Xi'an Jiaotong University (XJTU), Xi'an, China, in 2005. After visiting Ecole Centrale de Lyon for five months in France as a Visiting Researcher, he joined the Department of Computer Science and Technology to pursue the Post-Doctoral Research at XJTU, in 2005. He is currently a Professor with XJTU. His current research interests include image/video coding, processing, and communication, P2P technology, media content analysis, and VLSI design.



Jiang Wang received the B.S. and M.S. degrees in electronic from Fudan University, Shanghai, China, and the Ph.D. degree in Northwestern University Evanston, IL, USA. Now he is a researcher in Baidu Institute of Deep Learning, Sunnyvale, USA. His current research interests including human action recognition and deep learning.



Meng Yang received the Ph. D degree in the School of Electronic and Information Engineering, Xi'an Jiaotong University (XJTU), Xi'an, China, in 2013. He was a Visiting Scholar at the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA, USA, from 2011 to 2012. His current research interests include video coding, processing, and communication.



Nanning Zheng received the B.S. and M.S. degrees in information and control engineering from Xi'an Jiaotong University (XJTU), Xi'an, China, in 1975 and 1981, respectively, and the Ph.D. degree in electrical engineering from Keio University, Yokohama, Japan, in 1985. In 1975, he joined XJTU, where he is currently a Professor and the Director of the Institute of Artificial Intelligence and Robotics. His current research interests include computer vision, pattern recognition and image processing, and hardware implementation of intelligent systems. He became a member of the Chinese Academy of Engineering, in 1999, and is the Chinese Representative on the Governing Board of the International Association for Pattern Recognition. He also serves as the Executive Deputy Editor of the *Chinese Science Bulletin* and as an Associate Editor for the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS.